CS542-FGVC CHALLENGE

TASK 1 – SUPERVISED LEARNING

OVERVIEW

The first task required to classify 990 classes using the following datasets:

- Train set: 184024 images
- Validation set: 25805 images
- Test set: 52275 images

The best result obtained is around 30% top 5 error over the validation set.

APPROACH

The final approach relies on a ResNet18 model network trained in our data. The model was build from scratch, following the model structure described in the paper [https://arxiv.org/pdf/1512.03385.pdf], in order to modify it easily and experiment with different possible structures.

I added a custom preprocessing layer to the ResNet for the data augmentation block to zoom, translate, and rotate the image randomly during training. The color data augmentation is done during the encoding of the image and relies on adding random saturation, brightness, and contrast to the training image only.

In the last layers, I added a custom head layer to reduce the features extracted from the ResNet, analyze them, and output the correct number of classes. The final head model is composed of a global average pooling layer, a dense layer, a dropout layer, and a final dense layer for the output.

Furthermore to prevent overtraining as much as possible and to be able to recover a precise weight situation during training, I used a checkpoint and learning rate scheduler technique for reducing the plateau. At every epoch, the model was saving the weights, and cutting in half the learning rate if the validation loss was converging (not decreasing for more than 3 epochs).

layer name	output size	18-layer
conv1	112×112	
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$

ResNet18

```
Layer (type)                 Output Shape              Param #
=================================================================
data_augmentation (Sequentia (None, 254, 254, 3)       0
_________________________________________________________________
functional_15 (Functional)   (None, 8, 8, 512)         11186889
_________________________________________________________________
global_average_pooling2d_5 ( (None, 512)               0
_________________________________________________________________
dense_10 (Dense)             (None, 1024)              525312
_________________________________________________________________
dropout_5 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_11 (Dense)             (None, 990)               1014750
=================================================================
Total params: 12,726,951
Trainable params: 12,719,009
Non-trainable params: 7,942
_________________________________________________________________
```

Complete model

EXPERIMENTS

I have first experimented the transfer learning technique for this task. I tested both a Resnet50 and a VGG16 pre-trained on imagenet with a custom head model. The head model was first trained with all the backbone model weights frozen. In a second step, I proceed to fine-tune the last layer of the backbone model. The model was reaching a plateau around 63% of accuracy in the validation set (depending on the model type), indicating that the imagenet weights trained on 1100 class mostly animals were not adequate to classify a dataset of mostly plants.

Then, I experimented with many different customizations of the baseline model adding some batch Normalization layers, dropout, relu activation function, and training for 30 epochs reaching around 60% accuracy (showing that the model was too simple or not enough to extract the feature details needed to classify 990 classes correctly).

I finally came back to the ResNet models family. I'm confident that one of the right approaches was to train a ResNet, with no pre-trained weights in order to adapt better to this classification task.

The hardest part for me in this assignment, was to find the right tuning for training the model of the learning rate, batch size. I have first trained the model on a subset of the dataset composed of 200 classes, to find the correct hyper-parameters, and only after I tested the configuration on the complete dataset. Even by reaching a validation accuracy of 75/80% on the class subset, the model was not able to replicate the result when trained on the complete dataset (I suppose because of the larger set of classes to be considered).

I have used Adam as optimizer; I also tried SGD but the loss and accuracy were oscillating too much and the training process was slower than with Adam.

In order to stop overtraining, I also tried L2 regularization in the hidden layer before the output layer, with no success as the model was training slower and the validation was converging sooner.

I found a correlation between the time of convergence of the validation set and the batch size. For a small network like Resnet18 with a smaller batch size, like 8 or 16, I was able to push the convergence of the validation set a little further, resulting in higher accuracy. For a larger network like Resnet50 it was the opposite, a batch size below 24 was causing weaker results, I suppose because a smaller batch size also increases the amount of noise that the model learns from the single images.

RESULTS

Model	Accuracy top5	Hyperparameters	Notes
ResNet50 [pre-trained imagenet] With fine tunning	Train acc: 80 Val acc: 63	Epochs: 15 Learning rate: 0.0001 Batch size: 32	Over-training Validation converge too early
Baseline Model + BatchNorm (after) + Dropout	Train acc: 70 Val acc: 60	Epochs: 15 Learning rate: 0.0001 Batch size: 32	Validation set converge
ResNet18 [no pre-train] + Flatten + Dense layer (1024 neurons) + Dropout + Dense layer (out classes)	Train acc: 65 Val acc: 55	Epochs: 15 Learning rate: 0.0001 Batch size: 32	Unstable (loss oscillations of both sets) Val and train loss and accuracy converge
ResNet18 [no pre-train] + Global avg pooling + Dense layer (out classes)	Train acc: 80 Val acc: 63	Epochs: 30 Learning rate: 0.0001 Batch size: 32	Over-training Validation converge early
ResNet34 [no pre-train] + Global avg pooling + Dense layer (1024 neurons) + Dropout + Dense layer (out classes)	Train acc: 80 Val acc: 64	Epochs: 30 Learning rate: 0.0001 Batch size: 32	Over-training Validation converge early
ResNet18 [no pre-train] + Global avg pooling + Dense layer (1024 neurons) + Dropout + Dense layer (out classes)	Train acc: 85 Val acc: 70	Epochs: 30 Learning rate: 0.0001 (reduced after 16 epoch to 0.00005) Batch size: 16	Still over-training, but improvements

TASK 2 – SEMI SUPERVISED LEARNING

OVERVIEW

The second task required to classify 20 classes using the following datasets:

- Train set unlabeled: 4208 images
- Train set labeled: 100 images
- Validation set: 606 images
- Test set: 1224 images

The best result obtained was around 73% accuracy on the validation set.

APPROACH

FEATURES EXTRACTOR MODEL

The model I built is composed of a ResNet34 pre-trained on imagenet dataset (with frozen weights during training), and a custom "head" dense layer on top of the original ResNet model. The idea is to use the ResNet model to extract relevant high-level features (by leveraging on the transfer-learning concept) and then train the small DNN on top to classify the data using those features.

ResNet34

Layer name	ResNet34
conv1	7×7, 64, stride 2
pool1	3×3, max pool stride 2
conv2_x	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$
conv3_x	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$
conv4_x	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$
conv5_x	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$
fc1	9×1, 512, stride 1
pool_time	$1 \times N$, avg pool stride 1

Head model alone

```
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 512)               262656
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 20)                10260
=================================================================
Total params: 272,916
Trainable params: 272,916
Non-trainable params: 0
```

Due to the lack of examples available in the labeled training set, the model was not able to go over a maximum threshold of accuracy on the validation set. I still tried to push as much as possible the model accuracy by using: data augmentation, drop out layer, tuning the number of neurons in the dense layer, manipulate learning rate and a number of epochs, obtaining 67% accuracy on the validation set. The objective here was to tune the hyper-parameters and model structure to obtain the best classification model to then start experimenting with semi-supervised learning techniques.

SEMI-SUPERVISED TECHNIQUE

I used self training as semi-supervised technique. I built a python version of the following algorithm from scratch:

Input: labeled data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$, unlabeled data $\{\mathbf{x}_j\}_{j=l+1}^{l+u}$.
1. Initially, let $L = \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$ and $U = \{\mathbf{x}_j\}_{j=l+1}^{l+u}$.
2. Repeat:
3. Train f from L using supervised learning.
4. Apply f to the unlabeled instances in U.
5. Remove a subset S from U; add $\{(\mathbf{x}, f(\mathbf{x}))|\mathbf{x} \in S\}$ to L.

The following components have been added to the algorithm to obtain better results (more details in the next section):

- Training using categorical cross-entropy (softmax), and prediction using a sigmoid (this because I wanted a raw measure of the likelihood of every class predicted in the prediction stage, not rescaled by the softmax). Only the prediction with score above 95% of U were considered for "promotion" to L.

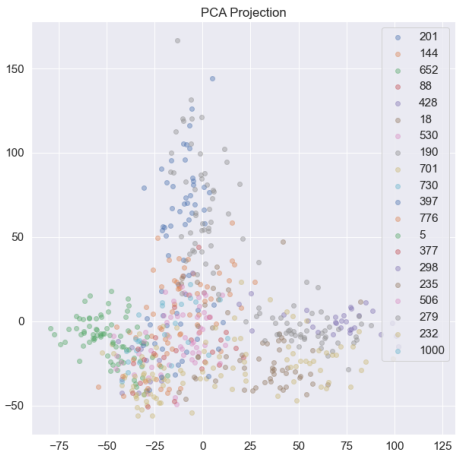
- To avoid the model to over-train on a specific class I added a constraint, at step 5 of the algorithm, to move and remove at each iteration only 1 top prediction sample for each class. The process was slowed down but at least I was guaranteeing a has much as possible a balanced training set at each iteration

EXPERIMENTS

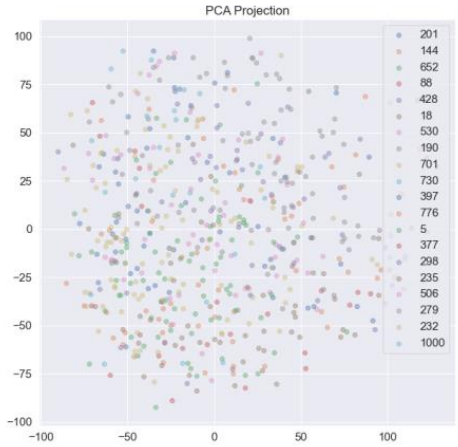
FINDING THE RIGHT IMAGENET PRE-TRAINED MODEL

I have evaluated multiple models pre-trained on imagenet, such as VGG16, MobileNet, ResNet (multiple sizes), Xception etc. By looking at the feature space that the model creates I was able too to observe and evaluate the classification power that those features will have in our classification task. Using PCA on both validation and training set I was able to observe, in the lower dimension, how the different models were extracting features in relation to each class label. I present here 3 examples showing different features representation by 3 different models. We can see that the best representation is done by the ResNet34 model on the left where the data is more separable into clusters (3 or 4 clusters observable by eye), the other 2 examples show no clear separation, due to too much or too low density of the data distribution. This hypothesis was confirmed when I used those features to train the head DNN model and compared the different accuracies (ResNet34 was performing better).

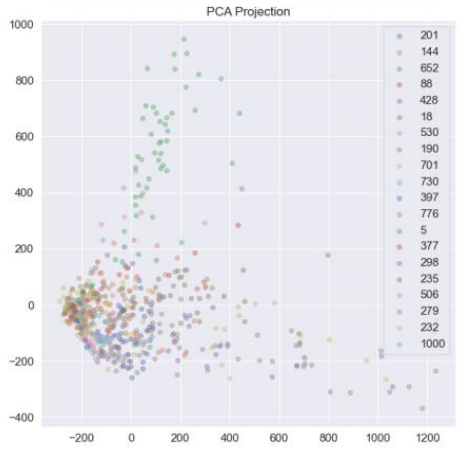
➔ Libraries used: Keras, Scikit-learn and Classification_models (*https://github.com/qubvel/classification_models*)



ResNet34



MobileNet



VGG16

MODIFICATIONS OF THE ORIGINAL SELF-TRAINING ALGORITHM

When I experimented self-training I found that the algorithm was quickly over-fitting after a few iterations. One of the reasons was the fact that the model was moving, from U to L set, always the same class. This caused a small raise in the val accuracy at the beginning of the process, but it was then getting worse due to the imbalanced train set L (observed using the confusion matrix plot).

To avoid this imbalance in the train set L during self-training, I have modified step 5 of the algorithm in the following way: only one sample of the U set of each class can be promoted to class L at each iteration (this to ensure a balanced L set for training at each iteration). Also, the algorithm considers only predictions with a high probability 95% to minimize the possibilities of misclassification.

Even with those considerations, the improvements were small and inconsistent from run to run. The best result was around 73% in the validation set, in the firsts iterations (best training and loss results).