Projet Transboost

Luc Blassel, Romain Gautron

13 Mars 2018

Table des matières

1 Contexte **2**

2 Transboost pour la classification d'images : principe **3**

3 Application **4**

3.1 Présentation des données 5

3.2 Construction d'un classificateur binaire fort sur le domaine source 5

3.3 Mise en pratique du TransBoost 6

3.4 Difficultés rencontrées 7

4 Comparaisons critiques portant sur la méthode transboost **8**

4.1 Projecteurs faibles vs. projecteurs forts 8

4.2 Projecteurs vs. nouveaux classificateurs 9

5 Conditions et méthodes d'expérimentation **9**

6 Resultats **10**

6.1 Construction du classificateur binaire 10

6.2 Boosting classique sur petit CNN 11

6.3 Méthode transboost 12

7 Expérimentations non réalisées **12**

7.1 Influence des domaines source et cible 12

7.2 Influence des hyper-paramètres : recherche d'un optimum précision/coût calculatoire 13

7.3 Transboost vs classical transfer learning 13

8 Conclusion **13**

Bibliographie **13**

Appendices **14**

A Code **14**

B Schéma du programme **14**

1 Contexte

L'apprentissage supervisé classique nécessite un grand nombre de données étiquetées, et dans certains cas une période de temps très importante pour pouvoir établir des modèles fiables. Ceci n'est pas toujours possible dans le monde réel, parfois il n'est simplement pas possible de collecter assez de données pour entraîner nos modèles ou alors le temps nécessaire pour l'entraînement du modèle est beaucoup trop long pour que ce soit utilisable en pratique avec des ressources conventionnelles. L'apprentissage par transfert peut nous aider à résoudre ce type de problèmes.

L'apprentissage par transfert nous permet de transférer des connaissances apprises depuis domaine source avec idéalement une grande quantité de données étiquetées de bonne qualité, vers un domaine cible. Cette approche permet de réutiliser des portions d'un modèle préalablement entrainé dans notre nouveau modèle. L'avantage est double : économie en temps de calcul et utilisation d'une quantité limitée de données en réinvestissant de l'information provenant d'une autre tâche. Cette méthode est considérée comme le prochain moteur de succès de l'apprentissage automatique après l'apprentissage supervisé.

La méthode TransBoost [2], introduite par Antoine Cornuéjols et ses collègues propose une implémentation de l'apprentissage par transfert différente de l'usage. Quand l'approche "classique" de l'apprentissage par transfert adapte l'hypothèse développée sur le domaine source au domaine cible, TransBoost en prend le contre-pied. En effet dans cette dernière on apprend l'hypothèse sur le domaine source et on projette ensuite les points du domaine cible sur le domaine source pour utiliser directement l'hypothèse source sur les points projetés.

Ainsi, on n'apprend pas de nouvelles frontière entre les points, on injecte plutôt les points des bons côtés des frontières source. La projection des points du domaine cible sur le domaine source se fait dans le cadre d'un algorithme de boosting, qui grâce à plusieurs projecteurs faibles, permet d'obtenir un projecteur fort. Ce dernier permet alors d'utiliser l'hypothèse source pour classifier les points du domaine cible. La différence d'approche entre apprentissage par transfert "usuel" et Transboost est illustrée sur le schéma 1.

L'approche Transboost a d'abord été testée sur la classification des séries temporelles incomplètes et à été un franc succès en ayant de bien meilleures performances que d'autres approches du même problème. Cependant, la classification d'images étant le mètre étalon en ce moment, le but de ce projet est d'adapter la méthode TransBoost à la classification d'images en utilisant des réseaux de convolution profonds (deep CNN). Ce travail ne présente que la classification binaire d'images.

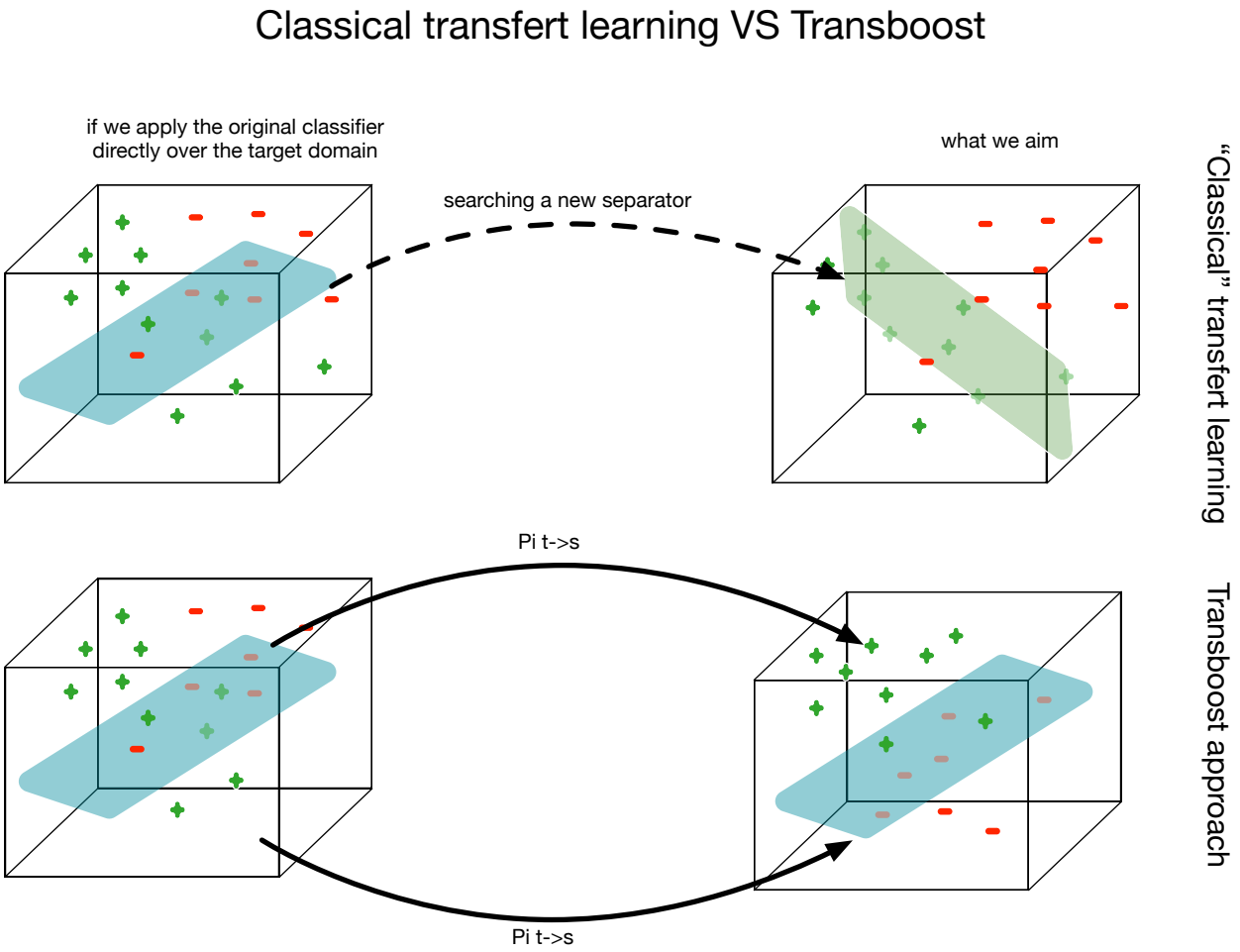

FIGURE 1 – Différences entre l'approche classique et l'approche TransBoost

2 Transboost pour la classification d'images : principe

L'application de la méthode TransBoost à la classification d'images oblige à se poser plusieurs questions. D'une part la très grande dimensionnalité des images force a utiliser des méthodes très lourdes telles que les réseaux de convolution profonds, c'est donc un défi en terme de puissance de calcul. D'autre part comment réaliser la projection des points du domaine cible dans le cas d'images ?

Le choix a été fait de modifier les premières couches du réseau "source" pour classifier les images du domaine cible. Ainsi, les premières couches du réseau existant se chargeront de trouver les bons descripteurs de faible niveau pour que la nouvelle tâche puisse être réalisée. Cependant il faudra trouver les meilleurs hyper-paramètres de ces projecteurs (voir infra).

On aurait pu également construire une réseau externe qui se charge d'être le projecteur des images et qui arrive en entrée du réseau "source". Celui-ci aurait eu en entrée des images et en sortie également des images (visualisation intéressante). A noter que cette option aurait été plus lourde en calculs (backpropagation supplémentaire) et ne sera pas explorée ici.

La construction d'un projecteur s'effectue comme suit :

— On obtient un réseau convolutionnel très performant sur une classification binaire source et aussi bon que le hasard pour une classification binaire cible. Les modèles pré-entraînés disponibles ont souvent un grand nombre de classes de sortie. Il est nécessaire de changer la couche de sortie du réseau et de l'entraîner pour l'ajuster à notre domaine source binaire.

— On gèle la partie supérieur dudit réseau en laissant les premières couches entraînables.

— On ré-entraîne ledit modèle partiellement gelé pour obtenir un projecteur en visant une valeur de métrique seuil pour arrêt. Nous choisissons la précision comme métrique, celle-ci étant parlante et pertinente dans le cas de datasets équilibrés en classes.

On construit itérativement un ensemble de projecteurs faibles spécialisés sur les erreurs des précédents selon l'algorithme Adaboost [3]. Notre hypothèse finale sur le domaine cible sera une combinaison linéaire de ces projecteurs faibles.

3 Application

Dans toute cette section, la figure 2 illustrera les propos.

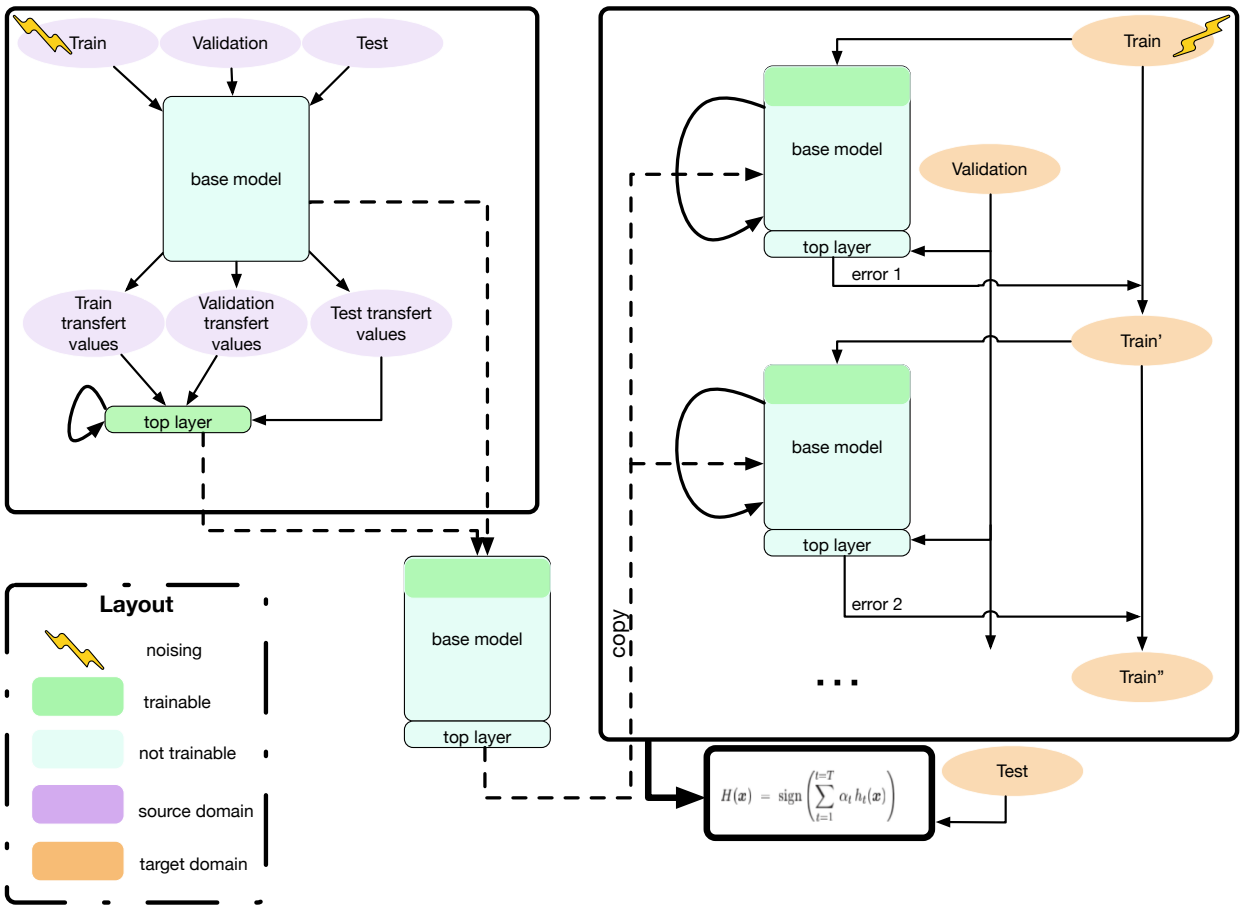

FIGURE 2 – Déroulement de la méthode TransBoost dans le cadre de ce projet

3.1 Présentation des données

Nous avons choisis le jeu de données CIFAR-10 composé d'images RGB 32x32. Celui-ci est composé de 60000 images se ventilant en 10 classes (avion,automobile,oiseau,chat,cerf,chien,grenouille,cheval,bateau,camion). Les motivations pour ce choix sont :

— qu'il s'agit d'un jeu de données de référence usuel dans le milieu
— que la faible de taille des images limite le volume des données à manipuler
— le faible nombre de classes permet néanmoins de constituer des couples plus ou moins ardus (avion/chat vs chien/chat)

Les données sont déjà reparties en 3 ensembles : entraînement, validation et test. Pour pallier au faible nombre des données dans la base d'entraînement on introduira du bruit dans celles-ci pour éviter tout sur-apprentissage. Ce bruit consiste en des rotations, zooms, déformations aléatoires. Attention, aucun bruitage n'est appliqué aux ensemble de validation et de test. En effet, on veut ceux-ci les plus représentatifs possible des images "réelles" pour éprouver le modèle.

Pour constituer les ensembles source et cible, il s'agit dans les faits de simple paramètres d'entrées qui permettent de changer les classes de manière aisée (exemple : classes_source = ['dog','truck'], classes_target = ['deer','horse']). A noter que le bruitage des sets d'entraînement est appliqué pour les deux domaines.

3.2 Construction d'un classificateur binaire fort sur le domaine source

Afin de pouvoir mettre en application l'idée du TransBoost, nous devons tout d'abord obtenir un classificateur binaire fort sur une tâche et aussi bon que le hasard sur une seconde. Pour cela, nous devons d'abord choisir quel sera notre modèle de base. On entend par modèle de base un réseau profond déjà entraîné et sans les dernières couches (couches connectées et softmax). `Keras` offre de nombreuses possibilités de modèles (voir ce lien). Le choix s'est porté sur le modèle Xception, d'une part pour la qualité des valeurs de transferts qu'il produit et d'autre part pour le temps qu'il faut pour générer ces valeurs de transfert. On appelle valeurs de transfert les valeurs des fonctions d'activation de la dernière couche du modèle de base pour un ensemble d'images.

Par soucis de parcimonie en temps de calcul, on fait passer l'ensemble des sets d'images dans le modèle de base une seule fois. On génère ainsi les valeurs de transfert qui serviront à l'entraînement de la dernière couche.
L'architecture de la dernière couche (plus exactement du dernier bloc) est :

— une couche entièrement connectée de taille 1024, activation "relu"
— un dropout à 50 %
— une couche entièrement connectée de taille 512, activation "relu"
— un dropout à 50 %
— un couche de taille 1, activation "sigmoïd"

Le choix de cette architecture s'est fait de manière empirique.

Comme pour le bruitage des images, le dropout est ici pour prévenir le sur-apprentissage compte tenu du faible nombre d'images présentées. Une fois la dernière couche entraînée et répondant aux caractéristiques désirées, on assemble le modèle de base et ladite dernière couche en un modèle complet.

3.3 Mise en pratique du TransBoost

Une fois que nous avons constitué notre classificateur fort, il faut voir la suite comme l'application classique d'Adaboost. La seule nuance est que, au lieu de repartir d'un classificateur "from scratch" on va cette fois partir d'un classificateur partiellement entraîné pour constituer notre projecteur faible.

On cherche, non pas comme dans l'apprentissage par transfert classique avec les CNN à réapprendre une nouvelle séparatrice linéaire en entraînant la dernière couche pour une nouvelle tâche. On cherche à faire rentrer pour la nouvelle tâche les points des bons cotés de la séparatrice pour la tâche précédente en cherchant des descripteurs de faibles niveau correspondants (i.e. en ré-entraînant les premiers blocs du réseaux de neurones profond) pour que cela soit réalisé.

Pour ce faire, on prend le modèle que nous avions à l'étape précédente, on dégèle les premières couches du modèle de base et on gèle tout le reste. A ce stade deux approches sont possibles : soit réinitialiser les poids des couches dégelées soit conserver les poids. Il se peut qu'en réinitialisant le poids de ces couches avec des descripteurs de faible niveau ne permette pas à l'algorithme de converger sur si peu de données. La réponse viendra des expérimentations. Dans les deux cas, le modèle résultant de l'étape citée servira de classificateur à entraîner lors des étapes de boosting.

Trois paramètres importants sont à chercher selon un compromis performances/coût calculatoire :

— la force des classificateurs (seuil de précision) à chaque étape de boosting
— le nombre de blocs convolutifs à entraîner
— le nombre de projecteurs faibles entraînés

La suite est celle de l'algorithme Adaboost (voir 1). A noter que l'erreur totale du modèle est mesurée sur le set de validation. Par contre le modèle est bien entraîné sur le set d'entraînement pondéré, et la pondération des points basée sur les prédictions faites sur ce même set d'entraînement.

A chaque étape, l'entraînement d'un projecteur s'arrête lorsqu'un seuil de précision est atteint (avec une limite d'un grand nombre d'epochs).

input : $\mathcal{X}_\mathcal{S} \to \mathcal{Y}_\mathcal{S}$: l'hypothèse source

$\mathcal{S}_\mathcal{T} = \{(\mathcal{X}_\mathrm{i}^\mathcal{T}, \mathcal{Y}_\mathrm{i}^\mathcal{T})\}_{1 \leq i \leq m}$: l'ensemble d'entraînement cible

Initialisation : de la distribution sur le jeu d'entraînement :$D_1(i) = 1/m \ for \ i = 1, \cdots, m$;

for $n = 1, \cdots, N$ **do**

Trouver une projection $\pi_i : \mathcal{X}_\mathcal{T} \to \mathcal{X}_\mathcal{S}$ tq. $h_S(\pi_i(.))$ soit meilleure que le hasard sur $D_n(\mathcal{S}_\mathcal{T})$;

Soit ϵ_n le taux d'erreur de $h_S(\pi_i(.))$ sur $D_n(\mathcal{S}_\mathcal{T})$: $\epsilon_n = P_{i\ D_n}[h_S(\pi_n(x_i)) \neq y_i]$(avec $\epsilon_n < 0.5$);

Calculer $\alpha_i = \frac{1}{2} log_2(\frac{1-\epsilon_i}{\epsilon_i})$;

Mettre a jour : **for** $i = 1, \cdots, m$ **do**

$$D_{n+1}(i) = \frac{D_n(i)}{Z_n} \times \begin{cases} e^{-\alpha_n} \ si \ h_\mathcal{S}(\pi_n(x_i^\mathcal{T})) = y_i^\mathcal{T} \\ e^{\alpha_n} \ si \ h_\mathcal{S}(\pi_n(x_i^\mathcal{T})) \neq y_i^\mathcal{T} \end{cases}$$
$$= \frac{D_n(i) exp(-\alpha_n y_i^{(\mathcal{T})} h_\mathcal{S}(\pi_n(x^{(\mathcal{T})})))}{Z_n}$$

Ou Z_n est un facteur de normalisation tq. D_{n+1} soit une distribution de $\mathcal{S}_\mathcal{T}$;

end

end

output: L'hypothese finale $H_\mathcal{T} : \mathcal{X}_\mathcal{T} \to \mathcal{Y}_\mathcal{T}$:

$$H_\mathcal{T}(x_\mathcal{T}) = signe\{\sum_{n=1}^{N} \alpha_n h_\mathcal{S}(\pi_n(x^\mathcal{T}))\}$$

Algorithm 1: Algorithme Transboost

3.4 Difficultés rencontrées

Nous avons pris la décision initiale de travailler dans un environnement purement `Tensor Flow`, puisque le modèle que nous avions choisi était disponible dans cette librairie. Cependant dès qu'il a été temps de modifier la structure du modèle (pour avoir une couche de sortie binaire par exemple) ou qu'il a fallu geler l'entraînement de certaines couches l'utilisation de `Tensor Flow` est devenue très compliquée. En effet l'objet du modèle était introuvable et il fallait modifier un graphe ce qui nous a posé beaucoup de problèmes. C'est pour cela que nous avons décidé d'utiliser la librairie `Keras` qui fonctionne comme surcouche de `Tensor Flow` ce qui nous a permis d'utiliser les modèles disponible dans `Tensor Flow` mais en ayant des outils et une syntaxe plus claire et plus simple d'utilisation, permettant des temps de développement beaucoup plus courts.

Nous avons aussi rencontré beaucoup de problèmes de ressources machine, puisque les machines physiques et virtuelles auxquelles nous avions accès présentaient des performances limitées. Ceci a engendré des temps d'exécution se comptant en dizaine d'heures et même en jours dans certains cas et aboutissant souvent a des erreurs de mémoire. Il est donc nécessaire d'avoir accès à une machine performante qui nous permettra débugger le programme et le perfectionner sans attendre des périodes très longues.

Au fur et a mesure de l'avancement du projet et avec la mise en disponibilité d'une machine plus puissante nous avons rencontré d'autres problèmes, en effet avec la modification constante du

réseau, réinitialisation des poids et sauvegarde nous avons décidé d'enregistrer les objets modèles dans une liste a chaque fin d'itération de boosting et de supprimer l'objet modèle actif avec la fonction `del` de Python. Cependant cette fonction ne garantit pas la suppression immédiate de l'objet à cause du système de gestion de mémoire vive de Python et même en forçant l'exécution du "garbage collector" du langage C (qui est utilise pour gérer la mémoire en Python) nous observions des comportements étranges avec des modèles qui n'étaient pas entièrement supprimés et donc des poids qui ne changeaient pas entre chaque étape du boosting.

Nous avons au final décide d'enregistrer les modèles a chaque étape sur le disque dur de la machine avec des fonctions de `Keras` qui permettent d'une part de sauvegarder l'architecture du modèle dans un fichier *.json* et les poids dans un autre. Etant donné que l'architecture du réseau ne change pas il est possible de ne l'enregistrer qu'une seule fois après l'entraînement du modèle sur l'espace source. Cette architecture indique également quelles couches sont entraînable lors de la phase de boosting, et par la suite a chaque fois que l'on souhaite enregistrer le modèle on peut se contenter d'enregistrer uniquement les poids.

Une fois que le modèle est sauvé sur le disque dur on peut appeler une fonction de `Tensor Flow` qui permet d'effacer tous les objets de la session `Tensor Flow` et ainsi s'assurer que plus aucun objet modèle précédemment entraîne ne persiste et ne perturbe l'entraînement de nouveaux modèles. Pour pouvoir entraîner un nouveau modèle il suffit, a chaque début d'itération de boosting de créé un réseau a partir de l'architecture sauvée sur disque et d'y charger les poids du modèle de base.

Une fois que tous les projecteurs ont été entraînés on peut effectuer les prédictions en chargeant séquentiellement chaque modèle (en chargeant l'architecture et les poids) et d'enregistrer ses prédictions avant de supprimer le modèle et de charger le suivant. A la fin la liste de prédiction est pondérée avec les α calculés lors du boosting et on obtient la prédiction finale issue du boosting

A priori cette méthode de tout enregistrer sur le disque avant de tout supprimer et tout recharger peut paraître lourde et pas forcement nécessaire, cependant ces étapes supplémentaires et le temps d'accès beaucoup plus long du disque dur par rapport a la mémoire vive sont de petits prix a payer par rapport a la garantie que l'on entraîne bien ce que l'on veut entraîner et qu'il ne peut pas y avoir de "contamination" entre les étapes du boosting.

4 Comparaisons critiques portant sur la méthode transboost

4.1 Projecteurs faibles vs. projecteurs forts

On veut comparer l'approche TransBoost avec faisant varier la force des projecteurs aux extrêmes. Si l'on obtient des performance supérieures avec un seul projecteur fort (i.e. les premiers groupes de couches convolutives entraînées aux maximum) alors on ne peut pas montrer un intérêt de travailler avec une multitudes de projecteurs faibles en boosting dans ce cadre précis.

4.2 Projecteurs vs. nouveaux classificateurs

Comme on le voit, la méthode du TransBoost appliquée avec le modèle complet (base et dernier bloc) est très gourmande en temps et en espace. En effet, après avoir entraîné un grand nombre de classificateurs il faut également stocker tous ceux-ci à fin de pouvoir réaliser les classification de nouveau points selon l'hypothèse finale. En fait, on peut être plus économe en espace. En effet, il suffit de la connaissance pour reconstruire tous les modèles issus du boosting :

— du modèle complet non modifié
— uniquement pour chaque étape de boosting du poids des couches modifiées

On souhaite mettre en compétitions deux approches : le TransBoost et une méthode de boosting classique.

On peut atteindre un seuil de précision relativement bas à chaque étape (de l'ordre de 0.7) simplement à l'aide d'un petit réseau convolutif initialisé (quelques blocs). Bien qu'avec l'économie en espace citée précédemment il n'y ait pas beaucoup de différence, l'économie en temps de calcul est bien là. En effet pour chaque prédiction à faire, le passage dans le petit CNN suffit. Tandis qu'à chaque étape avec le modèle de base augmenté du dernier bloc il faille calculer les activations dans toute la partie supérieure gelée du réseau avant de "backpropager" l'erreur.

5 Conditions et méthodes d'expérimentation

Toutes les expérimentations ont été faites sur une machine à 8 cœurs, 30 Go de mémoire RAM et 11 Go de GPU. Il s'agit d'un serveur Ubuntu avec Keras GPU. Nous avons utilisé le service Google Cloud. On interagit en lignes de commandes : on lance le programme dans un screen qui écrit un fichier *.log*. On peut alors détacher le screen et quitter la machine virtuelle et revenir consulter le log en cours de construction quand l'on souhaite.

Le programme est construit tel que suit : la configuration à faire tourner est spécifiée dans un fichier *.json* à part (cf. 1). De par la nature des expérimentations qui étaient prévues, le programme permet également d'appeler une liste de configurations pour par exemple tester différents seuils de projecteurs.

Listing 1 – Exemple de configuration

```
{
  "models_path" : "models",
  "models_weights_path" : "models_weights",
  "path_to_best_model" : "best_top_model.hdf5",
  "threshold" : 0.65,
  "proba_threshold" : 0.5,
  "transformation_ratio" : 0.05,
  "originalSize" : 32,
  "resizeFactor" : 5,
  "batch_size_source" : 10,
  "batch_size_target" : 10,
  "epochs_source" : 1000,
  "epochs_target" : 1000,
  "classes_source" : ["dog","truck"],
  "classes_target" : ["deer","horse"],
  "layerLimit" : 15,
  "times" : 1,
  "lr_source" : 0.0001,
  "lr_target" : 0.0001,
  "step" : 3,
  "recompute_transfer_values" : false,
  "train_top_model" : false,
  "reinitialize_bottom_layers" : false,
  "bigNet" : true,
  "verbose" : true
}
```

6 Resultats

6.1 Construction du classificateur binaire

Dans les faits, pour la base d'apprentissage "chien/camion", l'algorithme arrive à 98.9% de précision sur le set de test en seulement 2 epochs. D'un autre côté ce même modèle entraîné sur le dataset précédent a un précision de 50% en prédiction sur le dataset "deer/horse", ce qui répond bien à ce que l'on recherchait.

6.2 Boosting classique sur petit CNN

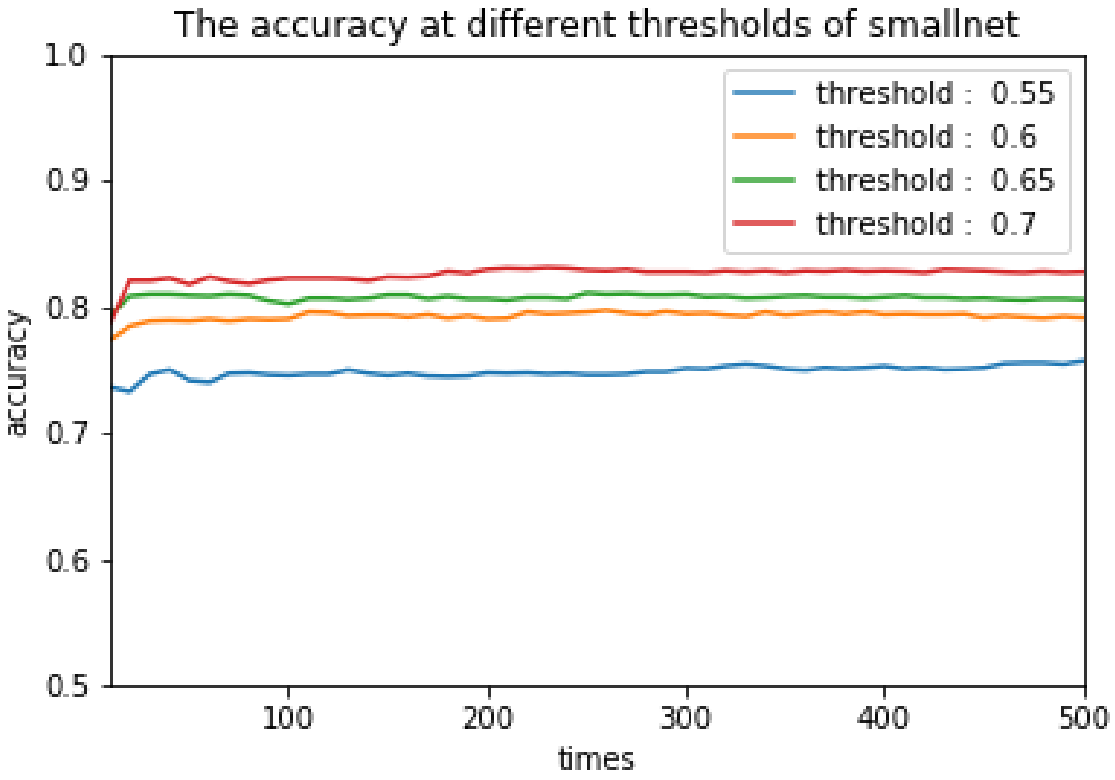

FIGURE 3 – Évolution de la précision de l'algorithme de Boosting avec des petits réseaux de convolution (sans transfert)

On a construit un simple réseau de neurones de convolution qui apprend directement la domaine cible avec l'algorithme de Boosting, sans l'apprentissage de transfert. Son architecture est :

- deux couches de convolution avec 32 filtres de taille 3*3 chacun, activation "relu", max-pooling de taille 2*2
- une couche de convolution avec 64 filtres de taille 3*3 chacun, activation "relu", max-pooling de taille 2*2
- une couche flatten
- une couche entièrement connectée de taille 64, activation "relu"
- un dropout à 50 %
- une couche entièrement connectée de taille 34, activation "relu"
- un dropout à 50 %
- un couche de sortie avec une activation sigmoïde

On utilise ce petit réseau de convolution à la place des projecteurs dans l'algorithme de boosting, et on mesure la précision de classification sur le set de test à différentes itérations du boosting. Les résultats de cette analyse sont visibles sur la figure 3. Les mesures ont été effectuées pour plusieurs seuils de précision d'entraînement lors du boosting.

Le set à apprendre est relativement difficile : il s'agit de "deer/horse".Ceci montre que l'algorithme de boosting a été correctement implémenté (au moins 10 points de gain de précision par rapport au seuil des projecteurs faibles). On peut voir que plus le projecteur est fort, plus la précision augmente. Au bout d'une dizaine de projecteurs on a ensuite une évolution très lente de performances.

6.3 Méthode transboost

Après plusieurs essais (learning rate, optimiseur, réinitialisation des couches modifiées ou non ...) l'algorithme ne converge pas ou bien pas dans un temps raisonnable. Pour que l'algorithme marche dans le boosting, celui ci doit au moins converger au seuil à la première itération assurant ainsi une performance supérieure ou égale lors du boosting.

Dans les conditions présentées (30 Go de RAM, 11 Go de GPU et 8 coeurs) l'algorithme ne converge pas à un seuil de 55% sur le set de validation au bout de 4h et ne peut dépasser 65 % sur le set d'entraînement. Ainsi, même si l'on vise un seuil de 70% en resubstitution pour atteindre entre 55% et 60% sur le set de validation, on ne pourra pas atteindre ces valeurs dans le 4h. Par conséquent on n'obtient que des projecteurs équivalents au hasard.

Même si l'algorithme arrivait à converger, s'il faut 4h par projecteur avec de telles ressources, on voit que cela est prohibitif. On peut expliquer cela par le fait qu'on déstabilise sur très peu de données les couches basses qui ont appris des descripteurs de faible niveau (les plus longs à apprendre). Il faut mettre ceci en regard du très peu de temps et de ressources calculatoires que nécessite le transfert learning classique avec de très bonnes performances.

7 Expérimentations non réalisées

Expérimentations impossibles du fait que l'algorithme ne converge pas (ou du moins en un temps raisonnable).

7.1 Influence des domaines source et cible

Peut t-on partir d'un domaine source simple (ex : "dog/truck") pour aller vers un domaine cible plus compliqué (ex :"deer/horse") ? Dans notre cas, nous souhaitons d'étudier le cas compliqué. Mais on peut aussi essayer d'autres cas. Notre objectif est vérifier la méthode de transboost, comme le domaine cible est assez compliqué, on n'a pas de preuve qu'il ne marche pas. Par contre, on pourra établir de modèle à partir d'un domaine source compliqué (ex :"deer/horse"),et l'appliquer sur le domaine cible (ex :"dog/truck")

7.2 Influence des hyper-paramètres : recherche d'un optimum précision/coût calculatoire

— Influence de la force des projecteurs

— Influence du nombre de projecteurs

— Influence du nombre de blocs entraînés.*(Dans l'idéal on veut en modifier le moins possible pour atteindre au plus vite le seuil de précision)*

7.3 Transboost vs classical transfer learning

Pour un domaine cible difficile (deer/horse), on arrive sans fine-tuning à une précision de 92.6% en moins de 10 epochs. Ce résultat peut être grandement amélioré à l'aide d'un "fine tuning" des derniers blocs convolutifs du modèle de base.

8 Conclusion

Si l'idée du transboost semble prometteuse dans le cadre des séries temporelles, l'application à la classification d'images avec ses problématiques inhérentes (taille de données et difficulté de classification) ne semble pas probante.

Comme on le présageait l'implémentation transboost nécessite de capacité de calcul très importantes. Le fait que l'algorithme ne converge pas (sauf erreur de notre part) ou bien pas dans un temps raisonnable ne justifie pas de poursuivre les expérimentations telles que nous les avons abordées. On pourrait par exemple créer un réseau à part entière qui sert de projecteur avec entrée et en sortie des images. Cependant, là encore il faudrait parcourir tout le réseau convolutionnel de base à chaque fois pour remonter l'erreur.

Bibliographie

[1] Francois Chollet. Building powerful image classification models using very little data, 2016.

[2] Murena P.A. Olivier R. Cornuejols A., Akkoyunlu S. Transfer learning by boosting projections from the target domain to the source domain.

[3] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55 :119–139, 1997.

Appendices

A Code

Le code est disponible sur gitlab. avec le compte d'utlisateur suivant :

— email : transboostproject@gmail.com

— password : transboostAPT2018

B Schéma du programme

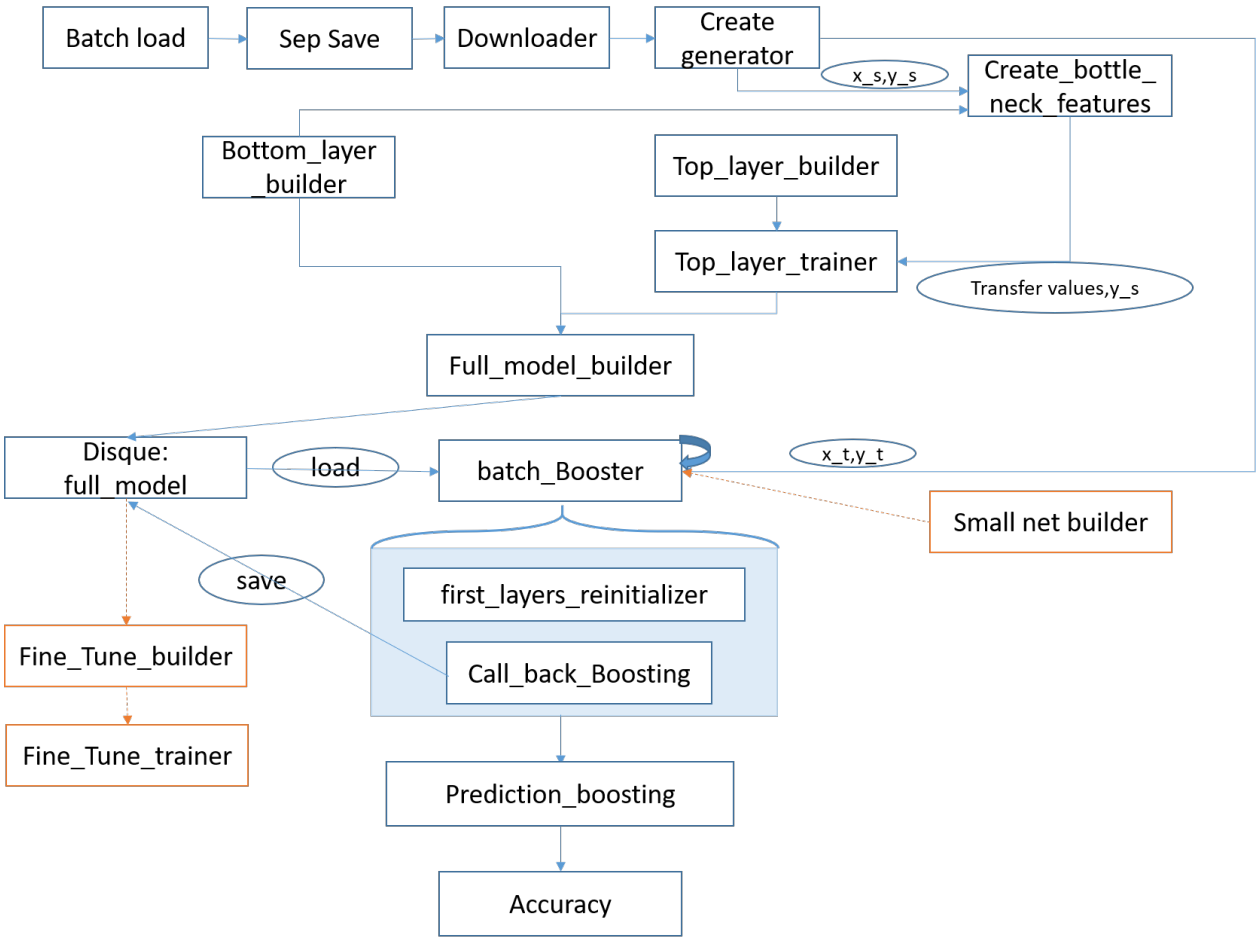

FIGURE 4 – schéma du programme

Descriptif des fonctions

Batch_loader : Prend les images d'un batch de CIFAR-10 et les enregistre dans un dictionnaire avec comme clé la classe de l'image.

Sep_saver : Parcourt le dictionnaire et enregistre les images de chaque classe dans un dossier correspondant.

Downloader : Télécharge les batches de CIFAR-10 et exécute les deux fonctions précédentes pour chaque batch d'entraînement, de test et de validation.

Create_generator : Crée des objets de type générateur qui contiennent des batches d'images avec du bruit.

Bottom_layer_builder : Charge et construit le modèle sans dernière couche (Xception dans ce cas).

Create_bottleneck_features : Calcule les valeurs de transfert pour chaque image des jeux d'entraînement, de test et de validation passant par le modèle issu de la fonction précédente.

Top_layer_builder : Construit la couche de sortie binaire.

Top_layer_trainer : Entraîne la dernière couche avec les valeurs de transfert.

Full_model_builder : Assemble le modèle sans dernière couche et la dernière couche nouvellement entraînée.

first_layers_reinitilizer : Gèle les poids des n dernières couches pour effectuer le boosting.

Callback_boosting : est appelé à chaque epoch de l'entraînement du modèle et si la précision est supérieure à un seuil α arrête l'entraînement et sauvegarde le modèle sur le disque dur.

batchBooster : Applique l'algorithme adaboost au réseau. chaque modèle entraîne faisant office de projecteur faible est construit puis entraîne en appelant les deux fonctions précédentes n fois.

Prediction_boosting : Effectue une prédiction sur la classe de l'image présentée en prenant en compte les modèles entraînés dans la fonction précédente.

Accuracy : Évalue la précision des prédictions.

small net builder : Construit un petit CNN avec 3 couches convolutives et 3 couches entièrement connectées avec décrochement

fine_tune_builder : Construit un modèle de fine tuning. Il prend les valeurs en utilisant le modèle précédent de domaine de source.

fine_tune_trainer : Entraîne le modèle de fine tuning.