

Projet Transboost

Luc Blassel, Romain Gautron, Xue Bai, Yifei Fan, Xuefan Xu

31 janvier 2018

Table des matières

1	Contexte	2
2	Transboost pour la classification d’images : principe	3
3	Application	4
3.1	Présentation des données	4
3.2	Construction d’un classifieur binaire fort sur le domaine source	5
3.3	Mise en pratique du TransBoost	5
3.4	Difficultés rencontrées	7
4	Comparaisons critiques portant sur la méthode transboost	7
4.1	Projecteurs faibles vs. projecteurs forts	7
4.2	Projecteurs vs. nouveaux classifieurs	8
5	Resultats	8
5.1	Construction du classifieur binaire	8
5.2	Boosting classique sur petit CNN	8
6	Experimentations a venir	8
6.1	Comportement de l’algorithme Transboost	8
6.1.1	Influence des domaines source et cible	8
6.1.2	Influence des hyper-paramètres : recherche d’un optimum précision/coût calculatoire	8
6.1.3	Projecteurs faibles vs. projecteurs forts	9
6.1.4	Projecteurs vs. nouveaux classifieurs	9
	Bibliographie	9
	Appendices	10
A	Code	10
B	Schéma du programme	10

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 entraîné 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 frontières 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.

Classical transfert learning VS Transboost

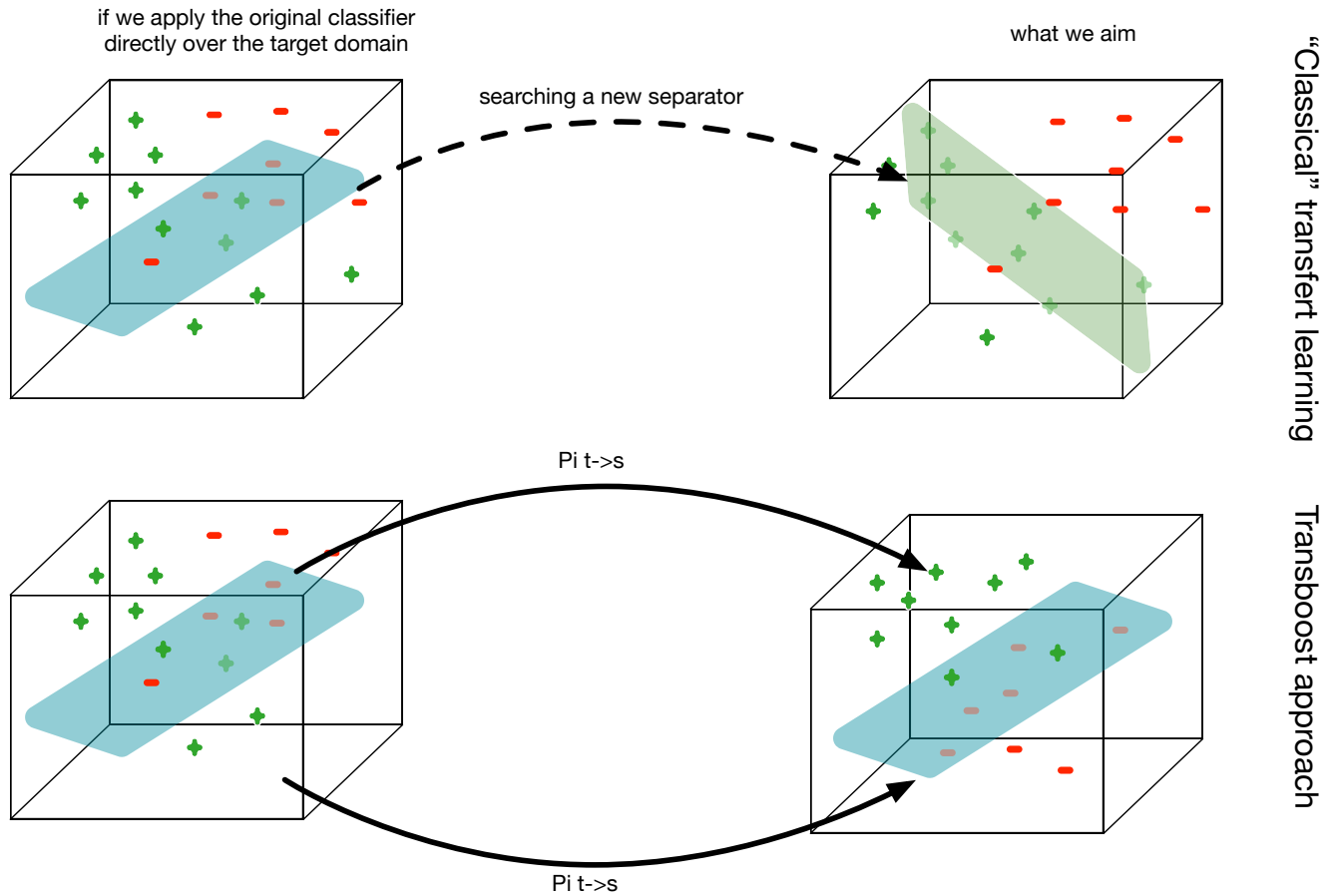


FIGURE 1 – Différences entre l'approche classique et l'approche TransBoost

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 à 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 un 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). À 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 de ces projecteurs faibles.

3 Application

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

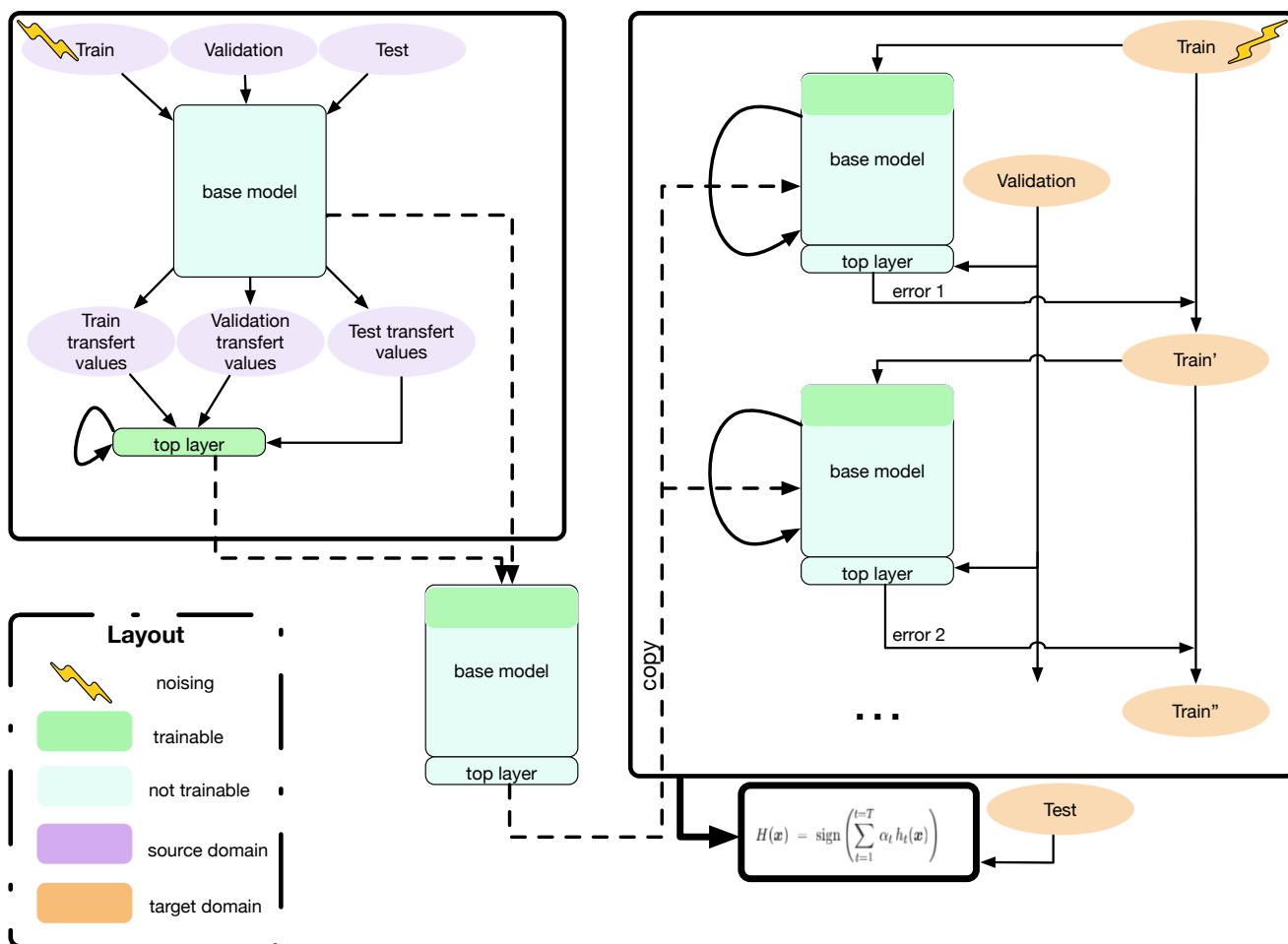


FIGURE 2 – Déroulement de la méthode TransBoost dans le cadre de ce projet

3.1 Présentation des données

Nous avons choisis le data set 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 dataset 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 ardues (avion/-chat vs chien/chat)

Les données sont déjà réparties 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 classifieur binaire fort sur le domaine source

Afin de pouvoir mettre en application l’idée du TransBoost, nous devons tout d’abord obtenir un classifieur 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 l’architecture s’est fait de manière empirique.

Comme pour le bruitage des images, le dropout est ici pour prévenir le surapprentissage 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 classifieur fort, il faut voir la suite comme l’application classique d’Adaboost. La seule nuance est que, au lieu de repartir d’un classifieur “from scratch” on va cette fois partir d’un classifieur partiellement entraîné pour constituer notre projecteur faible.

On cherche, non pas comme dans le transfert learning 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 avons 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 classifieur à entraîner lors des étapes de boosting.

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

- la force des classifieurs (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 : $\mathcal{X}_{\mathcal{S}} \rightarrow \mathcal{Y}_{\mathcal{S}}$: l'hypothese source
 $\mathcal{S}_{\mathcal{T}} = \{(\mathcal{X}_i^{\mathcal{T}}, \mathcal{Y}_i^{\mathcal{T}})\}_{1 \leq i \leq m}$: l'ensemble d'entraînement cible
Initialisation : de la distribution sur le jeu d'entraînement : $D_1(i) = 1/m$ for $i = 1, \dots, m$;
for $n = 1, \dots, N$ **do**
 Trouver une projection $\pi_i : \mathcal{X}_{\mathcal{T}} \rightarrow \mathcal{X}_{\mathcal{S}}$ tq. $h_{\mathcal{S}}(\pi_i(.))$ soit meilleure que le hasard sur $D_n(\mathcal{S}_{\mathcal{T}})$;
 Soit ϵ_n le taux d'erreur de $h_{\mathcal{S}}(\pi_i(.))$ sur $D_n(\mathcal{S}_{\mathcal{T}})$: $\epsilon_n = P_{i \sim D_n}[h_{\mathcal{S}}(\pi_n(x_i)) \neq y_i]$ (avec $\epsilon_n < 0.5$);
 Calculer $\alpha_i = \frac{1}{2} \log_2(\frac{1-\epsilon_i}{\epsilon_i})$;
 Mettre a jour : **for** $i = 1, \dots, m$ **do**

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

$$= \frac{D_n(i) \exp(-\alpha_n y_i^{(\mathcal{T})} h_{\mathcal{S}}(\pi_n(x^{\mathcal{T}})))}{Z_n}$$

 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 $H_{\mathcal{T}} : \mathcal{X}_{\mathcal{T}} \rightarrow \mathcal{Y}_{\mathcal{T}}$:

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

Algorithm 1: Algorithme Transboost

3.4 Difficultés rencontrées

Nous avons fait la décision initiale de travailler dans un environnement purement **Tensor Flow**, puisque le modèle que nous avons 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 avons 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 débbugger le programme et le perfectionner sans attendre des périodes très longues.

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 classifieurs

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 classifieurs 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 Resultats

5.1 Construction du classifieur 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 une précision de 50% en prédiction sur le dataset "deer/horse", ce qui répond bien à ce que l'on recherchait.

5.2 Boosting classique sur petit CNN

Avec un petit nombre de projecteurs (12) et un petit CNN (3 blocs de convolution) on arrive à une précision de 76.8% en test sur le dataset "horse/deer". Expérience à renouveler avec un 100 projecteurs dès qu'on aura la puissance de calcul ad hoc.

6 Experimentations a venir

6.1 Comportement de l'algorithme Transboost

6.1.1 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") ?

Bien sur dans les temps venant il faudra tester les critiques soulevées plus haut, en comparant les performances des projecteurs faibles contre un projecteur fort, et la performance de ces projecteurs par rapport à celle d'un classifieur nouvellement entraîné.

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

Influence de la force des projecteurs : A venir.

Influence du nombre de projecteurs A venir.

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

6.1.3 Projecteurs faibles vs. projecteurs forts

A venir.

6.1.4 Projecteurs vs. nouveaux classifieurs

A venir.

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

[Code sur gitlab.](#)

B Schéma du programme

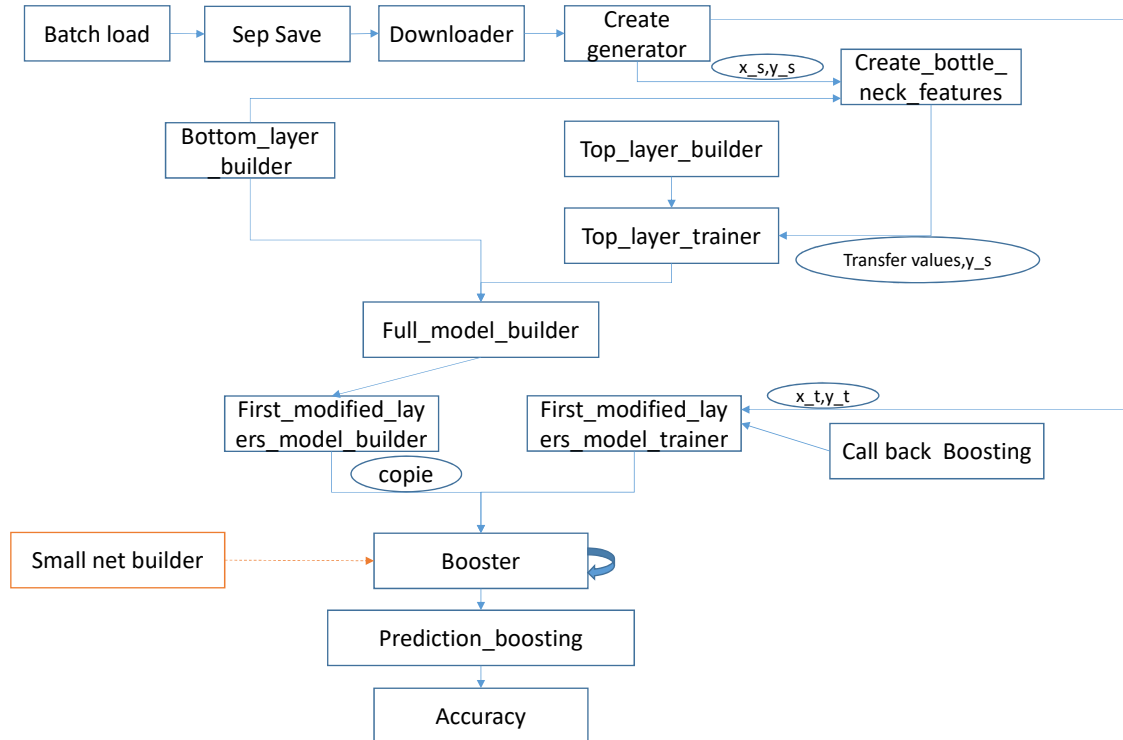


FIGURE 3 – schema du programme

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 :

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

First_modified_layers_model_trainer : Entraîne le modèle partiellement gelé sur le jeu d'entraînement.

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.

Booster : 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.