TDT4265 Assignment 3

Kristian Haga

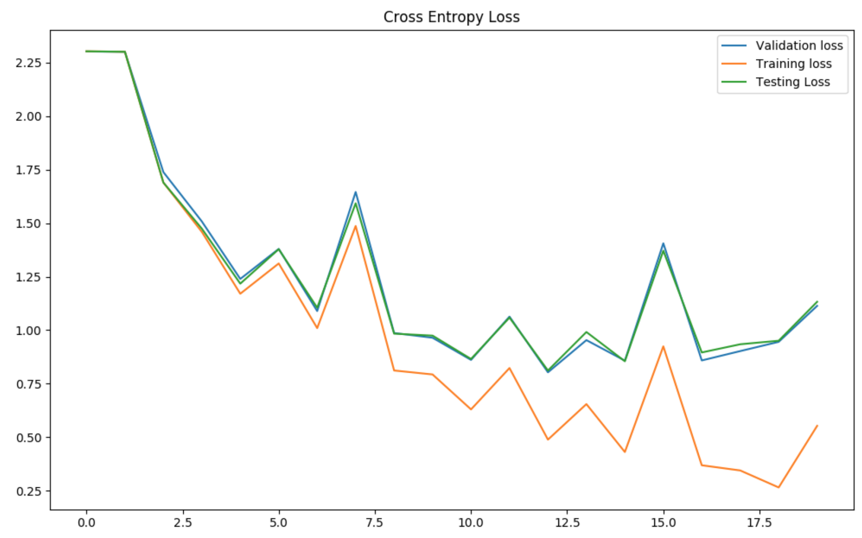
February 2019

1 Task 1

1.1 Task a

This task was completed by implementing the class named `LeNetModel(nn.Module)` in the source code.

Recorded loss during training

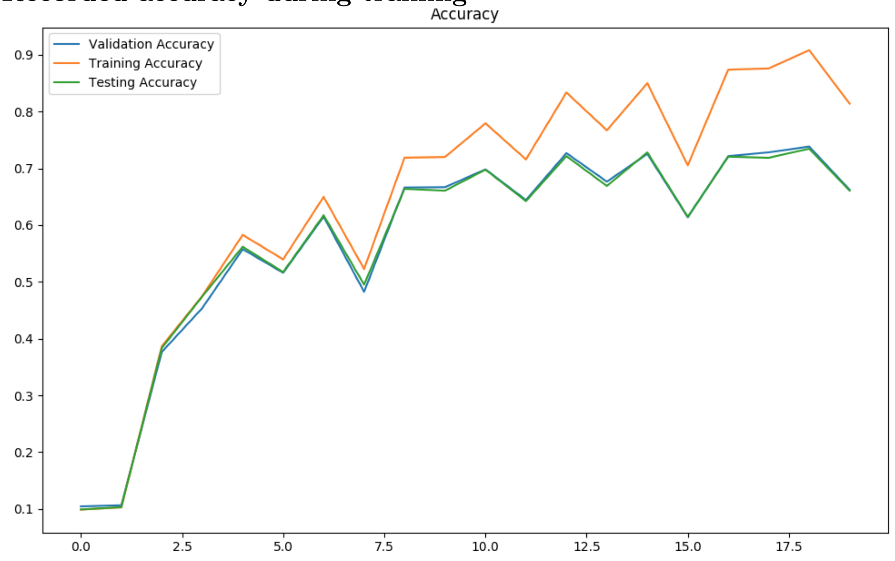


1.2 Task b

The recorded accuracies were:

- Validation set - 0.7214
- Test set - 0.7206
- Training set - 0.9041

Recorded accuracy during training



1.3 Task c

$$parameters\ in\ first\ convolution\ layer\ = F_H * F_W * C_1 * C_2 = 2400 \tag{1}$$

$$parameters\ in\ first\ layer\ = convolution\ parms + biases = 2432 \tag{2}$$

$$parameters\ in\ conv_2\ layer\ = 51264 \tag{3}$$

$$parameters\ in\ conv_3\ layer\ = 204928 \tag{4}$$

$$params\ in\ lin\ layer\ = out\ features *\ num\ outputs + biases = 131136 \tag{5}$$

$$parameters\ in\ lin_2\ layer\ = 650 \tag{6}$$

$$parameters\ in\ total = 390410 \tag{7}$$

2 Task 2

2.1 Network architectures

Architecture Model 2.1		
Layer Type	Number of units	Activation Function
Conv2D	32	ReLU
BatchNorm2D	-	-
Conv2D	64	ReLU
BatchNorm2D	-	-
MaxPool2D	-	-
Conv2D	128	ReLU
BatchNorm2D	-	-
Conv2D	256	ReLU
BatchNorm2D	-	-
MaxPool2D	-	-
Flatten	-	-
Dense	64	ReLU
Dense	10	Softmax

For Model 2.1 the solution used SGD as optimizer, a learning rate of 0.05, batch size of 64 and the weights in the convolution layers were initialized using Xavier Initialization. All convolution layers applies a 3x3 filter size with both stride and padding set to 1. By setting the padding to 1 we preserve the size of the image through each convolution layer.

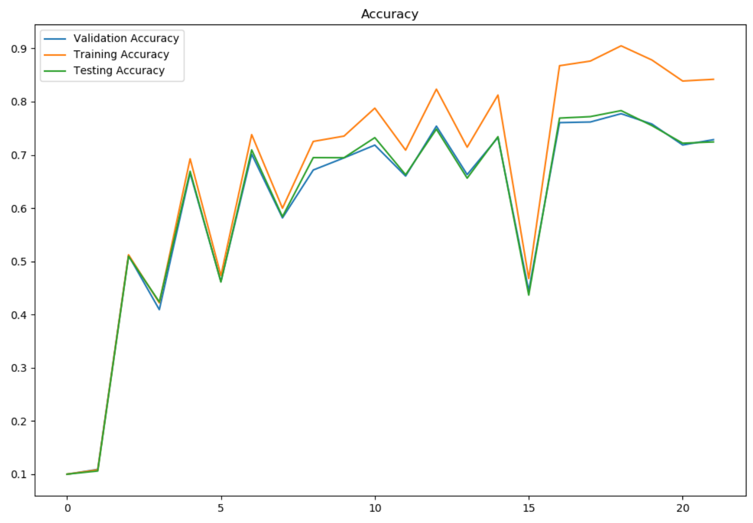
Architecture Model 2.2		
Layer Type	Number of units	Activation Function
Conv2D	32	ReLU
BatchNorm2D	-	-
Conv2D	32	ReLU
Dropout2D	-	-
BatchNorm2D	-	-
MaxPool2D	-	-
Conv2D	64	ReLU
BatchNorm2D	-	-
Conv2D	64	ReLU
Dropout2D	-	-
BatchNorm2D	-	-
MaxPool2D	-	-
Conv2D	128	ReLU
BatchNorm2D	-	-
Conv2D	128	ReLU
Dropout2D	-	-
BatchNorm2D	-	-
MaxPool2D	-	-
Flatten	-	-
Dense	64	ReLU
BatchNorm1d	-	-
Dense	10	Softmax

For Model 2.2 the solution used a batch size of 32, Adam optimizer with its default parameters and Xavier Initialization to initialize the convolutional weights. All convolution layers applies a 3x3 filter size with both stride and padding set to 1. By setting the padding to 1 we preserve the size of the image through each convolution layer.

2.2 Recorded loss and accuracy

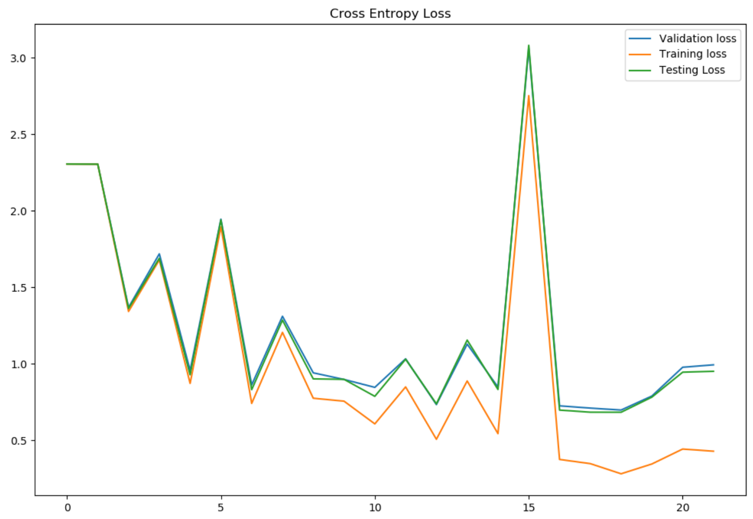
	Test		Validation		Train	
	Loss	Acc	Loss	Acc	Loss	Acc
Model 2.1	0.933	0.806	0.852	0.823	0.003	0.909
Model 2.2	0.714	0.783	0.717	0.782	0.258	0.912

2.3 Recorded Accuracy



Note that for a smoother graph the loss and accuracy where recorded at a rate of `len(self.dataloader_train) // 2`. As a result the numerical values on the horizontal axis are not equal to the number of epochs.

2.4 Recorded Loss



Note that for a smoother graph the loss and accuracy where recorded at a rate of `len(self.dataloader_train) // 2`. As a result the numerical values on the horizontal axis are not equal to the number of epochs.

2.5 Discussion

We observed an increase in accuracy by applying a smaller filter, e.i 3x3 instead of 5x5.

The smaller filter size has a smaller receptive field because it looks at fewer pixels at once. As a result highly local and complex features can be extracted. It also enhances weight sharing.

We did not find a good answer for what number of channels is better. By having more channels, a wider network can look for more features, but it comes at an expense of computational performance and can limit how deep we can make the network. A deeper network is able to look for features inside other features. A suggestion for further work is to periodically widen and deepen the network being developed, and by trial and error find what works best for the specific data set.

We also experienced an increase in learning speed by doing Xavier Initialization. Xavier Initialization initialize the weights such that the variance remains the same with each passing layer. The result of this is that we keep the signal from exploding to a high value or become zero.

Note that even with Xavier Initialization Model 2.1 is very unstable compared to Model 2.2. This might also be a result of using SGD as optimizer instead of the Adam optimizer used in Model 2.2.

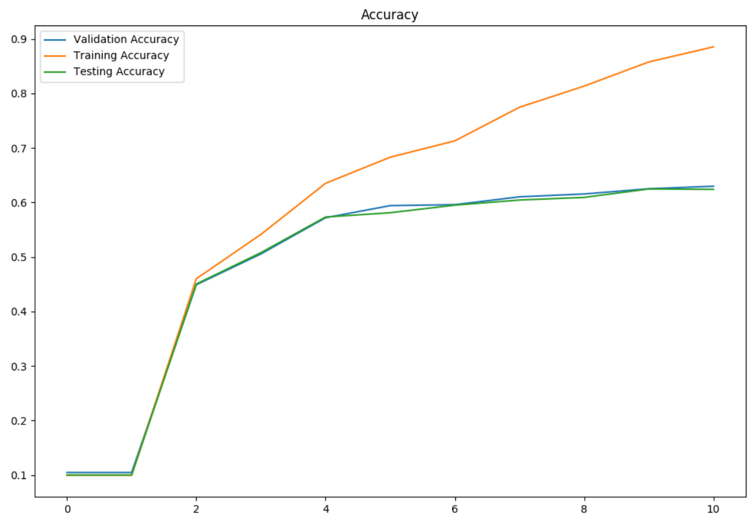
By adding dropout in the second model the learning speed decreased, but we experienced a more ”slow and steady” curve for both accuracy and loss.

3 Task 3

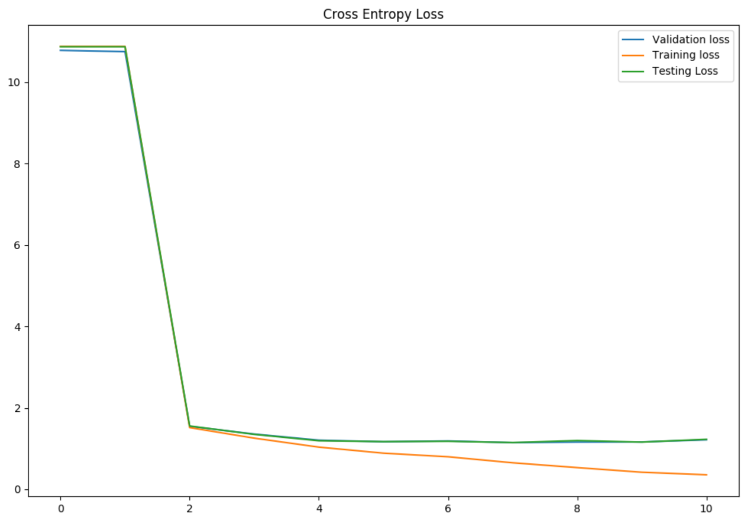
3.1 Hyperparameters in Task 3a

For the ResNet18 model with transfer learning implemented in the file `task3_resnet.py` we used an adam optimizer with a learning rate of $5 * 10^{-4}$ and a batch size of 32. No data augmentation were used.

3.2 Task b - Recorded accuracy

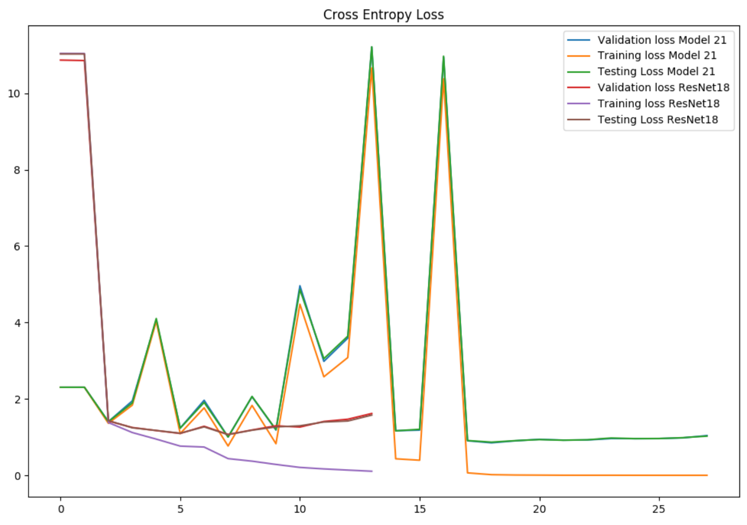


3.3 Task c -Recorded loss



3.4 Task d - Comparison with Model 2.1

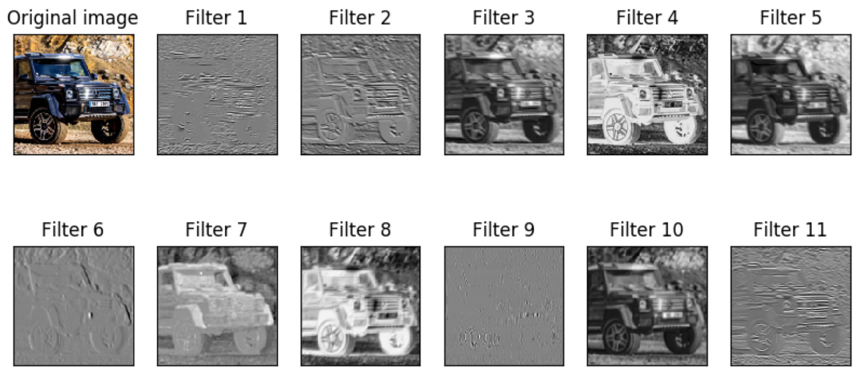
Recorded Loss



As stated previously Model 2.1 is highly unstable and this is easy to verify in this plot. The recorded losses of Model 2.1 varies a lot for the first couple of epochs before the recorded losses stabilizes. Note that the losses for model 2.1 stabilizes a few epochs after ResNet18 hit early stopping. For the pre-trained ResNet18 model we observe that the model makes a drastic decrease in loss after the first epoch and then steady improves before early stopping kicks in. The Transfer Learning clearly improves learning speed and makes the ResNet18 losses way more stable.

3.5 Task e - Filters in First Convolutional Layer

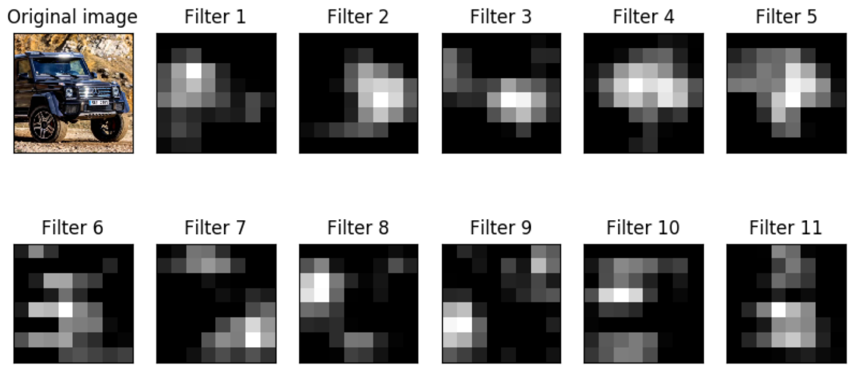
The solution to this task is implemented by
`def visualize_first_conv_layer_filters_resnet18(image, image_normalized, num_filters):` in `task3_visualisation.py` (don't mind the typo).



We observe that in the first convolutional layer the filters seems to be looking for well known shapes like straight lines and squares. The network is looking for spatial features.

3.6 Task f - Filters in Last Convolutional Layer

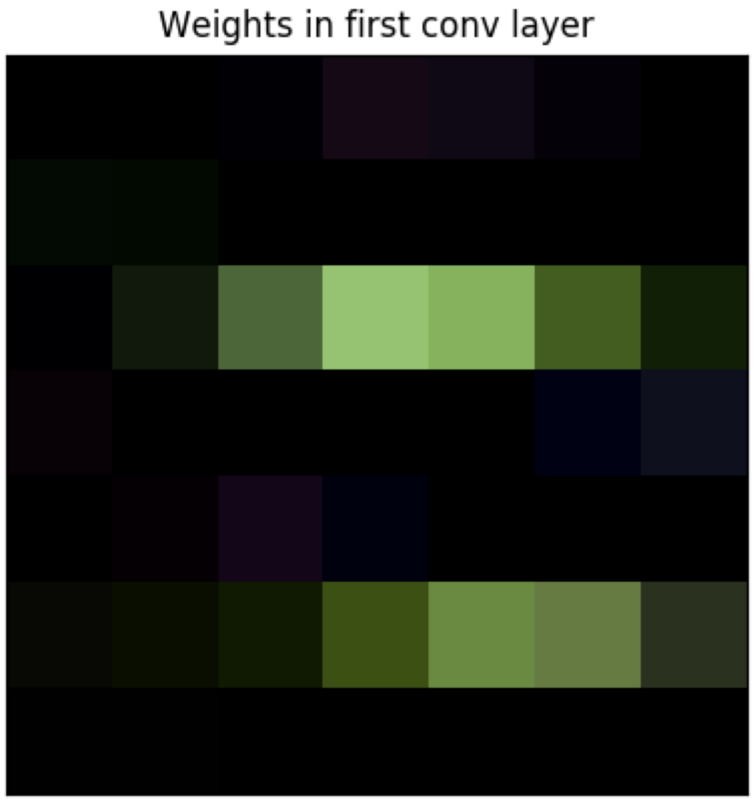
The solution to this task is implemented by
`def visualize_last_conv_layer_filters_resnet18(image, image_normalized, num_filters):`



In the last convolutional layer filters it is hard to determine what kind of features the network is looking for. We are now so 'deep' in the CNN that the abstraction of the original image and the features the CNN are looking for are not trivially clear to a human.

3.7 Task g - Weights in First Convolutional layer

The implementation of the solution of this task is
`def visualize_weights_first_conv_layer():`



The plot of the weights show that the network is recognizing many spatial features in the center of the image. This corresponds with the visualization of the filters in the first layer.