Task 1

Assignment 1 Report

Task 1a)

First of, we handle boundary condition by implementing zero-pooling. Given that the stride is 1 and as the convolved image should perserve the spatial size we determine the size of zero-padding by

$$P = (F - 1)/2 = (3 - 1)/2 = 1$$

In [2]:

```
import numpy as np
input_data = np.array(
    [
        [0, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 2, 3, 1, 0],
        [0, 3, 2, 0, 7, 0, 0],
        [0, 0, 6, 1, 1, 4, 0],
        [0, 0, 0, 0, 0, 0, 0]
    ]
)
flipped_kernel = np.array([
    [1, 0, -1],
    [2, 0, -2],
    [1, 0, -1]
])

convolved_output = np.zeros((3,5))

for i in range(1, input_data.shape[0]-1):
    for j in range(1, input_data.shape[1]-1):
        convolved_output[i-1, j-1] = np.sum(np.multiply(kernel, input_data[i-1:i+2, j-1:

convolved_output
```

Out[2]:

```
array([[ -2.,   1., -11.,   2.,  13.],
       [-10.,   4.,  -8.,  -2.,  18.],
       [-14.,   1.,   5.,  -6.,   9.]])
```

To calculate the spatial convolution, the original kernel/filter is flipped. Spatial convolution gives us the following output:

$$\begin{bmatrix} -2 & 1 & -11 & 2 & 13 \\ -10 & 4 & -8 & -2 & 18 \\ -14 & 1 & 5 & -6 & 9 \end{bmatrix}$$

Task 1b)

(iii) Max Pooling.

Max pooling increases invariance to translation as it disregards the spatial information, but keeps the max value in the area of inspection.

Why not (i) - Convolutional layer? From a mere intuitive standpoint, comparing how the output of a convolutional layer and a max pooling layer is calculated, it seems self explanatory that the max pooling is less prone to translational variations. If we look at the filter used in task 1a), a translational shift will also shift the outputs. Max pooling discards spatial information - and "only cares about" the max value in the area, meaning that the layer will give the same output even if the input is shifted.

Why not (ii) activation function? Does not make sense (?).

Task 1c)

Similar to task 1a, we wish to perserve the spatial size through convolution. Using the same formula, with: $S = 1$ and $F = 5$

$$P = (F - 1)/2 = (5 - 1)/2 = 2$$

Task 1d)

Given equal width and height, we only need to calculate one of the dimensions. Calculating width:

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$F = -(W_2 - 1)S + W_1 + 2P$$

which gives, with our values:

$$F = -(504 - 1) * 1 + 512 + 2 * 0 = 9$$

(Height) x (Width) is 9x9. Depth = K = 12.

Task 1e)

Pooling layers accepts a volume of size W, H D, and takes hyperparameters F and S as input.

The layer produces a volume of size: $W_2 = \frac{W_1 - F}{S} + 1$

As we are working with equal width and height, we only have to calculate one of them:

$$W_2 = \frac{W_1 - F}{S} + 1 = \frac{504 - 2}{2} + 1 = 252$$

The layer produces an output of size: (Height) x (Width) = 252x252. Depth is perserved.

Task 1f)

Using the same equation as in task 1d. We now have input, assuming output from the pooling layer in task 1e, of size W,H,D = 252, 252, 12. Again, equal height and width:

$$W_2 = \frac{W_1 - F + 2P}{S} + 1 = \frac{252 - 3}{1} + 1 = 250$$

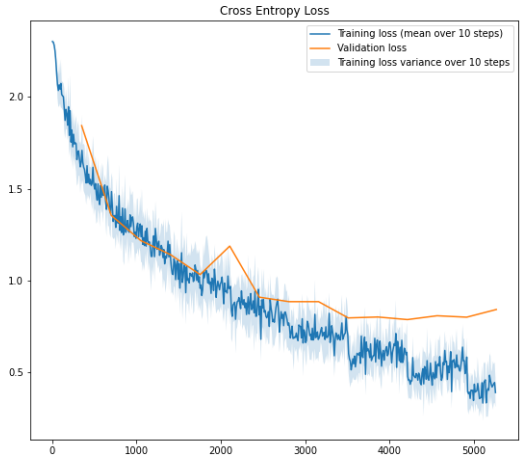
The size of the feature maps: (Height) x (Width) = 250x250

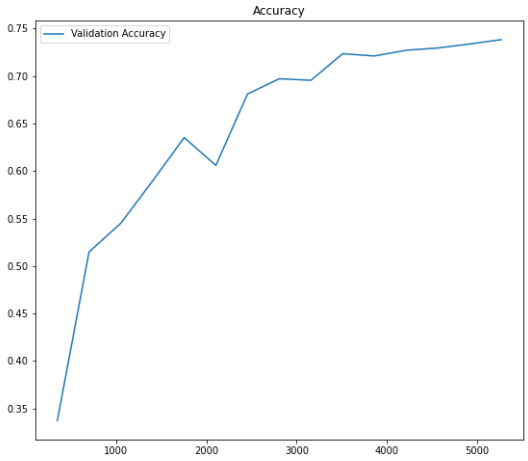
task 1g)

	Hyperparams	F	S	P		
	Conv	5	1	2		
	Pool	2	2	0		
	Layer	H	W	D	Hidden Units/Filters/Bias	No of Params
	Input	32	32	3		0
	conv	32	32	32	32	2432
	pool	16	16	32		0
	conv	16	16	64	64	51264
	pool	8	8	64		0
	conv	8	8	128	128	204928
	pool	4	4	128		0
	flatten	2048			1	
	fc	64			1	131136
	fc	10				650
					Total params	390410

Task 2

Task 2a)





Task 2b)

After 5 epocs early stopping kicks in Final training accuracy: 0.798 Final validation accuracy: 0.710 Final test accuracy: 0.704

Task 3

Task 3a)

Network 1

Network 1

Network one is implemented with the hyperparameters described in table 1 and architecture described in table 2. The convolutional layers used a kernel size of 3, and padding and stride of 1, and were implemented with ReLU activation function. After the activation function, we added batch normalization. For each pooling layer, we used a kernel size of 2x2, and a stride of 2. The weights were initialized with kaiming uniform, that is the default setting in PyTorch, and we used the SGD optimizer. Further on we used argumentation to extend our dataset. We used three different transformations: random horizontal flip, rotation, and crop.

Hyperparameters	
Epochs	10
Batch Size	64
Learning Rate	0.0005
Early Stop Count	4

Table 1

Layer	Layer Type	Number of Hidden Units/Number of Filters	Activation Function
1	Conv2d	32	ReLU
1	BatchNorm2d	-	-
2	Conv2d	-	ReLU
2	BatchNorm2d	-	-
2	MaxPool2d	-	-
3	Conv2d	64	ReLU
3	BatchNorm2d	-	-
3	Conv2d	-	ReLU
3	BatchNorm2d	-	-
3	MaxPool2d	-	-
4	Conv2d	128	ReLU
4	BatchNorm2d	-	-
4	Conv2d	-	ReLU
4	BatchNorm2d	-	-
4	MaxPool2d	-	-
	Flatten	-	-
6	Fully-Connected	64	ReLU
6	BatchNorm1d	-	-
7	Fully-Connected	10	Softmax

Table 2

Network 2

For the second Network, we used the same hyperparameters as for network 1, described in table 1. We changed the architecture and chose to apply one more layer with a trainable layer compared to the network in task 2. The architecture described in table 3. For this network, we implemented regularization, by using dropout with a 20% chance of dropout. In addition, we choose to replace all the ReLU activation functions with LeakyReLU. The convolutional layers used a kernel size of 5, padding of 2, and stride of 1. Every pooling layer used a kernel size of 2x2, and stride of 2. The weights were initialized with kaiming uniform, that is the default setting in PyTorch, and we used the SGD optimizer.

Layer	Layer Type	Number of Hidden Units/Number of Filters	Activation Function
1	Conv2d	32	LeakyReLU
1	MaxPool2d	-	-
1	Dropout	-	-
2	Conv2d	64	LeakyReLU
2	MaxPool2d	-	-
2	Dropout	-	-
3	Conv2d	128	LeakyReLU
3	MaxPool2d	-	-
3	Dropout	-	-
4	Conv2d	256	LeakyReLU
4	MaxPool2d	-	-
4	Dropout	-	-
	Flatten	-	-
5	Fully-Connected	64	LeakyReLU
5	Dropout	-	-
6	Fully-Connected	10	Softmax

Table 3

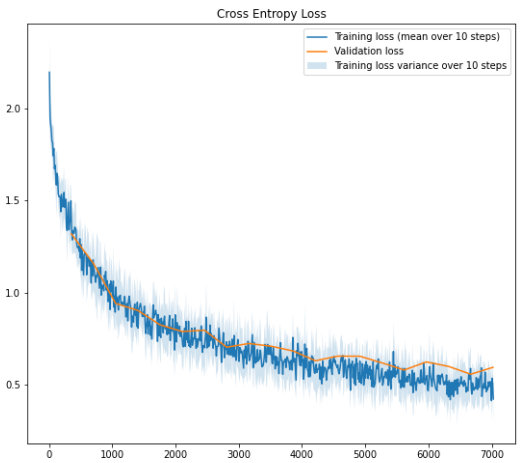
Task 3b)

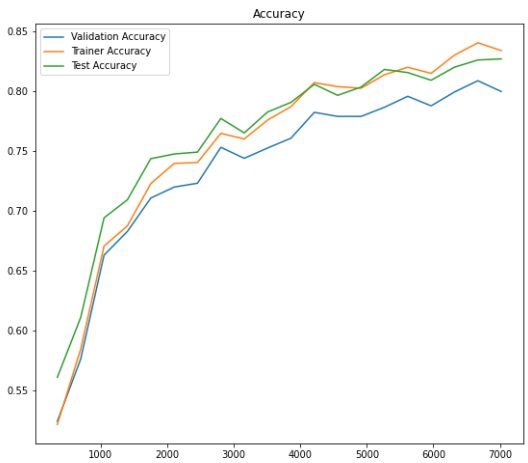
Final statistics from the two models.

Network	1	2
Training Loss	0.48	0.45
Training Accuracy	0.834	0.854
Validation Accuracy	0.800	0.770
Test Accuracy	0.827	0.771

Table 4

Plots from best model: Network 1





Task 3c)

Agumentation

We tried applying three different augmentation methods, a random horizontal flip, rotation, and crop. We tried each alone first and then all together to see if they improved the network. The horizontal flip increased the network's test accuracy to 0.769, rotation increased test accuracy to 0.739 and crop increased the network to 0.74. While adding them all together at the same time increased the accuracy to 0.752. We see that horizontal flip gave the largest increase alone.

Argumentation is used for extending your dataset by creating altered versions of the same image, resulting in more images to train on and therefore giving an increase in accuracy. The altered versions also give the classifier a wider variety to train on and therefore making the network more robust, it has a regularizing effect. There are many different transformations one can implement, but adding too many can distort the images too much and make it harder for the model to learn. This can be the reason that horizontal flip gives better accuracy alone, compared to all of them implemented at the same time. One also has to consider which transformation is logical to implement.

Regularization by Dropout

By applying dropout we increase the test accuracy by 6.0% and achieved 76.4% accuracy. We tried to use both p=0.2, 20%, for each dropout layer, and an increase of 10% for each dropout, therefore a harder dropout. A consistent dropout percentage of 20% gave the best results. Dropout regularization the network, and makes it less prone to overfitting. It does this by reducing its parameters by randomly drop out nodes along with their connection. Thereby the remaining nodes must adapt, and this is the reason why we see an increase in accuracy. The reason a dropout of 20% gave better results can be because of the number of parameters in the network and how deep it is, in a network with more layers and therefore more parameters it would probably give better results with a harder dropout.

Batch normalization

We can see with batch normalization the network train faster, we achieve the same accuracy earlier in the training. Compared to the model in task 2, which achieved a test accuracy at 70.4% with 5 epochs, we achieved the same accuracy after 2 epochs with normalization and ended up with a final test accuracy at 74.3%. This is an improvement by 3.9%. This is because normalization normalizes the activation of the previous layer at each batch, maintaining the activation mean close to 0 and standard deviation close to 1. Therefore addresses the problem of internal covariance shift, and with this, it stabilizes the learning and accelerates the learning process. It also acts as a regularizer.

Kernel size 3x3

Changing the kernel size to 3x3 and padding to 1 did not give a significant improvement. The test accuracy with kernel size 3x3 was 70.5%, therefore only a 1% increase. Using a larger kernel size (5x5) can result in losing detail in some smaller features, that a smaller kernel size would be able to detect better. On the other hand, a kernel with size 5x5 is expected to be able to detect bigger details that a 3x3 kernel might miss. The images in CIFAR10 have a low resolution and differ in the degree of detail. We believe that kernel size of 3x3 is better for classifying some of the images while 5x5 is better for the others, and in this case, it was more important with a number of layers than the kernel size.

LeakyRelu activation

Applying LeakyRelu with a=0.01 to the network achieved a test accuracy at 74.3%. The ReLU activation function can cause some weights not to be updated during backpropagation since not all of the neurons activate and therefore leaving the gradient to be zero. This can lead to a neuron not activate on any datapoint again, thereby "die". Using the LeakyReLU activation function we address this problem, by returning a very small value instead of zero, when x is negative, therefore result in better accuracy.

Applying a deeper network

Containing the same network architecture as task 2, but applying additional layers with the same structure. By adding one more layer we achieved a test accuracy of 73% and adding two layers resulted in a test accuracy of 75.1%.

We also tried to change the architecture structure to (conv-relu-conv-relu-pool)xN → (affine)xM → softmax, with the same amount of layers as the model in network 1 in task 3a. This gave an increase in test accuracy at 9.2%, resulting in a accuracy at 79.6%, and therefore the technique that we saw the largest amount of improvement.

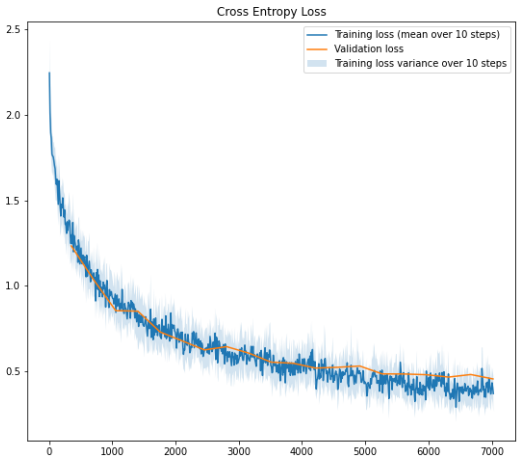
The advantage of deeper networks is that they can learn features at various levels of abstraction, whereas shallow networks are not able to learn as well. Training a network to classify more complex images require a more complex network that adds non-linearity to the model which a deeper CNN provides. This can clearly be seen in the increase of accuracy when applying more layers.

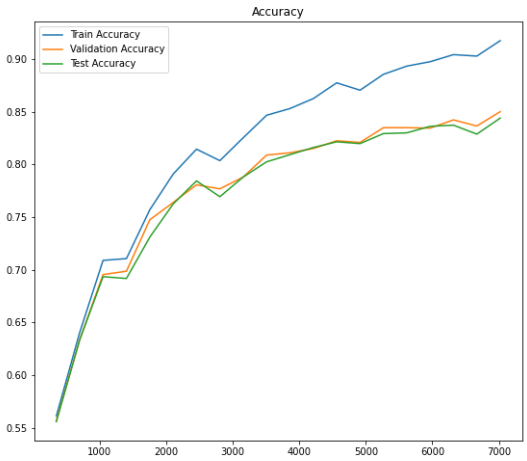
Task 3d)

Plot of validation loss with architecture from task 2 and with the architecture (conv-relu-conv-relu-pool)xN → (affine)xM → softmax, that gave the largest amount of improvement among the different techniques described in 3c).

Task 3e)

We decided to improve Network 1 even more by adding dropout with p=0.2 after max pooling, remove the crop and rotation transformation and replace ReLU with LeakyReLU. This is based on the results from task 3c). This resulted in even better test accuracy, and the final test accuracy is 0.844.





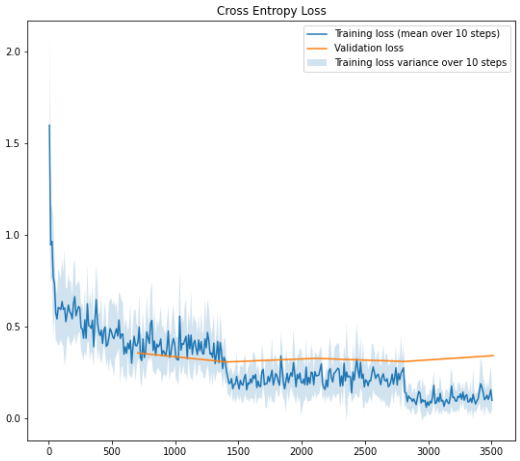
Task 3f)

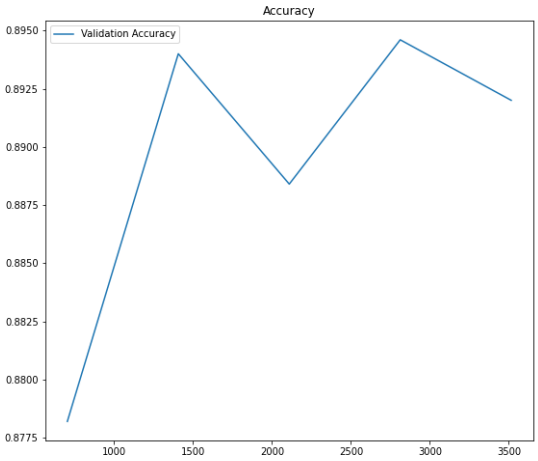
By observing the loss plot, the validation loss and training loss are quite similar, which suggest that the model is not overfitting. When analyzing the accuracy plot, we observe that the train accuracy is higher than the validation and test accuracy. Taking both plots into account it seems like the model is underfitting to some degree.

Task 4

Task 4a)

Hyperparameters	
Epochs	10
Batch Size	32
Learning Rate	0.0005
Early Stop Count	4

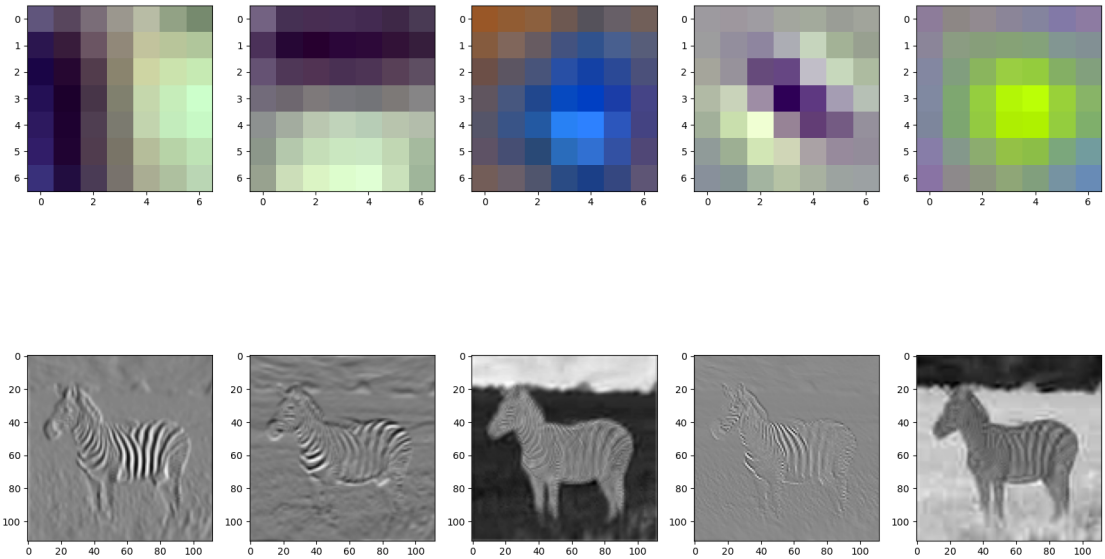




After 2 epochs early stopping criteria is met and we end up with a final test accuracy at 0.889.

Task 4b)

Filter activation one and two detect lines, the first activating on vertical lines and the second on horizontal lines. Plots 3 and 5 seem to be activated by larger features, *plot 3* on the sky and on the zebra, *plot 5* on the grass, both filter the image in a "gaussian way" - blurring out smaller details. Even though *Plot 4* is dominated by one shade of grey, the zebra is still easily identifiable. Thus, we believe *Plot 4* activates on contrasts or edges.



Task 4c)

To get a better impression of what the activation looks like after the last convolution layer, the zebra is set as a background and the activations are made transparent. Several of the activations are focused around the head and body area of the zebra. What might be of more interest, is the fact the we see very little activation in areas without the zebra (i.e. top right corner).