Question 2

Code B

Architecture

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 30, 30, 64)	1792
batch_normalization_1 (Batch	(None, 30, 30, 64)	256
activation_1 (Activation)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 64)	36928
batch_normalization_2 (Batch	(None, 28, 28, 64)	256
activation_2 (Activation)	(None, 28, 28, 64)	0
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 64)	0
conv2d_3 (Conv2D)	(None, 12, 12, 128)	73856
batch_normalization_3 (Batch	(None, 12, 12, 128)	512
activation_3 (Activation)	(None, 12, 12, 128)	0
conv2d_4 (Conv2D)	(None, 10, 10, 128)	147584
batch_normalization_4 (Batch	(None, 10, 10, 128)	512
activation_4 (Activation)	(None, 10, 10, 128)	0
flatten_1 (Flatten)	(None, 12800)	0
dense_1 (Dense)	(None, 64)	819264
batch_normalization_5 (Batch	(None, 64)	256

activation_5 (Activation)	(None, 64)	0
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2080
batch_normalization_6 (Batch	(None, 32)	128
activation_6 (Activation)	(None, 32)	0
dropout_2 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 10)	330

==========
Total params: 1,083,754
Trainable params: 1,082,794
Non-trainable params: 960
_____

- I took inspiration from the architecture of LeNet and VGGNet while designing this Neural Network.
  The input has been reshaped to channels first order for easy visualization.
  The early layers are convolutional so as to reduce the number of parameters at the beginning itself, allowing for fully connected layers at the deeper ends of the network.
- Usage of convolutional layers also prevents overfitting to a certain degree.
- The first and third convolutional layers have padding so as to ensure that the layer spatial dimensions don't get crunched along every layer. Maxpool layers have been added separately to achieve this where required.
- There is a Batch Normalization layer between every convolutional/fully connected layer and its corresponding activation layer so as to ensure that the outputs of every layer have 0 mean and unit variance. This improves gradient flow through the network and allows for higher learning rates.
- I have also added Dropout as a form of regularization (even though Ioffe and Szegedy (2015) claims that Batch Normalization makes it redundant) as I noticed that without it, the network begins to overfit on the data by the 25th epoch, with validation accuracy getting stuck at 60%.
- I tried AdaDelta and Adam optimizers, and chose to go with Adam with a learning rate of 1e-4 as it gave slightly better results.

Trials

- I tried both sigmoid and tanh. As expected, both led to comparatively poorer results, with sigmoid being a bit worse than tanh. This must have been because tanh output is zero centered, while this isn't true for sigmoid.

- In practice, tanh and sigmoid are avoided because they kill gradients when they get saturated. Batch Normalization prevents this, but it is still better to go for other activation functions.
- I also tried ReLU, LeakyReLU and ELU, and finally settled with ELU as it gave good results.
- I wished to attempt Xavier Initialization and try maxout activation, but I wasn't able to do so due to lack of time.

Code C

For this part, I chose to extract the outputs from the Batch Normalization layer after the first Fully Connected layer in my network.
This was done because Batch Normalization ensures that output is normalized along every dimension, which is needed in this case as SVMs are not scale invariant.

Code output for part b

```
Using TensorFlow backend.
(50000, 32, 32, 3)
50000
Train on 40000 samples, validate on 10000 samples
Epoch 1/30
2017-10-04 02:55:07.503675: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.1 instructions, but these are available on your machine and could speed up CPU computations.
2017-10-04 02:55:07.503692: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.2 instructions, but these are available on your machine and could speed up CPU computations.
2017-10-04 02:55:07.503715: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX instructions, but these are available on your machine and could speed up CPU computations.
2017-10-04 02:55:07.503720: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX2 instructions, but these are available on your machine and could speed up CPU computations.
2017-10-04 02:55:07.503724: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use FMA instructions, but these are available on your machine and could speed up CPU computations.
2017-10-04 02:55:07.596531: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:893] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2017-10-04 02:55:07.596809: I tensorflow/core/common_runtime/gpu/gpu_device.cc:955] Found device 0 with properties:
name: GeForce GTX 965M
major: 5 minor: 2 memoryClockRate (GHz) 1.15
pciBusID 0000:01:00.0
Total memory: 3.94GiB
Free memory: 3.50GiB
2017-10-04 02:55:07.596841: I tensorflow/core/common_runtime/gpu/gpu_device.cc:976] DMA: 0
2017-10-04 02:55:07.596846: I tensorflow/core/common_runtime/gpu/gpu_device.cc:986] 0: Y
2017-10-04 02:55:07.596852: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1045] Creating TensorFlow device (/gpu:0) -> (device: 0, name: GeForce GTX 965M, pci bus id: 0000:01:00.0)
40000/40000 [==============================] - 37s - loss: 2.0029 - acc: 0.3172 - val_loss: 1.4075 - val_acc: 0.5231
Epoch 2/30
40000/40000 [==============================] - 37s - loss: 1.6271 - acc: 0.4329 - val_loss: 1.2404 - val_acc: 0.5803
Epoch 3/30
```

```
40000/40000 [==============================] - 36s - loss: 1.4529 - acc: 0.4948 - val_loss: 1.1140 - val_acc: 0.6389
Epoch 4/30
40000/40000 [==============================] - 36s - loss: 1.3436 - acc: 0.5350 - val_loss: 1.0526 - val_acc: 0.6486
Epoch 5/30
40000/40000 [==============================] - 36s - loss: 1.2695 - acc: 0.5603 - val_loss: 0.9877 - val_acc: 0.6690
Epoch 6/30
40000/40000 [==============================] - 36s - loss: 1.1977 - acc: 0.5894 - val_loss: 0.9574 - val_acc: 0.6800
Epoch 7/30
40000/40000 [==============================] - 37s - loss: 1.1499 - acc: 0.6078 - val_loss: 0.9090 - val_acc: 0.6958
Epoch 8/30
40000/40000 [==============================] - 37s - loss: 1.0955 - acc: 0.6259 - val_loss: 0.9057 - val_acc: 0.6898
Epoch 9/30
40000/40000 [==============================] - 37s - loss: 1.0426 - acc: 0.6448 - val_loss: 0.8894 - val_acc: 0.6987
Epoch 10/30
40000/40000 [==============================] - 37s - loss: 1.0136 - acc: 0.6578 - val_loss: 0.8498 - val_acc: 0.7100
Epoch 11/30
40000/40000 [==============================] - 37s - loss: 0.9705 - acc: 0.6726 - val_loss: 0.8592 - val_acc: 0.7082
Epoch 12/30
40000/40000 [==============================] - 37s - loss: 0.9307 - acc: 0.6848 - val_loss: 0.8258 - val_acc: 0.7206
Epoch 13/30
40000/40000 [==============================] - 36s - loss: 0.8871 - acc: 0.7023 - val_loss: 0.8431 - val_acc: 0.7048
Epoch 14/30
40000/40000 [==============================] - 36s - loss: 0.8589 - acc: 0.7124 - val_loss: 0.8484 - val_acc: 0.7052
Epoch 15/30
40000/40000 [==============================] - 36s - loss: 0.8358 - acc: 0.7207 - val_loss: 0.7989 - val_acc: 0.7249
Epoch 16/30
40000/40000 [==============================] - 36s - loss: 0.8023 - acc: 0.7310 - val_loss: 0.7742 - val_acc: 0.7338
Epoch 17/30
40000/40000 [==============================] - 36s - loss: 0.7737 - acc: 0.7405 - val_loss: 0.7823 - val_acc: 0.7287
Epoch 18/30
```

40000/40000 [==============================] - 36s - loss: 0.7541 - acc: 0.7468 - val_loss: 0.7575 - val_acc: 0.7371
Epoch 19/30
40000/40000 [==============================] - 36s - loss: 0.7200 - acc: 0.7603 - val_loss: 0.7932 - val_acc: 0.7274
Epoch 20/30
40000/40000 [==============================] - 36s - loss: 0.6837 - acc: 0.7709 - val_loss: 0.8043 - val_acc: 0.7187
Epoch 21/30
40000/40000 [==============================] - 37s - loss: 0.6681 - acc: 0.7753 - val_loss: 0.7824 - val_acc: 0.7281
Epoch 22/30
40000/40000 [==============================] - 36s - loss: 0.6470 - acc: 0.7842 - val_loss: 0.7705 - val_acc: 0.7328
Epoch 23/30
40000/40000 [==============================] - 36s - loss: 0.6189 - acc: 0.7942 - val_loss: 0.7778 - val_acc: 0.7305
Epoch 24/30
40000/40000 [==============================] - 36s - loss: 0.5984 - acc: 0.8007 - val_loss: 0.7774 - val_acc: 0.7318
Epoch 25/30
40000/40000 [==============================] - 36s - loss: 0.5820 - acc: 0.8047 - val_loss: 0.7791 - val_acc: 0.7369
Epoch 26/30
40000/40000 [==============================] - 36s - loss: 0.5615 - acc: 0.8123 - val_loss: 0.7740 - val_acc: 0.7331
Epoch 27/30
40000/40000 [==============================] - 36s - loss: 0.5405 - acc: 0.8192 - val_loss: 0.7972 - val_acc: 0.7285
Epoch 28/30
40000/40000 [==============================] - 36s - loss: 0.5150 - acc: 0.8278 - val_loss: 0.7711 - val_acc: 0.7422
Epoch 29/30
40000/40000 [==============================] - 37s - loss: 0.5012 - acc: 0.8328 - val_loss: 0.7863 - val_acc: 0.7403
Epoch 30/30
40000/40000 [==============================] - 36s - loss: 0.4862 - acc: 0.8370 - val_loss: 0.8774 - val_acc: 0.7160

real	18m35.230s
user	14m49.580s
sys	2m20.844s