Deep Convolutional Network and Transfer Learning

Haakon Hukkelaas
haakohu@stud.ntnu.no

Stian Rikstad Hanssen
hanssen.stian@gmail.com

Chao Long
c3long@eng.ucsd.edu

Abstract

In this assignment we will look at classifying images from the CIFAR-10 dataset, and the Caltech256 dataset. We will first look at three different deep convolutional networks to classify images from the CIFAR-10 dataset. Afterwards we will experiment with transfer learning on a VGG16 network trained on ImageNet and transfer this to the Caltech 256 dataset. By these means, we achieved a accuracy of 85.28% on the CIFAR-10 dataset with a 6-layer CNN network, and a accuracy of 78.2% on the Caltech256 network, both based of the testing set.

1 Introduction

By using the python framework Torch, we will look into classifying 32×32 RGB Images from CIFAR-10 dataset consisting of 10 unique classes. By using methods like batch norm, dropout, max pooling and leaky ReLu we achieved a validation accuracy of 86.52% and a test accuracy of 85.28%.

The Caltech256 dataset consists of variant sizes of images, but we will scale all to 224×224. With the Caltech256 dataset we implemented a VGG16 network that was pretrained on ImageNet, and then trained the outermost layer of the network. By these means we achieved a validation accuracy of 74.5%, and a test accuracy of 78.2%.

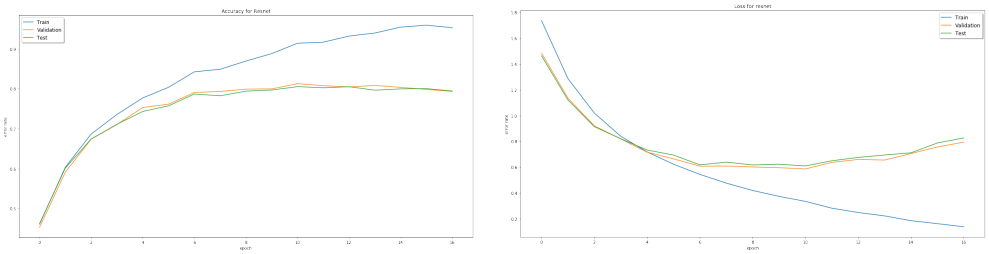
2 Deep Convolutional Network for Image Classification

The first implementation we tried out was Pytorch tutorial for Deep convultional network[1], which gave us an accuracy of about 64%. This was a simple 2-layer neural network, and we based our implementation on this. Through experiments with batch normalization, dropout, replacing max pooling with increased strides, different weight initialization, and different optimized we ended with these 3 models. The top model had a top accuracy of 85.28% on the test set, and 86.52% on the validation set.

2.1 Model 1 and Result

Figure 1 and Table 1 are the result for Model 1. In this model, we applied simple resnet combining with fully connected layers and convolution layers. The accuracy is 79.32% for validation set and 79.42% for testing set. And the training process stops at 16 epochs by early stopping. Besides, we applied Xavier initialization and Adam Optimizer in the training process, and the size of the validation set is 5000.

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.



(a) Accuracy (b) Cross entropy loss

Figure 1: Performance of model 1. Results after 18 epochs, reaching a maximum validation accuracy of 79.32%, and testing accuracy of 79.42%.

Table 1: Neuron Network Architecture for Model 1

Layer	Method	Output Channel Size
Input layer		3
Conv	kernel size(3, 3) stride=1 padding=1 batch normalization	16
Resnet	residual block with two Conv layers((16, 16)(16, 16)) and a shortcut for input layer batch normalization relu	16
Resnet	residual block withtwo Conv layers((16, 16)(16, 32)) and a shortcut for input layer batch normalization relu	32
Resnet	residual block with two Conv layers((32, 32)(32, 64)) and a shortcut for input layer batch normalization relu	64
Maxpool	kernel size 8	64
Fully Connected	linear transform from 64 to 10	10

2.2 Model 2 and Result

Figure 2 and Table 2 are the result and architecture for Model 2. For Model 2, we use 6 convolution layers and 3 fully connected layers. The accuracy is 86.52%for validation set and 85.28% for testing set and the training process stopped at at 144 epochs. Besides, we use Xavier initialization and Stochastic gradient descent with momentum(0.9), and the size of validation set is 5000.

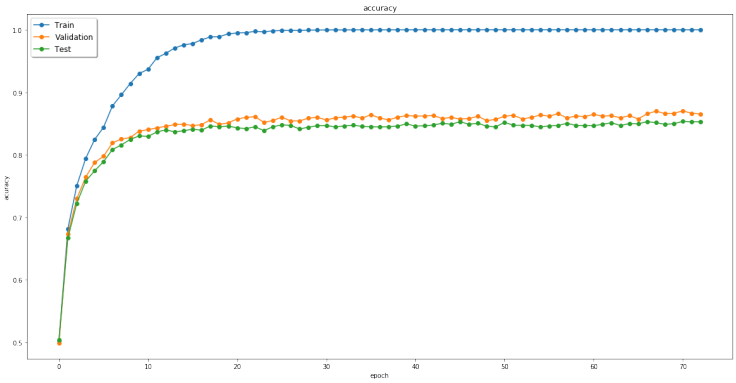


Figure 2: Performance of model 2(Best Model). Results after $144(72 * 2(displaystep))$ epochs, reaching a maximum validation accuracy of 86.52%, and testing accuracy of 85.28%.

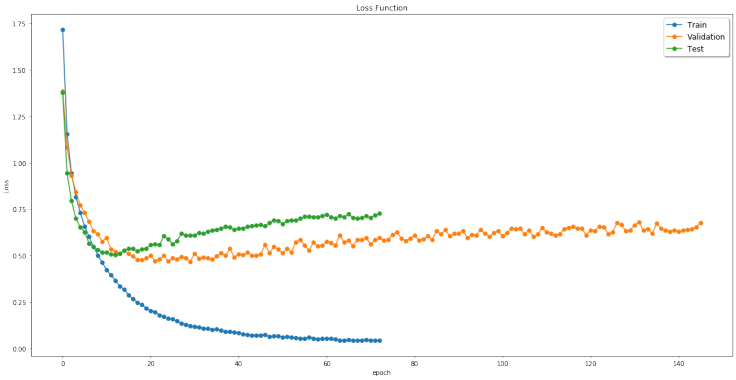


Figure 3: Cross entropy loss of model 2(Best Model). Early stopping at 144 epochs.

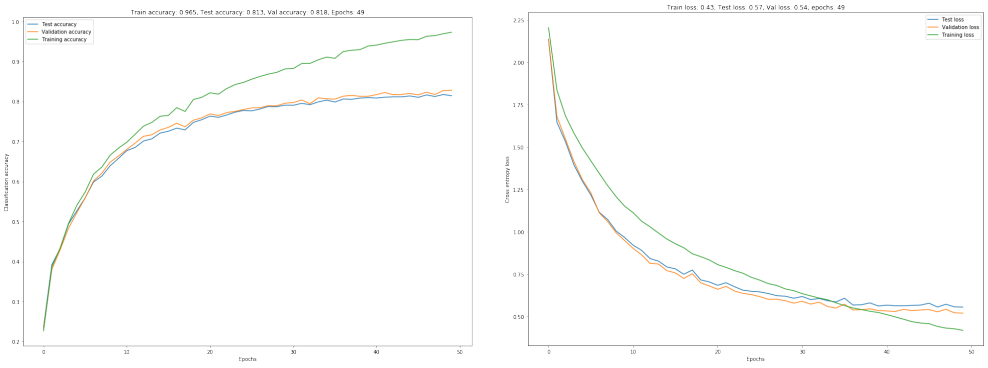
Table 2: Neuron Network Architecture for Model 2

Layer	Method	Output Channel Size
Input layer		3
Conv	kernel size(3 3) stride=1 padding=0 batch normalization relu	48
Conv	kernel size(3, 3) stride=1 padding=0 batch normalization relu maxpooling(2) dropout(0.5)	48
Conv	kernel size(3, 3) stride=1 padding=0 batch normalization relu	96
Conv	kernel size(3, 3) stride=1 padding=1 batch normalization relu maxpooling(2) dropout(0.5)	96
Conv	kernel size(3, 3) stride=1 padding=0 batch normalization relu	192
Conv	kernel size(3, 3) stride=1 padding=1 batch normalization relu maxpooling(2) dropout(0.5)	192
Fully Connected Layer	linear transform from 3*3*192 to 512 relu dropout(0.25)	512
Fully Connected Layer	linear transform from 512 to 128 relu dropout(0.25)	128
Fully Connected Layer	linear transform from 128 to 10	10

2.3 Model 3 and Result

Model 3 is a deep convolutional network with 6 convolutional layers, and a single fully connected layer. The full network structure can be seen in Table 3. This is using a Leaky ReLU activation, instead of the regular ReLU which was used in model 1 and 2. After 49 epochs it reached a validation

accuracy of 81.8%, and a testing accuracy of 81.3%. This model also has a dropout layer after each layer which gave it a very slow learning rate, but less overfitting which we can clearly see in Figure 4



(a) Accuracy (b) Cross entropy loss

Figure 4: Performance of model 3. Results after 49 epochs, reaching a maximum validation accuracy of 81.8%, and testing accuracy of 81.3%.

Table 3: Neuron Network Architecture for Model 3

Layer	Method	Output Channel Size
Input layer		3
Conv	kernel size(3, 3) stride=1 padding=1 batch normalization Leaky ReLU dropout(0.5)	32
Conv	kernel size(3, 3) stride=1 padding=1 batch normalization Leaky ReLU maxpooling(2) dropout(0.5)	64
Conv	kernel size(3, 3) stride=1 padding=1 batch normalization Leaky ReLU dropout(0.5)	128
Conv	kernel size(3, 3) stride=1 padding=1 batch normalization Leaky ReLU maxpooling(2) dropout(0.5)	256
Conv	kernel size(3, 3) stride=1 padding=1 batch normalization Leaky ReLU dropout(0.5)	512
Conv	kernel size(3, 3) stride=1 padding=1 batch normalization Leaky ReLU maxpooling(2) dropout(0.5)	1024
Fully Connected Layer	linear transform from 1024*4*4 to 10	10

2.4 Discussion

2.4.1 Changing Max pooling with stride

We looked at swapping out 2×2 max pooling with a stride of 2, however with this we experienced often a decrease in performance in terms of validation set accuracy.

2.4.2 Max pooling and Average pooling

We tried Max pooling and Average pooling in Model 2, and from our observations Max pooling outperforms average pooling in accuracy. This is reasonable since Average pooling sometimes can't extract good features because it takes all into count and results an average value while Max pooling extracts the most important features like edges.

2.4.3 Batch Normalization and Dropout

In our models, we used batch normalization and dropout methods and we noticed a significant improvement of performance. We know dropout is a good method for regularization. While dropout is applied, some hidden units will be marked as 0, thus it can prevent overfitting. For batch normalization, it can be seen as a method of regularization as well as normalization.

2.4.4 Resnet architecture

In Model 1, we tried resnet and it works better than the simple network from the pytorch tutorial[1]. Resnet is a new architecture for convolution neuron network, where it applies a shortcut to the current network before relu activation. It can prevent the weights from exploding. Thus the optimal result in our experiment is reasonable.

2.4.5 Xavier initialization

In all our models we used Xavier initialization. From the first experiments with this based on the simple network from the pytorch tutorial [1], we observed consistently better performance then without it. Therefore we continued using this initialization in all of our models. Xavier initialization improves the performance of the initialization such that the weights are neither too small or too big, reducing chance of weight explosion, or signal exploding.

2.4.6 Adam Optimizer

We tried Adam optimizer on the basic pytorch network [1], and observed a faster convergence rate. Therefore we decided to use this optimizer in all our models. It is known to be one of the better optimizer currently, as it realizes the benefits of both AdaGrad and RMSProp, which both accelerate the speed of learning.

2.5 Data Augmentation

We tried some data augmentation methods like randomCrop and randomhorizontalflip in Model 3 and observed better performance regarding accuracy. This is reasonable because we can obtain more useful data by cropping, rotating and flipping the input images. And generally, the more data we used for training, the better result will obtain in deep neuron networks.

3 Transfer learning

By using the pre-trained VGG16 network we added a new softmax-layer at the end with 256 outputs. We selected at random 32 images from each class for our training set, and 8 images per class for our testing set. Of the 32 in the training set 10% is taken out as a validation set. By training this with AdamOptimizer, and a learning rate of 0.000002, we achieved the results shown in Figure 5 and Figure 6. We achieved a validation accuracy of 74.5%, and a testing accuracy of 78.2%.

We noted a huge improvement of the network by normalizing the data by the use of the normalization that was used on the training. The normalization we used had a mean of $[0.485, 0.456, 0.406]$, and a standard deviation of $[0.229, 0.224, 0.225]$ on the three color channels Red, Green and Blue respectively.

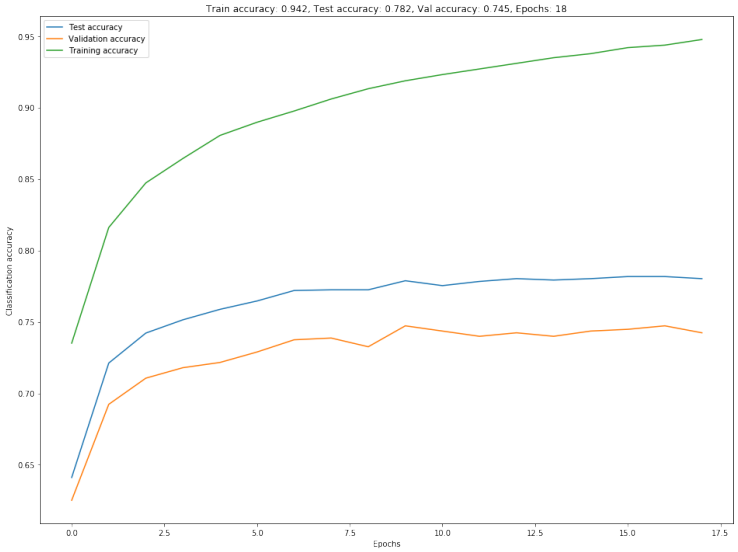


Figure 5: Classification accuracy of vgg16 model trained on Caltech256 dataset. Result after 18 epochs of training. Training accuracy = 94.2%, testing accuracy = 78.2% and validation accuracy = 74.5%.

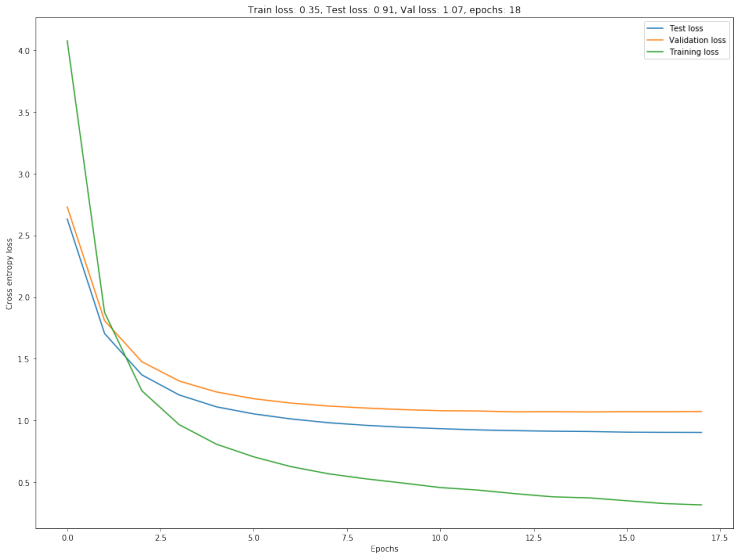


Figure 6: Cross-entropy loss of VGG16 model trained on Caltech256 dataset. Result after 18 epochs of training. Training loss = 0.35, testing accuracy = 0.91 and validation accuracy = 1.07.

3.1 Activation and weight visualization

3.1.1 Activations form various filters

By passing the images through the network and looking at the activation for the first and last convolution layer we obtain the results seen in figure 7, figure 8 and figure 9.

Figure 7: Activations for filter 0 in both layers. Images in order left to right: Normal image, first convolution layer activation and last convolution layer activation.

Figure 8: Activations for filter 1 in both layers. Images in order left to right: Normal image, first convolution layer activation and last convolution layer activation.

Figure 9: Activations for filter 2 in both layers. Images in order left to right: Normal image, first convolution layer activation and last convolution layer activation.

3.1.2 Weights of filters and their effect

A filter is of the shape 3x3x3 where the last dimension is shown as color. The original images are also 3-dimensional as they have color. Thus the filter, as it only places one value at a time, reduces the images down to two dimensions resulting in the black and white images shown below.

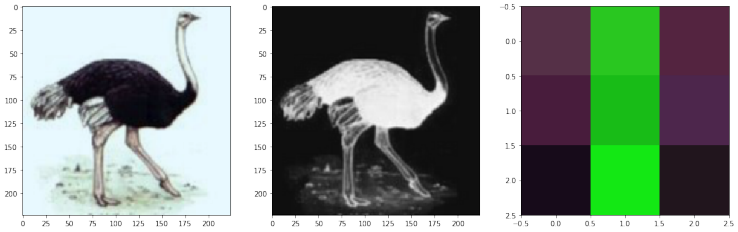


Figure 10: Weight of filter 12. Images shown in order left to right: Normal image, first convolution layer activation and the filter's weight.

As seen in figure 10 the filter works as an inverter. From both left and right side of the placement point in the kernel we have a dark color meaning all 3 color layers are low. The middle column is green meaning the green layer in the color dimension have high values in that column. This results to seemingly an inversion, though it is really a highlight of specific colors, while others are reduced to low values.

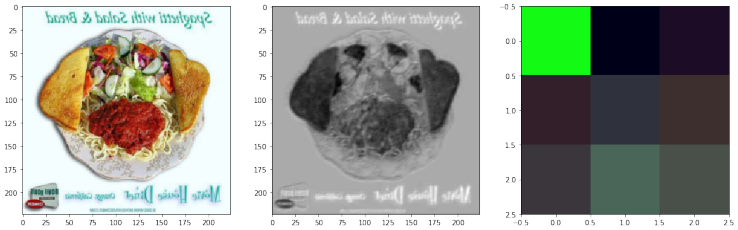


Figure 11: Weight of filter 13. Images shown in order left to right: Normal image, first convolution layer activation and the filter's weight.

The filter seen in figure 11 is in many ways similar to figure 10. This filter reduces all colors while convolving except for greens, meaning that if there is a green within the 3 by 3 area, the middle pixel will be highlighted more than others. This can be seen in the picture as values close to green remain lighter while other colors are darkened.

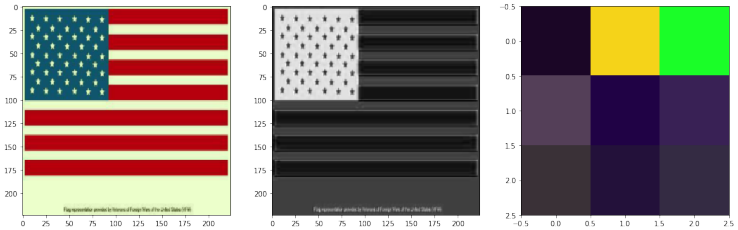


Figure 12: Weight of filter 37. Images shown in order left to right: Normal image, first convolution layer activation and the filter's weight.

The filter in figure 12 holds a lot of blue which can be seen in the activation with the very bright area where there was blue in the original image. It can also be seen that other colors such as yellows and greens are brighter than reds as the filter favors a little bit of yellow and green if it is above the center pixel in the filter. As these two colors are above the center of the filter, one can notice that the lines in the American flag are displaced slightly more down in the activation.

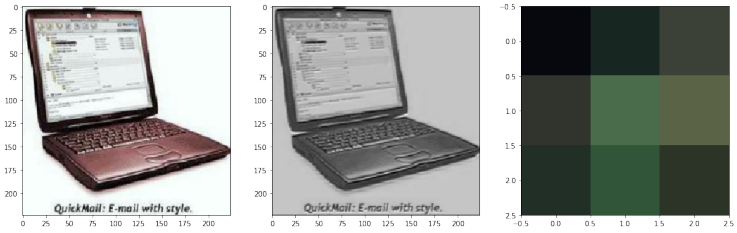


Figure 13: Weight of filter 41. Images shown in order left to right: Normal image, first convolution layer activation and the filter's weight.

Figure 13 shows a filter that is somewhat uniform. Most colors in the filter are dark in a tone of green. This small variation in the filter is reflected in the activation shown as it looks like a regular gray image. It is likely it would make greens a bit brighter than other colors.

3.2 Feature extraction

By using only parts of the pretrained VGG16 network, we can look at the networks classification accuracy by doing a softmax probe. We will look at the classification after 3 and 4 convolutional blocks. From these features we will add a 1024 large fully connected hidden layer, then into the outer softmax layer.

Due to memory limitations on the GPU cluster, we scaled the images down to 112×112 resolution. We also tried to fix this by reducing the batch size, but this resulted in the only way of running it being a batch size of 1. Another option we considered was adding another pooling layer at the end of the feature extraction, however we achieved satisfying results by scaling the images down.

Both of the networks was trained on learning rate of 0.000002, and a batch size of 16.

3.2.1 Feature extraction from 3 convolutional blocks

By using 3 convolutional blocks from the VGG16 network, we achieved a top accuracy of 24.8% on the testing set, and 23.4% on the validation set. This can be seen in Figure 14 and Figure 15. We notice that this network is overfitting quickly, and much more than the network with 4 convolutional blocks.

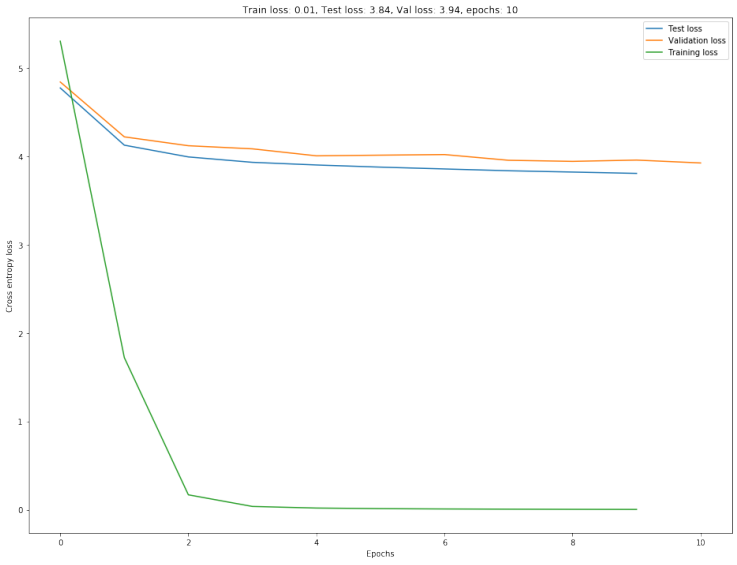


Figure 14: Cross entropy loss of 4 convolutional blocks from VGG16 network trained after 16 epochs. Train loss= 0.01, test loss=3.04, and validation loss = 3.94.

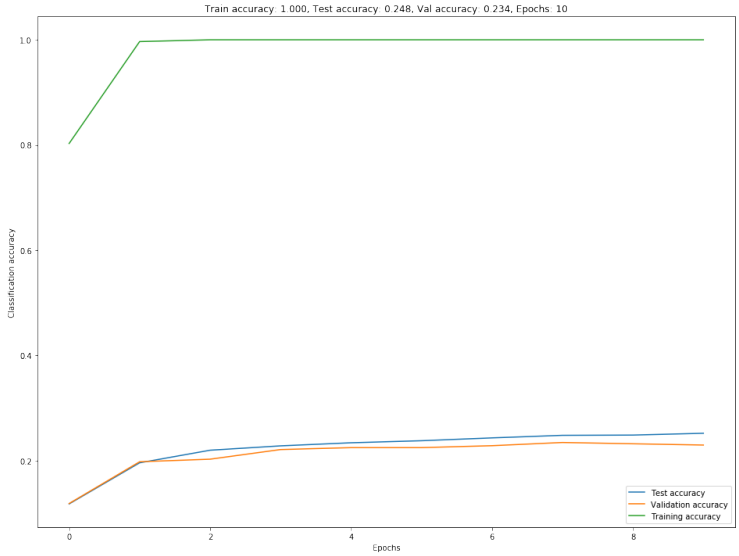


Figure 15: Classification accuracy of 4 convolutional blocks from VGG16 network trained over 16 epochs. Train accuracy = 100%, test accuracy = 24.8%, and validation accuracy=23.4%.

3.2.2 Feature extraction from 4 convolutional blocks

By using 4 convolutional blocks from the VGG16 network, we achieved an top accuracy of 49.6% on the testing set, and 47.9% on the validation set. This can be seen in Figure 16 and Figure 17.

We notice an huge overfitting on the training set, where we see the big gap between the training set and the validation set. The network is still able to fully learn the training set, and this is probably the fully connected network that is able to memorize some of the training examples. This might get better if we selected a larger training set from the Caltech256 dataset.

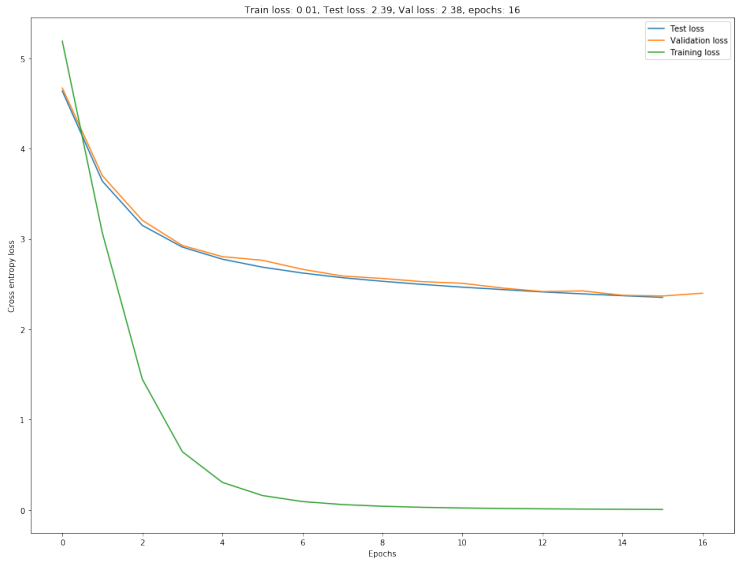


Figure 16: Cross entropy loss of 4 convolutional blocks from VGG16 network trained after 16 epochs. Train loss= 0.01, test loss=2.39, and validation loss = 2.38.

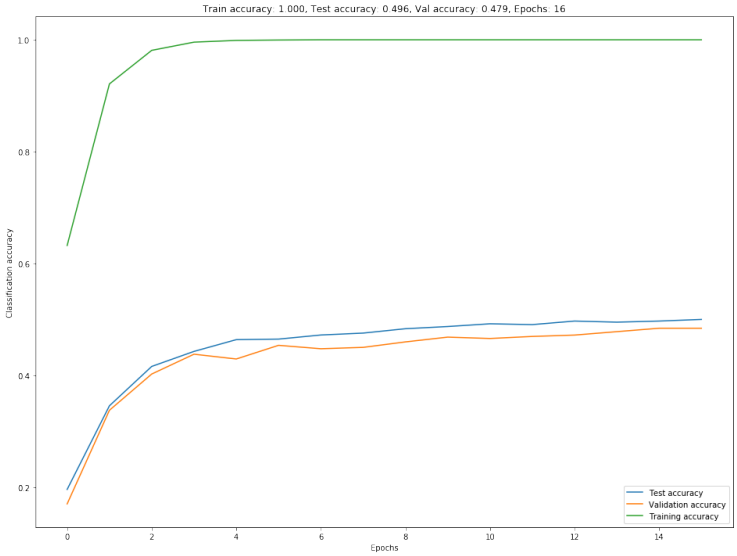


Figure 17: Classification accuracy of 4 convolutional blocks from VGG16 network trained over 16 epochs. Train accuracy = 100%, test accuracy = 49.6%, and validation accuracy=47.9%.

4 Team member contribution

4.1 Stian Rikstad Hanssen

Implementing the visualization of activations and weights in the convolutional network. Analyzing the effect of various kernels on images. Setting up the early implementation of transfer learning, later switched in favor of Haakon's implementation.

4.2 Haakon Hukkelaas

Implemented the main framework for mini-batch gradient descent, and model 3 in task 2. Implemented transfer learning on VGG, as well as the probing part.

4.3 Chao Long

Implemented Model 1 and Model 2 and explored methods to improve the performance of convolution neural network.

References

[1] Training a classifier. `http://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html`.