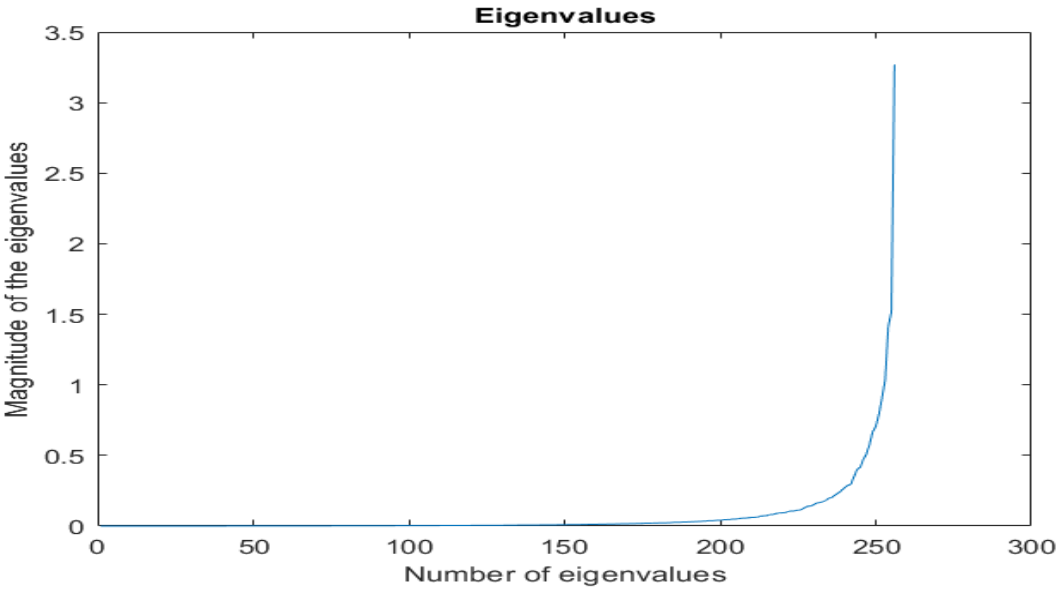
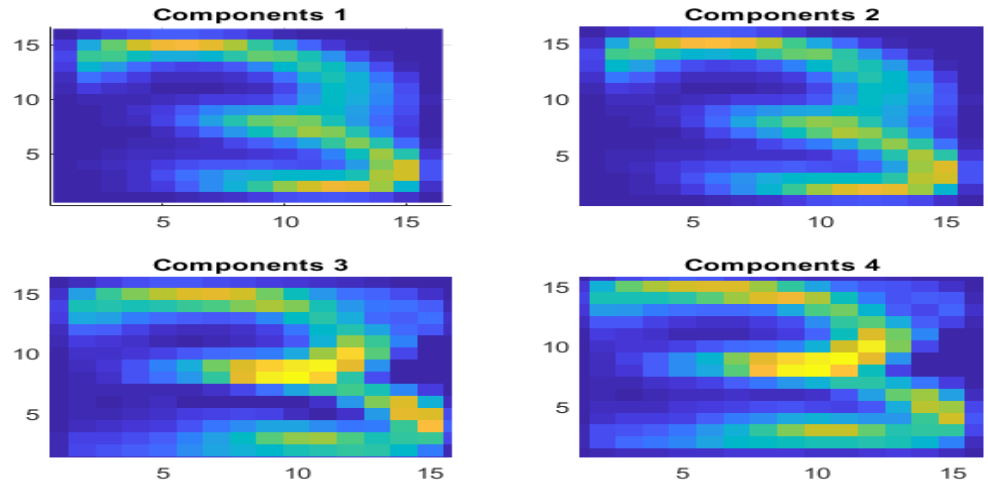
Assignment 3 (ANN & Deep learning)

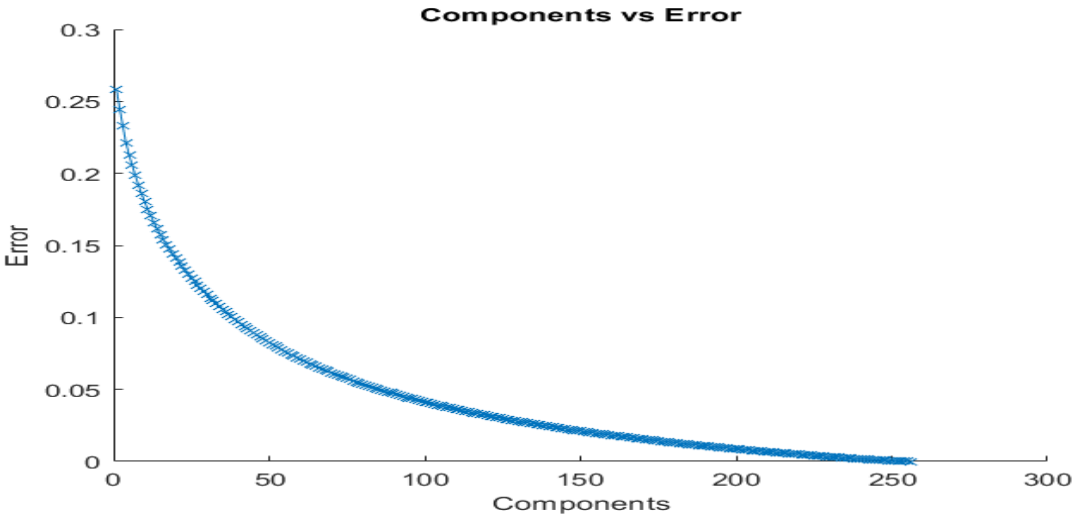
Student: Afraz Salim *r-number: r0439731*

In this section, we will build a PCA for the compression and reconstruction of images. Principle components can be computed in two ways. We can simply compute the covariance matrix. Since covariance matrix is symmetric, we know that it has eigen value decomposition. We can also compute principle components by making use of singular value decomposition. However, in this section, we will use simple approach. We take our data matrix and compute the covariance and then perform eigenvalue decomposition. Since, we are looking for axis, along which variance is maximum. We can plot eigenvalues and decide how many components we want to retain.

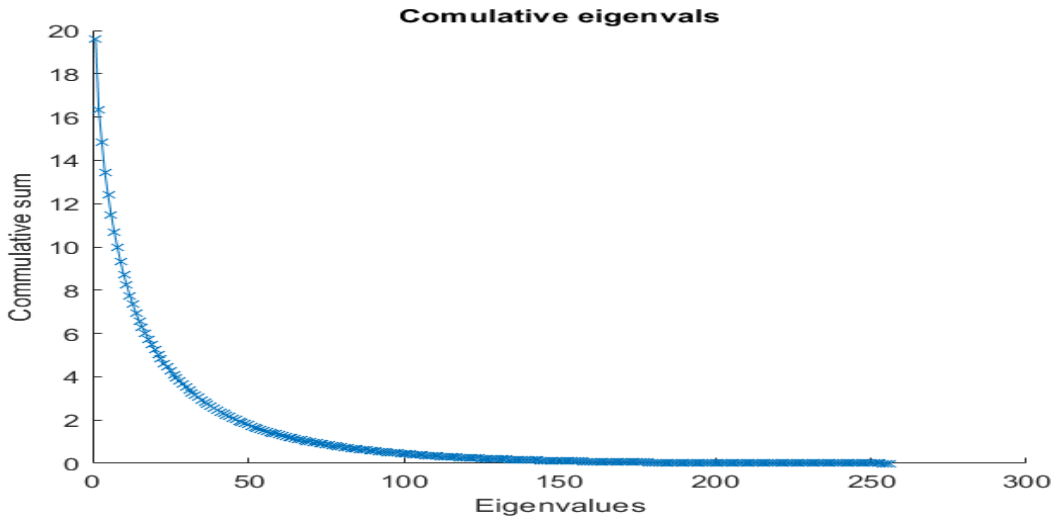




We have in total 256 eigenvalues and we see that only few eigenvalues have very high magnitude. Next we taken n principle components for compression and reconstruction of the data. Number of components are also shown in the figure above.
Now we compute the error between the original image and the image that we reconstruct. We will increase the number of components and we expect the error to minimize.



The error plot shows that if we use all principle components then the error will be zero. We compute it explicitly and found that the error is not 0 but $6{,}0267e-16$. This error seems to be result of machine epsilon.
Next, we can compare the error plot with cumulative eigenvalues plot. The figure below shows that the cumulative eigenvalues plot is same as error plot but on different scale.



Stacked autoencoders and MLP

In this section, we will compare the performance of stacked autoencoders with MLP for the reconstruction of digits. We first executed the *stackautoencoderdigitsclassification.m* and got the accuracy as 99,66.

Model	layers	Neurons	accuracy without Fine tune	Acc with fine tune
StackedAutoEnC(DeepNet)	3	[100, 50, 10]	82,06	99,66
MLP	2	[100, 10]	-	95,84
MLP	3	[100, 50, 10]	-	96,78
MLP	4	[100, 50, 50, 10]	-	95,82

Initially we obtain better results with stackedautoencoder after fine-tuning. Next, we tried to optimize MLP to get better results. However, changing the parameter or adding extra layers to the MLP does not give any better results. We see that validation checks fails reach their limit after very few epochs and it shows that network is trying to remember things. So instead of increasing the size of the hidden layers, we add one extra layer to check if there is any difference in performance. However, we found that it is very hard to stop model from overfitting. Increasing the size of the hidden layers results in few more epochs before validation check fail reaches it's limit. Increasing the size of the hidden layers also did not result in better performance. The table shown above, list different architecture and accuracy.

Fine-tuning:

Fine-tuning is an important step in order to obtain better results. Each layer of deep stacked autoencoder is trained separately and then these layers are combined. These layers expect an input as original data and not the outputs from the previous layers.
Fine-tuning helps the layers to adjust their weights according to the input from the previous layers. A stacked autoencoder is simply MLP which is trained in a different way. We saw while training the MLP, that initial layers start to remember things while adding layers at later stage does not result in any improvment. However, if we increased the size of the first hidden layer, then we got slightly better results. We can conclude from such behaviour of the MLP that initial layer is trying to learn more things than it should learn. We avoid such problem by training a MLP as stacked autoecnoder. In this way, each layer has almost same information about the input. However, there is some discontinuity between layer and such discontinuity is removed with fine-tuning.

Conclusion

We can conclude that it is hard task to train a MLP without overfitting. Overfitting is caused mostly by the first hidden layer and the layers after that layer do not learn much. Since first layer tries to remember things, we get validation check fails.

CNN

CNN architecture are special type of neural network architectures which are designed to learn from the data which is in grid format. In this section, we will discuss different parameters of convolutional neural networks. More specifically, we will analyze alexnet.

- Take a look at the first convolutional layer (layer 2) and at the dimension of the weights

```
(size(convnet.Layers(2).Weights))
```

  weights represent?

These weights represent filter. We have in total 96 filters at second layer and each filter has size 11×11 with depth 3. This can written in following format : $11 \times 11 \times 3 \times 96$. We have also one bias vector which is $1 \times 1 \times 96$. If we apply these filters then the dimension of the output should be:

$$\frac{input\_size - filter\_size}{strides} + 1$$

The first layer isthe input layer and images have size= 227. So wet get new image dimensions as $\frac{227-11}{4}+1$. In our case, we have 96 filters, so we get $96 \times 55 \times 55$.

- Inspect layers 1 to 5. If you know that a ReLU and a Cross Channel Normalization layer do not affect the dimension of the input, what is the dimension of the input at the start of layer 6 and why?

Layer 5 is a pooling layer. The input to this layer has dimension $55 \times 55 \times 96$. Layer 5 does the max_pooling with window size 3×3,stride $= 2$ and padding $= [0, 0, 0, 0]$. Since the input has same height and width, we represent width and height with W. Let F represent the window/filter size and S represents the stride. We can calculate the output as follows: $\frac{W-F+2*P}{strides} + 1 \rightarrow \frac{55-3+2*0}{2} + 1 = 27$. Hence, the input to layer 6 has dimension $27 \times 27 \times 96$.

- What is the dimension of the inputs before the final classification part of the network (i.e. before the fully connected layers)? How does this compare with the initial dimension? Briefly discuss the advantage of CNNs over fully connected networks for image classification.

The layer before the fully connected layer is a max pool layer with the output dimension equals to $6 \times 6 \times 256$. Max pool layers are used to reduce the size(they can also be used to upsample the images) and convolution layers are used to learn weights to extract the features. However, fully connected layers are used for the classification. Layer 17 is first fully connected layer which takes an input of dimension $6 \times 6 \times 256$ and outputs 4096 activation maps with dimension 1×1. These 4096 maps will be reduced to 1000 classes and each of them will be assigned a probability.

CNN are specially designed to preserve the spatial structure of images. Convolution and pooling layers preserve the spatial structure while fully connected layers at the end are used for predictions.

- The script *CNNDigits.m* runs a small CNN on the handwritten digits dataset. Use this script to investigate some CNN architectures. Try out some different amount of layers, combinations of different kinds of layers, dimensions of the weights, etc. Discuss your results. Be aware that some architectures will take a long time to train!

As mentioned previously, CNN are designed to preserve the spatial structure of the grid data. There are two type of layers, convolutional and maxpooling. Convolutional layers are used to extract the features while fully connected layers are used for classification. In this section, we will analyze different architecture of CNN and compare there performance. We will be focusing on the kernel size and it's impact and number of layers.
We start with the default architecture and try to improve it.

```
layers = [imageInputLayer([28 28 1]),
          convolution2dLayer(5,12),reluLayer
          ,maxPooling2dLayer(2,'Stride',2),
          convolution2dLayer(5,24),reluLayer
          ,fullyConnectedLayer(10),
          softmaxLayer,classificationLayer()];
```

The accuracy that we obtain with this model is 0,8376. Now the first improvement that we can make is by removing the maxpool layer because it removed many features from the activation maps. Now the accuracy that we obtain is 0,9600. However in real cases, maxpool layers are needed otherwise we need more convolutional layers to decrease the size of the image. Hence, more paramters are needed to be trained.
A second improvement will be to replace the kernel in first convolutional layer by a kernel of smaller size.

We want to extract more small features in begining and then later layers can combine those features to extract useful information. By replacing the kernel size(5) by the kernel of size 3, we get the accuracy of 0,9648.
Since we are decreasing the kernel size, we can increase the number of activation map(number of kernels). By doing so, we get the accuracy of 0,9676.
If we increase the kernel size then the accuracy curve shows more fluctuations. Such behaviour arises because by increasing the kernel size, we are extracting the features from a large portion of the image. Since a small area will have less changes from image to image then a large area, using smaller kernels results in smooth accuracy curve. Best results which were obtained are shown in 1.

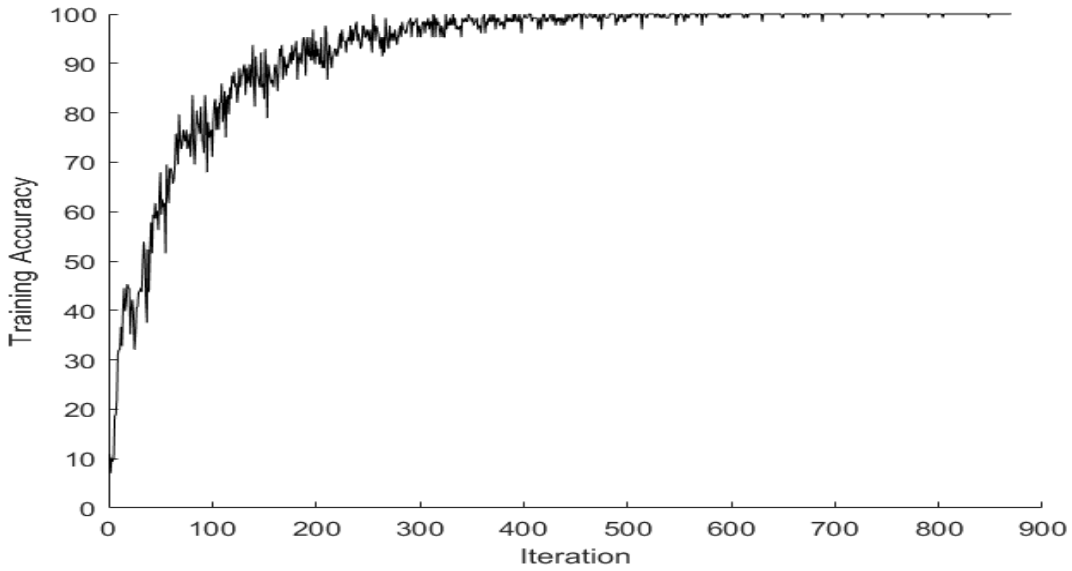


Figura 1: Best results obtained with accuracy 0,9696

Conclusion on CNN architectures:

Maxpool layers are used in real applications because we want to reduce the size of the image. However, reducing the size of the image results in loss of information. Small features are very important for classfication tasks. In other words, small feature values describe the characteristics of the objects in image.
Depending on the requirements, CNN architectures can be adjusted to obtain better results or we can allow more errors and train a model in less time.