Reconnaissance de dessins manuscrits : projet end-to-end

William Bourget et Samuel Lvesque

Dpartement d'informatique et de gnie logiciel
Universit Laval, Qubec, Qc, Canada

10 mai 2019

On souhaite réaliser un projet *end-to-end* en traitement d'images pour couvrir toutes les étapes d'un projet en analyse prédictive tout en se familiarisant avec les solutions d'infonuagique et de travailler avec des données massives.

En utilisant des transformations de données sur nos intrants vectoriels, on arrive à arrimer notre interface graphique avec nos données d'entraînement.

Aussi, en utilisant une technique d'échantillonnage ciblé, on obtient des gains de précision de 2% par rapport à un entraînement standard sur notre architecture *ResNet18*.

Avec un modèle par ensemble simple et nos deux classificateurs *ResNet18*, on obtient une *Mean Average Precision* sur 3 prédictions de 85.66% et les prédictions du modèle semblent naturelles pour l'humain même lorsqu'elles ne correpondent pas à l'étiquette réelle de l'image qu'on tente de classifier.

L'utilisation de l'évolution temporelle des dessins et classificateurs moins corrélés et plus spécialisés dans notre modèle par ensemble nous permettrait d'augmenter les performances globales de notre modèle prédictif et de se rapprocher de l'état de l'art qui fournit des performances avoisinants les 95% en MAP@3.

1 Introduction

Le traitement d'images est une des sphères les plus impressionnantes et applicables du domaine de l'intelligence artificielle. De par la grande importance de cette branche et à cause du fait que nous n'étions pas très familiers avec le traitement d'images, nous voulions utiliser ce projet pour nous initier à un problème réel de reconnaissance d'images.

Également, nous voulions nous obliger à utiliser des solutions d'infonuagique pour traiter de grandes quantités de données que nous ne pourrions pas traiter sur nos postes personnels afin de nous rapprocher de ce qui est vraiment fait en pratique dans le milieu de l'intelligence artificielle.

Finalement, nous avions comme but de produire une interface graphique utilisable par les utilisateurs pour tester en direct les performances de notre modèle.

Pour toutes ces raisons, la création d'une interface utilisateur de reconnaissance de dessins manuscrits en utilisant la base de données *Quick, Draw !* de Google nous semblait comme un projet parfait pour nous familiariser avec les éléments que nous souhaitions approfondir.

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. *Quick, Draw !, the Data* 2018.

(1 par classe) contenant principalement des vecteurs de positions du crayon dans le temps ayant permis de réaliser les traits du dessin, le pays de l'utilisateur et la classe du dessin.

Les dessins sont composés de vecteurs distincts représentants chacun des traits de crayon et chaque vecteur est composé de points tridimensionnels (x, y, temps écoulé depuis le premier point du vecteur). Cette manière de présenter les dessins permet de sauver une grande quantité d'espace mémoire puisque ces données sont beaucoup moins volumineuses que des fichiers d'images brutes.

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

— **340 classes** (Types de dessins différents).
— ≈ **150 000 images/classe**.
— Plus de **51 millions d'images** au total
— **24,4 Gb** de données sous format .csv.

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 techniques d'entraînement.

3.1 Enjeux d'une application réelle

La création d'une interface graphique pour les utilisateurs nous amène des contraintes au niveau du format de nos intrants. Puisqu'il aurait été très complexe de créer une application qui capture les positions des traits de crayon dans le temps et de convertir ces données pour qu'elles aient le même format que notre base de données d'entraînement, nous avons décidé d'utiliser des images en intrant de notre modèle plutôt que les vecteurs position/temps. Ce choix nous force à perdre l'information temporelle des dessins qui sont faits par les utilisateurs et à convertir nos données sources avant entraînement de notre modèle. 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 tel que mentionné dans plusieurs discussions *Kaggle* [2].

2. https://www.kaggle.com/c/quickdraw-doodle-recognition/discussion/73761#latest-436738

3.2 Enjeux de mémoire

À cause des contraintes venant de notre application, nous devons transformer nos données de format vectoriel en images. Étant donné la quantité massive de données, il ne semble pas optimal de transformer toutes les données du format vectoriel en images directement avant l'entraînement puisque cela amplifierait encore plus nos problèmes de stockage. Pour palier à ce problème et pour pouvoir utiliser des réseaux préentraînés, on transforme le format vectoriel des traits de crayon au moment où on charge les données dans le modèle. Malgré cela, les 25Gb de données vectorielles amènent d'importants enjeux de stockage lors de l'entraînement. Pour éviter de transférer nos données à chaque fois qu'on veut débuter un entraînement, nosua avons décidé la plateforme *Google Colab* qui permet de se connecter directement à *Google Drive* où nous stockons nos données.

3.3 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.

4 Transformations des données

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

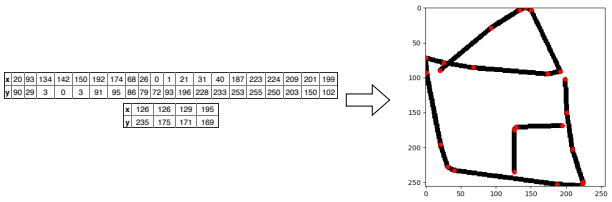

Figure 1 – *Transformation de vecteur à image*

On passe donc d'une série de positions en format vectoriel à une image 256x256 à un seul canal en noir et blanc lorsqu'on vient charger les données dans notre modèle. Cette technique nous évite de stocker toutes les données sous forme d'images qui doubleraient facilement la quantité de données à emmagasiner pour traitement.

La plupart des images sont loin d'être faciles à apprendre pour un ordinateur, puisqu'il existe une multitude de façons de dessiner une même classe. N'importe quel individu peut contribuer à cette base de données de dessins ce qui fait en sorte qu'on a une grande variété de dessins différents pour une classe donnée.. La figures 2 nous permet de voir différentes images pour la classe *frog*.

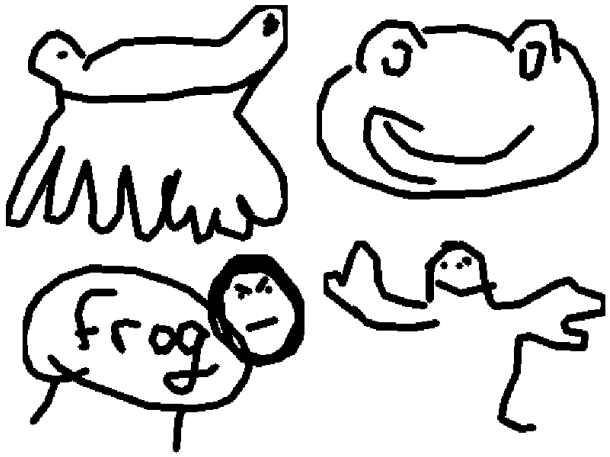

Figure 2 – *Différentes images de la classe frog*

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. Notre modèle doit donc apprendre à composer avec cette grande variabilité et faire une prédiction précise même lorsqu'un dessin est partiel.

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"). En entraînant de notre modèle de cette façon, il est possible que certaines données ne soient jamais vues et que d'autres le soient plusieurs fois.

Cette technique d'échantillonnage s'apparente au *Bootstrapping* utilisé pour les méthodes par ensemble. Elle agit à la fois comme méthode d'échantillonnage aléatoire et comme méthode de régularisation. On peut s'attendre à ce que le modèle n'overfit pas excessivement notre jeu de données d'entraînement en procédant de cette façon puisque les données ne seront pas vues plusieurs fois de façon rapide et le réseau ne peut donc pas les apprendre par coeur.

Si on fait 35 epochs de la sorte avec un nombre $N = 500$ (epoch de 170 000 données) on peut voir approximativement 11% des données totales 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 à cause de la quantité énorme de données disponibles. Toutefois, le modèle semble tout de même converger vers une précision de validation d'environ 76% comme on peut le voir sur la figure 3. En utilisant un *early stopping* et en utilisant les performances maximales sur le jeu de validation, on obtient une précision de validation de 77.44%.

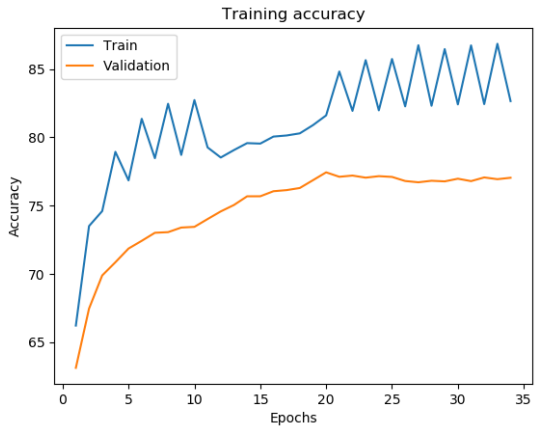

Figure 3 – *Historique d'entraînement avec entraînement standard*

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 durant l'entraînement. À chaque epoch, on calcule notre précision 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 performe déjà très bien pour que le modèle ne l'oublie pas lors la prochaine epoch. Par exemple, on sélectionnerait N données d'une classe avec 25% de précision et $0.35N$ données d'une classe qui a 90% de précision.

En appliquant cette technique d'apprentissage, on obtient une précision en validation de 79.53% avec de l'early stopping. Cette technique d'échantillonnage ciblé nous permet d'aller chercher une précision supplémentaires d'environ 2%.

5.3 Modèle par ensemble avec couche de classification

Le troisième modèle consiste à concaténer les sorties des 2 premiers modèles et à 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.

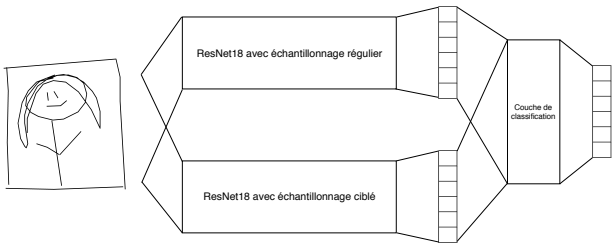

Figure 4 – *Modèle par ensemble avec couche de classification*

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.

On voit que la précision sur le jeu d'entraînement est pratiquement identique à celle de validation et qu'elles ne semblent pas encore avoir atteint un plateau lors de l'entraînement que ous avons fait. Il aurait été intéressant de voir les résultats après convergence puisque

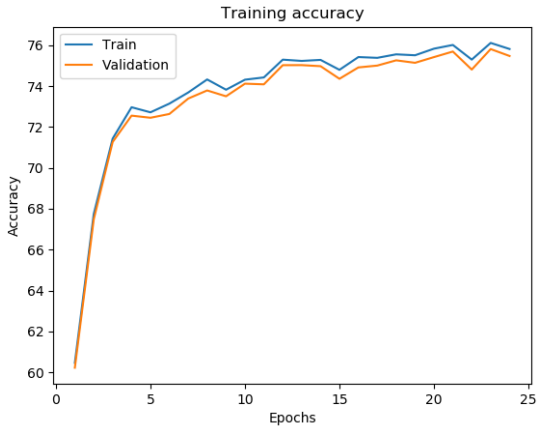

Figure 5 – *Historique d'entraînement*

nous passons que cette technique aurait pu nous permettre d'obtenir des performances légèrement supérieures à notre deuxième modèle. Avec un *early stopping*, on obtient une précision de 75.81% en validation.

5.4 Modèle par ensemble avec moyenne simple des 2 premiers modèles

Le dernier modèle que nous avons essayé est simplement d'utiliser la prédiction moyenne faite selon nos deux modèles *ResNet18*. Cette technique par ensemble nous permet d'obtenir une précision de 79,94% en validation, ce qui est légèrement supérieur à notre deuxième modèle entraîné avec échantillonnage ciblé.

6 Performances

6.1 Résultats en test

Pour comparer nos différents modèles, nous avons utilisé deux mesures : la précision et la *Mean Average Precision* sur trois prédictions (MAP@3). Le choix de cette deuxième mesure est le fait que c'est cette mesure qui est utilisée pour évaluer les modèles dans le projet Kaggle [3] présenté par *Google AI*. Cette mesure calcule la précision, mais donne des points partiels de 0.5 et 0.33 pour les cas où le modèle prédit la vraie classe en deuxième ou troisième position.

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

3. `https://www.kaggle.com/c/quickdraw-doodle-recognition`

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 de précision d'environ 2% (de 77.57% à 79.58% en précision et de 83.78% à 85.42% en MAP@3). C'est un résultat assez intéressant puisque cette technique n'est pas standard, mais apporte tout de même un gain de performance non négligeable. On peut supposer que cela a permis de gagner un peu en précision sur les classes moins bien prédites sans trop affecter celles qui étaient déjà bien classées. On peut expliquer cela par le fait que les classes les plus difficiles à prédire sont celles qui ont une grande variabilité dans les dessins faits par les utilisateurs. Il faut donc plus d'exemples différents de cette classe pour bien capter toutes leurs particularités alors qu'on n'a besoin que de quelques observations pour les classes plus simples.

Malheureusement, la modèle par ensemble avec la couche de classification supplémentaire ne donne pas les résultats escomptés. Avec une précision en test de 75.92%, c'est le plus faible de tous les modèles essayés. On pense toutefois que si on avait eu un entraînement qui converge, ce modèle aurait donné des performances au moins aussi bonnes que le pire des deux modèles *ResNet18*.

Le meilleur d'entre tous est le modèle qui fait une moyenne simple des deux premiers modèles avec une précision de 79.93% et une MAP@3 de 85.66% en test.

Ces résultats sont loins des meilleurs modèles qui ont été soumis pour la compétition Kaggle où les meilleurs compétiteurs ont obtenus des MAP@3 d'environ 95%, mais nous sommes tout de même satisfaits des résultats obtenus compte tenu des ressources de calcul limitées qui étaient à notre disposition. Également, on peut expliquer une bonne partie de cette différence de performance par le fait que notre modèle n'utilise pas l'information temporelle des vecteurs de dessins. Certaines équipes ont utilisé des modèles aussi simples que nous, mais ont utilisés plusieurs canaux d'image pour représenter l'évolution temporelle du dessin et ont pu obtenir très rapidement des MAP@3 supérieurs à 90%.

6.2 Analyse d'erreur

Il est intéressant de regarder quelques prédictions de notre modèle pour mieux comprendre où sont les erreurs qu'il commet. On peut également observer la précision 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 ne 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.

Pour aller chercher une précision supplémentaire sur ces classes problématiques, nous aurions pu construire des classificateurs binaires spécialisés pour ces classes et les utiliser dans notre modèle par ensemble.

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

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 pourrait interpréter ce dessin comme étant un bracelet.

Même si sa précision n'est pas si élevée, le modèle nous donne quand même l'impression de performer

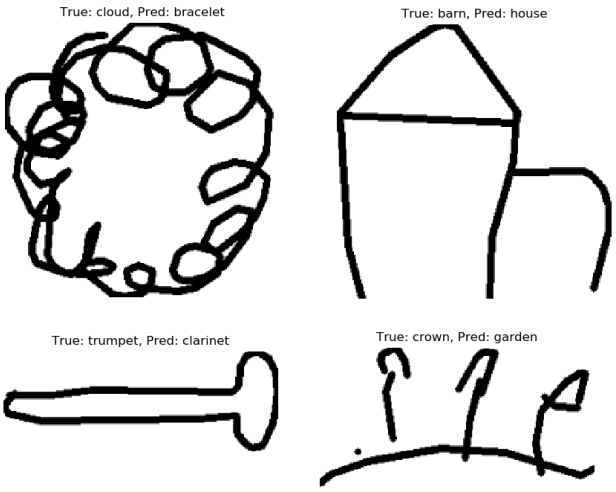

Figure 6 – *Images mal prédites*

assez bien étant donné que la plupart de ses prédictions font souvent du sens pour un humain même si elles ne sont pas parfaites. Il est possible de tester l'application créée dans le cadre de ce projet en clonant le projet Github du projet [4] pour tester d'autres exemples du prédiction du modèle.

7 Conclusion

Pour conclure, nous avons réalisé le but premier de ce projet qui était de réaliser un projet *end-to-end* en traitement d'images par la création d'une interface graphique de reconnaissance de dessins manuscrits. Nous avons également pu nous familiariser avec les outils d'infonuagique pour l'entraînement de notre réseau. Nous avons également un gain de performance non négligeable de 2% avec notre méthode d'échnatillonnage ciblé pour l'entraînement de notre modèle *ResNet18*.

Pour continuer d'améliorer les performances de notre réseau, plusieurs avenues peuevnt être testées. Entre autres, on pourrait modifier notre interface graphique pour extraire les composantes temporelles de nos dessins et les utiliser dans notre modèle prédictif.

Aussi, on pourrait intégrer d'autres modèles moins corrélés dans notre modèle par ensemble pour profiter des forces de chacun des classificateurs. Notamment, on pourrait introduire des classificateurs spécialisés pour les classes les plus difficiles à prédire pour aller chercher des performances supérieures là où notre modèle performe le moins bien.

4. *Projet GLO7030 Github Repo* 2019.

Bibliographie

Alpaydin, Ethem (2014). *Introduction to Machine Learning*. The MIT Press. isbn : 0262028182, 9780262028189.

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg : Springer-Verlag. isbn : 0387310738.

Goodfellow, Ian, Yoshua Bengio et Aaron Courville (2016). *Deep Learning*. The MIT Press. isbn : 0262035618, 9780262035613.

Murphy, Kevin P. (2012). *Machine Learning : A Probabilistic Perspective*. The MIT Press. isbn : 0262018020, 9780262018029.

Projet GLO7030 Github Repo (2019). `https://github.com/Samuel-Levesque/Projet_GLO7030/settings`.

PyQt Class Reference (2015). `https://www.riverbankcomputing.com/static/Docs/PyQt4/classes.html`.

Quick, Draw ! Doodle Recognition Challenge (2018). `https://www.kaggle.com/c/quickdraw-doodle-recognition`.

Quick, Draw !, the Data (2018). `https://quickdraw.withgoogle.com/data`.