

Algorithms and Application in Computer Vision - 046746

## **Homework #2**

*Alexander Shender 328626114*

*Vladimir Tchuiev 309206795*

Technion - Israel Institute of Technology

## Contents

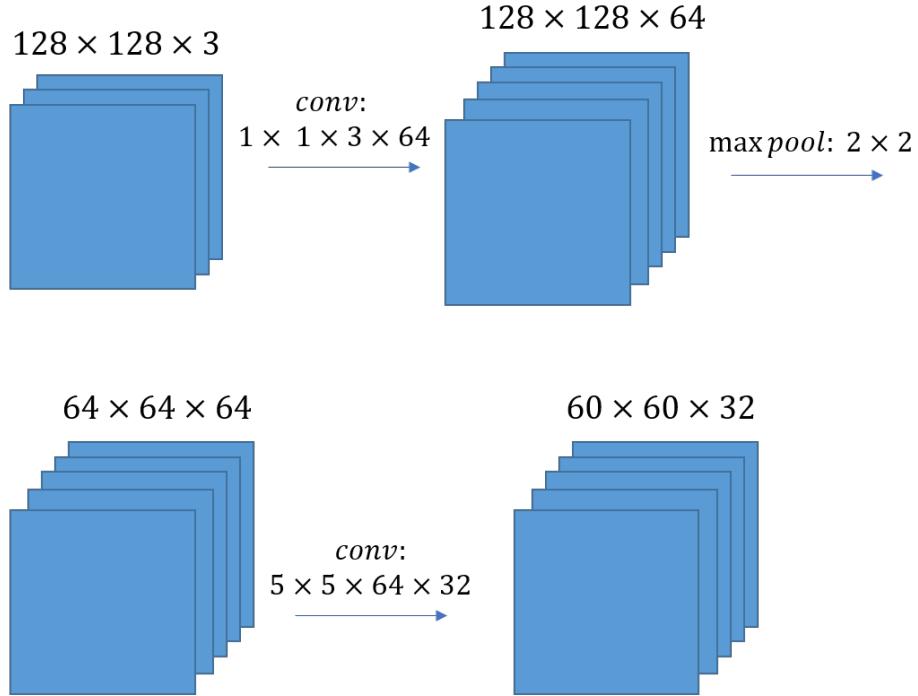
<b>I Dry section</b>	<b>3</b>
A Question 1.	3
1      a.	3
2      b.	3
3      c.	3
B Question 2.	6
C Question 3.	8
D Question 4.	9
E Question 5.	10
<b>II Wet section</b>	<b>12</b>
A Question 1.	12
1      a.	12
2      b.	15
3      c.	22
4      d.	26
B Question 2.	30
1	30
2	30
3	30
4	30
5	31
6	35
7	35
8	35
<b>III Image Stitching</b>	<b>42</b>

## I. Dry section

### A. Question 1.

1. a.

The dimensions of the layers change in the following way:



2. b.

The convolution of the size  $1 \times 1 \times (?)$  performs convolution on the same pixels in different channels. The input image contains 3 channels in our case, thus the convolution of the size  $1 \times 1 \times 3$  fits perfectly to result in a block of new layers without changing size (no need for padding). One kernel results in an output layer of size  $128 \times 128$ , but since we have 64 kernels, the depth of the next layers block is 64, accordingly.

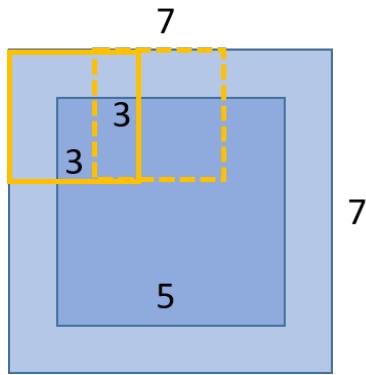
3. c.

Let's say, our normalized filter is the following:

$$\begin{bmatrix} 0.1 & 0.2 & 0.05 \\ 0.05 & 0.2 & 0.1 \\ 0.15 & 0.1 & 0.05 \end{bmatrix}$$

We choose 2 options for stride and padding:

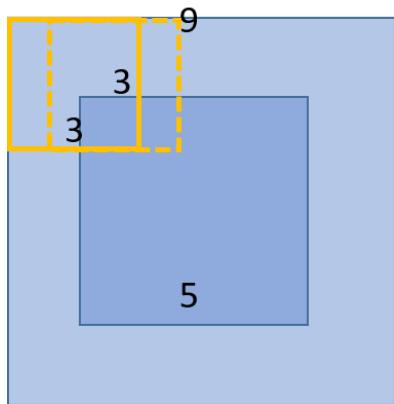
1.  $stride = 2, padding = 1$  The image now has a dimensions of  $9 \times 9$ , and with a stride of 1 it gives an output dimensions:  $3 \times 3$



Output result is the following:

$$\begin{bmatrix} 1.3 & 2.7 & 1.9 \\ 1.9 & 5.25 & 3.1 \\ 0.5 & 3.5 & 1.7 \end{bmatrix}$$

2.  $stride = 1, padding = 2$  The image now has a dimensions of  $7 \times 7$ , but with stride of 2 it fits with the filter. Output dimensions:  $7 \times 7$



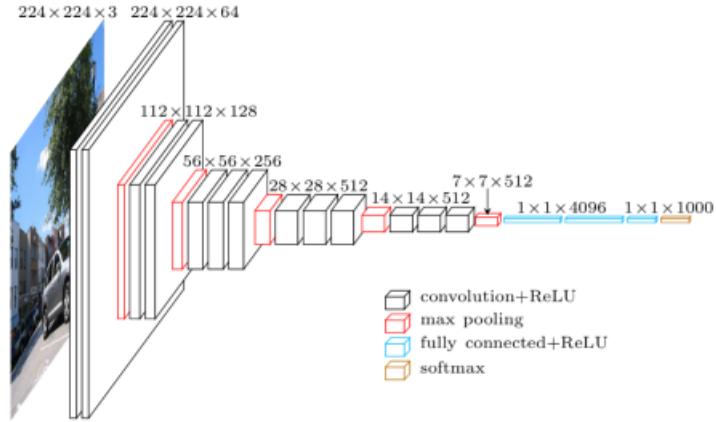
Output result is the following:

$$\begin{bmatrix} 0.2 & 0.45 & 1 & 0.8 & 1.15 & 0.45 & 0.45 \\ 0.55 & 1.3 & 2.0 & 2.7 & 2.65 & 1.9 & 0.45 \\ 0.6 & 2.1 & 3.3 & 5.15 & 4.6 & 2.6 & 1.45 \\ 0.4 & 1.9 & 3.75 & 5.25 & 5.145 & 3.1 & 1 \\ 0.2 & 1.3 & 3.5 & 4.7 & 4.0 & 3.35 & 1.45 \\ 0.05 & 0.5 & 2.05 & 3.5 & 2.55 & 1.7 & 0.5 \\ 0 & 0.05 & 0.5 & 1.5 & 1.6 & 1.2 & 0.4 \end{bmatrix}$$

. Those results were obtained using Python. The code is placed here: 'python\_code/hw2\_q1.py'.

## B. Question 2.

The architecture selected is the VGG16 architecture. The image found in the internet which describes the structure is the following:



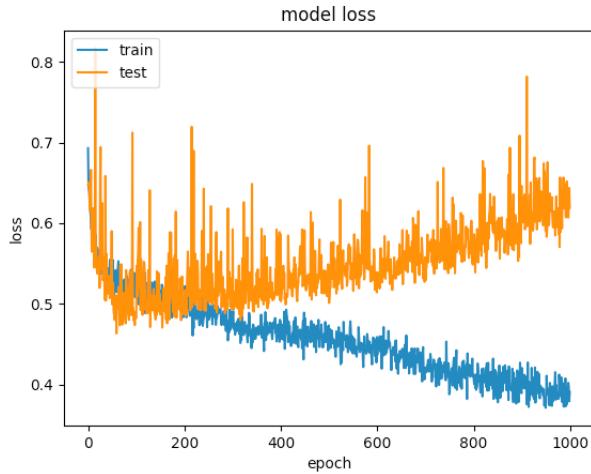
Writing the exact outputs for every layer:

Operation	Size	Padding	Stride	Output size
Conv block	[3X3X3]X64	[1 1]	1	[224X224X64]
Conv block	[3X3X64]X64	[1 1]	1	[224X224X64]
Pool 2D	[2 2]	N/A	N/A	[112X112X64]
Conv block	[3X3X64]X128	[1 1]	1	[112X112X128]
Conv block	[3X3X128]X128	[1 1]	1	[112X112X128]
Pool 2D	[2 2]	N/A	N/A	56X56X128
Conv block	[3X3X128]X256	[1 1]	1	[56X56X256]
Conv block	[3X3X256]X256	[1 1]	1	[56X56X256]
Conv block	[3X3X256]X256	[1 1]	1	[56X56X256]
Pool 2D	[2 2]	N/A	N/A	[28X28X256]
Conv block	[3X3X256]X512	[2 2]	1	[28X28X512]
Conv block	[3X3X512]X512	[2 2]	1	[28X28X512]
Conv block	[3X3X512]X512	[2 2]	1	[28X28X512]
Pool 2D	[2 2]	N/A	N/A	[14X14X512]
Conv block	[3X3X512]X512	[1 1]	1	[14X14X512]
Conv block	[3X3X512]X512	[1 1]	1	[14X14X512]
Conv block	[3X3X512]X512	[1 1]	1	[14X14X512]
Conv block	[3X3X512]X512	[1 1]	1	[14X14X512]
Pool 2D	[2 2]	N/A	N/A	[7X7X512]
Fully connected	[7·7·512]X4096	N/A	N/A	[1X1X4096]
Fully connected	[4096X4096]	N/A	N/A	[1X1X4096]
Fully connected	[4096X1000]	N/A	N/A	[1X1X1000]

### C. Question 3.

**Definition:** Overfitting - is a situation, where network is fitted too much to the training data, and finds it difficult to generalize to create predictions for the new data.

**How to spot:** First of all, a researcher will notice that the accuracy of the model on the Test Dataset decreases, while the accuracy on the Training Set will still be increasing. The typical graph visualizing error on the Test & Training set may be seen, demonstrating this exact situation:



**How to avoid:** There are numerous ways to avoid overfitting:

1. Stop the training before the accuracy for the validation set starts increasing. If the accuracy does not satisfy, find a better dataset / improve the network / apply other changes. Training for more time will worsen the situation.
2. Use one of the following methods: regularisation, lambda factor, dropout, etc.
3. Increase the dataset size. Feed the network new examples for learning.

#### D. Question 4.

The learned parameters are being updated using the backpropagation algorithm. The name derives from the way that the Error value propagates backward through the network, affecting the parameters according to the contribution that those gave to the error value. In the general view, this may be seen in update function for the parameters (in this case - weights):

$$W := W - \alpha \cdot \frac{\delta E}{\delta W}$$

where:

$W$  – lweight parameter

$\alpha$  – learning rate

$\frac{\delta E}{\delta W}$  – ”contribution” of the parameter to the loss

## E. Question 5.

**Definition:** Batch normalization - is a method which is used to normalize the layer inputs, in order to solve the problem called *internal covariate shift*.

*internal covariate shift:* the problem which arises in the intermediate layers during training because the distribution of the activations is constantly changing during training. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities.<sup>1</sup> <sup>2</sup>

So, actually we force the input of a specific layer to have approximately the same distribution in every training step. The batch normalization is performed in 4 steps (image taken from the original article):

<b>Input:</b> Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$ ;
Parameters to be learned: $\gamma, \beta$
<b>Output:</b> $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$
$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$ // mini-batch mean
$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$ // mini-batch variance
$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$ // normalize
$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i)$ // scale and shift

**Algorithm 1:** Batch Normalizing Transform, applied to activation  $x$  over a mini-batch.

Steps are the following:

1. Calculate the batch mean of the values  $x$  of a particular layer (that we do the normalization on)  $\mu_{\beta}$
2. Similarly, calculate the variance of those values  $x$  for this layer  $\sigma_{\beta}^2$
3. Normalize the values, subtracting the mean  $\mu_{\beta}$  and dividing by STD (+constant)  $\sqrt{\sigma_{\beta}^2 + \epsilon}$ . This will result in a new Gaussian distribution with mean of 0 and Variance of 1.
4. Scale and Shift by learnable parameters  $\gamma$  and  $\beta$ . Those parameters are being learned, and are inserted to make it possible to the distribution to be scaled and

---

<sup>1</sup>Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift , Sergey Ioffe, Christian Szegedy

<sup>2</sup>Towards Data Science: Batch normalization: theory and how to use it with Tensorflow

shifted if such is needed. For example, if it is of our interest to make the batch normalization an identity transform.

## II. Wet section

### A. Question 1.

The subsection numbers here do not match the numbers as in the questions booklet, but the order is correct. In order to run the code for this question, run the .py file : 'python\_code/mnist\_pytorch.py'. The script will automatically create the data folder (where the MNIST dataset will be downloaded), and the results folder (according to the current time in :day:hour:minute: format), where all the graphs which are presented here will be placed.

#### 1. a.

**This subsection answers the steps 1-6.**

Network is the following:

- Network as defined in step 2
- Loss function = CE (Cross Entropy)
- Optimizer: SDGM (momentum = 0.5)
- Learning rate = 0.01
- Epochs = 8
- Training minibatch size = 128
- Training/validation split = 50000/10000. (MNIST dataset includes 60000 examples)

Answering the step 5 question:

1. Loss value graphs:

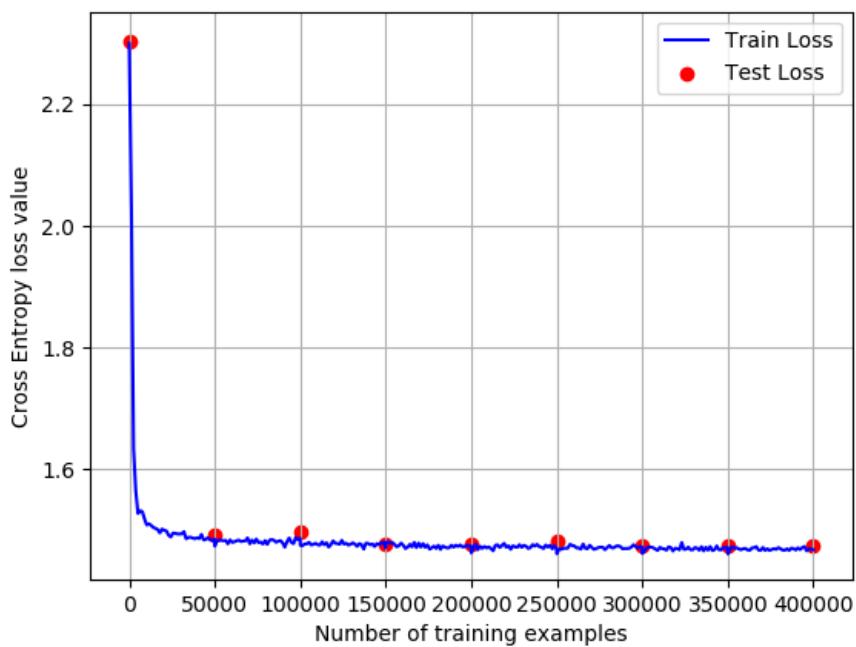


Figure 1: Loss value as a function of the training examples seen

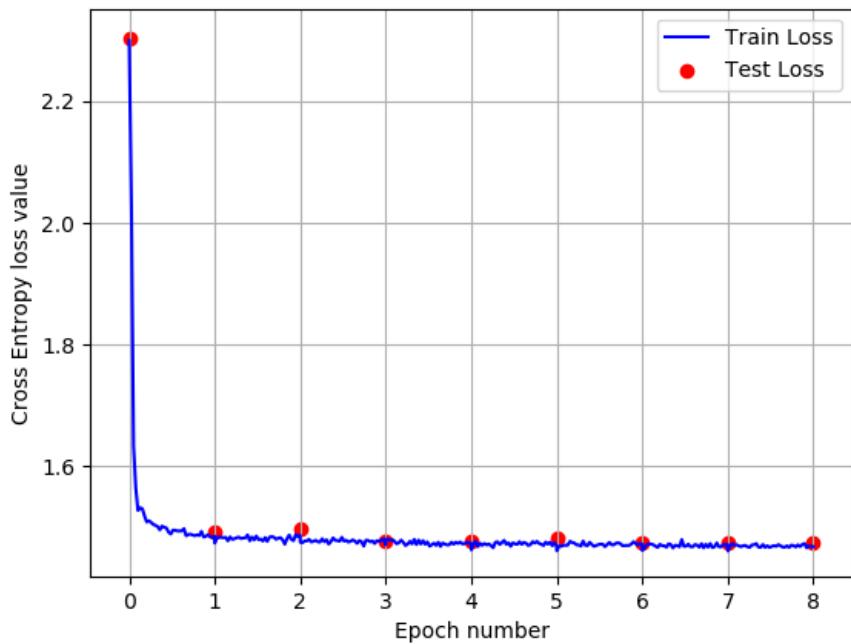


Figure 2: Loss value as a function of the Epoch number

## 2. Classification accuracy graphs

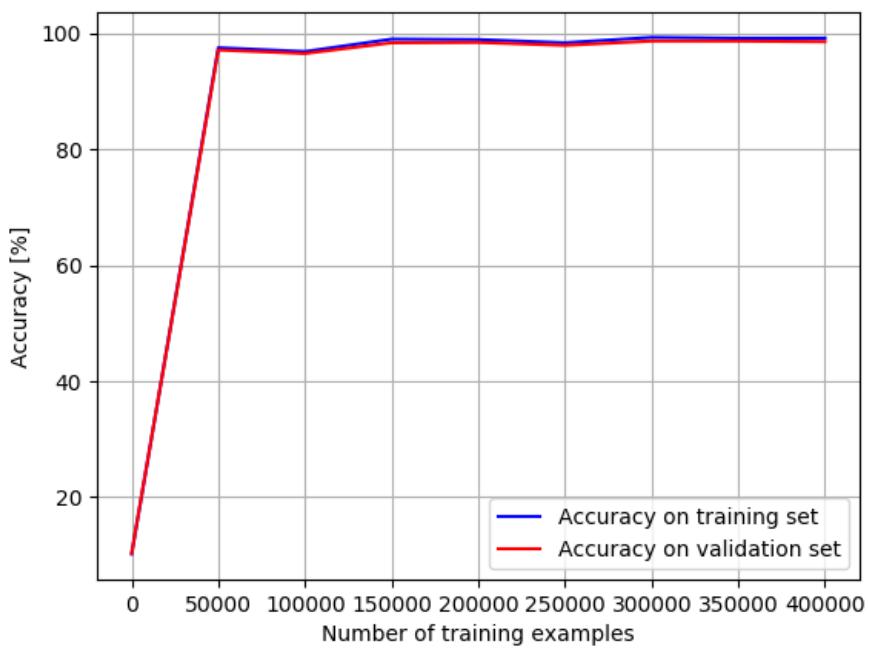


Figure 3: Classification accuracy as a function of the training examples seen

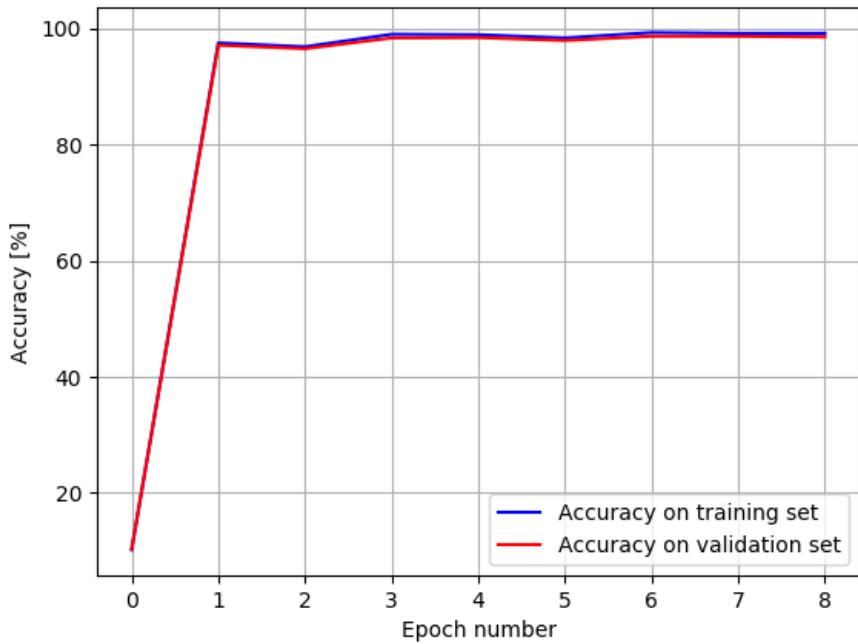


Figure 4: Classification accuracy as a function of the Epoch number

Answering the step 6 question: The classification accuracy reported is the following:

1. Training set: 99.186%
2. Test set: 98.58 %

2. b.

This subsection answers the step 7 - part 1.

First, we repeat the process with the different learning rate values, whereas all the other parameters defined in the previous subsection remain the same. And the results we get are the following:

1. For the same network with the learning rate of **0.1**:

- (a) Loss value graphs:

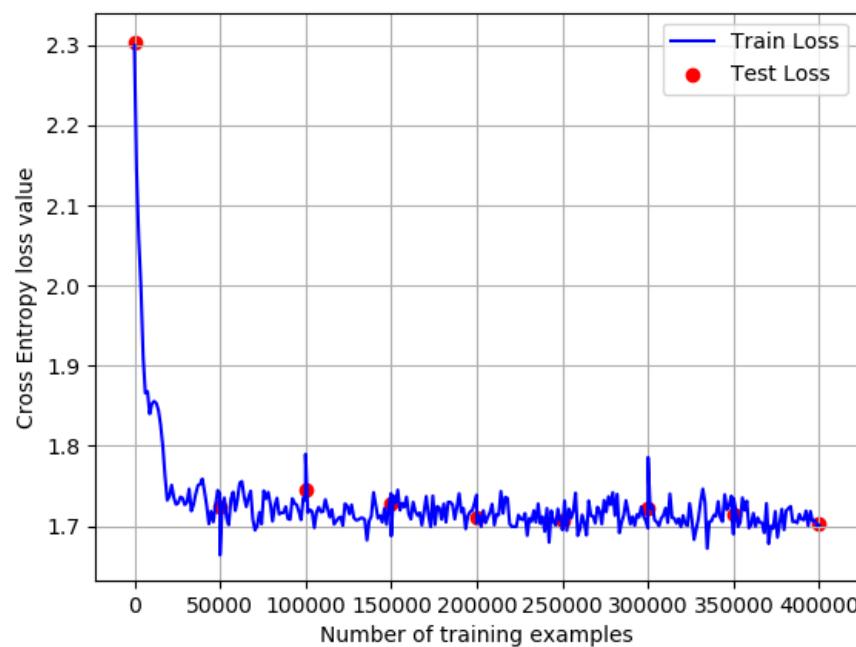


Figure 5: Loss vs. Training examples

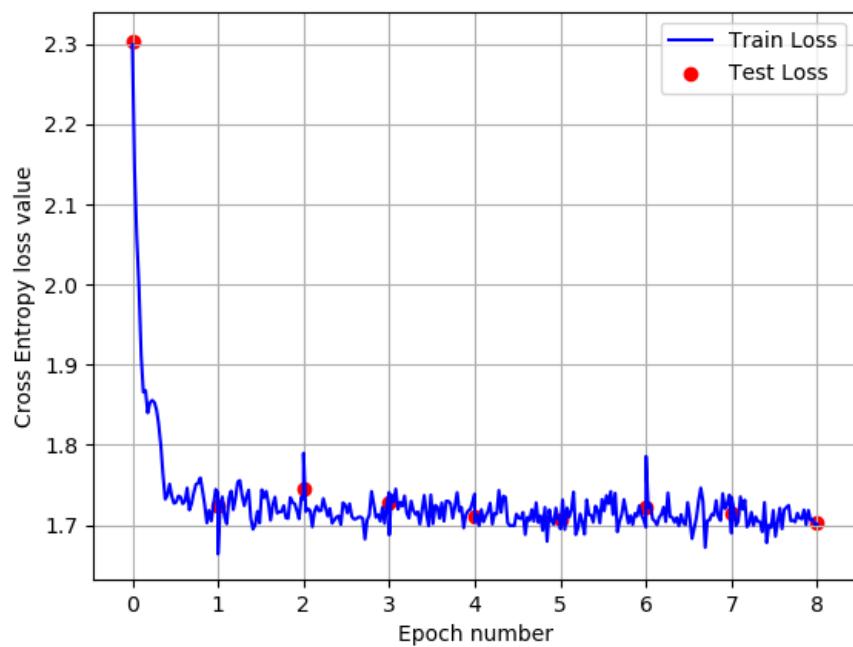


Figure 6: Loss vs. Epoch number

(b) Classification accuracy graphs

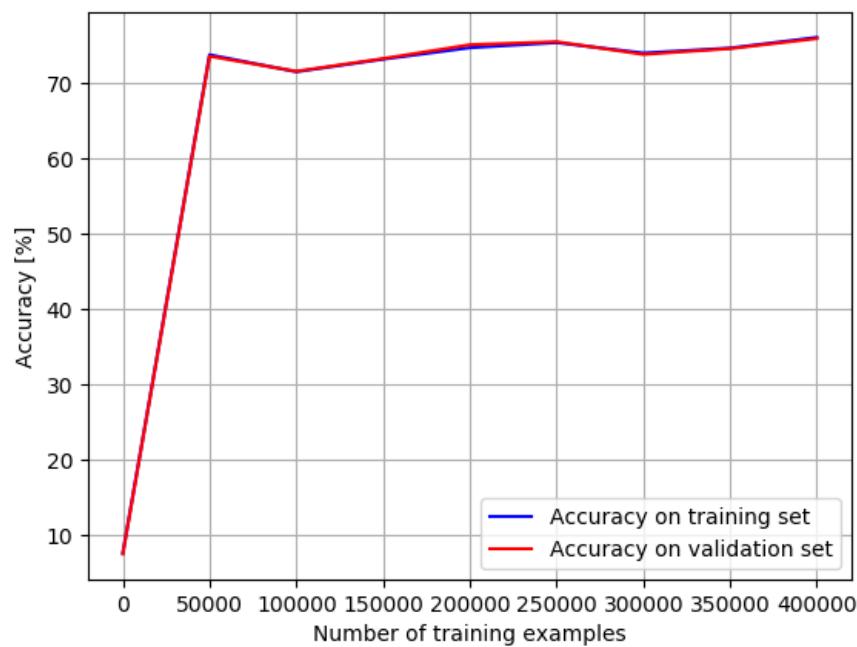


Figure 7: Accuracy vs. Training examples

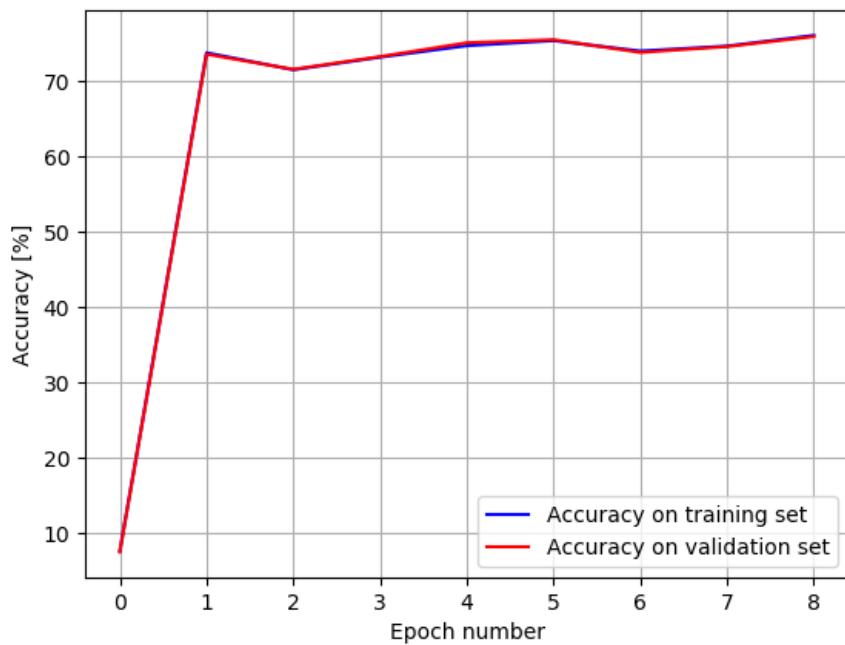


Figure 8: Accuracy vs. Epoch number

2. For the same network with the learning rate of **0.0001**:

(a) Loss value graphs:

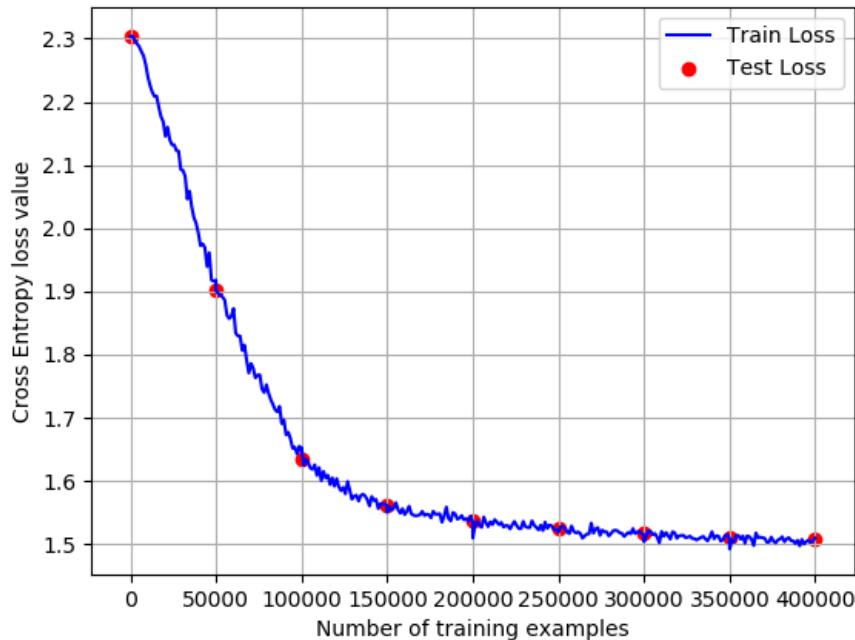


Figure 9: Loss vs. Training examples

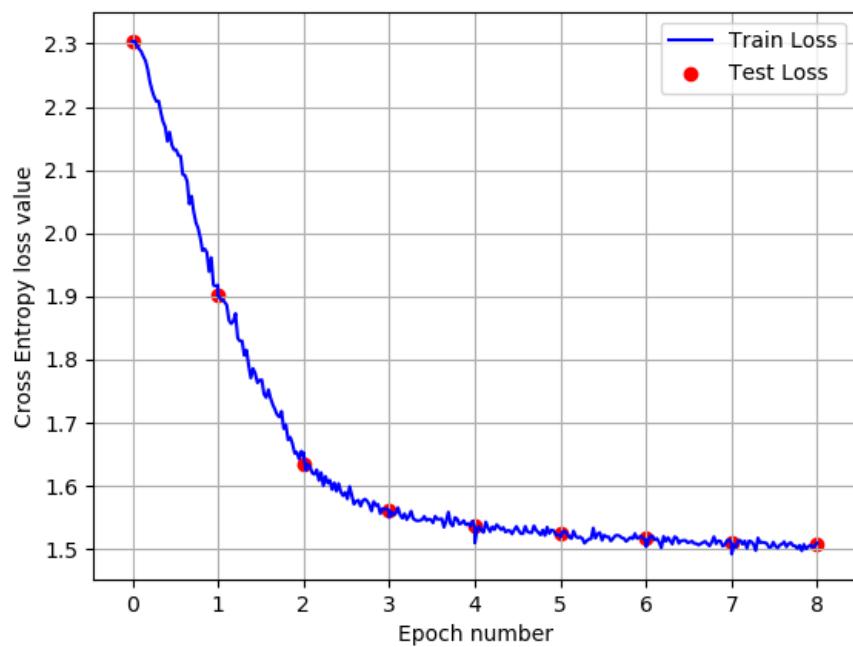


Figure 10: Loss vs. Epoch number

(b) Classification accuracy graphs

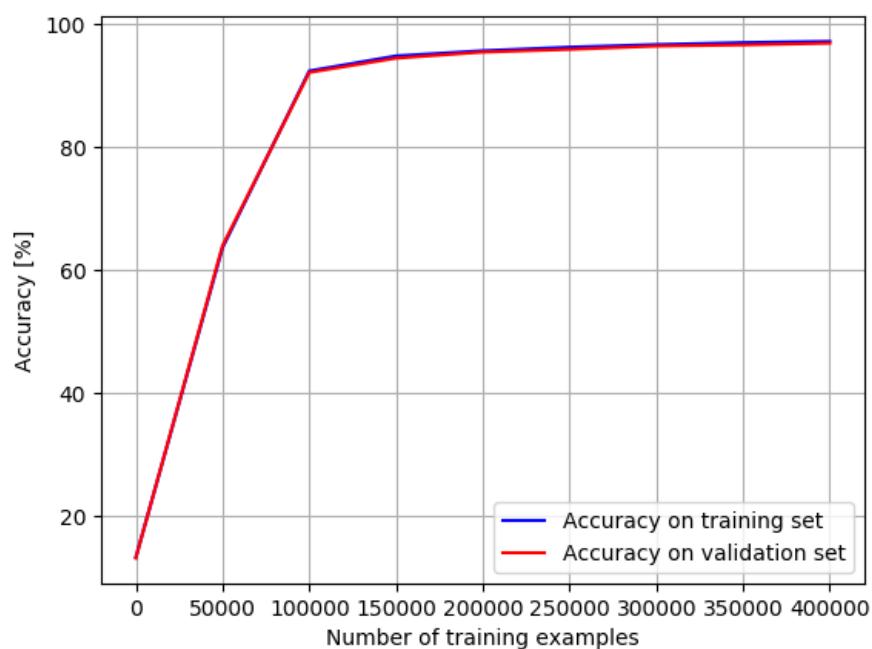


Figure 11: Accuracy vs. Training examples

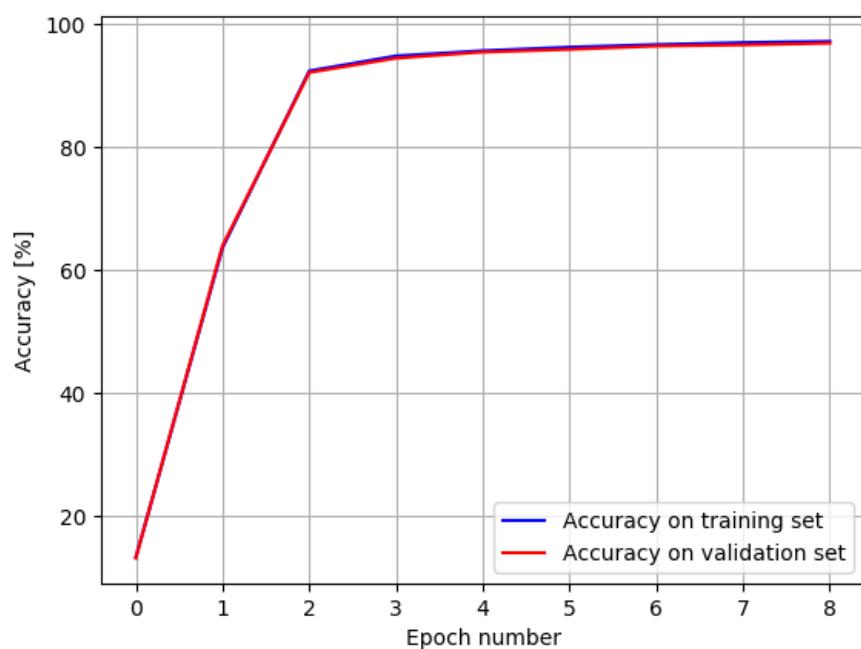


Figure 12: Accuracy vs. Epoch number

Answering the question asked, we compare the results and then draw conclusions. First, We compare the final accuracy on the validation set for 3 of the networks:

Network LR	Final test accuracy [%]	Final training accuracy [%]
0.01	98.58	99.186
0.1	75.88	76.036
0.0001	96.94	97.23

Then, we compare the accuracy of those 3 networks vs. the learning rate:

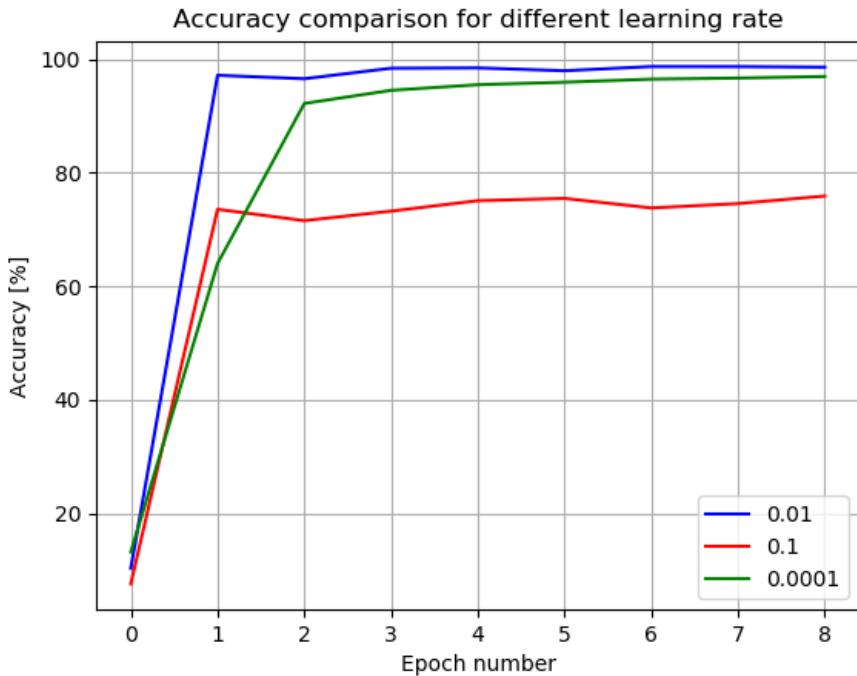


Figure 13: Accuracy vs. Epoch number

Several conclusions can be drawn from the overall results:

1. As we can see, the learning rate is a parameter which has to be tuned in order to obtain optimal results:
  - (a) Small learning rate leads to a network which will take a lot of time to converge, and this convergence will slow down even more when the Loss will decrease. We can see it in the case with  $\alpha = 0.0001$
  - (b) Large learning rate may give a quick jump towards the optimal solution in the beginning, but the network may never converge, since the parameters will be changed very drastically, and the Loss will not decrease. We can see it in the case with  $\alpha = 0.1$  (which is high learning rate)

There have been made a lot of research on this matter, and often the adaptive learning rate is being used. (which changes its value over time). From our example,

over the given training time, the case with  $\alpha = 0.01$  showed the best results

2. As mentioned above, the small learning rate makes small changes to the network parameter, so that the Loss value changes in a smooth way (Figure 10). Opposite is true for the high learning rate (Figure 6)
3. As mentioned above, high learning rate will reach a point where it can no longer decrease the Loss value, since the changes made for the parameters are very big. This is visible good in the Figure 6.
4. In all the cases, we can observe that the accuracy on the validation set is slightly smaller than the one on the training set. Which is logical, the network copes better with the data it has already seen and was trained on.

### 3. c.

This subsection answers the step 7 - part 2.

First, we repeat the process with the same learning rate ( $\alpha = 0.01$ ), but with a different Loss Function, using the L2 Norm function as the loss value. Reporting the same results, we obtain the following Loss value and the following Accuracy graphs:

1. Loss value graphs:

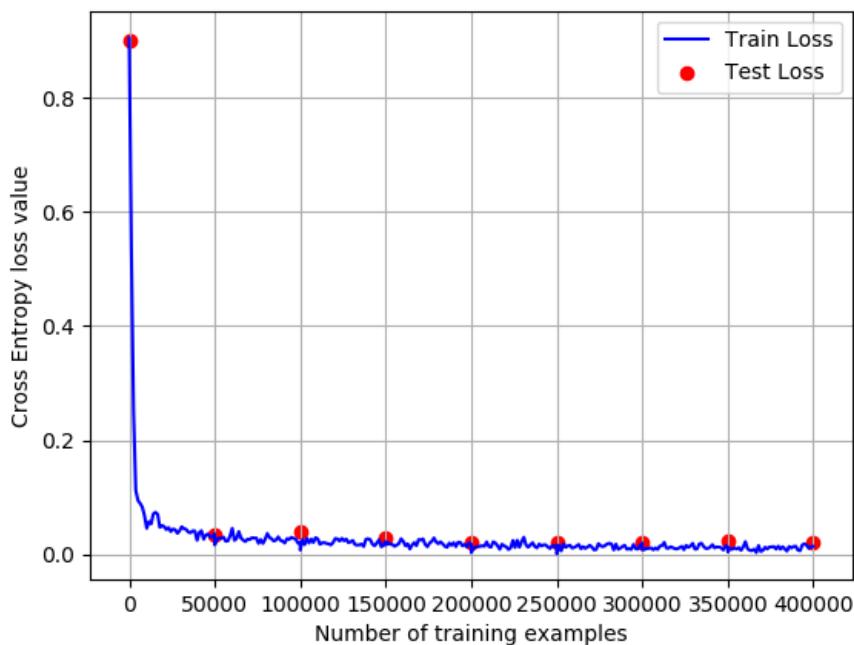


Figure 14: Loss vs. Training examples

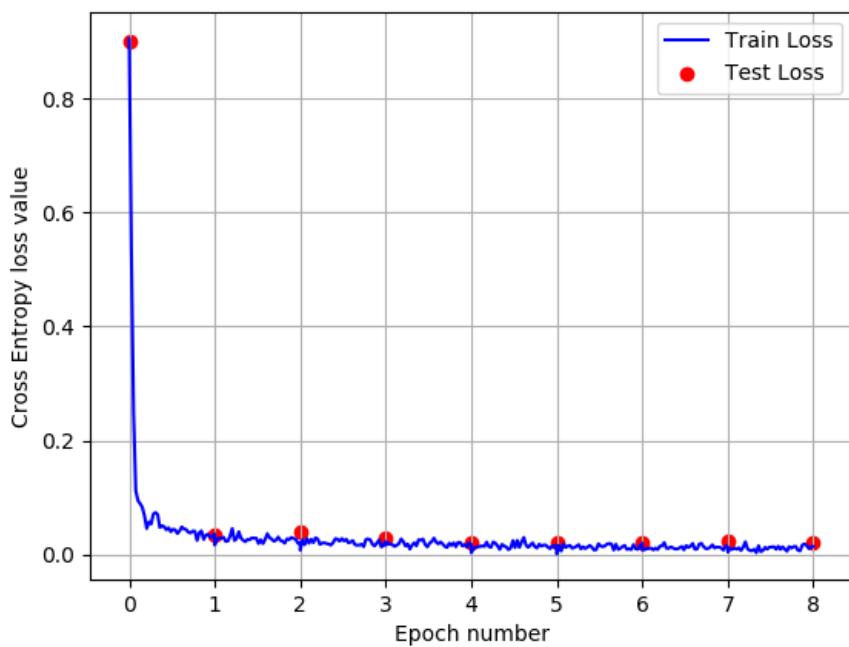


Figure 15: Loss vs. Epoch number

## 2. Classification accuracy graphs

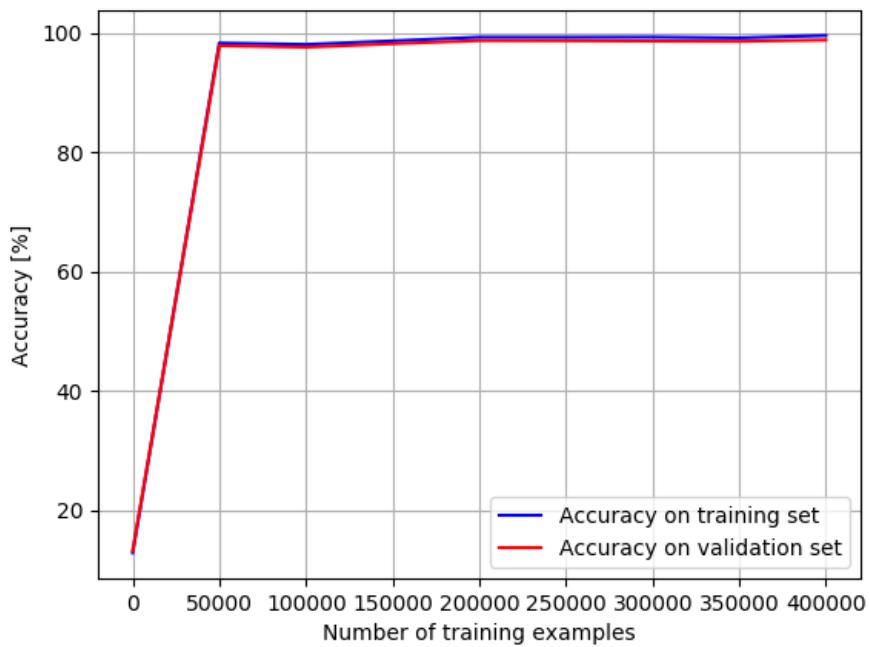


Figure 16: Accuracy vs. Training examples

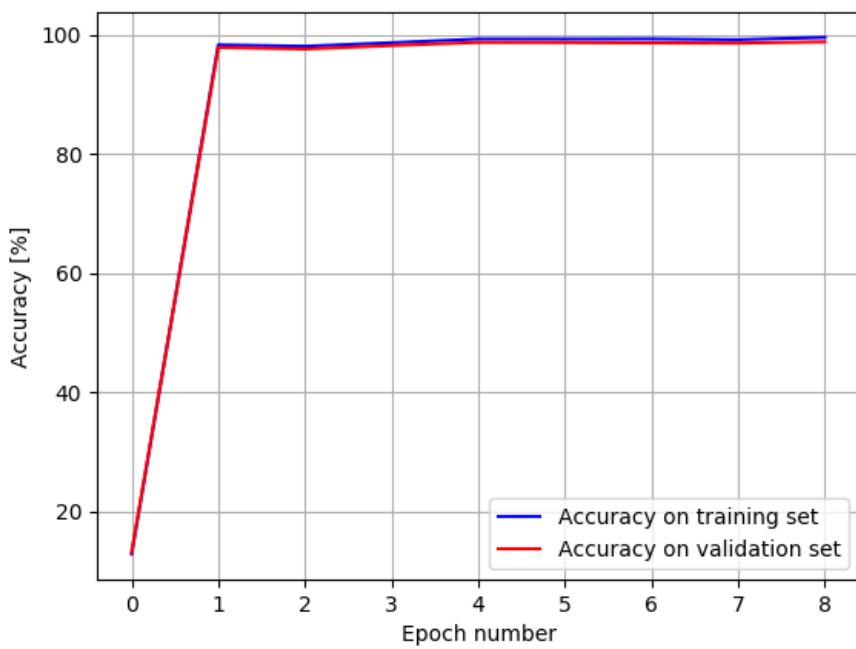


Figure 17: Accuracy vs. Epoch number

Comparing the accuracy rate on the trained network for both cases gives:

Network Loss function	Final test accuracy [%]	Final training accuracy [%]
Cross Entropy	98.58	99.186
N2 norm	98.77	99.514

Then, we compare the accuracy of those 2 networks:

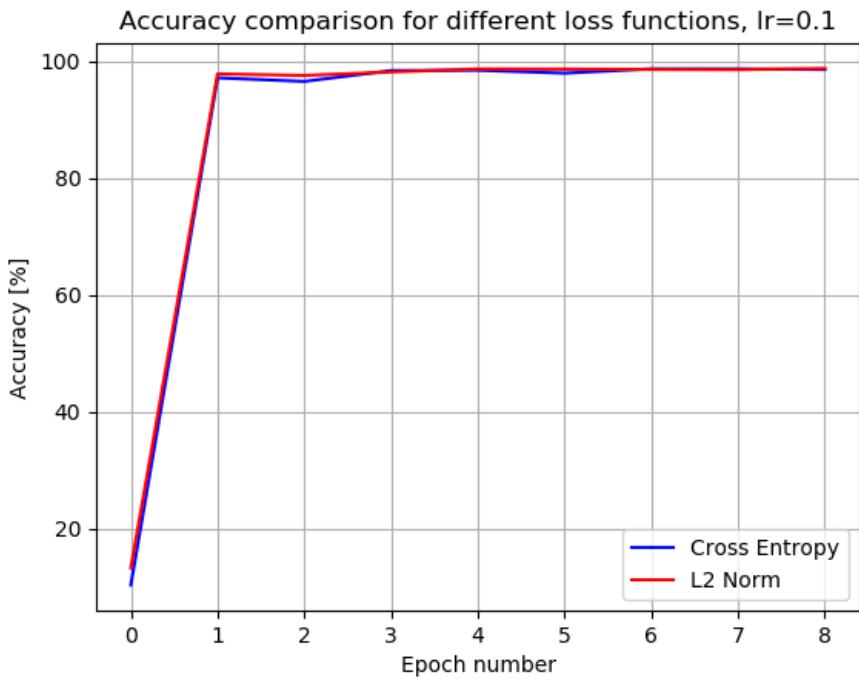


Figure 18: Accuracy vs. Epoch number

Several conclusions can be drawn from the overall results:

1. As we can observe, there is no much difference in the performance of both networks, and both classify the validation examples approximately on the same level. Although this is the case in our example, choosing the correct Loss Function is vital for a successful Network training in other, more complex environments.

4. d.

**This subsection answers the step 8.**

First, we repeat the process with all the same parameters, but the network structure is different. The following results are being obtained (for the new network):

1. Loss value graphs:

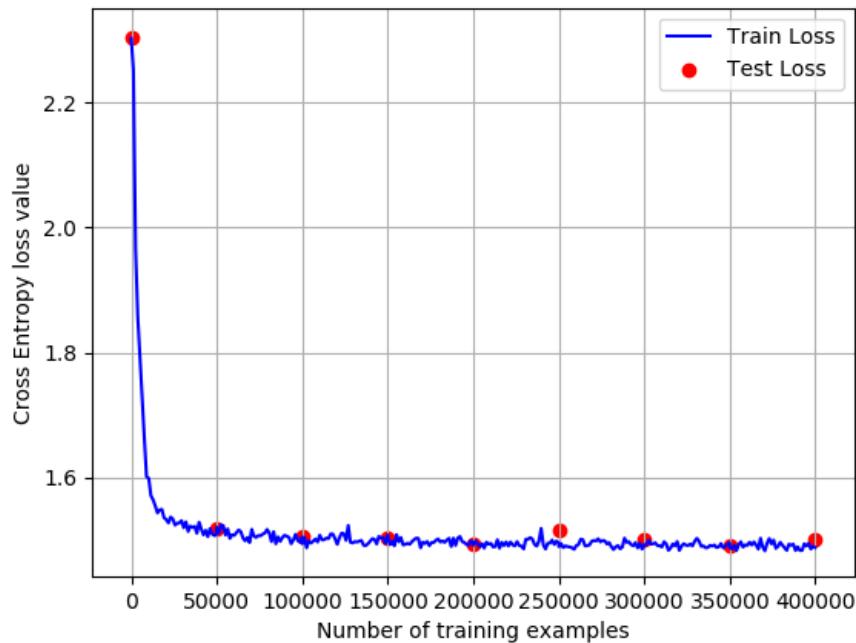


Figure 19: Loss vs. Training examples

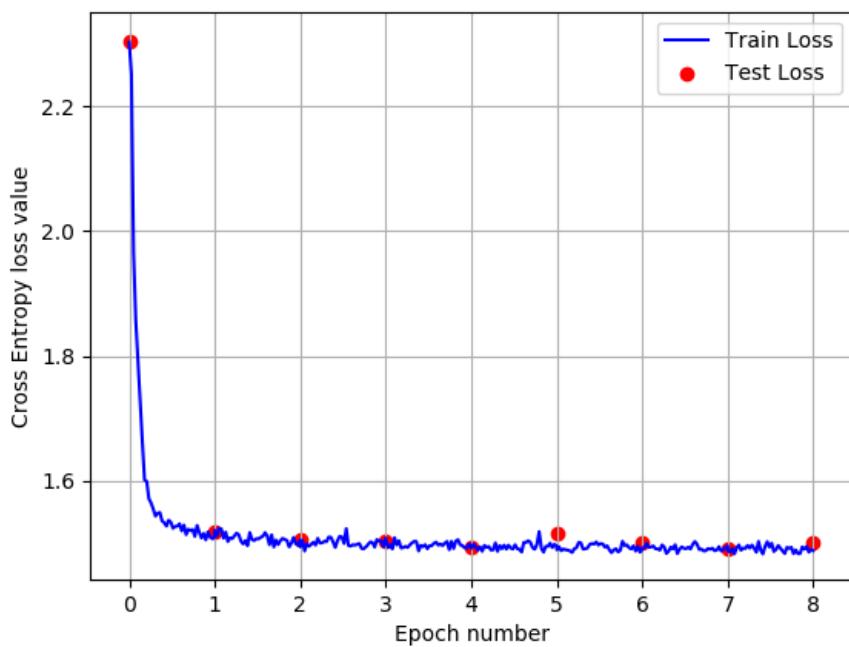


Figure 20: Loss vs. Epoch number

## 2. Classification accuracy graphs

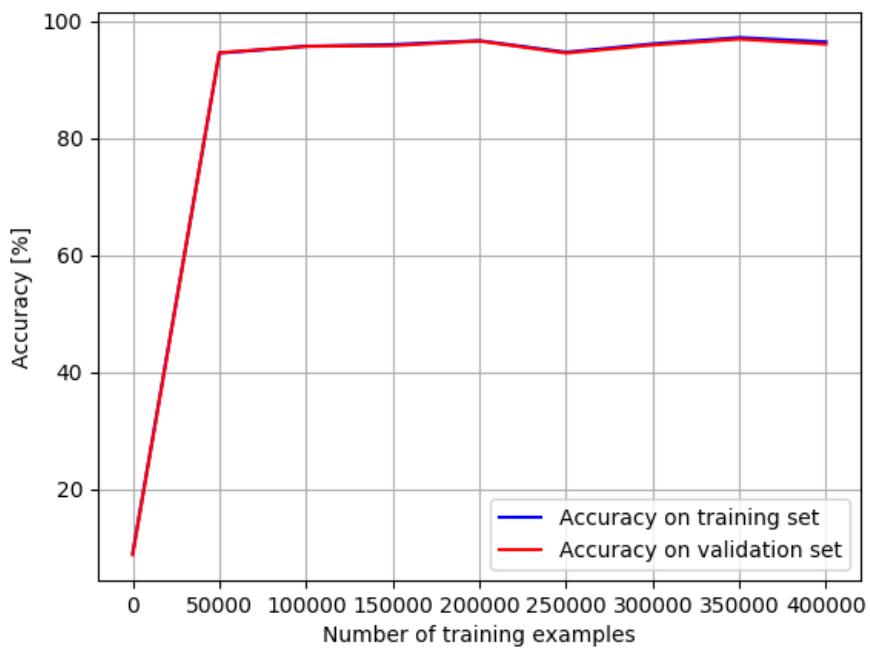


Figure 21: Accuracy vs. Training examples

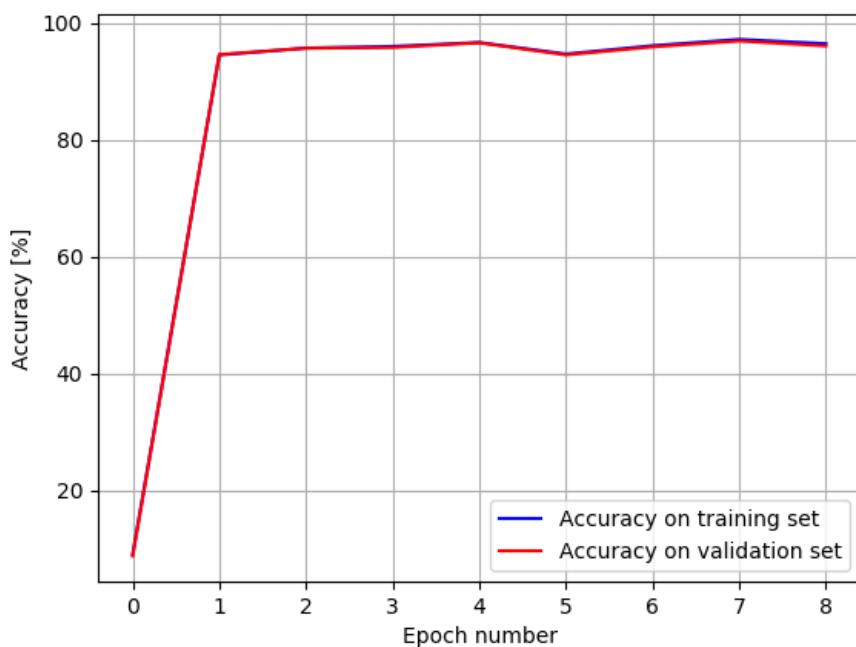


Figure 22: Accuracy vs. Epoch number

Comparing the accuracy rate on the trained network for both cases gives:

Network No.	Final test accuracy [%]	Final training accuracy [%]
1	98.76	99.25
2	96.13	96.52

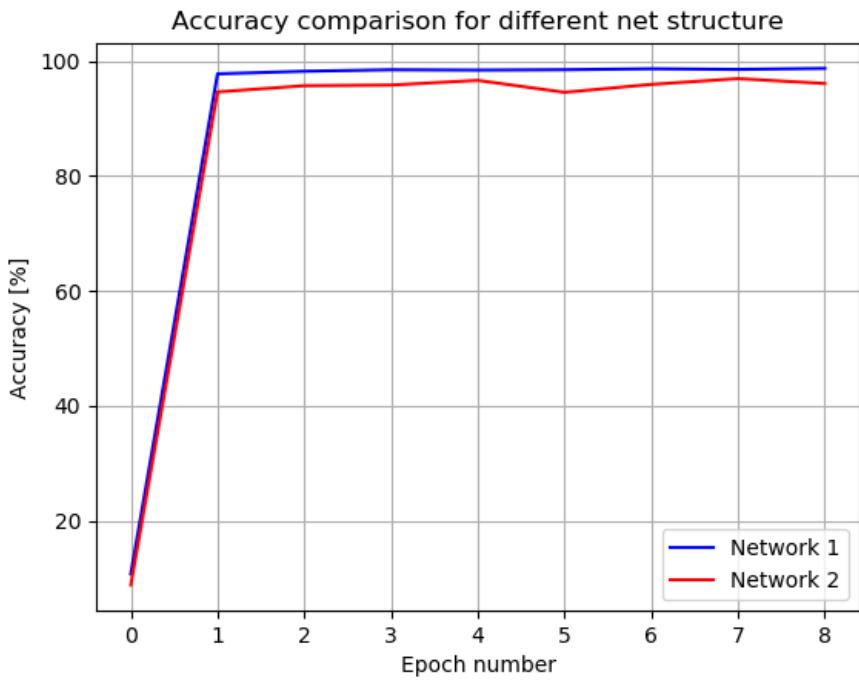


Figure 23: Accuracy vs. Epoch number

Several conclusions can be drawn from the overall results:

1. As we can observe, the new network structure contains **less** parameters, which means, its performance abilities are smaller than from the network with more parameters. This is also one of the big considerations when creating the network architecture - finding the optimal amount of layers & parameters / convolution size / etc. in each layer. From one side, the network should have enough parameters to supply the desired accuracy, from another side, an abundance of the parameters will lead to longer training time, and some other known problems (vanishing/exploding gradient, etc. )

## B. Question 2.

1.

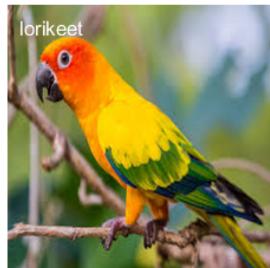
We have used the MATLAB version of the network, out of its simplicity to comply with the required tasks in this question. To run the code for this question, please run the **main.m** file inside the "matlab\_code" folder.

2.

The bird images were loaded. Since their size differed from the input size to the VGG16 network ([224X224X3]), we had to resize them in order to fit into the network. This was done thoughtout the exercise without saving the Aspect Ratio of the images. The predictions for the birds are the following:



(a) Bird #0 prediction



(b) Bird #1 prediction

Figure 24: Prediction on the input birds

As we can observe, the predictions are correct.

3.

The random image on the web that we have found is the image of the bear. The image was resized as well and has produced the following classification output, which is correct. (it says "Brown bear") (See Figure 25 )

4.

The following transformations have been applied:

1. **Geometric transformation:** we have used the 'shear', which is represented by the following affine transformation matrix:



(a) Bear input image

(b) Bear resized and classified

Figure 25: Bear prediction

shear matrix used:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0.45 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. **Color transformation:** we have affected the Hue channel, multiplying its values by 2.5
3. **Filter:** we used the Motion Blur filter defined in MATLAB, of size 20, with angle of 45 degrees

The results are presented in the next image. As we can see, all the classifications are still correct, which signals that the network is robust enough to all those modifications. (See Figure 26 ) We can also observe, than in comparison to not modified bear image, the prediction certainty was slightly decreased, where the most reduction was for the blurred image of the bear

5.

The filters in the first convolution layer are of the size [3X3X3], which means that the single filter on the input image [224X224X3] produces 1 output image [224X224X1], which is easy to vizualize.

The filters chosen randomly are 5 and 32. To vizualize the filters, their values were normalized (to fit into [0 1] range), and then viewed. The 3 depth layers are being presented as 3 2D filters. (but each filter is [3X3X3]). Filters are presented in Figure 27:

As we can observe, Filter 1 is trained to detect edges. Filter 2 detects places which have more light, where the texture is more visible.



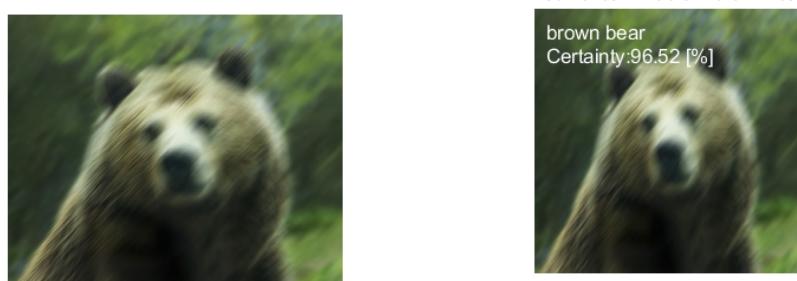
(a) Bear sheared input image

(b) Bear sheared classified



(c) Modified color bear

(d) Modified color bear classified



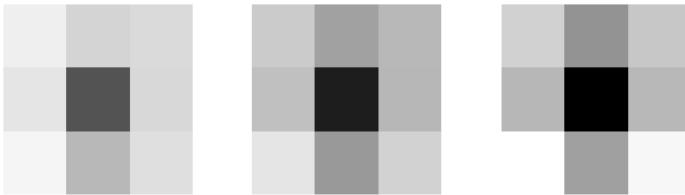
(e) Blurred bear

(f) Blurred bear classified

Figure 26: Bear modifications and its prediction



(a) Filter 1



(b) Filter 2

Figure 27: Filters 5 and 32

The responses of each of the input images are presented in the Figure 28, where the left column provides the responses for Filter 1, and the right - for Filter 2. So we can see the differences more clearly, the responses to the original bear image are also presented there

Some information on the image responses to the first layers in the CNN: "These images mostly contain edges and colors, which indicates that the filters at layer 'conv1' are edge detectors and color filters. The edge detectors are at different angles, which allows the network to construct more complex features in the later layers."<sup>3</sup>

By looking at the differences we can say the following:

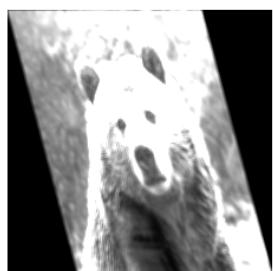
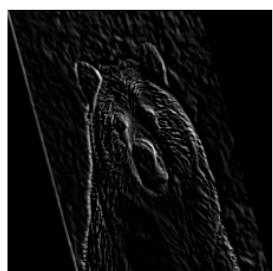
1. We can see that the Filter 1 is the edge detector, while the Filter 2 also responses to the Color.
2. Blurring of the image makes it harder to detect edges, and they become much more vague
3. The Hue change affects the second filter output a bit
4. The shear has affected the Filter 1 response - the edges detected are now at different angle. Some are detected more, some are less

---

<sup>3</sup><https://www.mathworks.com/help/deeplearning/examples/visualize-features-of-a-convolutional-neural-network.html>



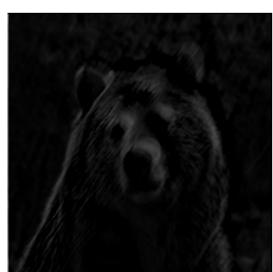
(a) Bear original response to Filter 1 (b) Bear original response to Filter 2



(c) Bear sheared response to Filter 1 (d) Bear sheared response to Filter 2



(e) Bear hue transformed response to (f) Bear hue transformed response to  
Filter 1 Filter 2



(g) Bear blurred response to Filter 1 (h) Bear blurred response to Filter 2

Figure 28: Transformed bear image responses to filters <sup>34</sup>

6.

Every image of cats and dogs was passed through the network, the Feature vectors of size 4096 were obtained for each image. Using the PCA method, the Principal Components were obtained and the Principal Component Scores for each of the samples (dogs, cats). The graphs below show the clusters of cats and dogs. It may be seen that there is a certain difference between the two. It was observed that the 3rd PC also was large, in relation to PC1 and PC2. Thus, a 3D scatter plot is also presented: (Figure 29)

7.

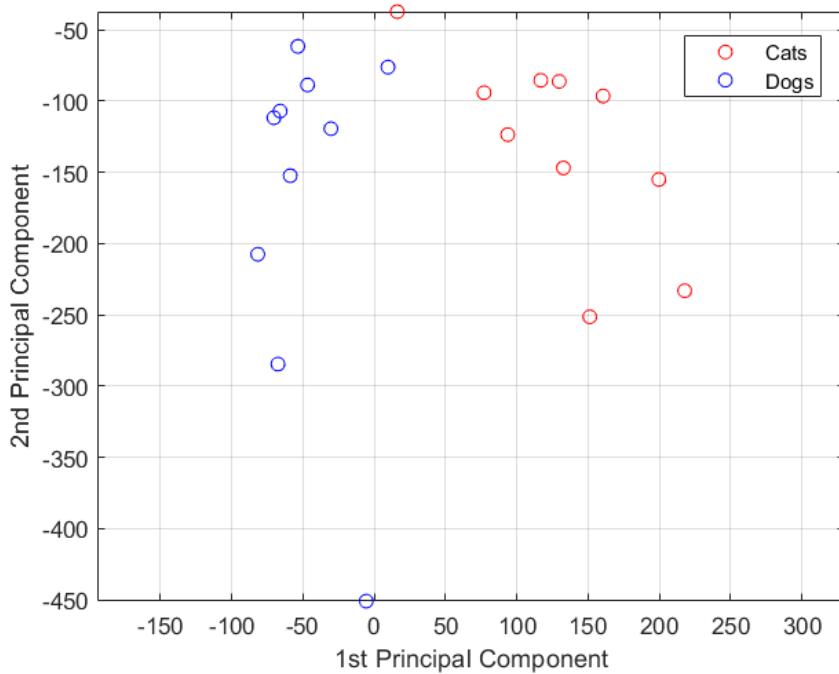
The images of a cat and a dog from the internet are preset in the Figure 30. The coefficient of the PCA transformation from previous question were used to obtain the PC scores for the new images. The new images are being displayed on the same PC scores graph (both 2D and 3D) and are presented in Figure 31. Then, the closest neighbor was found for both a cat and a dog through the Minimal Euclidean Distance (L2) (over the vector of all 20 features). The closest image of a cat with a new cat and the closest image of a dog are presented in 32. We can indeed notice that the pairs match and look similar.

8.

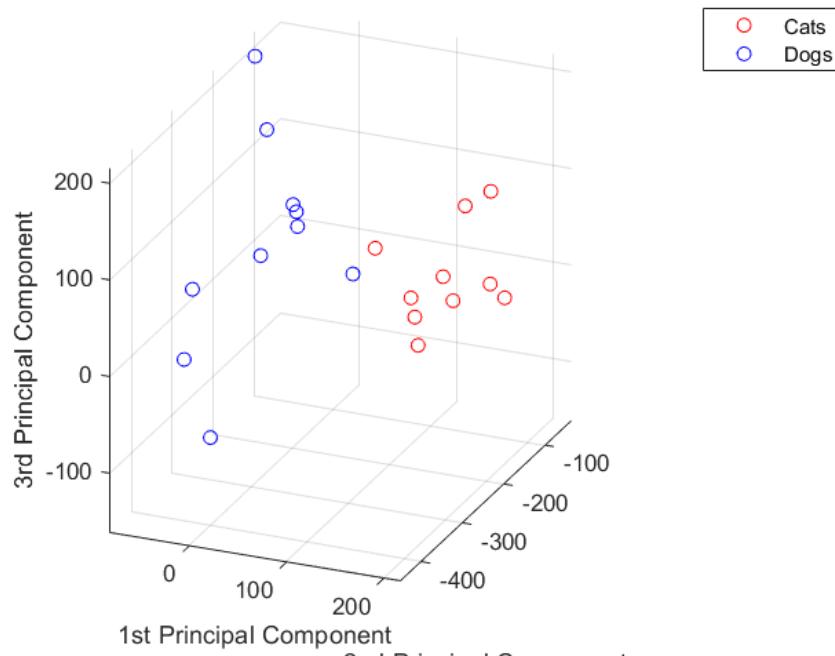
The same procedure is being done for the random tiger and wolf images from the internet. You can see the results in the figures:

1. PCA score graphs with new tiger and wolf images: Figure 33
2. The closest image to the tiger and to the wolf: Figure 34

We can see from the PC scores graph that the tiger and the wolf sit a bit further away from the cats and dogs clusters. The images of the closest cats and dogs do resemble the wolf and the tiger (the most, from all the other cats and dogs images). The cat has the similar texture, and the dog...has similar pose, face shape, ears, and the face texture.



(a) PCA with 2 PC



(b) PCA with 3 PC

Figure 29: PCA Scores graphs for cats and dogs

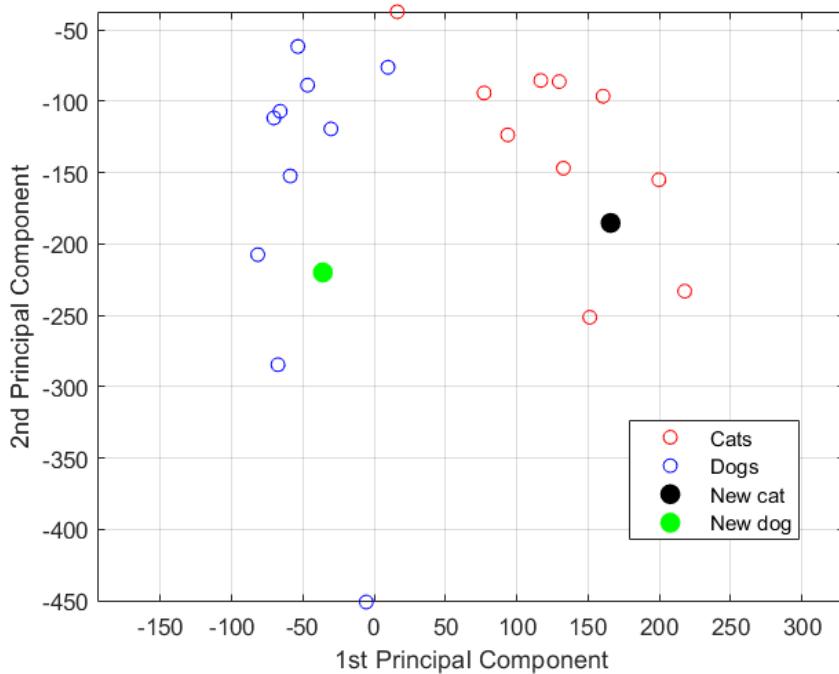


(a) a new cat image

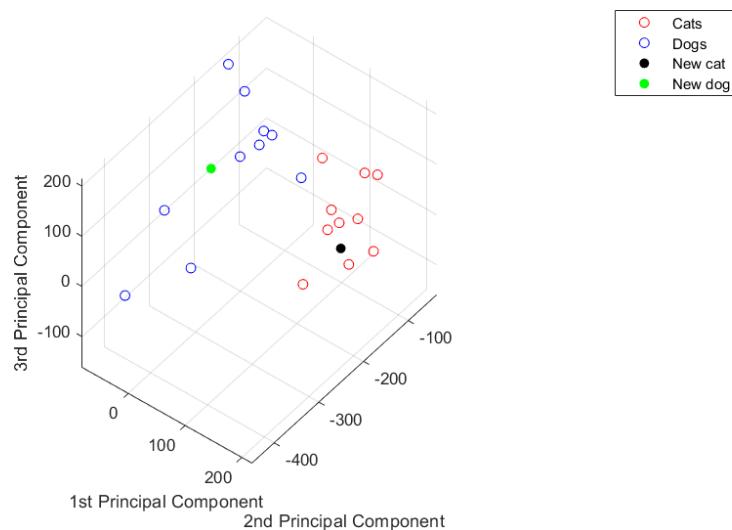


(b) a new dog image

Figure 30: a Cat and a Dog new images



(a) PCA with 2 PC

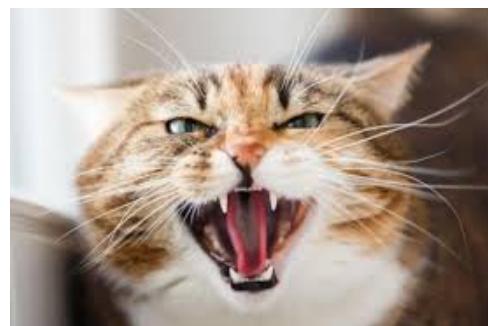


(b) PCA with 3 PC

Figure 31: new Cat and Dog images in PC scores graphs



(a) a new cat image



(b) cat from library - cat\_8

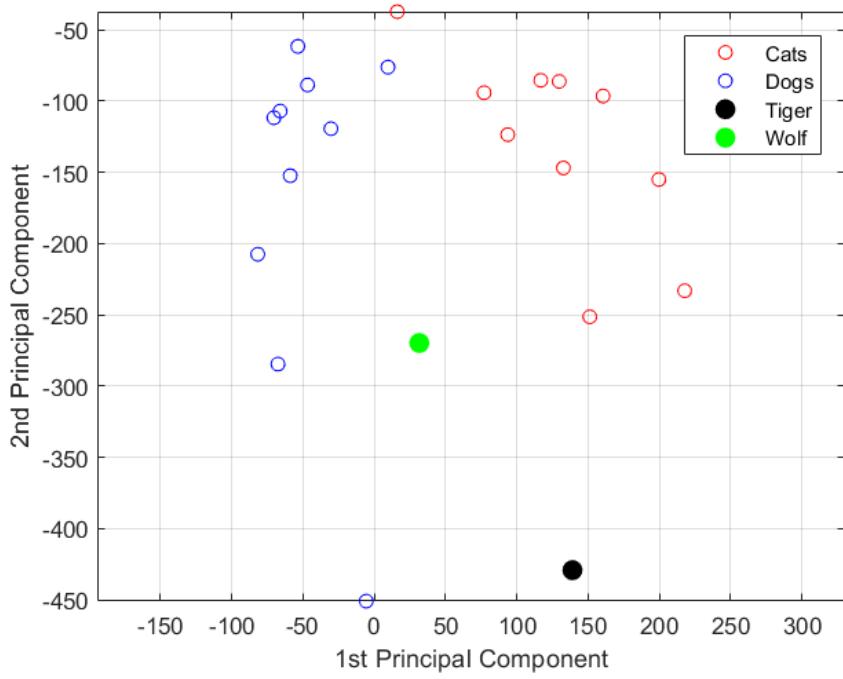


(c) a new dog image

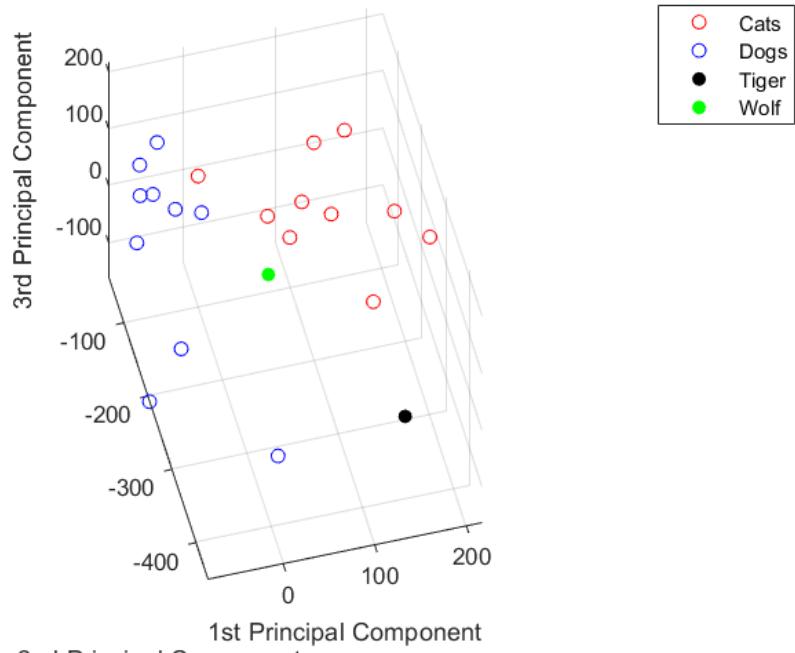


(d) dog from library - dog\_5

Figure 32: Closest cats and dogs images



(a) PCA with 2 PC



(b) PCA with 3 PC

Figure 33: Tiger and Wolf PCA scores on existing PCA graph



(a) Tiger



(b) cat from library - cat\_7



(c) Wolf



(d) dog from library - dog\_7

Figure 34: Closest cats and dogs images

### III. Image Stitching

## I. Image Stitching Section

### A. Step 1

We extracted sift features from interest points from all images in Fig. 1. The images with more details contain more interest points including the sky gradients. Those interest points will be discarded when matching those points.

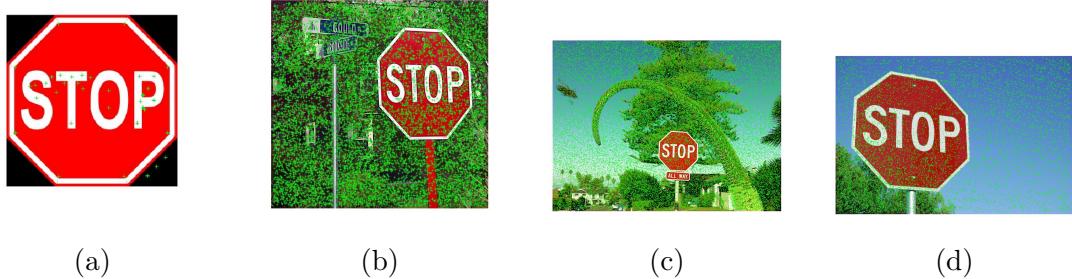


Figure 1: Sift feature points for each Stop sign image, each point presented as a green cross.

### B. Step 2

Using the *siftmatch* function we do a preliminary match between similar sift features in image 1 to all the rest (Fig. 2). While generally the SIFT match is good, there are visible outliers created by this algorithm (especially visible in Fig. 2a), outliers that will be cleaned next.

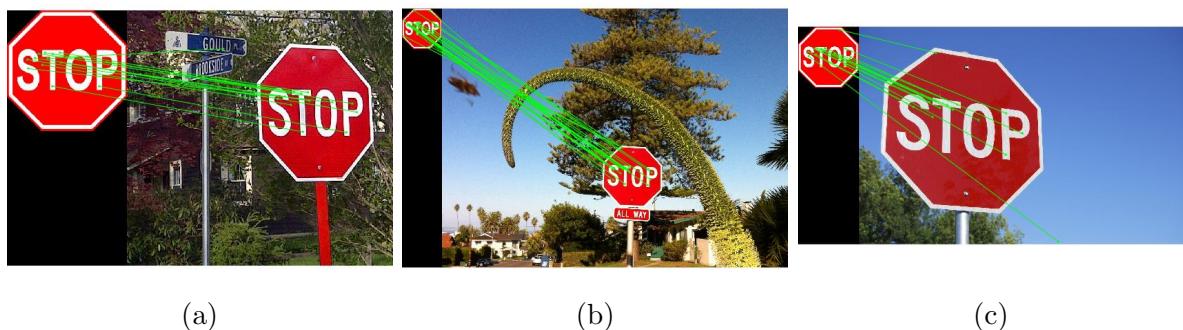


Figure 2: Sift feature matching between images and the nominal Stop sign, green lines connect between matching features.

### C. Step 3

The next step is to perform RANSAC to weed out the outliers and find the affine transformation matrices from all the inliers. Eq. I.C presents all the affine matrices, where  $H_i$  is the transformation matrix between image 1 and image  $i$ . As a requirement, perspective

distortion wasn't considered, and the effect will be evident in a later section where the stitching is not aligned perfectly with one of the images.

$$H_2 = \begin{bmatrix} 0.9382 & 0.0487 & 322.1 \\ 0.0193 & 1.1876 & 74.0131 \\ 0 & 0 & 1 \end{bmatrix} \quad H_3 = \begin{bmatrix} 1.4661 & -0.047 & 1212 \\ 0.053 & 1.5614 & 1011 \\ 0 & 0 & 1 \end{bmatrix} \quad H_4 = \begin{bmatrix} 2.613 & -0.1745 & 154.9 \\ 0.3577 & 2.564 & 59.74 \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

## D. Step 4

In this step, we present the interest point matches using both SIFT feature similarity and RANSAC outlier filtering in Fig. 3. We see most of the interest points were preserved and all the obvious outliers removed, in particular compare between Fig. 3a and Fig. 2a, where the obvious horizontal outlier was removed.

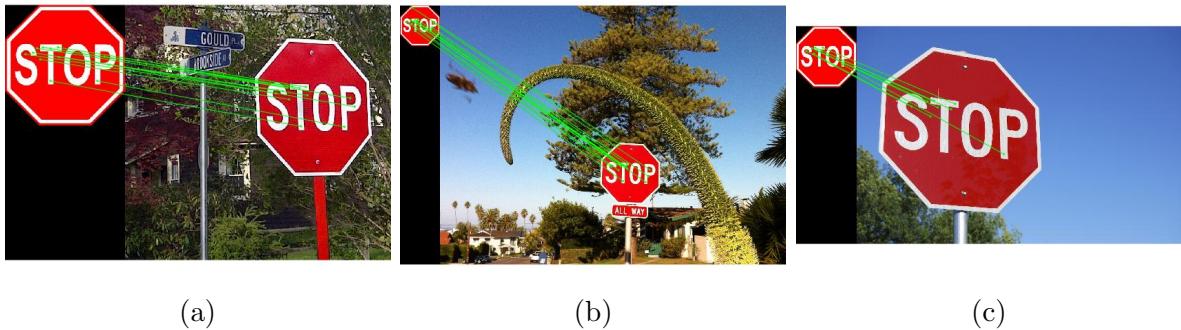


Figure 3: Matching features after RANSAC with the most inliers.

## E. Step 5

In this step we used the affine transformation to project the sign from image 1 to the sign location of the other images (Fig. 4). Note that in some of the images the stop sign had higher resolution than image 1, so our approach interpolated the expected pixel in those areas to avoid gaps within the pixels.

## F. Step 6

In this last step we stitched the projected sign from image 1 to the other images. We used NaN in projected pixels that don't appear in the first image, and we filtered out most of the black background of image 1 sign as well (for more accurate results, a mask of the stop sign can be used). Generally the sign was stitched well into the other images (not including lighting of course), but not perfectly as we can see best in Fig. 5c where the affine transformation was insufficient in describing the projection (requiring perspective as well).

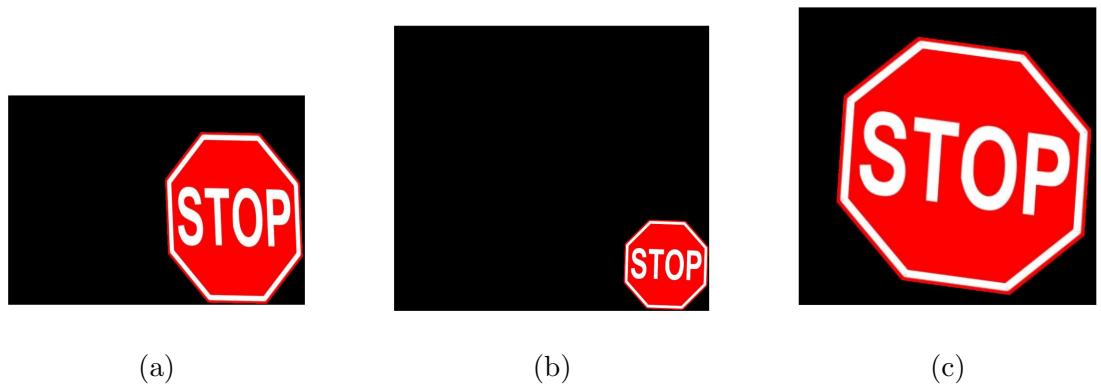


Figure 4: Stop sign transformation from 1st image to all other images using the affine transformation.

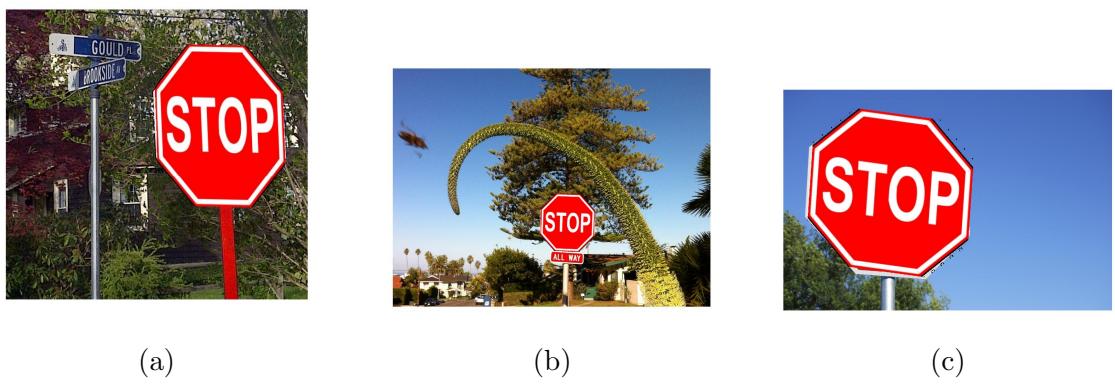


Figure 5: Stop sign from the 1st image glued into all others.