Titre cool

William Bourget et Samuel Lévesque

Université Laval

May 3, 2019

ABSTRACT

1 Introduction

C'est l'intro

das d as da sd as gsd h hhhhhhfghfgh tdh fgj fdj fg fg

2 Description des données

Google AI rend disponible une partie du jeux de données de *Quick, Draw*[1] qui est constitué de milliards de dessins fait à la main par différents utilisateurs sur leur plateforme web.

Les données sont fournies sous forme de fichiers csv (1 par classe) contenant principalement des vecteurs de positions ayant permis de réaliser les traits du dessin, le pays de l'utilisateur et la classe du dessin.

Il s'agit d'un jeu de données extrêmement volumineux, on note certaines caractéristiques :

- **340 classes**.
- $\approx$ **150 000 images/classe**.
- Plus de **51 millions d'images** au total
- **24,4 Gb** de données sous format .csv.

[1] Lien

3 Contraintes et enjeux

Une quantité de données aussi importante crée des enjeux et problèmes importants dans l'entraînement du réseau. On présente ici les enjeux principaux qui nous amènent à tester différentes manières d'entraînement.

3.1 Enjeux de mémoire

Pour pouvoir utiliser des réseaux de neurones à convolution, il faut utiliser des données étant sous forme d'image. Étant donné la quantité massive de données, il ne semble pas optimal de transformer toutes les données du format csv en images directement avant l'entraînement. Pour pallier à ce problème et pour pouvoir utiliser des réseaux pré-entraîné , on transforme le format csv des traits de crayon au moment où on charge les données dans le modèle.

3.2 Enjeux de performance

La quantité massive de données ne nous permet pas de faire des epochs traditionnelles. En effet, avec les moyens de calculs que nous utilisons (*Google Colab*) , une seule epoch peut prendre environ 1 mois en calculant 24h/24.

Pour cette même raison, on ne peut pas tester une grande quantité d'hyperparamètres. Il faut se limiter un ensemble d'hyperparamètres pour chacun des modèles que l'on veut tester.

3.3 Enjeux d'une application réelle

La mise en place d'une application concrète nous amène également quelques contraintes supplémentaires. L'application créée nous fait perdre l'information temporelle des traits de crayons puisque la sortie de l'application est une image de format png. On ne peut donc pas utiliser l'ordre dans lequel les traits ont été dessinés par l'utilisateur. Il nous est donc impossible d'utiliser différents canaux pour simuler l'évolution temporelle du dessin

4 Transformations des données

La figure 1 nous montre un exemple de transformation d'une image de maison réalisé en 2 traits de crayons.

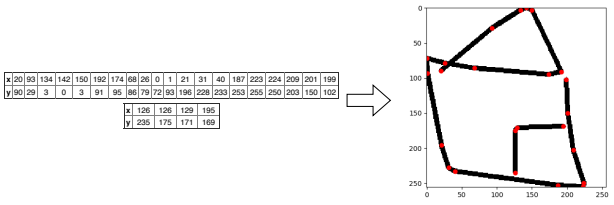

Figure 1: *Transformation de vecteur à image*

On passe donc d'une série de positions en format csv à une image 256x256 à un seul canal en noir et blanc lorsqu'on vient charger les données dans notre modèle. Cette technique évite d'avoir à stocker toutes les données sous forme d'images qui occuperait probablement plusieurs GO qui seraient instockables avec nos ressources actuelles.

La plupart des images sont loin d'être facile à apprendre pour un ordinateur, puisque que ces données ne sont pas filtrées par un humain qui peut valider si l'image est bonne. N'importe quel individu peut contribuer à cette base de données. Pour cette raison, on peut observer des images qui peuvent varier énormément pour une même classe. La figures 2 nous permet de voir différentes images pour la classe *frog*.

On peut voir que les utilisateurs n'adoptent pas tous les mêmes techniques pour dessiner un même objet. Également puisque les utilisateurs ont seulement 20 secondes pour faire leur dessin, certains d'entre eux sont parfois incomplets. Comme aucun filtrage n'a été fait, on peut constater également que certaines images ne sont pas réalisés dans le but exact du projet (certains utilisateurs ne dessinent pas la bonne catégorie ou bien ne font qu'écrire le nom de la classe)

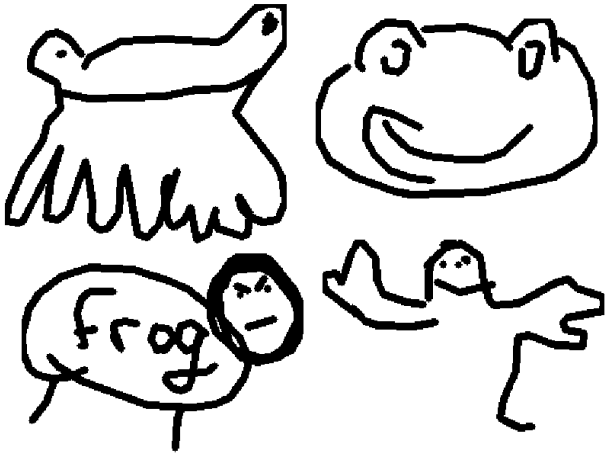

Figure 2: *Différentes images de la classe frog*

5 Structure du modèle et méthodologie

En gardant nos contraintes et enjeux en tête, il fallait trouver une façon efficace d'entraîner un modèle dans un temps raisonnable. Plusieurs techniques d'échantillonnage on pu être testées.

5.1 Échantillonnage fixe par classe

Le premier modèle testé est un modèle pré-entraîné avec architecture *Resnet18*, mais avec un seul canal de couleur au lieu de 3. Pour entraîner le modèle, on procède d'une manière alternative étant donné qu'on ne peut pas se permettre de faire de vraies epochs complètes.

Ainsi, à chacune de nos "epochs", on échantillonne (sans remise) un nombre N fixe de données par classe parmi nos 150 000 données par classe. Ces $340N$ données servent à faire une "epoch" pour notre modèle. On procède de la même façon à chaque "epoch" en réechantillonnant dans nos 150 000 données par classe (même celles pigées aux anciennes "epochs").

Cette technique a pour but de simuler un dérivé du *Bootsrapping*. Elle agit à la fois comme méthode d'échantillonnage aléatoire et comme une méthode de régularisation. On peut ainsi s'attendre à ce que le modèle n'overfit pas excessivement notre jeu de données d'entraînement en procédant de cette façon.

Si on fait 35 epochs de la sorte avec un nombre $N = 500$ (epoch de 170 000 données) on peut voir approximativement 11% de données uniques et environ 5.45% des données vues sont échantillonnées plus d'une fois parmi les 35 epochs. On peut voir qu'on est assez loin de voir de voir la totalité du jeu de données

même après les 35 epochs passées. Toutefois, le modèle semble tout de même converger vers une accuracy de validation d'environ 76% comme on peut le voir sur la figure 3. Si on applique un *early stopping*, on obtient une accuracy de validation de 77.44%

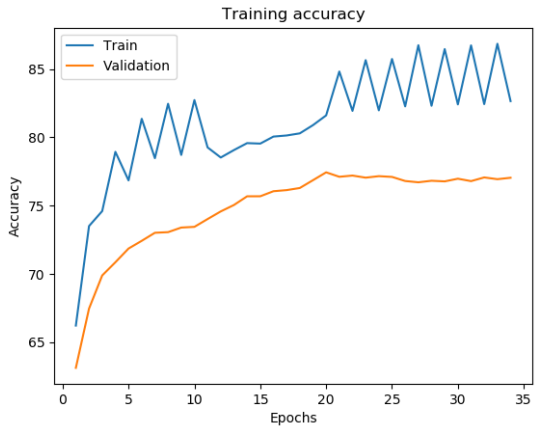

Figure 3: *Historique d'entraînement*

5.2 Échantillonnage variable par classe

La deuxième méthode d'entraînement est similaire à la première avec une légère modification qui essaie de privilégier les classes qui performment moins bien dans le but de simuler un processus de *boosting*.

L'idée de base est d'échantillonner un plus grand nombre de données des classes qui sont mal prédites et un plus petit nombre pour celles qui sont bien prédites. À chaque epoch, on calcule notre accuracy par classe A_i pour chacune des 340 classes. Pour effectuer notre epoch, on échantillonne N_i données pour la classe i:

$$N_i = N(1 - A_i) + 0.25N$$

Où:

N : Valeur constante pour toutes les classes (ex: 500)

Le terme $+0.25N$ permet de sélectionner au moins un minimum de données d'une classe qui perfomme déjà très bien pour que le modèle ne l'oublie pas lors la prochaine epoch. Ainsi, on sélectionne N données d'une classe avec 25% d'accuracy et $0.35N$ données d'une classe qui a 90% d'accuracy

En appliquant cette technique d'apprentissage, on obtient un accuracy de 79.53% avec de l'early stopping.

5.3 Modèle par ensemble avec couche de classification

Le troisième modèle consiste à concaténer les sorties des 2 premiers modèles et les passer dans une couche de classification linéaire. Pour l'entraînement de ce modèle, on gèle tous les paramètres des modèles *Resnet18* et on entraîne seulement la couche de classification avec un échantillonnage fixe. La figure 4 illustre le processus.

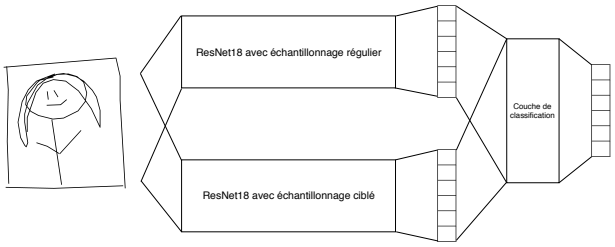

Figure 4: *Modèle par ensemble avec couche de classification*

Par manque de temps, l'entraînement du modèle n'a pas pu être complété à 100%, on peut toutefois voir l'historique d'entraînement du modèle sur la figure 5.

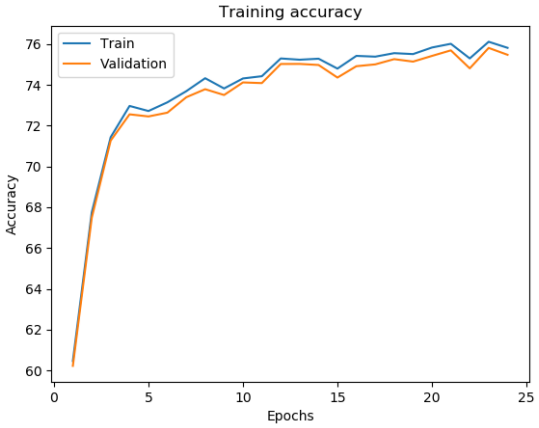

Figure 5: *Historique d'entraînement*

On voit que l'accuracy sur le jeu d'entraînement est pratiquement identique à celle de validation et qu'elles ne semblent pas encore avoir atteint un plateau. Il aurait été intéressant de voir les résultats d'un plus grand nombre d'epochs. Avec un *early stopping*, on obtient une accuracy de validation de 75.81%.

5.4 Modèle par ensemble avec moyenne simple des 2 premiers modèles

Le dernier modèle essayé consiste en une moyenne simple des 2 premiers modèles *Resnet18*. Le modèle renvoie tout simplement la moyenne des deux sorties.

6 Performances

6.1 Résultats en test

Voici la liste des résultats obtenus sur le jeu de données de test qui comporte 340 000 (1000 par classe) données:

Modèle	Accuracy	MAP@3
Resnet18	77.57	83.78
Resnet18 données ciblés	79.58	85.42
Ensemble avec classif	75.92	82.64
Ensemble avec moyenne	**79.93**	**85.66**

On peut voir que l'échantillonnage ciblé du second modèle nous apporte un gain en accuracy d'environ 2% (de 77.57% à 79.58%). C'est un résultat assez intéressant. On peut supposer que cela a permis de gagner un peu en accuracy sur les classes moins bien prédites sans trop affecter celles qui déjà bien classées.

Malheureusement, la modèle par ensemble avec la couche de classification supplémentaire ne donne pas les résultats escomptés, avec une accuracy en test de 75.92%, c'est le plus faible de tous les modèles essayés.

Le meilleur d'entre tous est le modèle qui fait une moyenne simple des deux premiers modèles avec une accuracy de 79.93%.

6.2 Analyse d'erreur

Il est intéressant de regarder quelques prédictions de notre modèle, cela permet de voir pourquoi il ne fonctionne pas toujours parfaitement. On peut également observer l'accuracy par classe en test pour voir quelles classes fonctionnent moins bien.

Voici un tableau de quelques classes avec une accuracy assez faible (modèle avec échantillons ciblés):

Classe	Accuracy
tornado	0.142
birthday cake	0.254
guitar	0.492
pool	0.558
trombone	0.640
frog	0.693

On constate que la plupart de ces classes de dessin de sont évidentes à dessiner et plusieurs techniques peuvent être utilisées pour faire certains de ses dessins. Un humain peut généralement reconnaître le dessin d'un même objet, même si celui est représenté de plusieurs manières différentes (ex: grenouille à la figure 2), ce n'est toutefois pas aussi évident pour réseau de convolution.

On peut observer des classes ayant bien performées:

Classe	Accuracy
snowman	0.953
star	0.951
ladder	0.951
envelope	0.946
rainbow	0.931
clock	0.924

On constate qu'il s'agit principalement de classes qui sont simples et rapides à dessiner au complet et il n'existe pas plusieurs façons possibles pour dessiner ces dessins, par exemple snowman, star et ladder sont souvent dessinés de la même façon.

On peut également observer quelques mauvaises prédictions de notre modèle sur la figure 6

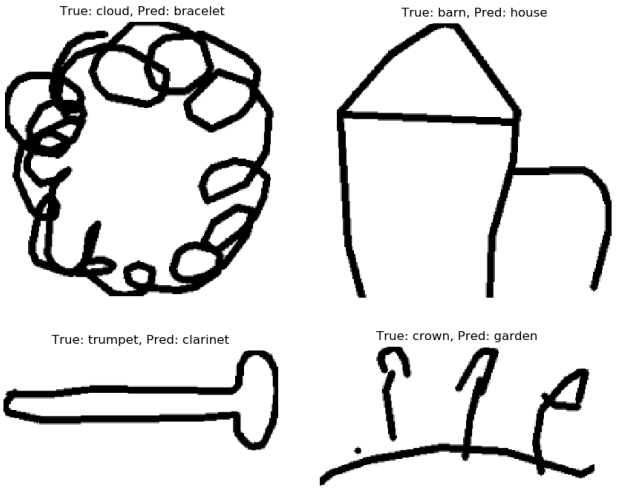

Figure 6: *Images mal prédites*

On peut voir que même quand le modèle se trompe, ses prédictions sont quand mêmes représentatives de ce qu'un humain pourrait voir. Par exemple, même si la première image est supposée être un nuage, on peut facilement interpréter ce dessin comme étant un bracelet.

Même si son accuracy n'est pas si élevé, le modèle nous donne quand même l'impression de performer assez bien étant donné que la plupart de ses prédictions font souvent du sens pour un humain même si elles ne sont pas parfaites. C'est un élément important à garder en tête lorsqu'on veut construire une application réelle en intelligence artificielle.

7 Conclusion

C’est l’intro

das d as da sd as gsd h hhhhhhfghfgh tdh fgj fdj fg fg