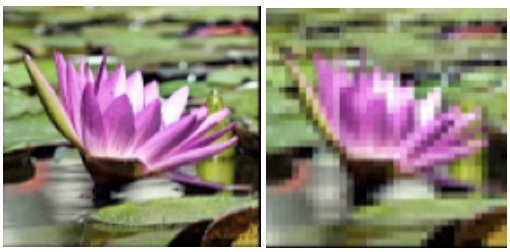
CHALLENGE

PREDICTION DE VECTEURS DE DESCRIPTION A PARTIR D'IMAGES BASSE RESOLUTION



LE CADRE DU CHALLENGE

- ### Les données :

100 000 images de visages en noir et blanc de basse résolution de dimension 48 * 48 pixels. Nous avons donc une matrice X train de 100 000 lignes * 2304 colonnes. La cible à atteindre pour chaque image est un vecteur de 128 valeurs (un « template ») qui a été construit sur la même image mais de haute résolution. La matrice y train est donc de taille 100 000 * 128.

- ### Le valid et le test set :

Les sets de validation et de test détiennent 10 000 images chacun (associées à 10 000 templates chacun).

- ### Objectif :

Notre objectif est de prédire le vecteur « template » de 128 valeurs pour chaque image du test set, qui se rapproche au plus près du template construit sur l'image de haute résolution.

- ### Critère de performance :

Le critère de performance de la prédiction est le score obtenu suivant la formule :

$$\text{score} = \frac{1}{N} \sum_{i=1}^{10000} \sum_{j=1}^{128} \left(\mathbf{Y}_{test}(i,j) - \hat{\mathbf{Y}}_{test}(i,j) \right)^2$$

POINTS ABORDES :

1. Etude des données et preprocessing
2. Présentation des deux modèles
3. Résultats
4. Conclusion

Un fichier COTADZE_maylis_challenge341.py est associé à ce pdf et contient tout le code du modèle2.

1. Etude des données et preprocessing

Je n'ai pas noté de différence au niveau des distributions du train, du valid ou du test set. Chaque image correspond à un vecteur de 2304 valeurs comprises entre 0 et 255 (ce sont des pixels). Chaque template est un vecteur de 128 valeurs centrées en 0. Il y a 10 % de duplicata sur le valid et le test set, quasiment aucun duplicata sur le train. Je n'ai pas jugé nécessaire de traiter ces duplicatas à part.

Etapes du preprocessing :

- Recentrage des valeurs des matrices X du train, du valid et du test set.
- Shuffle des lignes du train (afin d'avoir des batchs bien homogènes)

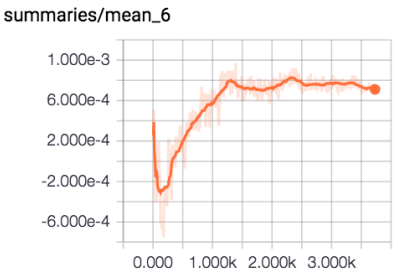
Remarque : je n'ai pas réduit les données du X, les résultats du modèle utilisé sont meilleurs lorsque les données ne sont pas réduites.

2. Les deux modèles utilisés

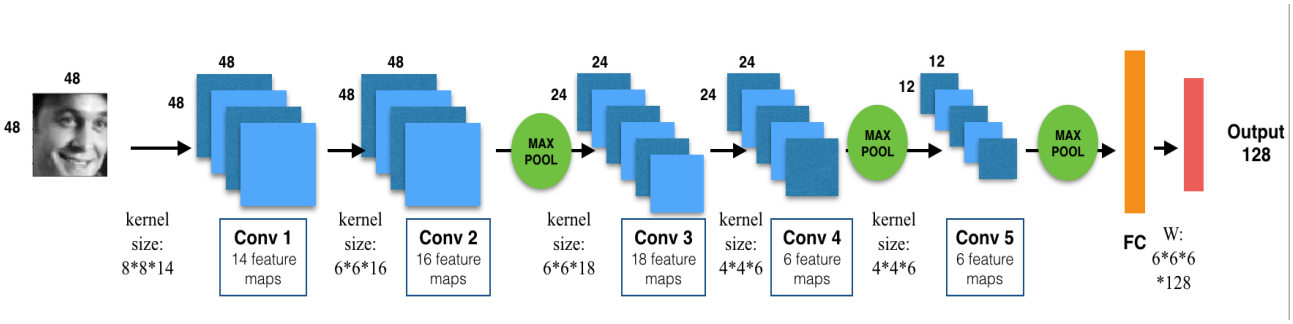
Pour définir mon modèle, j'ai cherché à m'appuyer sur la visualisation des différentes variables sur TensorBoard, en particulier sur :

- La **courbe Loss** qui correspond au graphe de la fonction optimisée par le modèle. Cette courbe loss doit décroitre avec une belle forme « ronde » de type exponentiel. La visualisation de cette courbe est très utile pour ajuster le learning rate : si la courbe est trop linéaire, le learning rate est surement trop faible, si la courbe diverge, ou ne décroit plus, le learning rate est trop élevé. Les fluctuations de la courbe permettent également de définir la taille du batch : si la courbe est trop chaotique, c'est peut-être qu'il faut augmenter la taille du batch (il y a trop de variance entre les différents batchs).

- Les poids (ou paramètres) des kernels (ou filtres) des différentes convolutions. La visualisation de ces poids permet de s'assurer qu'ils ne s'effondrent pas vers zéro (et donc qu'ils continuent de se mettre à jour) et d'ajuster leur initialisation.



Modèle 1 (en annexe, la visualisation sur TensorBoard) :



Ce modèle contient 5 couches de convolution, 3 max pooling et 1 couche fully connected.

Total paramètres :

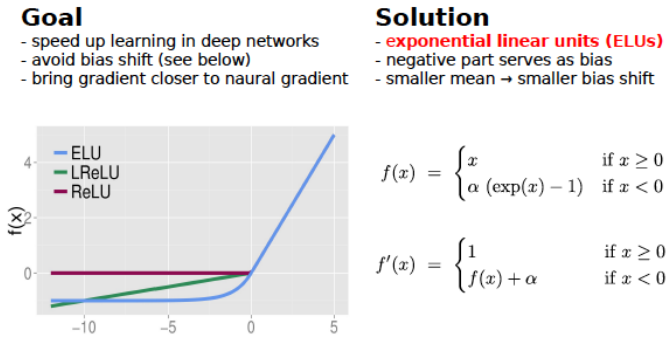
- Conv1 : 8 * 8 * 1 * 14 + 14 = 910
- Conv2 : 6 *6 * 14 * 16 + 16 = 8080
- Conv3 : 6 * 6 * 16 * 18 + 18 = 10386
- Conv4 : 4 * 4 * 18 * 6 + 6 = 1734
- Conv5 : 4 * 4 * 6 * 6 + 6 = 582
- FC : 6 * 6 * 6 * 128 + 128 = 27776

Total = 40 468 paramètres

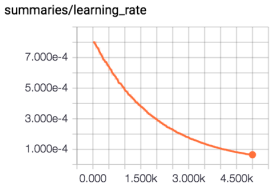
Taille des kernels : les premiers kernels ont une taille assez grande pour capturer plus de contexte. Les tailles diminuent lorsqu'on avance en profondeur dans les couches.

Fonction d'activation : La fonction d'activation des convolutions est la fonction ELU, qui permet d'atteindre un meilleur score que Relu ici.

Figure 1. Fonction ELU (source : http://image-net.org/challenges/posters/JKU_EN_RGB_Schwarz_poster.pdf)



Learning rate : le learning rate commence à 0 .0008 et décroit exponentiellement tous les 100 batchs.

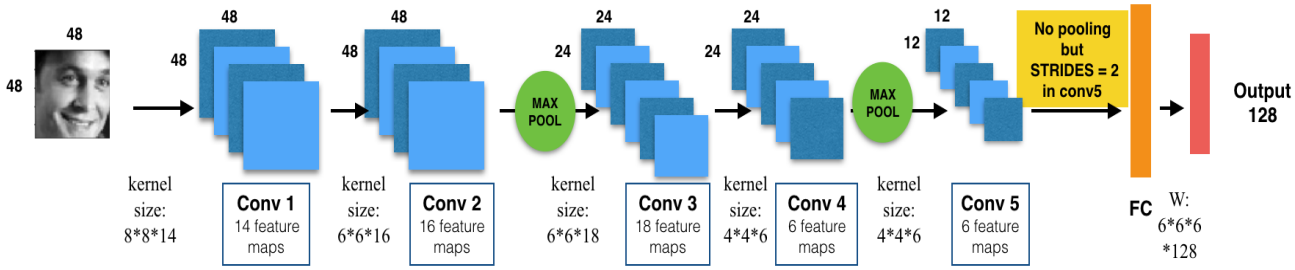


Dropout : le dropout est fixé à 1, équivalent à son absence totale. En effet, je n'ai pas trouvé une valeur du dropout qui améliore le score sur le test set. Je n'ai donc pas utilisé cette technique pour améliorer la généralisation du modèle.

Batch size et nombre d'epochs : Les batchs ont une taille de 100 images. Avec une taille de batch supérieure, l'apprentissage est plus lent. Le nombre d'epoch est fixé à 5, car le score sur le test continue de s'améliorer après 4 epochs.

(Batch normalization : la batch normalization provoque un lissage exponentiel des poids qui ne se mettent plus à jour au bout de quelques itérations, rendant l'apprentissage inefficace. Je n'ai donc pas appliqué de batch normalization)

Modèle 2



Le modèle 2 est identique au premier modèle sauf que le dernier max pooling est remplacé par une valeur du stride à 2 dans la conv 5 (qui assure la réduction de la taille de l'output).

Ce modèle contient le même nombre de paramètres que le premier.
Sa performance est légèrement supérieure au modèle 1.

Technique de généralisation utilisée :
La technique utilisée pour améliorer mon score sur le test set fut de moyenner les vecteurs prédits par plusieurs modèles de chacun des deux types dont les paramètres sont initialisés différemment (plus ou moins aléatoirement).

3. Résultats

Chaque modèle pris séparément a un score autour de 0.0072 sur le leaderboard. (donc sur le test set). Le modèle 2 est légèrement meilleur.

En moyennant une dizaine de modèles on arrive à un score de 0.0070.

4. Conclusion

En conclusion, ce challenge nous a permis de découvrir TensorFlow et j'ai beaucoup apprécié de pouvoir pratiquer sur cet librairie.
Concernant les performances de mon modèle, je pense qu'il y aurait de bonnes améliorations à mettre en place pour améliorer l'erreur de généralisation, et atteindre un meilleur score.

ANNEXE : Visualisation du modèle 1 sur TensorBoard