Transboost Project

Luc Blassel, Romain Gautron

March 13th 2018

Contents

1 context

Classical supervised learning requires a large amount of labeled data and, in certain cases, a very long amount of time to ba able to train reliable models. This is not always possible in a 'real world' setting, it could be that it is not possible to obtain sufficient data to train models or that the required time for training is so long that it becomes practically unfeasible with conventional resources. Transfer learning can help us solve this type of problem.

Transfer learning allows for a transfer of acquired knowledge from a source domain, ideally with a large amount of well labeled data, towards a target domain. With this approach, portions of a pre-trained model can be reused in a new model. The advantage is two-fold: training times are shortened and the amount of training data needed is smaller. This method is considered by some as the next motor of progress in automatic learning after supervised learning.

The TransBoost method [2], proposed by Antoine Cornuéjols et al., outlines a different approach than "classical" transfer learning. Where the latter adapts the hypothesis learned on the source space, to the target space, TransBoost does the opposite. The hypothesis is learned on the source space and the data points from the target space are projected in the source space to make use of the source hypothesis without the need to relearn it.

This way no new frontiers between points are learned, the target points are correctly projected at the right location in the source domain. This projection is done within a boosting algorithm, allowing for the building of a strong projector by combining several weak projectors. This difference in approaching the issue can be seen in Figure 1.

The TransBoost method has been tested on classification of incomplete time series data with real success, outperforming other methods used for this problem. However, image classification being the standard measure right now, this project intends to adapt the TransBoost method toimage classification unsing deep convolutionnal neural networks (CNN). In this project only binary classification is tried.

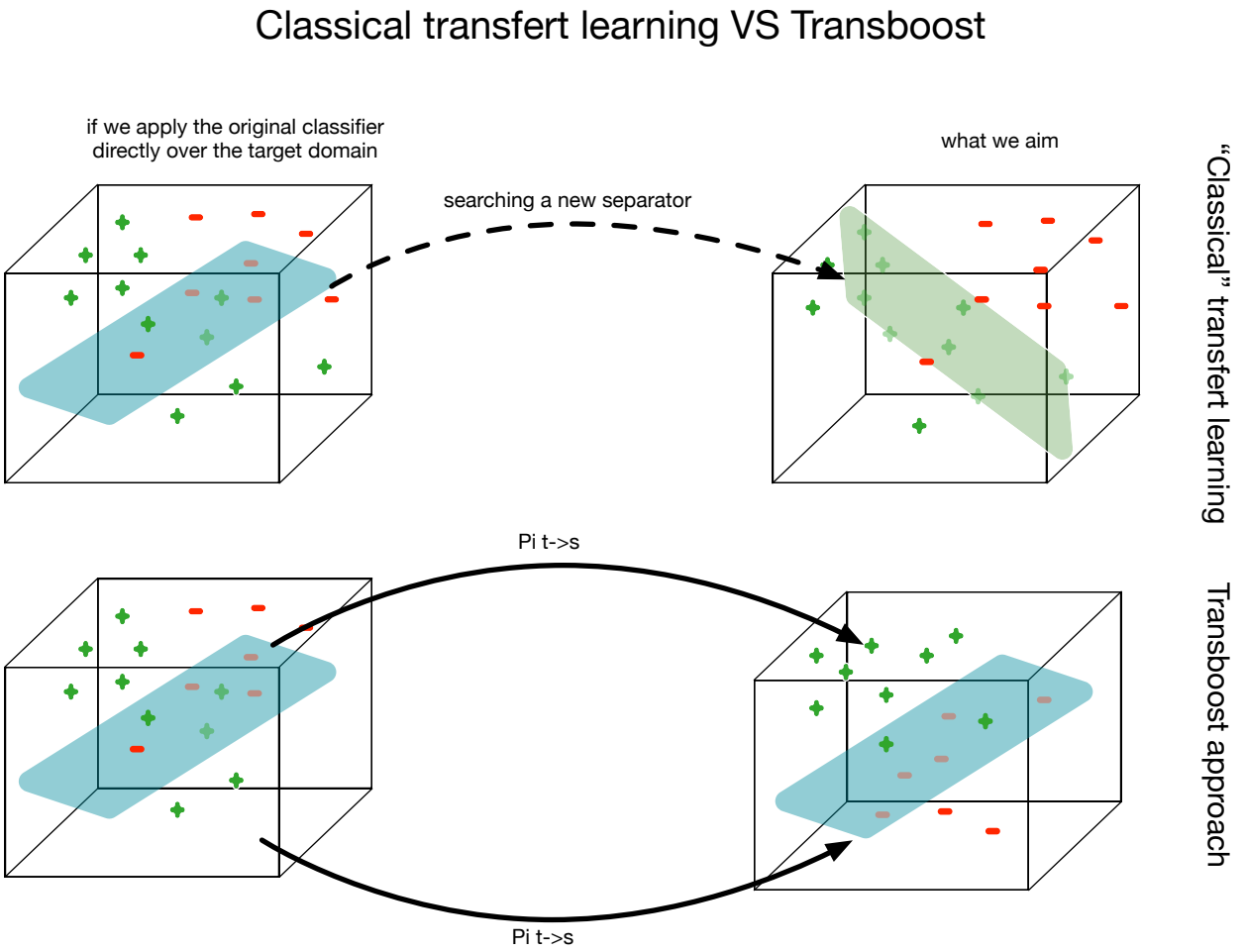

Figure 1: Differences between "classical" transfer learning and the TransBoost method

2 Transboost for image classification: working principles

Implementation of the TransBoost method for image classification requires consideration of several issues. First, the high dimensionality of image data induces the use of very heavy methods sur as CNNs. Second, how to project points of source to target domains in the case of images?

The choice has been made, to modify the lower layers of the source network to classify target images. Therefore, the lower layers of the existing network will find the right low-level descriptors so that the new task can be done. However hyper-parameter optimization will have to be done.

We could also have implemented a separate "projector" network that would take the source domain images and transform them in projected images. Howver, even though potentially visually interesting, it would have required larger computing power (more back-propagation) and was not implemented.

The construction of a projector is as follows:

- A pre-trained CNN is obtained. It is very performant for one source domain binary classification task and random for the target domain task. Since pre-trained models are available with a large number of outputs (general image classifiers can recognise hundreds of classes), it is necessary to modify the last layer and retrain it to have a binary output.
- The higher layers of the modified CNN are frozen, leaving only the lower layers trainable
- This partially frozen CNN is retrained to obtain a weak projector, using the accuracy metric as a stopping criterion. accuracy has been chosen because it is meaningful and pertinent for well-balanced data-sets.

A set of weak projectors, specialised on the error of the previous projector, is therefore iteratively built using the AdaBoost algorithm [3]. Our final hypothesis on the target domain in a linear combination of these weak projectors.

3 Application

In this section, Figure 2 wil be used as reference.

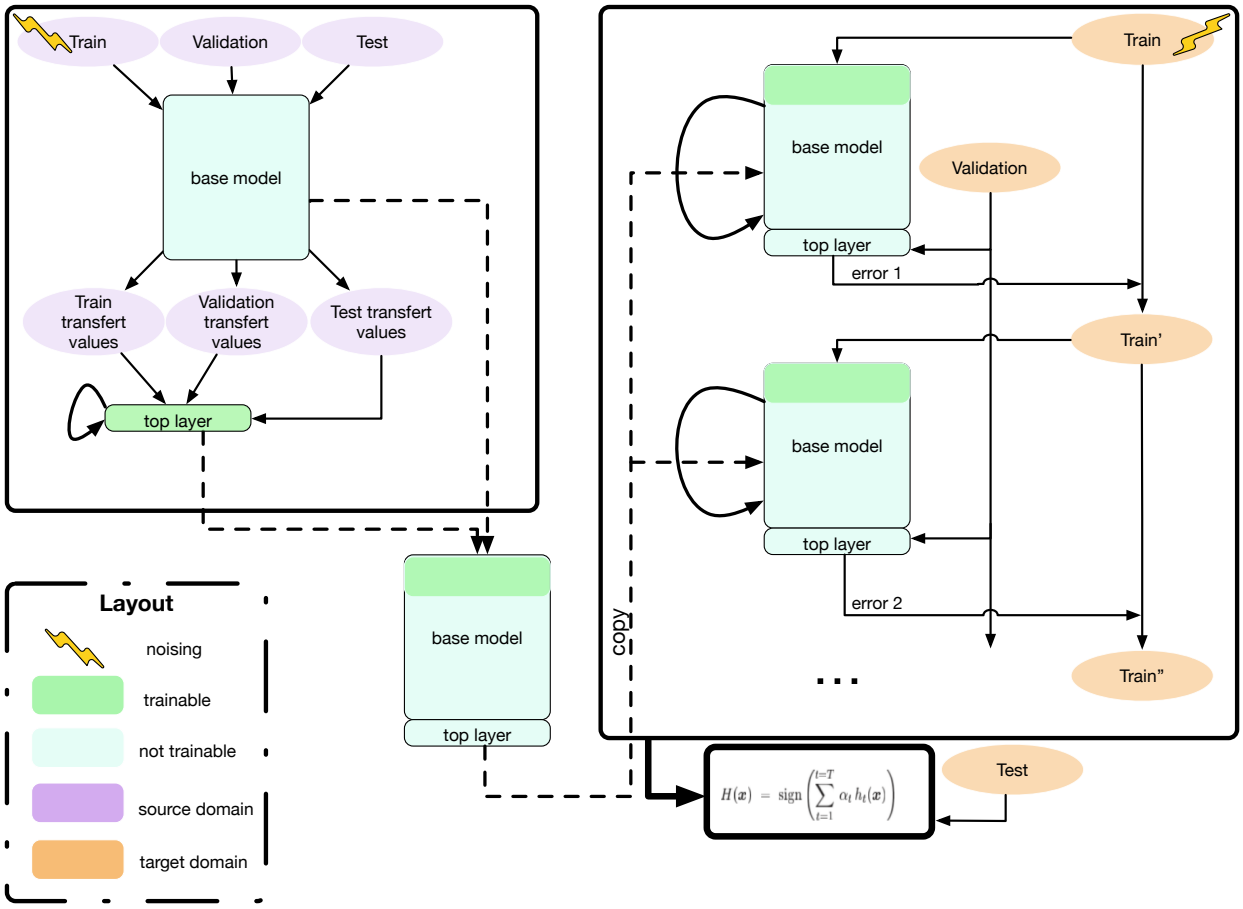

Figure 2: Execution steps of the TransBoost method, within this project

3.1 Our Data

We have chosen the CIFAR-10 image dataset. It comtains 60000 RGB images 32 by 32 pixles. Thes images are separated in 10 classes (airplane, car, bird, cat, deer, dog, frog, horse, ship and truck). The reasons for this choice are as follows:

- It is a reference dataset in image classification.
- The small size of the images limits the size of data to manipulate
- in spite of th esmall number of classes it is possible to make pairs of varying differentiability (plane/cat vs deer/horse)

Data is separated into 3 sets: training, validation and test. To counteract the low number of images in the training set, images are noised to avoid over-fitting. This noise consists of rotation, zooming and random deformations. No noisingis done for the test and validation sets. We want images in these sets to be as reprensatative as possible of "real world" images to test the model.

The target and source domains are just two different binary image sets *(eg. source = ['dog', 'truck'] and target = ['deer', 'horse']).* This can be easily controlled by input parameters. It is important to note that noising of training data is applied to both domains.

3.2 Elaboration of strong binary classifier on the source domain

To be able to apply TransBoost, it is necessary to have a strong binary classifier for one task that is very weak for another task. To obtain sych a model a base model is chosen, ie. a pre-trained model without the last layers (Dense layer and softmax layer). The `Keras` package offers a large choice of pre-trained CNNs (see this page). The choice has been made to use the Xception model. On the one hand it produces good transfer values and on the other hand the time necesary to generate these transfer values is low. Transfer values are the values of the activation function for all the neurons of the last layer of the considered network for a given set of images.

To keep computing times low, the transfer values, for all the images of our datasets, are computed only once and used to train the last layer.

The architecture of the last layers are:

- Densely connected layer, of size 1024, "relu" activation function
- 50% dropout
- Densely connected layer, of size 512, "relu" activation function
- 50% dropout
- Densely connected layer, of size 1, "sigmoïd" activation function

This topology was chosen empirically.

Just like the noising of input data, the dropout layers are there to prevent overfitting that could occur due to the low number of training inputs. After the last layer is trained to our desired output, the base model is reassembled from the frozen layer and the last layers to create a complete network.

3.3 Application of TransBoost

Once our strong classifier is trained, the next step is just the application of the AdaBoost algorithm. The slight nuance is that instead of building a new classifier "from scratch" we have a partially trained classifier as our weak projector.

The do not want to relearn a separator function by retraining the last layer(s), as would be done in classical transfer learning with CNNs. Instead we seek to project the input points on the correct "side" of the separator function by retraining the first layers of the deep neural network.

To do so we unfreeze the first layers and freeze the weight of all the other layers, so they become untrainable. At this point two separate approaches are possible: either reinitialize the weights of the unfrozen layers or keep these weights. It is possible that by reinitializing the weights will stop the model from converging with so little training data. The choice will be made on experimental results. In both cases, the resulting model will be used as the weak projector / classifier in the boosting steps.

There are three important parameters to optimize while trying to keep a good baance between performance and computing cost:

- The power of the weak classifiers (accuracy threshold)
- The number of convolution blocs to train
- The number of weak models to train

The next steps are just the AdaBoost algorithm *(cf. Algorithm 1)*. It is important to note that the model's total error is measured on the validation set. However the model is trained on the weighted training set. The weights of each training point is based on the prediciton error of the previous boosting model on this same training set.

At each step, training of a projector stops when the accuracy threshold is reached. A limit on training epochs is set in case of non convergence.

Algorithm 1: TransBoost Algorithm

input : $\mathcal{X}_\mathcal{S} \to \mathcal{Y}_\mathcal{S}$: source hypothesis

$\mathcal{S}_\mathcal{T} = \{(\mathcal{X}_\mathcal{I}^\mathcal{T}, \mathcal{Y}_\mathcal{I}^\mathcal{T})\}_{1 \leq i \leq m}$: target training set

Initialisation: distribution of training set:$D_1(i) = 1/m \ for \ i = 1, \ldots, m$;

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

Find a projection $\pi_i : \mathcal{X}_\mathcal{T} \to \mathcal{X}_\mathcal{S}$ tq. $h_S(\pi_i(.))$ That is better than random on $D_n(\mathcal{S}_\mathcal{T})$;

Let ϵ_n be the error rate $h_S(\pi_i(.))$ on $D_n(\mathcal{S}_\mathcal{T})$: $\epsilon_n = P_{i \ D_n}[h_S(\pi_n(x_i)) \neq y_i]$(with $\epsilon_n < 0.5$);

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

Update: **for** $i = 1, \ldots, m$ **do**

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

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

Where Z_n is a normalization factor such as D_{n+1} is a distribution of $\mathcal{S}_\mathcal{T}$;

end

end

output: The final hypothesis: $H_\mathcal{T} : \mathcal{X}_\mathcal{T} \to \mathcal{Y}_\mathcal{T}$:

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

3.4 Difficultés rencontrées

We initially made the decisio of working in a pure `Tensor Flow` environnment, since the model we had chosen was available in this package. However when we had to modify the structure of the network (to have a binary output for example), or when certain layers had to be frozen using `Tensor Flow` became quite complicated. The actual model object was not found so we had to directly modify the graph which caused us some problems. For this reason we decided to use the `Keras` package that has a `Tensor Flow` backend, allowing us to use pre-trained models from the `Tensor Flow` library, but with a much clearer syntax and easier to use tools, allowing us to bring down the development times.

We also encountered computing power issues, since the virtual machines we had access to were limited in resources. This meant dozens of hours of computing time for each run, some times days and ending some times in memory errors. It is therefore necessary to have a powerful machine that could allow us to debug and develop the program without long delays.

As the project advanced and new, more powerful machines were made available to us, we had other problems. Due to the frequent modification of te network, reinitializing of weights, and saving to disk, we decided to save all the model in a list as each boosting step finished, and to delete the active model with the native Python `del` function. However this function doesn't guaranty the immediate deletion of the object, due Python's memory managment system. Enven after forcing

the execution of C's garbage collector (which is used for managing memory in Python), there were strange behaviors with partaially deleted models, and therfore some weights that did not change for each boosting step.

The decision was therefore made to save the models to the hard drive with `Keras` functions that allow us to save the architecture and the weights of the model separatly in `.json` files. Since the architecture of the model does not change we can save it only once at the beginning, after training it on the source dataset, and only save the weights at each boosting step. This architecture also indicates which layers are frozen and which are trainable durin g the boosting phase.

Once the model is saved to disk, we can call a `Tensor Flow` function that deletes all the objects of the `Tensor Flow` session and therefore ensuring that no objects from the previous session persist and disrupts the trainng of subsequent models. To be able to train a new model we can just load the architecture at the beginning of the boosting iteration and populate it with the weights of the base model.

Once all the projectors are trained we can do prediction by sequentially loading all the stored models (architecture + weights) and to store the predictions before deleting the model and loading the next one. After the last step, all the prediction are weighted with the α_i (that are calculated during the boosting phase) to get the total boosting prediction.

A priori this method of storing the model sto disk before deleting them and reload everything can seem clunky and unnecessarily heavy, however these additional steps are of little cost in front of the guaranty of no "cross contamination" between boosting steps and that we are training the right model. Furthermore the longer acces times to hard disk storage compared to system memory is negligible before the training times.

4 Critical comparison for the TransBoost method

4.1 Weak vs. strong projectors

We want to compare the Transboost approach of boosting weak projectors to the training of one very good projector. If the performance is higher, with shorter training times, in the TransBoost approahch, then we can prove a real advantage of this method.

4.2 Projectors vs. new classifiers

The transboost method requires quite a lot of memory and computing time. Indeed after having trained a big number of classifiers it is necessary to store them so as to be able to reuse them for class prediction of testing points. The space needed can be lowered by storing the architecture of the base model and only the weights of the modified layers at each boosting step, and reconstructing each boosting model when we need to predict a class.

We want to compare the TransBoost method with a classical boosting method.

It is possible to reach a low accuracy score (0.7) at each step quite easily with a simple CNN (only a few convolution blocks). This allows for shorter computing times. This is tested lower.

5 Experimental setup

All experimentations were done on an 8 core CU, with 30 Gb of RAM, and a GPU with 11 Gb of video memory, within the google cloud framework. This virtual machine had a Ubuntu server OS with a Python 3 distribution on which we install the GPU version of `Keras`. All programs were launched in a "screen" environnment so that disconnection from the virtual machine would not stop the execution. All outputs were logged to a text file.

The program is built as follows: the configuration to run the given execution is specified in a file in the `.json` format *(cf. 1)*. Since we wanted to run several experimentations (to test different accuracy thresholds for example) the program can be given a list of configuration files and will execute itself with each of them sequentially.

Listing 1: configuration example

```
{
  "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 Results

6.1 Building the binary classifier

In practice, for the source binary classification task "dog/truck", the model reaches a 98.9% accuracy after only 2 training epochs. This same model has a 50% accuracy on the target binary task: "deer/horse". This corresponds well to what we wanted from the base model.

6.2 The TransBoost method

After several trying several different parameter tunings (learning rate, optimizer, reinitializing modified layers or not, ...) the method does not converge in a reasonnable time. So that the algorithm can work, it must at least reach the accuracy threshold during the first iteration allowing for better performance after boosting.

In the present conditions (30 Gb of RAM, 8 cores and GPU with 11 Gb of memory) the algorithm doesn't even reach 55% percent accuracy after 4h and cannot go further than 65% accuracy on the training set. Therefore evene if we are aiming for 70% with resubstituttion to reach a accuracy between 55 and 60% on the validation set, we will not be able to reach this goal in reasonnable time.

Even if the algorithm did manage to converge, if 4 hours are needed per projector with this cimputing power, it is easy to see how this could be prohibitive. This can be explainde by the fact that the lower layers, containing low level descriptors (that take the longest to learn), are destabilized with a low amout of examples to boot. It is important to see this in light of the lower amount of time and computing power necessary to reach good performances in a classical transfer learning setting.

6.3 Classical boosting on small CNN

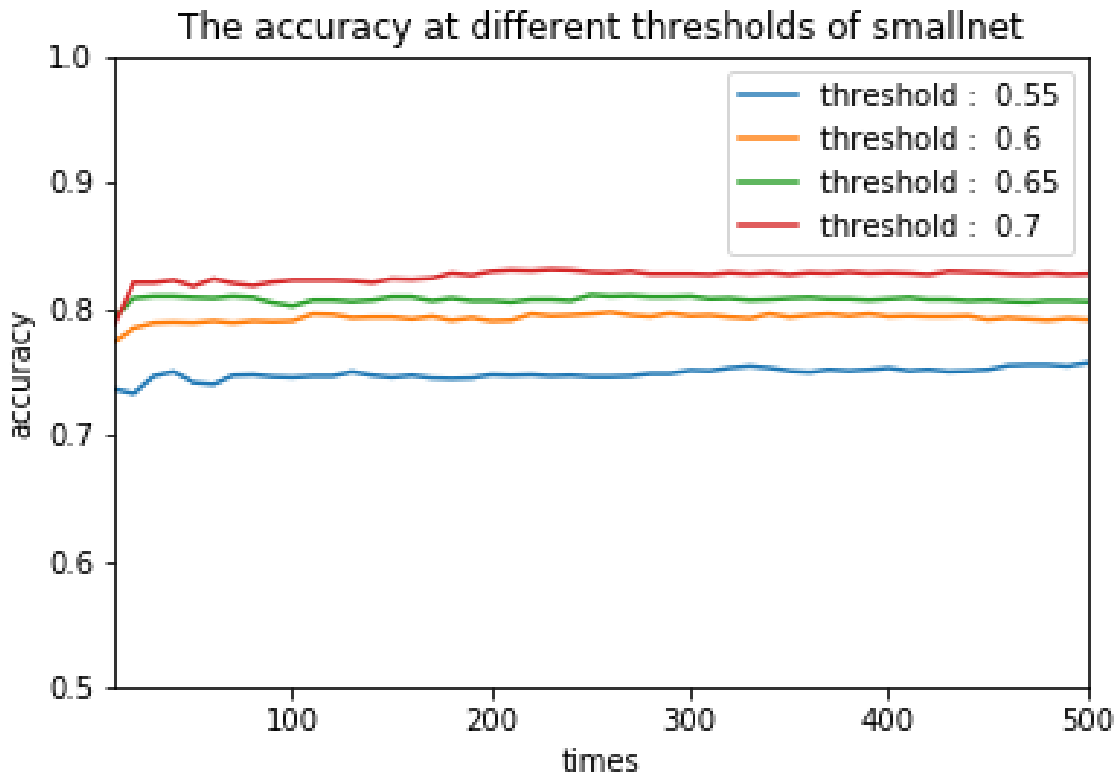

Figure 3: Evolution of the accuracy over time for a boosting algorithm with small CNNs as weak learners (no transfer)

A simple CNN was built. It learns on the target domain directly with a boosting algorithm. There is no transfer. It's architecture is as follows:

- two convolution layers woth 32 filters of 3*3 size each, "relu' activation function and max-pooling of size 2*2
- One convolution layer woth 64 filters of size 3*3, "relu" activation function and max-pooling of size 2*2
- one flatten layer
- One densely connected layer of size 64, "relu" activation function
- 50% dropout
- One densely connected layer of size 64, "relu" activation function
- 50% dropout
- One output layer with "sigmoid" activation function

This small CNN is used instead of the projectors in the boosting algorithm, and we measure the classification accuracy on the test set at sevral different boosting iterations. Results for this analysis are presented in Figure 3. These measures were done at several different accuracy thresholds for boosting learners.

The learning set is relatively difficult: "deer/horse". The results show that the boosting algorithm was correctly implemented since we see at least 10 points of gain in accuracy compared to th eweak learner threshold. It is interesting to note that however, the stronger the boosting learner is the higher the resulting accuracy is. We can see as well that a relatively small number of boosting steps is necessary before reaching an accuracy plateau where futher learning is unnecessary.

7 Follow-up experimentations

These are experimentations we would have wished to do if we had more time, and if the algorithm converged (at least in a reasonnable time).

7.1 Influence of source and target domains

Is it possible from a simple source domain *(ex: dog/truck)* to go to a complicated target domain *(ex: deer/horse)*. In our case we wish to study the complicated case but it would be interesting to study other cases. It would be interesting to see the influence of the different tasks on the execution. Are there "simple" and "complicated" tasks for the network?

7.2 Influence of hyper-parameters: optimization of accuracy/computing cost

- Influence of projector power
- Influence of the number of projectors
- Influence of the number of trainable layers *(idealy we want to modify the lowest possible number in order to go faster)*

7.3 Transboost vs. classical transfer learning

For the "deer/horse" target task, high accuracy (92.6%) can be reached in less than 10 epochs. This result can be improved by fine tuning the last convolution layers of the base model. So this classical transfer learning approach is very powerful.

8 Conclusion

Even though TransBoost seemed promising for time series classification, application of this method to image clasification, with it's inherent problems (data size, classification difficulty), is not very compelling.

Bibliography

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

[2] A. Cornuejols, S. Akkoyunlu, P.A. Murena, and R. Olivier. 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

The code is available at the following address:
https://github.com/zlanderous/transboost

B Overview of the program

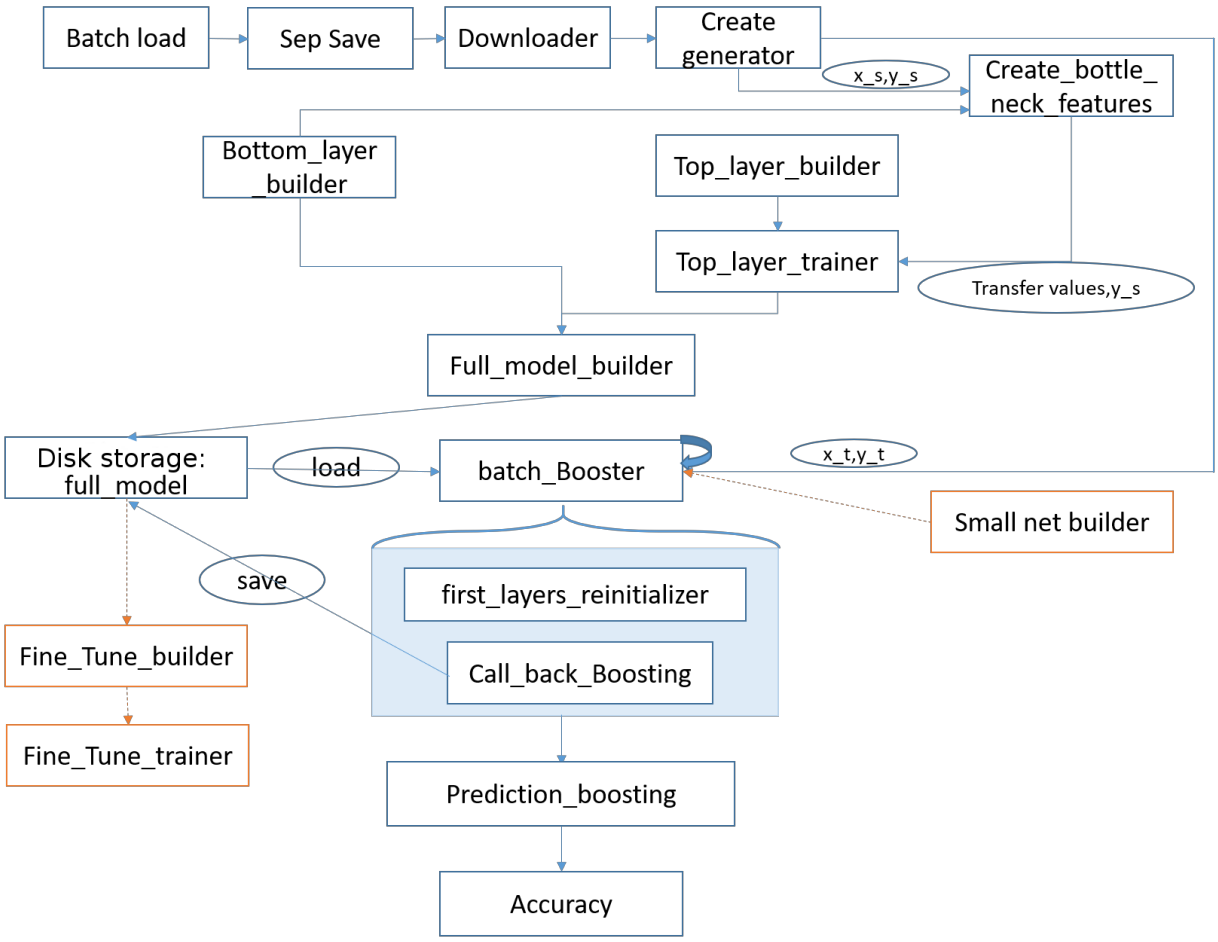

Figure 4: Schematic overview

functions:

Batch_loader: Takes the images from a CIFAR-10 batch and saves them in a dictionnary object with the image class as key.

Sep_saver: Goes through the dictionnary and saves images of each class in corresponding folders.

Downloader: Downloads CIFAR-1 batches and executes the 2 previous functions.

Create_generator: Creates generator objects from noised image batches.

Bottom_layer_builder: Loads and builds model without last layer (Xception model here).

Create_bottleneck_features: Computes transfer values for each image in the training, test and validation set with the model from the previous function.

Top_layer_builder: Creates last biary output layer for base model.

Top_layer_trainer: trains last layer with transfer values

Full_model_builder: Assembles the bottom layers of the model with the bewly trained output layer

first_layers_reinitializer: Freezes weights for n last layers and reinitializes weights of first layers for boosting

Callback_boosting: Called at the end of each training epoch. If accuracy is higher than a α threshold, training stops and the network weights are saved to disk.

batchBooster: Applies the AdaBoost algorithm to the network. Each model acting as weak projector is built and trained using the 2 previous functions.

Prediction_boosting: Predicts class of inputed image with trained models from boosting steps.

Accuracy: Evaluates accuracy of prediction.

small net builder: Builds small CNN with 3 convulution layers and 3 dense layers with droupout.

fine_tune_builder: Builds model to fine tune using base model and source domain data.

fine_tune_trainer: trains and fine tunes model.