Report for the Deep Learning Course Assignment 3

Daniel Sanchez Santolaya
daniel.sanchezsantolaya@student.uva.nl

Abstract

In this assignment Convolutional Neural Networks (CNN) are used for image classification on CIFAR10 dataset. First, a CNN with 2 convolutional layers is created and trained from scratch. Then, a siamese CNN is trained in order to separate the representations of the image features of diffent classes in the last layer. Finally, a pre-trained VGG Net is used as a convolutional part of a network, which increses the performance using transfer learning.

1 Task 1

In this section, we implement a convolutional neural network with the architecture described in the assignment. First, the experiment was made without regularization, using Adam with learning rate of 0.0001, batch size of 128 and a maximum of steps of 15000. The weights of the filters and the fully connected layers were initialized with a gaussian distribution with standard deviation of 0.01. Figures 1 and 2 show the loss and accuracy in the train and test sets with respect the number of steps. We see some symptoms of overfitting, as the test loss gets worse while the train keeps improving. The test accuracy obtains a maximum of 0.755.

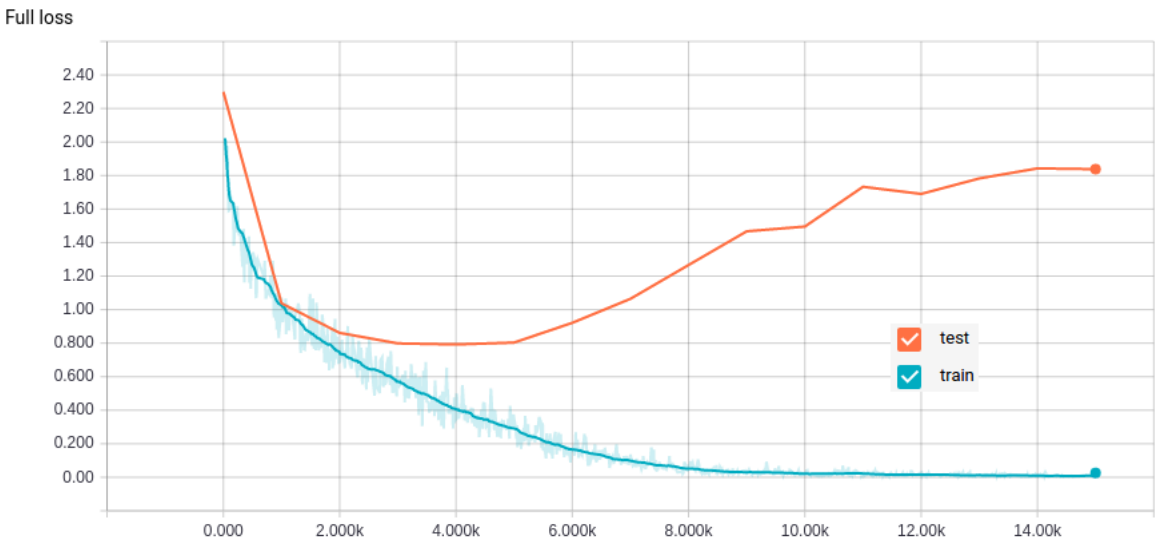


Figure 1: Training and test loss with respect the number of steps, without regularization

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

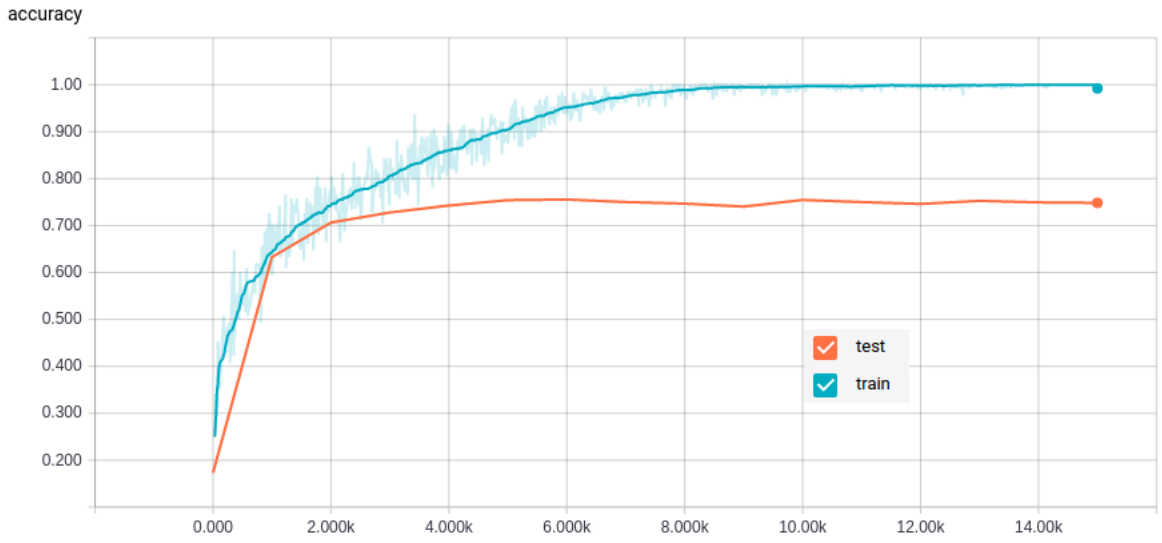


Figure 2: Training and test accuracy with respect the number of steps, without regularization.

In figures 3 and 4 we can observe the loss and accuracy of the same network and settings, but applying L2 regularization with 0.05 as weight decay on the fully connected layers. We can check that the loss of the train and test set are much more similar than the case without regularization, therefore the overfitting has been removed. In this case the achieved accuracy was 0.7763.

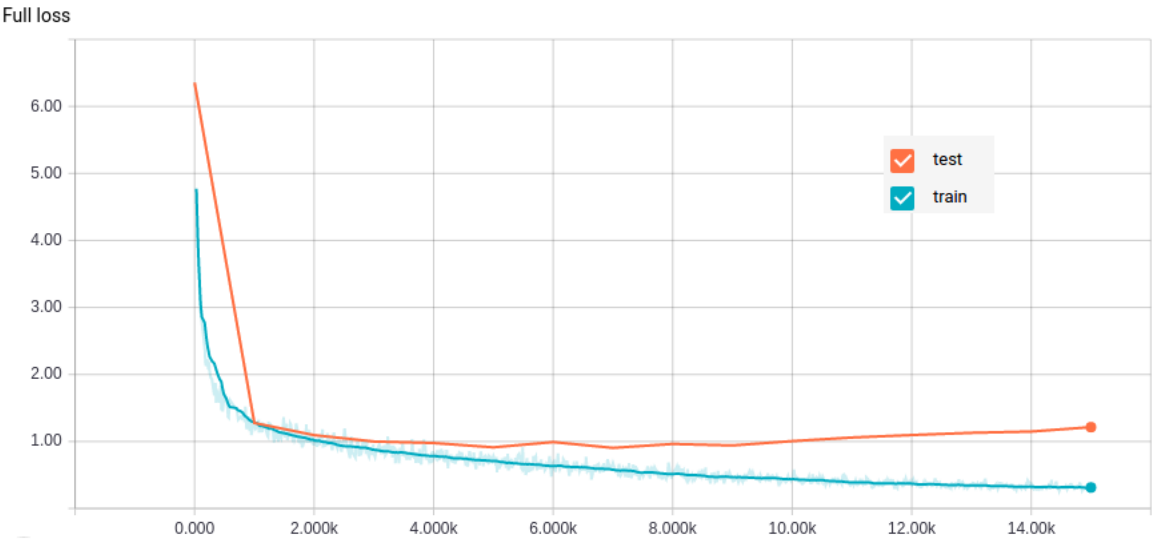


Figure 3: Training and test loss with respect the number of steps, with L2 regularization with 0.05 weight decay.

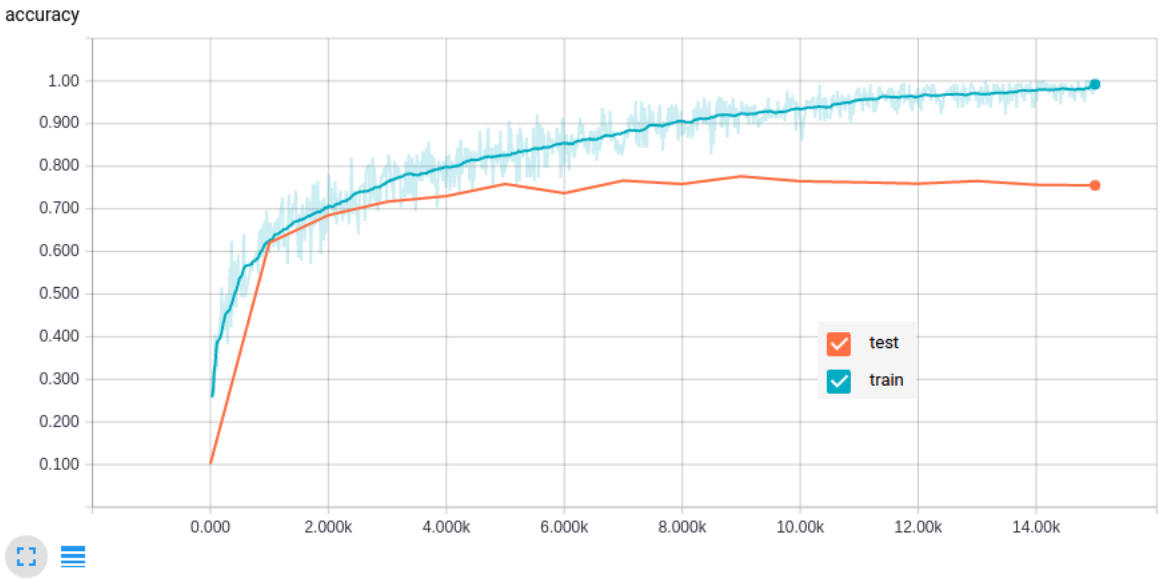


Figure 4: Training and test accuracy with respect the number of steps, with L2 regularization with 0.05 weight decay.

Table 1 shows the t-SNE representation of the output of the flatten, fc1 layer and fc2 layer, without regularization (left) and with L2 regularization with 0.05 web decay (right). Although it seems that there are some clusters, the flatten layers seem the one with more mixed classes in the t-SNE representation. Layers fc1 and fc2 have seems to have clearer clusters and the classes look more separable.

Table 2 shows the accuracy of training 10 linear one-vs-rest Support Vector Classification(SVC). We can compare the performance of using the features extracted at the layers flatten, fc1 and fc2. As we expected, fc1 and fc2 obtain a better accuracy than flatten, as they had the classes more separable in the t-SNE representation.

Note: There is only the results of the regularized networks. I was expecting to add the results for the no regularized networks, but the long queue in Cartesius the last day did not allow me execute it on time.

	Accuracy (without regularization)	Accuracy (with regularization)
flatten	X	0.634
fc1	X	0.7528
fc2	X	0.7689

Table 2: OneVsAll LinearSVC accuracy using the features extracted in the different layers.

2 Task 2

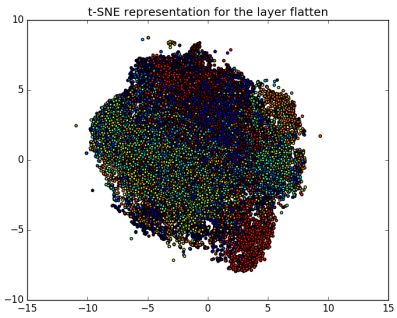
In this task, we train a Siamese CNN with the goal to map images of the same class nearby in the feature space of the last layer of the network. The Siamese CNN is formed by two CNN that share the parameters. Consider x_1 the outputs of the network in the first CNN, and x_2 the outputs of the network in the second CNN. Then, the loss function optimized is:

$L = Y \cdot d^2 + (1 - Y)max(margin - d^2, 0)$

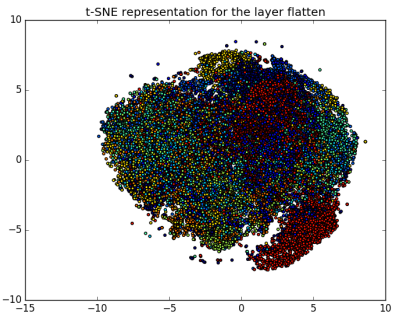
where

$d = \|\mathbf{x_1} - \mathbf{x_2}\|$

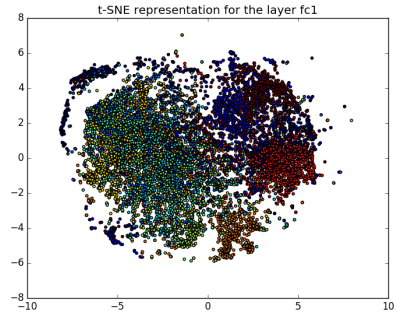
Y is the set of labels of every pair image x_1, x_2, such is 1 if the class of the images are the same and 0 otherwise.



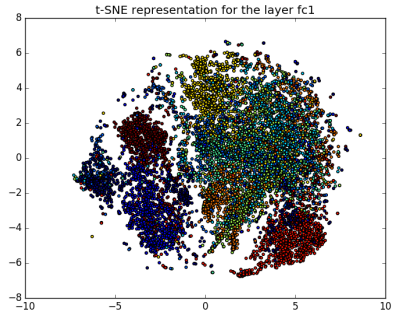
(a) layer flatten without regularization



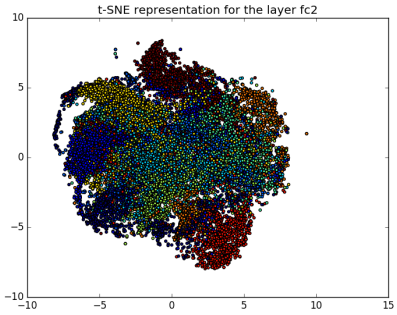
(b) layer flatten with regularization



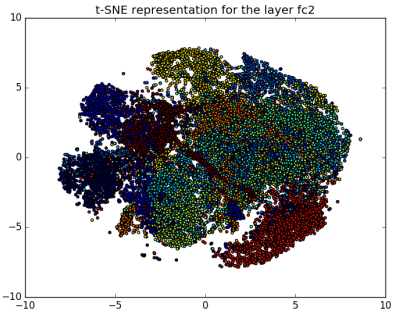
(c) layer fc1 without regularization



(d) layer fc1 with regularization



(e) layer fc2 without regularization



(f) layer fc2 with regularization

Table 1: t-SNE representations in the different layers without regularization(left) and with regularization(right)

This loss is called contrastive loss and measures how well the network is able to place similar images close in the output, and keep dissimilar images separated. The first component $Y \cdot d^2$ affects to the images of the same class, so it is a penalty of similar images that are far away. The second component $(1 - Y)max(margin - d^2, 0)$ affects to the images of different class, so it is a penalty of dissimilar images that are nearby. That makes this loss suitable when want to find images that are similar, which can be useful in web image search for example, when we want to recover similar images to one that we query.

The network was trained using Adam with learning rate 0.0001, batch size of 128 and 12000 steps. The weights were initialized with a gaussian distribution with standard deviation of 0.01, and L2 regularization was used with 0.01 web decay. The margin of the loss was set to 0.2. The sampling of the image for the next batch worked in the next way: for the first convolutional network, the next 128 images in the training set were assigned (just as in task1). For the second convolutional network, 0.2

of the images were selected of the same class of its belonging pair (always checking that the images are different). The other 0.8, were selected as random images from random classes.

Figure 5 shows the train and test loss for the siamese network. We see that it converges fast, achieving a test loss of 0.05522.

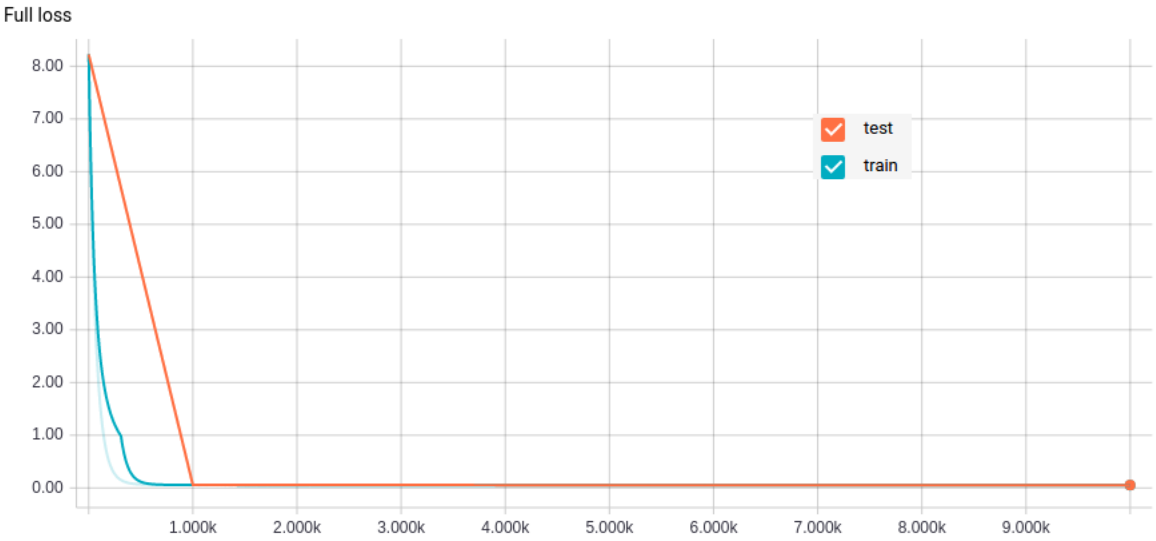


Figure 5: Training and test loss in for the siamese network, with respect the number of steps

Figure 6 shows the T-SNE representation for the features in the siamese network at L2-norm layer. Unfortunately, this representation does not belong to the loss shown in figure 5, but with a loss around 1.5 in the test set (with another set of parameters). The long queue in Cartesius the last day did not allowed me to execute the feature extraction with the last set of parameters. As we see, the classes does not look highly separable. Training a one-vs-rest classifier using the features at L2-norm layer leads to an accuracy of 0.2598, showing in effect that the network has not achieved separate the classes effectively for this set of parameters. However, with the set of parameters that lead to the lost shown in figure 5 we would expect better results.

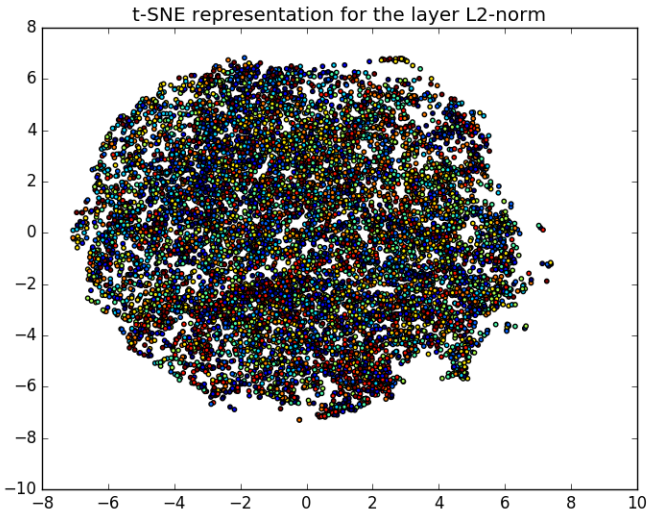


Figure 6: T-SNE representation for the features in the siamese network at L2-norm layer

3 Task 3

In this task3, the convolutional part of the network is formed by the convolutional part of the VGG Net. We use transfer learning so the weights of the convoluational part are initialized with the weights of a pretrained VGG-16. In all the experiments, network is optimized using adam, with learning size of 0.0001, batch size of 128, 10000 as number of steps. The weights of the fully connected layers were initialized from a gaussian distribution with standard deviation of 0.01 and they were regularized using L2 regularization with 0.05 as weight decay.

Note: Using the last layer of the VGG network (pool5) created some errors in tensorflow when executing in Cartesius. For this reason, instead I used pool4 as the last convolutional layer of the VGG Net.

3.1 Feature extraction

In this first experiment the VGG-16 is used as a feature extractor. The images are feed in the VGG Net, so then, we obtain the features at the layer pool4, which are feed to the fully connected layers. Only these fully-connected layers are trained, so the weights of the VGG-16 part always remain the same. In figure 7 we can see the graph of the network, where cond is a condition that stops the propagation of the gradients before the step k (in this experiment k > number of steps, so the weights of the vgg are never updated).

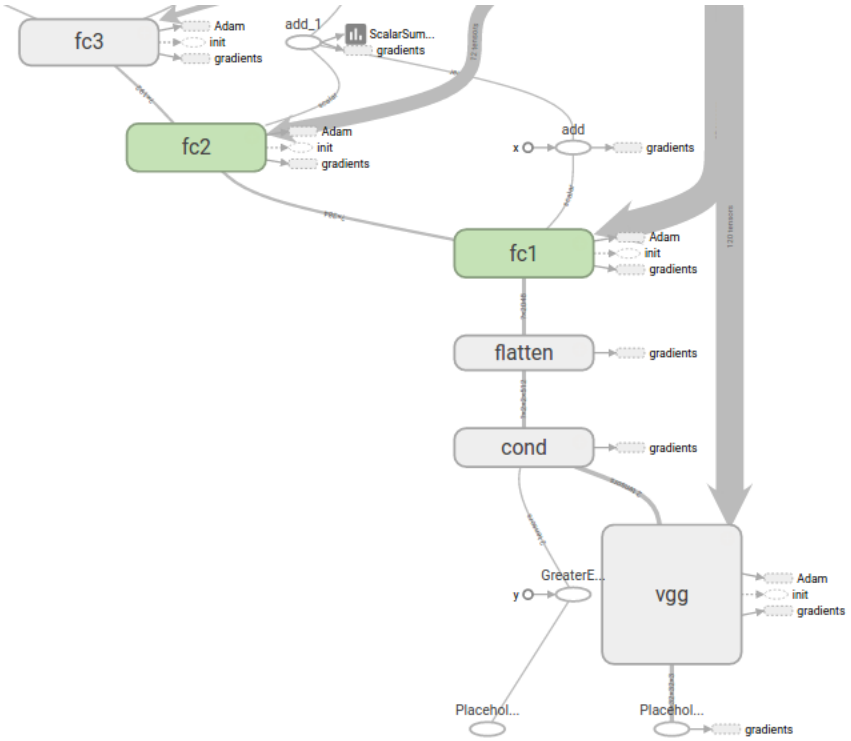


Figure 7: Graph of the network in tensorboard.

In figure 8 we can see the train and loss accuracy of the VGG Net used as feature extractor, and the network used in task1 with regularization. We can observe that the pre-trained weights allow to the VGG Net converge faster than the network trained from scratch. Probably the first layers of the VGG Net can be highly reused for the new classification task, as they represent generic features, for example edge detectors, which are useful in many task. On the contrary, the network trained in task1 has to learn this features, so that is probably the reason why the VGG performs better at the beginning. At the end, both accuracy converge to similar values, having VGG a slightly better performance (0.792 vs 0.7763), which could be also due to the fact that the VGG Net has two convolutional layers

more (although we cannot be sure, as the last layer could represent features more specific for the trained task).

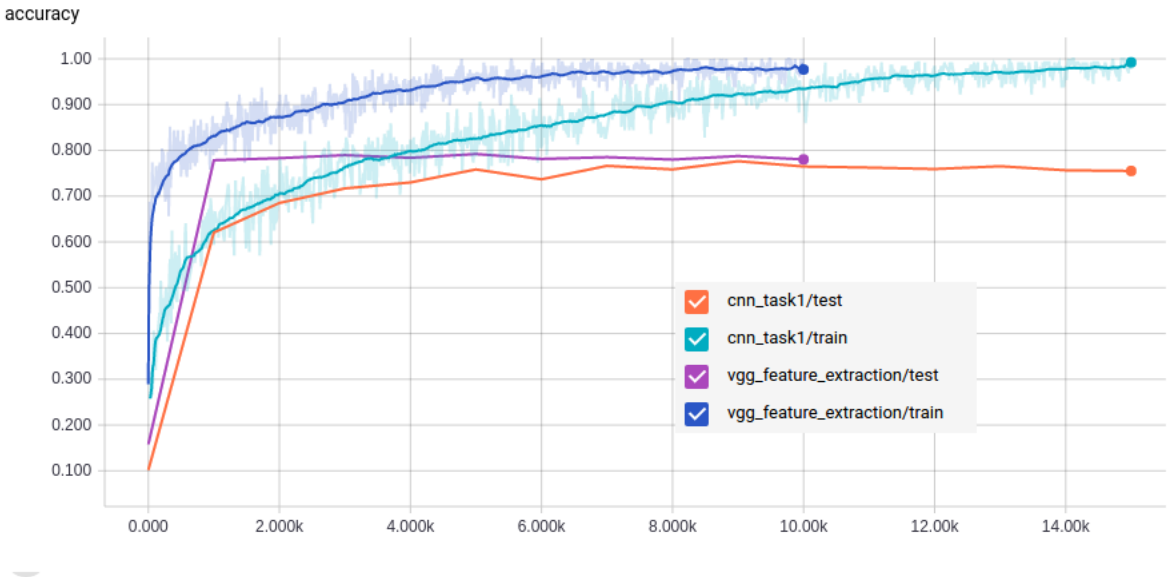


Figure 8: Train and test accuracy, for the VGG Net used as feature extractor, and the network used in task1 with regularization.

3.2 Retraining

As in the previous case, in this experiment we also use the convolutional pat of a pre-trained VGG net. The difference is that in this case we also update the parameters from the VGG layers (retrained network).

Figure 9 shows the train and test accuracy, for the retrained network compared with the VGG Net used only as feature extractor. As we see, the performance of the networks starts similarly, but after some iterations the retrained network has a better accuracy. The reason is that the retrained network can update the parameters in the convolutional network, so it can learn better and adapt these layers the task for which is trained. The best accuracy obtained by this network is 0.8639, in front of the 0.792 achieved in the last experiment, and the 0.7763 in task 1.

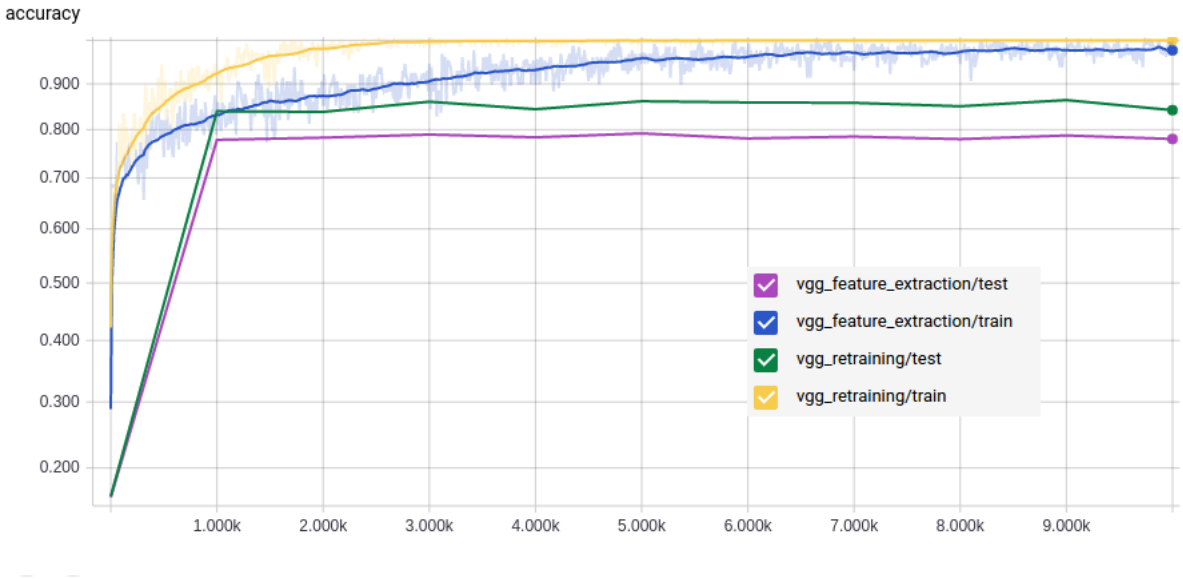


Figure 9: Train and test accuracy, for the retrained network, and the VGG Net used as feature extractor.

As we see in figure 10 the network updates the parameters of the convolutional layers. In the image we see the histogram of the weights in the first convolutional layer during the training. The distribution seems to do small changes, but as it is the first layer, it should not need big differences as it represent basic features.

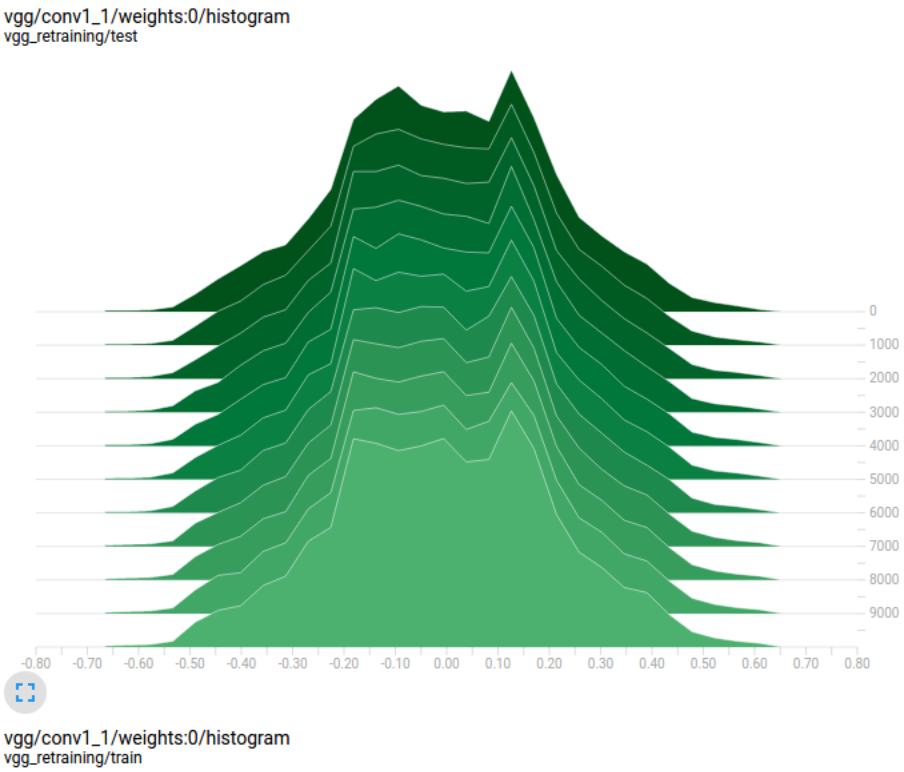


Figure 10: Weight histogram progression during training, for the first convolutional layer.

3.3 Refining

In this experiment the convolutional layers of the VGG-16 are not updated in the first k iterations. After the first k iterations, all the layers are updated. The experiments is performed for 100, 1000 and 2500 iterations. We will refer to the networks as refined_after_k, where k is the number of iterations used.

In figure 11 we observe the accuracy for the different networks refined_after_k and the retrained network used in the previous experiment. We see that all the networks converge to approximately the same accuracy in the test set, although none of the refined_after_k can outperform the retrained network. The second fact that we observe is that delaying the learning on the convolutional layers delays the learning, so the refined_after_1000 and refined_after_2500 are the networks need more iterations to converge, specially the last one. We can also observe that when the learning is activated in the convolutional layers the performance drops suddenly, although it recovers after some iterations.

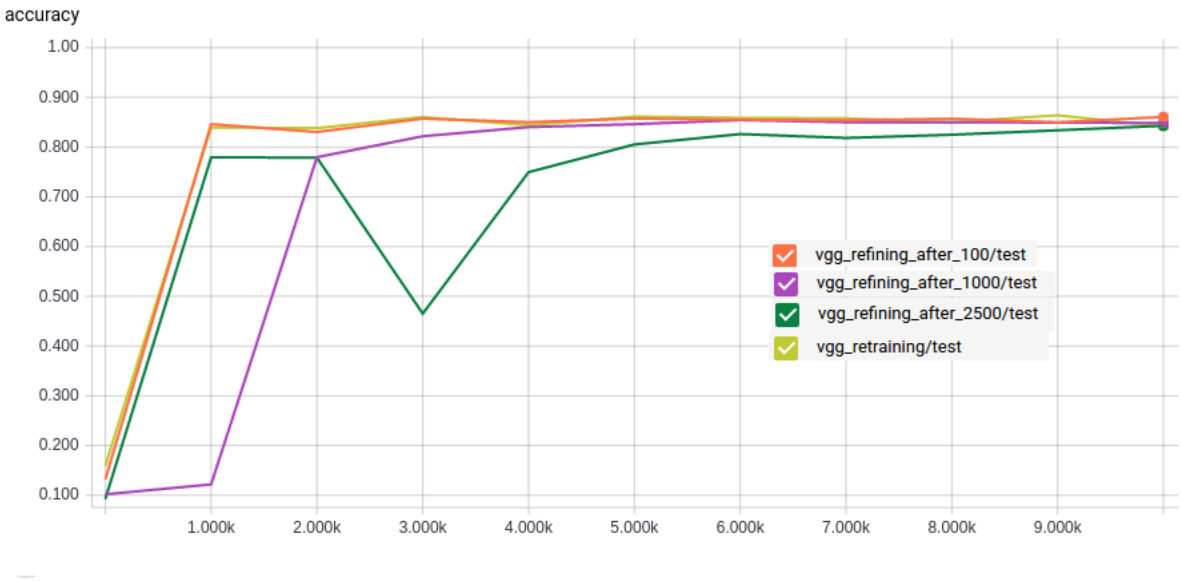


Figure 11: Test accuracy, for the retrained network, and the different refine_after_k networks.

In figure 12 we observe the accuracy of the same experiment, but in the train set. We can appreciate that also the accuracy drops in the case of refine_after_1000.

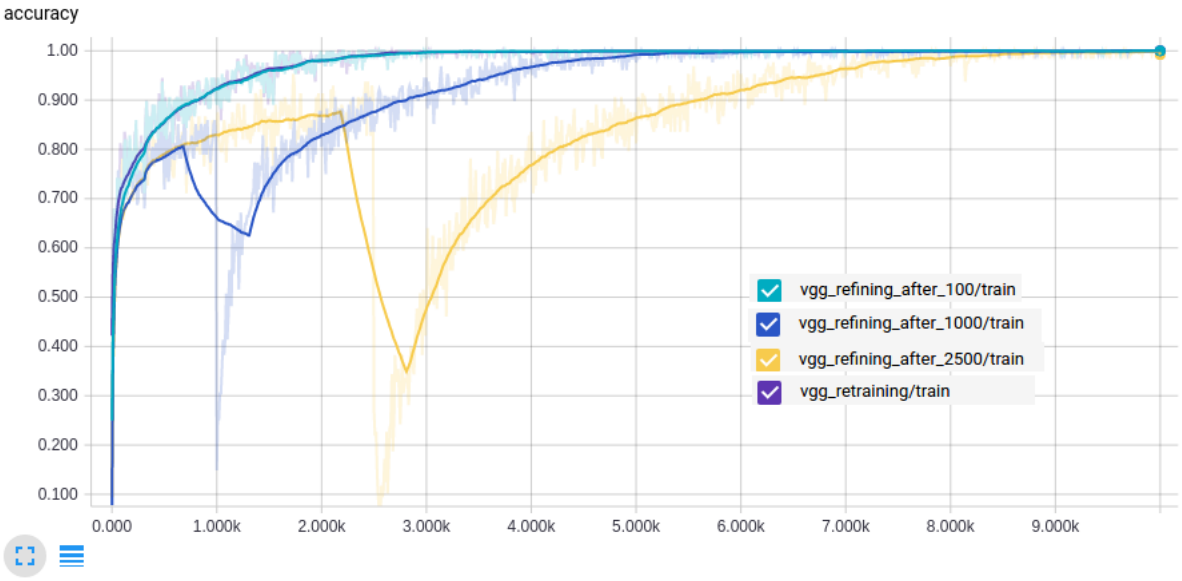


Figure 12: Train accuracy, for the retrained network, and the different refine_after_k networks.

In figure 13 we observe the weight distribution of the weights in the first convolutional layer in the refine_after_2500 network. As we see, the distribution starts to change after the first 2500 steps.

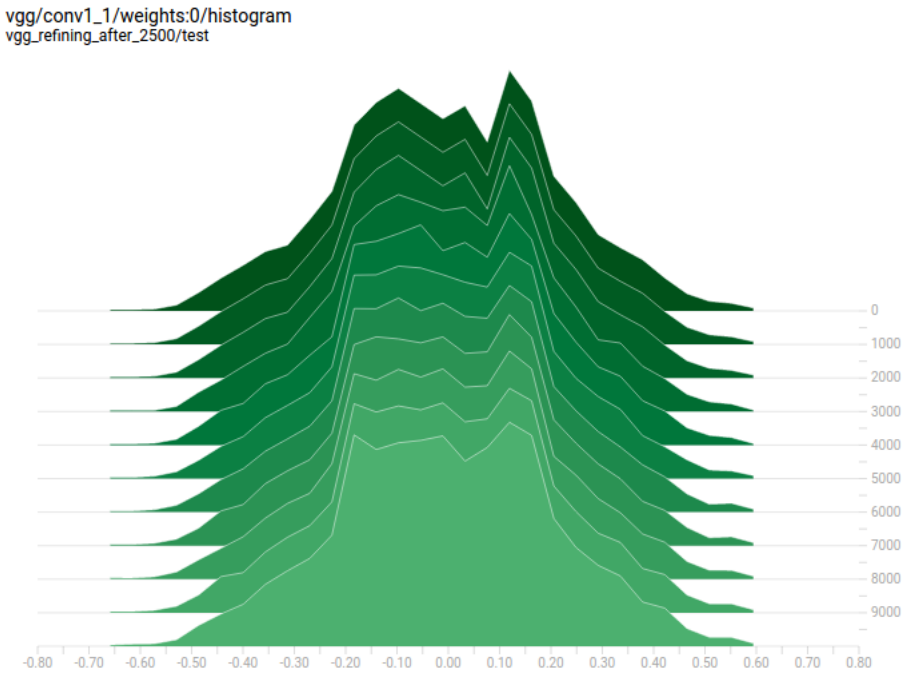


Figure 13: Weight histogram on the first convolutional layer for the refine_after_2500 network.

3.4 Further improvements

A factor that can influence the performance is the selected layers to transfer learning. The first layers of the network represent basic features (eg edge detectors), so using a pre-trained network seems an advantage to increase the performance. However, deeper layers can represent more specific features to the task for what the network has been trained, so it is possible that adding these pre-trained networks

doesn't help to achieve better performance. For example, some parts of the latest convolutional layers in the VGG net can be used to represent features specific to classes that exist in ImageNet but not in cifar10. Therefore, an interesting experiment is retrain or refine the parameters of the latest convolutional layers in the VGG part, and don't update the parameters of the first convolutional layers.

The factors that can influence the performance and have to be taken account in order to decide which layers fine-tune are mainly the size of the new dataset and the similarity. If the new data set is small we have the risk to overfit if we decide to fine-tune the convolutional layers. If the similarity between the data sets is high, then we can re-use more convolutional layers, as the features that they represent can be also useful in the new dataset. On the contrary, if there is a high dissimilarity then only the first layers would be useful.

In figure 14 we can see the results of a network where the convolutional layers 3 and 4 are retrained, but not the convolutional layers 1 and 2. In this case, we observe that the results are very similar to the completely retrained network, so there is no improvement. We could do more experiments trying to fine-tune different convolutional layers.

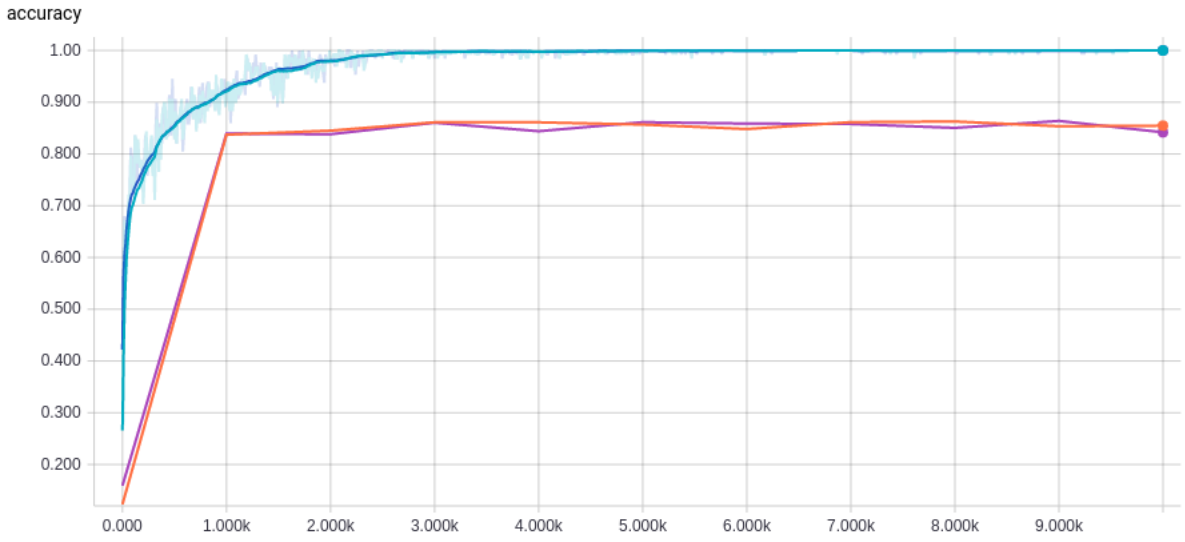


Figure 14: Train and test accuracy, for the completely retrained network, and a retrained network where the convolutional layers 3 and 4 are retrained, but not the convolutional layers 1 and 2.

4 Conclusion

In this assignment we have seen that CNN are very powerful in image classification, using different strategies. The best obtained accuracy is 0.8639, which outperforms with a high difference the 0.5726 obtained in the previous assignment with a MLP.