THESE DE DOCTORAT DE SORBONNE UNIVERSITE

Spécialité

Science des données
Ecole doctorale : EDITE ED 130

Présentée par

M. Danny Francis

Pour obtenir le grade de

DOCTEUR de SORBONNE UNIVERSITE

Sujet de la thèse :

Représentations sémantiques d'images et de vidéos

soutenue le 12 décembre 2019

devant le jury composé de : (préciser la qualité de chacun des membres).

SEMANTIC REPRESENTATIONS OF IMAGES AND VIDEOS

DANNY FRANCIS

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

Credits: xkcd.com

Supervisors: Prof. Bernard Merialdo & Prof. Benoit Huet

November 2019 – version 1.0

DECLARATION OF AUTHORSHIP

I declare that this thesis has been composed solely by myself. Except where states otherwise by reference or acknowledgment, the work presented is entirely my own.

Biot, France, November 2019

Danny Francis

ABSTRACT

Describing images or videos is a task that we all have been able to tackle since our earliest childhood. However, having a machine automatically describe visual objects or match them with texts is a tough endeavor, as it requires to extract complex semantic information from images or videos. Recent research in Deep Learning has sent the quality of results in multimedia tasks rocketing: thanks to the creation of big datasets of annotated images and videos, Deep Neural Networks (DNN) have outperformed other models in most cases.

In this thesis, we aim at developing novel DNN models for automatically deriving semantic representations of images and videos. In particular we focus on two main tasks : vision-text matching and image/video automatic captioning.

Addressing the matching task can be done by comparing visual objects and texts in a visual space, a textual space or a multimodal space. In this thesis, we experiment with these three possible methods. Moreover, based on recent works on capsule networks, we define two novel models to address the vision-text matching problem: Recurrent Capsule Networks and Gated Recurrent Capsules. We find that replacing Recurrent Neural Networks usually used for natural language processing such as Long Short-Term Memories or Gated Recurrent Units by our novel models improve results in matching tasks. On top of that, we show that intrinsic characteristics of our models should make them useful for other tasks.

In image and video captioning, we have to tackle a challenging task where a visual object has to be analyzed, and translated into a textual description in natural language. For that purpose, we propose two novel curriculum learning methods. Experiments on captioning datasets show that our methods lead to better results and faster convergence than usual methods. Moreover regarding video captioning, analyzing videos requires not only to parse still images, but also to draw correspondences through time. We propose a novel Learned Spatio-Temporal Adaptive Pooling (L-STAP) method for video captioning that combines spatial and temporal analysis. We show that our L-STAP method outperforms state-of-the-art methods on the video captioning task in terms of several evaluation metrics.

Extensive experiments are also conducted to discuss the interest of the different models and methods we introduce throughout this thesis, and in particular how results can be improved by jointly addressing the matching task and the captioning task.

RÉSUMÉ EN FRANÇAIS

Décrire des images ou des vidéos est une tâche que nous sommes tous en mesure d'accomplir avec succès depuis notre plus tendre enfance. Toutefois pour une machine, générer automatiquement des descriptions visuelles ou bien associer des images ou vidéos à des textes descriptifs est une tâche ardue, car elle nécessite d'extraire des informations sémantiques complexes à partir de ces images ou vidéos. Des recherches récentes en apprentissage profond ont fait monter en flèche la qualité des résultats dans les tâches multimédias : grâce à la création de grands ensembles de données d'images et de vidéos annotées, les réseaux de neurones profonds ont surpassé les autres modèles dans la plupart des cas. En particulier, nous pouvons citer le jeu de données ImageNet [16], constitué d'images réparties selon différentes catégories sémantiques. L'utilisation de cet immense jeu de données a permis de tirer profit des capacités des réseaux de neurones convolutionnels, et d'obtenir à partir de 2012 en classification d'images des résultats dépassant très largement ceux de l'état de art de l'époque [50]. Suite à cela, l'engouement pour les techniques d'apprentissage profond a été à l'origine d'un réel basculement dans les problèmes liés à l'intelligence artificielle, notamment concernant la vision par ordinateur ou le traitement du langage naturel. Ainsi, dans de nombreux problèmes multimédias, les solutions les meilleures actuellement font souvent appel aux réseaux de neurones convolutionnels en vision par ordinateur, ou aux réseaux de neurones récurrents [13, 38] en traitement du langage naturel. Cependant, la recherche de nouveaux modèles plus performants n'est pas au point mort, et de nouvelles idées semblent aujourd'hui prometteuses. Citons notamment les réseaux de capsules en vision par ordinateur, ou les transformateurs (*Transformers* en anglais) en traitement du langage naturel.

Dans ce travail de recherche, nous visons à développer de nouveaux modèles de réseaux de neurones profonds permettant de générer automatiquement des représentations sémantiques d'images et de vidéos. En particulier, nous nous concentrons sur deux tâches principales : la mise en relation de la vision et du langage naturel et la génération automatique de descriptions visuelles.

Nos contributions ont été les suivantes :

- nous avons défini, testé et validé deux modèles nouveaux pour l'appariement texte-vision :
    - les réseaux de capsules récurrentes ;
    - les *Gated Recurrent Capsules ;*

- nous avons défini, testé et validé un modèle nouveau pour la génération de descriptions visuelles de vidéos, et proposé une nouvelle méthode d'entraînement des modèles de génération de descriptions :
  - un modèle fondé sur une nouvelle méthode de *pooling* adaptatif spatio-temporel que nous proposons, permettant d'obtenir des représentations sémantiques fines des vidéos ;
  - une méthode d'apprentissage par curriculum adaptatif.

Ce résumé se présente ainsi : nous commençons par dresser un état de l'art du sujet. Nous présentons ensuite nos contributions. Enfin, nous expliquons de quelle manière nos modèles ont été évalués pour en justifier la pertinence.

ÉTAT DE L'ART

Appariements vision-texte

L'appariement d'images et de textes ou de vidéos et de textes suppose de trouver une représentation commune aux deux modalités en question. Plusieurs travaux ont en particulier cherché à construire des représentations vectorielles communes aux images ou vidéos et aux textes descriptifs. Dans [73], Frome *et al.* ont construit de telles représentations vectorielles à l'aide d'un modèle de type *skip-gram* [65] pour la partie textuelle, et à l'aide d'un réseau convolutionnel AlexNet [50] pour la partie visuelle. Karpathy *et al.* quant à eux ont proposé l'utilisation de représentations vectorielles par fragments [44]. L'idée sous-jacente est la suivante : comme les différentes parties d'une phrase descriptive correspondent à différentes parties d'une image, ils ont construit un modèle dressant des correspondances entre des bouts de phrases et des fragments d'images. Le modèle proposé par Kiros *et al.* dans [47] fonctionne de la manière suivante : les phrases descriptives sont traitées par une LSTM produisant des représentations vectorielles, lesquelles sont projetées dans un espace multimodal vers lequel sont également projetées des représentations visuelles générées à l'aide d'un réseau convolutionnel. Les deux projections (textuelle et visuelle) sont mises en correspondance par l'optimisation d'une fonction objectif visant à maximiser la similarité cosinus entre un vecteur d'image et un vecteur de description textuelle correspondante. La plupart des travaux sur l'appariement vision-langage portent sur des images ; il convient cependant de noter que des travaux sur les vidéos existent également, et qu'ils sont dans le principe assez similaires aux travaux existants sur les images [66].

D'autres travaux reposent sur l'utilisation de vecteurs de Fisher [73]. Dans [48], Klein *et al.* calculent des vecteurs de Fisher pour des phrases en se fondant sur un modèle de mélange gaussien (MMG) et un modèle de mélange hybride laplacien-gaussien (MMHLG). Dans [54], Lev *et al.* proposent un réseau récurrent faisant office de modèle génératif, utilisé en lieu et place des modèles probabilistes cités précédemment. Dans les deux derniers travaux cités, les relations entre vision et texte sont établies à l'aide d'un algorithme d'analyse canonique des corrélations [40]. Dans [24], Eisenschtat et Wolf calculent également des vecteurs de Fisher à l'aide d'un MMG et d'un MMHLG. Cependant, l'originalité de leur travail repose sur l'utilisation d'un réseau de neurones "2-Way" au lieu d'un algorithme d'analyse canonique des corrélations.

Des architectures plus complexes ont également été proposées par d'autres auteurs. Niu *et al.* [68] ont défini une LSTM hiérarchique multimodale, représentant les phrases sous forme d'arbres, ce qui permet d'établir des correspondances entre groupes de mots et régions d'une image. Nam *et al.* [67] ont proposé un réseau à attention duale (*Dual Attention Network*) reposant sur un mécanisme d'attention conjointe entre régions d'une image et mots d'une phrase descriptive. Plus récemment en 2018, Gu *et al.* [31] ont montré que l'entraînement conjoint de modèles génératifs (génération de phrases à partir d'images et génération d'images à partir de phrases) et de modèles d'appariement d'images et de textes permettait d'augmenter significativement l'efficacité de ces modèles d'appariement d'images et de textes. Enfin, il convient de noter que les derniers travaux en date reposent sur l'utilisation de caractéristiques locales extraites à l'aide d'un Faster R-CNN [75], un modèle de détection d'objets [53].

Génération de descriptions visuelles

La génération de descriptions visuelles peut être vue comme une tâche de traduction : une image ou une séquence d'images doivent être traduites dans un langage cible.

Certains travaux d'avant-garde comme [77] ont fait usage de méthodes de traduction automatique statistique pour générer des descriptions visuelles de vidéos. Cependant aujourd'hui, la plupart des travaux en génération de descriptions textuelles d'images ou de vidéos font usage de techniques d'apprentissage profond, en particulier fondées sur une architecture de type encodeur-décodeur telle que celle proposée dans [85] pour de la traduction d'un langage source vers un langage cible. Par ailleurs, l'emploi d'un mécanisme d'attention lors de la phase de décodage sur les états cachés de l'encodeur a permis de remarquables progrès en traduction automatique neuronale [60], ce qui a été confirmé par [101] pour la génération de textes descriptifs.

Pour les images, les principales améliorations de l'architecture basique encodeur-decodeur avec mécanisme d'attention font usage de techniques d'apprentissage par renforcement [76] ou de vecteurs de détections d'objets [3].

Concernant la génération de descriptions visuelles de vidéos, il est à noter que dans certains travaux, les vidéos sont sont divisées en trames, des caractéristiques globales sont calculées par trame à l'aide de réseaux convolutionnels [35, 50, 81, 86], et les vecteurs de caractéristiques ainsi obtenus sont traités séquentiellement par un encodeur [33, 55, 59, 69, 93, 96]. Le défaut de telles approches est qu'en se contentant de caractéristiques globales, elles perdent les informations d'ordre local, et donc de la précision.

D'autres approches permettent de pallier ce problème de perte d'informations locales. Dans [101], les auteurs divisent leur model en deux parties : un encodeur-décodeur usuel traite des caractéristiques globales, tandis qu'un réseau convolutionnel à trois dimensions est utilisé pour calculer des caractéristiques locales aidant à améliorer la générations des phrases descriptives. Dans [103], les auteurs font usage de caractéristiques locales pour suivre les trajectoires des différents objets présents dans les vidéos. Dans [97] également, les auteurs proposent une méthode permettant de suivre la trajectoire des objets au long de la vidéo. Dans les deux cas, les trajectoires sont combinées avec des vecteurs de caractéristiques globales afin de calculer des représentations sémantiques vectorielles des vidéos. Dans [94], des vecteurs de caractéristiques locales sont calculés et utilisés pour générer des phrases descriptives à l'aide d'un mécanisme d'attention. Enfin, d'autres travaux ont tenté d'utiliser des réseaux convolutionnels à trois dimensions [95] ou des réseaux récurrents convolutionnels [94] pour dresser des correspondances temporelles entre caractéristiques locales.

NOS CONTRIBUTIONS

Il est possible de traiter la tâche d'appariement en comparant des images ou vidéos et des textes dans un espace visuel, un espace textuel ou un espace multimodal. Dans cette thèse, nous expérimentons ces trois méthodes. De plus, sur la base de travaux récents sur les réseaux de capsules, nous définissons deux nouveaux modèles pour résoudre le problème de correspondance vision-texte : les réseaux de capsules récurrentes (*Recurrent Capsule Networks*) et les *Gated Recurrent Capsules*, inspirées des *Gated Recurrent Units*. Nous constatons que le remplacement des réseaux de neurones récurrents habituellement utilisés pour le traitement du langage naturel, tels que les *Long Short-Term Memories* ou les *Gated Recurrent Units*, par nos nouveaux modèles améliore les résultats des tâches de mise en relation vision-texte. De plus, nous expliquons que les caractéristiques intrinsèques de nos modèles pourraient potentiellement les rendre utiles pour d'autres tâches.

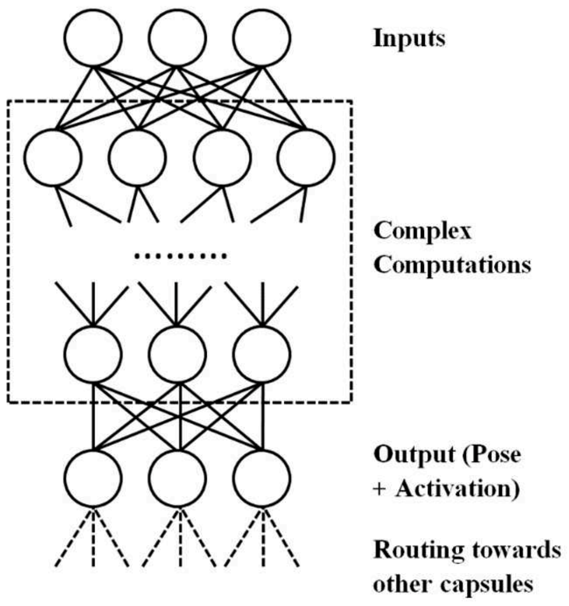


Figure 1 : Une caspsule générique pour vision par ordinateur

Appariement vision-langage

Une partie significative des modèles d'appariement vision-texte aujourd'hui utilisés sont constitués d'une partie en charge de l'analyse visuelle et d'une autre partie en charge de l'analyse textuelle (souvent un réseau de neurones récurrent). Nous nous sommes attachés à tenter d'améliorer les réseaux de neurones récurrents usuellement employés pour effectuer des tâches de traitement du langage naturel.

Comme expliqué précédemment, il a été montré que les réseaux de capsules avaient des résultats prometteurs en vision par ordinateur. Nous nous sommes demandés si l'idée de l'utilisation de capsules pouvait également améliorer les résultats obtenus en traitement du langage naturel. L'idée derrière les capsules telles que décrites dans [80] pour la vision par ordinateur est représentée dans la Figure 1. En vision par ordinateur, une capsule a pour rôle d'effectuer des calculs complexes et de renvoyer en sortie un vecteur de position et une activation (au lieu d'une simple activation pour un neurone classique). Ce vecteur de sortie est ensuite dirigé vers les capsules suivantes par le biais d'un algorithme de routage prédéfini. Le but de cette architecture est que chaque capsule apprenne à reconnaître un type de caractéristique visuelle, indépendamment des transformations qui lui ont été appliquées. Par exemple, certaines capsules pourraient reconnaître des yeux, un nez, une bouche et leur positions respectives. Elles enverraient ensuite leurs activations et vecteurs de positions vers une capsule suivante qui aurait pour but de reconnaître un visage entier. La sortie d'une capsule contenant non seulement une activation mais aussi des informations positionnelles, elles ne subissent pas les

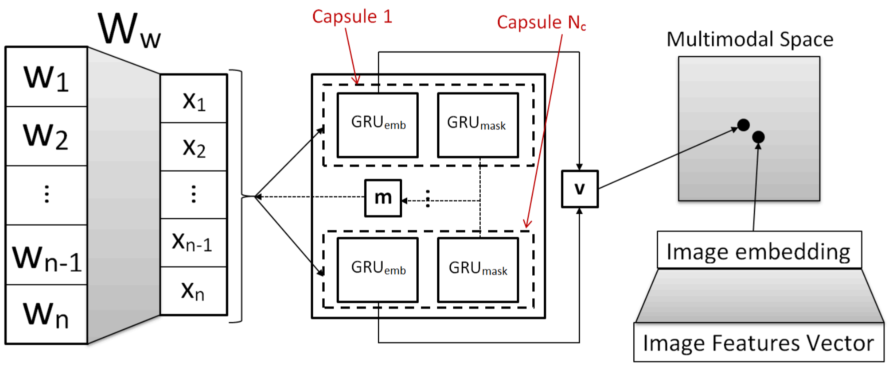


FIGURE 2 : Notre modèle. Les capsules sont représentées par des boîtes en pointillés. Les phrases sont représentées par des séquences d'encodages dits *one-hot* $(w_1, ..., w_n)$. Elles sont ensuite transformées en des séquences de vecteurs de mots de dimensions plus petites par le biais d'une multiplication par une matrice d'encodages W_w. Les phrases sont ensuite traitées par notre réseau de capsules récurrentes, qui finit par générer une représentation vectorielle v. Cette représentation vectorielle est comparée à une représentation vectorielle d'image appartenant au même espace multimodal.

défauts des opérations de *pooling*, qui sont obligatoires dans un réseau convolutionnel, et qui entraînent une perte d'information spatiale. Les deux modèles que nous avons imaginés sont inspirés des réseaux de capsules, et sont adaptés au traitement du langage naturel. Nous ne présenterons pas ici comment nous traitons les images pour les apparier avec des phrases car nous le faisons de façon très classique : nous extrayons des vecteurs de caractéristiques à l'aide d'un réseau convolutionnel, que nous projetons dans le même espace multimodal que les représentations vectorielles de phrases que nous générons. Les sections qui suivent font état de nos contributions, qui ont consisté en la définition de modèles nouveaux pour le traitement du langage naturel.

Réseaux de capsules récurrentes (voir Figure 2)

Le premier de ces modèles est un réseau de capsules récurrentes. Ici, chaque capsule contient deux GRU. Le rôle de la première est d'extraire ce que nous appelons un "masque" (l'équivalent d'une activation pour les capsules en vision par ordinateur). La seconde produit une représentation vectorielle de phrase, que nous pouvons assimiler au vecteur de position. La sortie de la première GRU de la i-ème capsule est notée $\text{GRU}_{\text{mask}_i}(\text{input})$ et la sortie de la deuxième GRU de la i-ème capsule est notée $\text{GRU}_{\text{emb}_i}(\text{input})$. Dans la première GRU, la fonction tanh est remplacée par une sigmoïde, de telle sorte que les masques soient constitués uniquement de nombres positifs entre 0 et 1. Ces masques jouent le rôle d'un mécanisme d'attention.

De façon plus formelle, si s est une phrase, nous codons chaque mot de s à l'aide d'un encodage *one-hot* : nous avons $s = (w_1, ..., w_L)$ avec L la longueur de s, et $w_1, ..., w_L$ appartenant à $\mathbb{R}^D$, où D est la taille de notre vocabulaire. Soit $W_w \in \mathbb{R}^{D \times V}$ une matrice de représentations vectorielles de mots. On notera dans la suite $x = W_w s$ au lieu de $(x_1, ..., x_L) = (W_w w_1, ..., W_w w_L)$ pour simplifier les notations. Si m est un vecteur de dimension V, alors on notera également $m \odot x$ au lieu de $(m \odot x_1, ..., m \odot x_L)$. Dans la suite, m est un masque et v est une représentation vectorielle de phrase. Ces représentations vectorielles sont calculées ainsi :

$$v_i^{(t)} = \mathrm{GRU}_{\mathrm{emb}_i}(m_i^{(t-1)} \odot x). \tag{1}$$

Les masques sont calculés en deux étapes. Tout d'abord, les capsules produisent un masque en fonction de leur phrase d'entrée et des masques calculés à l'étape précédente :

$$\tilde{m}_i^{(t)} = \mathrm{GRU}_{\mathrm{mask}_i}(m_i^{(t-1)} \odot x). \tag{2}$$

Ensuite, les m_i sont calculés en effectuant une somme pondérée de ces masques :

$$m_i^{(t)} = \sum_{j=1}^{N_c} \alpha_{ij}^{(t)} \tilde{m}_i^{(t)} \tag{3}$$

et les représentations vectorielles finales sont calculées ainsi :

$$v^{(t)} = \sum_{i=1}^{N_c} \sum_{j=1}^{N_c} \beta_{ij}^{(t)} \tilde{v}_i^{(t)}. \tag{4}$$

Les coefficients des sommes pondérées sont calculés selon les formules suivantes ($\langle v_1 | v_2 \rangle$ représente le produit scalaire usuel entre deux vecteurs v_1 et v_2) :

$$\alpha_{ij}^{(t)} = \frac{\left\langle v_i^{(t)} | v_j^{(t)} \right\rangle}{\sum_{k=1}^{N_c} \left\langle v_i^{(t)} | v_k^{(t)} \right\rangle}, \tag{5}$$

$$\beta_{ij}^{(t)} = \frac{\left\langle v_i^{(t)} | v_j^{(t)} \right\rangle}{\sum_{k=1}^{N_c} \sum_{l=1}^{N_c} \left\langle v_k^{(t)} | v_l^{(t)} \right\rangle}. \tag{6}$$

Nous attirons l'attention du lecteur sur le fait que pour $t = 0$, les masques sont des vecteurs dont toutes les coordonnées sont égales à

1 : cela revient à simplement entrer les phrases dans les GRU sans leur appliquer de masque. Les formules que nous avons définies peuvent être interprétées de façon intuitive : des capsules contribuent d'autant plus au calcul des masques et des représentations vectorielles finales qu'elles produisent des représentations vectorielles similaires. C'est en quelque sorte une variante du routage par accord (*routing-by-agreement*) tel que défini dans [80].

Gated Recurrent Capsules (GRC, voir Figure 3)

Schématiquement, notre but avec les GRC est de produire différentes représentations sémantiques pour une même phrase mettant l'accent sur ses différents éléments. Ainsi, une phrase est en quelque sorte divisée en sous-phrases, dont chacune prêterait attention à un élément particulier d'une image. Ces représentations de sous-phrases sont ensuite traitées pour construire une représentation globale de la phrase entière.

Dans notre modèle, toutes les capsules partagent les mêmes paramètres et sont similaires à des GRU. Dans la suite, nous allons expliquer les différences principales entre celles-ci et les GRU. Formellement, si nous considérons la k-ème capsule avec k dans $\{1, ..., N_c\}$, les équations des portes des *update gates* et des *reset gates* sont les mêmes que pour une GRU :

$$u_t^{(k)} = \sigma(W_{xu}x_t + W_{hu}h_{t-1}^{(k)} + b_u), \tag{7}$$

$$r_t^{(k)} = \sigma(W_{xr}x_t + W_{hr}h_{t-1}^{(k)} + b_r), \tag{8}$$

Nous calculons également $\tilde{h}_t^{(k)}$ comme nous le ferions dans une GRU :

$$\tilde{h}_t^{(k)} = \tanh(W_{xh}x_t + W_{hh}(r_t^{(k)} \odot h_{t-1}^{(k)}) + b_h), \tag{9}$$

Nous supposons maintenant que pour chaque capsule, pour un mot donné w_t, nous avons un coefficient $p_t^{(k)} \in [0, 1]$ tel que :

$$h_t^{(k)} = (1 - p_t^{(k)})h_{t-1}^{(k)} + p_t^{(k)}\hat{h}_t^{(k)} \tag{10}$$

avec

$$\hat{h}_t^{(k)} = u_t^{(k)} \odot \tilde{h}_t^{(k)} + (1 - u_t^{(k)}) \odot h_{t-1}^{(k)}, \tag{11}$$

ce qui correspond en fait à la mise à jour de l'état caché telle qu'elle est effectuée dans une GRU. Le coefficient $p_t^{(k)}$ est un coefficient de routage, décrivant à quel point une capsule donnée doit voir son état caché être mis à jour par le mot entré. De la même manière que dans [37], le routage peut être vu comme un mécanisme d'attention ; dans notre cas, l'attention est portée sur les mots considérés comme étant pertinents. Cependant, contrairement aux auteurs de [37], qui font usage de gaussiennes déterminées par espérance-maximisation pour calculer ce coefficient, nous proposons de le calculer d'une manière plus simple, comme nous l'expliquons un peu plus loin. Finalement, nous obtenons l'équation suivante :

$$h_t^{(k)} = (1 - p_t^{(k)} u_t^{(k)}) \odot h_{t-1}^{(k)} + p_t^{(k)} u_t^{(k)} \odot \tilde{h}_t^{(k)}. \tag{12}$$

Nous pouvons remarquer que cela revient à multiplier les *update gates* $u_t^{(k)}$ par un coefficient $p_t^{(k)}$. Il nous reste ensuite à calculer $p_t^{(k)}$. Pour ce faire, nous définissons un coefficient d'activation $a_t^{(k)}$ pour chaque capsule :

$$a_t^{(k)} = |\alpha_k| + \log(P_t^{(k)}). \tag{13}$$

Dans la dernière équation, les α_k sont des nombres aléatoires tirés d'une distribution gaussienne de moyenne 0,1 et d'écart-type 0,001. Ces α_k sont importants car les capsules partagent les mêmes paramètres : si toutes les activations sont les mêmes au début, elles resteront les mêmes en fin de calcul. Ces nombres aléatoires cassent la symétrie existant *de facto* entre les différentes capsules. $P_t^{(k)}$ a pour rôle de représenter la similarité sémantique entre l'état caché actuel de la capsule $h_{t-1}^{(k)}$ et le mot en entrée x_t : si celui-ci est sémantiquement proche de l'état caché précédent, alors $P_t^{(k)}$ doit être haut, et s'il est très différent cette grandeur doit être faible. On peut intuitivement imaginer que la similarité cosinus $\cos(h_{t-1}^{(k)}, \hat{h}_t^{(k)}) = \frac{\left\langle h_{t-1}^{(k)} | \hat{h}_t^{(k)} \right\rangle}{\|h_{t-1}^{(k)}\|_2 \times \|\hat{h}_t^{(k)}\|_2}$ correspond à une définition pertinente de la similarité sémantique entre l'état caché actuel de la capsule et le mot en entrée : si le mot en entrée a un sens différent des mots précédents, on peut s'attendre à ce que $\hat{h}_t^{(k)}$ reflètera cette différence de sens. C'est pourquoi nous définissons $P_t^{(k)}$ de la façon suivante :

$$P_t^{(k)} = \cos(h_{t-1}^{(k)}, \hat{h}_t^{(k)}). \tag{14}$$

Nous pouvons ensuite calculer $p_t^{(k)}$ à l'aide de la formule suivante :

$$p_t = \frac{\text{softmax}(\frac{a_t^{(1)}}{T}, ..., \frac{a_t^{(N)}}{T})}{M} \tag{15}$$

où M est la coordonnée de valeur maximale du vecteur $\text{softmax}(\frac{a_t^{(1)}}{T}, ..., \frac{a_t^{(N)}}{T})$ et T est un hyperparamètre contrôlant la finesse de la procédure de routage : plus T est grand, plus le plus grand poids de routage est proche de 1 et les autres proches de 0.

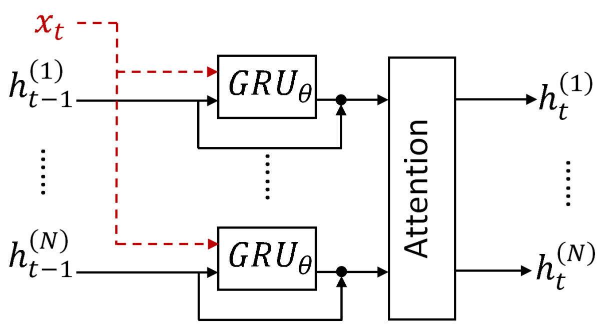


FIGURE 3 : Gated Recurrent Capsules : toutes les capsules partagent les mêmes paramètres appris θ. Les entrées de la capsule i au temps t sont des vecteurs de mots x_t et son état caché au temps $t-1$ est $h_{t-1}^{(i)}$. Sa sortie est $h_t^{(i)}$. Cette procédure de routage peut être vue comme un mécanisme d'attention : chaque sortie dépend des similarités sémantiques des mots de la phrase traitée avec les précédents mots de cette même phrase. Elle permet que chaque capsule génère une représentation vectorielle de phrase mettant l'accent sur un élément particulier de la phrase..

Le routage que nous proposons est différent de celui qui a été présenté dans [37, 80] : les sorties des capsules ne sont pas des combinaisons de toutes les précédentes capsules. Seuls les poids de routage dépendent de ces précédentes capsules.

Génération de descriptions visuelles (voir Figure 4)

Concernant la tâche de génération de descriptions visuelles, nous devons nous attaquer à un problème difficile dans lequel une image ou une vidéo doivent être analysés et traduits en une description textuelle en langage naturel. À cette fin, nous proposons deux nouvelles méthodes d'apprentissage par curriculum. Les expériences sur les jeux de données de génération de textes descriptifs montrent que nos méthodes conduisent à de meilleurs résultats et à une convergence plus rapide que les méthodes habituelles. En outre, en ce qui concerne la génération de descriptions de vidéos, l'analyse de celles-ci implique non seulement l'analyse d'images fixes, mais également l'établissement de correspondances dans le temps entre les différentes trames. Nous proposons une nouvelle méthode de regroupement adaptatif spatio-temporel (*Spatio-Temporal Adaptive Pooling* ou L-STAP) pour la génération de descriptions de vidéos, qui combine une analyse spatiale et une analyse temporelle de caractéristiques sémantiques visuelles de ces vidéos. Nous montrons que notre méthode L-STAP surpasse les méthodes de l'état de l'art en matière de génération de descriptions de vidéos selon plusieurs métriques d'évaluation.

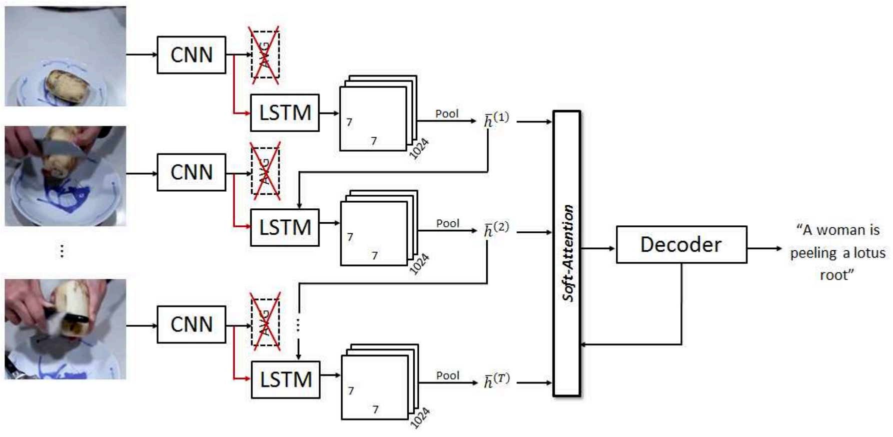


Figure 4 : Illustration de notre modèle, fondé sur la méthode L-STAP que nous proposons. Les trames sont traitées séquentiellement par un réseau convolutionnel (un ResNet-152 dans notre cas). Cependant, au lieu d'appliquer un regroupement par moyennage (*average pooling*) sur les caractéristiques locales comme cela se fait dans de nombreux travaux récents, nous faisons usage d'une LSTM pour détecter les dépendances temporelles. Des états cachés locaux sont calculés pour obtenir un tenseur de dimensions 7x7x1024. Ces états cachés locaux sont ensuite regroupés ensemble, soit en les moyennant soit en faisant usage d'un mécanisme d'attention, et traités par une LSTM permettant de produire une phrase descriptive.

Learned Spatio-Temporal Adaptive Pooling

La première étape est de générer une représentation vectorielle de la vidéo dont nous voulons obtenir une description textuelle.

Étant donnée une vidéo $V = (v^{(1)}, ..., v^{(T)})$, nous devons calculer des vecteurs de caractéristiques pour chaque trame $v^{(t)}$. Une façon usuelle de faire est de traiter chaque trame à l'aide d'un réseau convolutionnel pré-entraîné sur un grand jeu de données. Dans des travaux tels que [59], la sortie de l'avant-dernière couche d'un ResNet-152 a été choisie comme représentation de trame (il s'agit d'un vecteur de dimension 2048). Cependant, de telles représentations font abstractions des caractéristiques locales des trames, ce qui cause une perte d'information. C'est la raison pour laquelle nous avons choisi de récupérer plutôt la sortie de la dernière couche convolutionnelle d'un ResNet-152. Cela nous a permis d'obtenir des vecteurs de caractéristiques locales $(x^{(1)}, ..., x^{(T)}) = X$, avec $x^{(t)} \in \mathbb{R}^{7\times7\times2048}$ pour tout t. L'étape suivante est de combiner ces représentations locales, afin d'obtenir une représentation plus fine qu'avec un simple moyennage de vecteurs de caractéristiques locales.

Le but de la méthode L-STAP que nous proposons est de remplacer en fait l'opération de regroupement par moyennage (*average pooling*) après la dernière couche convolutionnelle du ResNet-152, et de prendre en considération dans notre regroupement l'évolution à la fois temporelle et spatiale des caractéristiques visuelles. Ainsi, on espère récupérer les endroits et moments de la vidéo où des actions importantes arrivent, et rejeter les endroits et moments n'étant pas pertinents pour générer une description textuelle concise de la vidéo. Pour ce faire, nous faisons usage d'une LSTM, prenant les caractéristiques locales comme entrées, ce qui nous permet d'obtenir des états cachés locaux ceux-ci sont combinés à l'aide d'une somme pondérée que nous décrivons en fin de section. Plus formellement, soient les caractéristiques locales $x_{ij}^{(t)} \in \mathbb{R}^{2048}$. Les caractéristiques locales regroupées $h_{ij}^{(t)}$ sont calculées de la façon suivante :

$$i_{ij}^{(t)} = \sigma(W_{ix}x_{ij}^{(t)} + W_{ih}\overline{h}^{(t-1)} + b_i) \tag{16}$$

$$f_{ij}^{(t)} = \sigma(W_{fx}x_{ij}^{(t)} + W_{fh}\overline{h}^{(t-1)} + b_f) \tag{17}$$

$$o_{ij}^{(t)} = \sigma(W_{ox}x_{ij}^{(t)} + W_{oh}\overline{h}^{(t-1)} + b_o) \tag{18}$$

$$c_{ij}^{(t)} = f_{ij}^{(t)} \circ \overline{c}^{(t-1)} + i_{ij}^{(t)} \tanh(W_{cx}x_{ij}^{(t)} + W_{ch}\overline{h}^{(t-1)} + b_c) \tag{19}$$

$$h_{ij}^{(t)} = o_{ij}^{(t)} \circ \tanh(c_{ij}^{(t)}) \tag{20}$$

où W_{ix}, W_{ih}, b_i, W_{fx}, W_{fh}, b_f, W_{ox}, W_{oh}, b_o, W_{cx}, W_{ch} et b_c sont des paramètres définis par descente de gradient, et $\overline{c}^{(t-1)}$ et $\overline{h}^{(t-1)}$ sont respectivement la mémoire et l'état caché de la LSTM. Notons que les mémoires et états cachés sont partagés en chaque endroit de la vidéo à l'instant t. La mémoire et l'état caché à l'instant t sont calculés ainsi :

$$\overline{c}^{(t)} = \sum_{i=1}^{7}\sum_{j=1}^{7} \alpha_{ij}^{(t)} c_{ij}^{(t)} \tag{21}$$

$$\overline{h}^{(t)} = \sum_{i=1}^{7}\sum_{j=1}^{7} \alpha_{ij}^{(t)} h_{ij}^{(t)} \tag{22}$$

où les $\alpha_{ij}^{(t)}$ sont des poids locaux. Nous avons testé deux types de poids locaux. Les premiers sont des poids uniformes :

$$\alpha_{ij}^{(t)} = \frac{1}{7 \times 7} \tag{23}$$

ce qui correspond en fait à un moyennage arithmétique simple des mémoires et des états cachés. Les deuxièmes types de poids ont été calculés à l'aide d'un mécanisme d'attention, comme suit :

$$\tilde{\alpha}_{ij}^{(t)} = w^{T} \tanh(W_{\alpha x} x_{ij}^{(t)} + W_{\alpha h} \overline{h}^{(t-1)} + b_{\alpha}). \tag{24}$$

$$\alpha_{ij}^{(t)} = \frac{\exp(\tilde{\alpha}_{ij}^{(t)})}{\sum_{k=1}^{7} \sum_{l=1}^{7} \exp(\tilde{\alpha}_{kl}^{(t)})}, \tag{25}$$

où $W_{\alpha x}$, $W_{\alpha h}$, b_{α} sont des paramètres définis lors de l'entraînement par descente de gradient.

Nous obtenons ainsi des représentations locales agrégées des vidéos, lesquelles sont utilisées en entrée d'une LSTM pour générer des phrases descriptives.

Apprentissage par curriculum

L'apprentissage par curriculum s'intéresse à la façon dont des données doivent être présentées pendant l'entraînement d'une intelligence artificielle. De la même manière qu'un professeur suit un programme (*curriculum* en anglais) pour enseigner à ses élèves, il peut être pertinent de prédéfinir des règles selon lesquelles l'apprentissage d'une intelligence artificielle doit être mené. Il a été montré que dans certains cas, l'apprentissage par curriculum pouvait accélérer la convergence d'un modèle, voire améliorer ses résultats [9].

Nous avons défini deux méthodes d'apprentissage par curriculum permettant d'entraîner des modèles de génération de textes descriptifs. La première méthode est une simple adaptation de [9]. La complexité d'une image ou d'une vidéo annotée par des phrases descriptives a été évaluée en calculant le score self-BLEU [108] du corpus de phrases descriptives correspondantes. En effet, le score self-BLEU permet de mesurer la diversité d'un corpus de textes : si les phrases descriptives correpsondant à une même image ou une même vidéo sont très diverses, on peut en déduire que le contenu de cette image ou de cette vidéo est sémantiquement complexe. Nous avons ensuite utilisé ce score pour présenter les données d'entraînement en introduisant petit à petit des éléments plus complexes.

La deuxième méthode que nous avons proposée est une méthode d'apprentissage par curriculum adaptatif : les données d'entraînement ont d'autant plus de chances d'être présentées au modèle que celui-ci est peu performant sur celles-ci. Pour calculer la performance d'un modèle sur une image ou une vidéo, nous avons procédé de la façon suivante. Soit V un objet visuel (image ou vidéo), s une phrase descriptive de la réalité-terrain, et $\hat{s}$ une phrase générée par notre modèle. Nous supposons que la fonction $r : V, s \mapsto r(V, s)$ associe un score numérique à un couple objet visuel-phrase descriptive ; dans notre cas, nous avons utilisé la métrique CIDEr. Le score associé à (V, s) pour notre modèle est défini de la manière suivante :

$$P(V, s, \hat{s}) = \min(1, \exp(\lambda(-\alpha + r(V, s) - r(V, \hat{s})))), \quad (26)$$

où λ et α sont des hyperparamètres. $P(V, s, \hat{s})$ correspond à la probabilité que le couple (V, s) soit présenté au modèle comme donnée d'entraînement.

ÉVALUER NOS RÉSULTATS

Différents jeux de données et différentes métriques ont été utilisés pour évaluer nos modèles par rapport à l'état de l'art. Les principaux jeux de données utilisés sont Flickr8k, Flickr30k et MSCOCO pour les tâches de type image-texte et MSVD et MSR-VTT pour les tâches de type video-texte. Pour les tâches d'appariement, nos résultats sont évalués en termes de rappel, et pour les tâches de génération de textes descriptifs, nous avons fait usage des métriques BLEU, ROUGE, METEOR et CIDEr usuellement employées pour évaluer de telles tâches.

CONCLUSIONS

Notre travail a été l'occasion de proposer de nouveaux modèles et de nouvelles méthodes pour la création de représentations sémantiques d'images ou de vidéos : représentations vectorielles multimodales, utiles pour les tâches d'appariement vision-langage, et aussi des représentations textuelles en langage naturel générées automatiquement. En particulier, il convient d'avoir à l'esprit les points suivants :

- Les modèles que nous avons proposés ont été comparés aux modèles usuellement employés. Nous avons montré que ceux-ci permettaient d'obtenir de meilleurs résultats qu'avec les méthodes usuelles.
- Par ailleurs, les modèles que nous avons proposés ont été appliqués à des tâches très spécifiques, mais nous estimons qu'ils pourraient également être utilisés dans un cadre plus général.
- Des expériences approfondies sont également menées pour discuter de l'intérêt des différents modèles et méthodes que nous avons présentés tout au long de cette thèse, et en particulier de la façon dont les résultats peuvent être améliorés en abordant conjointement la tâche de mise en correspondance et la tâche de génération de descriptions visuelles.

Les modèles et méthodes que nous avons proposés ont été appliqués dans le cadre du sujet de cette thèse ; néanmoins nous pensons que leur intérêt pourrait aller au-delà de ce sujet car ceux-ci sont conceptuellement généralisables à d'autres applications, notamment liées au traitement du langage naturel pour les réseaux de capsules récurrentes et les *gated recurrent capsules*, ou au traitement des vidéos pour notre méthode L-STAP.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS & NOTATIONS

ACRONYMS

CNN	Convolutional Neural Network
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
NLP	Natural Language Processing
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent

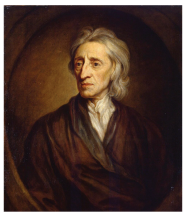
OPERATIONS

$\odot$	Hadamard product
A^T	Transpose matrix of A
$f \circ g$	Function composition of f and g
$\langle u \| v \rangle$	Scalar product of u and v
$\frac{\partial f}{\partial x}$	Partial derivative of f with respect to x
∇f	Gradient of f
$\mathcal{J}(f)$	Jacobian matrix of f

FUNCTIONS

exp	Exponential function
log	Natural logarithm function
cos	Cosine similarity function
σ	Sigmoid function
tanh	Hyperbolic tangent function
softmax	Softmax function

1 INTRODUCTION



"Reading furnishes the mind only with materials of knowledge; it is thinking that makes what we read ours."

— John Locke, *Of the Conduct of the Understanding*

Artificial intelligence has been a topic of interest since a long time ago. However, until recently, it was a topic falling into the category of fantasy or science fiction. Back to Ancient Greece, a legend told by Homer in the *Iliad* says that the god Hephaistos created multiple automatons to make daily life easier at the Olympus. The Renaissance has seen a lot of creativity in the conception of intelligent machines, with Leonardo da Vinci's automatons for instance – unfortunately, technical means at that time were not sufficient to implement these machines. Everyone knows the story of the Mechanical Turk designed by Wolfgang von Kempelen, who pretended that his automaton was able to play chess like a grandmaster; it appeared that someone was actually hidden inside and was making it move and play. In 1816, Ernest Theodor Amadeus Hoffmann wrote *The Sandman*, a very disturbing short story where Nathaniel, the main character, falls in love with an automaton: even though AI did not exist, some authors were already thinking of its implications.

Nowadays, artificial intelligence is not a fantasy anymore: efficient computers and big datasets have made AI part of our everyday life. Our work in this thesis has been to deal with one side of AI: how to give a meaning to what we see? Describing images or videos is a task that we all have been able to tackle since our earliest childhood. But how can we teach a machine to do the same thing? This is what our research work is about.

The first task that we focused on is the vision-text matching task. We illustrated that task on Figure 5. Humans can easily compare an image or a video and a text describing that image or video, and say whether the text is accurate or not. This is the matching task: given an image or video v and a describing sentence s, we aim at deriving a value $S(v, s)$ describing how much s correctly describes v.

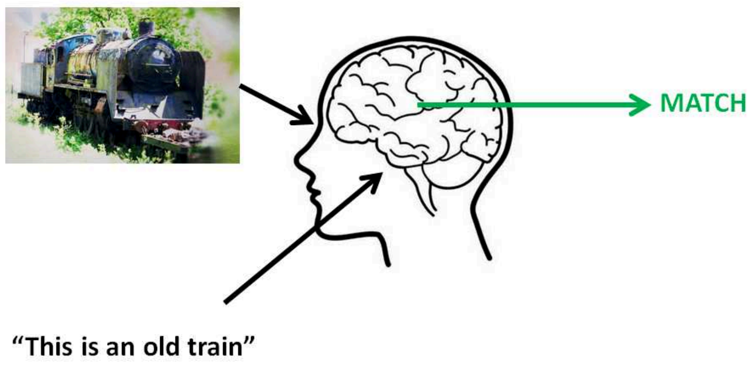


Figure 5: Illustration of the matching task. Given an image of an old train, and a text saying "this is an old train", a matching model should be able to say that the image and the text match.

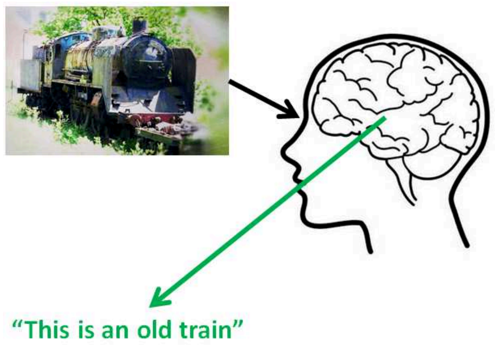


Figure 6: Illustration of the captioning task. Given an image of an old train, a captioning model should be able to output a describing sentence such as "this is an old train".

The second task on which we worked is the captioning task (see Figure 6 for an illustration). The captioning task is the following: given an image or a video, our goal is to generate a describing sentence in natural language, which should be as close as possible to the description that a human would have done. It means that giving a vague but accurate description is not sufficient : that description should also be as precise as a human description would be.

As we will see, deep neural networks can successfully address these tasks. In this thesis, we proposed several models or methods improving usual ones. The outline of this thesis is the following: in Chapter 2, we give an introduction to deep learning and present existing works in vision-text matching and in captioning. In Chapter 3, we introduce our work on the vision-text matching task. In Chapter 4, we introduce our work on the captioning task. We conclude the thesis in Chapter 5.

Our contributions in this work are as follows:

- In Chapter 3:

  - We proposed a method to weigh words in a sentence with respect to how the contribute to a visual description (Section 3.2.3.4).
  - We architectured a novel Recurrent Capsule Network for sentences embeddings. We showed that it outperformed usual models for sentences embeddings (Section 3.3).
  - We architectured a novel Gated Recurrent Capsule model for sentences embeddings. The model that we designed at Section 3.3 was efficient but was too computationally intensive. That novel model is much lighter, and we also showed that it outperformed usual models for sentences embeddings (Section 3.4).
  - We proposed a video-to-text matching model at Section 3.5 in the context of TRECVid VTT 2018, an evaluation campaign for video-to-text matching models. Our model scored third out of eleven competitors.
  - We proposed a fusion model for Ad-Hoc Video Search (Section 3.6).

- In Chapter 4:
  - We introduced at Section 4.2 a novel learned spatio-temporal adaptive pooling method that we designed for video processing and applied in the context of video captioning. We showed that a model based on this method outperformed state-of-the-art models on a standard video captioning dataset in terms of several metrics. We also assessed the interest of each element of our method through an ablation study.
  - We proposed an extension of a Curriculum Learning algorithm for captioning, and showed that it improved the training speed (Section 4.3.2).
  - We proposed a novel Adaptive Curriculum Learning algorithm for captioning. We showed on a standard dataset that our algorithm outperformed the usual gradient descent algorithm in the context of image captioning (Section 4.3.3).

2 PROLEGOMENA TO VISION AND NATURAL LANGUAGE PROCESSING

In this chapter, we will give some preliminary insights on Deep Learning and its applications in Computer Vision and Natural Language Processing. Then, we will explain how Deep Learning techniques have been used for matching vision and language, and for generating descriptions of images and videos in natural language.

2.1 TACKLING COMPLEXITY WITH DEEP LEARNING

Understanding photos and videos or texts in natural language is not straightforward for computers. However, Deep Neural Networks have had appalling results in these fields, even if they rely on artificial neurons, which are extremely simple mathematical objects, as we will see later in this section. How such simple objects can be used to solve such difficult problems? This is what we will explain in this section.

In this thesis, "artificial neuron" and "perceptron" will designate the same objects

2.1.1 *Artificial Neural Networks*

The idea of using mathematical objects inspired by neurons to modelize problems is not new: in 1943, McCulloch and Pitts make a link between real neuron networks and propositional logic [62]. In this section, we will define artificial neurons and how they can be combined to form artificial neural networks.

2.1.1.1 *The Perceptron*

Before defining artificial neurons (or perceptrons), let us describe briefly how real neurons work. Neurons are the main cells of the nervous system of all animals [1]. As shown on Figure 7, they are composed of dendrites, propagating electrochemical signals towards the cell body. If the sum of signals is above a certain threshold, then a resulting electrochemical signal is propagated in the axon towards subsequent neurons. According to stimuli from their environments, connections between their neurons, called synapses, are strengthened or weakened: this phenomenon is called Neuroplasticity. It allows them to act accordingly to environmental stimuli.

Perceptrons have been designed by Frank Rosenblatt in 1958 [78]. They are inspired by real neurons. As shown on Figure 8, external stimuli are represented by numerical values. A weighted sum

1 Apart from sponges and trichoplaxes...

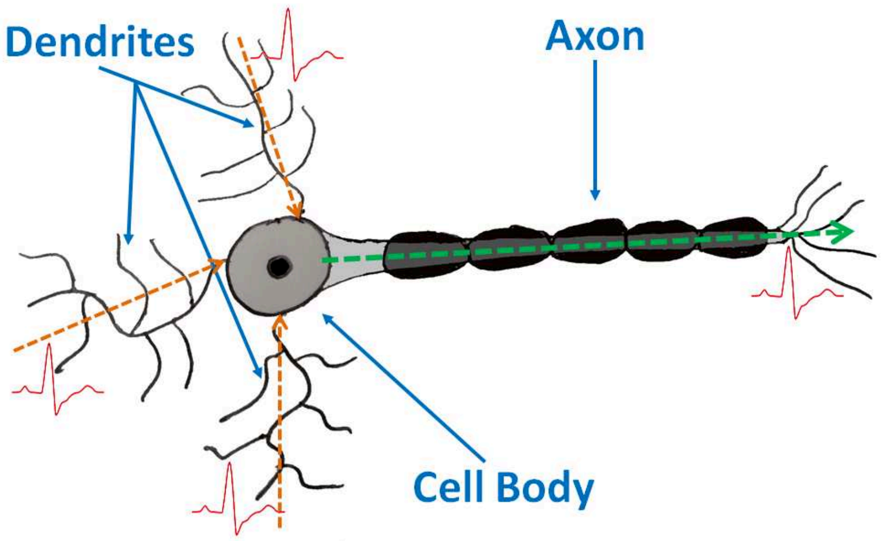


Figure 7: A sketch of a real neuron. Electrical signals (in orange) go to the cell body through dendrites. If the sum of signals is above a given threshold, then another electrical signal (in green) goes to other neurons through the axon.

combines them; weights correspond to the strength of synapses. A bias, corresponding to the threshold of real neurons, is added to the weighted sum. The numerical value thus obtained is then processed through an activation function, which defines the signal that is output by the perceptron. This activation function is generally a non-decreasing function. It is assumed to be differentiable, for reasons we will develop later.

Mathematically, a perceptron corresponds to an affine function from $\mathbb{R}^d$ to $\mathbb{R}$, where d is the number of incoming signals:

$$y = f(\langle w|x \rangle + b). \tag{27}$$

Perceptrons are often combined to deliver multi-dimensional signals. In that case, the previous equation become:

$$y = f(Wx + b) \tag{28}$$

where y is not a numerical value anymore but a real vector. Such a combination of perceptrons is called a layer of perceptrons.

A single perceptron, or even a layer of perceptrons are not sufficient to solve complex problems: even simple ones such as the XOR Problem cannot be solved by perceptrons, which are linear classifiers. We will see in next section how to combine layers of perceptrons to solve more complex problems.

2.1.1.2 *Multi-Layer Perceptrons*

As we stated in the previous section, if the sum of the electrochemical signals that enter a real neuron is above a certain threshold, it triggers

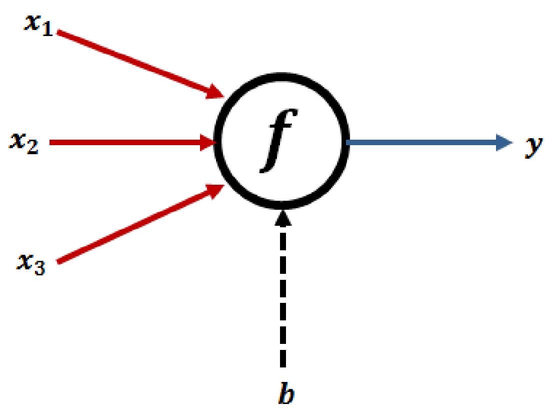


Figure 8: A sketch of an artificial neuron. Dendrites are replaced by three inputs x_1, x_2 and x_3. The artificial neuron makes a weighted sum of these inputs, and applies a transfer function f to compare it to a threshold b. The output y if actually the output of f: y is high when the weighted sum of inputs is high, and low if that weighted sum is low.

another signal that is also transferred to other neurons. One layer of perceptrons cannot solve complex problems: that is where the interest of Multi-Layer Perceptrons (or MLPs) relies. In MLPs, the output of a layer of perceptrons is used as the input of another layer. In that case, the mathematical expression of the output of an MLP is the following:

$$y = f_n(W_n ... f_2(W_2 f_1(W_1 x + b_1) + b_2) + ... + b_n), \tag{29}$$

where W_i, b_i and f_i correspond to the weights, the bias and the activation of layer i, respectively. Biases can be removed for convenience by including it in the weights and assuming that the inputs for the weights corresponding to biases are set to 1. We obtain the following formula:

$$y = f_n(W_n ... f_2(W_2 f_1(W_1 x)) ...). \tag{30}$$

We will stick to that convention in the rest of this chapter.

In 1989, George Cybenko showed that MLPs were able to approximate any real-valued function under some hypotheses [14]: MLPs can potentially solve nearly any complex problem if they are well-formulated. However, even though an efficient MLP can exist for a given complex task, how to find it? How to determine the right parameters? In Section 2.1.2, we explain how to train neural networks, including MLPs.

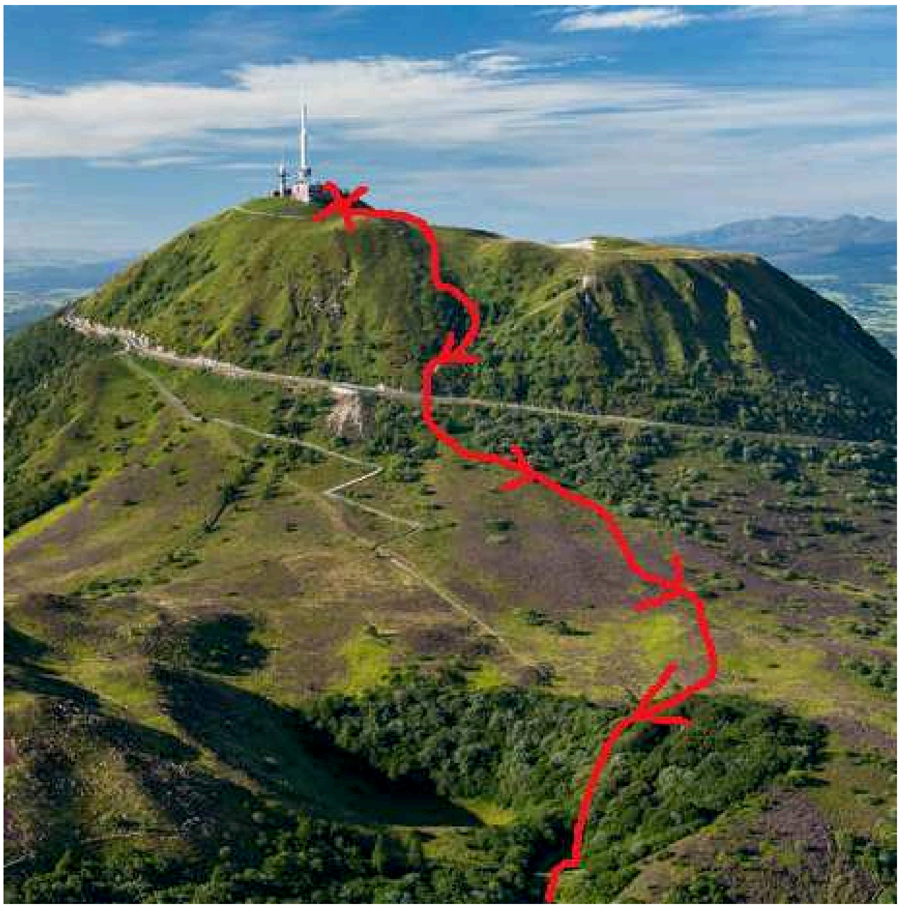


Figure 9: Illustration of the Gradient Descent algorithm. Starting from a random point corresponding to a certain value of the loss function (here the height), computing the gradient at this point gives an information on where the slope is going down.

2.1.2 *The Learning Process*

Training a neural network is not always self-evident, and simple algorithms such as gradient descent are often insufficient to obtain relevant results. In this section, we define the gradient descent algorithm, and explain how it has been improved for efficient neural network training.

2.1.2.1 *Gradient Descent*

We usually try to formulate AI-related problems as unconstrained optimization problems, with a differentiable function as the objective function. The gradient descent algorithm can be used to solve this kind of problems. As this algorithm is based on gradient computations, we must assume that all functions that we use are differentiable almost everywhere, including activation functions of neural networks. First of all, we define a loss function (also called "cost function"), which is an objective function to be minimized. Let us call it

$\mathcal{L}_X$, where X is the input of the neural network. The gradient descent algorithm is an iterative algorithm whose goal is to find optimal parameters, based on an input X. If $W^{(t)}$ are the weights of a neural network at iteration t, weights at iteration $t+1$ are derived based on the following update formula:

$$W^{(t+1)} = W^{(t)} - \lambda \nabla_W \mathcal{L}_X(W^{(t)}), \tag{31}$$

where λ is a hyperparameter called the learning rate.

Training is usually performed on a dataset containing multiple possible inputs. When the loss function is averaged over the set $\mathcal{X}$ all possible samples $X \in \mathcal{X}$, the algorithm is called Batch Gradient Descent, and the update formula is the following:

$$\begin{aligned} W^{(t+1)} &= W^{(t)} - \lambda \nabla_W \mathcal{L}_{\mathcal{X}}(W^{(t)}) \\ &\quad \text{where} \quad \mathcal{L}_{\mathcal{X}}(W^{(t)}) = \tfrac{1}{|\mathcal{X}|} \textstyle\sum_{X \in \mathcal{X}} \mathcal{L}_X(W^{(t)}). \end{aligned} \tag{32}$$

However Batch Gradient Descent is too slow if the training dataset is big: there are too many operations performed before updating the weights of the neural network. Therefore, Stochastic Gradient Descent is preferred: it consists in picking randomly one sample $X^{(t)}$ at each iteration t and updating weights after every random choice. The Stochastic Gradient Descent update formula is the following one:

$$W^{(t+1)} = W^{(t)} - \lambda \nabla_W \mathcal{L}_{X^{(t)}}(W^{(t)}). \tag{33}$$

The drawback of Stochastic Gradient Descent is that it is too noisy, and requires making computations on small pieces of data, which is not efficient for hardware-related reasons. Therefore, the Gradient Descent version that is employed in most cases is the Minibatch Gradient Descent. It is a trade-off between Batch Gradient Descent and Stochastic Gradient Descent: at each iteration t, a small subset $\mathcal{X}^{(t)}$ of $\mathcal{X}$ is sampled randomly, and weights are updated based on that sample of data:

$$W^{(t+1)} = W^{(t)} - \lambda \nabla_W \mathcal{L}_{\mathcal{X}^{(t)}}(W^{(t)}). \tag{34}$$

The convergence speed of Gradient Descent algorithms can further be improved a lot. The three methods that we will introduce are based on the first and second raw moments of weight updates. For simplicity, $\mathcal{L}$ will designate the loss function in the following.

The momentum method [74] consists in adding to the weight update a fraction of previous weight updates: it is based on the first raw moment of weight updates. It can be compared to Newton's principle of inertia: previous updates induce an "inertia" to following updates.

It accelerates convergence if many gradients point at a similar direction. Mathematically, the momentum method can be described as follows:

$$\begin{aligned} W^{(t+1)} &= W^{(t)} - \lambda M^{(t+1)} \\ M^{(t+1)} &= (1-\alpha) \times M^{(t)} + \alpha \times \nabla_W \mathcal{L}(W^{(t)}) \end{aligned} \tag{35}$$

where α is a hyperparameter between 0 and 1. If α is high, then inertia is low, and vice versa.

The RMSProp method [87] is based on the second raw moment of weight updates. It consists in adapting the learning rate for each parameter: when a parameter is often updated, its learning rate is reduced, and vice versa. Mathematically, the learning rate for a given weight is divided by the root mean square of the updates for that weight, as explained in the following formulas:

$$\begin{aligned} W^{(t+1)} &= W^{(t)} - \lambda \frac{\nabla_W \mathcal{L}(W^{(t)})}{\sqrt{G^{(t+1)}}} \\ G^{(t+1)} &= (1-\alpha) \times G^{(t)} + \alpha \times (\nabla_W \mathcal{L}(W^{(t)}))^2 \end{aligned} \tag{36}$$

where α is a hyperparameter controlling the importance of last steps in the moving average of square weight updates.

The Adam method [46] is a combination of Momentum and RMSProp. It can be described by the following formulas:

$$\begin{aligned} W^{(t+1)} &= W^{(t)} - \lambda \frac{\nabla_W \mathcal{L}(M^{(t+1)})}{\sqrt{G^{(t+1)}}} \\ G^{(t+1)} &= (1-\alpha_G) \times G^{(t)} + \alpha_G \times (\nabla_W \mathcal{L}(W^{(t)}))^2 \\ M^{(t+1)} &= (1-\alpha_M) \times M^{(t)} + \alpha_M \times \nabla_W \mathcal{L}(W^{(t)}) \end{aligned} \tag{37}$$

where α_G and α_M are hyperparameters.

Neural networks can contain a lot of layers, and computing weight updates of early layers can be computationally intensive if done naively. In the next section, we explain how weight updates can be computed efficiently, using the backpropagation algorithm.

2.1.2.2 *Backpropagation*

The backpropagation algorithm consists in applying recursively the chain rule to compute weight updates from the last layer of a neural network to the first one. We will describe the backpropagation algorithm for the standard SGD, but it can be used with no modifications for the other optimization methods we introduced previously (Momentum, RMSProp and Adam). First, we define the loss function $\mathcal{L}_X$ for an input X as follows:

$$\mathcal{L}_X(W) = L(f_n(W_n(f_{n-1}W_{n-1}...f_1(W_1X)...))), \tag{38}$$

where $L : \mathbb{R}^d \mapsto \mathbb{R}$ is a differentiable function (d is the dimension of the output of the neural network). For the t-th iteration, the update formula is:

$$W^{(t+1)} = W^{(t)} - \lambda \nabla \mathcal{L}_{X^{(t)}}(W), \tag{39}$$

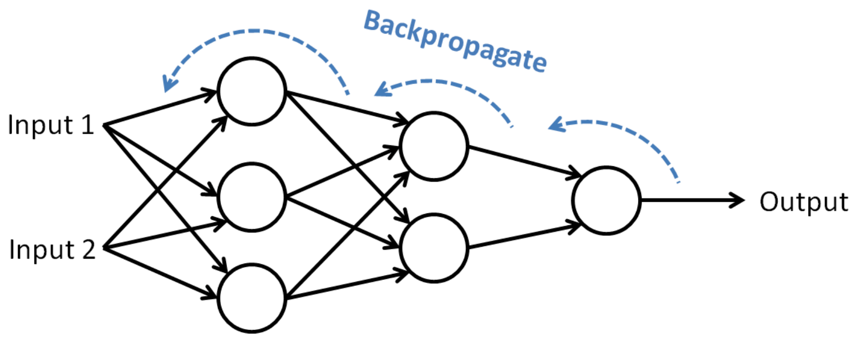


Figure 10: An illustration of the backpropagation algorithm. We give here an example with two inputs, one output and one hidden layer. First, the partial derivatives of the loss with respect to the loss are computed. Then, these partial derivatives are backpropagated to the hidden layer, to compute the partial derivatives of the weights leading to intermediate neurons. Eventually, these weights are also backpropagated to obtain weight updates for the first layer of the neural network.

where $X^{(t)}$ is the input at iteration t.

Let us also define $X_i^{(t)}$ for i in $\{1, ..., n\}$ as follows:

$$X_1^{(t)} = X^{(t)}, \quad X_{i+1}^{(t)} = f_i(W_i X_i^{(t)}). \tag{40}$$

As the backpropagation algorithm computes weight updates recursively, we define a loss function for each layer:

$$\begin{cases} \mathcal{L}_{n,X^{(t)}}(W_n) = L\left(f_n\left(W_n X_n^{(t)}\right)\right) \\ \mathcal{L}_{i,X^{(t)}}(W_i) = \varphi_{i+1}^{(t)}\left(f_i\left(W_i X_i^{(t)}\right)\right) \end{cases} \tag{41}$$

where $\varphi_i^{(t)}$ is defined as follows:

$$\begin{cases} \varphi_n^{(t)}(X_n) = L\left(f_n\left(W_n^{(t)} X_n\right)\right) \\ \varphi_i^{(t)}(X_i) = \varphi_{i+1}^{(t)}\left(f_i\left(W_i^{(t)} X_i\right)\right) \end{cases} \tag{42}$$

where $W_i^{(t)}$ is the weights matrix of layer i at iteration t. We also define the following helper functions:

$$\begin{cases} g_i^{(t)}(X_i) = f_i\left(W_i^{(t)} X_i\right) \\ h_i^{(t)}(W_i) = f_i\left(W_i X_i^{(t)}\right) \end{cases} . \tag{43}$$

Let us now compute the weights updates for each layer:

$$W_i^{(t+1)} = W_i^{(t)} - \lambda \nabla \mathcal{L}_{i,X^{(t)}}\left(W_i^{(t)}\right). \tag{44}$$

The partial loss function $\mathcal{L}_{i,X^{(t)}}$ can be rewritten as follows:

$$\nabla \mathcal{L}_{i,X^{(t)}}\left(W_i^{(t)}\right) = \nabla\left(\varphi_{i+1}^{(t)} \circ h_i^{(t)}\right)\left(W_i^{(t)}\right) \tag{45}$$

This gradient can be computed using the chain rule. Mathematically, if g and f are multivariate real functions and $g \circ f$ is a real-valued function, then the gradient $\nabla(g \circ f)$ is:

$$\nabla(g \circ f)(x) = \nabla g(f(x))^T \mathcal{J}(f)(x) \tag{46}$$

where $\mathcal{J}(f)$ is the Jacobian matrix of f. Using that rule, we can compute the gradient of $\varphi_{i+1}^{(t)} \circ h_i^{(t)}$:

$$\nabla\left(\varphi_{i+1}^{(t)} \circ h_i^{(t)}\right) = \left[\nabla\varphi_{i+1}^{(t)}\left(X_{i+1}^{(t)}\right)\right]^T \left[\mathcal{J}\left(h_i^{(t)}\right)\left(W_i^{(t)}\right)\right]. \tag{47}$$

In the last expression, $\mathcal{J}\left(h_i^{(t)}\right)\left(W_i^{(t)}\right)$ can be easily computed, as $h_i^{(t)}$ and $W_i^{(t)}$ are known. Now, we need to derive an expression for $\nabla\varphi_{i+1}^{(t)}\left(X_{i+1}^{(t)}\right)$. We notice that we have the following relation between $\varphi_{i+1}^{(t)}$ and $\varphi_{i+2}^{(t)}$:

$$\varphi_{i+1}^{(t)}\left(X_{i+1}^{(t)}\right) = \left(\varphi_{i+2}^{(t)} \circ g_{i+1}^{(t)}\right)\left(X_{i+1}^{(t)}\right). \tag{48}$$

This leads us to the final expression of $\nabla\varphi_{i+1}^{(t)}\left(X_{i+1}^{(t)}\right)$ by applying the chain rule:

$$\nabla\varphi_{i+1}^{(t)}\left(X_{i+1}^{(t)}\right) = \left[\nabla\varphi_{i+2}^{(t)}\left(X_{i+2}^{(t)}\right)\right]^T \left[\mathcal{J}\left(g_{i+1}^{(t)}\right)\left(X_{i+1}^{(t)}\right)\right]. \tag{49}$$

The backpropagation algorithm can be used to train any kind of neural networks. In the next sections, we will describe Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs).

2.1.3 *Convolutions for Vision*

Convolutions are operations that have been widely used in signal processing, and in particular in image processing. What are they, and why do they work well with images and videos? In this section, we will define them and relate them to Convolutional Neural Networks.

2.1.3.1 *What are Convolutions?*

Initially, convolutions have been designed as mathematical operations between functions. The convolution of two numerical functions f and K is defined by the following formula:

$$(f * K)(x) = \int_{\mathbb{R}} f(x-t)K(t)dt. \tag{50}$$

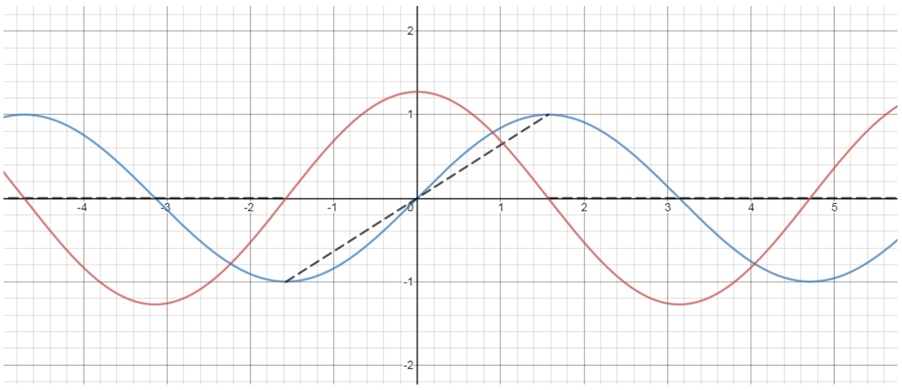


Figure 11: Graphs of the functions f (in blue), $x \mapsto K(-x)$ (in black) and $f * K$ (in red). As one can notice, the convolution $f * K$ corresponds to pattern recognition: the graph of $f * K$ reaches maximums when the graph of f is similar to the graph of $x \mapsto K(-x)$.

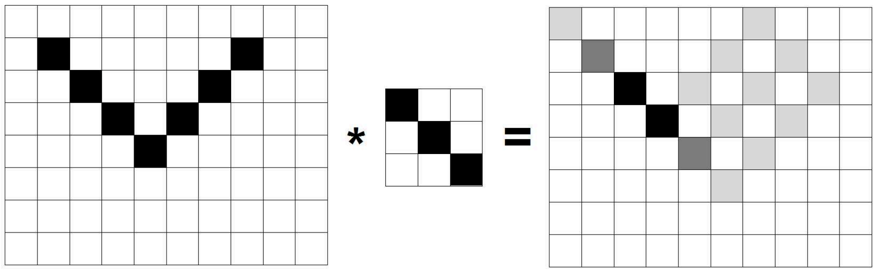


Figure 12: Illustration of a convolution on an image. The pattern to be recognized is the matrix represented at the right of the $*$ sign, the image is at the left and the rightmost matrix is the result of the convolution. In this example, the detected pattern is a type of edge.

That convolution can be intuitively interpreted as a pattern recognition operation. If we imagine that the graph of the function $t \mapsto K(-t)$ is a pattern, then $(f * K)(x)$ is high if that pattern is present around x in the graph of f, and low otherwise. Let us give an example. If $f(x) = \sin(x)$, and $K(x) = -2x/\pi$ for $x \in [-\pi/2, \pi/2]$ and $K(x) = 0$ otherwise, then we find that $(f * K)(x) = \frac{4\cos(x)}{\pi}$. We represented f, $x \mapsto K(-x)$ and $(f * K)$ on Figure 11. As one can notice, $f * K$ reaches maximums when the graph of f is similar to the graph of $x \mapsto K(-x)$, *ie* for $x = 0 \quad [2\pi]$.

Pattern recognition is a way to understand the content of images. Therefore convolutions have intuitively been adapted to image processing, similarly to the convolution operation we defined previously. If I is an image, represented by a matrix, and F is the pattern that we want to recognize in I, the convolution of I and F is defined as:

$$(I * F)(x, y) = \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} I(x - X + i, y - Y + j) F(i, j) \tag{51}$$

where W is the width of the pattern F, H its height, $X = \left\lfloor \frac{W-1}{2} \right\rfloor$ and $Y = \left\lfloor \frac{H-1}{2} \right\rfloor$. A straightforward application of convolutions is edge detection, as represented on Figure 12.

The convolution operation for images that we defined is based on 1-channel images. However, images have usually three channels, as usual images are RGB images. Convolutions have been adapted in 3D fashion, as follows (C is the total number of channels of images):

$$(I * F)(x, y) = \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} \sum_{c=1}^{C} I(x - X + i, y - Y + j, c)\, F(i, j, c). \quad (52)$$

Convolutions as we defined them are not sufficient to detect patterns that are more complex than simple edges. Tackling complexity needs to combine these patterns: this is where the interest of Convolutional Neural Networks lies.

2.1.3.2 *From Handcrafted Features to Learned Features*

In Convolutional Neural Networks (CNNs), convolution filters are not predefined but learned. CNNs are special cases of artificial neural networks, where some weights are set to zero and others are shared: the backpropagation algorithm used to compute weight updates during the gradient descent can be easily extended to train CNNs.

The problem induced by CNNs is that they need a lot of data to grasp all possible patterns that are relevant to understand images. Good results on simple datasets such as MNIST have been reached since 1998 by LeCun *et al.* [52]. However, results of CNNs on "real-world datasets" remained very poor until recently because of the lack of data to train them. At that time, CNNs had to remain shallow and to be used only on handcrafted features to obtain decent results. In 2012, in the context of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [79], a fully-trained CNN using raw images as inputs instead of handcrafted features outperformed by a large margin its competitors [50]. What had changed at that time? The challenge was based on ImageNet [16], a huge dataset of annotated images allowed for the first time to train successfully a deep CNN on raw images. Since then, CNNs are considered as the best models for Computer Vision in most applications.

CNNs are not only composed of learned convolutions detecting patterns. They also contain pooling operations to pool visual information together, as shown on Figure 13. Convolutions in the first layers of a CNN detect low-level patterns such as edges and simple geometric shapes. Recognized patterns become more and more complex in next layers. The last layers of a CNN recognize high-level features, which become more semantic than only geometric.

Even though CNNs lead to state-of-the-art results in most Computer Vision tasks, they are not perfect. In particular, pooling operations raise the problem of the loss of spatial information: when

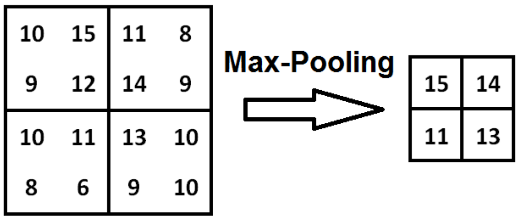


Figure 13: An illustration of pooling in CNNs. In this example, we perform max-pooling: the width and the height of an image is divided by two by pooling together four neighboring pixels.

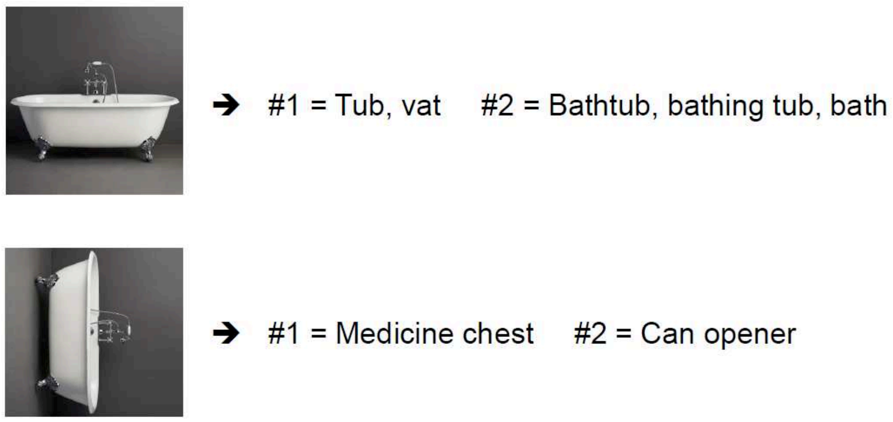

➔ #1 = Tub, vat #2 = Bathtub, bathing tub, bath
➔ #1 = Medicine chest #2 = Can opener

Figure 14: Result of classification using a state-of-the-art ResNet-152 trained on ImageNet. A simple rotation can lead to extremely bad results.

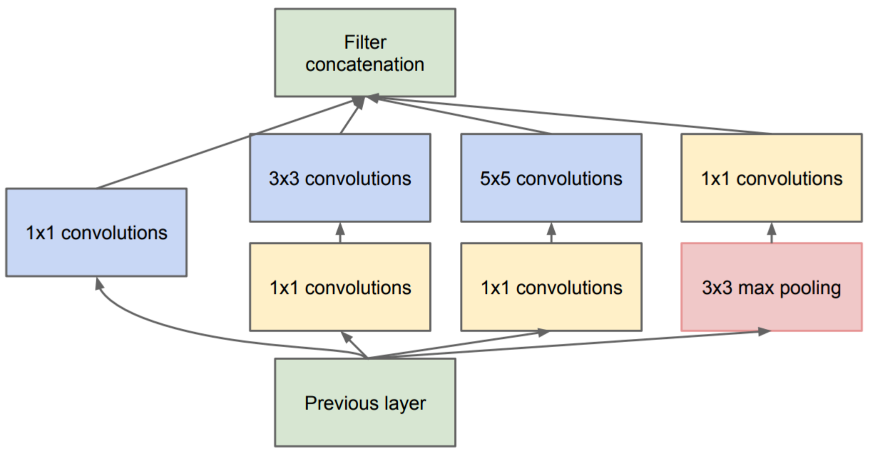


Figure 15: Illustration of an Inception unit. Inception units contain 1×1 convolutions, which increase the depth of the CNN without increasing a lot its number of trainable weights

outputs of convolutional layers are pooled together, they do not contain anymore information on how pooled features were located in the image with respect to the others. On Figure 14, we give an example of how a state-of-the-art ResNet-152 [35] image classifier can be deceived by a simple rotation. Some works have tried to solve that problem. In particular, Sabour *et al.* and Hinton *et al.* proposed Capsule Networks that get rid of these pooling layers. In Capsule Networks, neurons are replaced by capsules, that are actually small neural networks outputting an activation as neurons do, but also a pose vector containing spatial information. Even though that kind of architecture has shown promising results, it has not been applied yet on large-scale problems, because of the computationally intensive routing algorithms that they need to transfer outputs of one layer of capsules to the following one.

In the next section, we will give a description of some of the CNN architectures that we use in our research work.

2.1.3.3 *CNN Architectures*

In 2014 and 2015, the ILSVRC saw new architectures of CNNs improving the results that had been obtained in 2012. Let us describe these architectures, that we have been using in our research work.

In 2014, the first runner-up team came from the Visual Geometry Group (VGG) of the University of Oxford. The model they propose for the challenge was the VGG16 network, and consisted in a stack of 16 convolutional layers, with pooling layers inserted at some points. The problem that raises this network is that it is very long to train, because of its 138 millions of weights. The winner of the 2014 ILSVRC has proposed an architecture tackling that issue.

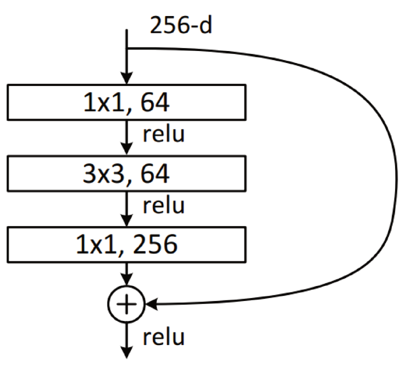


Figure 16: Illustration of a Residual Block. If X is the input of a Residual Block, and $f(X)$ is the result of the convolutions included in this Residual Block, its final output is $X + f(X)$. Residual blocks allow a better backpropagation of errors during gradient descent training.

The winner of the 2014 ILSVRC was a team from Google. They proposed a CNN based on what they called Inception units, that are represented on Figure 15. These inception units contain 1×1 convolutions, which increase the depth of the CNN without increasing a lot its number of trainable weights. This allows to reduce overfitting and also to increase the depth, and so the ability to recognize more complex pattern in images. Their Inception CNN contains two convolutional layers and nine inception layers. Each inception layer actually contains two layers of convolutions: their Inception CNN is composed of 20 layers of convolutions in total, but has only 5 millions of parameters.

In 2015, the winner proposed a CNN based on what they called "Residual Blocks". One of these residual blocks have been represented on Figure 16. They consist in adding a "residue" after a certain number of convolutions. More formally, if X is the input of a Residual Block, and $f(X)$ is the result of the convolutions included in this Residual Block, its final output is $X + f(X)$. A problem raised by very deep architectures is that during gradient descent training, derivatives of the loss function with respect to the weights of first layers are not well backpropagated. Adding a residue at some points of the neural network allows a better backpropagation of errors. The architecture they proposed to win the 2015 ILSVRC was indeed composed of 152 layers, using some 1×1 convolutions to reduce the number of parameters.

It has been shown that features derived by CNNs before the classification layer can be transferred for other applications [102]. In our work, we indeed made use of these CNNs to extract features by taking the output of their penultimate layer, before classification.

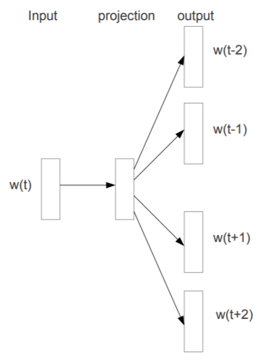


Figure 17: The skip-gram model from [65]. A vector representation is derived from a word. That representation is optimized so that it can predict surrounding words.

2.1.4 *Recurrences for Sentences*

Dealing with texts in natural language raises different issues than Computer Vision. The first one is that words are not easily represented by numbers. Applying convolutions to images to recognize patterns is intuitive, as we stated in Section 2.1.3.1, because they can be represented by matrices or tensors of numbers. What can be done with words? The second issue is that texts in natural language are sequences of words of undefined length, with complex dependences between them. How to deal with these sequences?

In this section, we will introduce word embeddings that deal with the first issue, and Recurrent Neural Networks that deal with the second one.

2.1.4.1 *Word Embeddings*

The simplest vector representation that we could think of for words is the one-hot encoding: each word is assigned an index, and is represented by a one-hot encoding corresponding to that index. However, even though that representation is simple, it reflects neither syntactic nor semantic information. Some better vector representations of words, called word embeddings, have been proposed.

Word2vec is one of them. It has been proposed by Mikolov *et al.* in [65]. It consists in using a skip-gram model that predicts surrounding words from an input word. More precisely, given the one-hot encoding of a word, an affine transform reduces the dimension of that encoding to obtain an embedding, which is used through other affine transforms to predict surrounding words, as shown on Figure 17. Another word embedding is GloVe [72]. It derives word embeddings based on global and local statistics of a text corpus. In this work, we also used PDC-HDC word embeddings [84], that are learned by jointly modeling syntagmatic and paradigmatic relations between words.

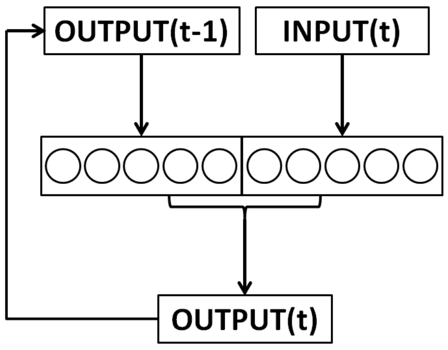


Figure 18: Illustration of a simple RNN. The output at a given iteration depends on the input, but also in the previous output.

Recently, word embeddings such as BERT [18] are based on a new Transformer architecture [88], but we did not use them in this thesis work.

How to process whole sentences? Making averages of word embeddings could be a simple solution, but it does not take into account relations between words in a sentence. This is where the interest of Recurrent Neural Networks (RNNs) lies.

2.1.4.2 *RNNs*

Sentences can be seen as sequences of words. We saw how to make word vector representations, but we need to combine them in a clever way. For that purpose, the use of Recurrent Neural Networks has shown good results. Recurrent Neural Networks are neural networks whose output depends not only on its current input but also on previous ones. A simple RNN with input $x^{(t+1)}$ and output $h^{(t+1)}$ can be described through the following equation:

$$h^{(t+1)} = f\left(W_h h^{(t)} + W_x^{(t+1)} x^{(t+1)} + b\right) \tag{53}$$

where W_h, W_x and b are trainable parameters. That architecture is represented on Figure 18. The output of an RNN is also called a hidden state, because it is used to compute its next output: RNNs can be seen as state machines whose inputs induce transitions to other states. This simple RNN however does not give good results on natural language processing tasks: why? We will see that some tricks are necessary to obtain desired results.

2.1.4.3 *LSTMs and GRUs*

The problem that is raised by the simple RNN architecture that we introduced in Section 2.1.4.2 is the vanishing or exploding gradient problem: with long sequences of words, during gradient descent training, it happens that very often, updates become very big or very small. In that case, the neural network does not train. Why is that happening? Let us understand this problem with a toy example. We assume that we are considering the architecture we defined in Section 2.1.4.2, with inputs and outputs of dimension 1, $W_h = 0$ and $x^{(t)} = 1$ for all t. The partial derivative of the output with respect to W_x is then the following:

$$\frac{\partial h^{(t+1)}}{\partial W_x} = \prod_{i=1}^{t} f'(W_x + b) = f'(W_x + b)^t \tag{54}$$

where f' is the derivative of f. We can see that the gradient can be very big if $|f'(W_x + b)| > 1$ and very small if $|f'(W_x + b)| < 1$. How to deal with the vanishing or exploding gradient problem?

In [38], the Long Short-Term Memory (LSTM) network has been proposed. A memory cell is introduced to keep long-term dependencies between words, and is also architectured to avoid the gradient problems we described previously. An illustration of an LSTM can be seen on Figure 19. An LSTM can be described by the following equations:

$$\begin{aligned}
i^{(t)} &= \sigma\left(W_{ih}h^{(t-1)} + W_{ix}x^{(t)} + b_i\right)\\
o^{(t)} &= \sigma\left(W_{oh}h^{(t-1)} + W_{ox}x^{(t)} + b_o\right)\\
f^{(t)} &= \sigma\left(W_{fh}h^{(t-1)} + W_{fx}x^{(t)} + b_f\right)\\
\overline{h}^{(t)} &= \tanh\left(W_{hh}h^{(t-1)} + W_{ix}x^{(t)} + b_i\right)\\
c^{(t)} &= i^{(t)} \odot \overline{h}^{(t)} + f^{(t)} \odot c^{(t)}\\
h^{(t)} &= o^{(t)} \odot c^{(t)}
\end{aligned} \tag{55}$$

where W_{ih}, W_{ix}, W_{oh}, W_{ox}, W_{fh}, W_{fx}, b_i, b_o and b_f are trainable parameters.

Let us see with the same toy example as in 2.1.4.2 if the gradient problem is still raised. We assume that $W_{hh} = 0$, and all inputs are equal to 1:

$$\frac{\partial h^{(t+1)}}{\partial W_{hx}} = o^{(t)}\left(i^{(t)} \tanh\left(W_{hx} + b\right) + f^{(t)}\frac{\partial c^{(t+1)}}{\partial w_{hx}}\right). \tag{56}$$

As one can see, the vanishing or exploding gradient problem does not appear anymore with that toy example.

Another type of Recurrent Neural Network called the Gated Recurrent Unit (GRU) has been proposed in [13]. We illustrated it on Figure 20. It usually leads to similar results, but contains less trainable parameters because the output gate has been removed. It can be described through the following equations:

$$\begin{aligned}
i^{(t)} &= \sigma\left(W_{ih}h^{(t-1)} + W_{ix}x^{(t)} + b_i\right)\\
f^{(t)} &= \sigma\left(W_{fh}h^{(t-1)} + W_{fx}x^{(t)} + b_f\right)\\
\overline{h}^{(t)} &= \tanh\left(W_{hh}\left(h^{(t-1)} \odot f^{(t)}\right) + W_{ix}x^{(t)} + b_i\right)\\
h^{(t)} &= i^{(t)} \odot \overline{h}^{(t)} + \left(1 - i^{(t)}\right) \odot h^{(t-1)}
\end{aligned} \tag{57}$$

where W_{ih}, W_{ix}, W_{fh}, W_{fx}, b_h and b_x are trainable parameters.

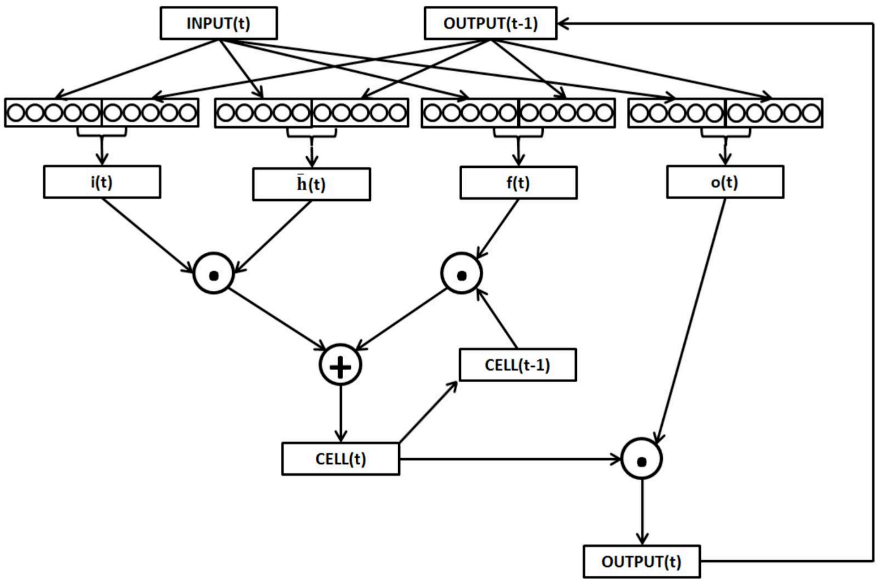


Figure 19: Illustration of an LSTM.

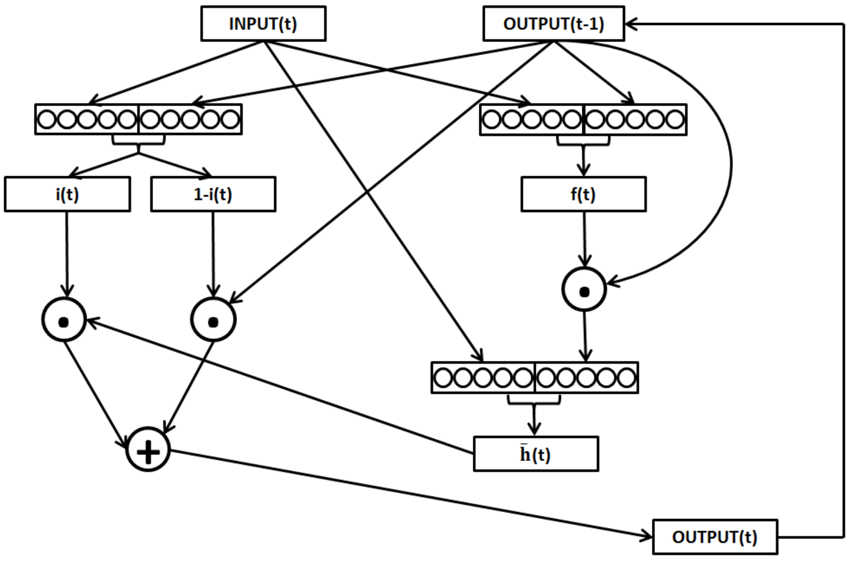


Figure 20: Illustration of a GRU.

2.1.4.4 *Non-Recurrent Neural Networks for NLP*

Some non-recurrent neural networks have been proposed for NLP. For instance, CNNs for sentence classification have been proposed in [45], and a Transformer model for neural machine translation has been presented in [88]. Even though these models give sometimes better results than LSTMs or GRUs on NLP tasks, we did not have the opportunity to experiment with them in this thesis work.

2.2 TIES BETWEEN VISION AND LANGUAGE

In Section 2.1, we explained how Deep Learning techniques could allow to deal with complex problems of Computer Vision and Natural Language Processing. However, in this thesis work, our goal is to derive semantic representations of images and videos. Therefore, we have to relate vision and language. What kind of semantic representations can we derive for images and videos? How to combine the advantages of CNNs and RNNs to deal with that problem?

In this section, we will give an overview of existing works dealing jointly with vision and language. The semantic representations that we will introduce are first vector representations, and then textual descriptions in natural language.

2.2.1 *Training Models to Match Vision and Language*

The vision-language matching task consists in designing models telling whether a given text describes accurately a given image or a given video. It requires being able to compare visual objects with sentences. For that purpose, a usual way to make that task feasible is to represent each modality with a vector belonging to a same space. Using a similarity metric allows then to compare them. Let S be a similarity metric such that $S(u, v)$ is high when vectors u and v are close, and low otherwise. The first intuition would be to try to maximize S, and so use $-S$ as a loss function. When the model is only trained to construct one of the two vectors, that loss function works well. However such a loss function is not relevant if both vectors are constructed by our model: an obvious minimum for that loss function is to project all inputs to the same vector. In that case, adding contrastive examples in the training loss function is essential. In most works in vision-language matching such as [44], a triplet loss is used. If s is a sentence and v is an image or a video, $\phi_W(s)$ the vector derived by our model for s and $\psi_W(v)$ the vector for v (W is the set of all trainable weights), the triplet loss is defined as:

$$\mathcal{L}(W) = \max(0, \alpha - S(\phi_W(s), \psi_W(v)) + S(\phi_W(s), \psi_W(\bar{v}))) \quad (58)$$

where $\bar{v}$ is a contrastive example for v and α is a hyperparameter. Please note that instead of taking a contrastive example for v, a contrastive example for s could also have been chosen. The first version actually optimizes a model for image or video retrieval whereas the second would optimize a model for sentence retrieval. Both losses can also be summed if both tasks need to be achieved.

Several similarity metrics can be used. The most employed ones are based on the cosine similarity, and on the mean square error. The cosine similarity is defined as:

$$\cos(u, v) = \frac{\langle u|v\rangle}{\|u\|\|v\|}. \tag{59}$$

The cosine similarity is already a similarity metric, so in that case, we have $S(u, v) = \cos(u, v)$. The mean square error (or MSE) is defined as:

$$\mathrm{MSE}(u, v) = \|u - v\|_2^2 \tag{60}$$

and in that case, we have $S(u, v) = -\mathrm{MSE}(u, v)$. Some other similarity metrics have been proposed such as the order similarity relying on the intrinsic asymmetry existing between vision and text as a descriptive text is always less precise than the image or the video it is describing [90]. However we did not experiment on these other similarity metrics in our thesis work.

When should we use the cosine similarity and when should we use the mean square error? The cosine similarity is interesting because of its normalizing terms: if the vectors of both modalities are derived by our model, the MSE loss leads to an unwanted situation where vector norms are minimized, because the lower the norm the lower the MSE. However, when the vectors of only one modality are built by our model, the MSE should be preferred as it keeps the geometry of the other modality.

2.2.2 *Dealing with Images...*

Several works have been done on building visual-semantic embeddings. In [73], Frome *et al.* made such embeddings through a model based on a skip-gram text modeling architecture [65] for the semantic part, and on AlexNet [50] for the visual part. Karpathy *et al.* proposed fragment embeddings in [44]: as the different parts of a descriptive sentence correspond to different parts of an image, they designed a model that would draw matches between image fragments and text fragments. The model proposed by Kiros *et al.* in [47] is similar to the models we will present in Section 3.3 and Section 3.4: they derive a sentence embedding with an LSTM and a features vector for a corresponding image through a CNN. Their objective function imposes a constrain on the two modalities so that they belong to the same space. The main difference between our models and [47] is that we make use of novel RNN architectures that we designed. More information is given in Section 3.3 and Section 3.4. Variations of Kiros' model have been suggested. Vendrov *et al.* have tried to induce an order between images and sentences by replacing the cosine similarity in the loss function by an order penalty [90]. More recently, Faghri *et al.* have shown that emphasizing hard negatives in the loss function would lead to high improvements [25].

Some other works are based on Fisher Vectors [73]. In [48], Klein *et al.* derive Fisher Vectors for sentences from a Gaussian Mixture Model (GMM) and a Hybrid Laplacian-Gaussian Mixture Model (HLGMM). In [54], Lev *et al.* propose an RNN that is used as a generative probabilistic model. In these last two works, matches between images and texts are made through the Canonical Correlation Analysis algorithm [40]. In [24], Eisenschtat and Wolf also derive Fisher Vectors for sentences from a GMM and an HLGMM, but they do not use CCA to match images and sentences. They designed a 2-Way neural network that projects the two modalities onto a common space.

More complex architectures have also been designed in some papers. Niu *et al.* [68] have proposed a hierarchical multimodal LSTM: a sentence is parsed as a tree and correspondences between phrases and image regions are drawn. Nam *et al.* [67] proposed a Dual Attention Network that is designed to attend jointly to image regions and corresponding words in text. Gu *et al.* [31] proposed a model that would match texts and images by learning high-level global features, but also low-level local features thanks to two generative models: one of them generates sentences from images and the other one generates images from sentences. More recently, the use of local features derived through an object detection model has shown good results in image-sentence matching [53].

2.2.3 *... and Videos*

There is less existing research work on video-text matching than in image-text matching. The principle of works in video-text matching are often similar to those we find in image-text matching. For instance, the Word2VisualVec model [20] has been designed for image but has been very simply extended to video. In [66], authors propose to build two multimodal spaces to match videos and texts: a first one is based on a CNN deriving features for video frames (they call it the object space) whereas a second one is based on an I3D model [11], which is a 4D-convolutional neural network trained to detect activities in videos (they call that second space the activity space). Some works have also been conducted in video search, where multiple videos need to be retrieved for one given natural language query; in these works, best results are usually obtained using concept detectors [23, 61]. Some other works have also investigated how natural language queries could be used to detect moments in videos. These works are based on the DiDeMo dataset [4], but we considered it was out of the

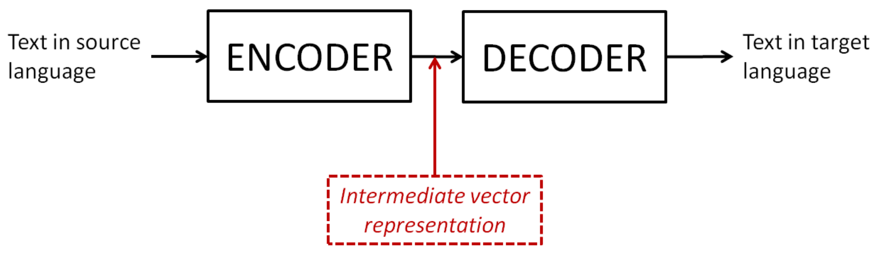


Figure 21: The encoder-decoder scheme. A text in a source language is input to an encoder (usually an LSTM or a GRU), which derives a vector representation of that text. That representation is then used by the decoder (also an LSTM or a GRU) to generate a sentence in a target language.

scope of this thesis work: therefore we did not conduct research work on it.

2.3 TELL ME WHAT YOU SEE

In the previous section, we explained how images and videos could be compared to texts. This can be useful for indexing and retrieval tasks. In this section, we make a review of works in captioning: how to generate descriptive sentences from images and videos?

2.3.1 *Captioning as a Neural Machine Translation Task*

Captioning can be seen as a translation task: an image or a sequence of frames, which can be compared to a sequence of words in a source language, have to be translated in a target language. Some pioneering works in image captioning such as [26] and in video captioning such as [77] make use of Statistical Machine Translation techniques to generate captions from images or videos, respectively. However nowadays, most of recent works on image or video captioning rely on Deep Learning techniques, and more particularly on the encoder-decoder framework that has been developed in [85] for text translation [19], and that we represented on Figure 21. Moreover, attending to the hidden states of the encoder during the decoding phase has shown to give significant improvements in Neural Machine Translation [60], which have been confirmed by [99] and [101] in the context of image captioning and video captioning, respectively.

Let us focus a little on attention mechanisms. The attention mechanisms that are usually employed in captioning tasks work as follows: the t-th generated word is used as a reference to help the decoder to generate the following $t+1$-th word. More formally, if x is the word embedding of the last generated word, and $v_1, ..., v_n$ are n visual features vectors (corresponding to image regions or video regions), then the features vector that will be input to the decoder to help it generate the next word is:

$$v = \sum_{i=1}^{n} \alpha_i v_i, \tag{61}$$

where the α_i are non-negative weights. These weights are derived through the following computations:

$$(\alpha_1, ..., \alpha_n) = \text{softmax}(r_1, ..., r_n), \tag{62}$$

where r_i for $i \in \{1, ..., n\}$ is defined as:

$$r_i = f(W_v v_i + W_x x + b), \tag{63}$$

where f is an activation function. In the last equation, W_v, W_x and b are trainable parameters that are derived training by gradient descent. The weighted sum of features vectors that is thus obtained emphasizes important regions for word prediction, and obliterates non-relevant ones. Let us now describe how captioning models are trained.

2.3.2 *Training Captioning Models*

In this section, we will explain how to train a captioning model, and how to evaluate it.

2.3.2.1 *Loss Functions*

The goal of a captioning model is to output a descriptive sentence given an image or video input. That task is usually performed using a cross-entropy function. Mathematically, given an image or video V, and a ground-truth sentence $s = (x_1, ..., x_L)$ of length L, the cross-entropy loss is defined as:

$$\mathcal{L}(W) = -\sum_{l=1}^{L} \log\left(p_W\left(x_l | x_1, ..., x_{l-1}, V\right)\right) \tag{64}$$

where W is the set of trainable weights of our captioning model. The problem of the cross-entropy loss is that there is a discrepancy with the metric that is used for evaluation. For that reason, some works such as [76] introduced a Reinforcement Learning loss aiming at directly optimizing the captioning model with respect to its evaluation metric. More formally, let $r(s)$ be the score assigned to a descriptive sentence s for a given input image or video ($r(s)$ is high when the sentence is good and low otherwise). Then, the new loss function is:

$$\mathcal{L}(W) = -r(s), \quad s \sim p_W. \tag{65}$$

That loss function is not differentiable, therefore it is not usable directly for gradient descent. However the expectation of L can be written as:

$$\begin{aligned}\nabla E\left[\mathcal{L}(W)\right] &= \nabla E\left[-r(s)\right] \\ &= -\textstyle\sum_s r(s)\nabla p_W(s) \\ &= -\textstyle\sum_s r(s)p_W(s)\nabla \log(p_W(s)) \\ &= -E\left[r(s)\nabla \log(p_W(s))\right] \\ &= \nabla E\left[-r(s)\log(p_W(s))\right]\end{aligned} \tag{66}$$

Therefore, it boils down to optimizing on the loss function $\mathcal{L}'$, defined as:

$$\begin{aligned}\mathcal{L}'(W) &= -r(s)\log(p_W(s)), \quad s \sim p_W \\ &= -r(s)\textstyle\sum_{l=1}^{L}\log\left(p_W\left(x_l|x_1,\dots,x_{l-1},V\right)\right).\end{aligned} \tag{67}$$

Let us now describe the evaluation metrics that are used for evaluating captioning models.

2.3.2.2 *Evaluation Metrics*

Several metrics have been designed to evaluate translation tasks or captioning models. As we explained before, the captioning task can be seen as a translation task: evaluation metrics for translation tasks are actually widely used to evaluate captioning. Let us describe the ones we used in this thesis work.

The BLEU-n metric has been defined in [70]. It counts the proportion of n-grams in the reference sentences that appear in the candidate sentence, which is called the n-gram precision. If an n-gram is not more than p times in any reference sentence, we count it not more than p times in the candidate, even if there are more occurrences of that n-gram in the candidate sentence. Let us give an example:
Candidate sentence: "a cat is lying on a blue couch"
Reference sentence 1: "there is a black cat on a couch"
Reference sentence 2: "the cat is lying on the couch"
BLEU-1 = 0.875 ($\underline{\text{a}}$ $\underline{\text{cat}}$ $\underline{\text{is}}$ $\underline{\text{lying}}$ $\underline{\text{on}}$ $\underline{\text{a}}$ blue $\underline{\text{couch}}$)
BLEU-2 = 0.429 (a cat, $\underline{\text{cat is}}$, $\underline{\text{is lying}}$, $\underline{\text{lying on}}$, $\underline{\text{on a}}$, a blue, blue couch).

The ROUGE$_L$ [57] metric is based on the longest common subsequence (LCS) between the candidate sentence and a reference sentence. It is computed as follows:

$$\text{ROUGE}_L(c,r) = \frac{(1+\beta^2)RP}{R+\beta^2 P}, \tag{68}$$

where c is the candidate sentence, r a reference, $P = \frac{\text{LCS}(c,r)}{\text{length}(c)}$, $R = \frac{\text{LCS}(c,r)}{\text{length}(r)}$ and $\beta = \frac{P}{R}$. In the previous example, the ROUGE$_L$ score of the candidate sentence with respect to the first reference sentence is 0.625, its ROUGE_L score with respect to the second candidate sentence is 0.661. Therefore, its global ROUGE_L score is 0.643.

The METEOR [17] and CIDEr_D [89] metrics are more computationally intensive: therefore we will not do the computations here by hand but only give the idea behind.

The METEOR metric is computed by first aligning candidate and reference sentences based on synonyms, stemming and phrase correspondences. Once the alignment done, a similarity score is computed between both sentences.

The CIDEr_D metric is derived as an average of cosine similarities of TF-IDF vectors of references and candidate based on 1-gram, 2-gram, 3-gram and 4-gram tokenizations.

For convenience, the ROUGE_L and the CIDEr_D metrics will be designated as ROUGE and CIDEr in the rest of this work.

2.3.3 *Image Captioning*

First works in image captioning were based on the simple Encoder-Decoder scheme we described, with no attention mechanism. NeuralTalk by Karpathy *et al.* [43] and Show and Tell by Vinyals *et al.* [92] fall into this category. After the image captioning model proposed by [99], using an attention mechanism to improve that Encoder-Decoder scheme has quickly become a standard in image captioning. From that Encoder-Decoder with attention baseline, different models for image captioning have been proposed, often taking into account local features [32] or object detections [3] instead of only global features.

2.3.4 *Video Captioning*

In some works, videos are split into frames, global features are derived for each frame using a CNN [35, 50, 81, 86], and the obtained features vectors are sequentially processed by the encoder [33, 55, 59, 69, 93, 96]. The drawback in such approaches is that spatial information is lost. In the approach we proposed in [27], we aim at taking into account this spatial information.

Other approaches take into account locality. However, these approaches have some significant differences with our approach. In [101], the authors separate their model into two parts: a usual encoder-decoder based on global features of frames, and a 3D-CNN that derives a single representation for a whole video. The 3D-CNN they employ does take into account locality, but it has two major conceptual differences with respect to our work in [27]. First, it is based on handcrafted features, which do not provide as much semantic information as CNN features. Moreover, the pooling operations that are used to get their video representations are neither learned nor adaptive. In our approach, pooling takes into account the relevance of local features in a frame with respect to previous frames. In [103], authors use local features to trace semantic concepts along videos, which is conceptually different from our approach, as we aim to derive a video representation based on these local features. In [97], authors propose another method to compute trajectories through videos. In both papers, these trajectories are combined with global features to build video representations. In [94], local features are used to generate video representations. However, local features from different spatial locations are not related together, contrary to the work we conducted in 4.2 and that we will introduce in 4.2, which proposes to attend to local features based on all local features from previous frames. Eventually, some other works used 3D-CNN architectures [95] or convolutional RNNs [94] to relate local features through time. However, due to the nature of convolution operations, relations drawn through these methods remain local: they are not able to spatially relate objects from the video which are far from each other in a video for instance.

2.3.4.1 *Dense Captioning*

Some works we did not cite yet focused on what is called Image Dense Captioning and Video Dense Captioning. The goal of the Image Dense Captioning task is to extracts all regions of interest of an image and to annotate them with natural language sentences. One of the pioneering works in Image Dense Captioning is DenseCap by Johnson *et al.* [42]. Video Dense Captioning is similar to Image Dense Captioning, but instead of annotating regions of interest, models aim at annotating moments of interest [107]. We did not have the opportunity to experiment with these kinds of models, even though they could have fallen in the category of image captioning or video captioning.

3 MATCHING VISION AND LANGUAGE

In this chapter, the subject of study is the vision-language matching task: how to tell whether texts describe accurately images or videos?

3.1 INTRODUCTION

Humans can intuitively match sensorial perceptions such as vision with language. For instance, one can easily describe a landscape, or imagine how one looks like after reading its description in a novel. For a computer, that seemingly mundane task raises some questions: how to represent sentences and images and how to link these representations of two different modalities? Some successful works have been done in that field, such as automatic image captioning [43] or visual question answering [2].

How has that been possible? As we explained in Section 2.1.3.2, big annotated databases of multimedia contents such as ImageNet [16] were built recently, and these collections permitted the emergence of deep neural networks, because they can perform very well if they are trained with a sufficient amount of training examples. In computer vision, convolutional neural networks (CNN) [35, 50] are now widely used. In natural language processing, recurrent neural networks (RNN) such as long short-term memory units (LSTM) [38] or gated recurrent units (GRU) [13] are a common choice. As mentioned in Chapter 2, these neural networks can learn representations of images and texts, and some works have shown that these representations can be used as a basis for comparisons [25, 31, 47, 67, 68, 90].

In this chapter, we will present our results in image-sentence matching and in video-sentence matching.

3.2 VISUAL VS TEXTUAL EMBEDDINGS

Our work for this thesis has started in 2016, in the context of the TRECVid Ad-hoc Video Search (AVS) campaign. But before introducing AVS, let us have some words about video retrieval. Managing large databases of multimedia content such as videos is more and more a topical issue. The problem is that in such databases, video content cannot be manually annotated. Therefore, searching videos in big video databases is especially a matter of using self-contained information. In addition, the lack of structure in such databases implies that queries must be built as freely as possible; preferably in natural language. Pattern Recognition techniques seem to be nowadays one of the most adapted to such problems: they offer great performance in both natural language and computer vision.

Natural Language Processing and Computer Vision problems can be addressed with Deep Learning techniques. In Natural Language Processing, Recurrent Neural Networks (RNN) [64] are now widely used to model languages: sentences are divided into word vectors [65, 72] that are processed one after the other by such networks. Convolutional Neural Network are well-adapted to Computer Vision tasks. After being trained they can detect visual concepts with very high precision [35, 50, 81]. Recent works have shown that these networks could be combined to match visual content with text content. For instance neural networks such as NeuralTalk [43] or DenseCap [42] can produce sentences to describe visual content and some combinations of techniques make it possible to perform zero-shot video search with simple text descriptions [15, 34].

The National Institute of Standards and Technology (NIST) organizes every year TRECVID [6], an international evaluation campaign on video information retrieval. In 2016, a new task called "Ad-Hoc Video Search" (AVS) was proposed. The goal was to process Natural Language queries to retrieve relevant shots from a large database containing about 600 hours of video, representing 300,000 shots. Participating teams were provided with 30 test topics, and could submit up to four runs. Each run had to be a list of 1,000 shots, ranked from the most relevant shot to the least one. The NIST performed a manual evaluation of the runs and gave the Mean Inferred Average Precision (an approximation of the Mean Average Precision) for each of them.

Our work for this thesis started at the end of the AVS TRECVid campaign of 2016. EURECOM took part in that AVS task and even though participants were allowed to submit only four runs many possible systems have been implemented. The data that were used to evaluate runs were published after the campaign, therefore we could evaluate all these systems. In this section, we present and analyze the performances of all the approaches EURECOM implemented for the AVS task. These approaches were built upon two orthogonal strategies:

- Strategy 1: use the natural language queries to take images from a web search engine, and compare keyframes with these images to select the best ones.

- Strategy 2: generate a text-based description of the keyframes and compare them to the queries.

These strategies have been implemented using tools that are freely available from the Internet.

- We got images from the Google ImageSearch engine to implement Strategy 1: we entered a text query and the search engine returned a list of images related to it. The implementation of this search engine is not open-source, but we think that it is likely to be based on the textual content that surrounds images.

- To get a text description from an image, we used several tools:
  - the VGG Deep Networks [81], which have been trained on part of the ImageNet [16] database and can analyze an image to provide scores for 1,000 predefined concepts,
  - the ImageNet Shuffle [63], which provides classifiers trained on a larger share of the ImageNet database, and analyze images to produce scores for up to 13,000 concepts,
  - the NeuralTalk [43] model, which generates sentences describing the visual content of images.
- To compare visual contents, we compute a visual feature vector for an image by applying the VGG Deep Network to each image and extracting the outputs of the one-before-last and two-before-last layers, to build visual vectors. The similarity between visual vectors is computed as the usual scalar product, sometimes with normalization.
- To compare textual content, we use the GloVe vector representations of words [72], to build a textual vector from either the topic description, the concept name or the descriptive sentence. The similarity between textual vectors is again computed as the usual scalar product.

We implemented several types of runs inasmuch as many combinations of these modules are possible, as well as different values of the parameters involved. These runs boil down to three types: runs based on Strategy 1, runs based on Strategy 2 and runs mixing both strategies. We noticed that performances where not the same depending on topics and on some other parameters. We will elaborate on what worked and what did not.

3.2.1 *EURECOM Runs at TRECVid AVS 2016*

In this section, we will describe how EURECOM runs were built.

3.2.1.1 *Generic Architecture*

The Figure 22 illustrates the generic architecture that we have put in place, corresponding modules. The green modules represent text-based information, the blue modules contain visual information, the yellow modules represent similarity computations. We tried various combinations to define the four runs that we submitted to the final evaluation.

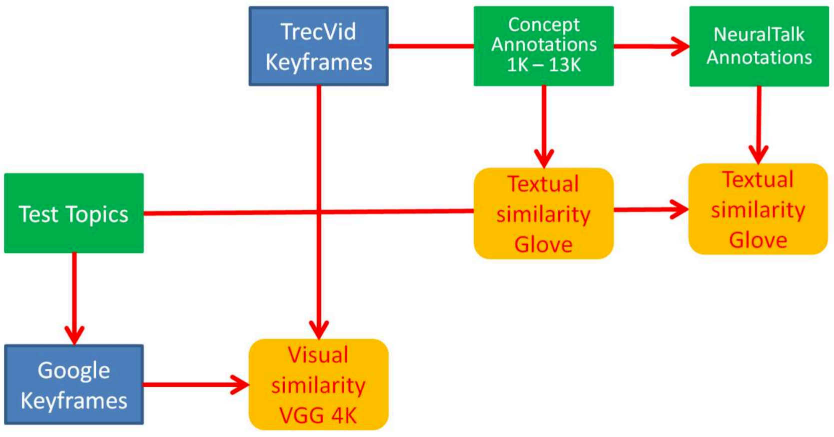


Figure 22: EURECOM Runs at TRECVid AVS 2016: Generic Architecture. The two strategies we mentioned are represented in this figure: either topics are input to Google Images to retrieve corresponding images that are compared to keyframes in a visual space, or keyframes are converted into textual annotations that are compared to topic in a textual space.

All our runs are of the "Fully Automatic" category, since no manual processing was done at any stage, and with the "D" training type, as we are using tools which were trained on data external to TRECVID.

3.2.1.2 *Runs Using the First Strategy*

For each of the topic, we performed a search using the Google Images engine, and retained the first 100 pictures of the ranked list. To each image, we applied the VGG Deep network, and kept either the last, the penultimate or the antepenultimate layer as feature vector of dimension 4K. Thus we obtained vectors for each of the 100 pictures. We tried to normalize them using L2-normalization and not to normalize them. We applied the same visual processing to each of the TRECVID keyframes in the test collection, and ranked them according to a Nearest Neighbor distance from Google images.

3.2.1.3 *Runs Using the Second Strategy*

We implemented two types of systems based on the second approach. The first one uses 13,000 ImageShuffle concepts. The second one is based on NeuralTalk. In both case we make a comparison between vectors. We tried to normalize them using L2-normalization and not to normalize them.
With ImageNet Shuffle Concepts – We used the ImageShuffle system to obtain scores for 13,000 concepts, which we used as feature vectors for each TRECVID keyframe. We used these scores as weights to compute a semantic vector of dimension 50 (resp. 100, 200 or 300) by a linear combination of the GloVe vectors corresponding to the

concepts. For each topic, we constructed a semantic vector of dimension 50 (resp. 100, 200 or 300) by averaging the GloVe vectors of the words appearing in the topic. Then we used the cosine similarity to find the images whose semantic vectors were most similar to the topics.

With NeuralTalk – We used the NeuralTalk system to generate text descriptions for each of the TRECVID keyframes. Then, we built a semantic vector of dimension 50 (resp. 100, 200 or 300) by averaging the GloVe vectors of dimension 50 (resp. 100, 200 or 300) of the words appearing in these descriptions. We did the same for the test topics. Finally, we used again the cosine similarity to find the images whose semantic vectors were most similar to the topics.

3.2.1.4 *Runs Combining both Approaches*

During the development phase, we experimented with a number of combinations of the modules that we have described, using different dimensions, different projections, different layers, different similarity measures. We tried several combinations of our previous approaches and computed a score for each image by averaging its inverse ranks in all results lists.

3.2.2 *Experimental Results*

We evaluated all our models. Their results are summed up in Table 1.

Type	**Best MAP**	**Average MAP**
Strategy 1	0.0173	0.0098
Strategy 2 (ImageNet Shuffle)	0.0285	0.0219
Strategy 2 (NeuralTalk)	0.0021	0.0016
Mix of both strategies	0.0167	0.0113

Table 1: Results of Our Models on AVS 2016 Data

The best strategy seems to be the second one, with ImageShuffle concepts. But we also found out that the efficiency of our strategies depended on the topic. The graph that is represented on Figure 23 is a PCA of the different runs: we built a vector for each run, whose coordinates are the AP corresponding to the different topics.

As one can notice on Figure 23, the two strategies seem to have orthogonal behaviors, and the mix of both strategies seems to be a "middle ground". Therefore, we argue that trying to find the best strategy for a given topic instead of mixing strategies would be worthwhile. We will elaborate on that later on.

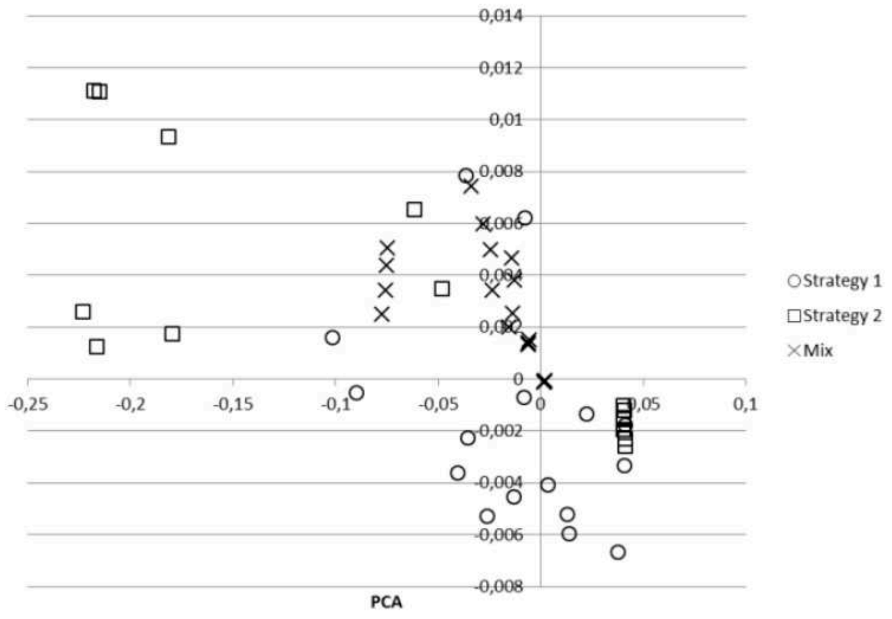


Figure 23: Behavior of our strategies (X-Axis: min = -0.223, max = 0.041, σ = 0.065; Y-Axis: min = -0.007, max = 0.011, σ = 0.003)

3.2.3 *Discussion*

Let us now discuss the effect of different settings, and how one strategy performs with respect to the other.

3.2.3.1 *Effect of* L_2*-Normalisation*

As said in the introduction, all our runs were based on vectors, and we wondered if it was worth normalizing them. Therefore we made tests with raw vectors and with L2-normalized vectors. In Figure 24, each point corresponds to a model. Its abscissa corresponds to its Mean Average Precision (MAP) without L_2-normalization and its ordinate corresponds to its MAP with L_2-normalization. The red line is the line of equation $y = x$.

As one can notice, L2-normalization often improves our results, and never deteriorates them.

3.2.3.2 *Dimension of GloVe Vectors*

Our models based on Strategy 2 need a word embedding. We used GloVe vectors, and tried different dimensions (50, 100, 200 and 300). In Figure 25 we give the MAP for the best five models based on Strategy 2, according to the dimension of the GloVe vectors we chose.

As expected the higher the dimension is, the higher the MAP.

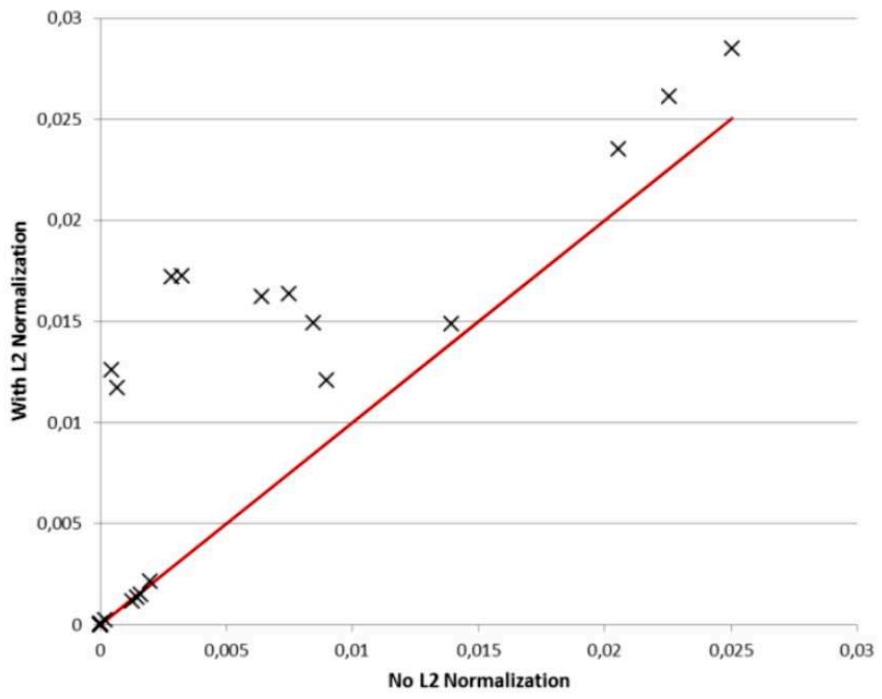


Figure 24: Influence of L_2-normalization

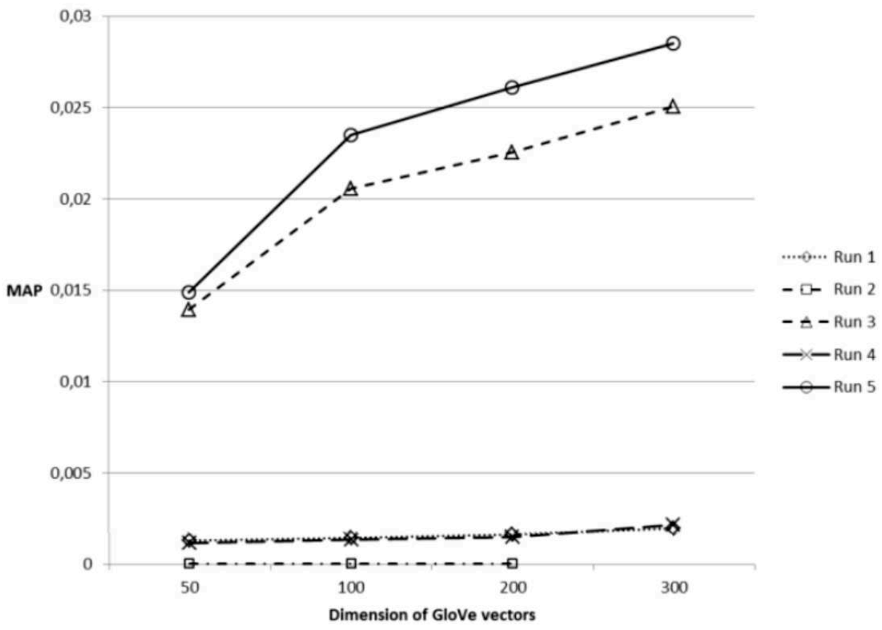


Figure 25: Influence of GloVe vectors dimensions

3.2.3.3 *How to Choose the Right Strategy?*

As we said in Section 3.2.2, Strategy 1 and Strategy 2 are orthogonal in that they do not have good results on the same topics. It would be an interesting challenge to find a way to decide whether we should use the first or the second strategy. More precisely our goal is to choose the best combination of two models from the ones we implemented and use a binary classifier to use the most adapted to a given topic. As expected, we found out that the best combination for the 30 topics of TRECVID was composed of one model based on Strategy 1 and one model based on Strategy 2:

- the first one uses the penultimate layer of the VGG Deep Network, with L2-normalization, to compute the vectors for Google images;
- the second one uses GloVe vectors of dimension 300 to represent ImageShuffle concepts (with L2 normalization).

The results we obtained are reported in Table 2.

Model	**MAP**
Best Model among All	0.0285
Best Combination of Two Models	0.0371

Table 2: Results of the Best Model and the Best Combination of Two Models

As one can notice the MAP increases by 30% if we choose the best combination of two models, and then apply the most relevant one for a given topic: it would be a significant improvement.

We tried to check if such a system was feasible. For each group of 29 topics, we found the best combination of two models. Then for each topic, we averaged the GloVe vectors of their words, thus getting an overall vector. Next we trained a linear SVM to classify topics according to which model is the most adapted. We eventually applied the SVM on the remaining topic.

Unfortunately we did not get good results, as the MAP of the resulting system was only 0.0084. We think that there are two main reasons explaining these bad results:

- as we only had 30 topics it was difficult to build a general model that would generalize to new topics;
- averaging GloVe vectors to obtain topic vectors may not be adapted, as one can see on Figure 26, representing a PCA of the topic vectors.

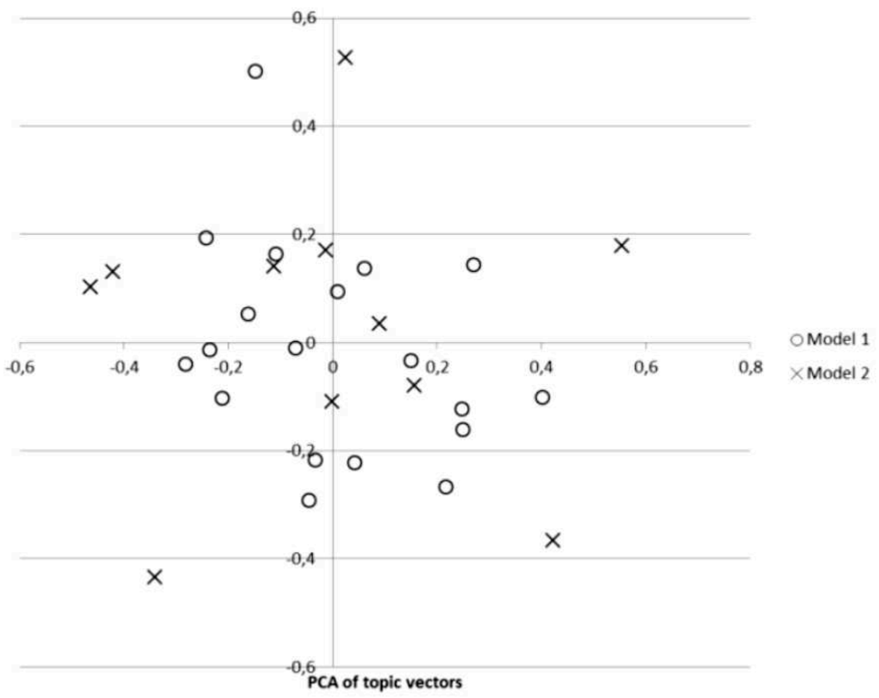


Figure 26: Topics cannot be easily separated into two groups (X-Axis: min = -0.465, max = 0.555, σ = 0.244; Y-Axis: min = -0.434, max = 0.526, σ = 0.217)

3.2.3.4 *Boosting the Second Strategy with Visual Weights*

After our participation in TRECVid AVS in 2016, we noticed that simply averaging word embeddings to make a sentence embedding was less efficient when sentences contained visually ambiguous words such as articles, pronouns or general concepts. As a result of that observation we decided to derive weights associated to their visual explicitness. For that purpose we used the MSCOCO database, containing 40k images with five sentence labels each. We computed 1k ImageNet scores for each image using a VGG Deep Network, thus obtaining 1k-dimensional vectors. If I is an image, let V_I denote its corresponding vector of ImageNet scores. Let w be a word and let S_w be the set of all image-sentence couples (I, s) in MSCOCO with s containing w. Eventually let $\overline{V}$ be the average vector of all images in MSCOCO. Then we derived word scores according to the following formula:

$$\text{score}(w) = \left\| \overline{V} - \frac{1}{|S_w|} \sum_{(I,s) \in S_w} V_I \right\|_2 . \tag{69}$$

We found that these scores worked like a visual tf-idf weighting: they were high both when words were uncommon and when they were explicitly designating a visual element of an image, as shown

Word	**Visual weight**
a	0.12
while	0.53
make	0.61
plenty	1.03
stuff	1.08
cat	1.22
piano	1.23
banana	1.29
clock	1.31

Table 3: Examples of visual weights for some words. It appears that words corresponding to something that can be visualized have higher weights than the others.

Model	Mean Average Precision
Without weights	0.039
With weights	0.047

Table 4: Results of our new implementation of our second strategy on AVS 2016 data. Visual weights lead to better results.

on Table 3. We wondered if making a weighted average of words embeddings based on these visual weights could lead to better results.

For that purpose, we replicated our second strategy based on 1k ImageNet concepts. To improve our results, we used a better concept extractor, released by [61] after AVS 2016, and PDC word embeddings [84] giving better results than GloVe word embeddings. We evaluated that new implementation of the second strategy with and without visual weights. As shown in the results we report in Table 4, visual weights seemed to improve results significantly.

Therefore, we decided to participate in AVS 2017 using these visual weights. Unfortunately, as shown on Table 5, the improvement we saw on AVS 2016 data did not appear on AVS 2017 data.

3.2.4 *Conclusion of Section 3.2*

We evaluated different models based on two main strategies, aiming at doing ad-hoc video search. The result of any of these models is a vector space that we use to compare queries and keyframes. We studied the importance of two factors: the suitability of L2-normalization and the dimension of word embeddings if they are needed. We also showed that our two strategies were orthogonal: they do not give good results on the same topics. We proposed visual weights, that

Model	Mean Average Precision
Without weights	0.094
With weights	0.090

Table 5: Results of our new implementation of our second strategy on AVS 2017 data. Surprisingly, visual weights lead to better results.

lead to a significant improvement on AVS 2016 data, but not on AVS 2017 data.

Would it be possible to improve results by embeddings sentences and images or videos in a same multimodal space? State-of-the-art works seem to show that it is the case. This is the reason why we decided to focus on multimodal embeddings, as we did in the next section.

3.3 RECURRENT CAPSULE NETWORKS FOR MULTIMODAL EMBEDDINGS

In the previous section, we discussed methods for comparing visual objects and text using visual or textual embeddings. Another way of comparing them is by representing them in a common multimodal space, with multimodal embeddings. This can be done by defining a constraint on the neural networks that are used to represent the two different modalities, so that they learn the same representation for them: in that case they would output a vector representation of images or texts so that an image and a corresponding text are close according to some similarity measure. This is what the model we introduce in this part is doing: we use a first neural network to compute an image embedding, a second one to compute a sentence embedding, and we train them jointly to make images and corresponding sentences match together.

At the end of 2017, a novel architecture of neural networks called "capsule" [80] has shown promising results on some computer vision tasks. A capsule is a group of neurons whose role is to make some complex computations, and to output a simple vector as a result of these computations. These outputs are then routed towards higher-level capsules. During training they are supposed to specialize in the recognition of specific patterns; lower capsules learn to recognize simple shapes and higher capsules use the results of the computation of the lower capsules to recognize complex shapes. The advantage of capsules with respect to more common CNNs is that they do not contain pooling operations losing spatial information in high-level layers. We think that the idea of capsule-like units can be generalized to other fields than computer vision. The model we introduce here uses capsule units to analyze sentences – we expect each capsule to perform well on a certain type of sentences. In our model, outputs of capsules

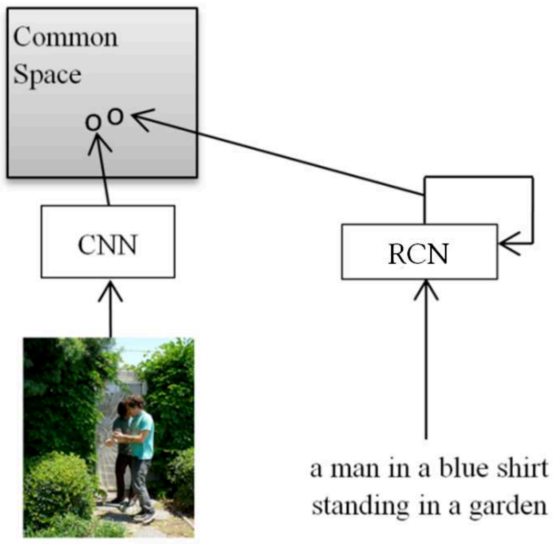


Figure 27: Overview of our RCN model: A first part computes image embeddings, a second one computes sentence embeddings through a Recurrent CapsNet. The two parts are constrained by a loss function to produce embeddings in the same multimodal space.

are not routed towards higher level capsules but towards capsules of the same layer in a recurrent fashion: Figure 27 depicts our proposed recurrent capsule network (Recurrent CapsNet or RCN) architecture. We will elaborate on this new deep model in subsequent parts of this section.

In this section, we present the model for visual sentence embeddings that we described in [28]. It is divided into two parts: the first part computes sentence embeddings thanks to an RCN – each capsule contains two GRUs as we will explain later on. The second part computes image embeddings based on feature vectors obtained with a ResNet-152 [35] (ResNet-152 often provides state-of-the-art results in computer vision tasks). We compare our model with some state of the art models used for image captioning and retrieval on the Flickr8k dataset [39] and show that our RCN performs better than most approaches and better than a simple GRU. Our contributions are the definition of the RCN, and the presentation of its results on an Image Annotation task and on an Image Retrieval task when it is used to produce sentence embeddings.

The rest of the presentation is organized as follows: in Section 3.3.1 we present some recent and related works. In Section 3.3.2 we give a formal definition of our model. In Section 3.3.3 we present the results of experiments we made on our model. We conclude our presentation in Section 3.3.4.

3.3.1 *Related Work*

The model we propose here is inspired by a kind of neural network architecture called "capsule" [36, 80]. A capsule is a group of neurons that is supposed to do some complex and very specific computations, and then output a low dimensional vector as a result of these computations. In computer vision, this architecture is worthy of interest because it could overcome the problem of pooling operations in CNNs that lose all spatial information in higher layers. As far as we know, capsules have only been proposed in computer vision. Our model is the first one using capsules in natural language processing. Our architecture is also the first recurrent one: CapsNet have only been proposed in a feed-forward fashion.

3.3.2 *A Recurrent CapsNet for Visual Sentence Embedding*

Let us now describe our novel Recurrent CapsNet (RCN) model in detail.

3.3.2.1 *Gated Recurrent Units (GRU)*

As stated in Section 2.1.4.3, Gated Recurrent Units were introduced by Cho *et al.* in [13]. They are similar to LSTMs: they have similar performances and are well adapted to NLP because they can handle long-term dependencies in sentences. We preferred GRUs to LSTMs because they have less parameters for similar performances. More formally, a GRU is composed of an update gate i_s and a reset gate f_s, and can be described with the following expressions:

$$\begin{aligned} i_s &= \sigma\left(W_{ih} h_{s-1} + W_{ix} x_s + b_i\right) \\ f_s &= \sigma\left(W_{fh} h_{s-1} + W_{fx} x_s + b_f\right) \\ \overline{h}_s &= \tanh\left(W_{hh}\left(h_{s-1} \odot f_s\right) + W_{ix} x_s + b_i\right) \\ h_s &= i_s \odot \overline{h}_s + \left(1 - i_s\right) \odot h_s \end{aligned} \tag{70}$$

with x_s the s-th input and h_s the s-th output of the GRU. Here and throughout the presentation, $\odot$ denotes the Hadamard product and σ denotes the sigmoid function. For more informations on the interest of GRUs, please refer to Section 2.1.4.3.

3.3.2.2 *Our Model*

Our model can be divided into two parts: a first part aims at deriving sentence embeddings based on an RCN and the second part is designed to project an image features vector onto the same space as the aforementioned sentence embeddings. Let us first describe the first part of our model.

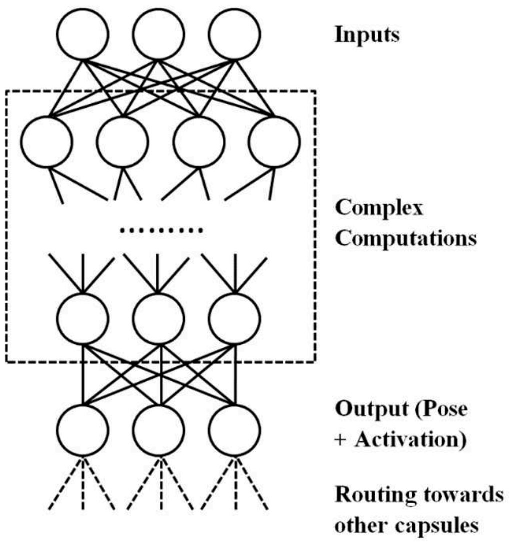


Figure 28: A generic capsule for computer vision

The idea behind capsules as they were designed by [80] for image processing is represented on Figure 28. It consists in making complex computations and outputting a pose vector and an activation. This output is then routed towards subsequent capsules according to some predefined routing algorithm. The goal of that architecture is to have each capsule learning to recognize a visual feature based on what previous capsules have recognized before. For instance, some capsules could recognize eyes, a nose, a mouth and their respective positions. Then they would send their outputs to another capsule aiming at recognizing a whole face. It is architectured to avoid losing spatial information as common CNN do due to pooling operations. We think that capsules can also successfully perform other tasks such as NLP-related tasks, as our proposed model does.

In our case, each capsule contains two GRUs. The role of the first GRU is to output what we call a "mask" (the equivalent of the activation for computer vision) and the second one outputs a sentence embedding (the equivalent of the pose). The output of the first GRU of the i-th capsule is denoted by $\mathrm{GRU}_{\mathrm{mask}_i}(\mathrm{input})$ and the output of the second GRU of the i-th capsule is denoted by $\mathrm{GRU}_{\mathrm{emb}_i}(\mathrm{input})$. In the first GRU, the tanh function is replaced by a sigmoid function so that masks are composed of positive numbers. The mask that is output plays the role of an attention mechanism; we will give more details in the following. The biggest difference with capsules as they were described in [80] is that they are applied in a recurrent fashion: masks that are produced at time step t are applied to the input sentence, which is then fed into the same capsules at time step $t+1$. Sentence embeddings are built according to the following steps:

- we represent each word of a given sentence by a one-hot vector;

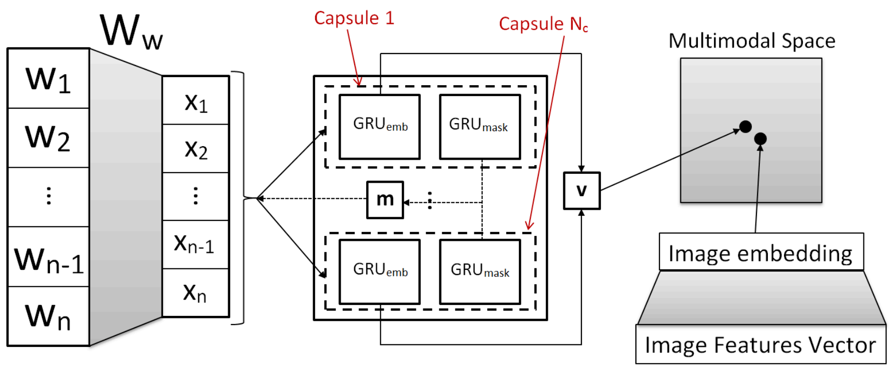


Figure 29: Our model. Capsules are represented with dashed boxes. In the sentence embedding part, a sentence is represented by a sequence of one-hot vectors $(w_1, ..., w_n)$. It is transformed into a list of word embeddings $(x_1, ..., x_n)$ through a multiplication by a word embedding matrix W_w. The sentence then goes through the Recurrent CapsNet a pre-defined number of times, and eventually the RCN outputs a sentence embedding v. In the image embedding part, an affine transformation is applied to a features vector to obtain an image embedding. Both embeddings belong to the same multimodal space.

- we multiply each one-hot vector by a word embedding matrix;
- at each time step we apply masks onto the initial input sentence, and produce new masks based on the new input;
- we eventually output a sentence embedding.

Let us describe more formally how we compute that sentence embedding. Let s be a sentence. We encode each word of s with a one-hot vector: we have $s = (w_1, ..., w_L)$ with L the length of s, and $w_1, ..., w_L$ belonging to $\mathbb{R}^D$ with D the size of our vocabulary. Let $W_w \in \mathbb{R}^{D \times V}$ be the word embedding matrix. Then $x = W_w s$ will denote $(x_1, ..., x_L) = (W_w w_1, ..., W_w w_L)$ in the following. If m is a V-dimensional vector, then $m \odot x$ will denote $(m \odot x_1, ..., m \odot x_L)$. In what follows m denotes a mask and v denotes an embedding. Embeddings are computed according to the following:

$$v_i^{(t)} = \mathrm{GRU}_{\mathrm{emb}_i}(m_i^{(t-1)} \odot x). \tag{71}$$

Masks are computed in two steps. First, capsules compute a mask according to the input sentence and the masks that were computed at the previous step:

$$\tilde{m}_i^{(t)} = \mathrm{GRU}_{\mathrm{mask}_i}(m_i^{(t-1)} \odot x). \tag{72}$$

Then, the m_i are computed as linear combinations of these masks, as follows:

$$m_i^{(t)} = \sum_{j=1}^{N_c} \alpha_{ij}^{(t)} \tilde{m}_i^{(t)} \tag{73}$$

and the final embeddings are computed in a similar way:

$$v^{(t)} = \sum_{i=1}^{N_c} \sum_{j=1}^{N_c} \beta_{ij}^{(t)} \tilde{v}_i^{(t)}. \tag{74}$$

The coefficients of the linear combinations are computed according to the following formulas ($\langle v_1 | v_2 \rangle$ denotes the scalar product between two vectors v_1 and v_2):

$$\alpha_{ij}^{(t)} = \frac{\left\langle v_i^{(t)} | v_j^{(t)} \right\rangle}{\sum_{k=1}^{N_c} \left\langle v_i^{(t)} | v_k^{(t)} \right\rangle}, \tag{75}$$

$$\beta_{ij}^{(t)} = \frac{\left\langle v_i^{(t)} | v_j^{(t)} \right\rangle}{\sum_{k=1}^{N_c} \sum_{l=1}^{N_c} \left\langle v_k^{(t)} | v_l^{(t)} \right\rangle}. \tag{76}$$

Please note that for $t = 0$, the masks are vectors whose coordinates are all equal to one: we actually just input sentences in the GRUs without applying any masks to them. These formulas can be interpreted in an intuitive way: if there are many capsules whose embeddings are similar to its own embedding, then they contribute a lot in the computation of masks and embeddings. If the embedding of a capsule is very different from other embeddings, its participation remains marginal. It can be viewed as a variation of the routing-by-agreement that is proposed in [80]: when other capsules output a sentence embedding that is very similar to the output of a given capsule, then this particular capsule plays an important role in the computation of the final embedding. If other capsules "do not agree", then it becomes more marginal. Regarding masks that are assigned to capsules, one can notice that a capsule contributes to the derivation of the mask of another capsule only if both capsules output a similar embedding. In that sense, embeddings that are output by capsules can be related to the pose vectors and masks can be viewed as activations as they were described in [80]: capsules send their activations to capsules agreeing to their pose. However, masks are not simple activations: multiplying term-by-term input words vectors by masks composed of positive numbers is more an attention mechanism that attends to a particular domain of the word embedding space.

The second part of our model is much simpler. First, we compute features vectors of images by keeping the output of the penultimate layer of a ResNet-152 that has been pre-trained on ImageNet 1000 classes and finetuned on MSCOCO [25]. More precisely we make nine crops of an image in the same way as it has been done in [90], and we compute the average of the corresponding features vectors of these crops. Then, we just apply an affine transform to these vectors. The parameters of that affine transform are derived during training. Figure 29 gives a summary of how our model has been defined.

3.3.2.3 *Loss Function*

The loss function that we apply is the hard-negative-based triplet loss that has been presented in [25]. If N_b is the number of samples in the mini-batch then we draw N_b image-sentence pairs in the training dataset. Then, for each pair of the mini-batch we take the contrastive image and the contrastive sentence for which our model is less efficient, and apply a penalty for these contrastive image and sentence. More formally the loss function is defined as follows (ε is a hyperparameter):

$$L_1 = \frac{\sum_{k=1}^{N_b} \max_{\overline{u_l} \neq u_k} \max(-\varepsilon, -\cos(u_k, v_k) + \cos(\overline{u_l}, v_k))}{N_b} \tag{77}$$

$$L_2 = \frac{\sum_{k=1}^{N_b} \max_{\overline{v_l} \neq v_k} \max(-\varepsilon, -\cos(u_k, v_k) + \cos(u_k, \overline{v_l}))}{N_b} \tag{78}$$

$$L = L_1 + L_2 \tag{79}$$

Optimizing the model boils down to finding parameters that minimize L. More information on triplet losses can be found in Section 2.2.1.

3.3.2.4 *Regularization*

Since we want to avoid having the same masks in each capsule, and we do not want a capsule to output only zero-masks, we added the following regularization term to the loss function

$$V = \left(\frac{\sum_{i=1}^{N_c} \|m_i - \overline{m}\|_2^2}{N_c \|\overline{m}\|_2^2} \min_i \|m_i\|_2 \right)^{-1} \tag{80}$$

where $\overline{m}$ denotes the average mask. The new loss function is then

$$L' = L + \lambda V, \tag{81}$$

with λ a hyperparameter. As we will see in the next section, that regularization term can lead to better results.

3.3.3 *Results and Discussion*

We will now present the experimental results of our model and discuss on them.

3.3.3.1 *Parameters and implementation*

We evaluated how our models performed on both the image annotation and the image retrieval tasks on the Flickr8k dataset [39]. This dataset comes with a predefined split between training, validation and testing samples; we use that split in our experiments. This dataset is composed of 8000 images with 5 sentences each: there are 6000 images in the training set, 1000 images in the validation set and 1000 images in the testing set.

Regarding the sentence embedding part of our model, we set its parameters as follows: we set the maximum sentence length to 16 (if longer the sentence is cut after the 16-th word), $D = 5000$ (we kept only the 5000 most common words and replaced all the other words by an "UNK" token) and $V = 300$. W_w was initialized according to precomputed Word2Vec embeddings [65]. Regarding the RCN, we found that a model with four capsules performed well. For a given sentence we kept $v^{(5)}$ as its corresponding embedding. For the image embedding part of our model, the dimension of image features vectors was 2048. The final embeddings dimension was 1024.

In the loss function, we set $\epsilon = 0.2$ and we tried different values for λ. We found that $\lambda = 0.05$ was giving good results. We trained our models using the Adam method [46] with mini-batches of 16 image-sentence pairs. The learning rate was 0.0002. We made all our implementations using the TensorFlow [1] library for Python.

3.3.3.2 *Experiments*

We compared our results on Flickr8k with some recent state-of-the-art models. In addition, we trained two different models on the Flickr8k dataset.

GRU. That model is simply the one we described in Section 3.3.2, but with a simple GRU instead of the RCN. It is also a special case of our model where the number of capsules is 1 and the number of recursions is 0.

RCN-λ. This is the model we described in Section 3.3.2. As mentioned in Section 3.3.3.1, we found that four capsules and four recursions lead to better results. λ is the same as in Section 3.3.2.

Table 6: Results of our experiments on Flickr8k. R@K denotes Recall at rank K (higher is better). Best results among other models and among our models are in bold. Best results among all models are underlined

Flickr8k						
Model	Image Annotation			Image Retrieval		
	R@1	**R@5**	**R@10**	**R@1**	**R@5**	**R@10**
Random	0.1	0.6	1.1	0.1	0.5	1.0
[48]	31.0	59.2	73.7	21.2	50.0	64.8
[54]	31.6	61.2	**74.3**	23.2	**53.3**	67.8
[68]	27	-	68.6	24.4	-	**68.1**
[24]	**43.4**	**63.2**	-	**29.3**	49.7	-
GRU	37.8	67.9	79.3	29.7	59.5	71.5
RCN-0	38.8	67.0	78.9	**30.3**	**60.4**	**72.5**
RCN-0.05	**41.5**	**70.6**	**81.2**	29.9	59.9	72.4

As one can notice looking at Table 6, results with our RCN are at state-of-the-art level. On top of that, capsules improve results of a single GRU, especially in the Image Annotation task. The regularization term seems to improve results for the Image Annotation task without having notable effects on the Image Retrieval task.

3.3.3.3 *Effect of the size of a simple GRU*

In this section, we investigated if simply increasing the number of hidden states in a GRU could lead to the same improvements as our RCN. For that purpose, we computed the sum of the recalls at 1, 5 and 10 for the Image Annotation task and the Image Retrieval task for our best model (RCN-0.05) and for GRUs with 64 to 2200 hidden states.

As one can notice on Figure 30, there is no improvement if the size of the hidden states GRU is increased above 800.

3.3.3.4 *Effect of the regularization term*

One can notice that the regularization term improves results with respect to simple GRU or non-regularized RCN. Why is that happening? Our hypothesis is that the regularization term imposes that capsules attend to different domains of the word embedding space. Therefore, sentence embeddings of different capsules tend to be more

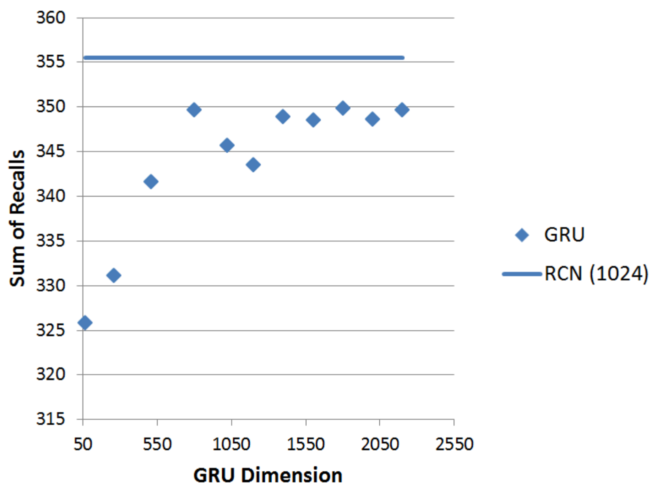


Figure 30: Performance of GRUs on Flickr8k according to the size of their hidden states. The x-axis represents the size of hidden states, the y-axis represents the performance of the GRUs. This performance is the sum of R@1, R@5 and R@10 for both Image Annotation and Image Retrieval tasks. The line corresponds to the performance of our best model (RCN-0.05 with hidden states of dimension 1024).

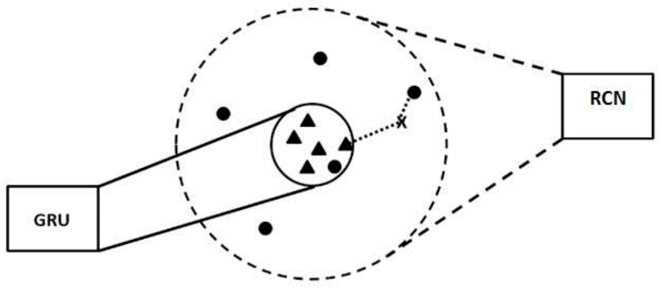


Figure 31: Our hypothesis. The cross corresponds to an image, circles correspond to sentence embeddings output by an RCN and triangles correspond to sentence embeddings output by a GRU. In that case, both the RCN and the GRU output sentence embeddings for a given image around the same point, but even if the RCN generates worse embeddings on average, its best sentence embedding is better than the best sentence embedding output by the GRU.

different than for a GRU or an RCN without normalization. As the Image Annotation task consists in retrieving one of five sentences for an image, the distribution that capsules induce may lead to have one out of these five sentences being closer to its corresponding image. We summarized our hypothesis in Figure 31.

3.3.4 *Conclusion of Section 3.3*

In [28], we proposed the RCN (Recurrent Capsule Network), a novel deep architecture for visual sentence embedding. It is based on the

CapsNet architecture that was recently proposed in [80], but it differs from it in three important ways: we applied it to natural language processing, it is built in a recurrent fashion whereas the original CapsNet was built in a fully-connected fashion and the routing is performed using one of the GRUs.

We obtained some promising results, especially for the Image Annotation task where our RCN performs better than GRUs. We explained these performances improvements by the distribution that the capsules induced in the computation of sentence embeddings. In addition, we showed that the results of our capsules could not only be explained by the fact that they had more parameters than their corresponding GRUs: Figure 30 showed that increasing the size of GRUs hidden states did not result in as good results as our models.

The main drawback of our model is that it is very large, and very long to train on bigger datasets such as MSCOCO [58]. Therefore, we architectured Gated Recurrent Capsules, requiring much less trainable parameters than the RCN we introduced in this section. The next section is dedicated to these Recurrent Gated Capsules.

3.4 GATED RECURRENT CAPSULES

In [29], we proposed another novel deep network architecture for the caption retrieval task: given a set of images and a set of sentences, we build a model that ought to find the closest sentence to an input image. As stated in Section 2.2.2, numerous works have attempted to address that task; most of them are making use of a multimodal space where sentences and images are projected and compared [25, 31, 44, 53]. Word2VisualVec [20, 21] relies on another approach, the authors built a model to project sentences directly in a space of visual features: as the quality of visual features is constantly improving, the authors stated that learning visual sentence embeddings rather than projecting them in a more complicated multimodal space was a promising approach. In our paper [29], a model following this unconventional approach is proposed.

Projecting images and sentences in the same space, whether multimodal or simply visual, implies that representations of images and sentences as mathematical objects must be derived. Since the recent breakthrough of deep learning, Convolutional Neural Networks (CNNs) have shown compellingly good performances in computer vision tasks. In particular, some of them [35, 50] are able to learn visual features that they use to classify images from a big dataset such as ImageNet [16]. Most recent works on caption retrieval have used features coming from a ResNet [35] which had been trained on ImageNet for a classification task [25, 31]. In our work, we extract features thanks to a ResNet that had been finetuned on MSCOCO [58] by the authors of [25], as we did for the RCN in Section 3.3. Deriving visual sentence representations is the main part of our work. Recurrent Neural Networks (RNNs) such as Long Short-Term Memory units (LSTMs) [38] and Gated Recurrent Units (GRUs) [13] have proved to deliver state-of-the-art results on various language modeling tasks such as translation [85], automatic image captioning [92] or caption retrieval [25]. In the last version of Word2VisualVec [21], the authors showed that concatenating a representation derived by a GRU with a Word2Vec [65] representation and a bag-of-words representation to get a multi-scale sentence representation lead to better results in visual sentence embedding for caption retrieval. However, we argue that using these kinds of representations cannot be optimal: pooling all words together without putting attention on relevant parts of the sentence does not reflect the complexity of images; and the current state-of-the-art model for image and caption retrieval is based on object-detection and cross-attention [53], which corroborates our statement that sentences should be processed in a finer way. Our work aims at proposing a new architecture corresponding to and addressing that issue: how to analyze a sentence so that important visual elements are emphasized?

As for the RCN, our research has been inspired by recent works on capsule networks [37, 80]. This new architecture shows promising results in computer vision. In capsule networks, the routing procedure can be seen as an attention mechanism that routes the output of capsules from one layer to the relevant ones in the next layer. We think that this principle can be successfully used in Recurrent Neural Networks, and the Gated Recurrent Capsule that we introduce in this section is a novel architecture, and is to our best knowledge the very first occurrence of recurrent unit using capsules.

The contributions that we presented in [29] are three-fold:

- we introduce Gated Recurrent Capsules (GRCs), a novel RNN architecture which extents conventional GRUs so that information flow focuses on critical data items;
- we propose to address the caption retrieval task using the newly proposed GRCs architecture;
- we demonstrate experimentally that GRC enable higher performance when compared to state of the art Word2VisualVec (employing GRUs) in the MSCOCO caption retrieval task.

Our presentation is divided in five sections. Having introduced the extent of the work we published in [29] in the current section, we will describe related works in Section 3.4.1. In Section 3.4.2 we will describe our model for caption retrieval. Section 3.4.3 details results obtained by our model. We will conclude our presentation in Section 3.4.5.

3.4.1 *Related Work*

Several works have been done on building visual-semantic embeddings. Most of them are based on the construction of a multimodal space where sentences and images are projected and compared. In [25], Faghri *et al.* used a GRU to map sentences to a multimodal space; images were simply mapped to that space through a linear transform. They obtained good results by finetuning the ResNet they used to produce visual features: that is the reason why we used one of their finetuned ResNets to produce visual features in our model. Another more complex model proposed by Gu *et al.* [31] showed that results could be boosted by the use of two generative models (one generating images and one generating sentences) in addition to a GRU and a ResNet. More recently, [53] has shown that even better performances could be reached by processing images with an object detection model combined with cross-attention instead of deriving global visual features.

Another approach has been proposed recently: instead of mapping images and sentences to a multimodal space, [20, 21] proposed to derive visual features from images and to map directly sentences to the space of visual features. This approach is promising as the quality of visual features is constantly increasing. Moreover, it avoids mapping images to a more complex space. Our work follows that unconventional approach. It has been inspired by recent works on capsule networks [37, 80]. Capsule networks have shown promising results in computer vision. However to our best knowledge they have not been used yet in a recurrent fashion for natural language processing apart from the work we introduced in Section 3.3 that we published in [28]; however, the architecture presented in [28] is using a complex GRUs setup to process and route the information, leading to much more learnable parameters, which is a drawback that our architecture does not have.

3.4.2 *Visual Sentence Embeddings*

Word2VisualVec is a non-conventional approach to caption retrieval, as it maps sentences directly to a visual features space. We decided to test that approach. We eventually compare it to the multimodal approach.

3.4.2.1 *Word2VisualVec*

In [20], a first version of Word2VisualVec was proposed. It consisted in applying a multilayer perceptron on vectorized sentences to project these sentences in a space of visual features. Three vectorization methods were discussed in that paper: bag-of-words, word hashing and averaging Word2Vec embeddings. In [21], the authors of [20] improved

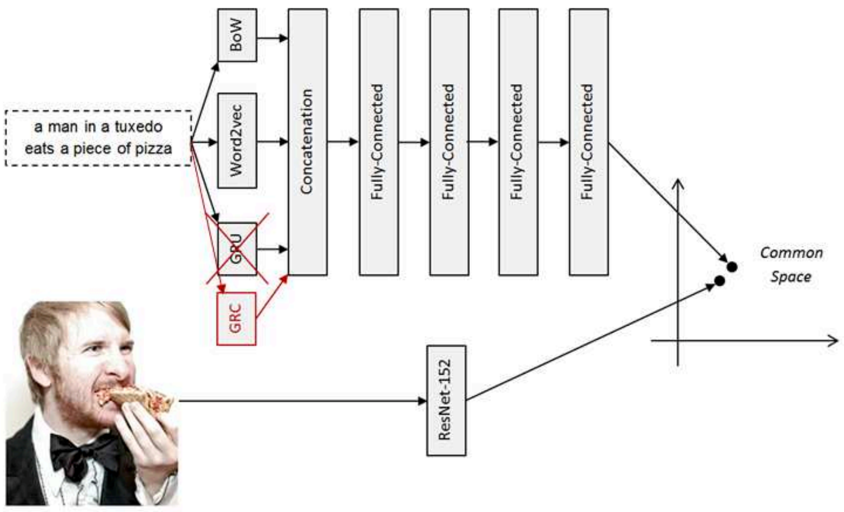


Figure 32: Word2VisualVec and our variant with a GRC. In Word2VisualVec, three sentence representations (Word2Vec, BoW and GRU) are concatenated and then mapped to a visual features space. In our model, we replaced the final hidden state of the GRU by the average of all final hidden states of a GRC.

Word2VisualVec by concatenating three sentence representations. In that paper, a sentence representation was produced by concatenating a bag-of-words, a Word2Vec and a GRU representation of the sentence. Then, it was projected in a space of visual features through a multilayer perceptron. Figure 32 shows how Word2VisualVec works in practice.

On top of good performances in caption retrieval, this visual representation of sentences showed an interest in multimodal query composition: the authors showed that visual words features could be added or subtracted to images features and form multimodal queries. Authors also stated that further gains could be expected by including locality in Word2VisualVec representations.

3.4.2.2 *Gated Recurrent Capsules*

Gated Recurrent Units were introduced by Cho *et al.* in [13]. They are similar to LSTMs: they have similar performances and are well adapted to NLP because they can handle long-term dependencies in sentences. We preferred GRUs to LSTMs because they have less parameters for similar performances. More formally, a GRU is composed of an update gate u_t and a reset gate r_t, and can be described with the following expressions:

$$u_t = \sigma(W_{xu}x_t + W_{hu}h_{t-1} + b_u), \tag{82}$$

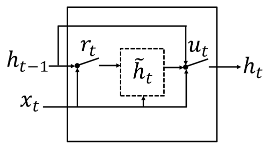


Figure 33: A Gated Recurrent Unit: for each input x_t, a new value $\tilde{h}_t$ is computed, based on x_t, r_t and h_{t-1}, where r_t expresses how much of h_{t-1} should be reset to compute $\tilde{h}_t$. Eventually, h_t is computed based on $\tilde{h}_t$, h_{t-1} and u_t, where u_t expresses how much of $\tilde{h}_t$ should be used to update h_{t-1} to h_t

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r), \tag{83}$$

$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h), \tag{84}$$

$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot \tilde{h}_t, \tag{85}$$

with x_t the t-th input and h_t the t-th output or hidden state of the GRU. The equations above can be explained as follows: for each input x_t, the GRU computes r_t and u_t based on the input and the previous state h_{t-1}. It computes a new value $\tilde{h}_t$ based on x_t, r_t and h_{t-1}, and r_t expresses how much of h_{t-1} should be reset to compute $\tilde{h}_t$. Eventually, h_t is computed based on $\tilde{h}_t$, h_{t-1} and u_t, and u_t expresses how much of $\tilde{h}_t$ should be used to update the hidden state h_t of the GRU. Learned parameters are W_{xu}, W_{hu}, b_u, W_{xr}, W_{hr}, b_r, W_{xh}, W_{hh}, and b_h. GRUs are described with more details in Section 2.1.4.3.

In our case, the x_t correspond to word embeddings: if s is a sentence of length L, then it is first converted into a list $(w_1, ..., w_L)$ of one-hot vectors, and each one-hot vector is mapped to a word embedding using a lookup matrix W_e. Therefore, we have $(x_1, ..., x_L) = (W_e w_1, ..., W_e w_L)$. The coefficients of W_e are learned, but they are initialized to precomputed word embeddings to avoid overfitting problems.

Capsules were designed by [80] for image processing. The idea behind capsules for computer vision consists in making complex computations and outputting a pose vector and an activation. This output is then routed towards subsequent capsules according to some predefined routing algorithm. The goal of that architecture is to have each capsule learning to recognize a visual feature based on what previous capsules have recognized before. For instance, some capsules could recognize eyes, a nose, a mouth and their respective positions. Then they would send their outputs to another capsule aiming at recognizing a whole face. It is architectured to avoid losing spatial information as common CNN do due to pooling operations. We think that capsules can also successfully perform other tasks such as NLP-related tasks, as our proposed model does.

In a nutshell, what we would like to do is to produce different embeddings that would attend to different semantic sides of the input sentence. A sentence would be divided into sub-sentences, and each of those sub-sentences would attend to a particular element of an image. These sub-sentences representations are then processed to build an embedding for the whole sentence.

In our model, all capsules share the same parameters and are similar to GRUs. In the following, we will explain the differences between them and actual GRUs. A recurrent capsule layer should process a sentence word-by-word and make updates in a way that would put attention on important words: the hidden state of each capsule should reflect one semantic side of the input sentence. Therefore, we need to define a routing procedure depending on current states and incoming words. For that purpose, we will use hidden states of capsules at time $t-1$ and the incoming word x_t to find how relevant a word is to a given capsule. More formally, if we consider the k-th capsule with $k \in \{1, ..., N_c\}$, update gates and reset gates will be the same as for a GRU:

$$u_t^{(k)} = \sigma(W_{xu}x_t + W_{hu}h_{t-1}^{(k)} + b_u), \tag{86}$$

$$r_t^{(k)} = \sigma(W_{xr}x_t + W_{hr}h_{t-1}^{(k)} + b_r), \tag{87}$$

We also compute $\tilde{h}_t^{(k)}$ as we do in a GRU:

$$\tilde{h}_t^{(k)} = \tanh(W_{xh}x_t + W_{hh}(r_t^{(k)} \odot h_{t-1}^{(k)}) + b_h), \tag{88}$$

We would like to make our routing procedure trainable via gradient descent, so we need to define differentiable operations. For that purpose, we will assume that for each capsule, for a given word w_t, we have a coefficient $p_t^{(k)} \in [0, 1]$ such that

$$h_t^{(k)} = (1 - p_t^{(k)})h_{t-1}^{(k)} + p_t^{(k)}\hat{h}_t^{(k)} \tag{89}$$

with

$$\hat{h}_t^{(k)} = u_t^{(k)} \odot \tilde{h}_t^{(k)} + (1 - u_t^{(k)}) \odot h_{t-1}^{(k)}, \tag{90}$$

which is the actual update computed in a GRU. The coefficient $p_t^{(k)}$ is a routing coefficient, describing to what extent a given capsule needs to be updated by the incoming word. As in [37], routing can be seen as an attention mechanism, putting attention on relevant words in our case. However, while the authors of [37] use Gaussians determined by EM-routing to compute this coefficient, we propose to compute it in a simpler manner. More details are provided in the next section. We can expand the last equation to get the following update:

$$h_t^{(k)} = (1 - p_t^{(k)} u_t^{(k)}) \odot h_{t-1}^{(k)} + p_t^{(k)} u_t^{(k)} \odot \tilde{h}_t^{(k)} \tag{91}$$

We can notice that it boils down to multiplying the update $u_t^{(k)}$ by a coefficient $p_t^{(k)}$. Then, how to compute $p_t^{(k)}$? For that purpose, we define an activation coefficient $a_t^{(k)}$ for each capsule:

$$a_t^{(k)} = |\alpha_k| + \log(P_t^{(k)}). \tag{92}$$

In the last equation, the α_k are random numbers drawn from a normal probability distribution (we found that 0.1 and 0.001 were good values for the mean and the standard deviation of the normal probability distribution). The α_k are important to our model because all capsules share the same parameters: if all activations are the same when they start processing a sentence, they will be all the same at the end. These random numbers break the symmetry between capsules; this is needed for our model to work properly. We assume $P_t^{(k)}$ ought to represent the semantic similarity between the current hidden state of the capsule $h_{t-1}^{(k)}$ and the incoming word x_t: if the incoming word is semantically similar to the previous hidden state, $P_t^{(k)}$ should be high, and if it is different, then it should be low. One can intuitively imagine that the cosine similarity $\cos(h_{t-1}^{(k)}, \hat{h}_t^{(k)}) = \frac{\langle h_{t-1}^{(k)} | \hat{h}_t^{(k)} \rangle}{\|h_{t-1}^{(k)}\|_2 \times \|\hat{h}_t^{(k)}\|_2}$ corresponds to a relevant definition of the semantic similarity between the current hidden state of the capsule and the incoming word: if the incoming word has a different meaning than previous words, then one can expect that $\hat{h}_t^{(k)}$ will reflect that different meaning. Therefore we define $P_t^{(k)}$ as:

$$P_t^{(k)} = \cos(h_{t-1}^{(k)}, \hat{h}_t^{(k)}). \tag{93}$$

Then we can compute $p_t^{(k)}$ according to the following formula:

$$p_t = \frac{\text{softmax}(\frac{a_t^{(1)}}{T}, \ldots, \frac{a_t^{(N)}}{T})}{M} \tag{94}$$

where M is the maximal coordinate of the vector $\text{softmax}(\frac{a_t^{(1)}}{T}, \ldots, \frac{a_t^{(N)}}{T})$ and T is a hyperparameter controlling the sharpness of the routing procedure (the higher T, the more we have one routing weight equal to 1 and all others equal to 0).

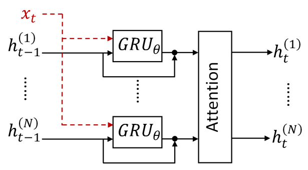


Figure 34: Gated Recurrent Capsules: all capsules share the same learned parameters θ. The inputs of capsule i at time t are a word embedding x_t and its hidden state at time $t-1$ $h_{t-1}^{(i)}$. Its output is $h_t^{(i)}$, and it is computed through the routing procedure described in Section 3.2. This routing procedure can be seen as an attention model: each output depends on how semantically similar the incoming word is to previously processed words. It ensures that each capsule generates a sentence embedding corresponding to one important visual element of the sentence.

Our routing is different from those that were introduced in [37, 80]: the outputs of capsules are not combinations of all previous capsules outputs. Only the weights of the routing procedure depend on these previous capsules outputs.

Please note that if $T \to +\infty$, then all capsules receive the same inputs and produce the same hidden states: it is strictly equivalent to a GRU. Therefore, GRCs are an extension of the GRUs. The interest of GRCs over GRUs is that they can provide different representations of the same sentence, with attention put on some relevant parts of it. This idea is shown on Figure 34. Moreover, a GRC has the same number of trainable parameters as a GRU, but it has the ability to make more complex computations: for that reason we think that this architecture could be successfully used for other tasks than caption retrieval.

The model we propose for caption retrieval is similar to Word2VisualVec, but we replace the GRU by a GRC, as shown on Figure 32. Instead of concatenating the last hidden state of a GRU to a Word2Vec and a bag-of-words representations, we concatenate the average of the last hidden states of a GRC. We also tried to derive a weighted average of the hidden states of a GRC based on a soft-attention mechanism described in [22] but results did not improve. We reported our results in Section 3.4.3 for information.

3.4.2.3 *Improving Word2VisualVec with GRC*

As we said in Section 3.4.2.1, Word2VisualVec relies on three representations of sentences: bag-of-words, average of Word2Vec embeddings and GRU. GRCs provide another representation that we can concatenate to the three previous ones. More precisely, let us assume that we processed a sentence of length L with a GRC containing N_c capsules. Then, if $h_L^{(1)}, \ldots, h_L^{(N_c)}$ are the final hidden states of its capsules, the corresponding representation v_{GRC} of the sentence is the average of all these hidden states:

$$v_{GRC} = \frac{1}{N_c} \sum_{k=1}^{N_c} h_L^{(k)}. \tag{95}$$

This representation is intermediate between the GRU and the Word2Vec representations: it is the sum of N_c different hidden states, each of them corresponding to a particular part of a whole sentence.

Our goal is to map sentences to corresponding images in a space of visual features. One way to measure the efficiency of that kind of mappings is to evaluate the model on caption retrieval. When the model projects both images and sentences in a common multimodal space, recent works have shown that triplet ranking losses were efficient [25]. However in our case, sentences are directly mapped to a space of visual features, no transformation is made on image feature vectors. We found, in accordance with [21], that using the mean squared error (MSE) gave better results than a triplet ranking loss. Therefore, considering a mini-batch $B = ((s_1, x_1), ..., (s_{N_b}, x_{N_b}))$ of sentence-image pairs (N_b is the size of the mini-batch), we defined the loss function $\mathcal{L}_{MSE}(B)$ as follows:

$$\mathcal{L}_{MSE}(B) = \frac{1}{N_b} \sum_{k=1}^{N_b} \|f_\theta(s_k) - \phi(x_k)\|_2^2, \tag{96}$$

where ϕ is a function mapping images to image features and f_θ is a function mapping sentences to image features where θ is the set of all trainable parameters. Our objective is to find a $\hat{\theta}$ minimizing $\mathcal{L}_{MSE}$:

$$\hat{\theta} = \text{argmin}_\theta(\mathcal{L}_{MSE}(\bar{B})) \tag{97}$$

where $\bar{B}$ is the set of all possible image-sentence pairs. We detail the advantages of the MSE loss in Section 2.1.2 for more information. We use the RMSProp method to optimize f_θ, following the procedure we describe in Section 2.1.2.1.

3.4.3 *Experimental Results*

In this section, we present our experimental results. We first introduce the dataset we used to perform our experiments, then we give some implementation details, and eventually we present and discuss our results with respect to the actual Word2VisualVec model, and with respect to a multimodal version of our model.

3.4.3.1 *Dataset*

We evaluated how our models performed on the caption retrieval task on the MSCOCO dataset [58]. This dataset contains 123000 images with 5 captions each, and we split it into a training set, a validation set and a test set according to [43]. The training set contains 113000 images, the validation set contains 5000 images and the test set contains 5000 images.

As for data preprocessing, we converted all sentences to lowercase and removed special characters (apart from spaces and hyphens). We limited the vocabulary to 5000 most used words, and replaced all other words by an "UNK" token. Regarding images, we projected them to a space of visual features. For that purpose, we used the penultimate layer of the ResNet-152 from [25] to get 2048-dimensional features vectors.

3.4.3.2 *Parameters*

Regarding the sentence embedding part of our model, we set its parameters as follows: we set the maximum sentence length to 24 (if longer the sentence is cut after the 24-th word). We initialized W_e using 500-dimensional Word2Vec embeddings trained on Flickr. We also used these embeddings to compute the Word2Vec part of sentences representations. These embeddings are the same as the ones that the authors of Word2VisualVec used in [21]. Regarding the GRC, we found that a model with 4 capsules and $T = 0.4$ performed well. The GRU in Word2VisualVec and the GRC capsules in our model have 1024-dimensional hidden states.

We trained our models using the RMSProp method [87] with mini-batches of 25 image-sentence pairs during 25 epochs. We followed the same learning rate decay procedure as in [21]: the learning rate was initially 0.0001 and we divided it by 2 when the performance of the model on the validation set did not increase during three consecutive epochs. We made all our implementations using the TensorFlow [1] library for Python and used the default parameters of the RMSProp optimizer: decay = 0.9, momentum = 0.0 and epsilon = 1e-10.

3.4.3.3 *Results and Discussion*

To prove the interest of our model, we compared it to Word2VisualVec. We compared the two versions we described in Section 3.4.2.2: the one with the average of final hidden states of capsules and the one with the soft-attention mechanism proposed in [22]. We reported our results in Table 7. They show that our model performs better than Word2VisualVec, and that the attention mechanism does not provide much improvement.

Moreover, we also wanted to see on which kind of sentences GRCs were more efficient than GRUs. For that purpose, we listed all the sentences that our model ranked in the top 9 sentences and that were ranked worse than rank 100 by Word2VisualVec. We also listed sen-

Table 7: Results of our experiments on MSCOCO. R@K denotes Recall at rank K (higher is better). Best results among all models are in bold.

MSCOCO			
Model	Caption Retrieval		
	R@1	**R@5**	**R@10**
Word2VisualVec	32.4	61.3	73.4
W2V + BoW + GRC	**33.4**	62.2	74.0
W2V + BoW + GRC + Attention	32.8	**62.3**	**74.2**

tences ranked by Word2VisualVec in the top 9 that were ranked worse than rank 100 by our model. Our results are summarized in Table 8.

Table 8: For each model: number of sentences ranked in top 9 for the right image by one model and above rank 100 by the other model. Ten sentences are ranked in top 9 by Word2VisualVec while ranked above rank 100 by our model, and seventeen sentences are ranked in top 9 by our model while ranked above rank 100 by Word2VisualVec. We also reported these numbers of sentences without counting sentences containing "UNK" tokens. This table shows that GRCs are performing better than GRUs on much more sentences than GRUs compared to GRCs.

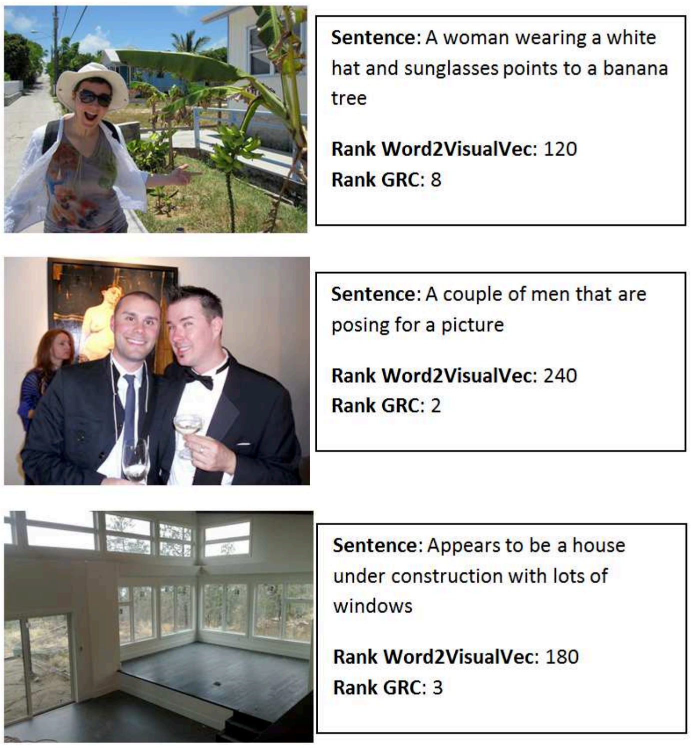
	Word2VisualVec	**Our model**
Total	10	17
Total without UNK tokens	3	11

We noticed that sentences on which GRCs were outperforming GRUs were more likely sentences containing multiple visual concepts. We provide some examples in Figure 35. We think that this observation implies that GRCs could be used efficiently to derive finer visual sentence embeddings, taking into account important local elements.

3.4.4 *Comparison Image Space vs Multimodal Space*

We have also trained a multimodal version of our model, replacing the visual features extractor by the visual embedding model we presented in Section 3.4.2.1. The results we obtained are reported in Table 9.

As one can notice, the multimodal model is much more efficient than the visual model on the caption retrieval task.



Sentence: A woman wearing a white hat and sunglasses points to a banana tree

Rank Word2VisualVec: 120
Rank GRC: 8

Sentence: A couple of men that are posing for a picture

Rank Word2VisualVec: 240
Rank GRC: 2

Sentence: Appears to be a house under construction with lots of windows

Rank Word2VisualVec: 180
Rank GRC: 3

Figure 35: Compared results of Word2VisualVec and our model on three images.

Table 9: Results of our experiments on MSCOCO: comparison between image space and multimodal space. R@K denotes Recall at rank K (higher is better). Best results among all models are in bold.

MSCOCO			
Model	Caption Retrieval		
	R@1	**R@5**	**R@10**
Image Space	33.4	62.2	74.0
Multimodal Space	**33.5**	64.1	77.0

3.4.5 *Conclusion of Section 3.4*

In [29], we introduced a novel RNN architecture called Gated Recurrent Capsules (GRCs). We built a model to address the caption retrieval task by mapping images and sentences to a visual features space. We showed in our experimental work that the models obtained using the proposed GRCs are surpassing those from earlier works (employing GRUs). Moreover, we stated that GRCs could potentially be used in any typical RNN tasks, as they are an extension of GRUs. Eventually, we showed that a multimodal version of our model was more efficient than the visual one. Therefore, as we wanted to assess our results at the TRECVid VTT Matching task that we will describe in Section 3.5, aiming at performing video caption retrieval, we decided to focus on multimodal models.

3.5 GRCS FOR THE VIDEO-TO-TEXT (VTT) TASK

EURECOM participated in the Sentence Matching subtask of the TRECVid 2018 [7] Video-to-Text (VTT) task for the first time. The approach we followed was to adapt the model we presented in Section 3.4 for video-sentence matching, following the setup of the winning team of 2017 [22]. The Sentence Matching subtask of the VTT task requires to link videos and sentences describing these videos. Testing data is composed of 1,000 videos, and five datasets of 1,000 sentences, each sentence corresponding to one video. For each video and for each dataset of sentences, teams are asked to rank sentences from the closest to the most dissimilar. Evaluation is performed on each sentence dataset using the Mean Inverse Rank measure.

3.5.1 *Our Model*

In this section, we will present the model that we used at TRECVid VTT and some variations we also submitted.

3.5.1.1 *Definition of our model*

As stated in the introduction of Section 3.5, our model aims at adapting the model we presented in Section 3.4 for video-sentence matching, following the setup of the winning team of 2017 [22]. In [22], video embeddings are derived as follows:

- frames are extracted every 0.5 second for each video;
- features vectors $(v_1, ..., v_n)$ are derived from these frames using the penultimate layer of a ResNet-152 [35];
- these features vectors are then fed sequentially into a GRU [13], whose hidden states $(h_1, ..., h_n)$ are concatenated to correspond-

ing features vectors, to obtain contextualized features vectors $(s_1, ..., s_n) = (v_1 \| h_1, ..., v_n \| h_n)$;

- these contextualized features vectors are combined through a soft attention mechanism to form a vector v, which is actually a weighted sum of $s_1, ..., s_n$;
- this vector v is then projected into a vector space after two fully-connected layers with ReLU activations, where each activation is preceded by a batch normalization.

In our model, the same process is applied for computing video embeddings. However, before feeding v into the two fully-connected layers, we concatenated it with a vector that we derived from the video using the last layer of an RGB-I3D [11]. Moreover, [22] used a ResNet-152 trained on ImageNet [16] whereas we used the ResNet-152 trained on ImageNet and finetuned on MSCOCO [58] proposed by [25].

In [22], text-embeddings are derived as follows:

- three text representations are derived (one using an average of Word2Vec [65] embeddings, a second one is a BoW representation and a third one is derived by taking the last hidden state of a GRU) and concatenated;
- the resulting vector is then fed into two consecutive fully-connected layers following the same process as for videos.

Regarding the text-embeddings part of our model, we tried to replace the GRU by GRCs [29] or a bidirectional GRU. GRCs are extensions of GRUs that we proposed in [29], where we showed that they could improve results of GRUs on multimodal matching tasks. Our model is summarized by Figure 36.

3.5.1.2 *Training*

We trained our model using a common hard-negative triplet ranking loss [25]. More formally, if v is the embedding computed for a video, s the embedding computed for a sentence corresponding to that video, then the loss $l(v, s)$ corresponding to the couple (v, s) is defined as follows:

$$l(v, s) = \max_{\bar{s} \neq s}(\max(0, \alpha - \cos(v, s) + \cos(v, \bar{s}))), \quad (98)$$

where α is a hyperparameter that we set to 0.2.

We used several datasets for training and validation:

- MSVD [12];
- MSR-VTT [98];

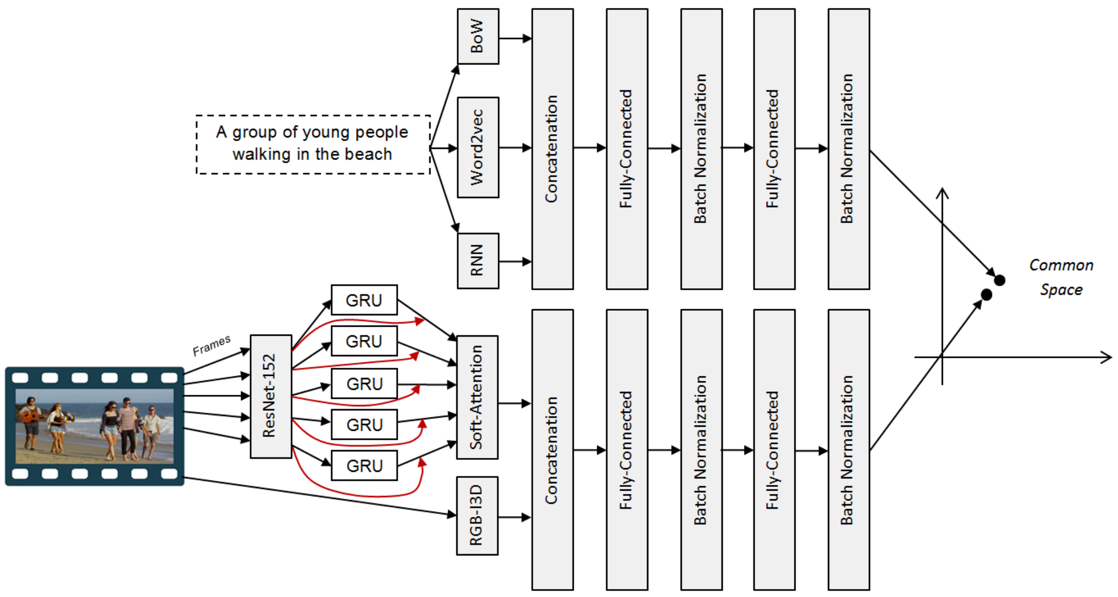


Figure 36: Our model. RNN can be a GRU, a GRC or a bidirectional GRU.

- TGIF [56] (for computer memory problems, we only used 60,000 sentence-video pairs from TGIF);
- TrecVid VTT 2016 test data [5];
- TrecVid VTT 2017 test data [6].

Our validation set was composed of 200 videos from TrecVid VTT 2016 test data and 200 videos from TrecVid VTT 2017 test data with corresponding sentences. Therefore, the validation set contained 400 different videos. We used MSVD, MSR-VTT and TGIF for training, for a total of 65,782 different videos with all corresponding sentences. Eventually, we used the remaining data from TrecVid VTT 2016 and TrecVid VTT 2017 to form a finetuning dataset of 3,088 videos.

We trained our models using the RMSProp method [87] with TensorFlow default parameters and gradient clipping between -5 and 5. We first trained our model on the training set, applying a learning rate of 0.00003 during 20 epochs (dividing the learning rate by two if validation loss did not improve during three consecutive epochs), with mini-batches of size 25. Then, we set the learning rate to 0.00002, and finetuned our model on the finetuning dataset during 60 epochs, dividing the learning rate by two if validation loss did not improve during three consecutive epochs.

3.5.2 *Our runs*

Now, we will describe the runs we actually submitted to TRECVid VTT, and the results we obtained with respect to other teams.

Table 10: Our results in terms of Mean Inverse Rank

Runs	Subset A	Subset B	Subset C	Subset D	Subset E
Run 1	0.194	0.190	0.194	0.193	0.199
Run 2	0.197	0.197	0.197	0.184	0.204
Run 3	0.202	0.209	0.206	0.186	0.212
Run 4	0.231	0.240	0.234	0.224	0.241

3.5.2.1 *Definition of runs*

EURECOM submitted four different runs to the VTT Sentence Matching subtask. The runs are numbered from 1 to 4, with the expected best runs having the highest numbers. For each run and for each video, sentences are ranked by decreasing cosine similarity.

RUN 1 : We apply the model we described in Section 3.5.1. The RNN we used for computing sentence embeddings was a simple GRU.

RUN 2 : This run was similar to RUN 1, but we replaced the GRU by a GRC.

RUN 3 : In this run, the GRU of RUN 1 is replaced by a bidirectional GRU.

RUN 4 : This final run is a merge of previous runs. The merge is performed by summing the cosine similarities of the three previous runs, to obtain a new score for each sentence.

3.5.2.2 *Results*

We reported our results in Table 10. Our runs are ranked as we expected on subsets A, B, C and E. It is not the case for subset D, as the simple GRU obtained better results than the GRC and the bidirectional GRU.

In Figures 37-41, all results on different sentences subsets are presented. Our results are in red. As one can see, our ensemble method did better than other methods.

Our results were very encouraging, as we ended third out of eleven teams. Thinking of our tries at AVS 2016 and AVS 2017 with textual and visual embeddings, we planned to participate in AVS 2019 with multimodal embeddings. However, results were not as good as we expected.

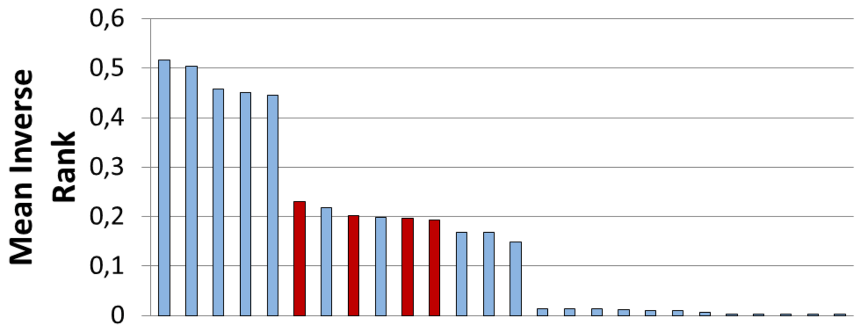


Figure 37: Results on subset A

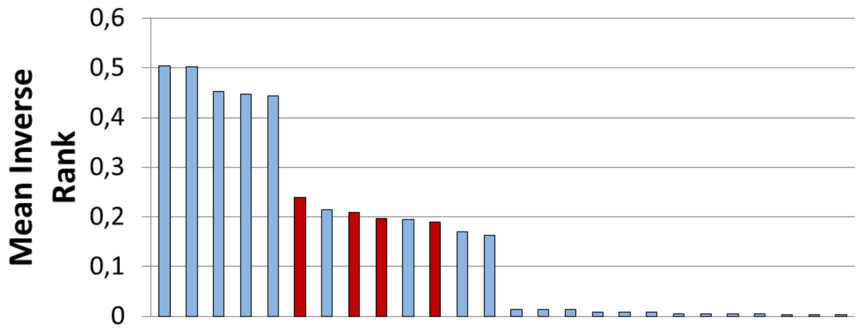


Figure 38: Results on subset B

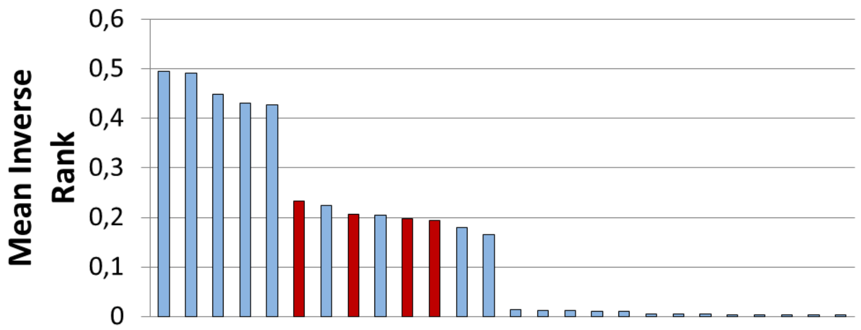


Figure 39: Results on subset C

3.6 Limits of Multimodal Spaces: Ad-Hoc Video Search

As mentioned in Section 3.2, Ad-Hoc Video Search (AVS) is a challenging task consisting in using natural language queries to retrieve relevant video segments from large datasets (see Figure 42). “Ad-Hoc” implies that the query follows no pattern, and the terms are drawn from an open vocabulary that is not restricted to any particular domain. Due to this nature, a query can be very specific (e.g., "find shots of a surfer standing on a surfboard, not in the water") or open ended

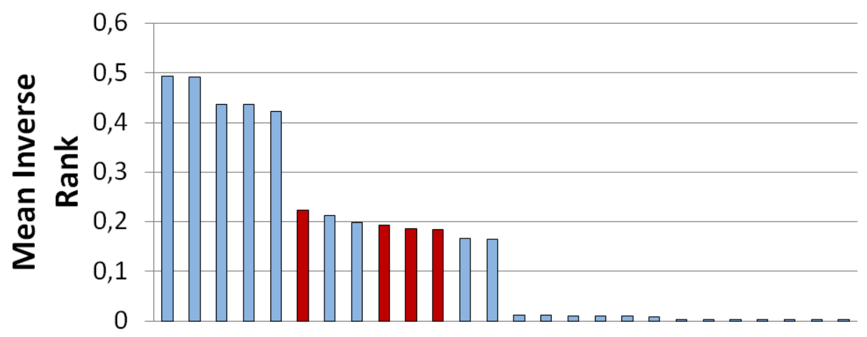


Figure 40: Results on subset D

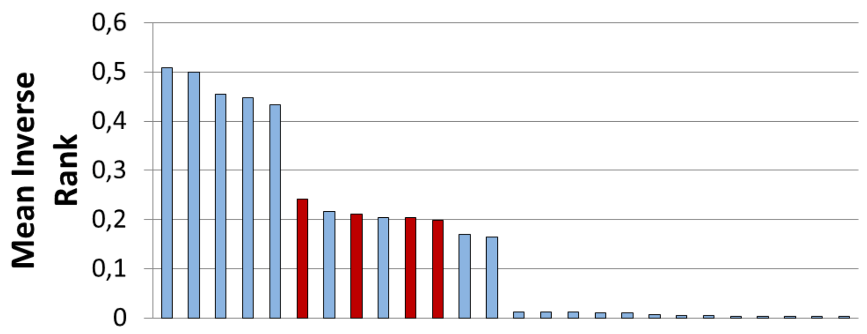


Figure 41: Results on subset E

(e.g., "find shots inside a moving car"). Therefore, a "good" model for AVS should be able to extract relevant high-level features from videos, parse text queries, and find a common representation of both modalities for relevance judgement. TRECVID, an evaluation campaign, is organized yearly by the NIST to evaluate state-of-the-art models for different video processing tasks, including the AVS task [6]. In this section, the focus of study is automatic AVS using external training data.

Most recent works in image and video processing and in text processing rely on Deep Learning techniques. For image processing, commonly used feature extractors or concepts detectors are deep Convolutional Neural Networks (CNNs) which have been pretrained on ImageNet 1000 categories [16]. For video processing, I3D trained on the Kinetics dataset has shown excellent results in activity detection [11]. Regarding text processing, recent works have shown that RNNs such as LSTMs [38] or GRUs [13] could be used to build efficient language models.

In [30], we proposed using a fusion of three multimodal modules trained on different datasets to tackle the AVS task. Our contributions in that work are two-fold:

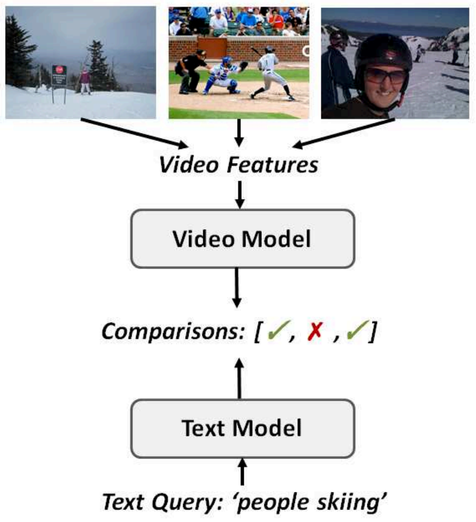


Figure 42: Principle of AVS. Video Features are derived from videos and processed by a Video Model to obtain a vector representation. At the same time, a text query is processed by a Text Model that also derives a vector representation. These two vector representations are then compared to list all relevant videos with respect to the text query.

- joint exploitation of object counting, activity detection and semantic concept annotation for query interpretation;
- a new fusion method that combines three modules trained on different datasets and shows competitive performance.

We also show the limits of the use of multimodal spaces to tackle the AVS task with respect to methods based on visual spaces or word spaces.

Our presentation is organized as follows. Section 3.6.2 introduces the related works in AVS. Section 3.6.3 introduces the cross-modal learning employed for training three different modules, while Section 3.6.3 describes the proposed fusion method. Section 3.6.4 provides empirical insights and Section 3.6.6 concludes this presentation.

3.6.1 *Related Works*

From AVS 2018 [7], the general approaches from the participants can be summarized as follows: linguistic analysis for query understanding combining different techniques for concept selection and fusion; or learning joint embedding space of textual queries and images; or the integration of two mentioned approaches. From the results of ten participants, we conclude that the approach of learning the embedding space is the key of success for AVS task. Following up this direction, we propose to learn three embedding spaces including objects counting, activities and semantic concepts separately, and a fusion method to incorporate these models.

3.6.2 *Cross-Modal Learning*

In this section we will describe the multimodal models we employed. More precisely we will first define their architecture and then how we trained them. Please note that in this section, we will consider images and videos, even though our models will be used for Ad-Hoc Video Search. The reason is that some of our models will be trained on images and applied at frame-level on videos. More informations will be given at Section 3.6.4.2.

3.6.2.1 *Feature Representation*

Let Q be a textual query and V an image or a video. We want to build a model so that Q and V can be compared. More precisely, we want to be able to assign a score to any (Q, V) to describe the relevance of V with respect to Q. For that purpose, we use a similar model to [25].

For processing textual queries, we represent any query Q of length L as a sequence $(w_1, ..., w_L)$ of one-hot vectors of dimension N, where N is the size of our vocabulary. These one-hot vectors are then embedded in a vector space of dimension D. More formally, we obtain a sequence of word embeddings $(x_1, ..., x_L)$ where $x_k = w_k W_e$ for each k in $\{1, ..., L\}$. The weights of the embedding matrix $W_e \in \mathbb{R}^{D \times N}$ are trainable.

The obtained sequence of word embeddings is then processed by a GRU, whose last hidden state is kept and input to a Fully-Connected layer to get a sentence embedding. Formally, a GRU is defined by the following equations:

$$u_t = \sigma(h_t W_{uh} + x_{t+1} W_{ux} + b_u) \tag{99}$$

$$r_t = \sigma(h_t W_{rh} + x_{t+1} W_{rx} + b_r) \tag{100}$$

$$\bar{h}_t = \tanh((h_t \odot r_t) W_{hh} + x_{t+1} W_{hx} + b_u) \tag{101}$$

$$h_{t+1} = (1 - u_t) \odot h_t + u_t \odot \bar{h}_t \tag{102}$$

where W_{uh}, W_{ux}, W_{rh}, W_{rx}, W_{hh}, W_{hx}, b_u, b_r and b_u are trainable parameters. More information on GRU is given in Section 2.1.4.3. If the length of the input sequence is L, then the final sequence embedding v_s is defined as:

$$v_s = h_L W_s + b_s \tag{103}$$

where W_s and b_s are trainable parameters.

Regarding visual objects, the generic process we employ is to extract a vector representation $\varphi(V)$ of a visual object V where φ corresponds to any relevant concepts or features extractor. Then, we input $\varphi(V)$ to a Fully-Connected layer to obtain a visual embedding v_v:

$$v_v = \varphi(V)W_v + b_v \tag{104}$$

where W_v and b_v are trainable parameters.

Our goal is to train these models to be able to compare v_s and v_v. We will explain how these models are trained in Section 3.6.2.2.

3.6.2.2 *Model Training*

The objective is to learn a mapping such that the relevancy of a pair of a query and a video (Q, V) can be evaluated. As explained in Section 3.6.2.1, our model derives a query representation v_s from Q and a video representation v_v from V. Triplet loss is used as the loss function for model training. Mathematically, if we consider a query representation v_s, a positive video representation v_v (corresponding to v_s) and a negative video representation $\bar{v}_v$ (that does not correspond to v_s), the triplet loss $\mathcal{L}$ for $(v_s, v_v, \bar{v}_v)$ to minimize is defined as follows:

$$\mathcal{L}(v_s, v_v, \bar{v}_v) = \max(0, \alpha - \cos(v_s, v_v) + \cos(v_s, \bar{v}_v)) \tag{105}$$

where α is a margin hyperparameter. We chose to employ the hard-margin loss presented in [25], where $\bar{v}_v$ is chosen to be the representation of the negative video with the highest similarity with the query representation v_s among all videos in the current training mini-batch.

3.6.3 *Fusion Strategy*

In this section we will describe the three multimodal modules we used and how we fused them.

3.6.3.1 *Multimodal Modules*

Our model relies on three multimodal modules: a counting module, an activity module and a concepts module (see Figure 43). Each of them has the architecture we described in Section 3.6.2.1 and has been trained according to the optimization scheme we defined in Section 3.6.2.2.

The counting module is based on a Faster-RCNN [75] trained on the OpenImagesv4 dataset [51]. It takes images as inputs. For each input, it detects objects belonging to the 600 classes of OpenImagesv4 and counts them to obtain a vector of dimension 600, where the value at index i corresponds to the number of detected objects of class i. Embeddings are then derived from that vector.

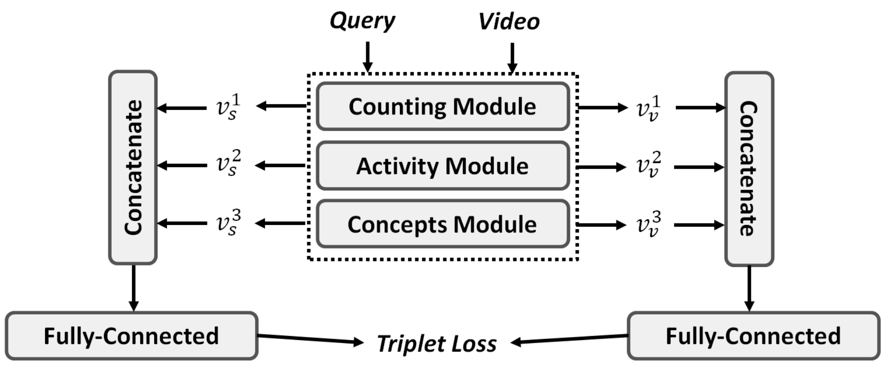


Figure 43: Proposed model. We extract embeddings from three modules: a counting module, an activity module and a concepts module. These embeddings are then concatenated and input to Fully-Connected layers to obtain new embeddings. That model is also trained using a triplet loss.

The activity module relies on an I3D trained on Kinetics-600 and takes video inputs. Each input is processed by the I3D, which returns a vector of 600 logits corresponding to the 600 activities of the Kinetics-600 dataset. That vector is then processed as described in Section 3.6.2.1 to obtain an embedding.

The concepts module takes as input concepts detections coming from four different concept detectors. These concept detectors are ResNet [35] models trained on ImageNet1k, Places-365 [106], TRECVID SIN [105] and HAVIC [83]. Following the same process as for other two modules, we generate embeddings from the concatenation of the concept detections coming from these four detectors.

3.6.3.2 *Fusion Model*

Instead of simply averaging similarity scores to compare videos and queries, we chose to train a model to draw finer similarities between them. For that purpose, we derived embeddings from our modules for videos and queries, and passed them through Fully-Connected layers to obtain new embeddings. More formally, if v_v^1, v_v^2 and v_v^3 are video embeddings respectively generated by the counting module, the activity module and the concepts module, we derived the new video embedding v_v as follows:

$$v_v = \text{concat}(v_v^1, v_v^2, v_v^3) W_v^{\text{fuse}} + b_v^{\text{fuse}} \tag{106}$$

where W_v^{fuse} and b_v^{fuse} are trainable parameters. Similarly, if v_s^1, v_s^2 and v_s^3 are query embeddings, we obtain a new query embedding as follows:

$$v_s = \text{concat}(v_s^1, v_s^2, v_s^3) W_s^{\text{fuse}} + b_s^{\text{fuse}} \tag{107}$$

where W_s^{fuse} and b_s^{fuse} are trainable parameters.

We trained our fusion models using the same triplet loss as we did for multimodal modules, as decribed in Section 3.6.2.2.

3.6.4 *Experiments*

In this section, we describe how we implemented and trained our models, and present our experimental results.

3.6.4.1 *Datasets*

We used the MSCOCO [58] dataset to train the counting module (not the Faster-RCNN itself) and the concepts module. MSCOCO is composed of about 120k images, and five captions per image. We trained modules on the whole dataset, using 1k images for validation: we did not employ the usual train/validation/test split.

Regarding the activity module, it has been trained on the TGIF [56] dataset (containing about 100k animated GIF images and 125k captions) and on the MSVD [12] dataset (containing 1970 videos and about 70k captions).

Fusion models have been trained on the MSR-VTT [98] dataset, containing 10k videos with 20 captions each. We used the usual split: 6513 videos for training, 497 for validation and 2990 for testing.

Our models have also been evaluated in terms of mean average precision based on 10,000 retrieved shots on V3C1 [10], containing 7475 videos split into 1,082,657 shots, using the provided groundtruth results for six queries.

3.6.4.2 *Implementation details*

We implemented our models using the Tensorflow [1] framework for Python. Each of them has been trained for 150k iterations with minibatches of size 64. We used the RMSProp [87] algorithm, with gradients capped to values between -5 and 5 and a learning rate of 10^{-4}. Hidden dimensions of GRUs are always 1024, and embeddings output by multimodal modules and fusion models are of dimension 512. The size of vocabularies has been set to 20k. We applied dropout [82] with rate 0.3 to all outputs of Fully-Connected layers, and batch normalization [41] to the inputs of our models. In triplet losses, the α parameter has been set to 0.2.

Modules trained on images (counting and concepts modules) are used for videos during testing in two different ways. For tests on MSR-VTT, we extracted uniformly one frame every fifteen frames, applied the extractor on each frame (Faster-RCNN for the counting module or concepts extractors for the concepts module) and averaged obtained vectors. For tests on V3C1, we processed provided keyframes instead of entire videos.

3.6.4.3 *Performance of Modules*

Model	R@1	R@5	R@10	medR
M_1 (Counting)	2.95%	7.16%	11.40%	264
M_2 (Activity)	2.83%	8.80%	13.59%	**167**
M_3 (Concepts)	**3.88%**	**10.69%**	**15.44%**	168

Table 11: Results on the MSR-VTT test dataset of three modules.

Model	mAP
M_1 (Counting)	2.15%
M_2 (Activity)	0.00%
M_3 (Concepts)	2.15%

Table 12: Results on the V3C1 dataset of three modules.

In this section, we report results of each module on the unique video retrieval task (evaluated on MSR-VTT) and the multiple videos retrieval task (evaluated on V3C1). Results are reported in Table 11 and Table 12.

One can notice that relative results of modules are completely different with respect to the task. On the unique video retrieval task, the counting module has the worst results, and the concepts module has the best results. On the multiple videos retrieval task, the counting module and the concepts module perform similarly, and the activity module has very bad results.

We think that these results are due to the fact that shots in the V3C1 datasets are much shorter than the videos on which the I3D activity extractor has been trained. For that reason, we will not report results of fusions involving the I3D on V3C1 in the following. Regarding the fact that the counting module performs as well as the concepts module on the multiple videos retrieval task, our hypothesis is that the multiple videos retrieval task requires less precision than the unique video retrieval task: the concepts module covers a large range of visual concepts, which is useful when looking for a specific video, but less useful when the goal is to retrieve as many videos as possible.

In the next section, we will present results of fusions

3.6.4.4 *Performance of Fusions*

Results of fusion models are reported in Table 13 and Table 14. Two types of fusions have been tested : $M_i + M_j$ means that we summed up similarity scores between modules M_i and M_j, and $F(M_i, M_j)$ means that we applied the fusion scheme we described in Section 3.6.3.

Model	**R@1**	**R@5**	**R@10**	**medR**
$M_1 + M_2$	3.91%	11.31%	16.87%	133
$M_1 + M_3$	4.29%	11.56%	16.22%	149
$M_2 + M_3$	4.69%	13.31%	19.19%	105
$M_1 + M_2 + M_3$	**5.00%**	**13.70%**	**19.37%**	**104**
$F(M_1, M_2)$	5.20%	15.78%	23.69%	59
$F(M_1, M_3)$	4.80%	14.70%	22.09%	70
$F(M_2, M_3)$	5.90%	18.00%	26.39%	49
$F(M_1, M_2, M_3)$	**6.48%**	**19.27%**	**27.99%**	**42**
Sum of best	6.72%	17.80%	24.72%	67

Table 13: Results on the MSR-VTT test dataset of fusions of modules. Sum of best is the sum of $M_1 + M_2 + M_3$ and $F(M_1, M_2, M_3)$.

Model	**mAP**
$M_1 + M_3$	4.54%
$F(M_1, M_3)$	4.01%
$F(M_1, M_3) + M_1 + M_3$	**5.41%**

Table 14: Results on the V3C1 dataset of fusions of modules.

In each case, the best model involves a fusion according to our fusion method. In the unique video retrieval task, the fusion alone performs better than other models whereas in the multiple videos retrieval task, the sum of similarity scores of modules and of their fusion has the best results. The reason may be that our fusion scheme makes finer representations of videos, which is less useful for multiple videos retrieval than for unique video retrieval.

3.6.5 *Results on TRECVid AVS 2019*

For AVS 2019, we submitted three runs: one was based on our counting module, another one on our concepts module and the last one was a fusion of both modules. Best results of each team have been reported on Figure 44.

As one can notice, we performed very poorly, as we ended as the last team. It seems that using concepts detectors without multimodal embeddings and big ensemble methods leads to better results. Looking at these results, we think that the approach to AVS must be different from the approach to VTT. In the VTT task, we have the prior knowledge that there is one and only one textual description corresponding to a given video. However for AVS, we just need to retrieve as many videos possible corresponding to one given query. Training a model on matching data may induce too much attention to details,

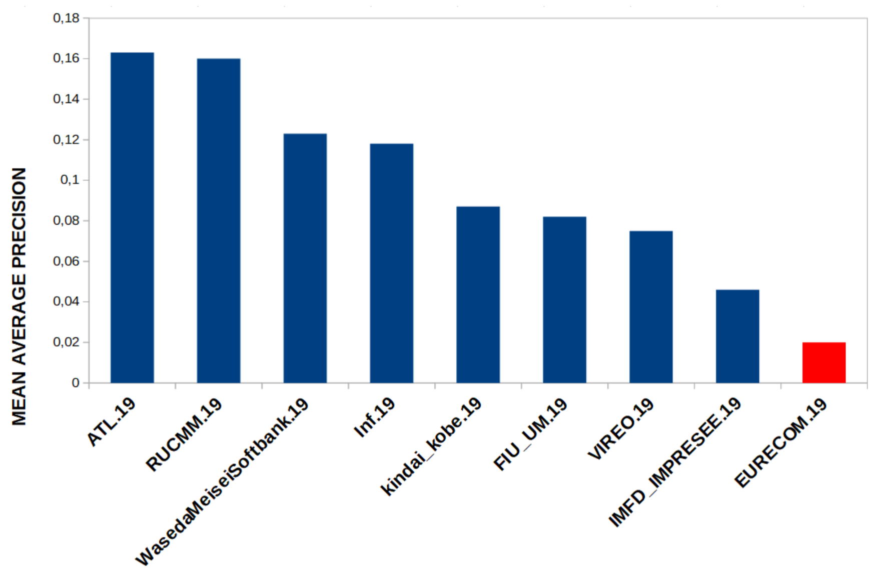


Figure 44: Best result for each team at TRECVid AVS 2019. We performed poorly, as we finished last.

which can be relevant when the task is to find the only corresponding text to a video. However, even though transforming a textual query into a list of visual concepts may be less precise, it is probably more inclined to find a high number of videos that are related to a given textual query.

3.6.6 *Conclusion*

In [30], we proposed to tackle the AVS problem using three modules: a counting module, an activity module and a concepts module. Each of these modules analyzes videos and derives embeddings in a multimodal space. We showed that jointly taking advantage of counting objects, detecting activities and detecting semantic concepts in videos allowed to deal efficiently with the complexity of the AVS task. Moreover, we proposed a method to fuse modules trained on different datasets that appeared to lead to significantly better results than simpler fusion methods.

However, we also showed that using fully learned multimodal spaces was not efficient for the AVS task, even though it was efficient for the VTT task, as mentioned in Section 3.6.5.

3.7 GENERAL CONCLUSION OF CHAPTER 3

In this chapter, we tackled several tasks related to matching images and texts or videos and texts. For that purpose, we architectured two neural network models inspired by capsule networks: the Recurrent Capsule Network and the Gated Recurrent Capsules. We also showed that for matching a single image or a single video to a single text, embedding both modalities in a multimodal space was more efficient than embedding them in a visual space. We assessed our results using common datasets such as Flickr 8k and MSCOCO.

However, we found that the multimodal approach was not as efficient for the AVS task, where multiple videos need to be retrieved based on one natural language query. We think that in that case, a method based on concepts extraction would be a better idea, and we think that it should be the right research direction to improve results on the AVS task.

4 ATTENTION AND CURRICULUM LEARNING FOR CAPTIONING

4.1 INTRODUCTION

In Chapter 3, we introduced models for image-sentence matching and video-sentence matching. The semantic representations that we used for these matching tasks were mostly multimodal embeddings. In this chapter, we propose to generate descriptive sentences from images and videos, which is another type of semantic representation. These representations can be useful for other applications than multimodal embeddings. Automatic image captioning can be useful for visually impaired people, to give automatically a textual descriptions of images they cannot see, on websites for instance. Automatic video captioning can be used for instance to enrich TV programs with textual information on scenes, that can improve the user experience. Multimodal embeddings are not straightforwardly readable by humans, therefore they are mostly useful for back-end applications, such as multimedia indexing and retrieval.

The image captioning task and the video captioning task consist in automatically generating short textual descriptions for images and videos respectively. They are challenging multimedia tasks as they require to grasp all information contained in a visual document, such as objects, persons, context, actions, location, and to translate this information into text. This task can be compared to a translation task: except instead of translating a sequence of words in a source language into a sequence of words in a target language, the aim is to translate a photograph or a sequence of frames into a sequence of words. Therefore, most of recent works in captioning rely on the encoder-decoder framework proposed in [85], initially for text translation. In image or video captioning, the encoder aims at deriving an image or video representation, respectively. Recent advances in deep learning have shown to fit very well to that task. In particular, Convolutional Neural Networks (CNNs) have proved to give excellent results in producing highly descriptive image representations or video representations. The decoder part aims at generating a sentence based on the representation produced by the encoder. Long Short-Term Units (LSTMs) [38] and Gated Recurrent Units (GRUs) [13] are usually chosen for that task. Image captioning [92] and video captioning can seem to be similar tasks, as both of them require to "translate" a visual object into a textual one. However, video captioning poses a problem that makes it more challenging than image captioning: it requires to take into account temporality.

Our contributions in that chapter are two-fold:

- we propose a Learned Spatio Temporal Pooling (L-STAP) method to tackle both temporal and spatial complexity in videos to generate video captions;
- we propose two Curriculum Learning algorithms to improve results in captioning.

Section 4.2 deals with the L-STAP method. In Section 4.3, we introduce our two Curriculum Learning algorithms. We conclude the chapter in Section 4.4.

4.2 VIDEO CAPTIONING WITH ATTENTION

In the work we will introduce in this section, we aim at attending to relevant regions of a video based on previous frames, because the relevance of objects, persons or actions relies on the context in which they appear; and that context should be inferred from previous frames. More precisely, after deriving frame-level local features using the last convolutional layer of a ResNet-152 [35], we do not apply an average pooling to pool these local features. We process them by Learned Spatio Temporal Adaptive Pooling (L-STAP). L-STAP attends to specific regions of a frame based on what occurred previously in the video. Our pooling method is learned because it is based on an LSTM whose parameters are learned. It is spatio-temporal because it takes into account space and time in a joint fashion. In addition, it is adaptive because the attention paid to local regions is based on previous hidden states of the LSTM; pooling depends not only on the processed frame but also on previous ones. A high-level schematic view of our proposed model is depicted in Figure 45.

We evaluated our results on two common datasets used for benchmarking video captioning tasks: MSVD [12] and MSR-VTT [98]. Results show that our model based on L-STAP outperforms state-of-the-art models in terms of several metrics. An ablation study also shows that our method leads to significant improvements with respect to state-of-the-art methods.

Our contributions can be summarized as follows: we propose a novel pooling method for video processing, which we evaluate on the video captioning task, even though it could be applied to any other task involving video processing, such as video classification. Moreover, we demonstrate the interest of our pooling method over usual approaches.

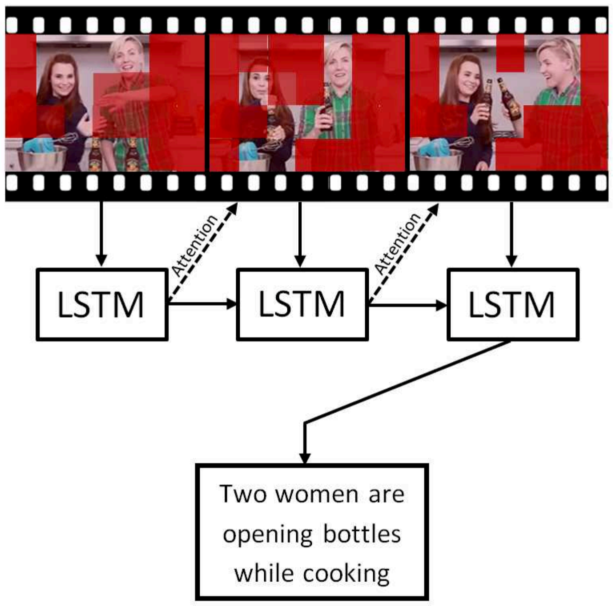


Figure 45: Overview of our L-STAP method. Frame-level local features are derived using a ResNet-152. Then, an LSTM processes these local features, and updates its hidden state by attending to them based on previous frames. The result is that space and time are jointly taken into account to build video representations.

4.2.1 *L-STAP: Our Model*

Let us first formulate the problem we are to deal with. Given a video V, which is a sequence of T frames $(v^{(1)}, ..., v^{(T)})$, our goal is to derive a descriptive sentence $Y = (y_1, ..., y_L)$. The approach that we have followed is based on the encoder-decoder framework. The encoder first derives frame-level representations $(x^{(1)}, ..., x^{(T)}) = X$, and then pool these representations together to form frame-level video representations $(\overline{h}^{(1)}, ..., \overline{h}^{(T)})$. Based on these representations, the decoder reconstructs a descriptive sentence in a recurrent fashion. Figure 46 summarizes the important steps our model. In the following section, we will describe it in detail and report how we train it.

4.2.1.1 *Grasping Spatio-Temporal Dependencies with L-STAP*

As we stated above, the first step is to produce a representation of the input video. In the following subsections, we will explain how we derive frame-level features, and how we pool them together.

Given a video $V = (v^{(1)}, ..., v^{(T)})$, we need to derive features for each frame $v^{(t)}$. A common way to do so is to process each frame using a CNN, which has been previously pretrained on a large-scale dataset. In works such as [59], the outputs of the penultimate layer of a ResNet-152 have been chosen as frames representations, which consist of 2048-dimensional vectors. However, such representations

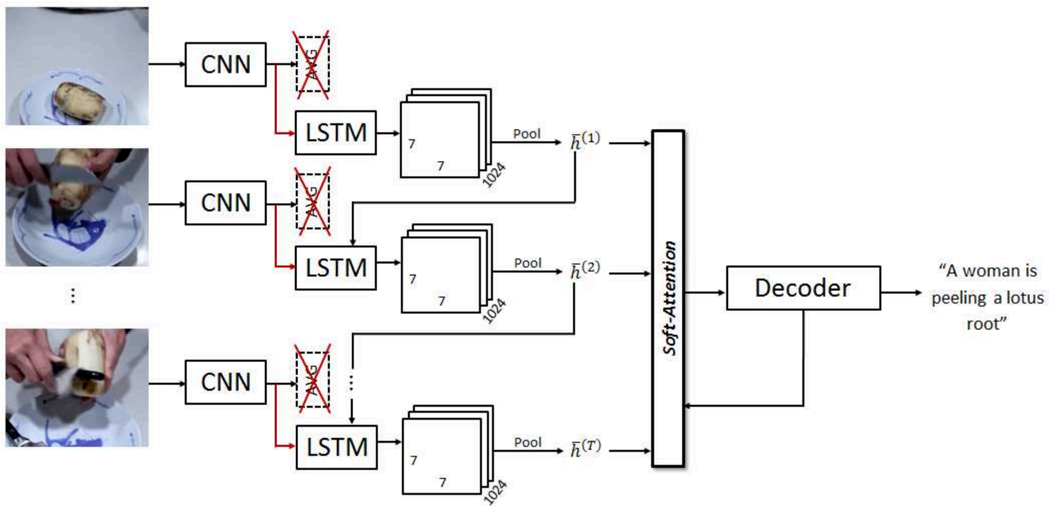


Figure 46: Illustration of our model, based on the proposed L-STAP method. Frames are processed sequentially by a CNN (a ResNet-152 in this case). However, instead of applying an average pooling on local features as some recent works do, we make use of an LSTM to capture time dependencies. Local hidden states are computed to obtain a 7x7x1024-dimensional tensor. These local hidden states are then pooled together (using average pooling or soft attention), and processed by an LSTM decoder to output a sentence.

discard locality, which results in loss of information. Therefore, in this work, we choose to take the output of the last convolutional layer of a ResNet-152. Thus, we obtain frame-level representations $(x^{(1)}, ..., x^{(T)}) = X$, where $x^{(t)} \in \mathbb{R}^{7\times7\times2048}$ for all t. The next step is to process these dense frame-level representations to derive compact frame-level representations, using the proposed L-STAP method instead of conventional pooling.

L-STAP aims at replacing the average pooling operation after the last convolutional layer in a CNN, and to pool local features according to previous frames. The goal is to capture where important actions are occurring, and to discard locations that are not relevant to summarize what is happening in a video. For that purpose, we use an LSTM, taking local features as inputs, resulting in local hidden states, which are then combined in a way we will describe later in this subsection. More formally, given local features $x_{ij}^{(t)} \in \mathbb{R}^{2048}$, the aggregated local features $h_{ij}^{(t)}$ are computed recursively as follows:

$$i_{ij}^{(t)} = \sigma(W_{ix}x_{ij}^{(t)} + W_{ih}\overline{h}^{(t-1)} + b_i) \tag{108}$$

$$f_{ij}^{(t)} = \sigma(W_{fx}x_{ij}^{(t)} + W_{fh}\overline{h}^{(t-1)} + b_f) \tag{109}$$

$$o_{ij}^{(t)} = \sigma(W_{ox}x_{ij}^{(t)} + W_{oh}\overline{h}^{(t-1)} + b_o) \tag{110}$$

$$c_{ij}^{(t)} = f_{ij}^{(t)} \circ \overline{c}^{(t-1)} + i_{ij}^{(t)} \tanh(W_{cx} x_{ij}^{(t)} + W_{ch} \overline{h}^{(t-1)} + b_c) \quad (111)$$

$$h_{ij}^{(t)} = o_{ij}^{(t)} \circ \tanh(c_{ij}^{(t)}) \quad (112)$$

where W_{ix}, W_{ih}, b_i, W_{fx}, W_{fh}, b_f, W_{ox}, W_{oh}, b_o, W_{cx}, W_{ch} and b_c are trainable parameters, and $\overline{c}^{(t-1)}$ and $\overline{h}^{(t-1)}$ are respectively the memory cell and the hidden state of the LSTM. Please note that memory cells and hidden states are shared for computing all aggregated local features. The memory cell and the hidden state at time t are computed as follows:

$$\overline{c}^{(t)} = \sum_{i=1}^{7} \sum_{j=1}^{7} \alpha_{ij}^{(t)} c_{ij}^{(t)} \quad (113)$$

$$\overline{h}^{(t)} = \sum_{i=1}^{7} \sum_{j=1}^{7} \alpha_{ij}^{(t)} h_{ij}^{(t)} \quad (114)$$

where $\alpha_{ij}^{(t)}$ are local weights. In our work, we experimented with two types of local weights. We first tried to use uniform weights:

$$\alpha_{ij}^{(t)} = \frac{1}{7 \times 7} \quad (115)$$

which actually correspond to an average pooling of aggregated local features. The second solution that we tried was to derive local weights using an attention mechanism, as follows:

$$\tilde{\alpha}_{ij}^{(t)} = w^{T} \tanh(W_{\alpha x} x_{ij}^{(t)} + W_{\alpha h} \overline{h}^{(t-1)} + b_{\alpha}). \quad (116)$$

$$\alpha_{ij}^{(t)} = \frac{\exp(\tilde{\alpha}_{ij}^{(t)})}{\sum_{k=1}^{7} \sum_{l=1}^{7} \exp(\tilde{\alpha}_{kl}^{(t)})}, \quad (117)$$

where $W_{\alpha x}$, $W_{\alpha h}$, b_{α} are trainable parameters.

4.2.1.2 *Encoding Videos*

In our model, we encode videos using the L-STAP method we presented previously. We initialized the memory cell and the hidden state of the LSTM using the output of an I3D [11] (before the final softmax) which had been trained on Kinetics-600 [11]. More formally, if V is an input video:

$$c_{ij}^{(0)} = \tanh(W_c^e e(V) + b_c^e) \tag{118}$$

$$h_{ij}^{(0)} = \tanh(W_h^e e(V) + b_h^e) \tag{119}$$

where W_c^e, b_c^e, W_h^e and b_h^e are trainable parameters. The decoder produces $\bar{c}^{(T)}$ and $\bar{h}^{(T)}$ as outputs, where T is the length of the input video. These outputs will be used to initialize the sentence decoder that we will introduce in the next section.

4.2.1.3 *Decoding Sentences*

For decoding sentences, we chose to use an LSTM. In the following, we assume that sentences Y are represented by sequences of one-hot vectors $y_1, ..., y_L \in \mathbb{R}^N$ where N is the vocabulary size. The aim of the LSTM is to compute the probabilities $P(y_l|y_{l-1}, ..., y_1, V; \theta)$ for $l \in \{1, ..., L\}$, where θ is the set of all parameters in the encoder and the decoder, and V an input video. In the following, we will describe formally how we compute these probabilities.

We initialize the memory cell and the hidden state of the decoder LSTM using the last memory cell and the last hidden state of the encoder:

$$c_0^d = \bar{c}^{(T)}, \tag{120}$$

$$h_0^d = \bar{h}^{(T)}. \tag{121}$$

It has been shown in [60] for text translation tasks that attending to hidden states of the encoder during the decoding phase improved results. Some works in video captioning have followed that approach successfully [100, 101]. We followed a similar approach for our decoding phase. More precisely, at each step l, we compute a weighted sum of hidden states of the encoder:

$$\varphi(\bar{h}, h_{l-1}^d) = \sum_{t=1}^{T} \beta_l^{(t)} \bar{h}^{(t)} \tag{122}$$

where $\beta_l^{(1)}$, ..., $\beta_l^{(T)}$ are computed as follows:

$$\tilde{\beta}_l^{(t)} = w_\beta^T \tanh(W_{\beta e} \bar{h}^{(t)} + W_{\beta h} h_{l-1}^d + b_\beta), \tag{123}$$

$$\beta_l^{(t)} = \frac{\exp(\tilde{\beta}_l^{(t)})}{\sum_{k=1}^{L} \exp(\tilde{\beta}_k^{(t)})}, \tag{124}$$

where $W_{\beta e}$, $W_{\beta h}$, b_β are trainable parameters. Assuming that the word y_{l-1} has been decoded at step $l-1$, we aim to decode y_l based on y_{l-1} and $\varphi(\overline{h}, h^d_{l-1})$. For that purpose, we first compute a word embedding x^d_l:

$$w^d_l = W_{\text{emb}} y_{l-1}, \tag{125}$$

where W_{emb} is a learned embedding matrix. Then, we concatenate w^d_l and $\varphi(\overline{h}, h^d_{l-1})$ to obtain and x^d_l:

$$x^d_l = [w^d_l; \varphi(\overline{h}, h^d_{l-1})] \tag{126}$$

Eventually, we input x^d_l to the decoder LSTM:

$$i^d_l = \sigma(W^d_{ix} x^d_l + W^d_{ih} h^d_{l-1} + b^d_i) \tag{127}$$

$$f^d_l = \sigma(W^d_{fx} x^d_l + W^d_{fh} h^d_{l-1} + b^d_f) \tag{128}$$

$$o^d_l = \sigma(W^d_{ox} x^d_l + W^d_{oh} h^d_{l-1} + b^d_o) \tag{129}$$

$$c^d_l = f^d_l \circ c^d_{l-1} + i^d_l \tanh(W^d_{cx} x^d_l + W^d_{ch} h^d_{l-1} + b^d_c) \tag{130}$$

$$h^d_l = o^d_l \circ \tanh(c^d_l) \tag{131}$$

where W^d_{ix}, W^d_{ih}, b^d_i, W^d_{fx}, W^d_{fh}, b^d_f, W^d_{ox}, W^d_{oh}, b^d_o, W^d_{cx}, W^d_{ch} and b^d_c are trainable parameters.

The last step is to infer a word y_l. For that purpose, we derive $\tilde{y}_l$ as follows:

$$\tilde{y}_l = \text{softmax}(W_d h^d_l) \tag{132}$$

where W_d is a trainable parameter. We state that y_l is the one-hot vector corresponding to the maximum coordinate of $\tilde{y}_l$.

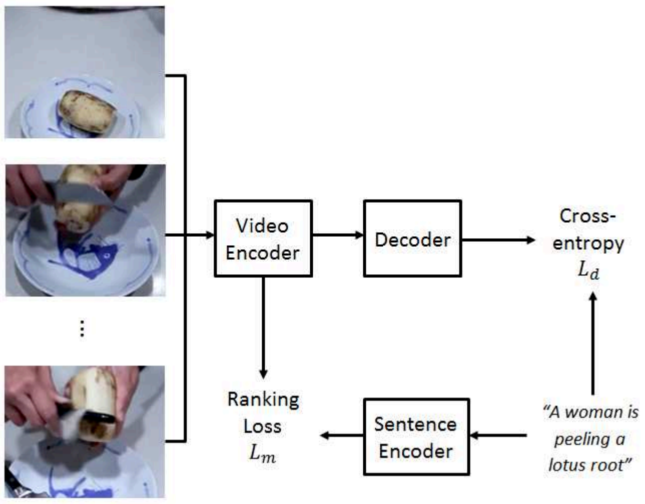


Figure 47: Overview of our training losses. The first training loss is the Cross-Entropy loss, which aims to make the probability distribution of sentences in the training set and the probability distribution of the inferred sentences match. The second one is a ranking loss, aiming to bridge the semantic gap between video representations and sentences.

4.2.2 *Training*

Assuming that $y_1, ..., y_L$ correspond to ground-truth words, we aim to minimize the following cross-entropy loss:

$$\mathcal{L}_d(\theta) = -\sum_{l=1}^{L} \log P(\tilde{y}_l | y_{l-1}, ..., y_1, V; \theta) \qquad (133)$$

where V is a video corresponding to the caption $(y_1, ..., y_L)$. More information is given on the cross-entropy loss in Section 2.3.

In addition to that, some works have shown that regularizing the cross-entropy loss with a matching loss between video encodings and ground-truth sentences could improve results by bridging the semantic gap between them [33, 59]. As reported in Section 4.2.2, such improvement has been noticed in our experiments. The matching model we employed is described in the following. Let us assume that $Y = (y_1, ..., y_L)$ is a sentence corresponding to a video V. First, we translate this sequence of one-hot vectors into a sequence of word embeddings $(x_1^s, ..., x_L^s)$ using the matrix W_{emb} from Section 4.2.1.3. Then, we compute a sentence embedding $\psi(Y)$ by processing this sequence of word embeddings into another LSTM: each word embedding is entered sequentially as an input to that LSTM, and $\psi(Y)$ is defined to be its last hidden state. We want the initialization of the decoder to be as close as possible to an accurate representation of its corresponding sentence. Therefore, if $\varphi(V) = \overline{h}^{(T)}$ is the initial hidden state of the decoder, we will aim to minimize the following ranking loss from [25]:

$$\begin{aligned}\mathcal{L}_m(\theta) = &\max_{\overline{V} \neq V} \left(\max(0, \alpha - S(\varphi(V), \psi(Y)) + S(\varphi(\overline{V}), \psi(Y)))\right) \\ &+ \max_{\overline{Y} \neq Y} \left(\max(0, \alpha - S(\varphi(V), \psi(Y)) + S(\varphi(V), \psi(\overline{Y})))\right)\end{aligned} \tag{134}$$

where $\overline{V}$ is a negative video sample, and $\overline{Y}$ is a negative sentence sample coming from another video than V. More information on ranking losses is given in Section 2.2.1 The final loss is the following:

$$\mathcal{L}(\theta) = \mathcal{L}_d(\theta) + \lambda \mathcal{L}_m(\theta) \tag{135}$$

where λ is a hyperparameter that we set to 0.4 according to results on validation.

4.2.3 *Experiments*

In this section, we will deal with our experimental results.

4.2.3.1 *Datasets*

We evaluated our models on two video captioning datasets: MSVD [12] and MSR-VTT [98]. MSVD is a dataset composed of 1,970 videos from YouTube, which have been annotated by Amazon Mechanical Turks (AMT). Each video has approximately 40 captions in English. We split that dataset following [91]: 1,200 videos for training, 100 videos for validation and 670 videos for testing. MSR-VTT is a similar dataset, but with much more videos, and less captions per video. It is composed of 10,000 videos, and 20 captions per video. Following [98], we split that dataset into 6,513 videos for training, 497 videos for validation and 2,990 videos for testing.

For both datasets, we uniformly sampled 30 frames per video as done in [104], and extracted features for each frame based on the last convolutional layer of a ResNet-152 [35], which had been trained on the image-text matching task on MSCOCO [58], after pre-training on ImageNet-1000 [16] following [25]. In addition, we extracted activity features for each video using an I3D pretrained on Kinetics-600 [11]. For MSVD, we converted sentences to lowercase and removed special characters, which lead to a vocabulary of about 14k words. We converted each word into an integer, and cut sentences after the thirtieth word if their lengths were higher than thirty. The same approach for MSR-VTT lead to a much bigger vocabulary size of about 29k words. Therefore, we kept only the 15k most common words, and replaced all the others by an <UNK> token. We applied the same process otherwise.

4.2.3.2 *Implementation Details*

Our models have been implemented with the TensorFlow framework [1]. We use 1024-dimensional LSTMs in both encoder and decoder. Soft attention spaces are 256-dimensional. Word embeddings are 300-dimensional.

We trained our model using the RMSProp algorithm [87], with decay = 0.9, momentum = 0.0 and epsilon = 1e-10. Batch size is set to 64. Learning rate is 1e-4, and we apply gradient clipping to a threshold of 5. Eventually, we apply dropout on the output of the decoder (before the prediction layer) with a rate of 0.5 to avoid overfitting.

4.2.3.3 *Results on MSVD and MSR-VTT*

Model	Bleu-4	ROUGE	METEOR	CIDEr
TSL [97]	51.7	-	34.0	74.9
RecNet [93]	52.3	69.8	34.1	80.3
mGRU [71]	53.8	-	34.5	81.2
AGHA [104]	**55.1**	-	35.3	83.3
SAM [94]	54.0	-	35.3	87.4
E2E* [55]	50.3	70.8	34.1	87.5
SibNet [59]	54.2	71.7	34.8	**88.2**
L-STAP (Ours)	**55.1**	**72.7**	**35.4**	86.7

Table 15: Results on the MSVD dataset. The * sign means that the model is using reinforcement learning techniques to optimize over the CIDEr metric. Best results are in bold characters.

We evaluated our models in terms of BLEU [70], ROUGE [57], METEOR [17] and CIDEr [89] scores, which are metrics commonly used to evaluate automated captioning tasks. We compared them to the following recent models for video captioning. Our results on MSVD are presented in Table 15. Results on MSR-VTT are presented in Table 16.

On MSVD, it can be noticed that L-STAP achieves the best results on three out of four metrics. It is also relevant to mention that E2E [55], which achieves better CIDEr results than our model, has been trained using reinforcement learning techniques to be optimized regarding that CIDEr metric. Works on image captioning and video captioning have shown that significant improvements could be done using such techniques [3, 76, 96], at the price of much longer training times. We did not use reinforcement learning to train our models, instead we use cross-entropy minimization which has the advantage of being fast and simpler to implement.

Results on MSR-VTT show that our model outperforms models trained using a cross-entropy loss on two metrics out of four (ME-

TEOR and ROUGE). HRL [96] obtains better results overall, however it makes use of reinforcement learning techniques, which leads to better results as stated in the previous paragraph.

We report some qualitative results of our model on MSR-VTT in Figure 48. On the second video, the man who is singing appears during a very limited amount of time. This shows that our model has been able to attend to important frames to identify what the main action of the video was. In the first video, a woman starts talking about makeup, and then puts some lipstick on her lips. The caption generated by our model shows that it has been able to draw a relation between the first and the second parts of the video. Moreover, the lipstick is applied on a very localized part of the video frames: we can infer that our model could efficiently attend to the right part of the frame to generate a caption. The fourth video shows that results could be improved by adding sound processing to our model: it was not possible from the video only to know that colors were said.

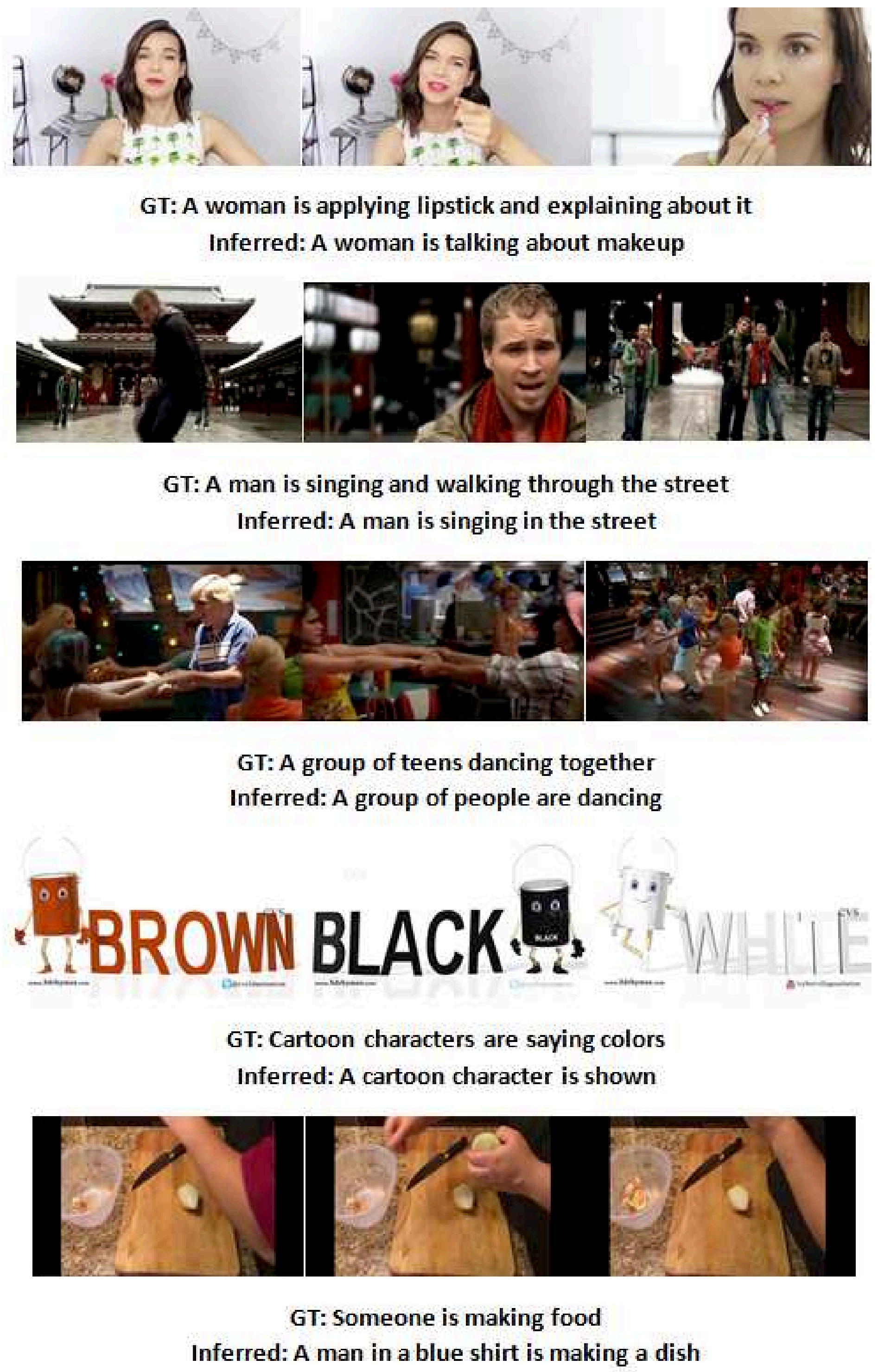
Model	Bleu-4	ROUGE	METEOR	CIDEr
RecNet [93]	39.1	59.3	26.6	42.7
E2E* [55]	40.1	61.0	27.0	48.3
SibNet [59]	40.9	60.2	27.5	47.5
HRL* [96]	**41.3**	**61.7**	**28.7**	**48.0**
L-STAP (Ours)	40.7	61.2	27.6	44.8

Table 16: Results on the MSR-VTT dataset. The * sign means that the model is using reinforcement learning techniques to optimize over the CIDEr metric. Best results are in bold characters.

4.2.3.4 *Ablation Study*

Results of an ablation study on the MSVD dataset are reported in Table 17. The encoder we used in our baseline model is an Long-term Recurrent Convolutional Network (LRCN) [19]. As shown in previous works such as [33, 59], adding a component to the training loss to make video representations match sentence representations improves results. Two interpretations can be given to these results. A first one is that adding a ranking loss to match video representations and sentence representations helps bridging the semantic gap between these two modalities. A second one could be that propagating the gradient across all the layers of the decoder could make it vanish through depth. Thus, adding a matching loss to the cross-entropy loss could be seen as a skip-connection between the sentence to be generated and the video representation used by the decoder. We illustrate that second interpretation in Figure 49.

Replacing the average pooling at the end of a CNN by our L-STAP induces a major improvement with respect to all metrics as reported



GT: A woman is applying lipstick and explaining about it
Inferred: A woman is talking about makeup

GT: A man is singing and walking through the street
Inferred: A man is singing in the street

GT: A group of teens dancing together
Inferred: A group of people are dancing

GT: Cartoon characters are saying colors
Inferred: A cartoon character is shown

GT: Someone is making food
Inferred: A man in a blue shirt is making a dish

Figure 48: Some qualitative results of L-STAP on MSR-VTT.

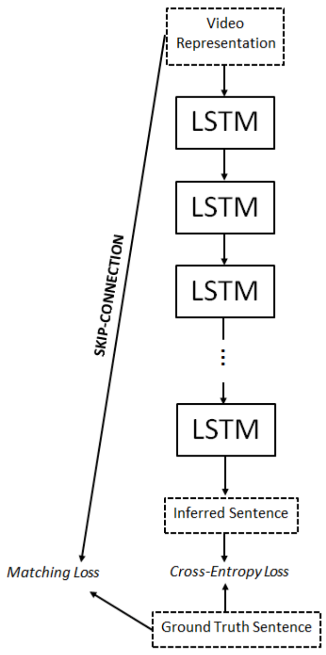


Figure 49: Our second interpretation about the efficiency of the second term of our loss function. Skip connections between video representations and ground-truth sentences improve results.

in Table 17. On top of that, results shown in Table 15 demonstrate that L-STAP leads to better results than other models based on local features such as AGHA and SAM, and results shown in both Table 15 and Table 16 show the interest of L-STAP over average pooling.

We can notice in Table 17 that using a soft-attention mechanism to pool local hidden states in the encoder does not provide significant improvements over average pooling for all metrics except from CIDEr. Our interpretation is that the LSTM of the encoder can learn to attend to relevant local features by itself: before applying the average pooling, attention has already been drawn quite efficiently.

4.2.4 *Conclusion of Section 4.2*

In [27], we presented a novel Learned Spatio-Temporal Adaptive Pooling (L-STAP) method for video captioning. It consists in taking into account spatial and temporal information jointly in a video to produce good video representations. As we have shown, these video representations can be successfully used to perform automated video captioning. We demonstrated the quality of our models based on L-STAP by comparing them with state-of-the-art models on MSVD and MSR-VTT, which are two video captioning datasets. On top of that, we assessed the interest of L-STAP through an ablation study. Although this presentation concentrates on video captioning we be-

Model	Bleu-4	ROUGE	METEOR	CIDEr
Baseline	52.7	71.4	34.1	79.5
Baseline + matching	53.3	71.2	34.5	82.2
L-STAP (avg) + matching	55.1	72.3	35.4	84.3
L-STAP (att) + matching	55.1	72.7	35.4	86.8

Table 17: Results of ablation study on MSVD. Results show that a significant improvement can be reached using our Learned Spatio-Temporal Adaptive Pooling instead of the usual average pooling. Pooling hidden states of the encoder using soft-attention (line 4) instead of average pooling (line 3) does not always improve results. Our interpretation of that outcome is that the LSTM actually performs a kind of attention on local features before local hidden states are pooled together.

lieve that the proposed L-STAP method could be also applied to other video-related tasks such as video classification.

4.3 CURRICULUM LEARNING FOR CAPTIONING

In [9], Bengio *et al.* proposed a training method called Curriculum Learning. Instead of training models based on randomly drawn mini-batches from a training set, their method incrementally widens the training set, starting from easiest samples, and adding some harder ones after each epoch. They showed that training could get faster, and sometimes lead to better results. In this section, we propose two Curriculum Learning methods for captioning. The first one is a simple adaptation of the algorithm that has been introduced in [9]: we propose to evaluate the complexity of training samples based on the self-BLEU [108] metric. The second one is an Adaptive Curriculum Learning algorithm: instead of using a predefined curriculum, we propose an algorithm that adapts the content of the training set to the actual performances of our model. We apply our method to image captioning, but it can also be straightforwardly adapted to video captioning.

Our contributions in this section are two-fold:

- we show in the context of curriculum learning that the self-BLEU metric that has initially been proposed to evaluate diversity can also be used to assess the complexity of captioned images;
- we propose an Adaptive Curriculum Learning algorithm and we show experimentally that it performs better than the usual mini-batch gradient descent algorithm.

The rest of this section is organized as follows: in Section 4.3.1, we define the image captioning model we use in our experiments. In Sec-

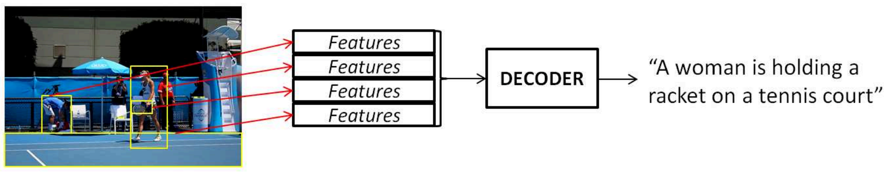


Figure 50: The image captioning model we employed. A Faster-RCNN derives features vectors corresponding to object detection boxes from an input image. These features vectors are then used by a decoder through a soft-attention mechanism to produce a caption for the input image.

tion 4.3.2, we explain how self-BLEU can assess the complexity of a captioned image. In Section 4.3.3, we present our novel Adaptive Curriculum Learning algorithm. We show and discuss our experimental results in Section 4.3.4, and we conclude in Section 4.3.5.

4.3.1 *Image Captioning with Attention*

The model that we use for image captioning is the one on Figure 50. A Faster-RCNN [75] pretrained on VisualGenome [49] is used to extract detection boxes and corresponding features vectors. These features vectors are then used as the input of a decoder GRU *via* an attention mechanism to guide the generation of a caption. More formally, let $(v_1, ..., v_n)$ be n object detections features vectors obtained by a Faster-RCNN on a given image. Let w_t be the t-th decoded word. Then, the input x_{t+1} of the decoder GRU for decoding the $t+1$-th word is as follows:

$$x_{t+1} = \sum_{i=1}^{n} \alpha_{i,t+1} v_i, \tag{136}$$

where the $\alpha_{i,t+1}$ are non-negative weights. These weights are derived through a soft-attention mechanism:

$$(\alpha_{1,t+1}, ... \alpha_{n,t+1}) = \text{softmax}(r_{1,t+1}, ..., r_{n,t+1}), \tag{137}$$

where for each $i \in \{1, ..., n\}$, $r_{i,t+1}$ is derived as:

$$r_{i,t+1} = \langle W_v v_i | W_w x_t \rangle\,, \tag{138}$$

where W_v and W_w are trainable parameters. The output of the decoder GRU is then converted into a word through an affine transform, as we did in Section 4.2.

Let us now describe the two Curriculum Learning methods that we propose.

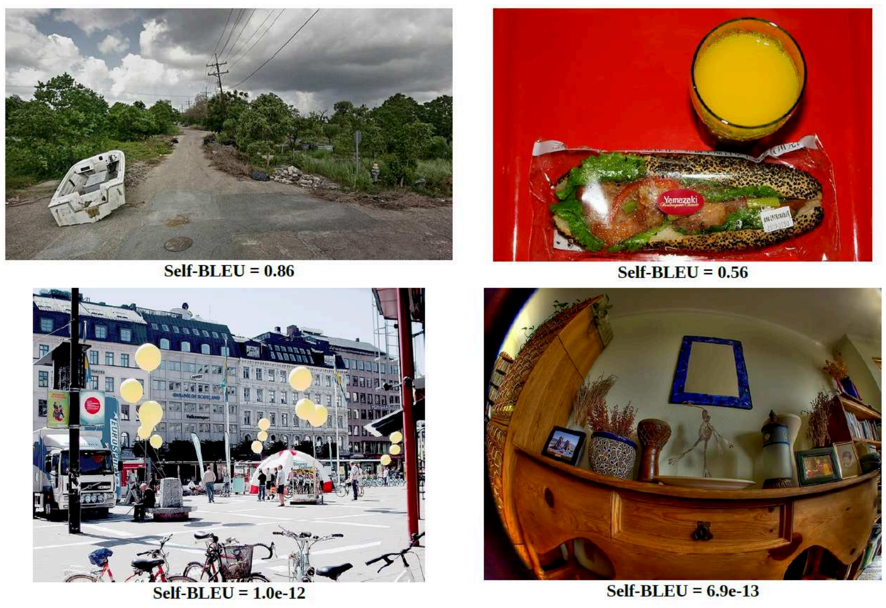


Figure 51: Examples of images taken from the MSCOCO dataset and their corresponding self-BLEU scores. Images that contain few elements and that are easily described have a high score, whereas complex images have a low score.

4.3.2 *Curriculum Learning with Self-BLEU*

BLEU [70] has initially been proposed as a metric to assess the quality of automatic translations. It has been also widely used for evaluating captioning models. In [108], Zhu *et al.* propose self-BLEU, a method to evaluate the diversity of generated text data. It consists in averaging the BLEU scores of all generated texts with respect to the others. More formally, let $S = \{s_1, ..., s_p\}$ a corpus of generated texts, and let $\text{BLEU}_A(s)$ be the BLEU score of s with respect to the set A. Then, the self-BLEU score of S is derived as follows:

$$\text{self-BLEU}(S) = \frac{1}{p}\sum_{i=1}^{p} \text{BLEU}_{S\setminus\{s_i\}}(s_i). \tag{139}$$

As shown on Figure 51, this self-BLEU metric can also be used to measure the complexity of visual contents that have been captioned by human beings. If there is a big diversity in captions for a given image, it means that describing this image is not a straightforward task. Conversely, if the self-BLEU score is high, it means that independent humans use approximately the same sentences to describe it: the content is easily understandable.

We propose then an algorithm to train an image captioning model with Curriculum Learning based on the self-BLEU score. The principle is simple: training starts with a subset of easy samples from the training set (with a high self-BLEU score), and after each epoch, we add a fixed number of samples of increasing difficulty from the training set to the training subset we mentioned. That training process is described in the following algorithm. In that algorithm, we define a

Algorithm 1 Curriculum Learning with self-BLEU (α is a hyperparameter)

```
Ensure: self-BLEU($s_i$) $\geqslant$ self-BLEU($s_{i+1}$)    $\forall i \in \{1, ..., n-1\}$
  train_set $\leftarrow \{s_1, ..., s_n\}$
  $k \leftarrow k_0$
  for epoch $\in \{1, ..., \#\text{epochs}\}$ do
    train_subset $\leftarrow \{s_1, ..., s_k\}$
    $k \leftarrow \min(k + \alpha, n)$
    Train model on train_subset
  end for
```

hyperparameter α corresponding to how much we increase the size of the training subset after each epoch.

4.3.3 *Adaptive Curriculum Learning*

The idea of Curriculum Learning comes from an analogy between a student and an artificial intelligence: in both cases, they are taught thanks to data that they are provided with in an increasing order of difficulty. However, even though [9] has showed that training could be made significantly faster with Curriculum Learning, it is not clear that final results of models trained with that method obtain systematically better results than with a simple mini-batch gradient descent. We think that the reason is that after all, a model ends by being provided with all training data, and therefore it is always optimized to fit the same data distribution.

How to avoid that issue? We think that despite these problems, the idea of Curriculum Learning should be further investigated: using a fixed curriculum is only one possible way of providing a model with training data. In this section, we propose an Adaptive Curriculum Learning algorithm, that adapts a curriculum to the model that is training. The idea behind our algorithm is simple. If our model is doing well on some data, that data should not be often shown to it. However, if its results on that data is bad, then it should be provided more often to it.

An analogy between our Adaptive Curriculum Learning algorithm and flashcard-based spaced repetition [8] could be made: it has been shown that humans could learn efficiently and in the long term if they were provided with difficult data frequently, and less frequently with easy data. For a neural network model, we made the assumption that easy data correspond to frequent data in the training set, and difficult data correspond to rare data: neural networks perform better on data containing features they are often provided with because they are optimized to fit their training data. Applying that principle to neural networks training using our Adaptive Curriculum Learning algorithm can be seen as compensating imbalance in training data.

More formally, at each iteration t, we draw randomly a mini-batch from the training set. In the following, the expression $\text{CIDEr}_A(s)$ denotes the CIDEr score [89] of a sentence s based on a reference set A, and $\text{model}(I)$ denotes the sentence generated by our captioning model for an image I. Each sample $x_i = (I_i, s_i, S_i)$ from that mini-batch (I_i is an image, s_i is a corresponding sentence randomly drawn from the set S_i of annotations) is then assigned a score a_i, defined as follows:

$$a_i = \text{CIDEr}_{S_i \setminus s_i}(s_i) - \text{CIDEr}_{S_i \setminus s_i}(\text{model}(I_i)). \tag{140}$$

That score a_i if then used to compute the probability p_i that the corresponding sample is actually used for training:

$$p_i = \min\left(\exp\left(\frac{a_i - \mu}{\sigma}\right), 1\right), \tag{141}$$

where μ and σ are two hyperparameters. The more our model is doing well on a given sample, the more that sample has a chance to be dismissed during training. The algorithm, that we called the Flashcard Algorithm accordingly to the analogy we made before, is described below (p is the size of mini-batches).

Algorithm 2 The Flashcard algorithm: an Adaptive Curriculum Learning algorithm (μ and σ are hyperparameters)

```
train_set ← $\{s_1, ..., s_n\}$
**for** epoch ∈ $\{1, ..., \#\text{epochs}\}$ **do**
  **while** Training data is remaining **do**
    minibatch ← $\{(I_1, s_1, S_1), ..., (I_p, s_p, S_p)\}$
    **for** $i \in \{1, ..., p\}$ **do**
      $a_i = \text{CIDEr}_{S_i \setminus s_i}(s_i) - \text{CIDEr}_{S_i \setminus s_i}(\text{model}(I_i))$
      $p_i = \min\left(\exp\left(\frac{a_i-\mu}{\sigma}\right), 1\right)$
      minibatch ← minibatch $\setminus (I_i, s_i, S_i)$ with probability $1 - p_i$
    **end for**
    Train model on resulting minibatch
  **end while**
**end for**
```

Let us now describe how we trained our image captioning models.

4.3.3.1 *Optimization*

Assuming that $y_1, ..., y_L$ correspond to ground-truth words for a given image, we aim to minimize the following cross-entropy loss:

$$\mathcal{L}(\theta) = -\sum_{l=1}^{L} \log P(\tilde{y}_l | y_{l-1}, ..., y_1, I; \theta) \tag{142}$$

where I is an image corresponding to the caption $(y_1, ..., y_L)$ and $\tilde{y}_l$ is the output of the decoder at iteration l. More information is given on the cross-entropy loss in Section 2.3.

We use the Curriculum Learning algorithms that we defined previously to guide training. In Section 2.3, we also stated that an alternative to the usual cross-entropy loss to train captioning models was using reinforcement learning. Reinforcement learning has shown to lead to very significant improvements in image or video captioning. We wondered if using one of our Curriculum Learning methods could compensate the drawbacks of the cross-entropy loss that the reinforcement learning loss do not have. Therefore, we also train our image captioning model using reinforcement learning with CIDEr reward. The reinforcement learning loss to minimize is defined as follows:

$$\mathcal{L}_{RL}(\theta) = -r(s), \quad s \sim p_\theta \tag{143}$$

where $r(s)$ is the reward for a sentence s. As neural networks are trained by gradient descent, that loss needs to be adapted to be differentiable with respect to θ. As we showed in Section 2.3, minimizing the reinforcement learning loss was equivalent to minimizing the following loss:

$$\begin{aligned} \mathcal{L}'_{RL}(\theta) &= -r(s)\log(p_\theta(s)), \quad s \sim p_\theta \\ &= -r(s)\textstyle\sum_{l=1}^{L} \log\left(p_\theta\left(y_l | y_1, ..., x_{l-1}, I; \theta\right)\right). \end{aligned} \tag{144}$$

4.3.4 *Experiments*

In this section, we give our experimental results. First, we explain how we implemented our models, then we describe the dataset on which we train them, and eventually we give our results and discuss them.

4.3.4.1 *Implementation Details*

All our models have been implemented using the Tensorflow framework for Python [1]. We use 1024-dimensional LSTMs in both encoder and decoder. Soft attention spaces are 256-dimensional. Word embeddings are 300-dimensional.

We trained our model using the RMSProp algorithm [87], with decay = 0.9, momentum = 0.0 and epsilon = 1e-10. Batch size is set to 64. Learning rate is 8e-5, and we apply gradient clipping to a threshold of 5. Neither dropout nor batch normalization have been applied. Regarding the hyperparameters of the Adaptive Curriculum Learning algorithm, we found by cross-validation that $\mu = -0.6$ and $\sigma = 1.0$ lead to the best results.

4.3.4.2 *Dataset*

We evaluated how our models performed on the caption retrieval task on the MSCOCO dataset [58]. This dataset contains 123,000 images with 5 captions each, and we split it into a training set, a validation set and a test set according to [43]. The training set contains 113000 images, the validation set contains 5000 images and the test set contains 5000 images.

As for data preprocessing, we converted all sentences to lowercase and removed special characters (apart from spaces and hyphens). We limited the vocabulary to 10,000 most used words, and replaced all other words by an "<UNK>" token. Regarding images, we utilized the ResNet-101 features of object detections that have been released by [3]. These object detections have been derived using a Faster-RCNN which has been trained on the VisualGenome dataset. For each image, we utilized a fixed amount of 36 object detections.

4.3.4.3 *Results and Discussion*

We reported the experimental results we obtained on MSCOCO in Table 18. As one can notice, the self-BLEU-based Curriculum Learning algorithm leads to nearly the same results as the baseline. We think that the reason why results do not improve is that eventually, with or without a simple Curriculum Learning algorithm, models are trained on the same data distributions: therefore we can intuitively deduce that they obtain on average nearly the same results.

The Adaptive Curriculum Learning algorithm leads to an improvement in terms of all metrics with respect to the baseline. The main difference between the usual Curriculum Learning algorithm based on self-BLEU and our Adaptive Curriculum Learning algorithm is that the latter does not always lead to training a model on the same distribution of training data, but on a more balanced distribution as it adapts the probability to deliver training data to the actual results of the model.

However, results using a Reinforcement Learning loss are still above our Adaptive Curriculum Learning algorithm. But both methods are not exclusive, and there is no reason why the Reinforcement Learning method cannot be combined with our Adaptive Curriculum Learning algorithm. We think that an interesting direction of research would be to investigate to what extent Adaptive Curriculum Learning can improve results of Reinforcement Learning in captioning.

We also plotted the validation curves of the baseline model trained with a simple mini-batch gradient descent, and using the self-BLEU-based Curriculum Learning algorithm. That plot has been reported on Figure 52. We can see that training is slightly faster using Curriculum Learning, accordingly to the findings of [9]. However, both curves end with similar validation scores. An interesting direction of

Model	BLEU-4	ROUGE	METEOR	CIDEr
Baseline	30.9	53.8	25.2	98.5
Baseline + Self-BLEU	31.1	53.7	25.0	97.3
Baseline + Adaptive	31.1	54.1	25.7	100.4
Baseline + Reinforcement	32.2	54.8	25.8	108.5

Table 18: Our results on the MSCOCO dataset. As one can notice, the self-BLEU-based Curriculum Learning algorithm leads to nearly the same results as the baseline. To the contrary, the Adaptive Curriculum Learning algorithm improves the results of the baseline in terms of all metrics. However, it does not induce as much improvement as a Reinforcement Learning loss induces.

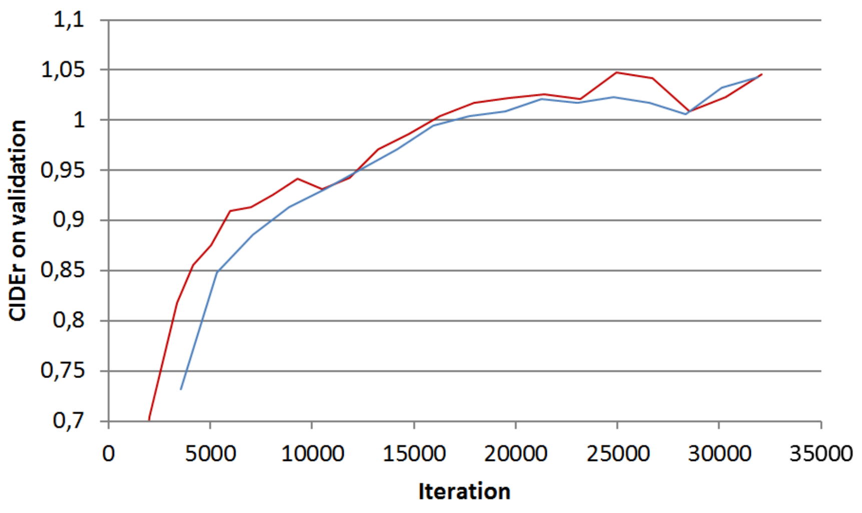


Figure 52: Plot of validation scores of the baseline trained with simple mini-batch gradient descent (in blue) and using self-BLEU-based Curriculum Learning (in red). We can notice that Curriculum Learning make training a bit faster, even though the best validation loss is eventually nearly the same for both methods.

research would be to define a metric making the training of a captioning model with Curriculum Learning even faster.

4.3.5 *Conclusion*

In this section, we introduced our results in image captioning using two methods of Curriculum Learning we designed: the first one is similar to [9] using the self-BLEU metric to measure the complexity of annotated images, whereas the second one is an Adaptive Curriculum Learning method, that adapts the distribution of training data to the actual performances of the captioning model during training. We experimented with these methods on the MSCOCO dataset for image captioning, but our methods can be easily extended also to video captioning.

We found that, even though results on test data for the self-BLEU-based Curriculum Learning algorithm were similar to the usual mini-batch gradient descent, training could be made faster, accordingly to the findings of [9]. We also found that our Adaptive Curriculum Learning algorithm lead to an improvement in terms of all the metrics we used with respect to the mini-batch gradient descent baseline. However, this improvement is not as big as the improvement induced by a reinforcement learning loss.

We think that two interesting directions of research would be to try to find a complexity metric making the usual Curriculum Learning algorithm even faster for captioning, and to find a way to combine our Adaptive Curriculum Learning algorithm with reinforcement learning to further improve results.

4.4 GENERAL CONCLUSION OF CHAPTER 4

In this chapter, we introduced models we developed for image and video captioning. In particular, we proposed the Learned Spatio Temporal Adaptive Pooling (L-STAP) method for video captioning, that can be used to replace the final global pooling in a CNN when used to process videos. With that pooling method, one can derive local embeddings where temporality is taken into account. Our video captioning model based on the L-STAP method outperforms state-of-the-art video captioning models according to our experiments. An ablation study has been able to show to what extent each component of our method contributes to improving results. We think that the interest of our L-STAP method is not restricted to video captioning: the same method can be applied to any other task related to video processing.

We also proposed two Curriculum Learning algorithms. The first one is based on the self-BLEU score, that we used to measure the complexity of an annotated image: if the self-BLEU score of such an annotated image is high, it means that annotators agreed a lot on the captions they proposed. In that case, we can infer that describing the image was straightforward: the complexity of the image is low. To the contrary, if the self-BLEU score is low, it means that annotators did not agree on the captions, which allows us to say that the image was complex. Using the self-BLEU score with a usual Curriculum Learning algorithm improved the speed of training, but not the final result. We proposed an explanation to that: with the usual Curriculum Learning algorithm, models are eventually trained to fit the same distribution of data as models trained without Curriculum Learning. Therefore, to tackle that issue, we proposed a novel Adaptive Curriculum Learning algorithm that we called the Flashcard algorithm. It consists in showing data to the model with high probability when it does not perform well on it, and with low probability if it does. With that Flashcard algorithm, results improved with respect to a simple mini-batch gradient descent. However, we also showed that it did not improve results as much as Reinforcement Learning did. An interesting direction of research would be to find how to combine our Adaptive Curriculum Learning algorithm with Reinforcement Learning to further improve results in captioning.

5 GENERAL CONCLUSION

Our research work was the occasion to propose novel models and novel methods for the creation of semantic representations of images or videos: multimodal vector representations that are useful in the context of vision-text matching, and also textual representations in natural language that are automatically generated. In particular, we would like to emphasize the following points:

- The models that we proposed in this thesis work have been compared to models usually employed for the same tasks. We assessed that our models lead to better results than usual models through experiments on standard datasets evaluated using standard metrics.
- Moreover, the models that we proposed have been applied to specific tasks, but we think that they can also be applied in a more general context.
- Extensive experiments have been conducted throughout this work to assess the interest of each element of our models and methods.

The models and methods we have proposed have been applied in the context of this thesis research work; nevertheless we think that their interest could go beyond our research topic because they are conceptually generalizable to other applications, in particular related to natural language processing regarding our Recurrent Capsule Networks and our Gated Recurrent Capsules, or related to video processing regarding our L-STAP method. Our work could for instance be extended to tasks that we did not have the opportunity to deal with in that thesis work, such as Dense Captioning, or Visual Question Answering.

We think that the main idea to be remembered from our work is that, as says [88], "Attention Is All You Need". The models we proposed throughout this thesis are actually based on attention mechanisms, emphasizing relevant data and obliterating irrelevant data. Also, the Curriculum Learning algorithms that we proposed can be seen as attention in a certain way: the right data is attended to during mini-batch selection, and given to the model we train. We started this manuscript with a citation of John Locke: "Reading furnishes the mind only with materials of knowledge; it is thinking that makes what we read ours." This is exactly the message we would like to convey: big datasets have given birth to the current AI trend, making it grow up will need that models "think" and "understand" these datasets.

That leads us to the future perspectives of our work. We showed in Section 3.4 with our Gated Recurrent Capsules that focusing on sub-parts of a sentence was a way to improve results in image-sentence matching. We also showed in Section 4.2 that our L-STAP method, based on local features, was more efficient than using global features. Locality in both text and vision improved our results: would it be also beneficial to study how sub-parts of sentences and local features of images or videos match? We think that this perspective should be further investigated.

We showed in Chapter 3 that even though multimodal models were efficient for matching vision and text, they lost their efficiency in tasks such as Ad-Hoc Video Search (AVS), and concept detectors seemed to be more efficient. We think that it is worth focusing on what kind of information should preferably be extracted from vision-text data to address the AVS task.

We have showed in Section 4.3 how our Adaptive Curriculum Learning algorithm was able to improve image captioning. However, the improvement induced by Reinforcement Learning is bigger. How could we combine both Adaptive Curriculum Learning and Reinforcement Learning?

The matching term of the loss function of the model we implemented and tested, based on out L-STAP method induced an improvement of our results. What are the links between vision-text matching and captioning? How solving one task can provide information to help solving the second one?

As a conclusion and a general summary, we showed that attention was a key to success to derive semantic representations for images and videos. We think that a right direction for pursuing our research would be to focus on attention models on one hand, and on analyzing how and what training data is fed to models during optimization on the other hand.

BIBLIOGRAPHY

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "Tensorflow: A system for large-scale machine learning." In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 265–283.

[2] Aishwarya Agrawal, Jiasen Lu, Stanislaw Antol, Margaret Mitchell, C Lawrence Zitnick, Devi Parikh, and Dhruv Batra. "Vqa: Visual question answering." In: *International Journal of Computer Vision* 123.1 (2017), pp. 4–31.

[3] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. "Bottom-up and top-down attention for image captioning and visual question answering." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 6077–6086.

[4] Lisa Anne Hendricks, Oliver Wang, Eli Shechtman, Josef Sivic, Trevor Darrell, and Bryan Russell. "Localizing moments in video with natural language." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 5803–5812.

[5] George Awad, Jonathan Fiscus, David Joy, Martial Michel, Alan Smeaton, Wessel Kraaij, Maria Eskevich, Robin Aly, Roeland Ordelman, Marc Ritter, et al. "Trecvid 2016: Evaluating video search, video event detection, localization, and hyperlinking." In: 2016.

[6] George Awad, Jonathan Fiscus, David Joy, Martial Michel, Alan F Smeaton, Wessel Kraaij, Georges Quénot-Georges, Maria Eskevich-M, Roeland Ordelman, Marc Ritter, et al. "TRECVID 2016: Evaluating Video Search, Video Event Detection, Localization, and Hyperlinking." In: (2017).

[7] George Awad, Asad Butt, Keith Curtis, Yooyoung Lee, Jonathan Fiscus, Afzad Godil, David Joy, Andrew Delgado, Alan Smeaton, Yvette Graham, et al. "Trecvid 2018: Benchmarking video activity detection, video captioning and matching, video storytelling linking and video search." In: 2018.

[8] Alan D Baddeley. *Human memory: Theory and practice*. Psychology Press, 1997.

[9] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. "Curriculum learning." In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 41–48.

[10] Fabian Berns, Luca Rossetto, Klaus Schoeffmann, Christian Beecks, and George Awad. "V3C1 Dataset: An Evaluation of Content Characteristics." In: *Proceedings of the 2019 on International Conference on Multimedia Retrieval*. ICMR '19. Ottawa ON, Canada: ACM, 2019, pp. 334–338. ISBN: 978-1-4503-6765-3. DOI: 10.1145/3323873.3325051. URL: http://doi.acm.org/10.1145/3323873.3325051.

[11] Joao Carreira and Andrew Zisserman. "Quo vadis, action recognition? a new model and the kinetics dataset." In: *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6299–6308.

[12] David L Chen and William B Dolan. "Collecting highly parallel data for paraphrase evaluation." In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics. 2011, pp. 190–200.

[13] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." In: *arXiv preprint arXiv:1406.1078* (2014).

[14] George Cybenko. "Approximation by superpositions of a sigmoidal function." In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[15] Jeffrey Dalton, James Allan, and Pranav Mirajkar. "Zero-shot video retrieval using content and concepts." In: *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. ACM. 2013, pp. 1857–1860.

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[17] Michael Denkowski and Alon Lavie. "Meteor universal: Language specific translation evaluation for any target language." In: *Proceedings of the ninth workshop on statistical machine translation*. 2014, pp. 376–380.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." In: *arXiv preprint arXiv:1810.04805* (2018).

[19] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. "Long-term recurrent convolutional networks for visual recognition and description." In: *Proceedings of the*

IEEE conference on computer vision and pattern recognition. 2015, pp. 2625–2634.

[20] Jianfeng Dong, Xirong Li, and Cees GM Snoek. "Word2visualvec: Image and video to sentence matching by visual feature prediction." In: *arXiv preprint arXiv:1604.06838* (2016).

[21] Jianfeng Dong, Xirong Li, and Cees GM Snoek. "Predicting visual features from text for image and video caption retrieval." In: *IEEE Transactions on Multimedia* 20.12 (2018), pp. 3377–3388.

[22] Jianfeng Dong, Shaoli Huang, Duanqing Xu, and Dacheng Tao. "Dl-61-86 at TRECVID 2017: Video-to-text description." In: 2017.

[23] Jianfeng Dong, Xirong Li, Chaoxi Xu, Shouling Ji, Yuan He, Gang Yang, and Xun Wang. "Dual Encoding for Zero-Example Video Retrieval." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9346–9355.

[24] Aviv Eisenschtat and Lior Wolf. "Linking image and text with 2-way nets." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4601–4611.

[25] Fartash Faghri, David J Fleet, Jamie Ryan Kiros, and Sanja Fidler. "Vse++: Improving visual-semantic embeddings with hard negatives." In: *arXiv preprint arXiv:1707.05612* (2017).

[26] Hao Fang, Saurabh Gupta, Forrest Iandola, Rupesh K Srivastava, Li Deng, Piotr Dollár, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John C Platt, et al. "From captions to visual concepts and back." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1473–1482.

[27] Danny Francis and Benoit Huet. "L-STAP : Learned Spatio-Temporal Adaptive Pooling for Video Captioning." In: *First International Workshop on AI for Smart TV Content Production (AI4TV)@MM19*. ACM. 2019.

[28] Danny Francis, Benoit Huet, and Bernard Merialdo. "Embedding images and sentences in a common space with a recurrent capsule network." In: *2018 International Conference on Content-Based Multimedia Indexing (CBMI)*. IEEE. 2018, pp. 1–6.

[29] Danny Francis, Benoit Huet, and Bernard Merialdo. "Gated Recurrent Capsules for Visual Word Embeddings." In: *International Conference on Multimedia Modeling*. Springer. 2019, pp. 278–290.

[30] Danny Francis, Phuong Anh Nguyen, Benoit Huet, and Chong-Wah Ngo. "Fusion of Multimodal Embeddings for Ad-Hoc Video Search." In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. IEEE. 2019.

[31] Jiuxiang Gu, Jianfei Cai, Shafiq R Joty, Li Niu, and Gang Wang. "Look, imagine and match: Improving textual-visual cross-modal retrieval with generative models." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7181–7189.

[32] Jiuxiang Gu, Jianfei Cai, Gang Wang, and Tsuhan Chen. "Stack-captioning: Coarse-to-fine learning for image captioning." In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[33] Zhao Guo, Lianli Gao, Jingkuan Song, Xing Xu, Jie Shao, and Heng Tao Shen. "Attention-based LSTM with semantic consistency for videos captioning." In: *Proceedings of the 24th ACM international conference on Multimedia*. ACM. 2016, pp. 357–361.

[34] Amirhossein Habibian, Thomas Mensink, and Cees GM Snoek. "Composite concept discovery for zero-shot video event detection." In: *Proceedings of International Conference on Multimedia Retrieval*. ACM. 2014, p. 17.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[36] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. "Transforming auto-encoders." In: *International Conference on Artificial Neural Networks*. Springer. 2011, pp. 44–51.

[37] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. "Matrix capsules with EM routing." In: (2018).

[38] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[39] Micah Hodosh, Peter Young, and Julia Hockenmaier. "Framing image description as a ranking task: Data, models and evaluation metrics." In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 853–899.

[40] Harold Hotelling. "Relations between two sets of variates." In: *Breakthroughs in statistics*. Springer, 1992, pp. 162–190.

[41] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *International Conference on Machine Learning*. 2015, pp. 448–456.

[42] Justin Johnson, Andrej Karpathy, and Li Fei-Fei. "Densecap: Fully convolutional localization networks for dense captioning." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4565–4574.

[43] Andrej Karpathy and Li Fei-Fei. “Deep visual-semantic alignments for generating image descriptions.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3128–3137.

[44] Andrej Karpathy, Armand Joulin, and Li F Fei-Fei. “Deep fragment embeddings for bidirectional image sentence mapping.” In: *Advances in neural information processing systems*. 2014, pp. 1889–1897.

[45] Yoon Kim. “Convolutional Neural Networks for Sentence Classification.” In: ().

[46] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 2014. URL: http://arxiv.org/abs/1412.6980.

[47] Ryan Kiros, Ruslan Salakhutdinov, and Richard S Zemel. “Unifying visual-semantic embeddings with multimodal neural language models.” In: *arXiv preprint arXiv:1411.2539* (2014).

[48] Benjamin Klein, Guy Lev, Gil Sadeh, and Lior Wolf. “Associating neural word embeddings with deep image representations using fisher vectors.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 4437–4446.

[49] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, et al. “Visual genome: Connecting language and vision using crowdsourced dense image annotations.” In: *International Journal of Computer Vision* 123.1 (2017), pp. 32–73.

[50] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks.” In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[51] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, et al. “The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale.” In: *arXiv preprint arXiv:1811.00982* (2018).

[52] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. “Gradient-based learning applied to document recognition.” In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[53] Kuang-Huei Lee, Xi Chen, Gang Hua, Houdong Hu, and Xiaodong He. "Stacked cross attention for image-text matching." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 201–216.

[54] Guy Lev, Gil Sadeh, Benjamin Klein, and Lior Wolf. "Rnn fisher vectors for action recognition and image annotation." In: *European Conference on Computer Vision*. Springer. 2016, pp. 833–850.

[55] Lijun Li and Boqing Gong. "End-to-end video captioning with multitask reinforcement learning." In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2019, pp. 339–348.

[56] Yuncheng Li, Yale Song, Liangliang Cao, Joel Tetreault, Larry Goldberg, Alejandro Jaimes, and Jiebo Luo. "TGIF: A new dataset and benchmark on animated GIF description." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4641–4650.

[57] Chin-Yew Lin. "Rouge: A package for automatic evaluation of summaries." In: *Text Summarization Branches Out* (2004).

[58] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context." In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[59] Sheng Liu, Zhou Ren, and Junsong Yuan. "SibNet: Sibling Convolutional Encoder for Video Captioning." In: *2018 ACM Multimedia Conference on Multimedia Conference*. ACM. 2018, pp. 1425–1434.

[60] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. "Effective approaches to attention-based neural machine translation." In: *arXiv preprint arXiv:1508.04025* (2015).

[61] Foteini Markatopoulou, Damianos Galanopoulos, Vasileios Mezaris, and Ioannis Patras. "Query and keyframe representations for ad-hoc video search." In: *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*. ACM. 2017, pp. 407–411.

[62] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[63] Pascal Mettes, Dennis C Koelma, and Cees GM Snoek. "The imagenet shuffle: Reorganized pre-training for video event detection." In: *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*. ACM. 2016, pp. 175–182.

[64] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. "Recurrent neural network based language model." In: *Eleventh annual conference of the international speech communication association*. 2010.

[65] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality." In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[66] Niluthpol C Mithun, Juncheng Li, Florian Metze, and Amit K Roy-Chowdhury. "Joint embeddings with multimodal cues for video-text retrieval." In: *International Journal of Multimedia Information Retrieval* 8.1 (2019), pp. 3–18.

[67] Hyeonseob Nam, Jung-Woo Ha, and Jeonghee Kim. "Dual attention networks for multimodal reasoning and matching." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 299–307.

[68] Zhenxing Niu, Mo Zhou, Le Wang, Xinbo Gao, and Gang Hua. "Hierarchical multimodal lstm for dense visual-semantic embedding." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1881–1889.

[69] Yingwei Pan, Ting Yao, Houqiang Li, and Tao Mei. "Video captioning with transferred semantic attributes." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 6504–6512.

[70] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. "BLEU: a method for automatic evaluation of machine translation." In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, pp. 311–318.

[71] Ramakanth Pasunuru and Mohit Bansal. "Multi-task video captioning with video and entailment generation." In: *arXiv preprint arXiv:1704.07489* (2017).

[72] Jeffrey Pennington, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[73] Florent Perronnin and Christopher Dance. "Fisher kernels on visual vocabularies for image categorization." In: *2007 IEEE conference on computer vision and pattern recognition*. IEEE. 2007, pp. 1–8.

[74] Boris T Polyak. "Some methods of speeding up the convergence of iteration methods." In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17.

[75] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks." In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[76] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. "Self-critical sequence training for image captioning." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 7008–7024.

[77] Marcus Rohrbach, Wei Qiu, Ivan Titov, Stefan Thater, Manfred Pinkal, and Bernt Schiele. "Translating video content to natural language descriptions." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 433–440.

[78] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[79] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. "Imagenet large scale visual recognition challenge." In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[80] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. "Dynamic routing between capsules." In: *Advances in neural information processing systems*. 2017, pp. 3856–3866.

[81] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (2014).

[82] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[83] Stephanie M Strassel, Amanda Morris, Jonathan G Fiscus, Christopher Caruso, Haejoong Lee, Paul Over, James Fiumara, Barbara Shaw, Brian Antonishek, and Martial Michel. "Creating HAVIC: Heterogeneous Audio Visual Internet Collection." In: Citeseer.

[84] Fei Sun, Jiafeng Guo, Yanyan Lan, Jun Xu, and Xueqi Cheng. "Learning word representations by jointly modeling syntagmatic and paradigmatic relations." In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2015, pp. 136–145.

[85] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks." In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[86] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[87] T. Tieleman and G. Hinton. *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning. 2012.

[88] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[89] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. "Cider: Consensus-based image description evaluation." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 4566–4575.

[90] Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. "Order-embeddings of images and language." In: *arXiv preprint arXiv:1511.06361* (2015).

[91] Subhashini Venugopalan, Marcus Rohrbach, Jeffrey Donahue, Raymond Mooney, Trevor Darrell, and Kate Saenko. "Sequence to sequence-video to text." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4534–4542.

[92] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. "Show and tell: A neural image caption generator." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3156–3164.

[93] Bairui Wang, Lin Ma, Wei Zhang, and Wei Liu. "Reconstruction network for video captioning." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7622–7631.

[94] Huiyun Wang, Youjiang Xu, and Yahong Han. "Spotting and aggregating salient regions for video captioning." In: *2018 ACM Multimedia Conference on Multimedia Conference*. ACM. 2018, pp. 1519–1526.

[95] Junbo Wang, Wei Wang, Yan Huang, Liang Wang, and Tieniu Tan. "M3: multimodal memory modelling for video captioning." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7512–7520.

[96] Xin Wang, Wenhu Chen, Jiawei Wu, Yuan-Fang Wang, and William Yang Wang. "Video captioning via hierarchical reinforcement learning." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4213–4222.

[97] Xian Wu, Guanbin Li, Qingxing Cao, Qingge Ji, and Liang Lin. "Interpretable Video Captioning via Trajectory Structured Localization." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 6829–6837.

[98] Jun Xu, Tao Mei, Ting Yao, and Yong Rui. "Msr-vtt: A large video description dataset for bridging video and language." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 5288–5296.

[99] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. "Show, attend and tell: Neural image caption generation with visual attention." In: *International conference on machine learning*. 2015, pp. 2048–2057.

[100] Ziwei Yang, Yahong Han, and Zheng Wang. "Catching the temporal regions-of-interest for video captioning." In: *Proceedings of the 25th ACM international conference on Multimedia*. ACM. 2017, pp. 146–153.

[101] Li Yao, Atousa Torabi, Kyunghyun Cho, Nicolas Ballas, Christopher Pal, Hugo Larochelle, and Aaron Courville. "Describing videos by exploiting temporal structure." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4507–4515.

[102] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. "How transferable are features in deep neural networks?" In: *Advances in neural information processing systems*. 2014, pp. 3320–3328.

[103] Youngjae Yu, Hyungjin Ko, Jongwook Choi, and Gunhee Kim. "End-to-end concept word detection for video captioning, retrieval, and question answering." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 3165–3173.

[104] Junchao Zhang and Yuxin Peng. "Hierarchical Vision-Language Alignment for Video Captioning." In: *International Conference on Multimedia Modeling*. Springer. 2019, pp. 42–54.

[105] W Zhang, H Zhang, T Yao, Y Lu, J Chen, and CW Ngo. "VIREO@ TRECVID 2014: instance search and semantic indexing." In: *NIST TRECVID Workshop*. 2014.

[106] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. "Places: A 10 million Image Database for Scene Recognition." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017).

[107] Luowei Zhou, Yingbo Zhou, Jason J Corso, Richard Socher, and Caiming Xiong. "End-to-end dense video captioning with masked transformer." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8739–8748.

[108] Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. "Texygen: A benchmarking platform for text generation models." In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM. 2018, pp. 1097–1100.