Capstone Project

December 24, 2022

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 (File -> Download as -> PDF via LaTeX). You should 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
```

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.

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]: # Run this cell to load the dataset

        train = loadmat('data/train_32x32.mat')
        test = loadmat('data/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]: #Extract the training and testing images and labels separately from the train and test

        X_train = train['X']
```

```
        X_test = test['X']
        y_train = train['y']
        y_test = test['y']
```

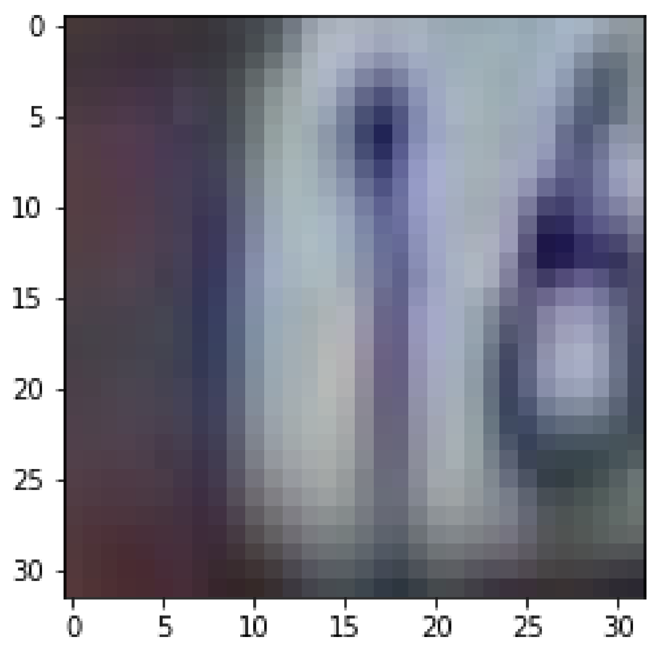
```
In [4]: X_train.shape,X_test.shape
```

```
Out[4]: ((32, 32, 3, 73257), (32, 32, 3, 26032))
```

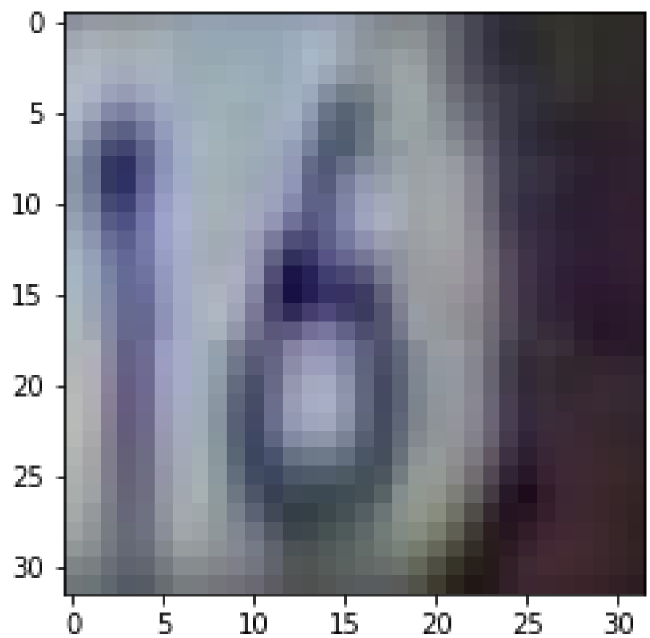
```
In [5]: import numpy as np

        X_train = np.moveaxis(X_train, -1, 0)
        X_test = np.moveaxis(X_test, -1 , 0)

        X_train.shape,X_test.shape
```

```
Out[5]: ((73257, 32, 32, 3), (26032, 32, 32, 3))
```

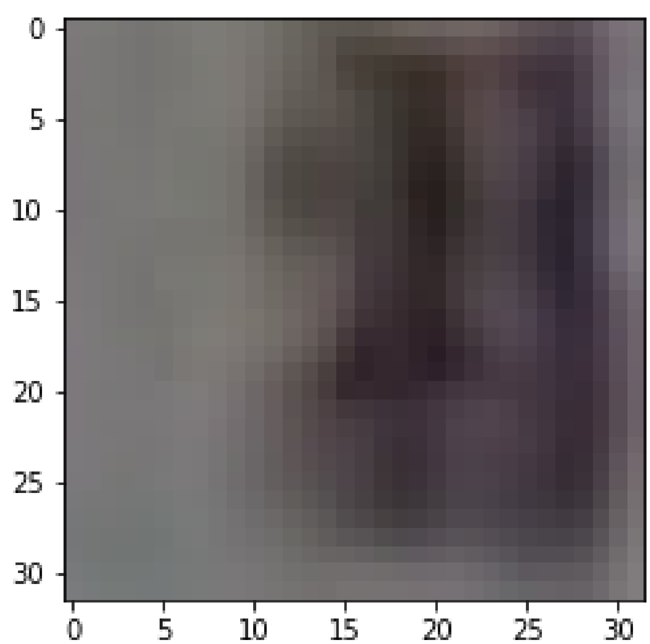
```
In [6]: #Select a random sample of images and corresponding labels from the dataset (at least

        import matplotlib.pyplot as plt
        %matplotlib inline

        for i in range(20,30):
            plt.imshow(X_train[i, :, :, :,])
            plt.show()
            print(y_train[i])
```



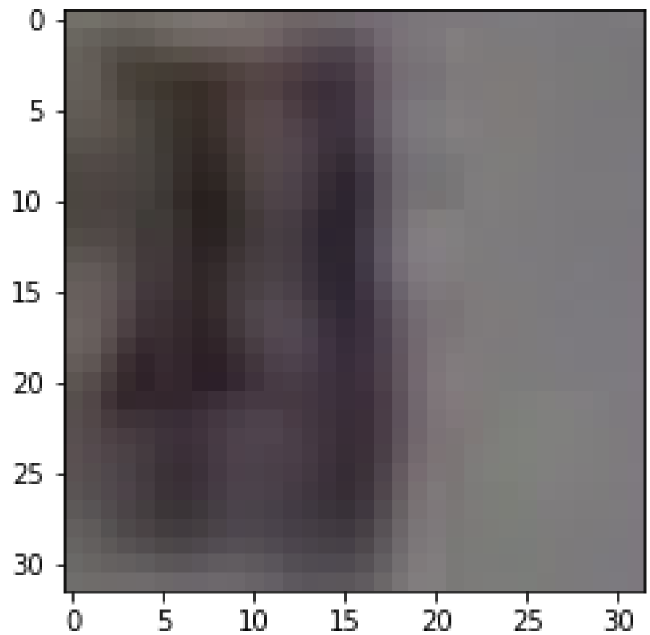
[1]



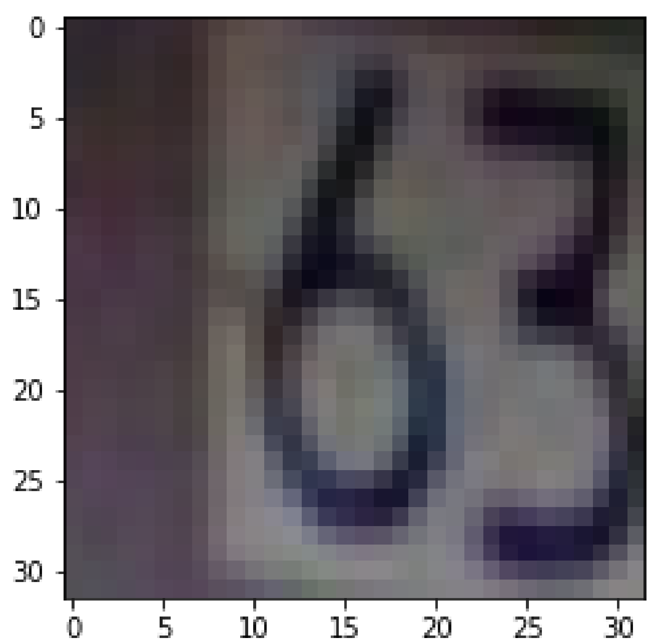
[6]



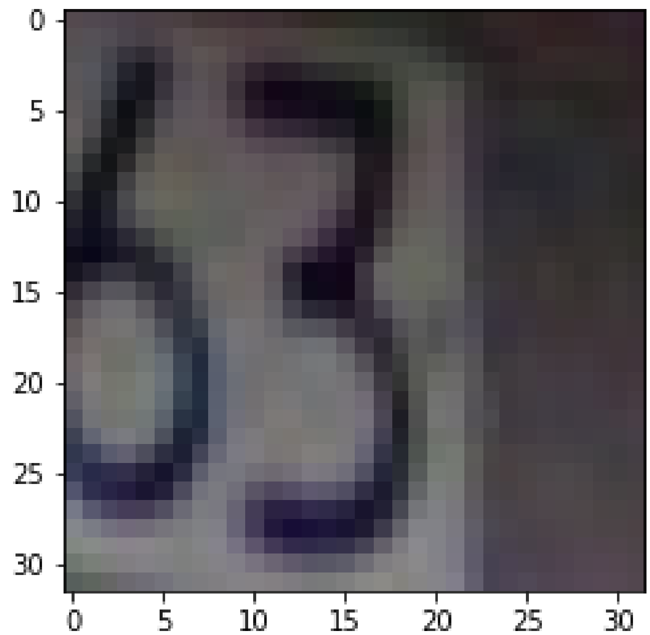
[2]



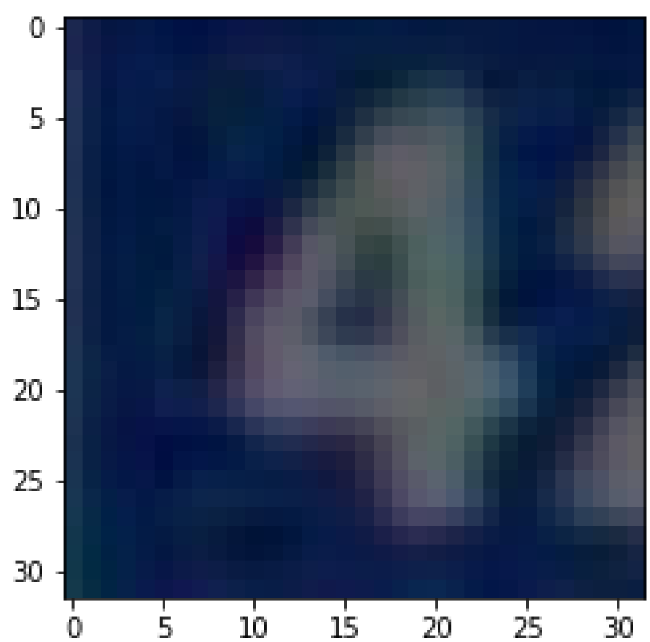
[3]



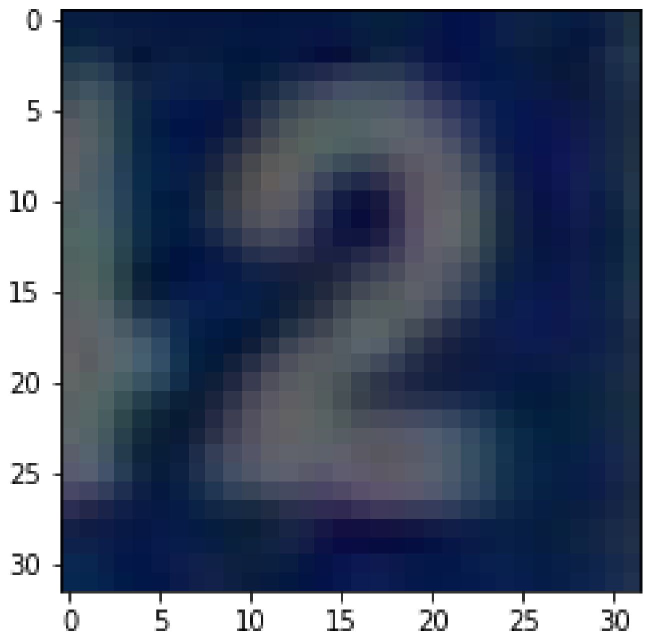
[6]



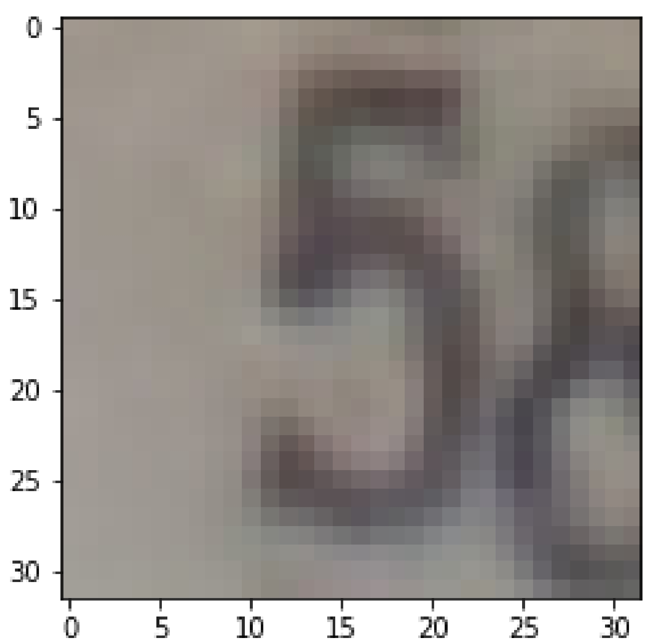
[3]



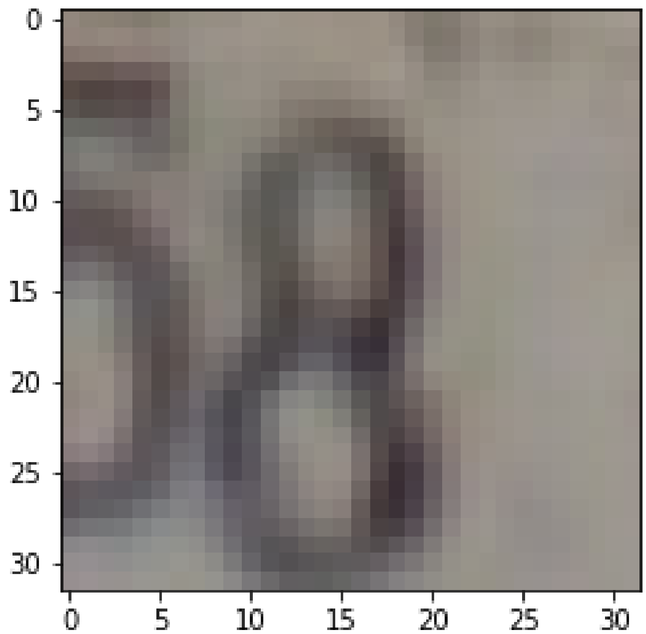
[4]



[2]



[5]

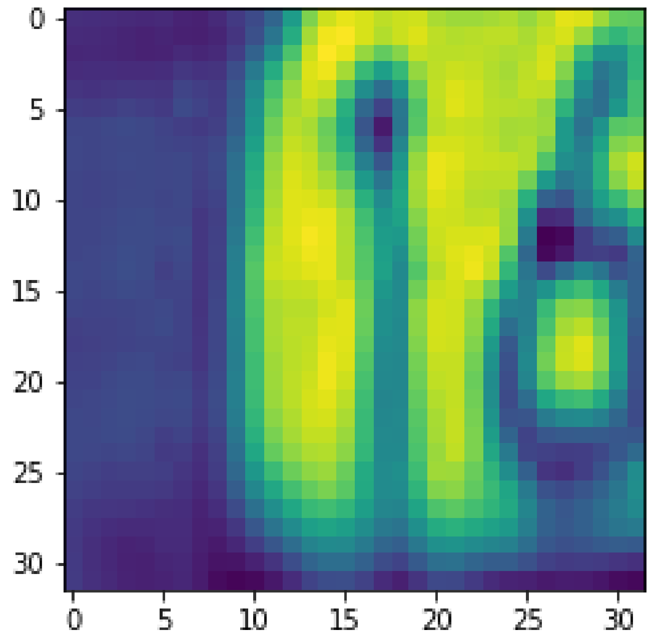


[8]

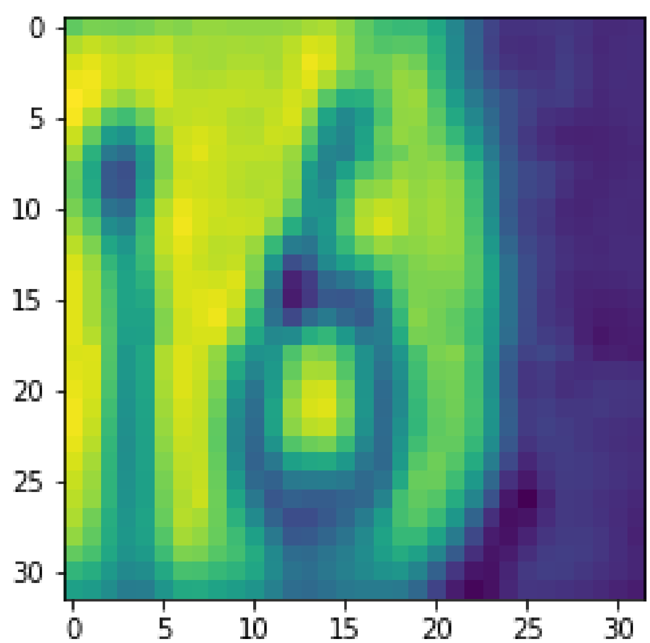
```
In [7]: #Convert the training and test images to grayscale by taking the average across all co
        #Hint: retain the channel dimension, which will now have size 1.

        X_train_grayscale = np.mean(X_train, 3).reshape(73257, 32, 32, 1)/255
        X_test_grayscale = np.mean(X_test,3).reshape(26032, 32,32 ,1)/255
```

```
In [8]: X_train_grayscale_plot = np.mean(X_train,3)
```

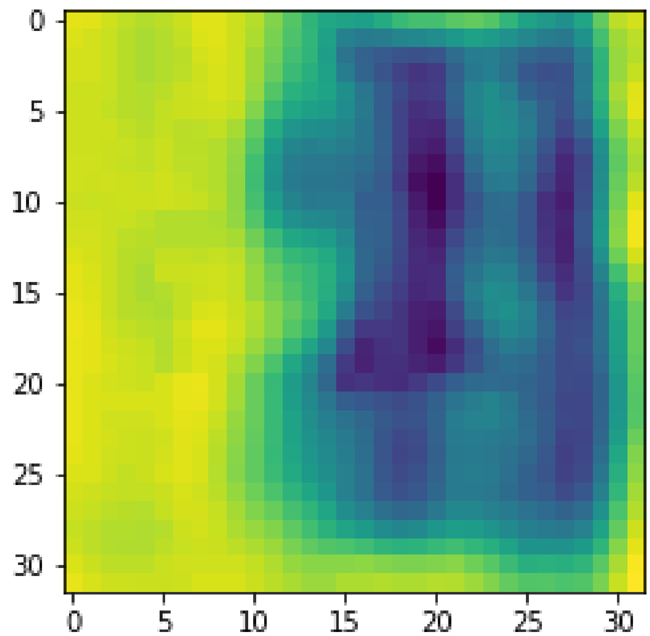
```
In [9]: #Select a random sample of the grayscale images and corresponding labels from the data
        #and display them in a figure

        for i in range(20,30):
            plt.imshow(X_train_grayscale_plot[i, :, :,])
            plt.show()
            print(y_train[i])
```



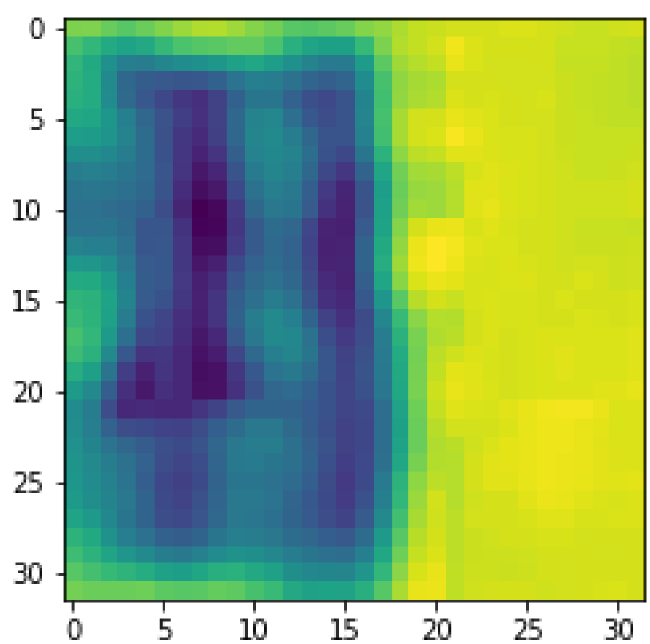
[1]



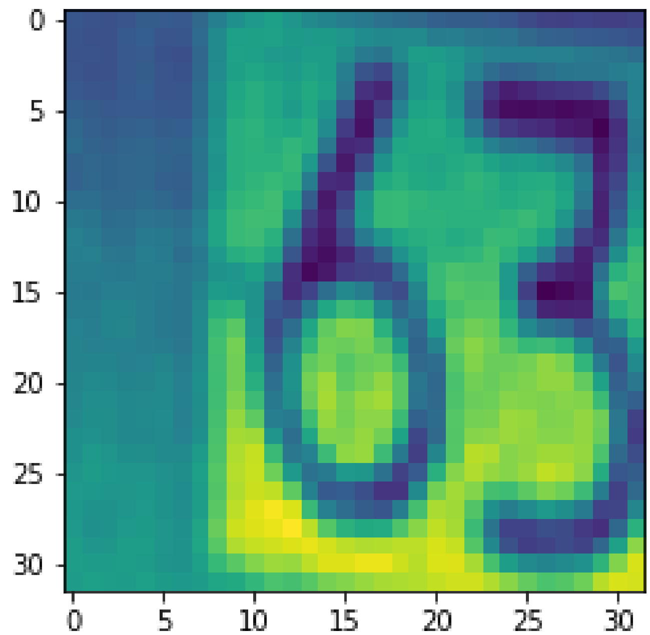
[6]



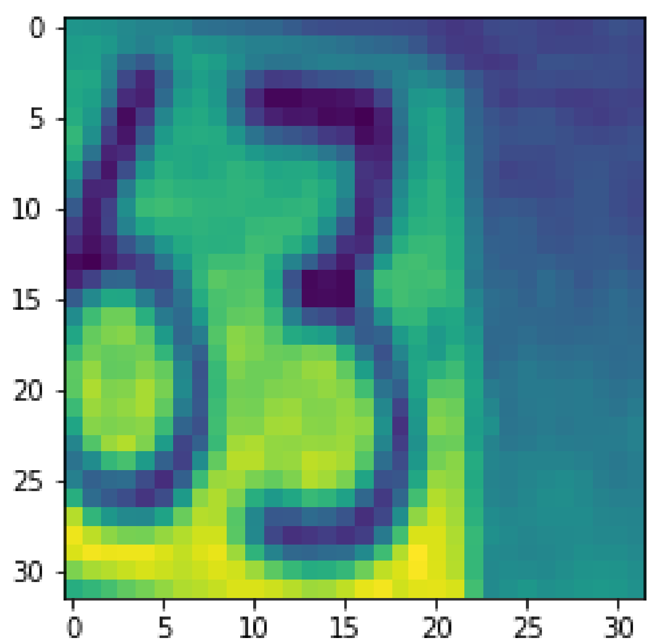
[2]



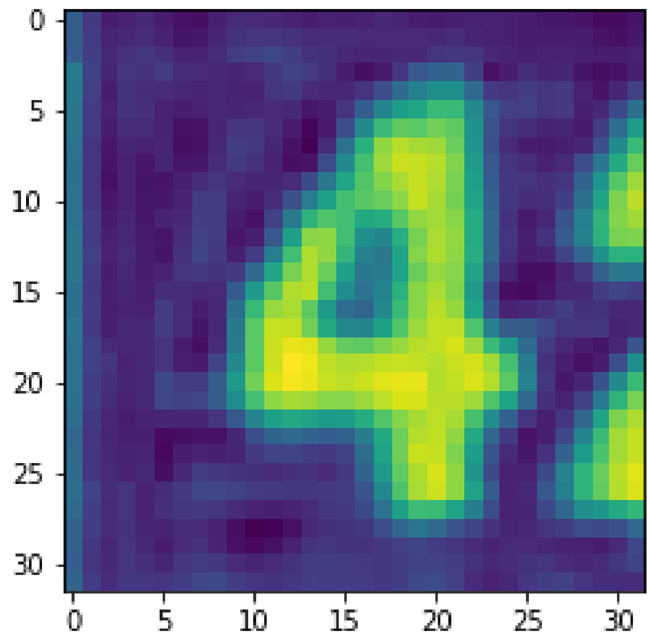
[3]



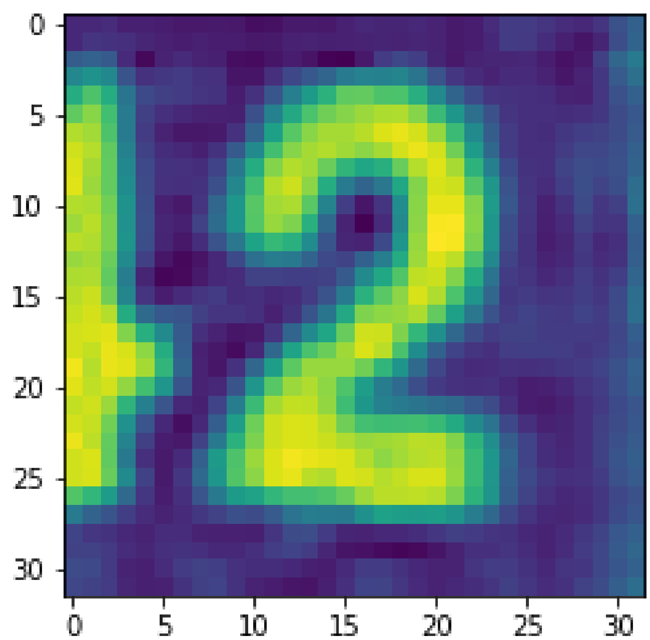
[6]



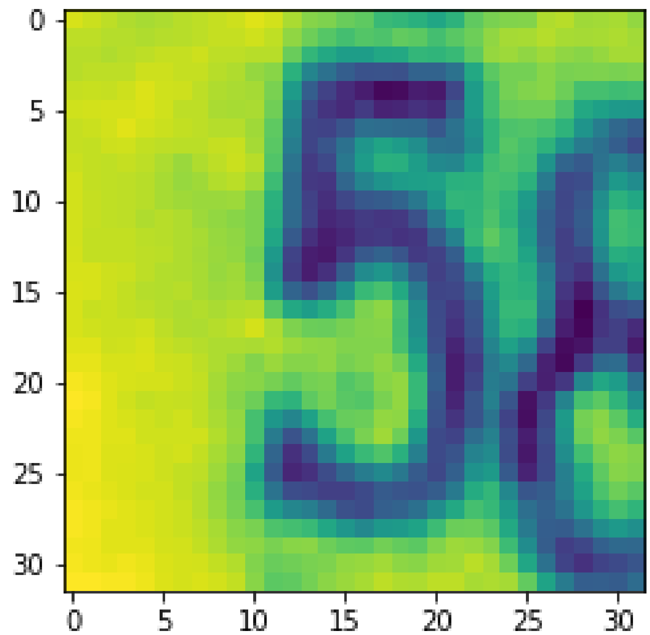
[3]



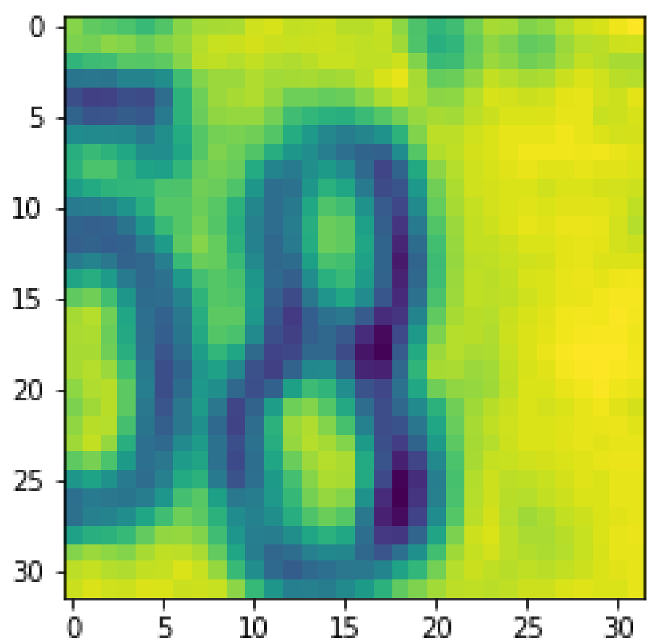
[4]



[2]



[5]



[8]

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 [11]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout,BatchNormalization, Conv2I
```

In [12]:
```
# from tensorflow.keras.utils import to_categorical
# y_train_binary = to_categorical(y_train)
# y_test_binary = to_categorical(y_test)
```

In [13]:
```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder().fit(y_train)
y_train_encoded = enc.transform(y_train).toarray()
y_test_encoded = enc.transform(y_test).toarray()
```

In [14]:
```
X_train[0].shape
```

Out[14]:
```
(32, 32, 3)
```

In [23]:
```
model_mlp = Sequential([
    Flatten(input_shape=(32,32,3)),
    Dense(256, activation='relu'),
    BatchNormalization(),
    Dense(64,activation='relu'),
    Dropout(0.2),
    Dense(32,activation='relu'),
    Dense(10,activation='softmax')
])
model_mlp.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
model_mlp.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_3 (Flatten)          (None, 3072)              0         
_________________________________________________________________
dense_12 (Dense)             (None, 256)               786688    
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024      
_________________________________________________________________
dense_13 (Dense)             (None, 64)                16448     
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_14 (Dense)             (None, 32)                2080      
_________________________________________________________________
dense_15 (Dense)             (None, 10)                330       
=================================================================
Total params: 806,570
Trainable params: 806,058
Non-trainable params: 512
_________________________________________________________________
```

In [24]:
```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

checkpoint = ModelCheckpoint(filepath = "mlp_weights",
                             save_best_only=True, save_weights_only=True,
                             monitor='val_loss', verbose=1)
early_stopping = EarlyStopping(patience=3, monitor='loss')
```

In [25]:
```
history = model_mlp.fit(X_train, y_train_encoded,
                        batch_size=128,
                        callbacks=[checkpoint, early_stopping],
                        validation_data=(X_test, y_test_encoded), epochs=25)
```

```
Train on 73257 samples, validate on 26032 samples
Epoch 1/25
73216/73257 [============================>.] - ETA: 0s - loss: 1.5482 - acc: 0.4857
Epoch 00001: val_loss improved from inf to 1.72838, saving model to mlp_weights
73257/73257 [==============================] - 45s 617us/sample - loss: 1.5481 - acc: 0.4857 -
Epoch 2/25
73216/73257 [============================>.] - ETA: 0s - loss: 1.1664 - acc: 0.6351
Epoch 00002: val_loss improved from 1.72838 to 1.25377, saving model to mlp_weights
73257/73257 [==============================] - 43s 588us/sample - loss: 1.1663 - acc: 0.6351 -
Epoch 3/25
73216/73257 [============================>.] - ETA: 0s - loss: 1.0766 - acc: 0.6689
Epoch 00003: val_loss did not improve from 1.25377
```

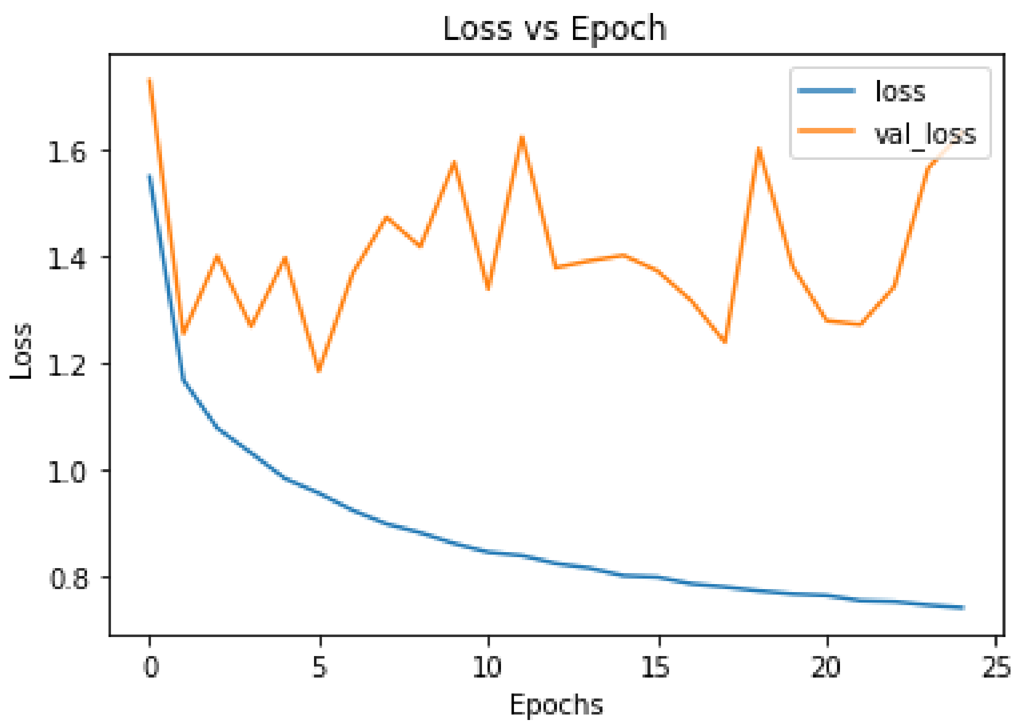
```
73257/73257 [==============================] - 43s 581us/sample - loss: 1.0764 - acc: 0.6690 -
Epoch 4/25
73088/73257 [============================>.] - ETA: 0s - loss: 1.0297 - acc: 0.6847
Epoch 00004: val_loss did not improve from 1.25377
73257/73257 [==============================] - 43s 590us/sample - loss: 1.0297 - acc: 0.6847 -
Epoch 5/25
73088/73257 [============================>.] - ETA: 0s - loss: 0.9817 - acc: 0.6999
Epoch 00005: val_loss did not improve from 1.25377
73257/73257 [==============================] - 43s 584us/sample - loss: 0.9819 - acc: 0.6998 -
Epoch 6/25
73088/73257 [============================>.] - ETA: 0s - loss: 0.9547 - acc: 0.7074
Epoch 00006: val_loss improved from 1.25377 to 1.18353, saving model to mlp_weights
73257/73257 [==============================] - 43s 584us/sample - loss: 0.9545 - acc: 0.7075 -
Epoch 7/25
73088/73257 [============================>.] - ETA: 0s - loss: 0.9220 - acc: 0.7168
Epoch 00007: val_loss did not improve from 1.18353
73257/73257 [==============================] - 43s 583us/sample - loss: 0.9223 - acc: 0.7167 -
Epoch 8/25
73088/73257 [============================>.] - ETA: 0s - loss: 0.8971 - acc: 0.7244
Epoch 00008: val_loss did not improve from 1.18353
73257/73257 [==============================] - 43s 583us/sample - loss: 0.8967 - acc: 0.7245 -
Epoch 9/25
73216/73257 [============================>.] - ETA: 0s - loss: 0.8801 - acc: 0.7284
Epoch 00009: val_loss did not improve from 1.18353
73257/73257 [==============================] - 43s 584us/sample - loss: 0.8802 - acc: 0.7284 -
Epoch 10/25
73216/73257 [============================>.] - ETA: 0s - loss: 0.8597 - acc: 0.7371
Epoch 00010: val_loss did not improve from 1.18353
73257/73257 [==============================] - 43s 580us/sample - loss: 0.8598 - acc: 0.7371 -
Epoch 11/25
73216/73257 [============================>.] - ETA: 0s - loss: 0.8437 - acc: 0.7404
Epoch 00011: val_loss did not improve from 1.18353
73257/73257 [==============================] - 43s 591us/sample - loss: 0.8437 - acc: 0.7404 -
Epoch 12/25
73216/73257 [============================>.] - ETA: 0s - loss: 0.8370 - acc: 0.7429
Epoch 00012: val_loss did not improve from 1.18353
73257/73257 [==============================] - 44s 599us/sample - loss: 0.8373 - acc: 0.7428 -
Epoch 13/25
73216/73257 [============================>.] - ETA: 0s - loss: 0.8224 - acc: 0.7474
Epoch 00013: val_loss did not improve from 1.18353
73257/73257 [==============================] - 44s 597us/sample - loss: 0.8224 - acc: 0.7474 -
Epoch 14/25
73216/73257 [============================>.] - ETA: 0s - loss: 0.8137 - acc: 0.7497
Epoch 00014: val_loss did not improve from 1.18353
73257/73257 [==============================] - 44s 595us/sample - loss: 0.8136 - acc: 0.7497 -
Epoch 15/25
73216/73257 [============================>.] - ETA: 0s - loss: 0.7992 - acc: 0.7544
Epoch 00015: val_loss did not improve from 1.18353
```

```
73257/73257 [==============================] - 44s 599us/sample - loss: 0.7994 - acc: 0.7544 -
Epoch 16/25
73216/73257 [============================>.] - ETA: 0s - loss: 0.7968 - acc: 0.7552
Epoch 00016: val_loss did not improve from 1.18353
73257/73257 [==============================] - 44s 595us/sample - loss: 0.7968 - acc: 0.7552 -
Epoch 17/25
73216/73257 [============================>.] - ETA: 0s - loss: 0.7842 - acc: 0.7581
Epoch 00017: val_loss did not improve from 1.18353
73257/73257 [==============================] - 44s 597us/sample - loss: 0.7843 - acc: 0.7581 -
Epoch 18/25
73088/73257 [============================>.] - ETA: 0s - loss: 0.7783 - acc: 0.7615
Epoch 00018: val_loss did not improve from 1.18353
73257/73257 [==============================] - 44s 595us/sample - loss: 0.7783 - acc: 0.7615 -
Epoch 19/25
73216/73257 [============================>.] - ETA: 0s - loss: 0.7710 - acc: 0.7631
Epoch 00019: val_loss did not improve from 1.18353
73257/73257 [==============================] - 43s 580us/sample - loss: 0.7710 - acc: 0.7631 -
Epoch 20/25
73088/73257 [============================>.] - ETA: 0s - loss: 0.7651 - acc: 0.7657
Epoch 00020: val_loss did not improve from 1.18353
73257/73257 [==============================] - 42s 580us/sample - loss: 0.7652 - acc: 0.7656 -
Epoch 21/25
73216/73257 [============================>.] - ETA: 0s - loss: 0.7623 - acc: 0.7651
Epoch 00021: val_loss did not improve from 1.18353
73257/73257 [==============================] - 43s 588us/sample - loss: 0.7624 - acc: 0.7651 -
Epoch 22/25
73088/73257 [============================>.] - ETA: 0s - loss: 0.7528 - acc: 0.7691
Epoch 00022: val_loss did not improve from 1.18353
73257/73257 [==============================] - 43s 586us/sample - loss: 0.7526 - acc: 0.7692 -
Epoch 23/25
73216/73257 [============================>.] - ETA: 0s - loss: 0.7508 - acc: 0.7677
Epoch 00023: val_loss did not improve from 1.18353
73257/73257 [==============================] - 43s 592us/sample - loss: 0.7508 - acc: 0.7677 -
Epoch 24/25
73088/73257 [============================>.] - ETA: 0s - loss: 0.7445 - acc: 0.7712
Epoch 00024: val_loss did not improve from 1.18353
73257/73257 [==============================] - 42s 580us/sample - loss: 0.7444 - acc: 0.7714 -
Epoch 25/25
73216/73257 [============================>.] - ETA: 0s - loss: 0.7394 - acc: 0.7731
Epoch 00025: val_loss did not improve from 1.18353
73257/73257 [==============================] - 43s 582us/sample - loss: 0.7394 - acc: 0.7731 -
```

In [26]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['loss','val_loss'], loc='upper right')
```

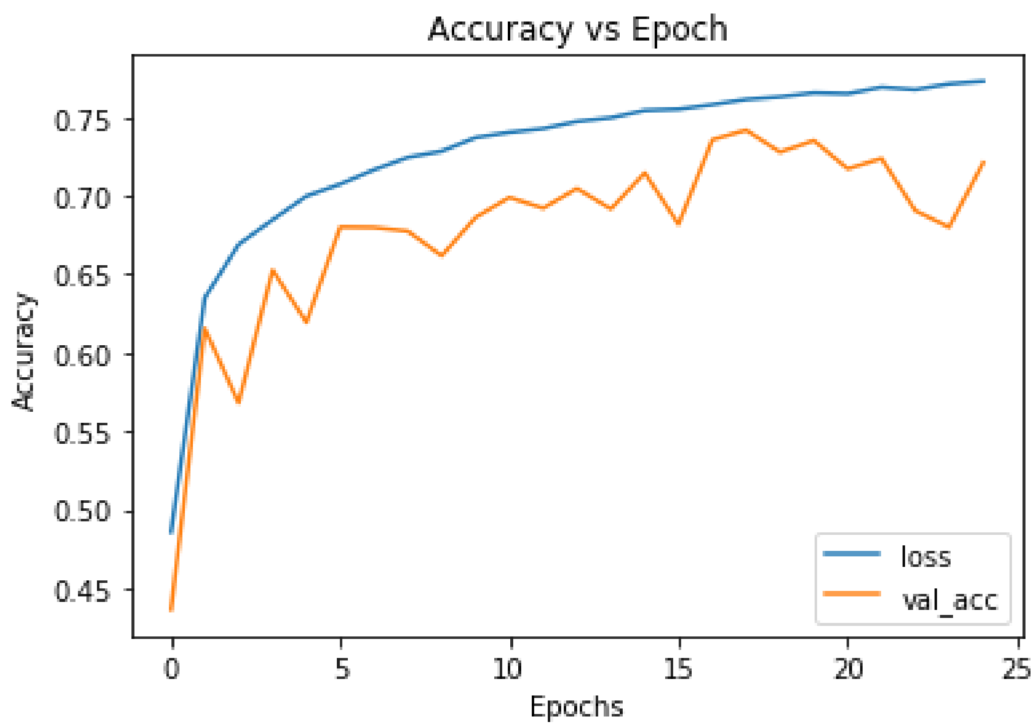
```
plt.title("Loss vs Epoch")
```

Out[26]: Text(0.5, 1.0, 'Loss vs Epoch')



In [27]:
```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['loss','val_acc'], loc='lower right')
plt.title("Accuracy vs Epoch")
```

Out[27]: Text(0.5, 1.0, 'Accuracy vs Epoch')



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 [19]: model_cnn = Sequential([
             Conv2D(filters=32, input_shape=(32,32,3),kernel_size=(3,3),activation='relu'),
             MaxPool2D(pool_size=(3,3)),
             Conv2D(filters=64,kernel_size=(3,3),padding='same',activation='relu'),
             MaxPool2D(pool_size=(3,3)),
             BatchNormalization(momentum=0.98),
```

```
            Conv2D(filters=64,kernel_size=(3,3),padding='same',activation='relu'),
            Dropout(0.02),
            Flatten(),
            Dense(32,activation='relu'),
            Dense(10,activation='softmax')
        ])
        model_cnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
        model_cnn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 10, 10, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 64)        18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 3, 3, 64)          0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 3, 3, 64)          256       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928     
_________________________________________________________________
dropout_1 (Dropout)          (None, 3, 3, 64)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 32)                18464     
_________________________________________________________________
dense_7 (Dense)              (None, 10)                330       
=================================================================
Total params: 75,370
Trainable params: 75,242
Non-trainable params: 128
_________________________________________________________________
```

```
In [20]: checkpoint_cnn = ModelCheckpoint(filepath="cnn_weights",save_best_only=True, save_wei
                                          monitor='val_loss',mode='min',verbose=1)
         earlystopping_cnn = EarlyStopping(patience=3,monitor='val_loss')
```

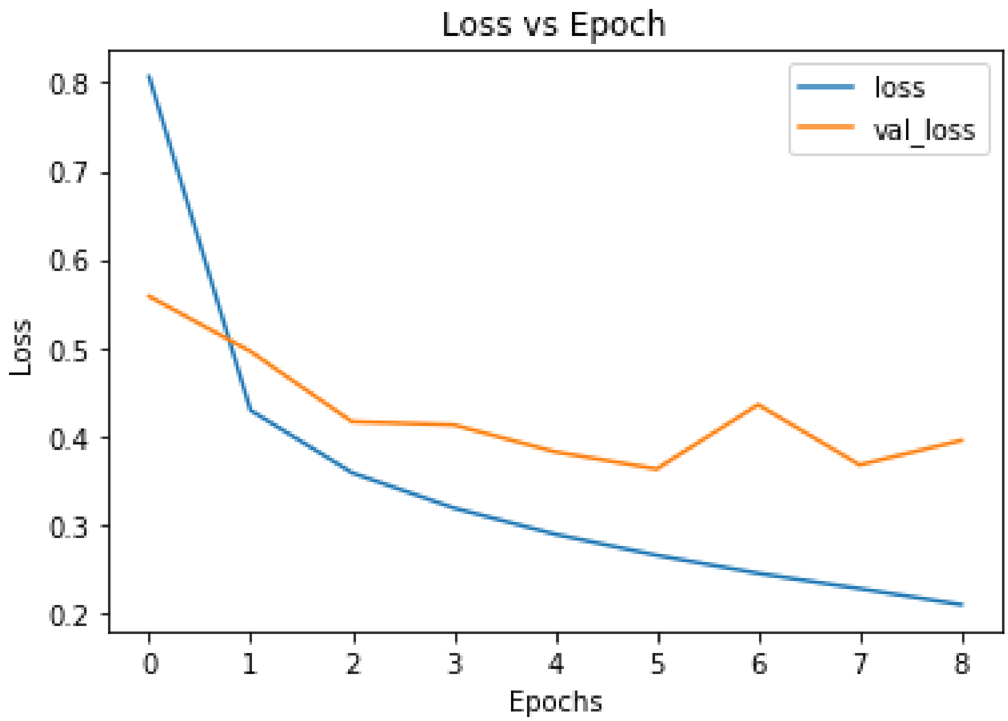
```
In [21]: history = model_cnn.fit(X_train, y_train_encoded,
                                batch_size=64,
                                callbacks=[checkpoint_cnn, earlystopping_cnn],
                                validation_data=(X_test, y_test_encoded), epochs=15)
```

```
Train on 73257 samples, validate on 26032 samples
Epoch 1/15
```

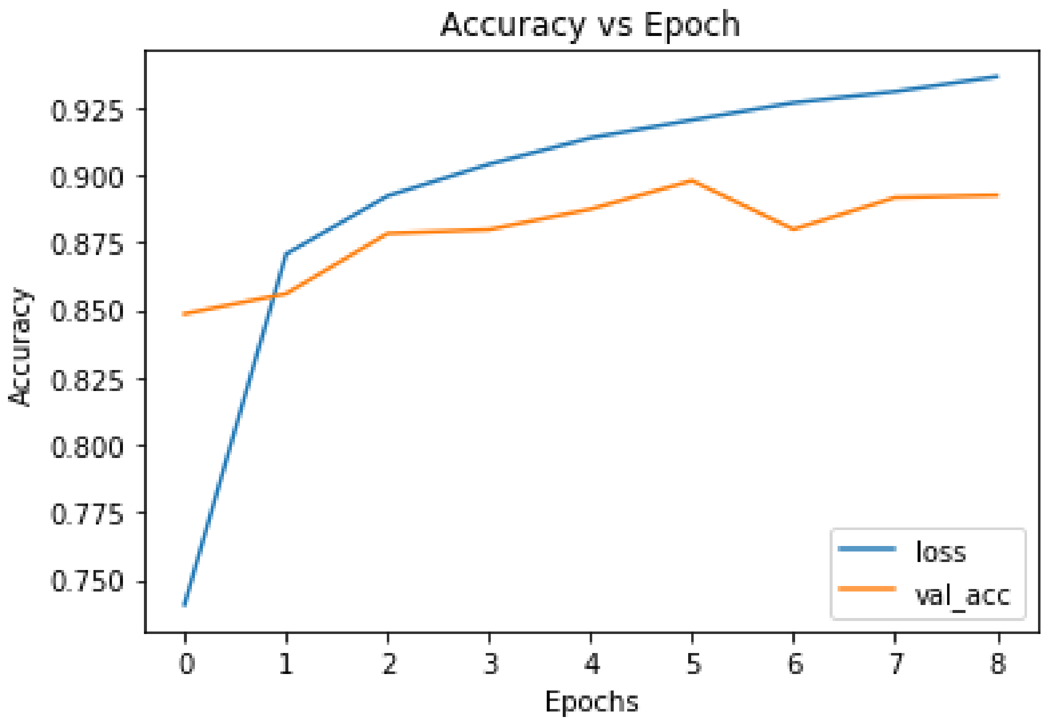
```
73216/73257 [============================>.] - ETA: 0s - loss: 0.8068 - acc: 0.7407
Epoch 00001: val_loss improved from inf to 0.55817, saving model to cnn_weights
73257/73257 [==============================] - 378s 5ms/sample - loss: 0.8066 - acc: 0.7407 - v
Epoch 2/15
73216/73257 [============================>.] - ETA: 0s - loss: 0.4293 - acc: 0.8706
Epoch 00002: val_loss improved from 0.55817 to 0.49575, saving model to cnn_weights
73257/73257 [==============================] - 360s 5ms/sample - loss: 0.4292 - acc: 0.8707 - v
Epoch 3/15
73216/73257 [============================>.] - ETA: 0s - loss: 0.3586 - acc: 0.8923
Epoch 00003: val_loss improved from 0.49575 to 0.41634, saving model to cnn_weights
73257/73257 [==============================] - 372s 5ms/sample - loss: 0.3586 - acc: 0.8923 - v
Epoch 4/15
73216/73257 [============================>.] - ETA: 0s - loss: 0.3188 - acc: 0.9041
Epoch 00004: val_loss improved from 0.41634 to 0.41275, saving model to cnn_weights
73257/73257 [==============================] - 371s 5ms/sample - loss: 0.3188 - acc: 0.9041 - v
Epoch 5/15
73216/73257 [============================>.] - ETA: 0s - loss: 0.2890 - acc: 0.9137
Epoch 00005: val_loss improved from 0.41275 to 0.38219, saving model to cnn_weights
73257/73257 [==============================] - 382s 5ms/sample - loss: 0.2890 - acc: 0.9138 - v
Epoch 6/15
73216/73257 [============================>.] - ETA: 0s - loss: 0.2652 - acc: 0.9204- ETA: 7s -
Epoch 00006: val_loss improved from 0.38219 to 0.36283, saving model to cnn_weights
73257/73257 [==============================] - 371s 5ms/sample - loss: 0.2652 - acc: 0.9204 - v
Epoch 7/15
73216/73257 [============================>.] - ETA: 0s - loss: 0.2447 - acc: 0.9269
Epoch 00007: val_loss did not improve from 0.36283
73257/73257 [==============================] - 359s 5ms/sample - loss: 0.2448 - acc: 0.9268 - v
Epoch 8/15
73216/73257 [============================>.] - ETA: 0s - loss: 0.2277 - acc: 0.9309
Epoch 00008: val_loss did not improve from 0.36283
73257/73257 [==============================] - 364s 5ms/sample - loss: 0.2276 - acc: 0.9310 - v
Epoch 9/15
73216/73257 [============================>.] - ETA: 0s - loss: 0.2095 - acc: 0.9365
Epoch 00009: val_loss did not improve from 0.36283
73257/73257 [==============================] - 364s 5ms/sample - loss: 0.2095 - acc: 0.9365 - v
```

In [22]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['loss','val_loss'], loc='upper right')
plt.title("Loss vs Epoch")
```

Out[22]:
```
Text(0.5, 1.0, 'Loss vs Epoch')
```




```
In [23]: plt.plot(history.history['acc'])
         plt.plot(history.history['val_acc'])
         plt.xlabel('Epochs')
         plt.ylabel('Accuracy')
         plt.legend(['loss','val_acc'], loc='lower right')
         plt.title("Accuracy vs Epoch")

Out[23]: Text(0.5, 1.0, 'Accuracy vs Epoch')
```



In [ ]:

In [ ]:

In [ ]:

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 [25]:
```
model_cnn.load_weights("cnn_weights")
```

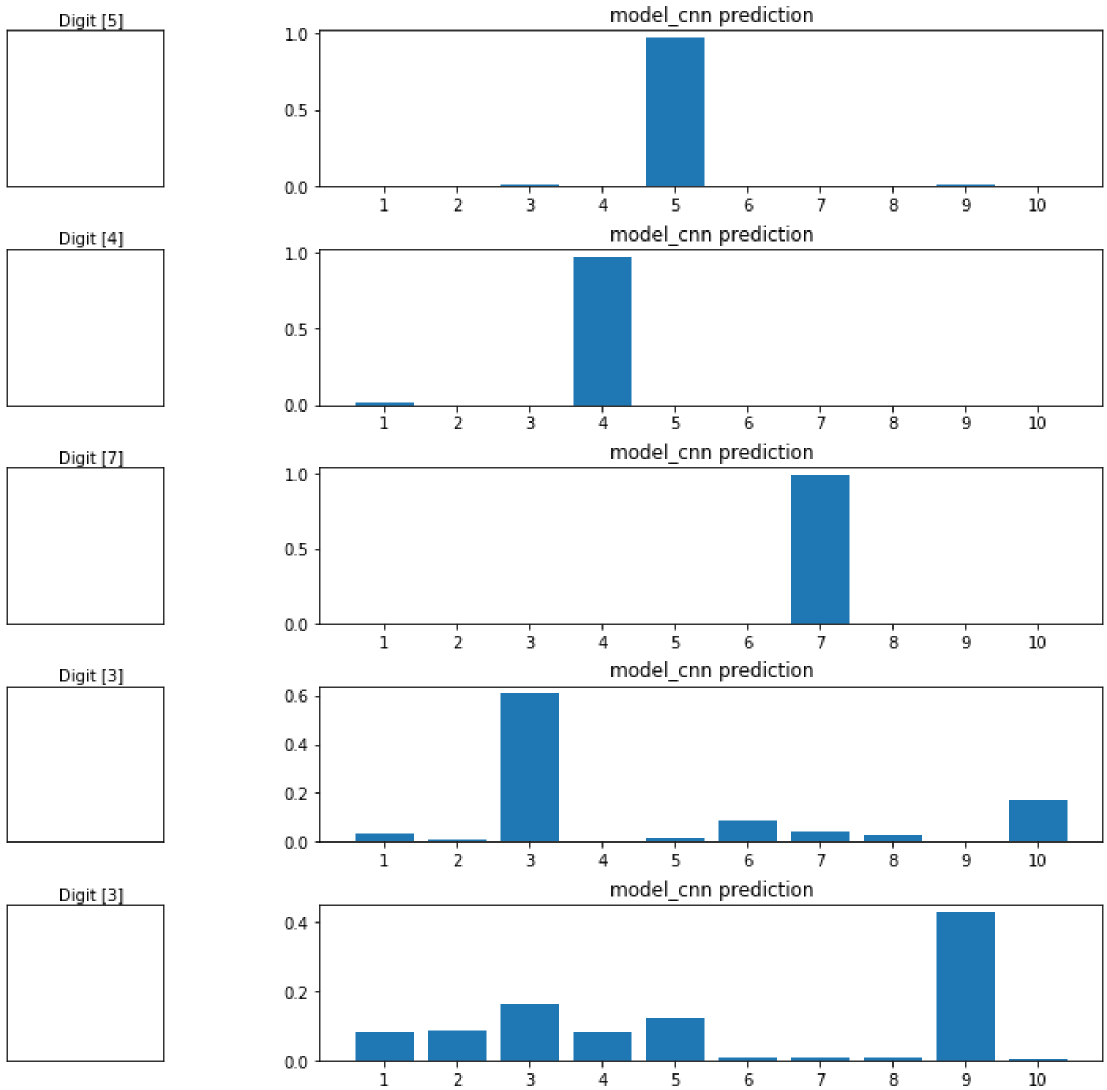
Out[25]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f7d686603c8>
```

In [ ]:

In [32]:
```
import random

num_test_images = X_test.shape[0]

random_inx = np.random.choice(num_test_images, 5)
```

```python
random_test_images = X_test[random_inx, ...]
random_test_images = tf.cast(random_test_images, tf.float32)
random_test_labels = y_test[random_inx, ...]

predictions = model_cnn.predict(random_test_images)

fig, axes = plt.subplots(5, 2, figsize=(16, 12))
fig.subplots_adjust(hspace=0.4, wspace=-0.2)

for i, (prediction, image, label) in enumerate(zip(predictions, random_test_images, ra
    axes[i, 0].imshow(np.squeeze(image))
    axes[i, 0].get_xaxis().set_visible(False)
    axes[i, 0].get_yaxis().set_visible(False)
    axes[i, 0].text(10., -1.5, f'Digit {label}')
    axes[i, 1].bar(np.arange(1,11), prediction)
    axes[i, 1].set_xticks(np.arange(1,11))
    axes[i, 1].set_title("model_cnn prediction")

plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
```

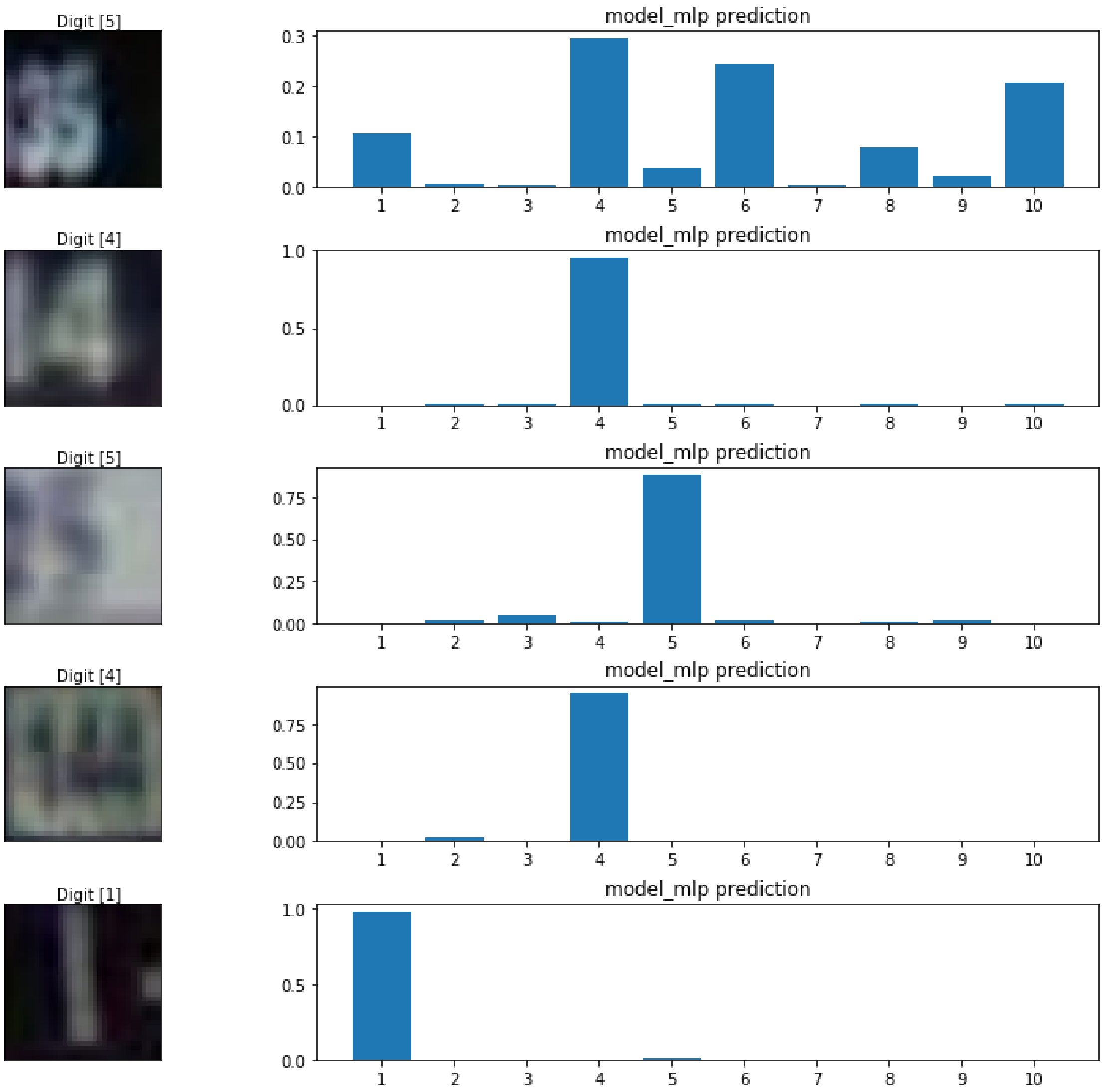


In [28]:
```
model_mlp.load_weights("mlp_weights")
```

Out[28]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f66c85b1978>

In [30]:
```
num_test_images = X_test.shape[0]

random_inx = np.random.choice(num_test_images, 5)
random_test_images = X_test[random_inx, ...]
random_test_labels = y_test[random_inx, ...]

predictions = model_mlp.predict(random_test_images)

fig, axes = plt.subplots(5, 2, figsize=(16, 12))
fig.subplots_adjust(hspace=0.4, wspace=-0.2)
```

```
for i, (prediction, image, label) in enumerate(zip(predictions, random_test_images, ra
    axes[i, 0].imshow(np.squeeze(image))
    axes[i, 0].get_xaxis().set_visible(False)
    axes[i, 0].get_yaxis().set_visible(False)
    axes[i, 0].text(10., -1.5, f'Digit {label}')
    axes[i, 1].bar(np.arange(1,11), prediction)
    axes[i, 1].set_xticks(np.arange(1,11))
    axes[i, 1].set_title("model_mlp prediction")

plt.show()
```




```
In [ ]: ! ls -lh
```

In [ ]:

In [ ]: