DL - Transfer Learning Lab

Jason Klimack and Gonzalo Recio

Introduction

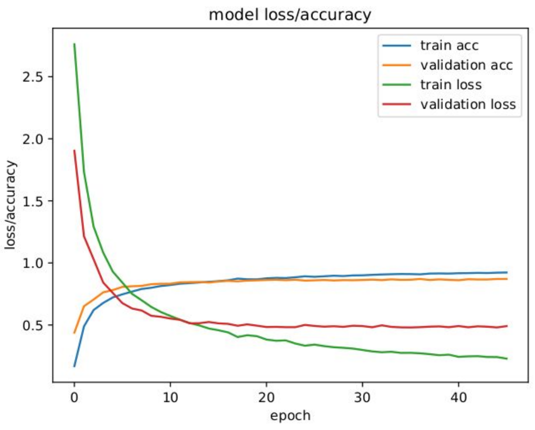
We are going to explore transfer learning with several deep NN that have been State of the Art in image classification for the recent years (VGG16, InceptionV3, ResNet50).

We aim to explore several configurations of fine tuning and feature extraction. In order to do it, we are going to start using only the VGG16 model and then compare the best configuration with other networks: ResNet50 and InceptionV3.

Also, to do so, we are going to complicate the problem as we managed to reach **0.96** accuracy on the test set in the first lab. We are going to reduce the data to e.g. 500 instances per class and remove data augmentation.

Recap from Lab 1

We achieved a final model in lab 1 that reached 0.96 with all the dataset samples, but tested again with the reduced dataset and without data augmentation, we now obtain an accuracy of **0.86**. Also, training time was of 27 epochs * 3 s/epoch = **81 s**.



These are still pretty good and competitive results regarding the small amount of data with quite fast training. This will establish a baseline from which we can compare the impact of using a pretrained model for fine tuning of feature extraction to achieve similar or even better results than training from scratch.

Fine Tuning

Goal: use a pre-trained model related to image classification with its pre-trained weights and adjust them to our problem. The idea is that as these models are already pre-trained to solve image task classification, their weights should already be good enough for extracting image features and only need to adjust them a bit to perform our task (faster convergence and less training time). We are using *ADAM* optimizer, LR=0.001 and early stopping with patience of 10 epochs. To complicate the problem, we are going to use only 500 instances per class.

For fine tuning, we loaded the VGG16 model without the last dense layers (with include_top= False) and added our custom ones. This is because the last layers are trained to perform the specific task for which the authors trained the whole model (ImageNet classification problem).

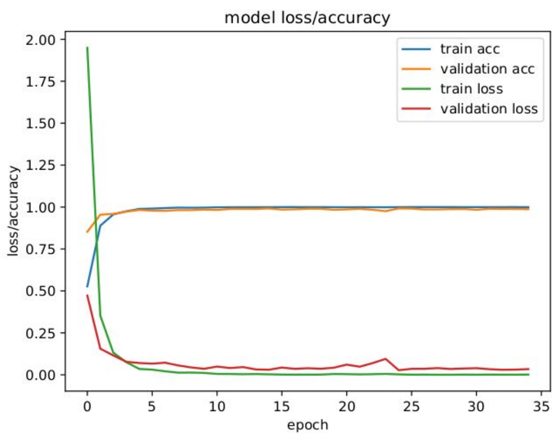
```
x = model_trained.output
x = Flatten()(x)
x = Dense(512, activation="relu")(x)
x = Dropout(0.2)(x)
x = Dense(256, activation="relu")(x)
predictions = Dense(num_classes, activation="softmax")(x)
model = Model(inputs=model_trained.input, output=predictions)
```

We can also freeze some layers in case we don't want all the pre-trained parameters to be re-updated. In the following section we are going to try fine tuning freezing some layers vs. not freezing any of them.

VGG16 Freezing first half of the layers

We freeze the first half of the total layers of the VGG16 network, as follows:

```
half = int(len(model_trained.layers)/2)
for layer in model_trained.layers[:half]:
   layer.trainable = False
```



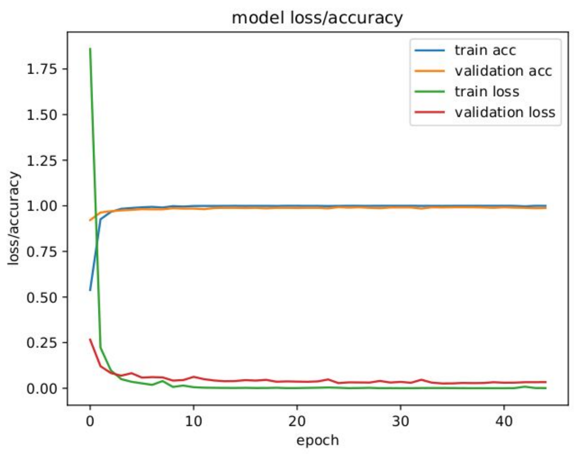
Results:
Accuracy (test) = **0.9835**
Time=35 epochs * 24s/epoch = 840s = **14 min**.

In the plot, we can see a very fast convergence to a great accuracy of almost 1 within the very first epochs.

VGG16 Not freezing any layer

We let the model adjust all the parameters by setting all layers of the model as "trainable".

```
for layer in model_trained.layers:
   layer.trainable = True
```



Accuracy (test) = **0.9917**
Time = 45 epoch * 41 s/epoch = **30.75** min

Once again, in the plot we observe a very fast convergence to an accuracy of almost 1.

VGG16 using all dataset samples

We also wanted to test the pre-trained model with all the dataset samples to see if it improves our model from Lab 1, without freezing any layer, aiming to see which is the best accuracy it can reach.

Results:
Accuracy (test) = **0.9993**
Train time = 43 epochs * 151 s/epoch = **108.21 min**

Result analysis of Fine Tuning

In table 1, we show results from working with 500 instances per class of fine tuning and feature extraction.

Training with 500 instances		Accuracy	Training Time
Baseline	CNN from lab 1	0.86	81 s
Fine Tuning	VGG16 half frozen	0.9835	14 min
	VGG16 none frozen	0.9914	30.75 min
Feature Extraction	VGG16 + NN	0.9826	6.1 min
	VGG16 + SVM	**0.9926**	**39 s**

Table 1. Comparison of the results obtained with Fine Tuning and Feature extraction

We can observe that results with transfer learning using the VGG16 model outperform our CNN model from Lab 1. Especially, feature extraction provided great results considering the few training time required.

In table 2, we can see the results obtained when training the models with the whole dataset.

Training with all the dataset	Accuracy	Training Time
CNN from lab 1 (all)	0.96	46.6 min
VGG16 none frozen	**0.9993**	108.21 min

Table 2. Comparison of the results obtained with our CNN and the VGG16 when using all the dataset

With the VGG16 we achieved a near 1.0 accuracy (0.9993), which motivates the fact of reducing the dataset to complicate the problem.

Feature Extraction

Goal: use the output of a pretrained model as input features for a classifier.
To use the pretrained models as feature extractors, we are going to freeze their weights and add a classifier that is going to be added afterwards. We are going to try 2 classifiers: a NN and a SVM.

VGG16 + Custom NN

We set all trained layers from VGG16 to “not trainable”, added 2 dense layers, 1 dropout layer and set the output layer with softmax activation: Dense(512), Dropout(0.2), Dense(256), Dense(softmax).

To do this, we freeze all the layers of the VGG16 model:

```
model_trained = VGG16(include_top=False,input_shape=input_shape)
for layer in model_trained.layers:
   layer.trainable = False
```

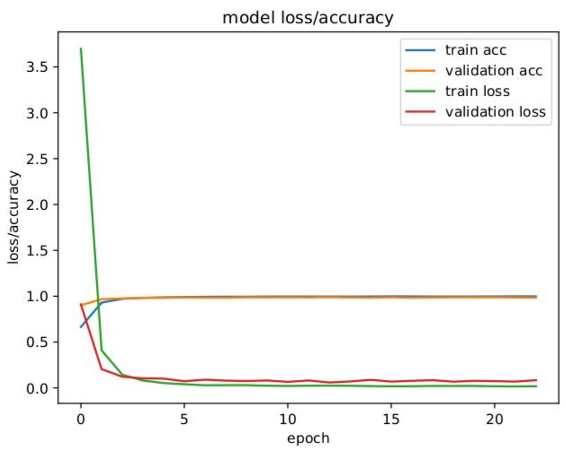
and add our custom Dense layers at the end:

```
x = model_trained.output
x = Flatten()(x)
x = Dense(512, activation="relu")(x)
x = Dropout(0.2)(x)
x = Dense(256, activation="relu")(x)
predictions = Dense(num_classes, activation="softmax")(x)
model = Model(inputs=model_trained.input, output=predictions)
```

This way, we only have to train the last 3 layers which are going to work as a NN classifier having the features obtained from VGG16 as input.

Results
Accuracy (test) **= 0.9826**
Time = 23 epochs * 16 s/epoch = 368s = **6.1 min**
(training time of the last added layers)

We can notice the fast convergence achieved using the features from the pre-trained model with only few epochs.



VGG16 + SVM

In this section we are adding a linear SVM to the features obtained by the VGG16 model. To achieve this, we are going to load the VGG16 model without the last layers, and add a flatten layer in order to convert the features of the last layer (7x7x512) into the unidimensional shape of (25088x1).

```
model_trained = VGG16(include_top=False, input_shape=input_shape)
x = model_trained.output
features = Flatten()(x)
model = Model(inputs=model_trained.input, output=features)
```

Now, our input features are going to be the output (prediction) of this model (VGG16 + Flatten).

```
train_feats = model.predict(x_train)
test_feats = model.predict(x_test)
```

Then these features are directly passed to the SVM to train.

```
clf = svm.LinearSVC()
clf.fit(X = train_feats, y = np.argmax(y_train, axis=1))
```

Results
Accuracy (test) = **0.9926**
Time = **39 s** (time of training the SVM)

Comparison between pre-trained models

In the following plots, we can see a comparison of these models performed by Canziani et al.[1], where it can be observed that Inception-v3 appears to be better than ResNet50 and, the same time, ResNet seems to outperform VGG16 (Inception-v3 > ResNet50 > VGG16).

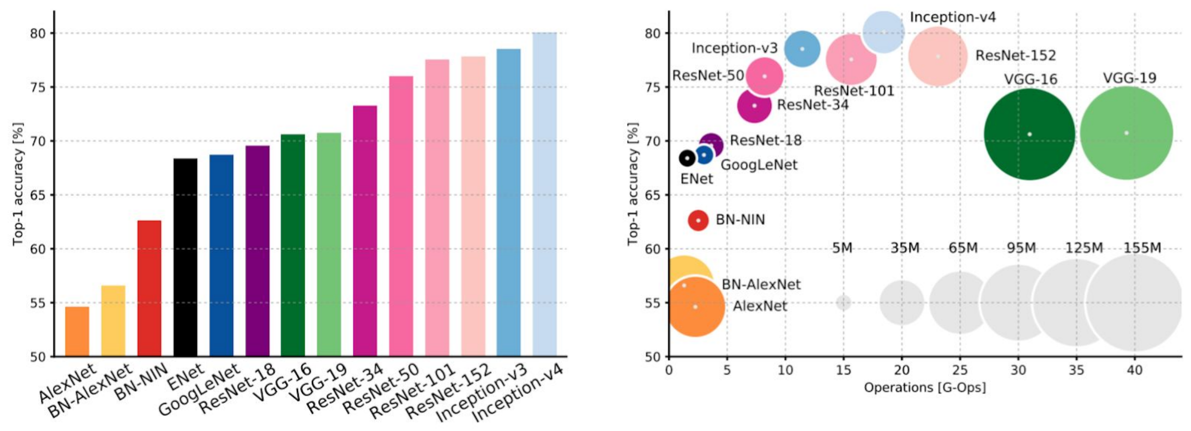


Figure 1: Performance amb complexity comparison of the SoTA image classifiers.[1]

[1] Canziani, A., Paszke, A., & Culurciello, E. (2016). An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*. (paper)

As we have seen in the previous sections, with the VGG16 model we can reach 0.99 test accuracy. For this reason, we are going to make it even more challenging and only use 100 instances per class, putting ourselves in a scenario where we have very few limited examples to build a robust CNN.

Results and model info:

- CNN from lab 1
  - Number of layers: 9
  - Accuracy (test): **0.6590**
  - Train time: 27s

- VGG16 + SVM
  - Number of layers in the trained model: 19
  - Feature dimensionality: 1x25088
  - Accuracy (test) = **0.9863**
  - Train Time (SVM): 7s

- ResNet50 + SVM
  - Number of layers in the trained model: 175
  - Feature dimensionality: 1x100352
  - Accuracy (test): **0.9863**
  - Train Time (SVM): 40s

- InceptionV3 + SVM
  - Number of layers in the trained model: 311
  - Feature dimensionality: 1x51200
  - Accuracy (test) = **0.9272**
  - Train Time (SVM): 57s

Result Analysis of Feature Extraction

In table 2, we can observe the results of feature extraction experiments from working with 100 instances per class with the different pre-trained networks.

	Accuracy	Training Time	Number of layers
CNN lab 1	0.65	27s	9
VGG16 + SVM	**0.9864**	**7s**	19
ResNet50 + SVM	**0.9864**	40s	175
InceptionV3 + SVM	0.9272	57s	311

Table 2: Comparison of Feature Extraction results between our CNN, VGG16, ResNet50 and InceptionV3

As we can see, all pretrained networks work excellent. In concrete, we expected the InceptionV3 model to outperform the others as shown in the by Canziani et al. However, we found that VGG16 and ResNet50 both outperformed the InceptionV3 for our dataset. This could be explained because InceptionV3 has a lot of layers (331) in comparison to VGG16 (19) and ResNet50 (175) - considering all layers, trainable or not - and it may happen that the last convolutional layers of the InceptionV3 network are quite specialized in the specific task of ImageNet classification and, hence, its features that are not 100% suitable for our problem (maybe we should have removed some of the last conv. layers).

Conclusions

Transfer learning is not a plug-and-play methodology, you have to carefully know the preprocessing performed by the authors of the network and resize the input images (in case of not fully convolutional networks). For instance, in our first model we scaled the image pixel values from 0 to 1 by dividing by 255. In the VGG16 model, the authors performed an scaling centering values at 0 (subtracting the mean values of each color channel). Therefore, in our case, the image scaling was different from the VGG16 authors, and without following their same preprocessing steps, the network did not work at all by giving 0.04 accuracy, which is completely understandable.

Regarding fine tuning, we have experimented with the VGG16 model by freezing the first half of the layers vs. not freezing any layer. As we can see in table 1, accuracy is improved when setting all parameters as 'trainable', but the training time increases proportionally (it doubles in time).

Regarding feature extraction, we have seen that linear SVM seems to be better than NN when classifying the features obtained from pretrained models, as what it is suggested in the work of Tang, Y[2]. In their work they state that replacing softmax with linear SVMs gives significant gains on popular deep learning datasets. In our case, although both results are incredibly great, SVM also seems to classify slightly better than the dense NN, which could be explained due to the better classification boundaries obtained by SVMs (boundaries that creates a clear gap that is as wide as possible).

To conclude, we can state that transfer learning is an excellent approach when dealing with few data samples or when the resources for training huge networks from scratch are not available. Also, considering our classification problem, we would recommend first to try transfer learning for feature extraction as we obtained better results and the training is faster (these pretrained models are quite large and, hence, they demand resources such as GPU and a lot of memory to train them). On the other hand, fine tuning is also a good option taking into account the fast network convergence, which only requires few epochs to reach a great classification performance, although it is a more time and resource consuming option.

[2] Tang, Y. (2013). Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*. (paper)