Projet 8

Participez à une compétition Kaggle !

Bengali.AI Handwritten Grapheme Classification

Plan de présentation

1. Problématique
2. Exploration de données
3. Feature Engineering
4. Implémentation des modèles multi-output
5. Résultats
6. Conclusion

1. PROBLÉMATIQUE

1. Problématique

kaggle

- Plateforme web organisant les compétitions en Data Science, créé en 2010 par Anthony Goldbloom

- Les compétitions consistent à résoudre des problèmes sur les données réelles, souvent fourni soit par les entreprises ou par les organismes de recherche

- Chaque compétition a son critère d'évaluation, souvent la précision des prédictions. Les meilleurs résultats peuvent être rémunérés financièrement, parfois aussi par une proposition d'embauche

- Kaggle rempli également une fonction éducative en proposant aussi des nombreux tutoriels. Les participants peuvent partager leurs notebooks et échanger sur le forum de discussion

- Mise à disposition d'une machine virtuelle équipée par GPU (utilisation gratuite 30h/semaine)

- Mise à disposition de données de nombreux domaines

1. Problématique

Compétition choisie : Classification de graphèmes de bengali écrit

- Bengali est 5ème langue le plus parlé dans le monde, avec une population de centaines de millions de locuteurs

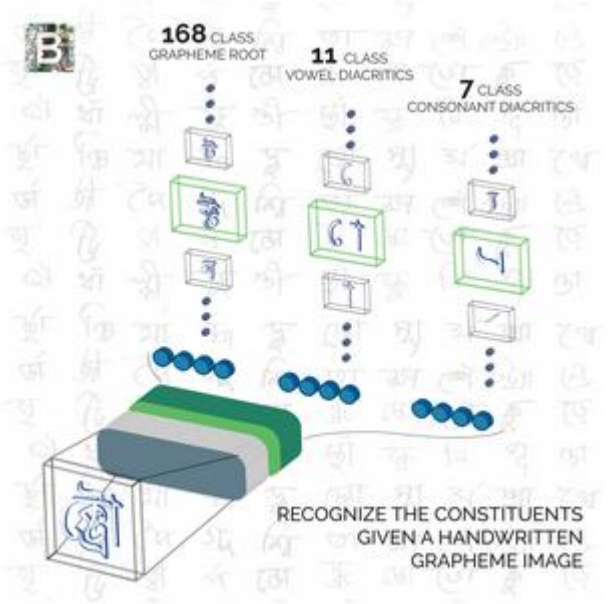
- Il s'agit de la langue officielle de Bangladesh et de Bengale Occidentale (Inde)

- Bengali est composé de 49 lettres (11 voyelles et 38 consonnes) et 18 accents (11 potentiellement attribuables aux voyelles et 7 aux consonnes)

- Les graphèmes sont formés par les syllabes. Le nombre de variations potentielles est donc d'ordre assez important (~13 000 graphèmes différents)

1. Problématique

Le but de la compétition est :



- Améliorer les approches de reconnaissance de Bengali écrit, qui sont potentiellement extensibles aux autres langues issue de la famille de sanscrit
- Démocratiser et accélérer la recherche dans les technologies linguistiques
- Promouvoir l'éducation en Machine Learning

Les conditions :

- Critère d'évaluation : Recall pondéré (graphème racine 2x plus de poids que les graphèmes diacritique)
- Temps d'exécution ne peut pas dépasser 2h sur GPU ou 9h sur CPU

Le but du projet est :

- Construire ou customiser un algorithme de machine learning pour classifier les graphèmes de bengali

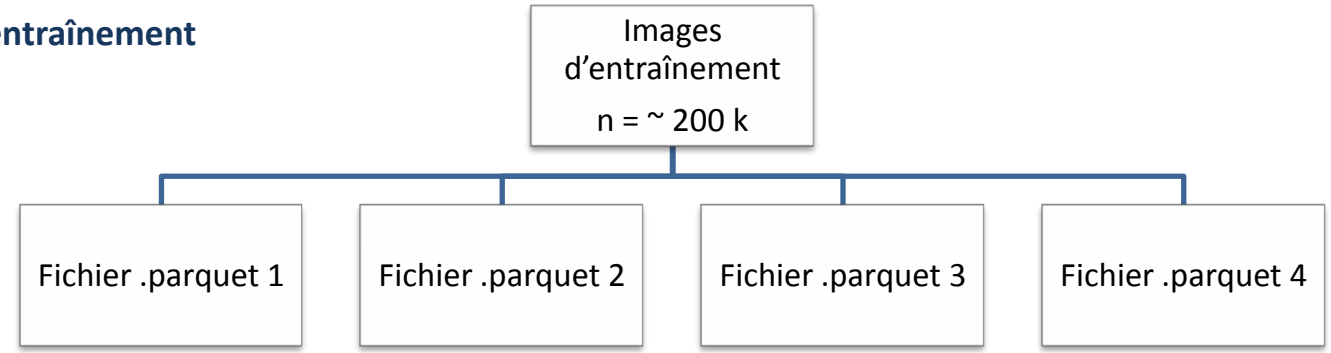
- La sortie de modèle est multioutput : le modèle devrait être capable d'identifier à la fois le graphème racine et des potentiels graphèmes de diacritique voyelle et consonne

- Le but est de participer activement à la compétition Kaggle :
  - Créer un équipe avec les autres étudiants
  - S'inspirer des kernels (notebooks) des autres utilisateurs
  - Publier notre propre kernel et partager notre travail avec la communauté
  - Soumettre les résultats de notre modèle final

2. ANALYSE EXPLORATOIRE DE DONNÉES

2. Analyse exploratoire de données

Données d'entraînement



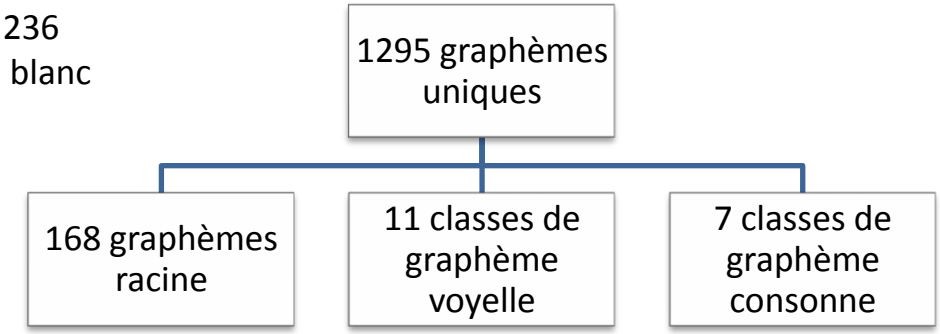
	image_id	grapheme_root	vowel_diacritic	consonant_diacritic	grapheme
0	Train_0	15	9	5	ক্ট্রো
1	Train_1	159	0	0	হ
2	Train_2	22	3	5	খ্রী

Données de test

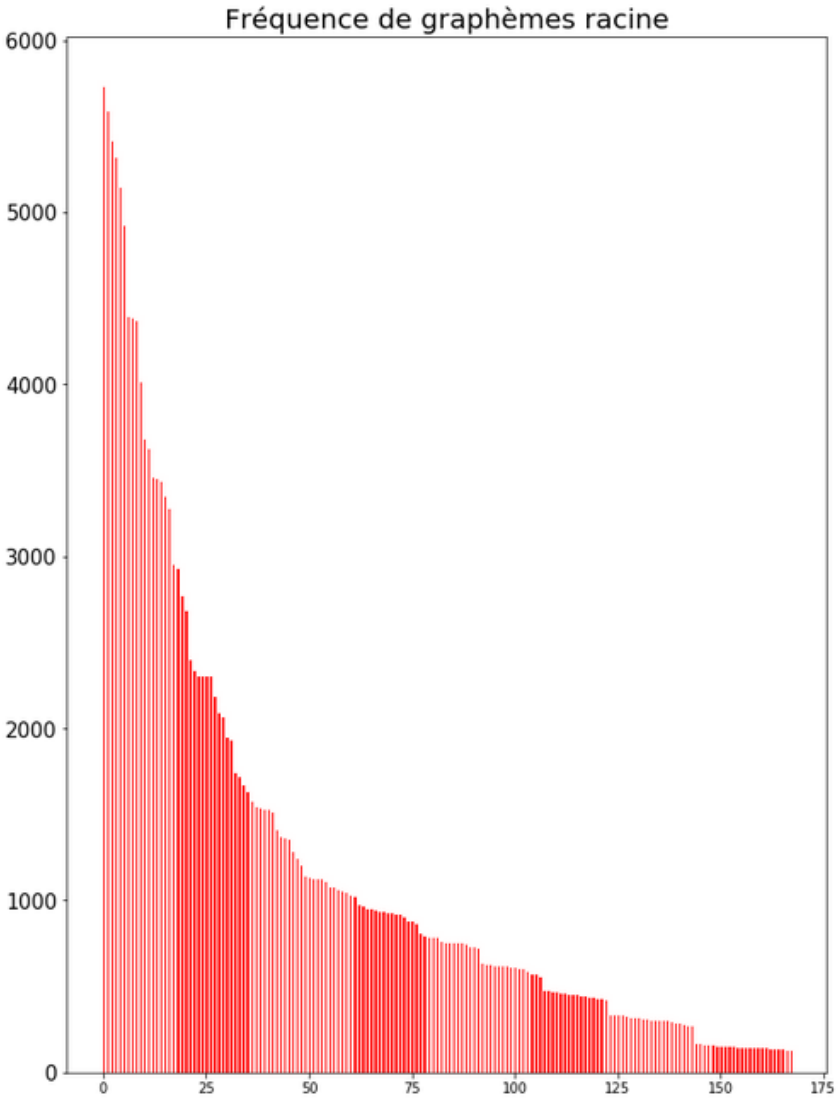
- 12 images de test
- 36 prédictions à faire et à stocker dans un fichier .csv

Description d'images

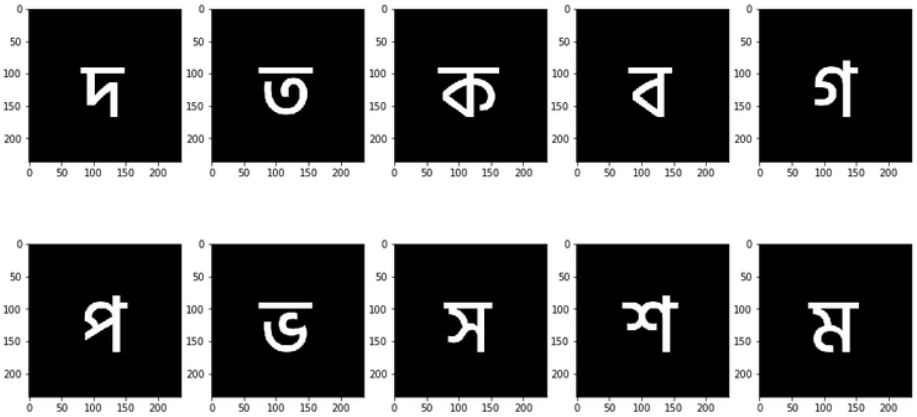
- Taille d'images : 137 x 236
- Images sont en noir et blanc



2. Analyse exploratoire de données



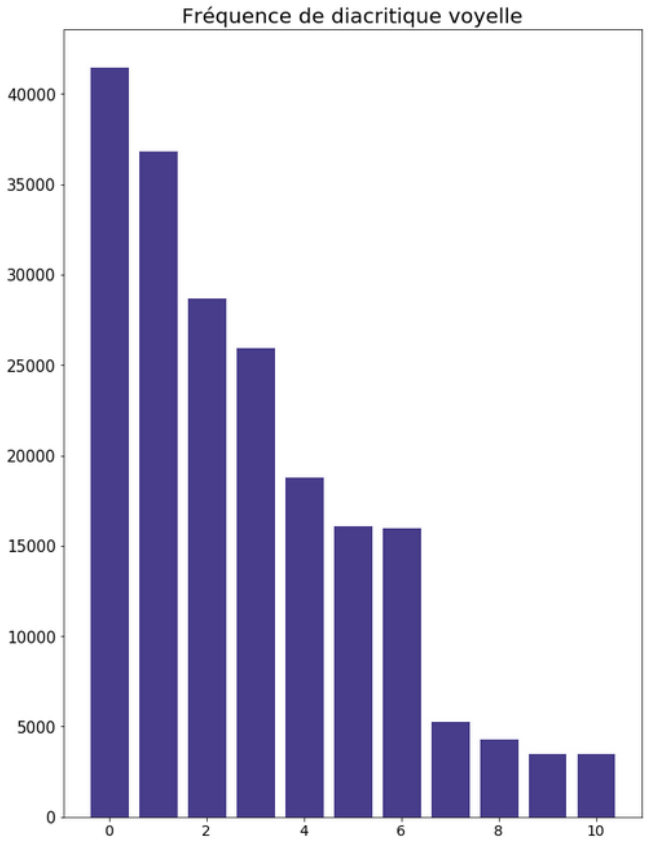
Fréquence max = 5 736

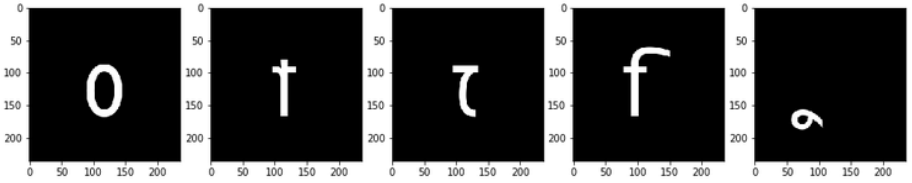


Fréquence min = 149

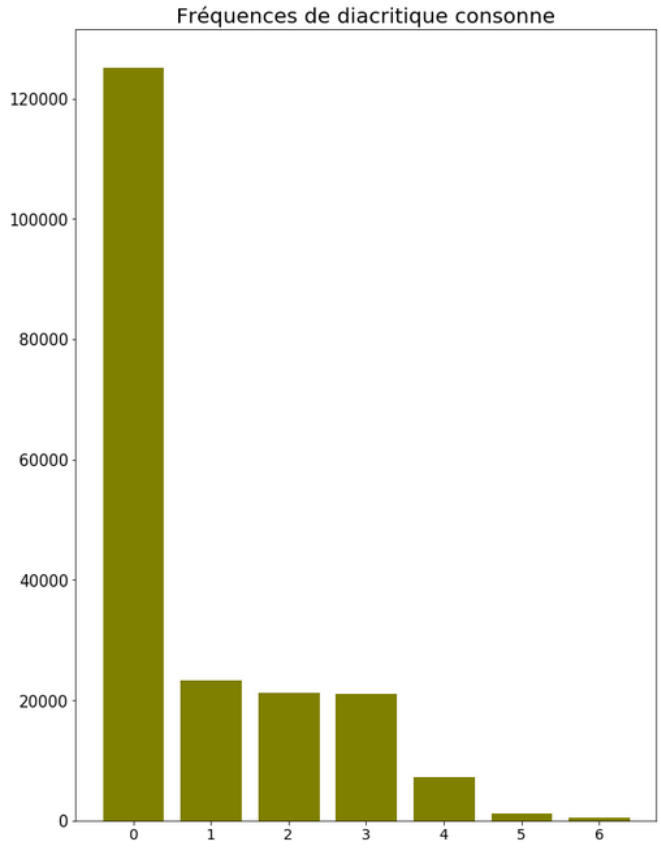
2. Analyse exploratoire de données

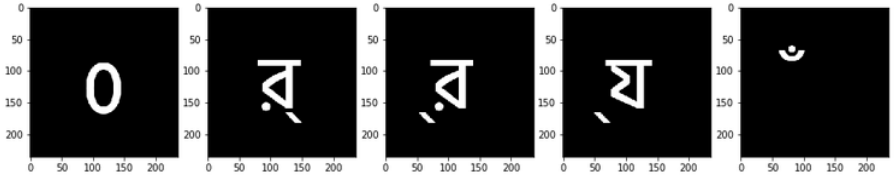
Signes de diacritique voyelle





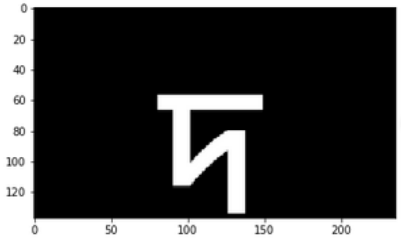
Signes de diacritique consonne



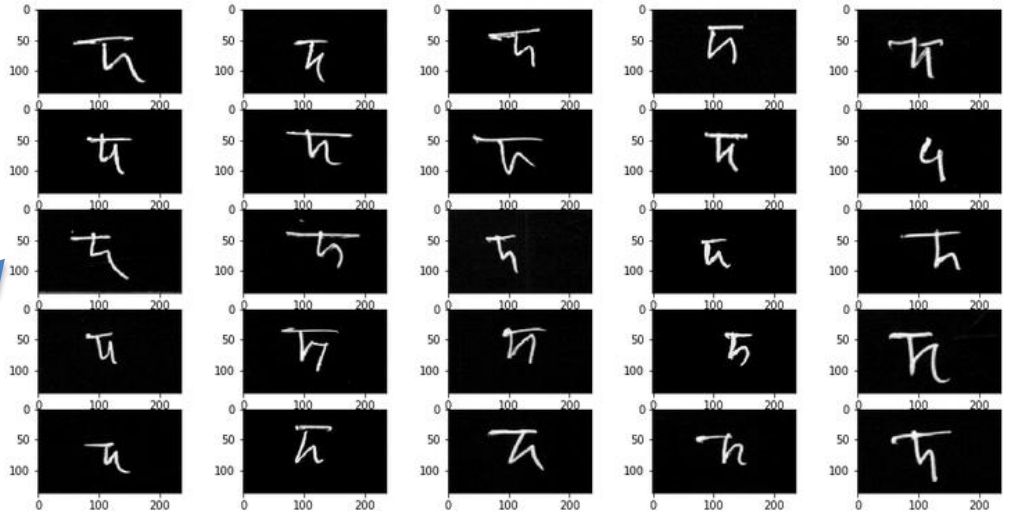


2. Analyse exploratoire de données
Graphème racine le plus courant

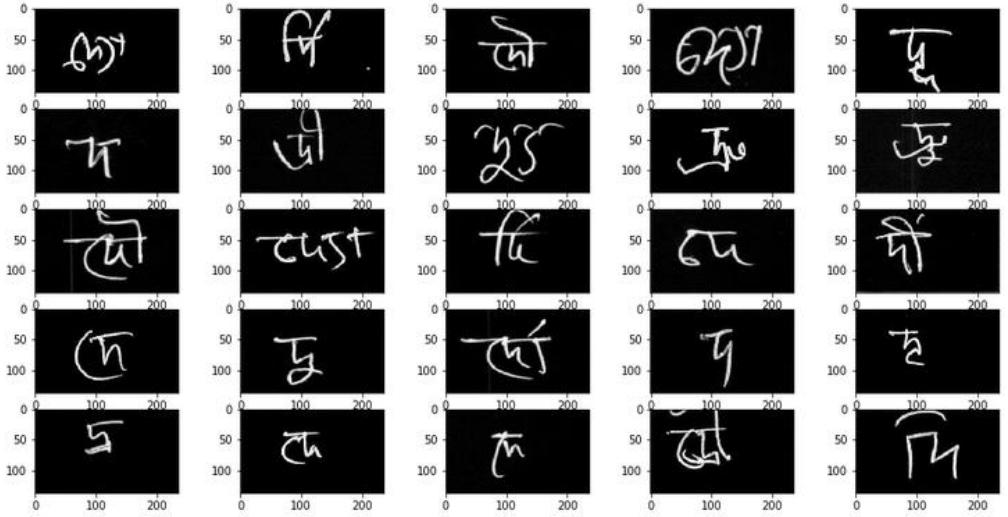
Version imprimée :



Variantes écrites sans diacritique :



Variantes écrites avec diacritique :



3. FEATURE ENGINEERING

3. Feature engineering

Deux possibilités de gestion de données d'entraînement :

1. **Lecture fichier par fichier**
   - Le principe consiste en création d'une boucle d'entraînement, exécutée pour chaque fichier à la suite
   - **Avantages :**
     - Possibilité de faire le traitement des images (par exemple augmentation etc.) à l'intérieur de la boucle d'entraînement
     - Moins exigeant au niveau de mémoire
   - **Désavantage :**
     - Possibilité que les images ne sont pas classées de façon aléatoire => risque de déséquilibre de représentation de classes dans les batchs

2. **Lecture de 4 fichiers à la fois**
   - **Avantage :**
     - Pas de risque de batchs déséquilibrés, tirage d'images de façon aléatoire
   - **Désavantages :**
     - Exigeant au niveau de mémoire, il faut lire tous les images à la fois
     - Stockage d'images traitées

Kaggle met à disposition un environnement d'exécution avec 13 GB de RAM (si nous optons pour l'utilisation le GPU).

=> Pas assez de mémoire pour charger les 4 fichiers à la fois
=> Choix d'option #1

3. Feature engineering

Feature engineering effectué :

1. Redimensionner les images
   - Taille originale : 137 x 236 pixels en NB
   - Taille de sortie : 64 x 64 pixels en NB

2. Normalisation
   - Les valeurs en pixels (entre 0 et 255) divisées par 255

3. Data Augmentation :
   - Rotation de +/- 8°
   - Zoom de +/- 15 % de la taille d'image
   - Recentrage horizontale et verticale de +/- 15% de la taille d'image

4. Création d'une classe MultiOutputDataGenerator (classe enfant de ImageDataGenerator) afin de pouvoir attribuer plusieurs cibles aux images

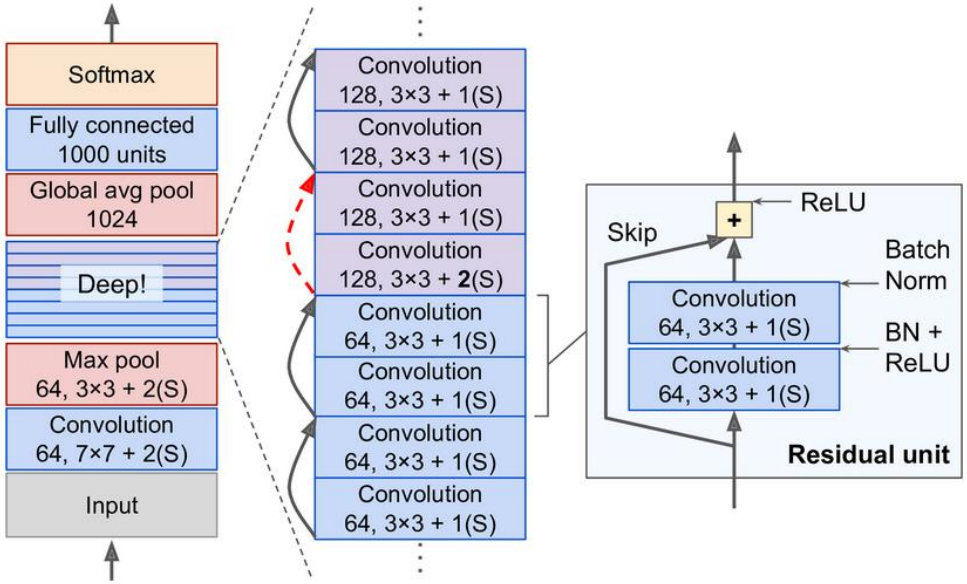
4. IMPLÉMENTATION DE MODÈLES MULTI-OUTPUT

4. Implémentation de modèles multi-output
Présentation de familles de modèles

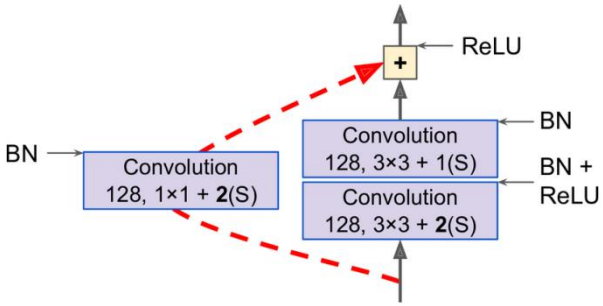
ResNet (Residual Network)

- Kaiming He et al. [4] a introduit le modèle qui a emporté la compétition ILSVCR en 2015
- Le modèle développe l'idée d'utiliser un grand nombre de couches avec un peu de paramètres
- Introduction « d'unité résiduelle » : Le signal qui rentre dans la couche est additionné à l'output de la couche située plus haut
- Quand la taille de output ne corresponde pas à la taille de input, le signal passe par « skip connexion »
- Variantes : ResNet18, ResNet34, ResNet50, ResNet152

Architecture de ResNet



Skip connexion



Source : Aurélien Géron : Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow

4. Implémentation de modèles multi-output
Présentation de familles de modèles

SENet (Squeeze and Excitation Networks)

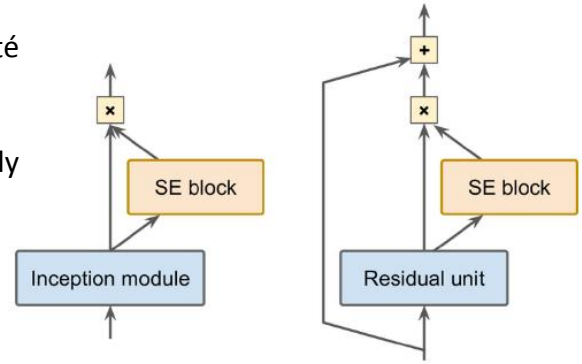
- Introduit par Jie Hu et col. [6], gagnant de compétition ILSVRC 2017

- Ajoute le module « squeeze and excitation » après chaque module Inception ou Unité résiduelle (ResNet)

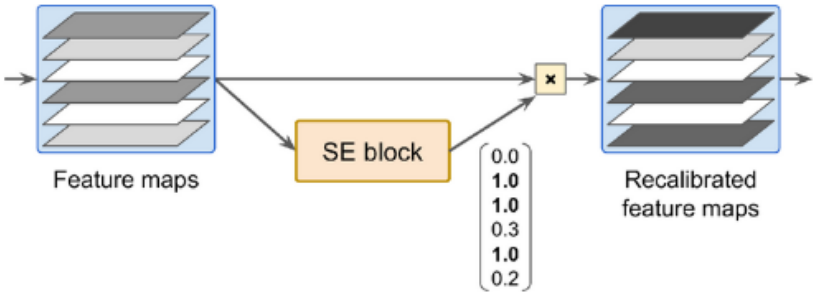
- Module est composé par une couche Global average pooling et deux couches fully connected

- **But** : re-calibrer les poids de feature maps

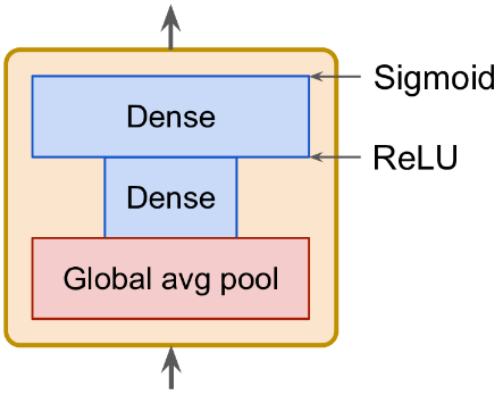
- **Pourquoi on souhaite re-calibrer les poids** :
  - Le modèle apprend des liens entre des éléments dans les images
  - Exemple : Les yeux, le nez et la bouche sont souvent sur la même photo
  - Si nous avons une forte activation des maps qui représentent les éléments nez + yeux, mais une faible activation de map qui corresponde à la bouche, le modèle va booster la feature map correspondante à la bouche



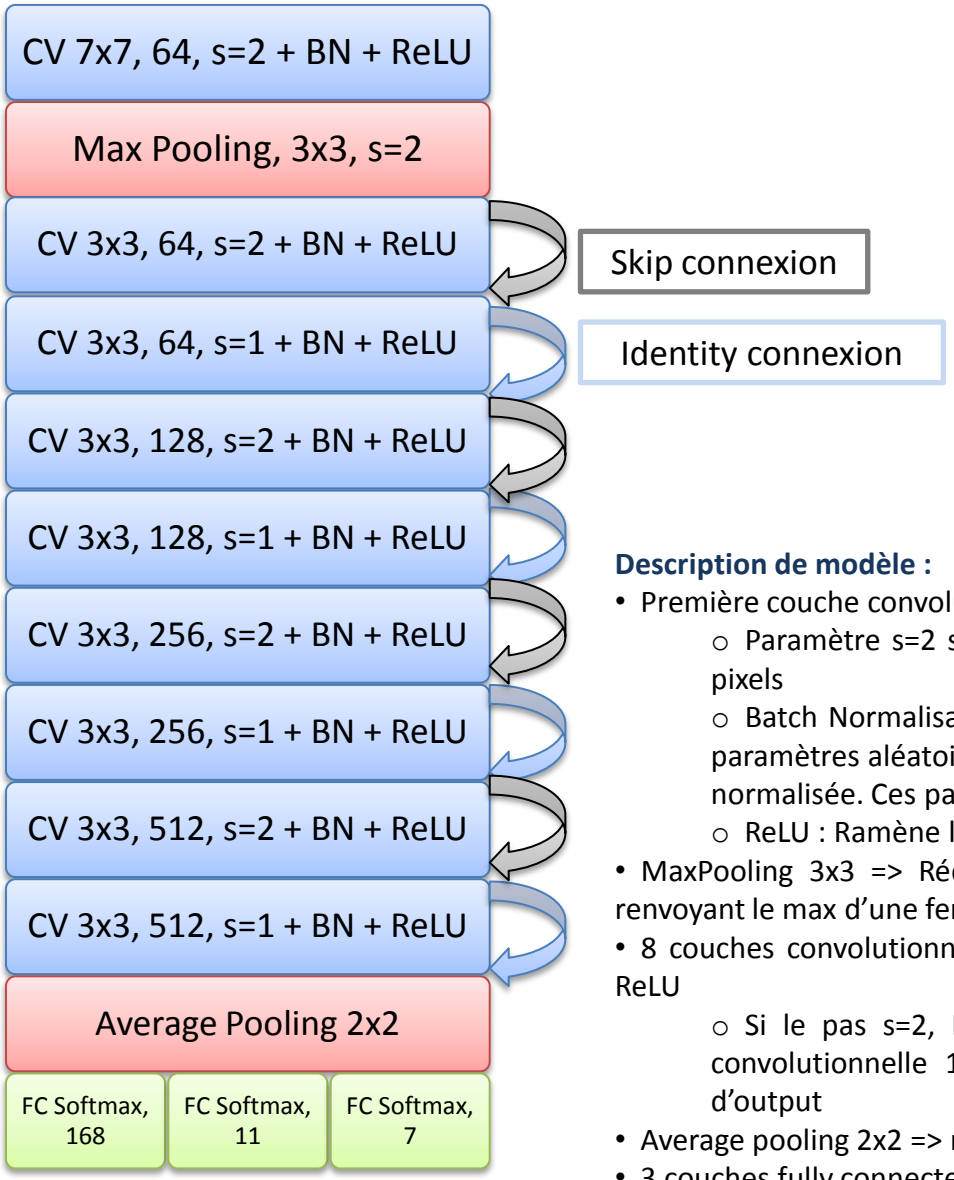
Le module SE re-calibre feature maps à la sortie



Architecture de module SE



4. Implémentation de modèles multi-output ResNet18



Choix de ResNet18 :

• Réduction de nombre de couches par rapport au ResNet50 => nombre de paramètres d'entraînement : 24 millions -> 5 millions

Modèle ResNet18 créé à l'aide de 3 fonctions :

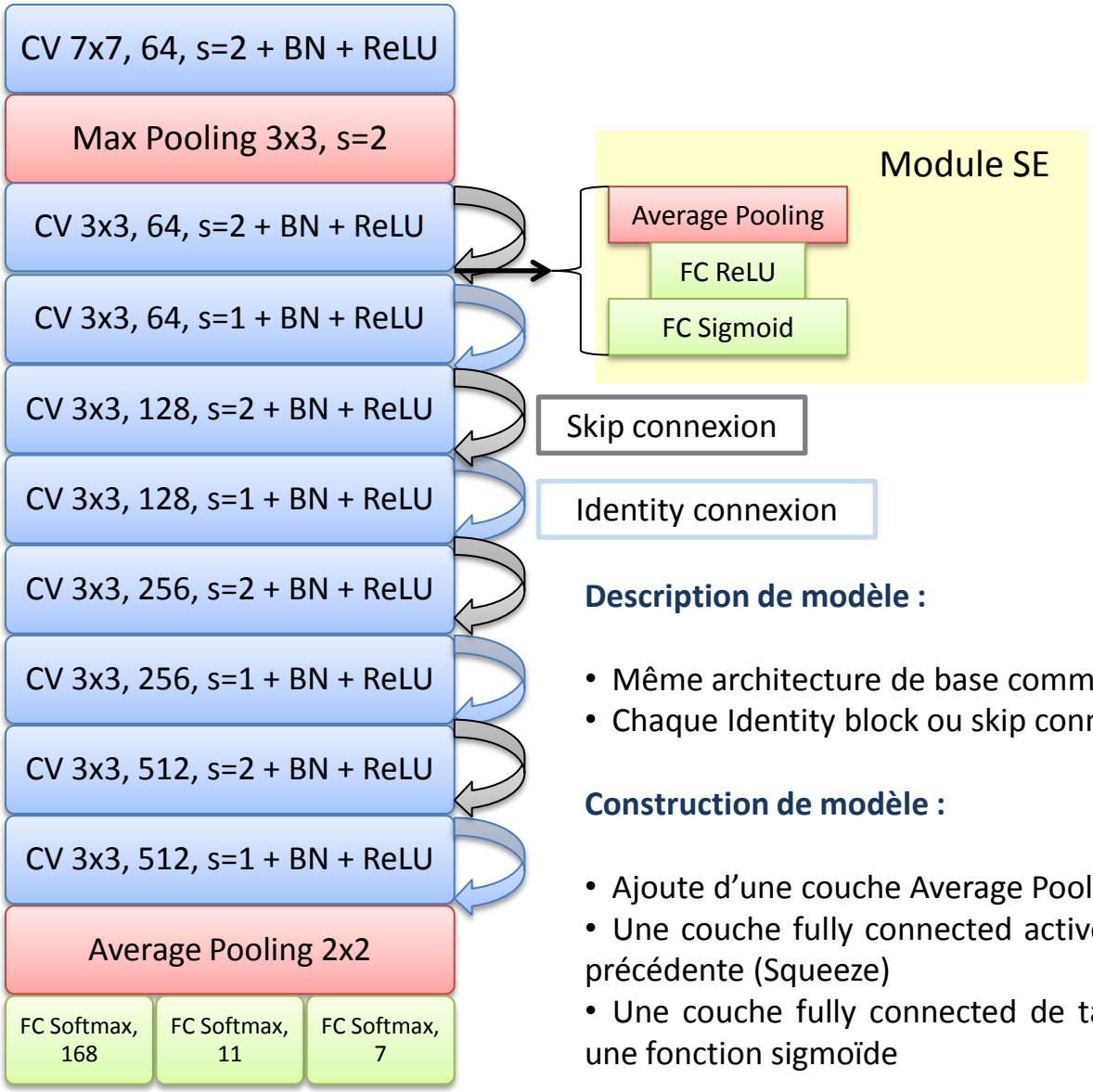
- Indentity block => identity connexion
- Convolutional block => skip connexion
- ResNet18 pour mettre ensemble les différentes couches

Source : Kernel Bengali Grapheme : ResNet 50, architecture modifiés pour correspondre au ResNet18

Description de modèle :

- Première couche convolutionnelle 7x7 avec 64 filtres.
  - Paramètre s=2 signifie que la fenêtre de filtre « se déplace » avec un pas de 2 pixels
  - Batch Normalisation : Normalise output de la couche précédente + ajoute deux paramètres aléatoires. Le premier est additionnés et le deuxième multiplie la valeur normalisée. Ces paramètres sont entraînés par le modèle
  - ReLU : Ramène la valeur à 0 si négative
- MaxPooling 3x3 => Réduit les dimensions de la sortie de la couche précédente en renvoyant le max d'une fenêtre de taille 3x3 pixels
- 8 couches convolutionnelles avec le pas de 1 ou 2 pixels, chacune terminée par BN + ReLU
  - Si le pas s=2, Identity block est remplacé par skip connexion pour (couche convolutionnelle 1x1, s=2) qui nous permet d'avoir la même taille d'input et d'output
- Average pooling 2x2 => renvoie la moyenne de la fenêtre de 2x2 pixels
- 3 couches fully connected activées par Softmax => une pour chaque output

4. Implémentation de modèles multi-output SE-ResNet18



Description de modèle :

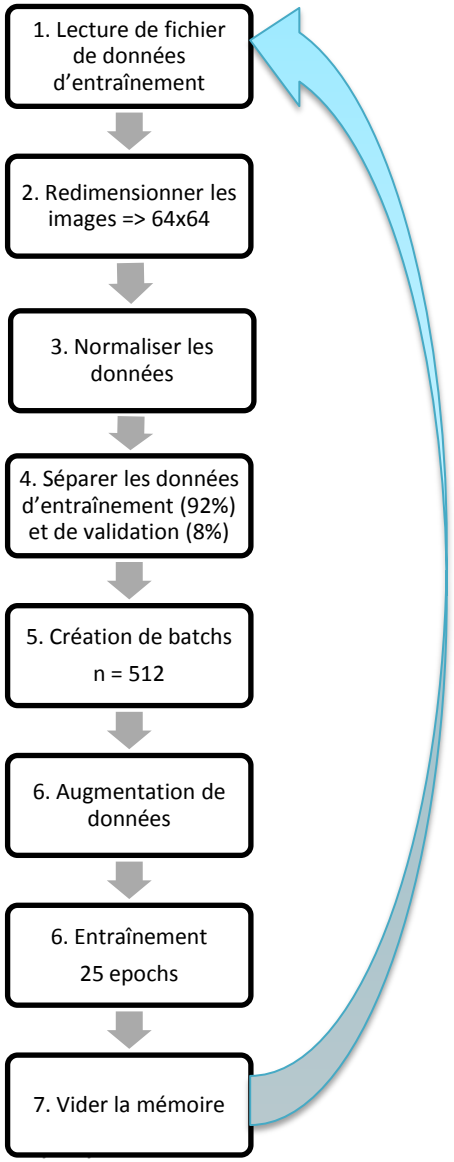
- Même architecture de base comme ResNet18
- Chaque Identity block ou skip connexion est suivi d'un module SE

Construction de modèle :

- Ajoute d'une couche Average Pooling
- Une couche fully connected activée par ReLU de taille 1/16 de couche précédente (Squeeze)
- Une couche fully connected de taille originale (Excitation) activée par une fonction sigmoïde

4. Implémentation de modèles multi-output
Options d'exécution

Workflow



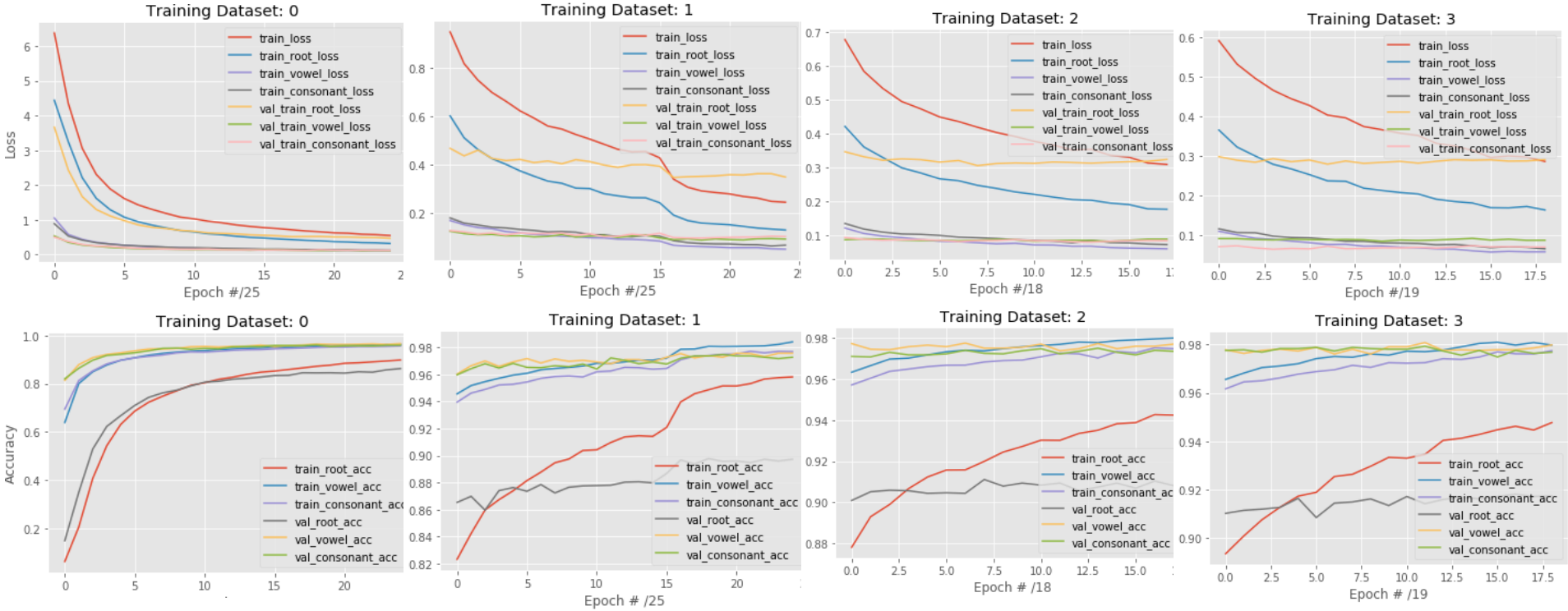
- Boucle de 4 exécutions (par fichier de données)
  - Batch size = 512
    - ➢ Les batchs sont donc tirés à l'intérieur de chaque fichier
  - Nombre d'epochs = 25
  - A la fin de chaque 25 epochs, nous la mémoire est vidé et nous procédons à la lecture de fichier suivant

- Callbacks :
  - ReduceLROnPlateau = Si l'entraînement ne fait plus de progrès au bout de 3 epochs, réduire learning rate de 0.5. LR min = 10e-6
  - ModelCheckpoint = enregistrement de meilleur modèle selon le minimum de fonction de perte de validation globale
  - Early stopping = si la valeur de fonction de perte arrête de diminuer au bout de 10 epochs, arrêter l'entraînement

- Temps d'exécution (GPU Google colab) ~ 40min

5. RÉSULTATS

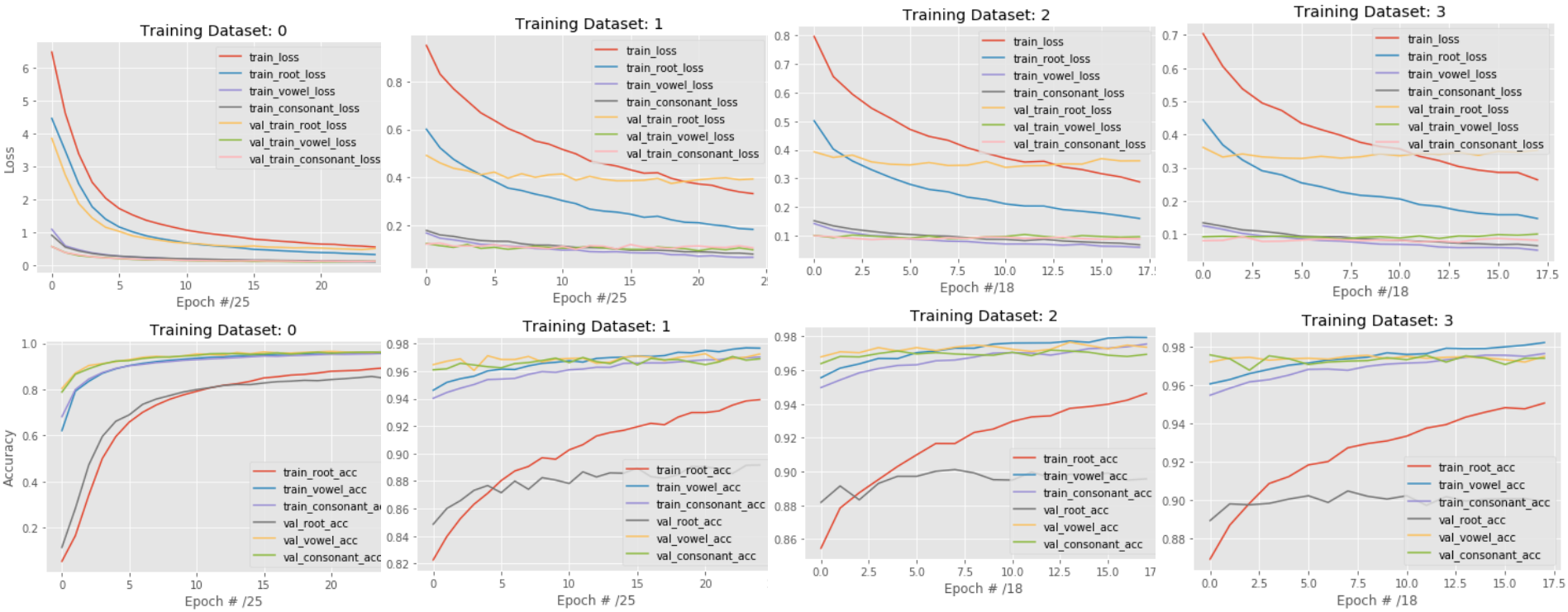
5. Résultats
ResNet18



- Meilleure précision sur les données de validation :
  - Graphème racine : 0.8566
  - Graphème diacritique voyelle : 0.9689
  - Graphème diacritique consonne : 0.9647

- Précision estimée (moyenne pondérée ou le poids de graphème racine = 2) : 0.9147

5. Résultats
SE-ResNet18



- Meilleure précision sur les données de validation :
  - Graphème racine : 0.8588
  - Graphème diacritique voyelle : 0.9691
  - Graphème diacritique consonne : 0.9659

- Précision estimée (moyenne pondérée ou le poids de graphème racine = 2) : 0.9131

6. CONCLUSION

6. Conclusion

- Participation à la compétition Kaggle :
  - Résultats de la 1ère soumission :
    - ResNet18 : 92.47 %
    - SE-ResNet18 : 80.81 %
    - Classification (le 27/01/2020) : 671 /852
    - Le meilleur résultat : 99.23 %
  - Création d'un kernel public :
    - https://www.kaggle.com/lenkast/bengali-graphemes-multioutput-resnet18-keras/notebook
    - 3 évaluations positives
    - 109 vues
- La suite de projet:
  - Nous serons ravis de continuer à améliorer nos résultats
  - Les pistes à explorer :
    - Approfondir la recherche dans la littérature et dans les autres kernels partagés
    - Tests sur différentes options d'augmentation de données, taille d'images, etc.
    - Tuning de paramètres du modèle (learning rate, optimizers etc.)
    - Tester d'autres modèles ou modifier la structure de modèle actuel