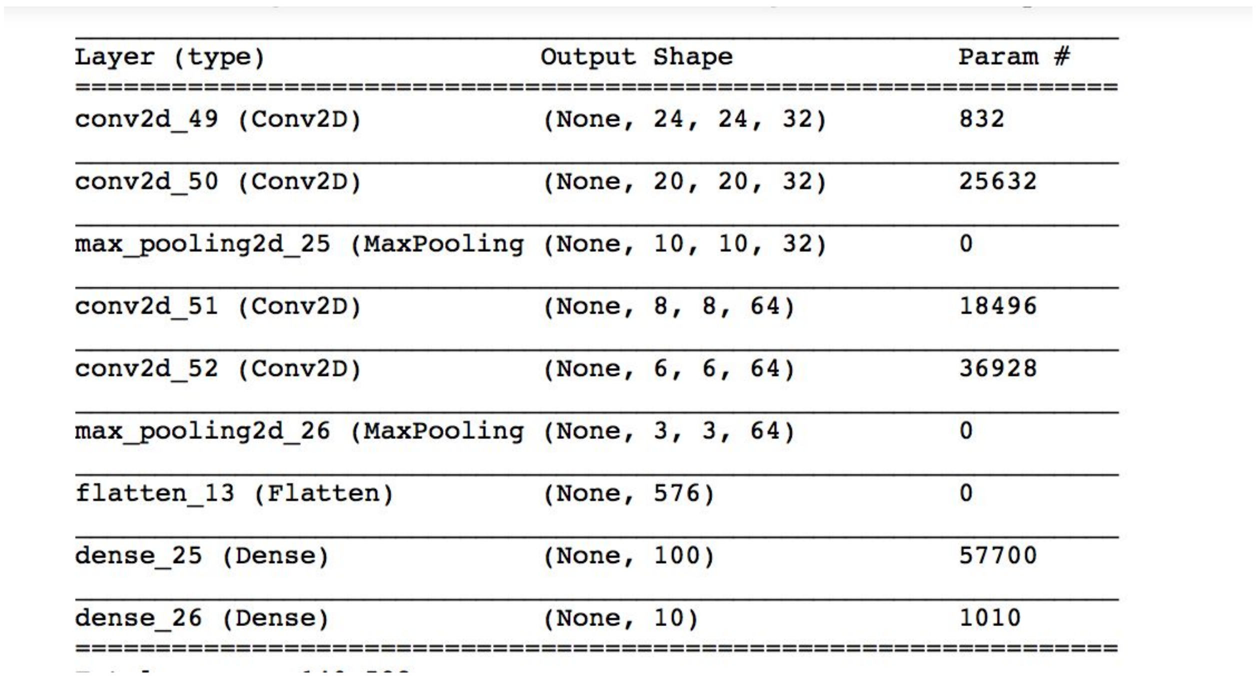
Problem 1

We construct the Convolutional Neural Network with the following architecture. The input we consider here is MNIST handwritten digit dataset.

Layer #	Type of Layer	Units	Kernel/Pool Size	Stride	Activation Function
0	2D Convolution	32	(5,5)	(1,1)	ReLU
1	2D Convolution	32	(5,5)	(1,1)	ReLU
2	2D Max Pooling	-	(2, 2)	(2, 2)	-
3	2D Convolution	64	(3, 3)	(1, 1)	ReLU
4	2D Convolution	64	(3, 3)	(1, 1)	ReLU
5	2D MaxPooling	-	(2, 2)	(2, 2)	-
6	Flatten	-	-	-	-
7	Dense	100	-	-	ReLU
8	Dense	10	-	-	Softmax

Optimizer: SGD (lr = 0.01, momentum=0.9) Accuracy: 99.42% (Test) Loss: Categorical Cross. Entropy



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_49 (Conv2D)           (None, 24, 24, 32)        832
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 20, 20, 32)        25632
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 10, 10, 32)        0
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 8, 8, 64)          18496
_________________________________________________________________
conv2d_52 (Conv2D)           (None, 6, 6, 64)          36928
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 3, 3, 64)          0
_________________________________________________________________
flatten_13 (Flatten)         (None, 576)               0
_________________________________________________________________
dense_25 (Dense)             (None, 100)               57700
_________________________________________________________________
dense_26 (Dense)             (None, 10)                1010
=================================================================
```

Fig: Network Summary

Part 1:

In this problem we visualize the filters of first 3 convolutional layer which is shown in below

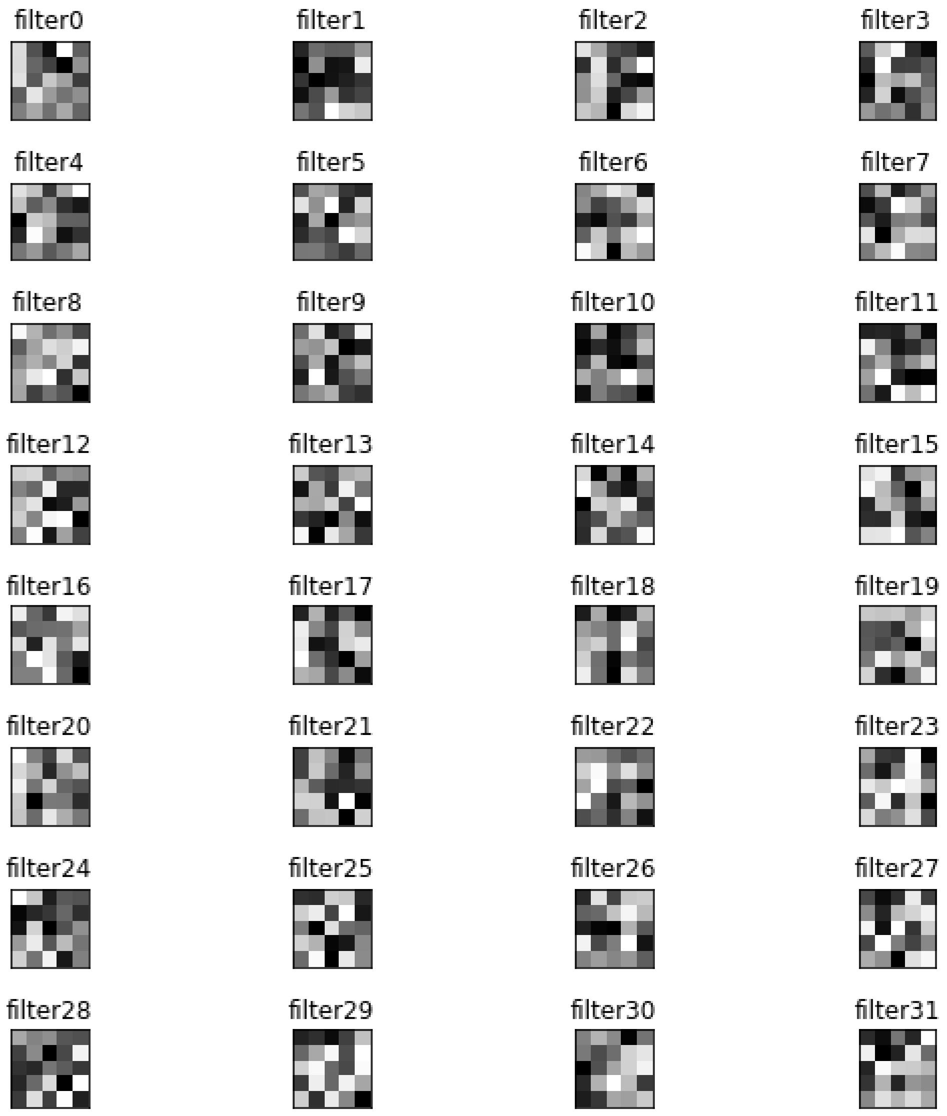


Fig: conv2D 1st layer (32 filters)

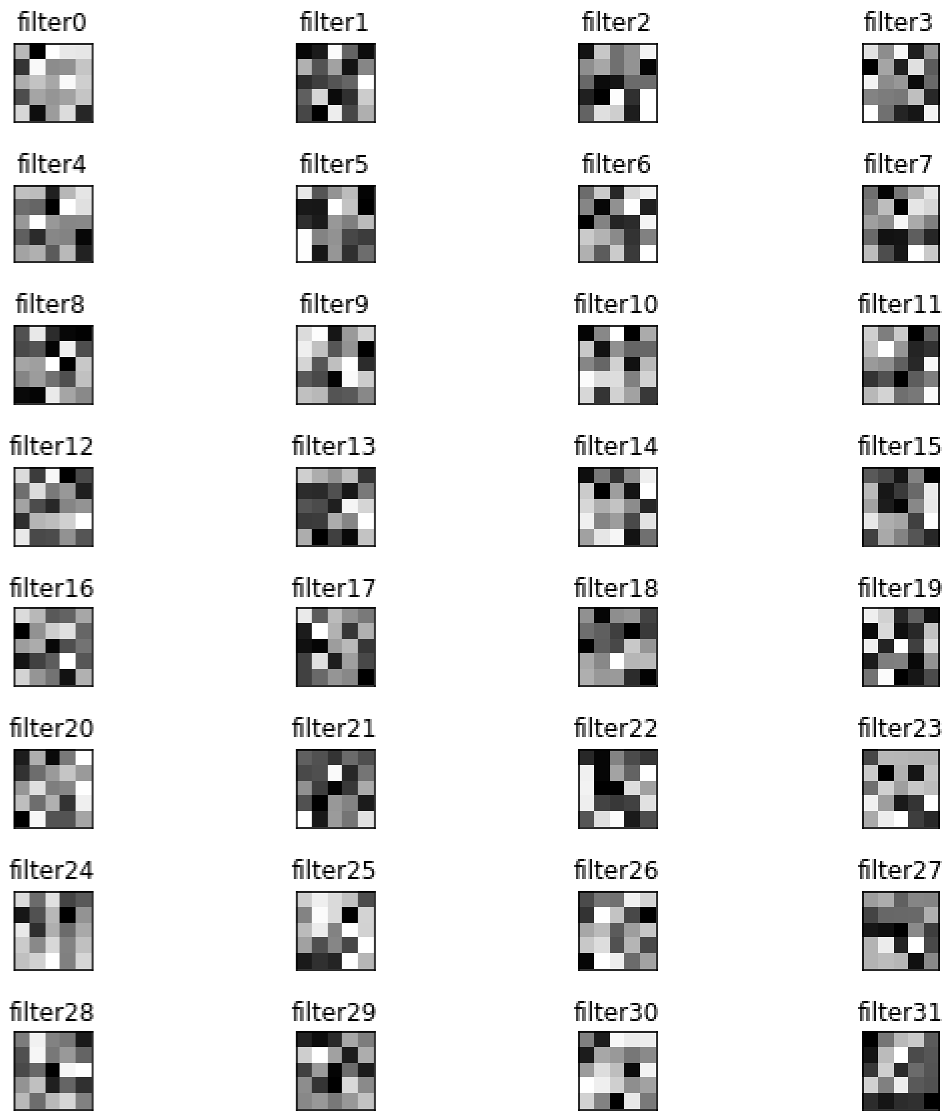


Fig: conv2D 2nd layer (32 filters)

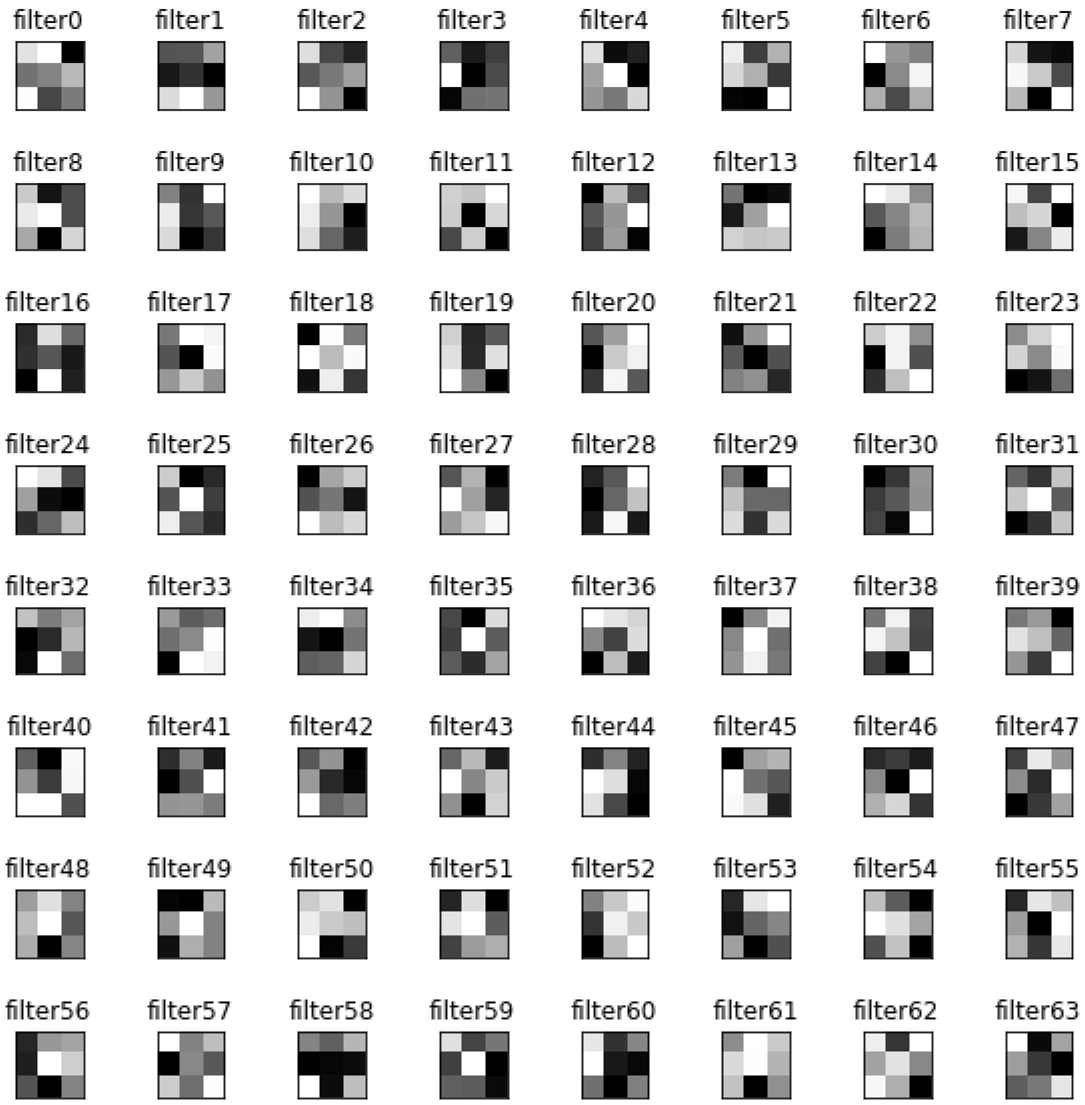


Fig: conv2D 3rd layer (64 filters)

Part 2

Following the instruction of this problem at first, We selected 2 samples (‘0’ and ‘8’) from the testing set where the prediction made by the network initially is also true. Then, we fed these inside the network and later on, the feature maps (images) of these samples in the 3rd convolutional layer are taken as the output. The feature maps are shown below-

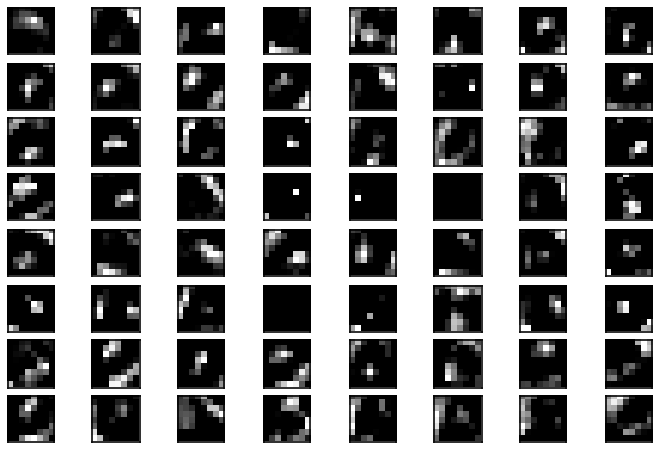


Fig: Feature map of ‘0’ in third Conv Layer for 64 filters

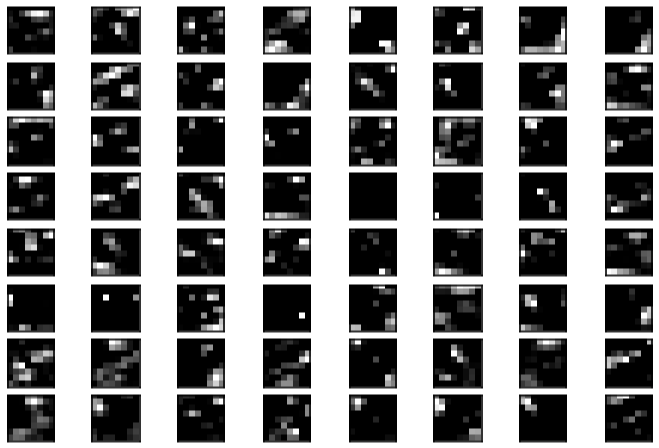
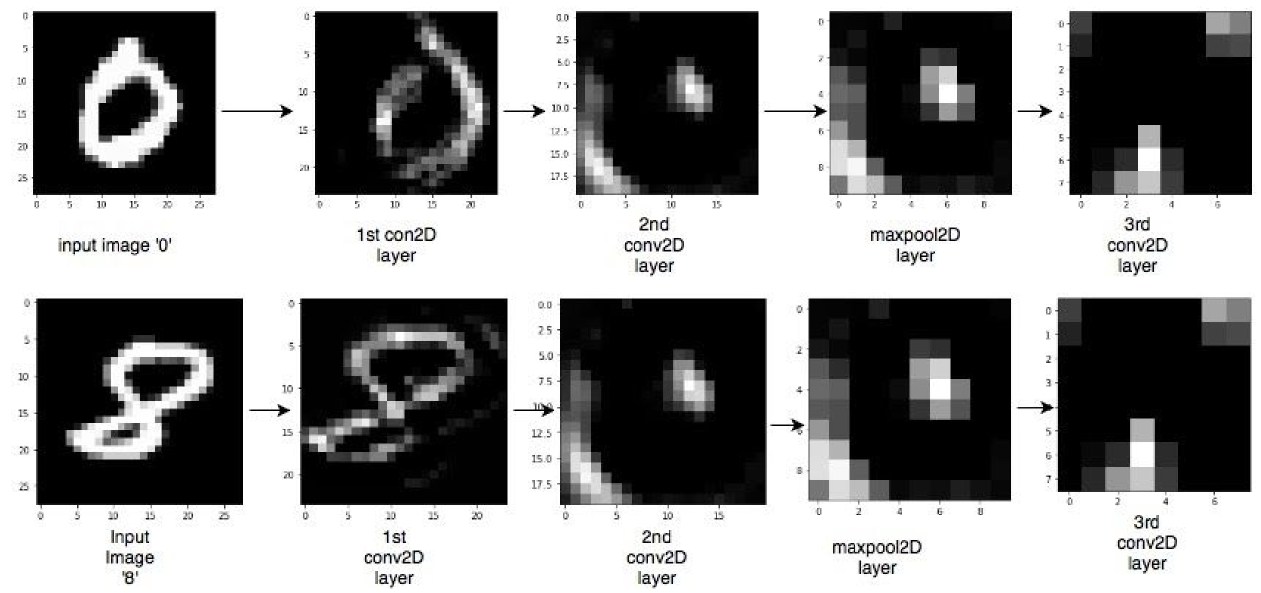


Fig: Feature map of ‘8’ in third Conv Layer for 64 filters

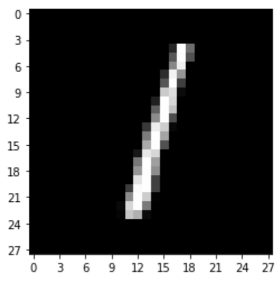
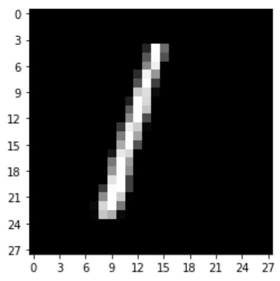
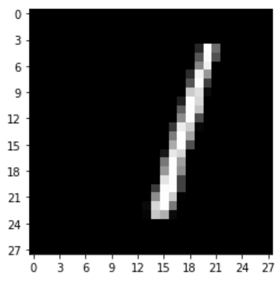
Just looking at the feature maps of the third convolutional layer will yield any information as to what are the most important discriminatory features, as the only point we can infer from these feature maps is that the network will consider an image
to be 0 if the output (6,0) and (5,0) of the third convolutional layer have a high value, while an image will be classified as an 8 based on the value of
the outputs (1,1) and (5,2) of the third convolutional layer. However, this does not give us any actual information as to what these outputs mean; in other words, what particular patches/areas will make the value of these 4 output cells change. In order to get a more in-depth analysis of that, we need to see how the output of the convolutional layer was generated from the previous layer, and so on. This
is shown in the 2 series of images below:



As we can see, as we go through the outputs of each layer from the start, till the 3rd convolutional layer, we see that the main discriminatory feature between the 2 images are mainly the region in the middle of the 2 images. In case of 0, the region in the middle is fairly wide, and the 2 sides do not converge, as opposed to 8, where there is a slight depression in the middle of the images. Thus, the 2 outputs at the end of the 3rd convolutional layer, can be somewhat construed as an indicator of whether or not the middle region has just 2 wide curved lines (0), or a converging curved lines (8).

Part 3

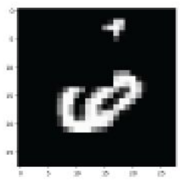
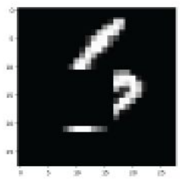
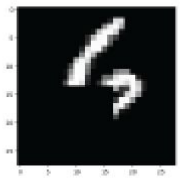
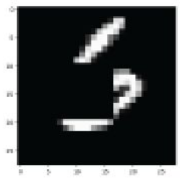
Using keras image preprocessing library, we shifted the image of ‘1’ left by 3 pixels and to the right 3 pixels, using the default border padding method. The image and the prediction of the network is given below.

Image	Remarks	Prediction	Prediction Probability
	Actual Image	1	0.99985516
	Move left by 3 pixels	1	0.99613357
	Move right by 3 pixels	1	0.72105975

Problem 2

Part 1

For the 2nd problem, we generated a large number of images by sliding a 8x8 black patch with stride 1, from left to right, top to bottom of a sample of an image of ‘6’. All such possible images were then fed to the network, which gave a test accuracy of 97.723%. Among all of these images, we chose the 4 significant cases of maps, which are shown below, with the probability of the image being of label ‘6’, the classified label, and the confident of classification (probability of
the classified label) :

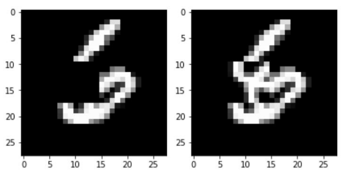
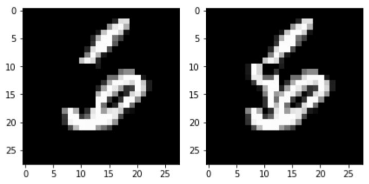
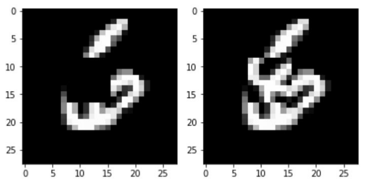
Image (x)	P(y=6|x)	y_pred	P(y_pred|x)
	1.0	6	1.0
	1.0	6	1.0
	0.0	5	1.0
	0.0	1	1.0

Part 2

From the above table, it can be interpreted that the bottom-left region, where the 'handle' of 6 meets the round part, is the most important feature, which the network looks at to recognize the model. Of more importance, however, is the fact of sensitivity of the network to this region. Looking at the 2nd and the 4th image illustrates this hypothesis. The black patch is just a few pixels off between the 2 images, yet it perfectly classifies the 2nd image, while completely misses the 4th one. One conjecture, may be the fact that while the model, after looking at the whole of the 2nd image guesses that a black patch is obscuring a region of 6, in the 4th image, the model gets confused due to the length of the 'handle' of 6 being similar to that of an image of 1 from the training set, and thus considers it to be a 1, instead of a 6.

Part 3

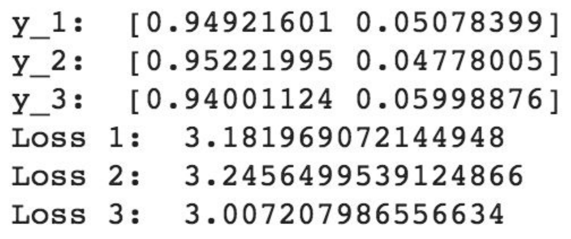
From the results, we realized that the most important feature of 6 is its bottom left area. The paper given in the assignment also helped us understand how we can easily make adversarial examples, by superimposing parts of other images. Wehypothesiz, that if we can cover up some patches of 6 using images of other digits, we can get an adversarial example. Now, again, considering we know the most important and distinguishing region of any image of 6 according to the network, we can easily create an adversarial network by imposing the image of 5 (closest structure to 6) by putting part of 5, on the bottom-left region. We can, obviously, create adversarial examples, by putting patches of 5 in any other regions, but they won't be misclassified as frequently as putting part of 5 on the bottom-left area. A similar reasoning goes for picking an image of 5; it is very close in structure, to 6, and thus, we may be able to easily confuse the model by superimposing part of 5 in an image of 6. We again consider a 10x10 patch sliding over the previously chosen image of 6, but in this case, instead of that patch being completely black, it basically consists of the pixel values of an image of 5 at that corresponding area. We can consider the patch as some form of a window, that exposes the image of a 5 placed below the image of 6. We again slide this from left to right, top to bottom, and get all the possible images that can be generated thus. Then we test these images on the model. All the images' labels are 6. The model returns an accuracy of 87.658%, in this case, a sharp decrease from Part 1. Amongst the incorrect images, we pick 5 samples again, and show their predicted class, in the table which is shown below:

Image (x)	Y_pred with Just black patch	Y-pred with with adversarial
	3	5
	6	5
	6	5

Problem 3

Part 1:

In this problem, we simulated the working of an RNN with W, U, V, b, c, given to us, on $x^{(1)}, x^{(2)}, x^{(3)}$. The values of the output, as well as the losses are given in the table below:



```
y_1:  [0.94921601 0.05078399]
y_2:  [0.95221995 0.04778005]
y_3:  [0.94001124 0.05998876]
Loss 1:  3.181969072144948
Loss 2:  3.2456499539124866
Loss 3:  3.007207986556634
```

Part 2

The values of the output, as well as the losses are given in the table below:

$\frac{\partial \mathbf{L}^{(3)}}{\partial \mathbf{b}_1}$	-0.009961
$\frac{\partial \mathbf{L}^{(3)}}{\partial \mathbf{b}_2}$	0.434009

Part 3

We compute the actual gradient of loss with respect to the 2 bias terms below:
We know that for an RNN, the related equations are -

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{t}$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)}$$

$$\hat{\mathbf{y}}^{\mathbf{t}} = \mathbf{softmax}(\mathbf{o}^{\mathbf{t}})$$

Now, L depends on b1 through a1 and a2. Thus, we need to compute those derivatives. These derivatives can be found as:

$$\frac{\partial \mathbf{L}}{\partial \mathbf{a}_1^{(3)}} = \left[\frac{\partial \mathbf{L}}{\partial \mathbf{o}_1^{(3)}}\frac{\partial \mathbf{o}_1^{(3)}}{\partial \mathbf{h}_1^{(3)}} + \frac{\partial \mathbf{L}}{\partial \mathbf{o}_2^{(3)}}\frac{\partial \mathbf{o}_2^{(3)}}{\partial \mathbf{h}_1^{(3)}} \right]\frac{\partial \mathbf{h}_1^{(3)}}{\partial \mathbf{a}_1^{(3)}} \quad , \quad \frac{\partial \mathbf{L}}{\partial \mathbf{a}_2^{(3)}} = \left[\frac{\partial \mathbf{L}}{\partial \mathbf{o}_1^{(3)}}\frac{\partial \mathbf{o}_1^{(3)}}{\partial \mathbf{h}_2^{(3)}} + \frac{\partial \mathbf{L}}{\partial \mathbf{o}_2^{(3)}}\frac{\partial \mathbf{o}_2^{(3)}}{\partial \mathbf{h}_2^{(3)}} \right]\frac{\partial \mathbf{h}_2^{(3)}}{\partial \mathbf{a}_2^{(3)}}$$

Now, as we know the loss function to be

$$\mathbf{L} = (\hat{\mathbf{y}}_1^{\mathbf{t}} - \mathbf{0.5})^2 - \log(\hat{\mathbf{y}}_2^{\mathbf{t}})$$

Thus, we can easily get

$$\frac{\partial \mathbf{L}}{\partial \mathbf{o}_1^{(3)}} = [\ 2(\hat{\mathbf{y}}_1^t - 0.5)(\hat{\mathbf{y}}_1^t - \hat{\mathbf{y}_1^t}^2)] \qquad \frac{\partial \mathbf{o}_1^{(3)}}{\partial \mathbf{h}_1^{(3)}} = \mathbf{V}_{11} \quad \frac{\partial \mathbf{o}_2^{(3)}}{\partial \mathbf{h}_1^{(3)}} = \mathbf{V}_{21}$$

$$\frac{\partial \mathbf{L}}{\partial \mathbf{o}_2^{(3)}} = [\ 2(\hat{\mathbf{y}}_1^3 - 0.5)\ \hat{\mathbf{y}}_1^3\ \hat{\mathbf{y}}_2^3\ +\ (\hat{\mathbf{y}}_2^3 - \hat{\mathbf{y}_2^3}^2)\frac{\mathbf{1}}{\hat{\mathbf{y}}_2^3}] \qquad \frac{\partial \mathbf{o}_1^{(3)}}{\partial \mathbf{h}_2^{(3)}} = \mathbf{V}_{12} \quad \frac{\partial \mathbf{o}_2^{(3)}}{\partial \mathbf{h}_2^{(3)}} = \mathbf{V}_{22}$$

$$\frac{\partial \mathbf{h}_1^{(3)}}{\partial \mathbf{a}_1^{(3)}} = [\ \mathbf{1} - \mathbf{h}_1^{(3)2}\ ] \qquad \frac{\partial \mathbf{h}_2^{(3)}}{\partial \mathbf{a}_2^{(3)}} = [\ \mathbf{1} - \mathbf{h}_2^{(3)2}\ ]$$

Now, in case of $\frac{\partial \mathbf{a}_1^{(3)}}{\partial \mathbf{b}_1}$, it gets unrolled as

$$\frac{\partial \mathbf{a}_1^{(3)}}{\partial \mathbf{b}_1} = \mathbf{1} + \mathbf{W}_{11} \cdot \frac{\partial \mathbf{h}_1^{(2)}}{\partial \mathbf{a}_1^{(2)}} \cdot \frac{\partial \mathbf{a}_1^{(2)}}{\partial \mathbf{b}_1} + \mathbf{W}_{12} \cdot \frac{\partial \mathbf{h}_2^{(2)}}{\partial \mathbf{a}_1^{(2)}} \cdot \frac{\partial \mathbf{a}_1^{(2)}}{\partial \mathbf{b}_1}\text{, and } \frac{\partial \mathbf{a}_2^{(3)}}{\partial \mathbf{b}_1}\text{, we can write}$$

$$\frac{\partial \mathbf{a}_2^{(3)}}{\partial \mathbf{b}_1} = \mathbf{W}_{21} \cdot \frac{\partial \mathbf{h}_1^{(2)}}{\partial \mathbf{a}_2^{(2)}} \cdot \frac{\partial \mathbf{a}_2^{(2)}}{\partial \mathbf{b}_1} + \mathbf{W}_{22} \cdot \frac{\partial \mathbf{h}_2^{(2)}}{\partial \mathbf{a}_2^{(2)}} \cdot \frac{\partial \mathbf{a}_2^{(2)}}{\partial \mathbf{b}_1}$$

Thus, we see a recursive situation here, which after complete unrolling would result in the following composite values of the 2 derivatives above -

Thus, we see a recursive situation here, which after complete unrolling would result in the following composite values of the 2 derivatives above -

$\frac{\partial \mathbf{a}_1^{(3)}}{\partial \mathbf{b}_1} = 2.572895$, and $\frac{\partial \mathbf{a}_2^{(3)}}{\partial \mathbf{b}_1} = 0$

Thus, now, putting these values in the 2 Loss derivative equations, we see that,

$$\frac{\partial \mathbf{L}}{\partial \mathbf{b}_1} = [\, \mathbf{2}(\hat{\mathbf{y}}_1^t - \mathbf{0\,.\,5})(\hat{\mathbf{y}}_1^t - \hat{\mathbf{y}_1^t}^2)\,.\,\mathbf{V}_{11}] + [\, \mathbf{2}(\hat{\mathbf{y}}_1^3 - \mathbf{0\,.\,5})\; \hat{\mathbf{y}}_1^3\, \hat{\mathbf{y}}_2^3 \;+\; (\hat{\mathbf{y}}_2^3 - \hat{\mathbf{y}_2^3}^2)\frac{\mathbf{1}}{\hat{\mathbf{y}}_2^3}\,.\,\mathbf{V}_{21}]$$

$$[\, \mathbf{1} - \mathbf{h}_1^{(3)^2}\,].\, 2.572895 + \frac{\partial \mathbf{L}}{\partial \mathbf{a}_2^{(3)}} . \frac{\partial \mathbf{a}_2^{(3)}}{\partial \mathbf{b}_1} = -0.009961 + 0 = -0.009961$$

For $\frac{\partial \mathbf{L}}{\partial \mathbf{b}_2}$, we shall have the same equations as we got last time, but the values of the derivatives will be different. As it may be too long to work out the whole derivation, we simply put the values of $\frac{\partial \mathbf{a}_1^{(3)}}{\partial \mathbf{b}_2} = -0.01329$ and $\frac{\partial \mathbf{a}_2^{(3)}}{\partial \mathbf{b}_2} = 1.02599$

Plugging these values into $\frac{\partial \mathbf{L}}{\partial \mathbf{b}_2}$ we get $\frac{\partial \mathbf{L}}{\partial \mathbf{b}_2} = \frac{\partial \mathbf{L}}{\partial \mathbf{a}_1^{(3)}} \frac{\partial \mathbf{a}_1^{(3)}}{\partial \mathbf{b}_2} + \frac{\partial \mathbf{L}}{\partial \mathbf{a}_2^{(3)}} \frac{\partial \mathbf{a}_2^{(3)}}{\partial \mathbf{b}_2} = 0.43401$

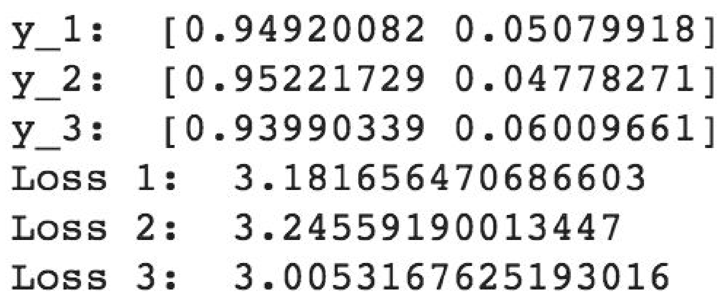
Part 4

We now compute one step of gradient descent using the derivative we calculated earlier. The learning rate is taken as 0.01. The new values for bias term are as follows:

b1	-1.02379892
b2	0.99573221

Part 5

Using the new values of bias, we now recompute the whole run again, with the same inputs. The loss and outputs at each time is given below:



```
y_1:  [0.94920082 0.05079918]
y_2:  [0.95221729 0.04778271]
y_3:  [0.93990339 0.06009661]
Loss 1:  3.181656470686603
Loss 2:  3.24559190013447
Loss 3:  3.0053167625193016
```

Problem 4

While using the RNN to learn and classify long protein sequences, we will have a lot of problems in trying to learn the features towards the end of the sequence. The longer the sequence, the harder the learning will become. This is because of the fact that the RNNs suffer from vanishing/exploding gradient problems. This happens due to the following reason. we know from chain rule that, for any random weight w in weight matrix,

$$\frac{\partial h_i^{(\tau)}}{\partial w} \propto \frac{\partial h_i^{(\tau)}}{\partial a_i^{(\tau)}} \cdot \frac{\partial a_i^{(\tau)}}{\partial w}$$

Now,

$$\frac{\partial a_i^{(\tau)}}{\partial w} \propto \frac{\partial h_i^{(\tau-1)}}{\partial w}$$

Part 2:

If we intend to classify the protein sequence using an echo state network model, we would keep the input -> hidden layer connection manually fixed at random values, while the training of the
model will simply enable it to learn the hidden->output connections.

Part 3:

The LSTM cell fixes the RNN's vanishing/exploding gradient problem in a very simple method. For an LSTM with weights W , bias b , hidden states h , cell state C , and a scaling value i , and forget term f , and **σ** be the activation, the equations are:

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t, \text{ where}$$

$$\hat{C}_t = \sigma(W . h_{t-1} + U . x_t + b)$$