# Copy of Copy of Capstone Project

September 3, 2021

# 1 Capstone Project

## 1.1 Image classifier for the SVHN dataset

### 1.1.1 Instructions

In this notebook, you will create a neural network that classifies real-world images digits. You will use concepts from throughout this course in building, training, testing, validating and saving your Tensorflow classifier model.

This project is peer-assessed. Within this notebook you will find instructions in each section for how to complete the project. Pay close attention to the instructions as the peer review will be carried out according to a grading rubric that checks key parts of the project instructions. Feel free to add extra cells into the notebook as required.

### 1.1.2 How to submit

When you have completed the Capstone project notebook, you will submit a pdf of the notebook for peer review. First ensure that the notebook has been fully executed from beginning to end, and all of the cell outputs are visible. This is important, as the grading rubric depends on the reviewer being able to view the outputs of your notebook. Save the notebook as a pdf (you could download the notebook with File -> Download .ipynb, open the notebook locally, and then File -> Download as -> PDF via LaTeX), and then submit this pdf for review.

### 1.1.3 Let's get started!

We'll start by running some imports, and loading the dataset. For this project you are free to make further imports throughout the notebook as you wish.

```
In [1]: import tensorflow as tf
        from scipy.io import loadmat

        from tensorflow.keras.preprocessing.image import load_img, img_to_array
        from tensorflow.keras.models import Sequential, load_model
        from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
        from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
        from tensorflow.keras.models import load_model
        import numpy as np
        import matplotlib.pyplot as plt
```

```
import random

from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/co
```

For the capstone project, you will use the SVHN dataset. This is an image dataset of over 600,000 digit images in all, and is a harder dataset than MNIST as the numbers appear in the context of natural scene images. SVHN is obtained from house numbers in Google Street View images.

- Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng. "Reading Digits in Natural Images with Unsupervised Feature Learning". NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2011.

The train and test datasets required for this project can be downloaded from here and here. Once unzipped, you will have two files: `train_32x32.mat` and `test_32x32.mat`. You should store these files in Drive for use in this Colab notebook.

Your goal is to develop an end-to-end workflow for building, training, validating, evaluating and saving a neural network that classifies a real-world image into one of ten classes.

```
In [2]: # Load the dataset from your Drive folder

        train = loadmat('/content/drive/MyDrive/Colab Notebooks/train_32x32.mat')
        test = loadmat('/content/drive/MyDrive/Colab Notebooks/test_32x32.mat')
```

Both `train` and `test` are dictionaries with keys `X` and `y` for the input images and labels respectively.

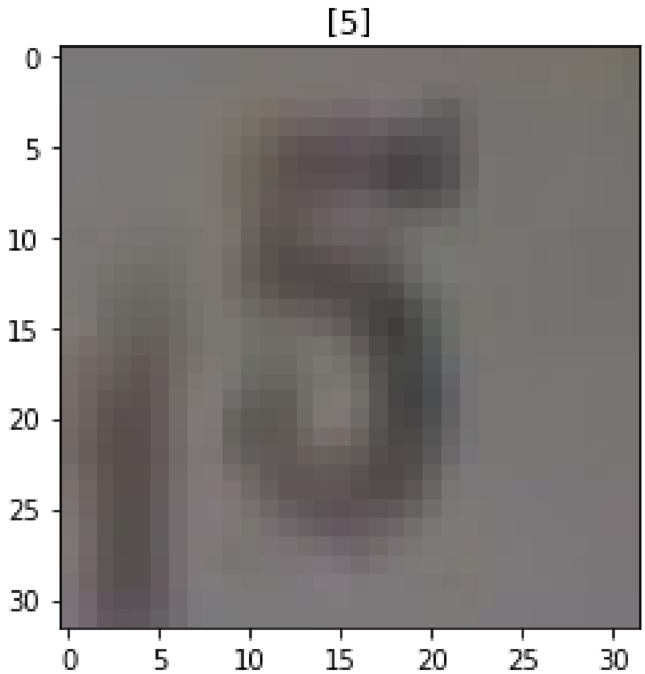
## 1.2 1. Inspect and preprocess the dataset

- Extract the training and testing images and labels separately from the train and test dictionaries loaded for you.
- Select a random sample of images and corresponding labels from the dataset (at least 10), and display them in a figure.
- Convert the training and test images to grayscale by taking the average across all colour channels for each pixel. *Hint: retain the channel dimension, which will now have size 1.*
- Select a random sample of the grayscale images and corresponding labels from the dataset (at least 10), and display them in a figure.

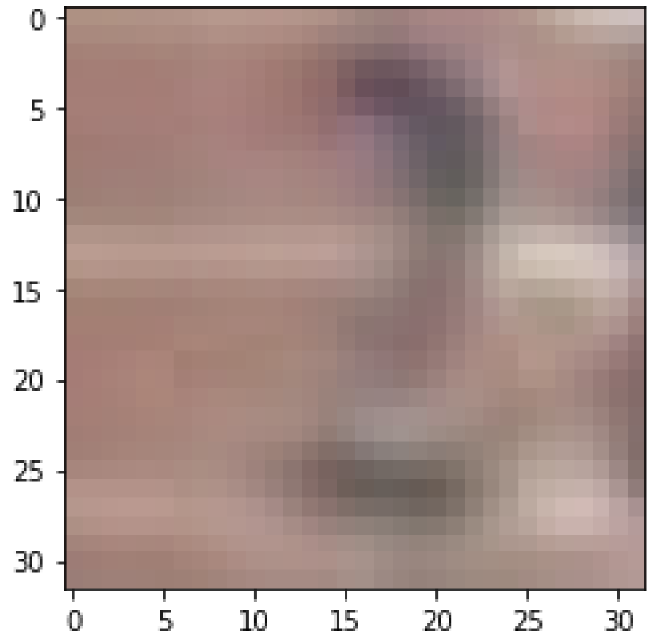
```
In [3]: training_x = train['X']
        training_y = train['y']
        testing_x = test['X']
        testing_y = test['y']
        for i in range(0, len(training_y)):
            if training_y[i]  == 10:
                training_y[i] = 0
```

```
for i in range(0, len(testing_y)):
    if testing_y[i]  == 10:
        testing_y[i] = 0
```

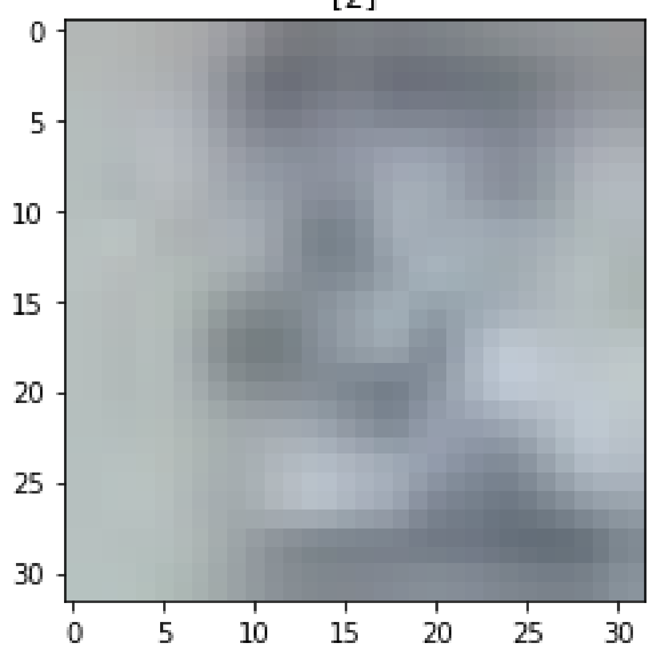
```
In [4]: for j in range(0,10):
            i = random.randint(0,training_x.shape[3])
            plt.imshow(training_x[:,:,:,i])
            plt.title(str(training_y[i]))
            plt.show()
```

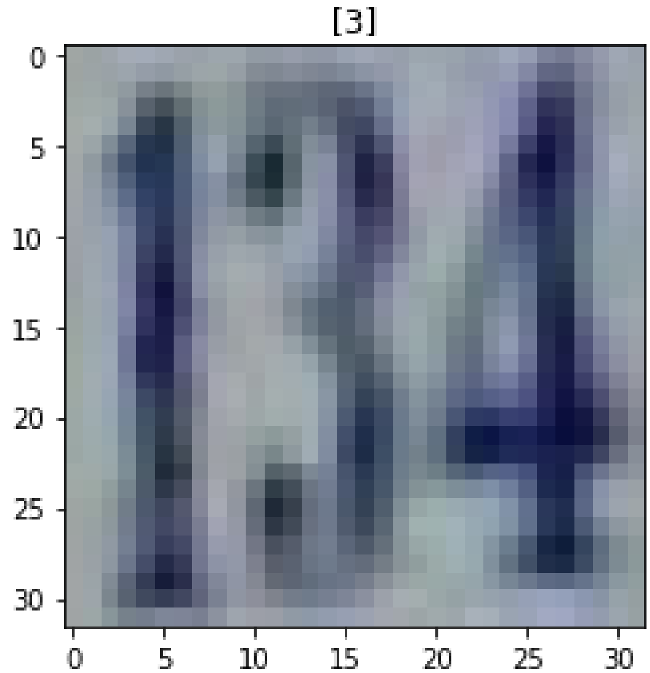


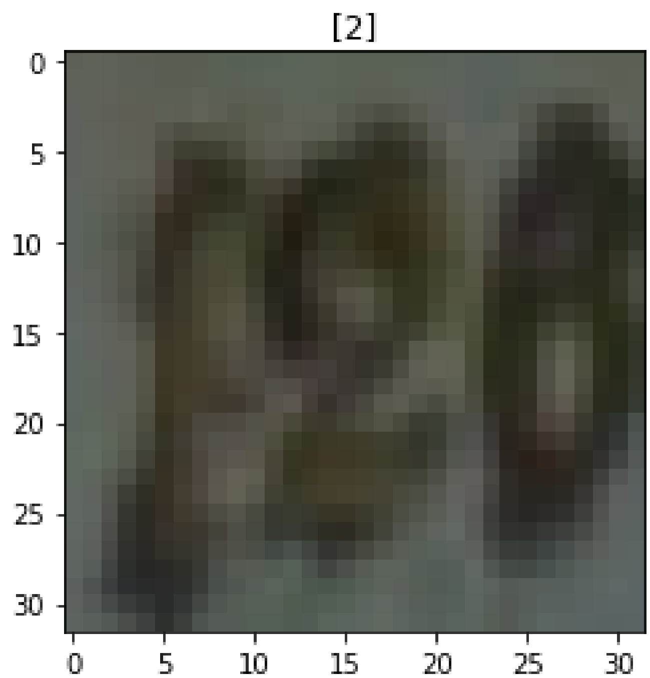
[2]

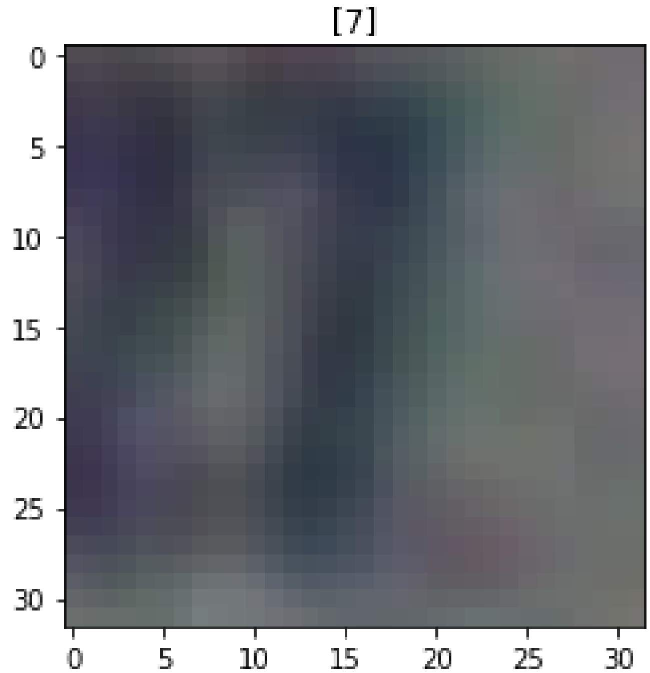


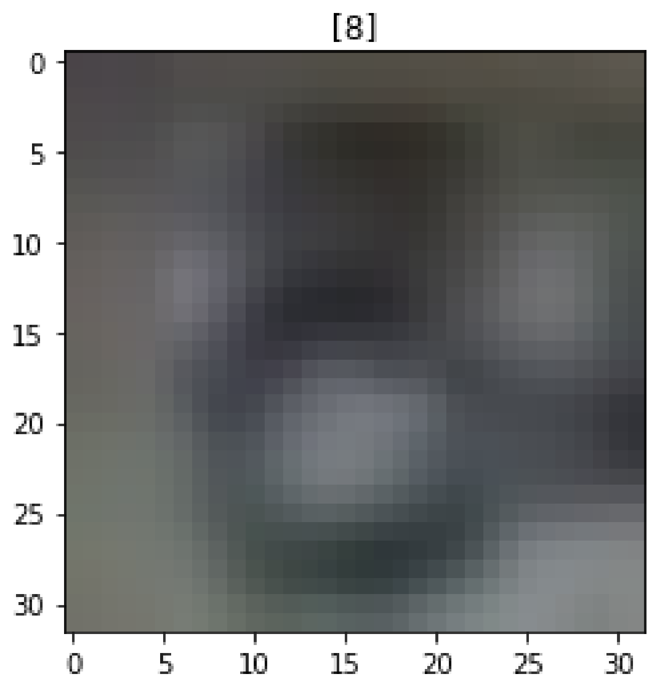

[2]

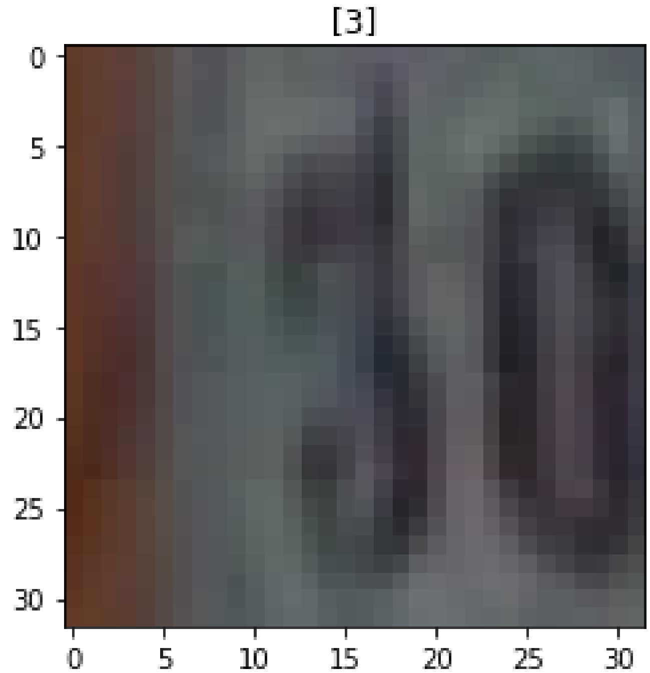


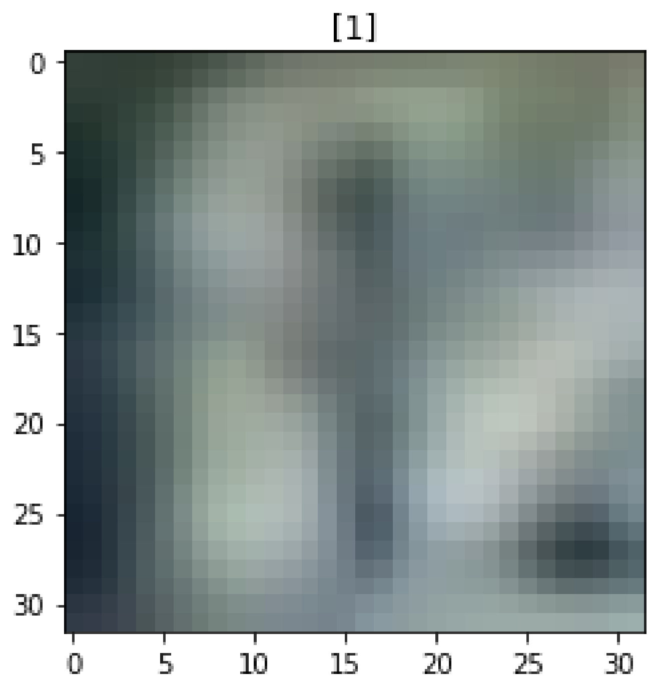

[3]
0
5
10
15
20
25
30
0
5
10
15
20
25
30

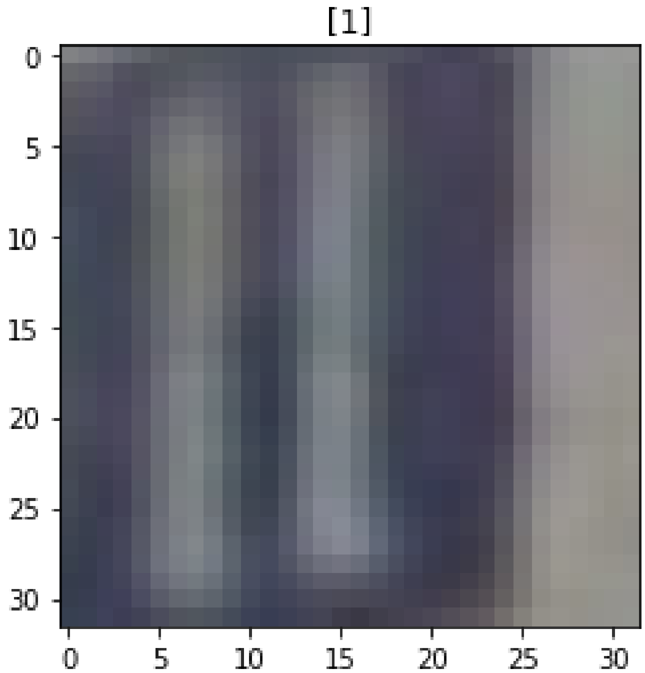

[2]
0
5
10
15
20
25
30
0
5
10
15
20
25
30


[7]
0 5 10 15 20 25 30
0 5 10 15 20 25 30


[8]
0 5 10 15 20 25 30
0 5 10 15 20 25 30

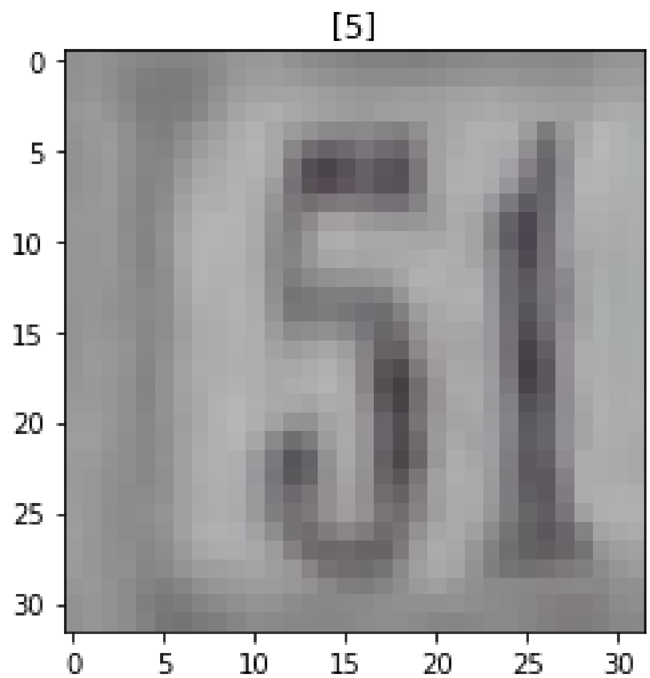

[3]
0
5
10
15
20
25
30
0
5
10
15
20
25
30

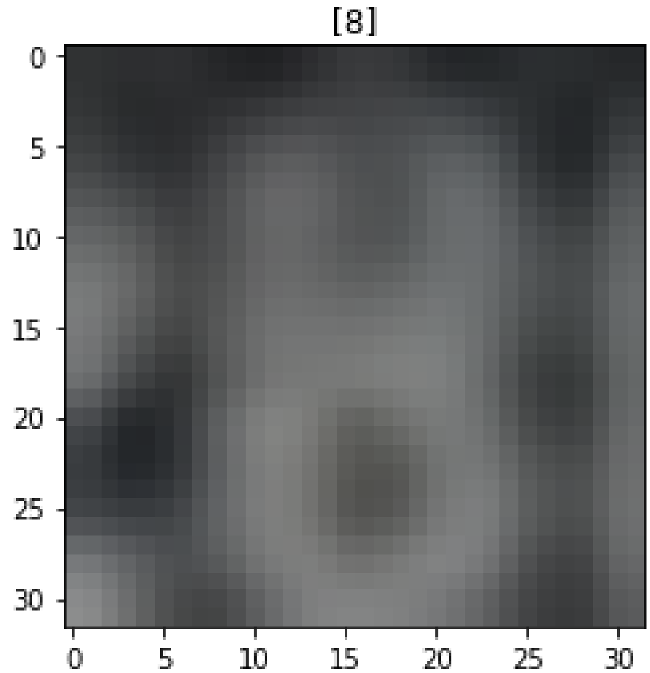

[1]
0
5
10
15
20
25
30
0
5
10
15
20
25
30

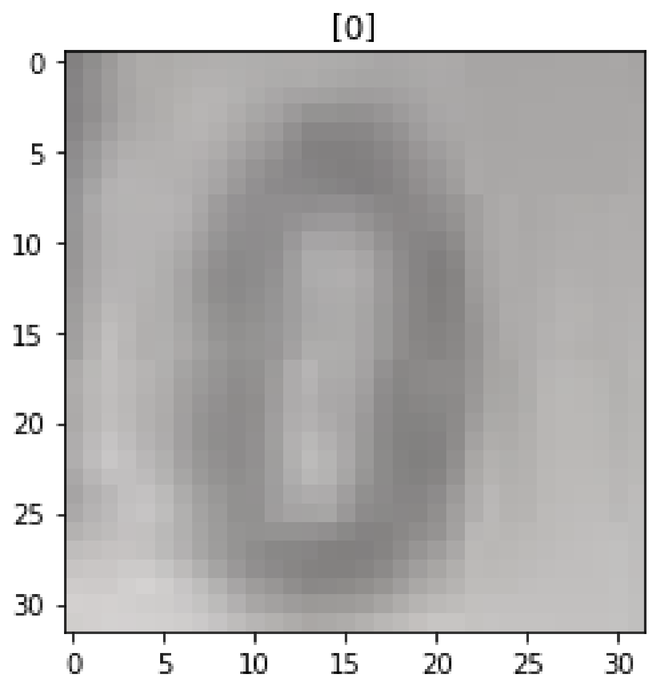


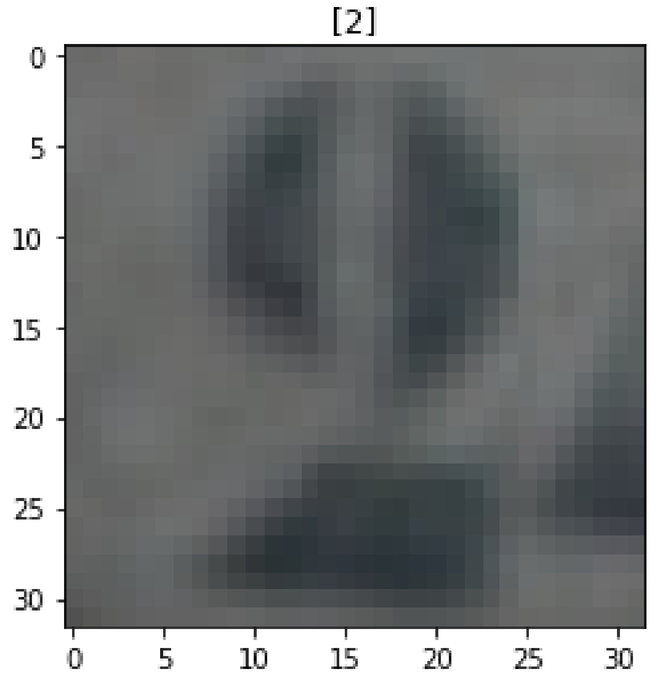
```
In [5]: # print(training_x.shape)
        for i in range(0,training_x.shape[3]):
            training_x[:,:,0,i] = np.mean(training_x[:,:,:,i], axis=2)
            training_x[:,:,1,i] = np.mean(training_x[:,:,:,i], axis=2)
            training_x[:,:,2,i] = np.mean(training_x[:,:,:,i], axis=2)

        for i in range(0,testing_x.shape[3]):
            testing_x[:,:,0,i] = np.mean(testing_x[:,:,:,i], axis=2)
            testing_x[:,:,1,i] = np.mean(testing_x[:,:,:,i], axis=2)
            testing_x[:,:,2,i] = np.mean(testing_x[:,:,:,i], axis=2)

        for j in range(0,10):
            i = random.randint(0,training_x.shape[3])
            plt.imshow(training_x[:,:,:,i])
            plt.title(str(training_y[i]))
            plt.show()

        xx = np.concatenate((training_x,testing_x),axis = 3)
        yy = np.concatenate((training_y,testing_y),axis = 0)

        xx=np.swapaxes(xx,3,0)
        xx=np.swapaxes(xx,3,1)
        xx=np.swapaxes(xx,3,2)

        testing_x=np.swapaxes(testing_x,3,0)
```

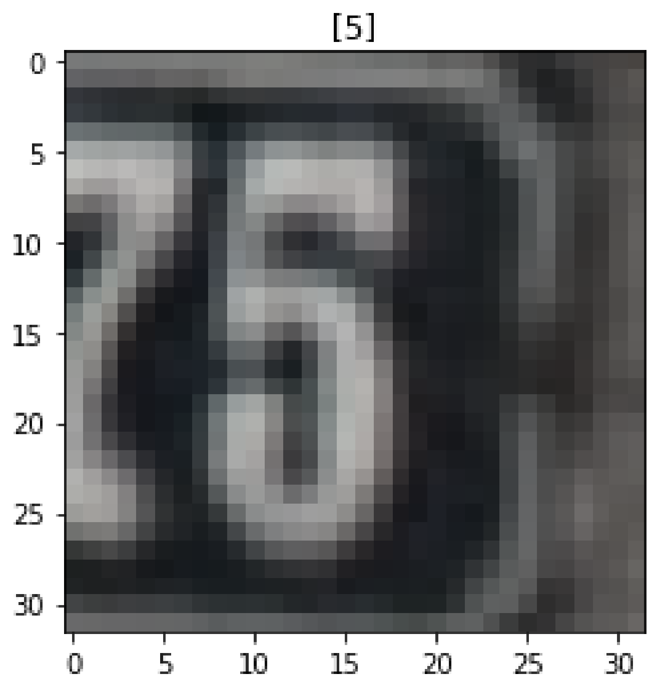
```
testing_x=np.swapaxes(testing_x,3,1)
testing_x=np.swapaxes(testing_x,3,2)

training_x=np.swapaxes(training_x,3,0)
training_x=np.swapaxes(training_x,3,1)
training_x=np.swapaxes(training_x,3,2)

MLPtesting_x=testing_x[:,:,:,0]
MLPtraining_x=training_x[:,:,:,0]
MLPxx = xx[:,:,:,0]
```

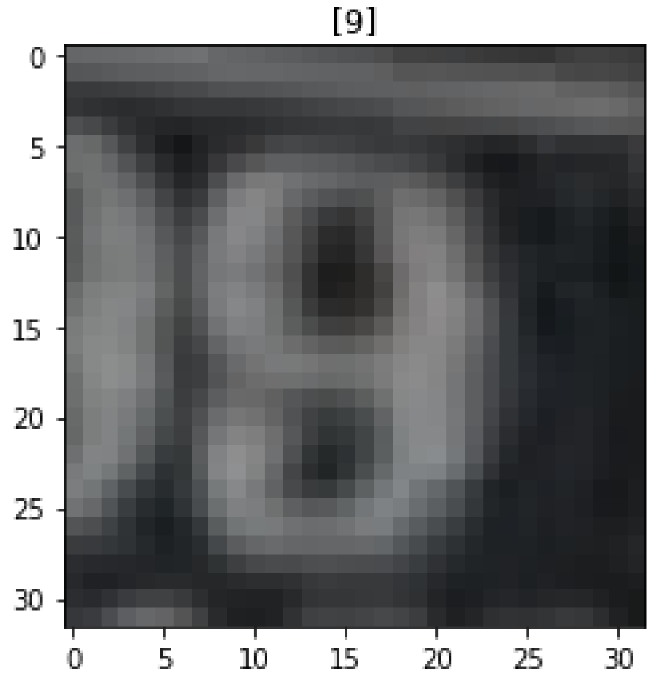


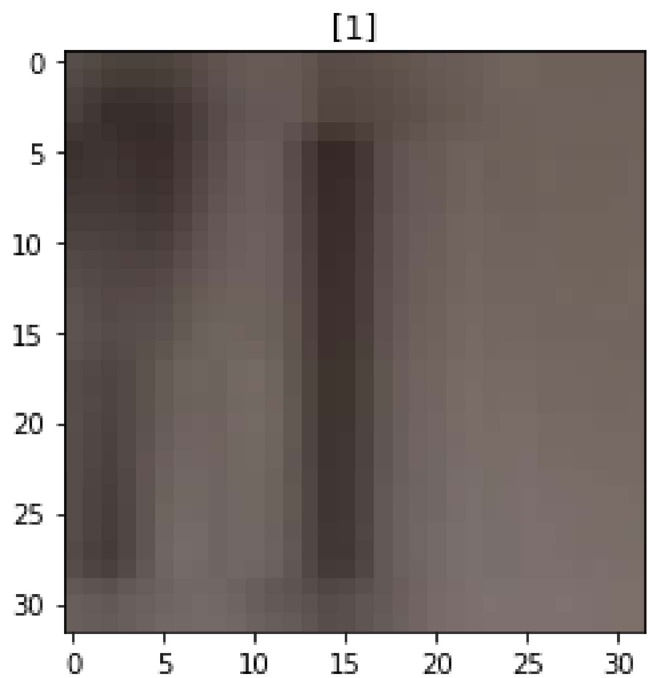

[8]

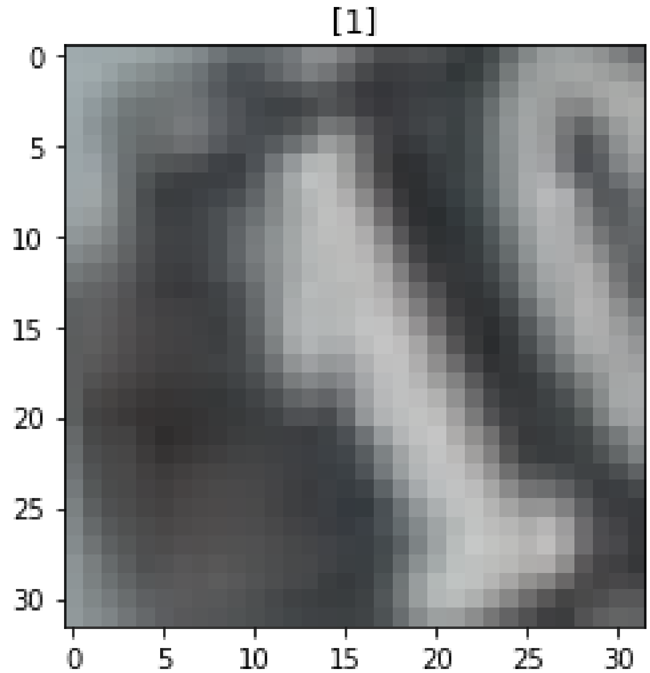

[0]

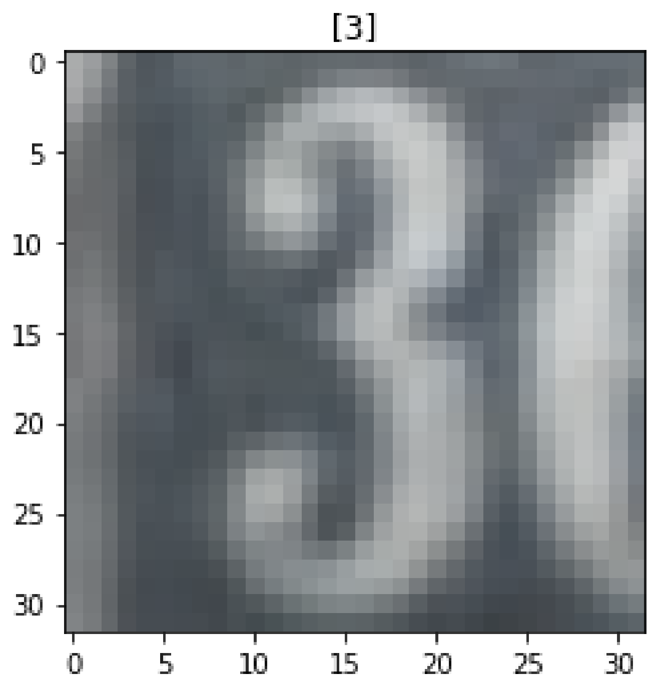

[2]
0
5
10
15
20
25
30
0
5
10
15
20
25
30

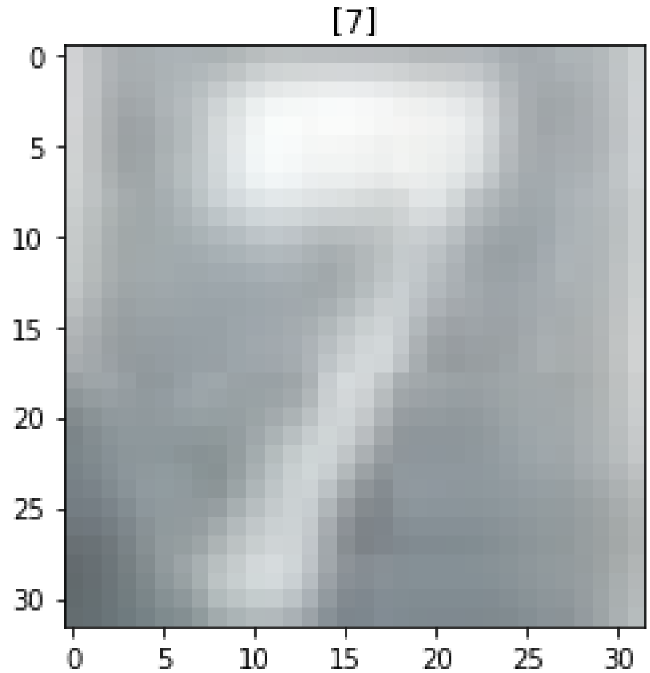

[5]
0
5
10
15
20
25
30
0
5
10
15
20
25
30


[9]
0
5
10
15
20
25
30
0
5
10
15
20
25
30


[1]
0
5
10
15
20
25
30
0
5
10
15
20
25
30


[1]
0
5
10
15
20
25
30


[3]
0
5
10
15
20
25
30




### 1.3 2. MLP neural network classifier

- Build an MLP classifier model using the Sequential API. Your model should use only Flatten and Dense layers, with the final layer having a 10-way softmax output.
- You should design and build the model yourself. Feel free to experiment with different MLP architectures. *Hint: to achieve a reasonable accuracy you won't need to use more than 4 or 5 layers.*
- Print out the model summary (using the summary() method)
- Compile and train the model (we recommend a maximum of 30 epochs), making use of both training and validation sets during the training run.
- Your model should track at least one appropriate metric, and use at least two callbacks during training, one of which should be a ModelCheckpoint callback.
- As a guide, you should aim to achieve a final categorical cross entropy training loss of less than 1.0 (the validation loss might be higher).
- Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and validation sets.
- Compute and display the loss and accuracy of the trained model on the test set.

In [6]:
```
# print(training_x.shape)

model = Sequential([
            Flatten(name='flatten',input_shape=(32,32)),
            Dense(256,activation = 'relu',name = 'dense_1'),
            Dense(128,activation = 'relu',name = 'dense_2'),
            Dense(90,activation = 'relu',name = 'dense_3'),
            Dense(10,activation = 'softmax',name = 'dense_5')
```

```
        ])

model.compile(
    optimizer = 'adam',
    loss = 'sparse_categorical_crossentropy',
    metrics = ['acc']
    )

Ckpt = tf.keras.callbacks.ModelCheckpoint(
    'checkpoints_best_only/checkpoint',
    monitor="val_acc",
    verbose=1,
    save_best_only=True,
    save_weights_only=True
    )

ES = tf.keras.callbacks.EarlyStopping(
    monitor="val_acc",
    patience=5,
    verbose=1,
    )

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               262400
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 90)                11610
_________________________________________________________________
dense_5 (Dense)              (None, 10)                910
=================================================================
Total params: 307,816
Trainable params: 307,816
Non-trainable params: 0
_________________________________________________________________
```

```
In [7]: history = model.fit(MLPtraining_x, training_y, epochs=30, validation_data=(MLPxx, yy),
```

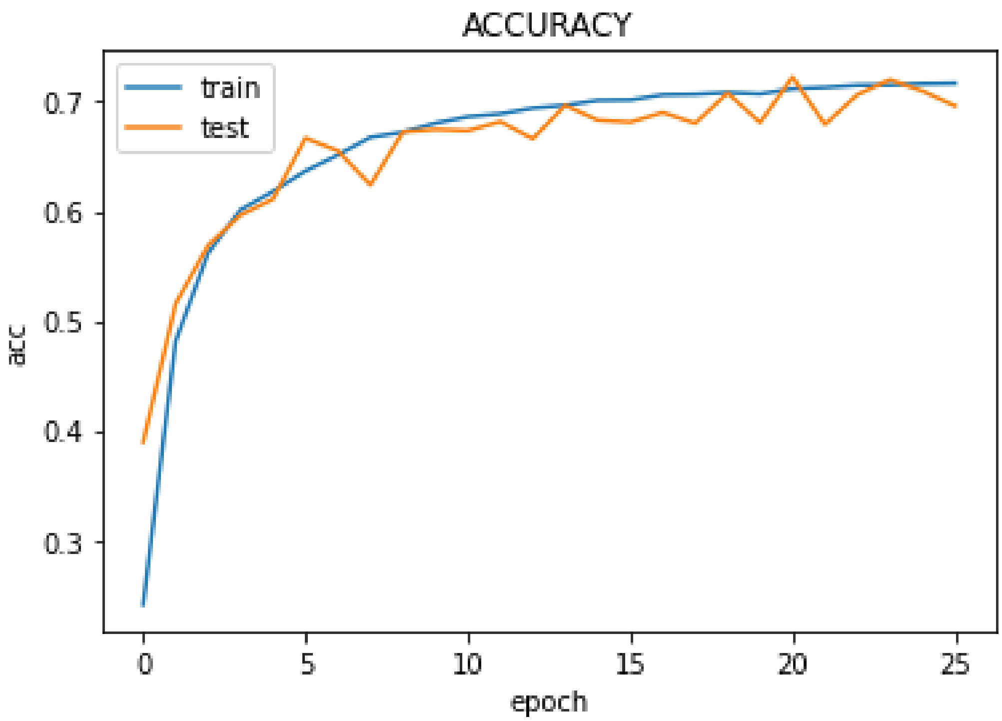
```
Epoch 1/30
2290/2290 [==============================] - 15s 6ms/step - loss: 3.7526 - acc: 0.2410 - val_lo
```

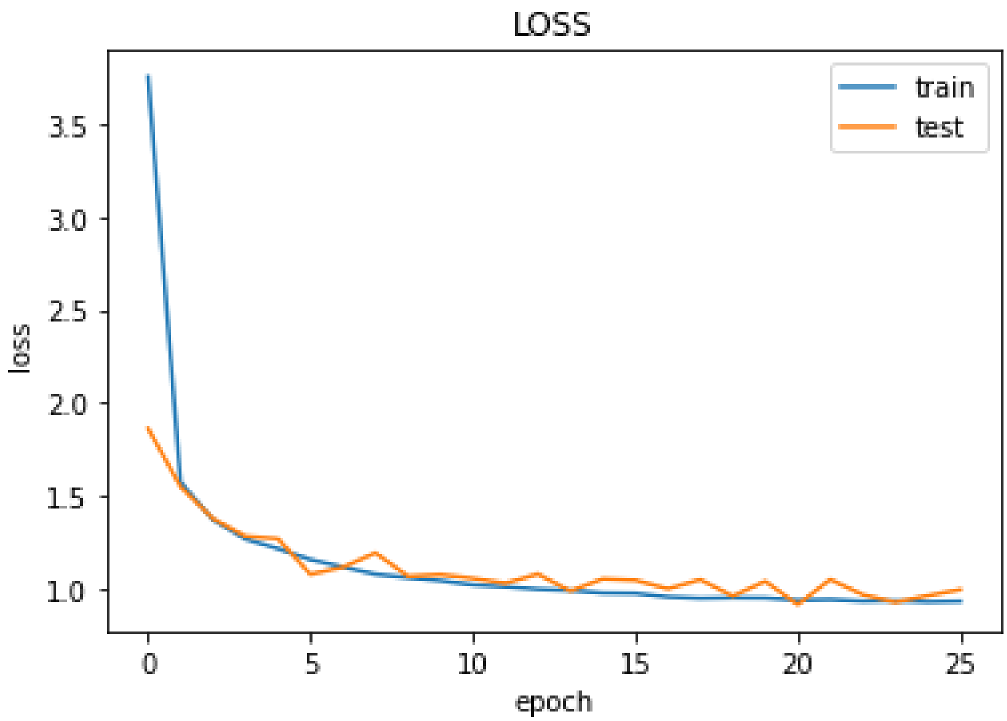
```
Epoch 00001: val_acc improved from -inf to 0.38927, saving model to checkpoints_best_only/checl
Epoch 2/30
2290/2290 [==============================] - 14s 6ms/step - loss: 1.5732 - acc: 0.4815 - val_lo

Epoch 00002: val_acc improved from 0.38927 to 0.51580, saving model to checkpoints_best_only/cl
Epoch 3/30
2290/2290 [==============================] - 14s 6ms/step - loss: 1.3710 - acc: 0.5619 - val_lo

Epoch 00003: val_acc improved from 0.51580 to 0.56890, saving model to checkpoints_best_only/cl
Epoch 4/30
2290/2290 [==============================] - 15s 6ms/step - loss: 1.2672 - acc: 0.6013 - val_lo

Epoch 00004: val_acc improved from 0.56890 to 0.59656, saving model to checkpoints_best_only/cl
Epoch 5/30
2290/2290 [==============================] - 15s 6ms/step - loss: 1.2158 - acc: 0.6182 - val_lo

Epoch 00005: val_acc improved from 0.59656 to 0.61116, saving model to checkpoints_best_only/cl
Epoch 6/30
2290/2290 [==============================] - 14s 6ms/step - loss: 1.1567 - acc: 0.6368 - val_lo

Epoch 00006: val_acc improved from 0.61116 to 0.66691, saving model to checkpoints_best_only/cl
Epoch 7/30
2290/2290 [==============================] - 15s 7ms/step - loss: 1.1170 - acc: 0.6515 - val_lo

Epoch 00007: val_acc did not improve from 0.66691
Epoch 8/30
2290/2290 [==============================] - 15s 7ms/step - loss: 1.0782 - acc: 0.6676 - val_lo

Epoch 00008: val_acc did not improve from 0.66691
Epoch 9/30
2290/2290 [==============================] - 15s 6ms/step - loss: 1.0603 - acc: 0.6724 - val_lo

Epoch 00009: val_acc improved from 0.66691 to 0.67261, saving model to checkpoints_best_only/cl
Epoch 10/30
2290/2290 [==============================] - 14s 6ms/step - loss: 1.0440 - acc: 0.6805 - val_lo

Epoch 00010: val_acc improved from 0.67261 to 0.67470, saving model to checkpoints_best_only/cl
Epoch 11/30
2290/2290 [==============================] - 14s 6ms/step - loss: 1.0217 - acc: 0.6864 - val_lo

Epoch 00011: val_acc did not improve from 0.67470
Epoch 12/30
2290/2290 [==============================] - 14s 6ms/step - loss: 1.0115 - acc: 0.6890 - val_lo

Epoch 00012: val_acc improved from 0.67470 to 0.68190, saving model to checkpoints_best_only/cl
Epoch 13/30
2290/2290 [==============================] - 14s 6ms/step - loss: 0.9981 - acc: 0.6942 - val_lo
```

```
Epoch 00013: val_acc did not improve from 0.68190
Epoch 14/30
2290/2290 [==============================] - 14s 6ms/step - loss: 0.9917 - acc: 0.6968 - val_lo

Epoch 00014: val_acc improved from 0.68190 to 0.69677, saving model to checkpoints_best_only/ch
Epoch 15/30
2290/2290 [==============================] - 14s 6ms/step - loss: 0.9777 - acc: 0.7013 - val_lo

Epoch 00015: val_acc did not improve from 0.69677
Epoch 16/30
2290/2290 [==============================] - 14s 6ms/step - loss: 0.9760 - acc: 0.7017 - val_lo

Epoch 00016: val_acc did not improve from 0.69677
Epoch 17/30
2290/2290 [==============================] - 15s 6ms/step - loss: 0.9563 - acc: 0.7064 - val_lo

Epoch 00017: val_acc did not improve from 0.69677
Epoch 18/30
2290/2290 [==============================] - 14s 6ms/step - loss: 0.9495 - acc: 0.7069 - val_lo

Epoch 00018: val_acc did not improve from 0.69677
Epoch 19/30
2290/2290 [==============================] - 14s 6ms/step - loss: 0.9521 - acc: 0.7086 - val_lo

Epoch 00019: val_acc improved from 0.69677 to 0.70810, saving model to checkpoints_best_only/ch
Epoch 20/30
2290/2290 [==============================] - 15s 6ms/step - loss: 0.9510 - acc: 0.7073 - val_lo

Epoch 00020: val_acc did not improve from 0.70810
Epoch 21/30
2290/2290 [==============================] - 15s 6ms/step - loss: 0.9396 - acc: 0.7119 - val_lo

Epoch 00021: val_acc improved from 0.70810 to 0.72255, saving model to checkpoints_best_only/ch
Epoch 22/30
2290/2290 [==============================] - 15s 6ms/step - loss: 0.9428 - acc: 0.7132 - val_lo

Epoch 00022: val_acc did not improve from 0.72255
Epoch 23/30
2290/2290 [==============================] - 14s 6ms/step - loss: 0.9339 - acc: 0.7151 - val_lo

Epoch 00023: val_acc did not improve from 0.72255
Epoch 24/30
2290/2290 [==============================] - 15s 6ms/step - loss: 0.9362 - acc: 0.7154 - val_lo

Epoch 00024: val_acc did not improve from 0.72255
Epoch 25/30
2290/2290 [==============================] - 14s 6ms/step - loss: 0.9307 - acc: 0.7164 - val_lo
```

```
Epoch 00025: val_acc did not improve from 0.72255
Epoch 26/30
2290/2290 [==============================] - 14s 6ms/step - loss: 0.9333 - acc: 0.7168 - val_lo

Epoch 00026: val_acc did not improve from 0.72255
Epoch 00026: early stopping
```

```
In [8]: plt.plot(history.history['acc'])
        plt.plot(history.history['val_acc'])
        plt.title('ACCURACY')
        plt.ylabel('acc')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'])
        plt.show()

        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title('LOSS')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'])
        plt.show()
```






```
In [9]: test_loss, test_acc = model.evaluate(x=MLPtesting_x, y=testing_y, verbose=0)
        print('test_loss',test_loss)
        print('test_acc',test_acc)
```

```
test_loss 1.111208438873291
test_acc 0.673401951789856
```

### 1.4 3. CNN neural network classifier

- Build a CNN classifier model using the Sequential API. Your model should use the Conv2D, MaxPool2D, BatchNormalization, Flatten, Dense and Dropout layers. The final layer should again have a 10-way softmax output.
- You should design and build the model yourself. Feel free to experiment with different CNN architectures. *Hint: to achieve a reasonable accuracy you won't need to use more than 2 or 3 convolutional layers and 2 fully connected layers.)*
- The CNN model should use fewer trainable parameters than your MLP model.
- Compile and train the model (we recommend a maximum of 30 epochs), making use of both training and validation sets during the training run.
- Your model should track at least one appropriate metric, and use at least two callbacks during training, one of which should be a ModelCheckpoint callback.
- You should aim to beat the MLP model performance with fewer parameters!
- Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and validation sets.

- Compute and display the loss and accuracy of the trained model on the test set.

```
In [10]: from tensorflow.keras.layers import BatchNormalization,Dropout
         modelCNN = Sequential([
             Conv2D(16,3,activation = 'relu', padding = 'same',name ='conv_1',input_shape=(32,3
             Conv2D(8,(3,3),activation = 'relu', padding = 'same',name ='conv_2'),
             MaxPooling2D((8,8),name = 'pool_1'),
             Flatten(name='flatten'),
             BatchNormalization(name = 'norm'),
             Dense(64,activation = 'relu',name = 'dense_1'),
             Dropout(0.3),
             Dense(10,activation = 'softmax',name = 'dense_2')
         ])
         modelCNN.compile(
             optimizer = 'adam',
             loss = 'sparse_categorical_crossentropy',
             metrics = ['acc']
         )
         modelCNN.summary()
         CkptCNN = tf.keras.callbacks.ModelCheckpoint(
             'checkpoints_best_onlyCNN/checkpointCNN',
             monitor="val_acc",
             verbose=1,
             save_best_only=True,
             save_weights_only=True
             )
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_1 (Conv2D)              (None, 32, 32, 16)        448
_________________________________________________________________
conv_2 (Conv2D)              (None, 32, 32, 8)         1160
_________________________________________________________________
pool_1 (MaxPooling2D)        (None, 4, 4, 8)           0
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
norm (BatchNormalization)    (None, 128)               512
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 11,026
```

```
Trainable params: 10,770
Non-trainable params: 256
_________________________________________________________________
```

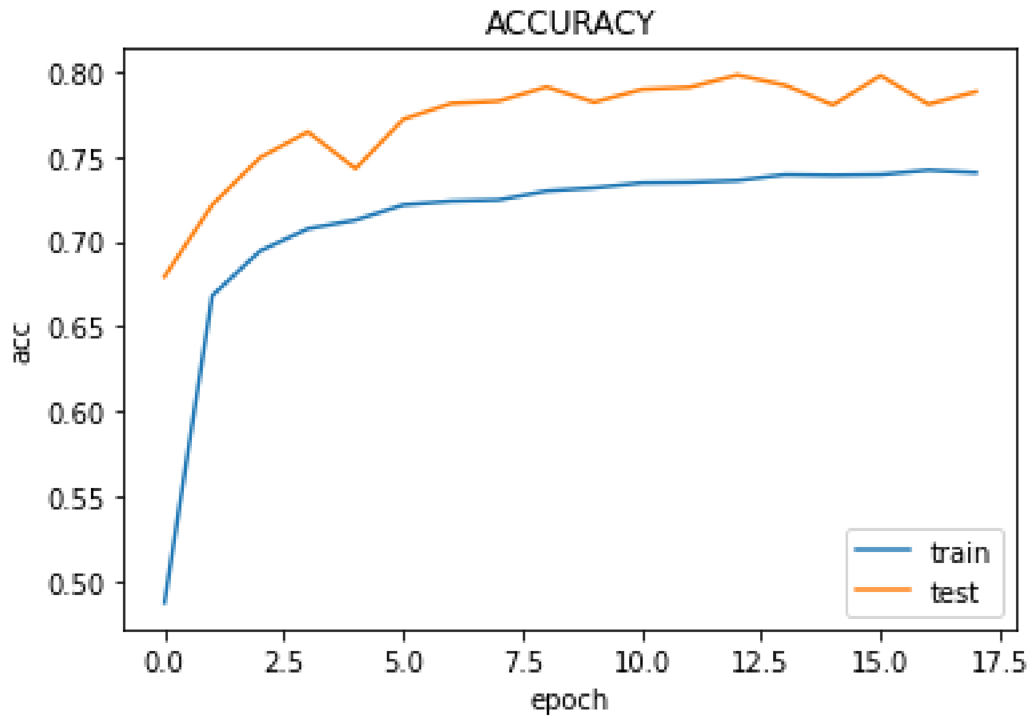
```
In [11]: historyCNN = modelCNN.fit(training_x, training_y, epochs=30, validation_data=(xx, yy)
```

```
Epoch 1/30
2290/2290 [==============================] - 125s 54ms/step - loss: 1.5055 - acc: 0.4873 - val_

Epoch 00001: val_acc improved from -inf to 0.67935, saving model to checkpoints_best_onlyCNN/cl
Epoch 2/30
2290/2290 [==============================] - 124s 54ms/step - loss: 1.0301 - acc: 0.6682 - val_

Epoch 00002: val_acc improved from 0.67935 to 0.72158, saving model to checkpoints_best_onlyCNI
Epoch 3/30
2290/2290 [==============================] - 130s 57ms/step - loss: 0.9536 - acc: 0.6944 - val_

Epoch 00003: val_acc improved from 0.72158 to 0.74947, saving model to checkpoints_best_onlyCNI
Epoch 4/30
2290/2290 [==============================] - 125s 54ms/step - loss: 0.9171 - acc: 0.7075 - val_

Epoch 00004: val_acc improved from 0.74947 to 0.76464, saving model to checkpoints_best_onlyCNI
Epoch 5/30
2290/2290 [==============================] - 125s 55ms/step - loss: 0.8961 - acc: 0.7126 - val_

Epoch 00005: val_acc did not improve from 0.76464
Epoch 6/30
2290/2290 [==============================] - 126s 55ms/step - loss: 0.8732 - acc: 0.7216 - val_

Epoch 00006: val_acc improved from 0.76464 to 0.77204, saving model to checkpoints_best_onlyCNI
Epoch 7/30
2290/2290 [==============================] - 126s 55ms/step - loss: 0.8666 - acc: 0.7237 - val_

Epoch 00007: val_acc improved from 0.77204 to 0.78137, saving model to checkpoints_best_onlyCNI
Epoch 8/30
2290/2290 [==============================] - 125s 55ms/step - loss: 0.8571 - acc: 0.7245 - val_

Epoch 00008: val_acc improved from 0.78137 to 0.78278, saving model to checkpoints_best_onlyCNI
Epoch 9/30
2290/2290 [==============================] - 124s 54ms/step - loss: 0.8462 - acc: 0.7297 - val_

Epoch 00009: val_acc improved from 0.78278 to 0.79108, saving model to checkpoints_best_onlyCNI
Epoch 10/30
2290/2290 [==============================] - 124s 54ms/step - loss: 0.8426 - acc: 0.7316 - val_

Epoch 00010: val_acc did not improve from 0.79108
Epoch 11/30
```

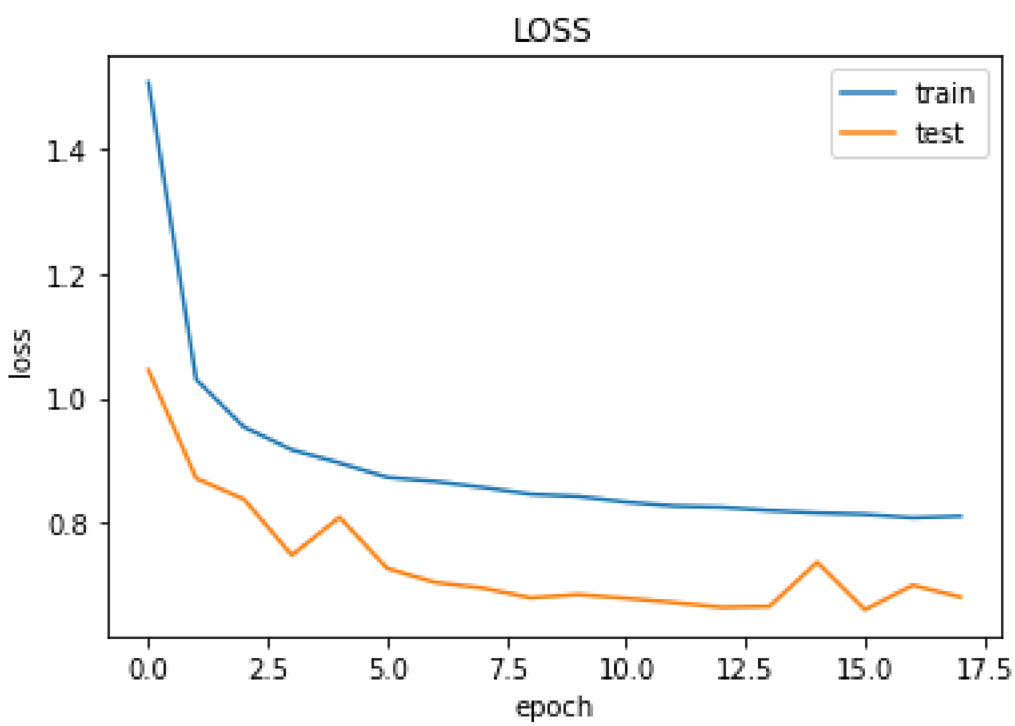
```
2290/2290 [==============================] - 125s 55ms/step - loss: 0.8335 - acc: 0.7345 - val_

Epoch 00011: val_acc did not improve from 0.79108
Epoch 12/30
2290/2290 [==============================] - 126s 55ms/step - loss: 0.8272 - acc: 0.7349 - val_

Epoch 00012: val_acc did not improve from 0.79108
Epoch 13/30
2290/2290 [==============================] - 126s 55ms/step - loss: 0.8252 - acc: 0.7359 - val_

Epoch 00013: val_acc improved from 0.79108 to 0.79804, saving model to checkpoints_best_onlyCNN
Epoch 14/30
2290/2290 [==============================] - 125s 55ms/step - loss: 0.8200 - acc: 0.7394 - val_

Epoch 00014: val_acc did not improve from 0.79804
Epoch 15/30
2290/2290 [==============================] - 125s 55ms/step - loss: 0.8162 - acc: 0.7390 - val_

Epoch 00015: val_acc did not improve from 0.79804
Epoch 16/30
2290/2290 [==============================] - 125s 55ms/step - loss: 0.8139 - acc: 0.7394 - val_

Epoch 00016: val_acc did not improve from 0.79804
Epoch 17/30
2290/2290 [==============================] - 125s 55ms/step - loss: 0.8084 - acc: 0.7421 - val_

Epoch 00017: val_acc did not improve from 0.79804
Epoch 18/30
2290/2290 [==============================] - 126s 55ms/step - loss: 0.8109 - acc: 0.7406 - val_

Epoch 00018: val_acc did not improve from 0.79804
Epoch 00018: early stopping
```

In [12]:
```
plt.plot(historyCNN.history['acc'])
plt.plot(historyCNN.history['val_acc'])
plt.title('ACCURACY')
plt.ylabel('acc')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()

plt.plot(historyCNN.history['loss'])
plt.plot(historyCNN.history['val_loss'])
plt.title('LOSS')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
```

```
plt.show()
```



ACCURACY
0.80
0.75
0.70
0.65
0.60
0.55
0.50
acc
train
test
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epoch

LOSS
train
test
1.4
1.2
1.0
0.8
loss
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epoch

```
In [13]: test_loss, test_acc = modelCNN.evaluate(x=testing_x, y=testing_y, verbose=0)
         print('test_loss',test_loss)
         print('test_acc',test_acc)
```

```
test_loss 0.7494969964027405
test_acc 0.7697833180427551
```

### 1.5 4. Get model predictions

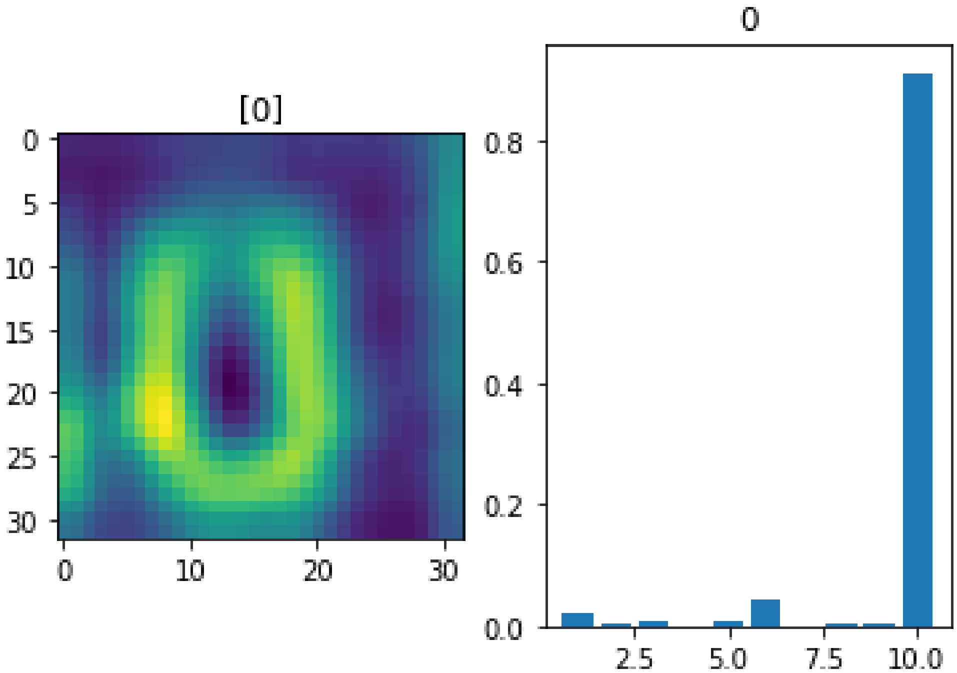
- Load the best weights for the MLP and CNN models that you saved during the training run.
- Randomly select 5 images and corresponding labels from the test set and display the images with their labels.
- Alongside the image and label, show each model's predictive distribution as a bar chart, and the final model prediction given by the label with maximum probability.

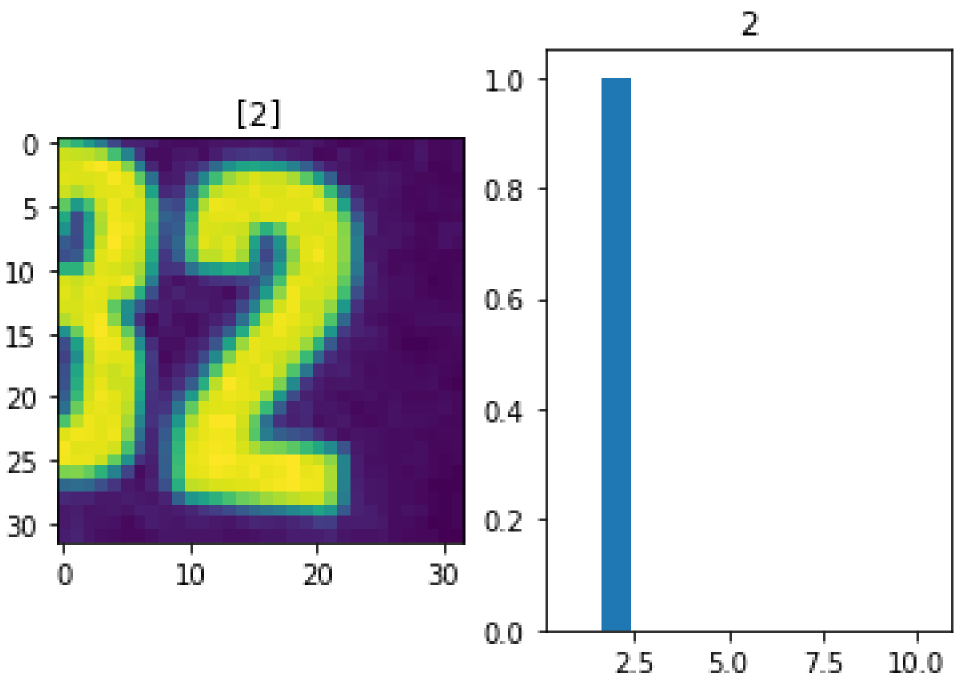
```
In [ ]: ckMLP = tf.train.latest_checkpoint('checkpoints_best_only')
        model.load_weights(ckMLP)
```

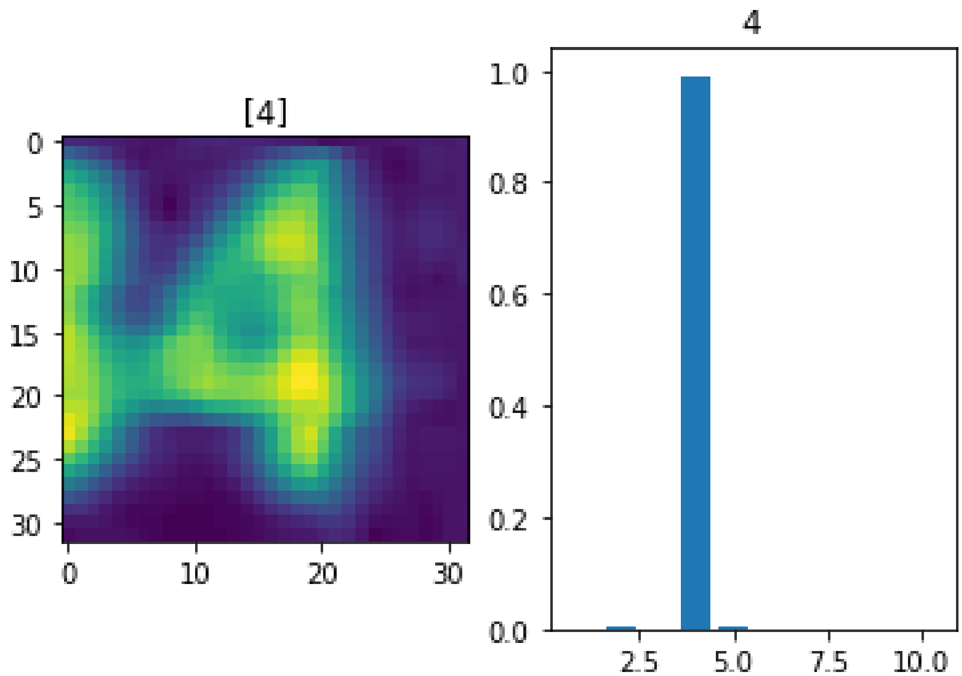
```
In [ ]: ckCNN = tf.train.latest_checkpoint('checkpoints_best_onlyCNN')
        modelCNN.load_weights(ckCNN)
```

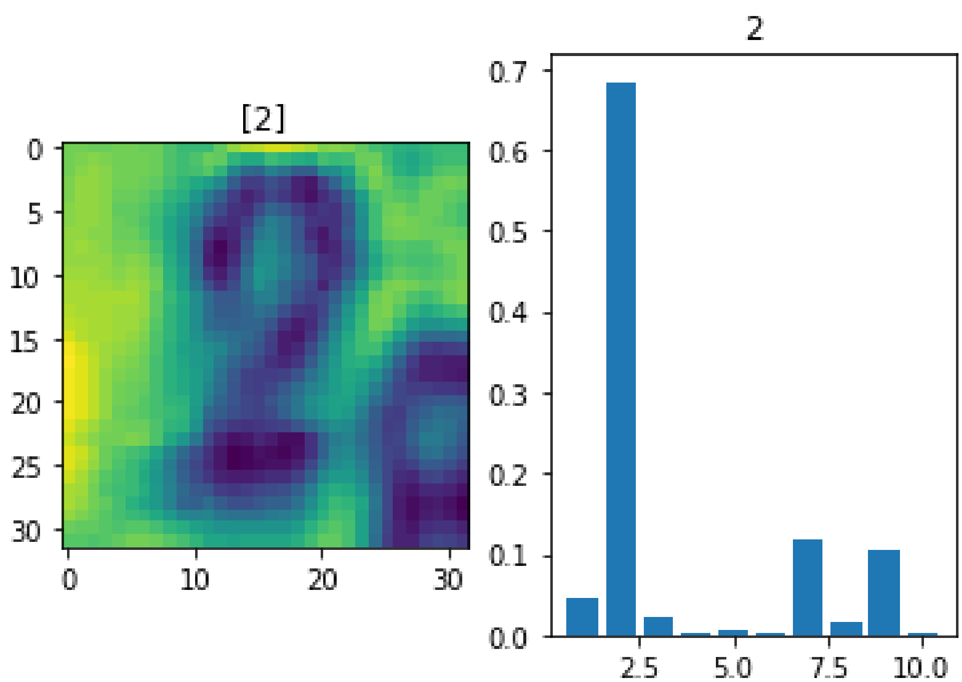
```
In [24]: ans = modelCNN.predict(testing_x)
         ans = model.predict(MLPtesting_x)
```

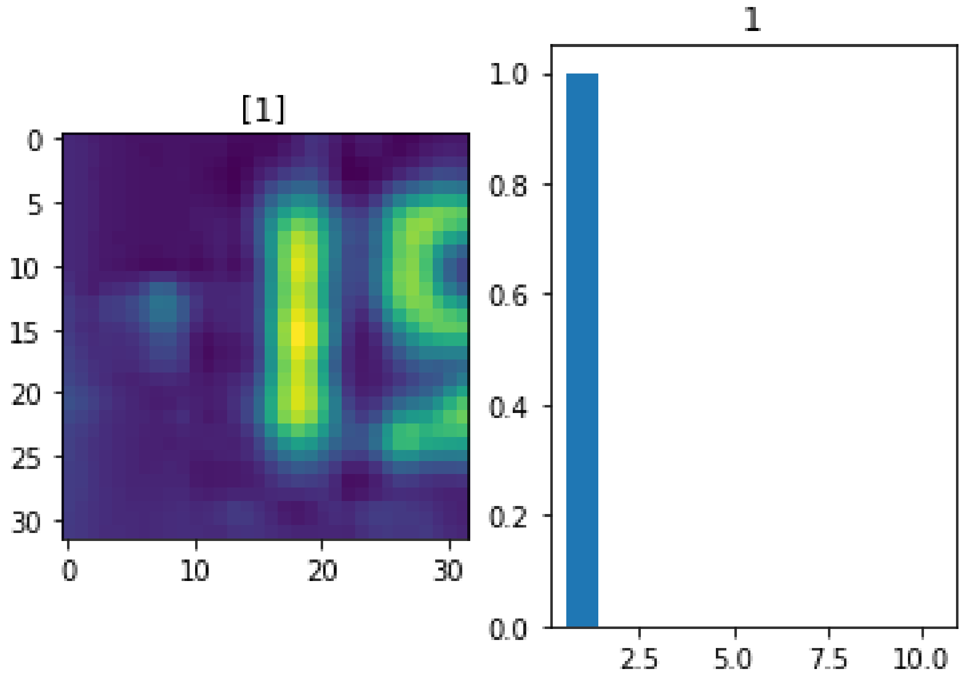
```
In [38]: for j in range(0,5):
             f, ax  = plt.subplots(1,2)

             i = random.randint(0,MLPtesting_x.shape[0])
             ax[0].imshow(MLPtesting_x[i])
             ax[0].set_title(str(testing_y[i]))

             ans = model.predict(np.array([MLPtesting_x[i]]))
             ax[1].bar([10,1,2,3,4,5,6,7,8,9],ans[0])
             ax[1].set_title(str(np.argmax(ans)))
             plt.show()
```


[0]
0
0.8
0.6
0.4
0.2
0.0
2.5
5.0
7.5
10.0
0
5
10
15
20
25
30

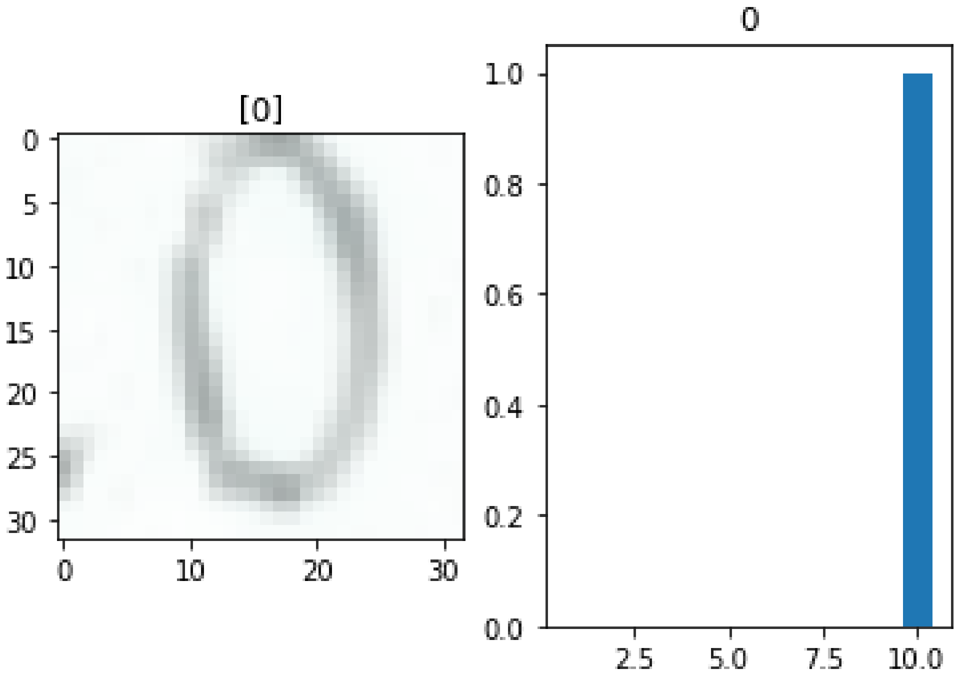

[2]
2
1.0
0.8
0.6
0.4
0.2
0.0
2.5
5.0
7.5
10.0
0
5
10
15
20
25
30

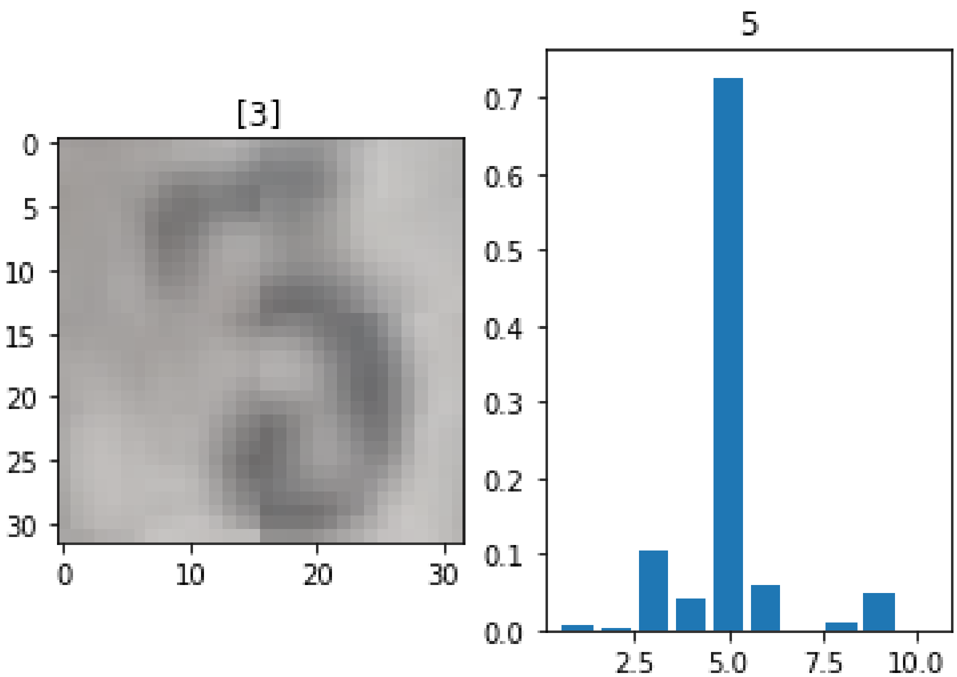

[4]
0
5
10
15
20
25
30
0
10
20
30
4
1.0
0.8
0.6
0.4
0.2
0.0
2.5
5.0
7.5
10.0

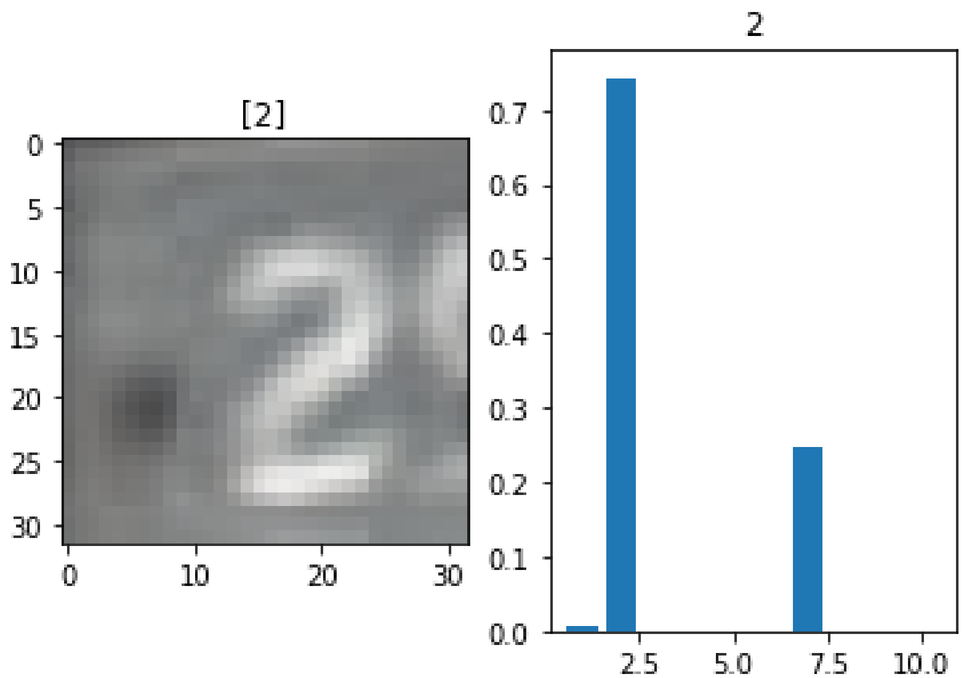

[2]
0
5
10
15
20
25
30
0
10
20
30
2
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2.5
5.0
7.5
10.0

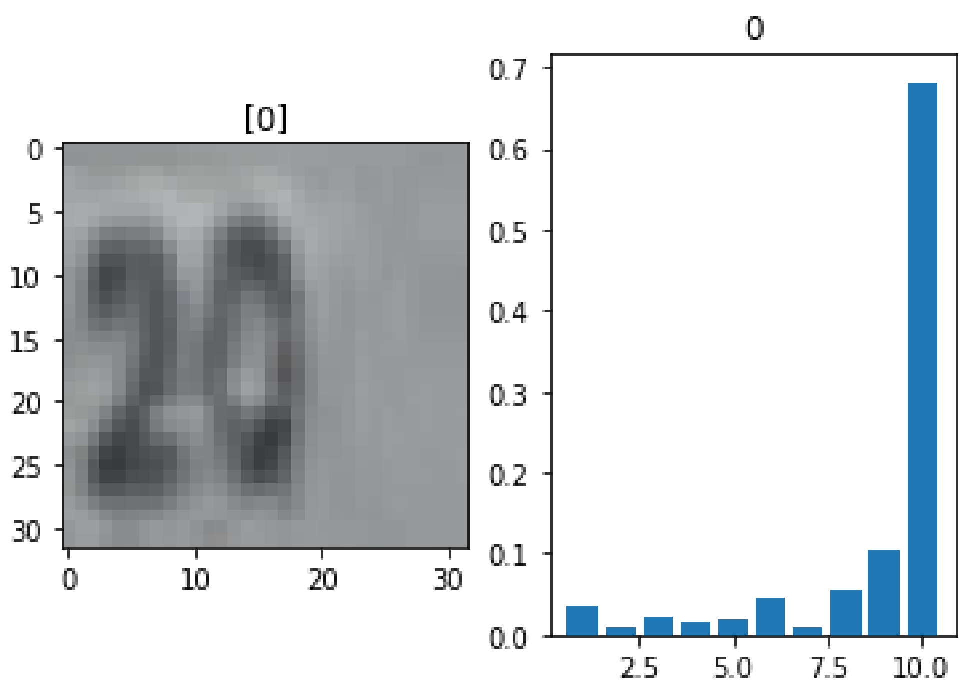


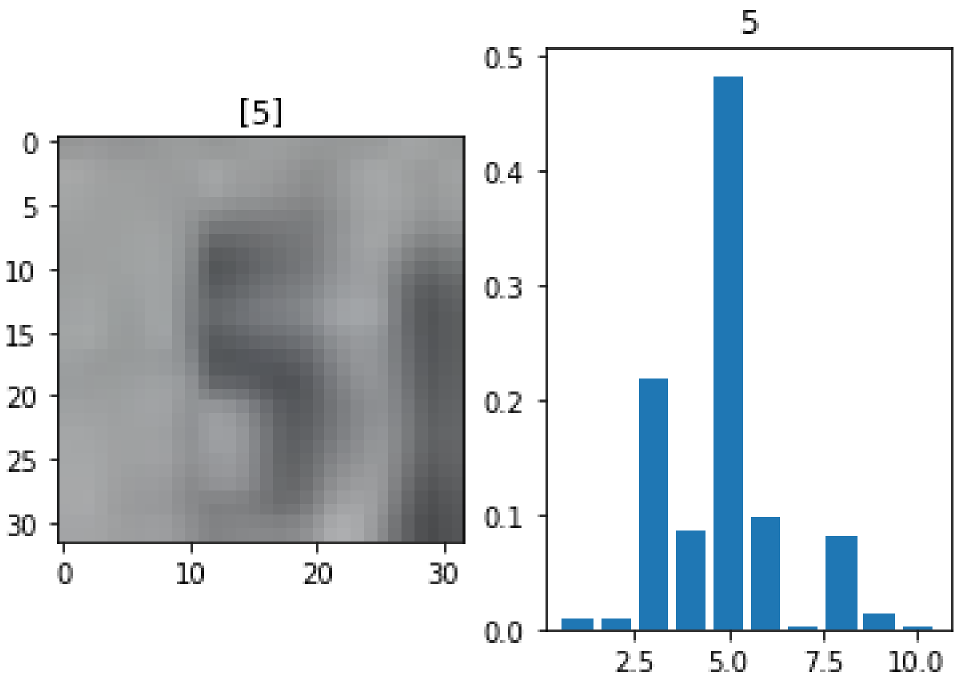

```
In [37]: for j in range(0,5):
             f, ax  = plt.subplots(1,2)

             i = random.randint(0,testing_x.shape[0])
             ax[0].imshow(testing_x[i])
             ax[0].set_title(str(testing_y[i]))

             ans = modelCNN.predict(np.array([testing_x[i]]))
             ax[1].bar([10,1,2,3,4,5,6,7,8,9],ans[0])
             ax[1].set_title(str(np.argmax(ans)))
             plt.show()
```


[0]
0
0 5 10 15 20 25 30
0 10 20 30
0.0 0.2 0.4 0.6 0.8 1.0
2.5 5.0 7.5 10.0


[3]
5
0 5 10 15 20 25 30
0 10 20 30
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7
2.5 5.0 7.5 10.0


[2]
2
0
5
10
15
20
25
30
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
2.5
5.0
7.5
10.0


[0]
0
0
5
10
15
20
25
30
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
2.5
5.0
7.5
10.0


[5]
5
0
5
10
15
20
25
30
0
10
20
30
0.5
0.4
0.3
0.2
0.1
0.0
2.5
5.0
7.5
10.0