RCP217 : Recherches d'images à partir de descriptions

Table des matières

I- Présentation du projet

L'objectif de ce projet est de retrouver les images qui correspondent le mieux à une description. Pour la réalisation de ce projet, j'ai à ma disposition le jeu de données fourni lors du TP5, ce dataset se nomme Flickr 8k. Chaque élément du dataset est constitué de cette façon : une image et une description associées. Nous avons un total de 5 descriptions par images et un total de 8092 images. Sachant que dans le TP5 on nous a fourni ce dataset découpé en données d'entraînement et de test. Je me suis servi en partie de ce découpage lors de ce projet. Nous verrons que j'ai modifié l'organisation de ce dataset tout au long de ce projet et je présenterai ces modifications en détails dans les parties suivantes.

II- La problématique

La problématique de ce projet est de trouver une solution permettant de comparer une image et un texte au cas par cas afin de fournir un taux de correspondance entre l'image et le texte. Cela permettra alors en comparant le taux pour chaque image de savoir laquelle correspond le mieux à notre description selon notre modèle.

III- Première réflexion

La première question que je me suis posé en commençant le projet, c'est comment comparer une image et un texte. Il faut que je trouve un support de comparaison commun. En reprenant le TP5, la première solution qui m'est venue à l'esprit était de générer une description à partir d'une image et d'ensuite comparer la description obtenue avec la description fournie. Mais cette solution ne me convenait pas, car je me suis dit que je ne comparais pas directement une image et un texte alors que c'est l'objectif principal de ce projet. Je devais alors trouver ce support de comparaison. Je me suis alors rendu compte qu'avec mon idée de base de comparer les textes entre eux, l'idée était de comparer des embedding de texte soit des vecteurs. Je me suis alors convaincu que la solution était dans la comparaison de vecteur.

Donc, maintenant, il fallait que je trouve une solution pour vectoriser ces éléments.

- Pour les données textuelles avec ce que l'on a pu voir en cours, je me suis tourné vers BERT, ce qui me permet de profiter d'un modèle préentraîner permettant de sortir les features d'un texte sous forme vectorielle.
- Pour les images dans la même lignée que pour le texte j'ai décidé d'exploiter les Deep Features de l'image obtenues à partir des Convolutionnal Neuronal Network (CNN). Pour ce faire, j'ai choisi d'utiliser des modèle préentraîner à partir d'ImageNet. Et mon premier choix, s'est tourné vers ResNet18, car c'est un modèle léger et qui permet de générer des Deep Feature rapidement ce qui est idéal pour réaliser des tests plus fréquemment.

Avec ces éléments, j'ai maintenant une solution pour obtenir des supports de comparaison. J'ai malheureusement supprimé le code contenant cette réflexion.

IV- Prétraitement

Pour pouvoir avoir un modèle performant, il est important d'avoir des données de bonne qualité. Pour ce faire, il est nécessaire d'effectuer quelques traitements sur nos données pour que celle-ci soit exploitables.

A- Prétraitement des images

Concernant les images, je n'ai pas réalisé beaucoup de prétraitement. Pour la raison que trop changer l'image avec des prétraitement pourrait impacter la qualité des descriptions fournies pour cette image. Donc les prétraitements réalisés sont ceux permettant d'utiliser l'image dans ResNet18, ce qui inclut :

- Un redimensionnement de l'image en taille 224x224
- Transformer l'image en tenseur
- Normaliser le tenseur

Toutes ces étapes sont présentées dans la documentation présente sur pytorch.org. Donc la seconde étape du prétraitement des images et de passer celle-ci dans le CNN afin d'en extraire les Deep Features.

B- Prétraitement des textes

Pour les textes, j'ai réalisé beaucoup de prétraitements afin d'avoir une description la plus discriminante possible. Voici les prétraitements mis en place :

- Suppression des balise « start » et « end » afin d'utiliser celle de BERT
- Suppression des stop words, car ils sont peu porteurs d'information
- Suppression des ponctuations, car aussi peu porteuses d'information pour la description d'une image
- Lemmatisation des mots afin de normaliser le texte
- TF-IDF afin de me construire un vocabulaire des mots les plus discriminant

Une fois tous ces traitements appliqués à ma description, celle-ci n'était pas compréhensible pour l'homme, mais à mon sens elle était discriminante (Je reviendrai plus tard sur cette décision). Ensuite, j'applique BERT sur chaque description afin d'en extraire un vecteur.

V- Premier modèle

Le résultat que j'ai voulu obtenir avec ce modèle est un taux de similitude entre l'image et la description. Donc pour ce faire, il me fallait en sortie de mon modèle une valeur comprise entre 0 et 1. Pour ce faire, j'ai choisi d'utiliser une fonction sigmoïde.

Donc mon choix, s'est porté sur un modèle complétement connecté sous cette forme :

- Une couche complétement connectée qui a en entrée un vecteur issu de notre dataset et une sortie un vecteur deux fois plus petit que celui de l'entrée
- Une couche d'activation Relu
- Une couche complétement connectée qui a en sortir un vecteur de taille 1
- Une fonction sigmoïde

Donc, j'ai mon modèle qui est défini, je sais qu'en entrée, je dois lui passer un vecteur de taille fixe. Le problème, c'est que j'ai à ma disposition 2 vecteurs. J'ai déjà fait une première constatation, peu importe la taille de ma description BERT à toujours en sortie un vecteur de taille fixe. Il en va de même pour ResNet18 qui me sort toujours un vecteur de la même taille. Ce qui m'évite de réduire la taille des vecteurs afin d'obtenir toujours des vecteurs de même

taille. J'ai alors eu l'idée de concaténer les vecteurs afin de le passer à mon modèle. J'ai aussi ajouté une valeur de similitude pour que la fonction de perte puisse sortir un résultat en comparant la valeur prédite avec la valeur attendue. Tous les vecteurs avaient alors une similitude à prédire de 1.

Pour entraîner, mon modèle j'ai choisi comme fonction de perte l'erreur des moindres carrés et comme optimiseur ADAM avec un learning rate de 0,001. C'est lors de l'entraînement de ce modèle que j'ai eu mes premiers soucis. L'entraînement durait sur 10 époques.

A- Un modèle qui n'apprend pas

Quand j'ai voulu entraîner mon modèle, je me suis rendu compte que tout au long de l'entraînement la fonction de loss ne diminuait pas. Après de nombreuses heures de recherche et de relecture, je ne voyais pas l'erreur donc j'ai contacté monsieur Rosmorduc pour avoir des conseils. Premièrement, il m'a repris sur le prétraitement des données textuelles en me rappelant que BERT est entraîné sur des phrases complètes donc si je lui donne des phrases complètes plutôt que mes phrases prétraitées, il devrait me sortir de meilleurs résultats. Donc j'ai modifié la phase de prétraitement du texte et maintenant, elle supprime juste les balises « start » et « end ». Cette remarque ne m'a pas permis de régler mon problème d'apprentissage, mais va sûrement me permettre d'obtenir de meilleurs résultats dans le futur. Après plusieurs échanges, monsieur Rosmorduc m'a mis en évidence un problème au niveau du nommage des modèles que j'ai utilisés, par conséquent ma fonction de perte ne s'applique pas sur le bon modèle. Après avoir corrigé cela, mon modèle a pu commencer à apprendre.

B- Surapprentissage

Lors de l'apprentissage de mon modèle, j'ai pu constater assez rapidement que celui-ci obtenait une valeur pour la perte de 0 et que lorsque j'essayais de faire de la prédiction, il avait toujours tendance à prédire un taux de similitude proche de 1 même pour des images fortement dissimilaires. Ces deux éléments réunis, j'ai conclu que mon modèle avait surappris. Je devais alors en trouver la cause. Je n'ai pas mis longtemps à la trouver. Effectivement, vu que lors de l'entraînement mon modèle n'a vu que des valeurs dont la similitude était de 1, il a donc alors appris à prédire uniquement des taux de similitude égaux à cette valeur. Et donc, ne sait prédire que cette valeur.

a- Nouveau dataset

Donc comme vu, le problème venait alors de mon dataset qui n'avait que des paires images/descriptions similaires ce qui engendrait un surapprentissage de mon modèle. Je devais alors trouver une solution pour résoudre ce problème. Voici les options que j'ai envisagé :

- Trouver un dataset qui en plus des paires images/descriptions a un taux de similitude associé. Après de nombreuses recherches sur Internet, je n'ai pas trouvé de dataset correspondant à mes attentes.
- Fabriquer moi-même un dataset permettant de résoudre la problématique de surapprentissage. Et c'est cette solution que j'ai alors choisie.

Pour construire ce dataset, voici la façon avec laquelle j'ai procédé. Pour chaque description, je choisis une image aléatoire dans mon dataset. Je vérifie si celle-ci n'est pas l'image qui correspond à ma description, si c'est le cas j'en tire une nouvelle. Cette opération permet de limiter les incohérences (sans pour autant les éviter à 100%) dans mon dataset en ne disant qu'une image est à la fois similaire et dissimilaire à une description. Je finis par tagguer ma paire image/description avec une similitude de 0. Ainsi, j'ai doublé la taille de mon dataset en ayant autant de paires similaires que dissimilaires. Nous verrons plus tard que j'aurais sûrement pu construire un dataset plus pertinent lors de la sélection des images.

À la suite de ce changement, mon modèle continuait à surapprendre. Mais cette fois-ci, au lieu de prédire constamment 1, il prédit constamment 0.5, comme s'il n'était pas capable de prendre une décision. De plus, la valeur de la perte était complétement aléatoire en passant de 1.5 à 29 d'une époque à l'autre. Donc je me suis dit qu'il y avait sûrement une autre raison à mon problème de surapprentissage.

b- Changement de la fonction de la loss

Avec mon nouveau jeu de données, on peut voir ma problématique comme un problème de classification binaire, ma paire est similaire ou dissimilaire. Et selon un article que m'a conseillé monsieur Rosmorduc qui est le suivant : « Why Using Mean Squared Error(MSE) Cost Function for Binary Classification is a Bad Idea? », la fonction de perte « Erreur des moindre carré » n'est pas une bonne idée pour ce genre de problème pour deux raisons :

- Utiliser MSE signifie que nous faisons l'hypothèse que nos données ont été générées par une distribution normale alors que pour le cas d'une classification binaire, les données suivent plus une distribution de Bernoulli
- La fonction MSE n'est pas convexe pour la classification binaire

Pour ces raisons, ils recommandent dans le doute d'utiliser une fonction de perte gaussienne. C'est ce que j'ai alors fait par la suite pour entraîner mon modèle. En réalisant ce changement de fonction de perte, j'avais encore mon problème de surapprentissage et donc le modèle ne donnait pas de résultat satisfaisant. Je me suis alors dit que traiter ce problème sous la forme d'une classification binaire (aidé par les propos de monsieur Rosmorduc) n'était peut-être pas la solution et j'ai alors décidé de changer d'approche pour traiter ce problème. Nous verrons plus tard que d'autres facteurs en plus d'un mauvais choix de problématique, ont peut-être impacté les résultats de mon modèle, mais sans le détailler pour autant.

VI- Seconde réflexion

Après ne pas avoir réussi à obtenir des résultats satisfaisants sur le modèle précédent et malgré de nombreuses heures de travail, j'avais la sensation d'aller droit dans un mur. J'ai alors choisi de changer d'approche pour trouver un taux de similitude entre mes éléments.

De ma réflexion précédente j'ai juste gardé la transformation de mes textes et de mes images en vecteurs. Pour le reste, j'ai décidé de repartir à zéro.

Pour cette réflexion, je me suis dit que le mieux à faire n'était pas forcément de sortir une valeur en partant d'une concaténation de ces vecteurs, mais plutôt de comparer celui de l'image avec le vecteur de la description. Je devais alors trouver un outil permettant de réaliser la tâche. L'outil doit me sortir à quel point l'image et le texte sont différents ou similaires. C'est en partant de cette réflexion que je me suis dit que l'utilisation de la similarité cosinus me semble une bonne idée. Pour rappel, la similarité cosinus permet de donner un taux de similarité (compris entre -1 et 1) entre deux vecteurs. C'est sur cette réflexion, que j'ai commencé à travailler sur un nouveau modèle. J'ai malheureusement supprimé le code contenant cette réflexion.

VII- Second modèle

Donc maintenant, j'ai dû réfléchir à trouver une solution pour faire un modèle me permettant d'utiliser à la fin une similarité cosinus. Pour commencer, la similarité cosinus ne peut comparer que des vecteurs de même taille. Ce qui génère alors un premier problème, car mes vecteurs images et textes sont de tailles différentes. Donc première étape, réduire le vecteur de l'image pour qu'il fasse la même taille que celui du texte. Pour ce faire, j'ai eu l'idée de mettre en place une structure que l'on a vu en cours qui est les encodeurs décodeurs.

Mon idée s'organise de cette façon :

1- Je fais passer le vecteur de mon image dans l'entrée de mon encodeur. L'encodeur est fait d'une couche complétement connectée qui me permettra de sortir de mon encodeur un vecteur de la même taille que mon vecteur de texte
2- Je passe à mon décodeur en entrée mon nouveau vecteur pour l'image et mon vecteur de texte. Enfin, à la sortir de mon décodeur, j'ai le résultat de ma similarité cosinus.

Pour ce modèle, j'utilise toujours un optimiseur Adam avec un learning rate faible de 0.0001. J'ai fait ce choix en me rappelant ce que j'avais en sorti de ma fonction de perte lors de mon modèle précédent. Ainsi, ce choix permet de mettre à jour les paramètres du modèle de manière lente ce qui évitera en théorie d'avoir des résultats complètement différents en sortie de ma fonction de perte d'un batch à l'autre. Pour la fonction de perte, j'ai choisi d'utiliser « CosineEmbeddingLoss », cette fonction est créée pour calculer la perte à la sortie d'une similarité cosinus.

J'ai donc pu commencer l'entraînement de mon modèle. Ici, encore j'ai eu une grosse déception en voyant que la sortie de ma fonction de perte était complétement différente d'un batch à l'autre. J'ai donc choisi d'essayer avec d'autres fonctions de perte, gaussien ou erreur des moindre carrée, mais encore une fois sans succès. J'avais beau relire mon modèle je ne voyais pas d'erreur et j'arrivais au bout de mes idées.

J'ai donc choisi de partir sur une piste totalement différente pour vérifier certains choix.

VIII- Vérification des choix

A- Comparaison de texte

Dans un premier temps, j'ai voulu me rassurer sur le fait que choisir une similarité cosinus pour comparer deux vecteurs était le bon choix. Pour ce faire, j'ai choisi de comparer la description du texte fourni avec d'autres descriptions plus ou moins similaires en me basant sur cette similarité. Pour générer les embeddings pour les descriptions, j'ai utilisé un modèle préentraîner fourni par Hugging Face. Ce modèle est « sentence-transformers/all-MiniLM-L6-v2 » et il est notamment utilisé par une API permettant de calculer la similarité entre des phrases. J'ai donc appliqué ce modèle avec une similarité cosinus pour vérifier si mon choix était le bon. La conclusion de ce test est bonne, car j'obtiens les bons taux de similarité entre les phrases en fonction de ce que je demande. Donc le choix de la similarité cosinus est validé définitivement à ce moment-là.

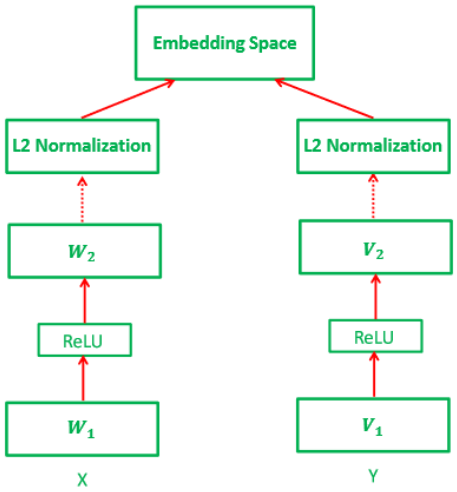
B- Nouvelle idée et contrôle de ce choix

Je me suis dit que le problème venait peut-être de mes vecteurs issus de BERT et de ResNet18. Et qu'il serait peut-être bien de normaliser l'un par rapport à l'autre afin de pouvoir les comparer dans un espace qui est comparable. Pour réaliser ce traitement, je me suis encore dit qu'une solution en encodeur décodeur me semble adapté. Mais avant de me lancer dans le développement de ce modèle, j'ai décidé de me renseigner afin de me conforter dans cette décision. En parcourant la littérature concernant ce sujet, je suis tombé sur un article de Liwei Wang, Yin Li et Svetlana Lazebnik s'intitulant : « Learning Deep Structure-Preserving Image-Text Embeddings ». Lors de la lecture de cet article, je me suis rendu compte qu'il correspondait assez bien à la vision que j'avais désormais du problème. J'ai donc décidé de m'inspirer de celui-ci pour mener à bien le projet. Sans pour autant reprendre la totalité de leur logique afin de pouvoir apprendre de mes idées et les remettre en question.

IX- Mise en place du dernier modèle

A- *Création du modèle*

Comme dit précédemment, je vais m'inspirer du modèle de l'article présenté au-dessus pour développer mon modèle. L'objectif du modèle de l'article et de créer un espace d'embedding commun entre les features de l'image et du texte afin de pouvoir les comparer. Ce qui correspondait assez bien à mon idée présentée au début du point précédent.



Vous pouvez voir comment est constituer le modèle sur l'image affichée ici.

Pour mon problème, le réseau présenté dans l'article me servira pour générer deux encodeurs. Un premier pour génerer l'embedding de l'image et le deuxième pour l'embedding du texte. Ensuite, j'aurai un décodeur qui sera une fonction de similarité cosinus qui me permettra d'obtenir un taux de similarité entre les embedding.

B- *Entraînement du modèle*

Pour entraîner mon modèle, je suis une nouvelle fois parti sur une fonction d'optimisation Adam avec un learning rate faible de 0.0001 pour les mêmes raisons qu'évoquées précédemment. Je me suis posé plus de questions pour le choix de la fonction de perte au vu des mauvais résultats présentés dans les parties précédentes. Mais finalement vu que l'objectif final est d'obtenir une similarité cosinus le choix de la fonction « CosineEmbeddingLoss » m'a semblé comme une évidence après réflexion. J'ai choisi d'entraîner mon modèle avec des batchs de taille 64 afin de trouver un compromis entre fréquence de mise à jour des paramètres et temps de traitement. Finalement, l'entraînement dure sur 10 époques. Comme vous pouvez le constater, je peux uniquement entraîner mes encodeurs car mon décodeur n'est finalement qu'une fonction.

Comment procède l'entraînement, je donne à mon modèle les features issus de BERT et celles issus de ResNET18. Celui-ci me donne en sortie deux tenseurs, un pour le texte et un pour l'image chacun issus d'un encodeur. Je passe les tenseurs à ma fonction de perte en

précisant si je dois obtenir une valeur proche de la similarité ou l'inverse (1 ou 0). Je mets à jours les paramètres du modèle en fonction de la perte et je répète cette opération autant de fois que nécessaire.

Durant l'entraînement, ma loss évolue de façon aléatoire et donc j'obtiens des résultats très mauvais lors des tests. Mais cette fois-ci je suis sûr d'avoir fait le bon choix pour le modèle et les outils utilisés, par conséquent, j'approfondis mes recherches afin de trouver les erreurs.

C- Résoudre le problème d'apprentissage

Pour commencer, ayant commis plusieurs erreurs flagrantes de développement lors de la réalisation de mon projet, j'ai commencé par relire l'intégralité de mon code, mais aucune erreur ne me saute aux yeux.

J'ai alors commencé à relire la documentation des différentes fonctions que j'utilise pour créer, entraîner et utiliser mon modèle. En faisant ça, je me suis rendu compte que je ne passais pas les bonnes valeurs à ma fonction de perte. En effet, elle attendait comme valeur dans la prédiction, soit -1 soit 1 alors que de mon côté je lui donnais soit 0 soit 1 ce qui ne correspond pas à ses attentes donc je modifie mon dataset pour passer les bonnes valeurs à ma fonction. Je réentraîne mon modèle, mais mon problème d'apprentissage persiste toujours. À ce moment-là, je ne vois pas vraiment de solution pour m'en sortir. Donc je reparcours mon code. Lors de ce parcours, je constate qu'il y a une chose que je n'ai jamais testée, les vecteurs issus de BERT et de ResNet18. Je devais alors trouver une solution pour vérifier la qualité de ces informations. J'ai encore une fois pensé à la fonction de similarité cosinus. En effet, deux vecteurs BERT vont être dans le même espace vectoriel, donc comparables et il en va de même pour deux vecteurs issus de ResNet18. Donc, je devrais pouvoir mesurer un taux de similarité entre les deux vecteurs pour vérifier s'ils sont comparables et donc par la même occasion générés correctement. Je fais d'abord le test avec les vecteurs BERT et ils semblent corrects, de plus, j'obtiens des résultats de probabilité identiques à ceux obtenus par l'utilisation du modèle fourni par Hugging Face. Maintenant, vient le tour des vecteurs images, mais impossible de les comparer, à chaque fois j'obtiens des résultats proches de 1 peu importe les vecteurs que je compare, même s'ils sont issus d'images très différentes. Donc, j'en conclus que mon problème au niveau de ma fonction de perte vient des données que je fournis au modèle lors de l'apprentissage qui sont mauvaises. Je reprends alors mon code permettant de générer les features des images et en effet depuis le début de

projet ce code ne fonctionne pas correctement ce qui a par conséquent, aussi impacté les résultats des modèles précédents. Ce qui peut expliquer en partie les mauvais résultats obtenus. Après avoir corrigé cette génération des données, je réentraîne mon modèle et le tout fonctionne comme il faut mon modèle apprend correctement et mes tests sont meilleurs. Néanmoins, ils ne sont toujours pas satisfaisants, je vais donc chercher une solution pour les améliorer. Mais, avant cela je vais vous décrire comment s'organise mes tests.

X- Réalisation de mes tests

Maintenant que j'ai un modèle fonctionnel, je vais devoir trouver un moyen de tester efficacement le modèle afin de pouvoir évaluer sa performance. Le mode opératoire que j'ai choisi dans un premier temps est le suivant : je prends les descriptions fournies par le dataset de test (À partir du dataset de test que qui était fourni dans le TP5 j'ai sorti deux datasets un de validation qui contient 3000 (x2 pour les images différentes) éléments et un de test qui en contient 2000), je génère un taux de similitude pour chacune de mes images (cela prend environ 12 secondes par image) et je regarde si l'image avec le plus fort taux de similitude est celle attendue par le dataset. En utilisant cette méthode, j'obtiens des résultats peu satisfaisants. Mais j'ai vu dans différents articles dont j'ai perdu la source et aussi dans celui qui me sert d'exemple que pour savoir si leur résultat était correct, il ne comparait pas si l'image prédite par le modèle était exactement celle attendue, mais si la similarité de l'image prédite était proche de celle attendue. Ce qui se justifie assez facilement, en effet si la description est « Des chiens jouant dans la neige », il peut avoir plusieurs images dans le dataset correspondant à cette description. Donc, il peut être compliqué pour le modèle de sortir exactement la bonne image voulue. Alors que n'importe quelle image avec des chiens jouant dans la neige correspond à la description finalement. Donc pour classifier la prédiction comme un échec ou un succès, j'effectue une distance cosinus entre les features vecteurs de l'image prédite et celle de l'image attendue, et si j'obtiens un score supérieur à 0.7 alors je comptabilise la prédiction comme étant correcte. Donc, en suivant cette méthode, je vais pouvoir savoir combien de bonnes images j'ai pu trouver en fonction des prédictions de mon jeu de test. Ce qui me permit d'avoir une augmentation de mon taux de bonnes prédictions.

XI- Amélioration des résultats

Après avoir mis en place ma logique de test, j'ai pu commencer à obtenir mes premiers résultats. Dans un premier temps, j'ai testé mon modèle en regardant le taux de bonnes prédictions obtenues lorsque je prédis uniquement l'image qui a la plus grande similitude avec ma description. En utilisant cette technique de prédiction, j'obtenais des résultats vraiment peu satisfaisants même en utilisant ma solution de calcul de similitude inter image. J'ai donc repris ma logique en me disant qu'il n'y avait pas qu'une seule bonne image par prédiction et j'ai donc refait une phase de test avec les 5 et 10 meilleures prédictions. Avec cette technique, mes taux de bonnes prédictions ont explosé. Néanmoins, ils restaient insuffisants et je devais trouver une solution pour augmenter ce taux.

A- Modification de l'entraînement

Dans un premier temps, je me suis dit qu'augmenter le nombre d'époques d'entraînement pourrait être une bonne idée afin de faire diminuer la valeur de la perte le plus possible. Mais plus on augmente le temps d'entraînement plus on augmente le risque de surapprentissage ce qui est à éviter. Pour ce faire, j'ai mis en place une couche de dropout avec une valeur de 0.5 comme il était conseillé dans l'article dont je me suis inspiré pour faire mon modèle. J'ai donc par la suite augmenté le nombre d'époques d'entraînement du modèle à 30. De plus, grâce à la mise en place d'un dataset de validation, j'ai pu mettre en place du earlystop durant l'apprentissage ce qui me permettra de limiter encore le risque de surapprentissage tout en restaurant les paramètres du modèle avec lesquels il a été le plus performant. La mise en place de ces outils m'a permis d'augmenter, encore, les performances de mon modèle, mais celui-ci restait en deçà de mes attentes donc j'ai continué à chercher d'autres solutions pour l'améliorer.

B- Modifications des données

En relisant l'article de Liwei Wang, Yin Li et Svetlana Lazebnik je me suis rendu compte qu'ils utilisaient VGG19 pour extraire les features de leurs images. Et je me suis aussi rappelé que corriger les données d'entraînement m'a permis d'améliorer l'entraînement de mon modèle plus tôt dans le projet. C'est donc en partant de ces deux éléments que j'ai modifié mon dataset afin que celui-ci importe les factures de VGG19 à la place de ResNet18. J'ai donc

réentraîné mon modèle et puis refait une phase de test. J'ai pu à la suite de cette phase constaté une amélioration dans mes résultats que je vais commenter dans la prochaine partie.

XII- Les résultats

Pour commencer cette partie, avant de parler des résultats, il faut savoir que la phase de test est quelque chose de très chronophage en temps machine, car pour prédire les meilleures images pour les 2000 prédictions cela me prend plus de 6 heures (2000 x 12 secondes).

Je vais commencer par vous afficher un tableau comparatif entre ResNet18 et VGG19 après avoir appliqué toutes les améliorations présentées et donc avec les meilleurs modèles que j'ai réussi à obtenir :

K meilleurs prédiction	Taux de précision (en %) VGG19	Taux de précision (en %) ResNet18
10	67.5	38.65
5	57.5	31.4
1	29.05	12.4

En regardant ce tableau, on peut facilement constater que le choix de la manière d'extraire les features des images est un élément déterminant pour avoir un modèle performant. Quand on prend uniquement la meilleure prédiction pour VGG, on obtient quasiment les mêmes résultats qu'avec ResNet pour les 5 meilleurs. Cette comparaison montre bien la différence d'efficacité entre les modèles.

Pour continuer, je n'ai pas les chiffres à ma disposition, mais l'ajout du dropout et du earlystop a permis d'améliorer grandement les résultats, car déjà on passe d'un modèle qui s'entraîne sur 10 époques à un modèle qui s'entraîne sur environ 20 époques tout en restaurant les meilleurs paramètres trouvés.

Pour finir, avec cette partie, on peut constater que le fait d'augmenter le nombre de prédiction augmente grandement la véracité des résultats car cela permet de plus que triplé le taux de bonne prédiction obtenu ce qui est une amélioration plus que significative. Et je

trouve que visuellement, même les images qui sont dites éloignées par le calcul de similarité entre les images et le seuil fixé ne sont pas non plus incohérentes.

Finalement, avec toutes ces améliorations, mes résultats sont assez corrects et ne sont pas si éloignés des résultats obtenus par la solution présentée dans l'article qui me sert d'inspiration.

XIII- Axes d'améliorations

Après tout ce travail, je ne prétends pas avoir un modèle parfait. Voici quelques idées qui pourraient être mises en place pour améliorer les résultats selon moi. Déjà, si dans un premier temps changer la façon d'extraire les features de mes images a permis d'améliorer mon modèle, je pense que changer la façon d'extraire les features de mon texte peut aussi l'améliorer, d'autant plus que dans l'article dont je me suis inspiré, ils n'utilisent pas la même méthode d'extraction.

Ensuite, changer simplement de dataset avec un qui a plus de données et donc qui est plus varié pourrait aussi potentiellement améliorer le modèle, car on aurait plus d'éléments discriminants. Mais aussi lors de la génération de mon dataset afin d'avoir des paires dissimilaires plus pertinentes, au lieu de choisir une image aléatoirement, j'aurai pu choisir l'image qui est la plus différente par rapport à l'image à prédire. Cette technique aurait sûrement permis d'avoir un modèle encore plus discriminant.

De plus, utiliser une autre distance que la distance cosinus peut être aussi envisageable et pourrait peut-être être une meilleure idée. Même changer, complètement le modèle afin d'avoir une prédiction avec en sortie une fonction sigmoïde comme prévu au départ pourrait peut-être être une solution envisageable. Je n'ai pas eu le temps de redévelopper cette solution avec toutes les améliorations que j'ai pu faire sur mon dataset et aussi avec l'expérience que j'ai pu obtenir lors de la réalisation de ce projet. Mais cela pourrait être un point intéressant à développer.

Pour terminer avec cette partie, il existe d'autres méthodes pour trouver des images et textes qui match ensemble, comme présenté par Haiwen Diao, Ying Zhang, Lin Ma et Huchuan Lu dans l'article «Similarity Reasoning and Filtration for Image-Text Matching » où on trouve

en détails la solution qu'ils ont imaginée mais aussi pleins d'autres solutions. On peut d'ailleurs constater qu'ils ont de bien meilleurs résultats que ce que j'ai pu obtenir.

XIV- Conclusion

J'ai vraiment rencontré des difficultés pour mener à bout ce projet. J'ai eu différentes idées à mettre en place pour répondre à cette problématique, mais à chaque fois, je n'arrivais pas au résultat escompté ce qui m'a poussé dans mes retranchements. Mais finalement, j'obtiens des résultats que je trouve satisfaisants et qui permettent d'obtenir une image proche de la description dans la plupart des cas. Cependant, je compte continuer à travailler sur ce projet même après l'avoir rendu afin de tester les solutions que je n'ai pas réussi à mettre en place au début du projet avec mes nouvelles connaissances, mais aussi mettre en place différentes solutions présentées dans les articles que j'ai pu voir.

S'il y a un point que je retiens de ce projet, c'est que je ne me lancerai plus avec mes idées de débutant dans un projet sans m'inspirer au préalable de la littérature mise à disposition par les experts du domaine. Je pense que si j'avais commencé par cette démarche, j'aurais gagné beaucoup de temps et j'aurai pu proposer, je pense une solution plus performante.

En conclusion, j'ai pu me rendre compte que malgré tout ce que j'ai pu voir en cours avec vous, j'ai principalement acquis les bases afin de pouvoir comprendre et mettre en place des modèles plus complexes que ce que j'aurais pu imaginer grâce à la littérature mise à disposition. Ce qui m'encourage à en apprendre davantage afin de pouvoir mettre en place des solutions de plus en plus performantes. Je vais commencer cette démarche en continuant mon apprentissage avec l'UE RCP 211 dispensé par le CNAM.