Capstone Project

Image classifier for the SVHN dataset

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.

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.

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. 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]:

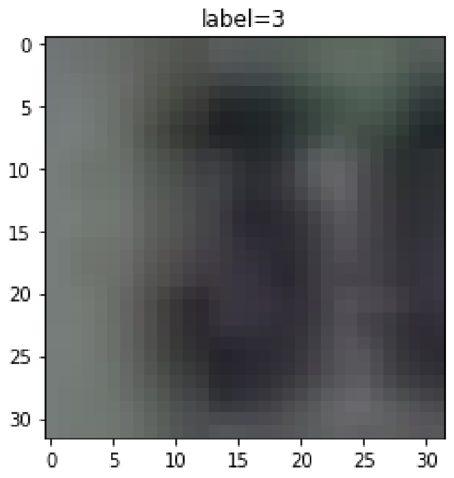
```
# Extract the training and testing images and labels separately from the train and test d
ictionaries loaded for you.
train_images = train['X']
train_labels = train['y']
test_images = test['X']
test_labels = test['y']
```

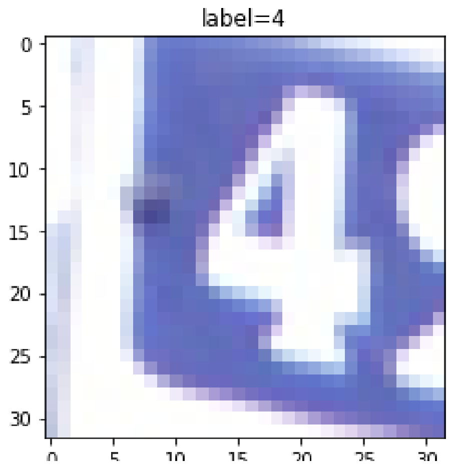
In [4]:

```
# Select a random sample of images and corresponding labels from the dataset (at least 10
), and display them in a figure.
def random_show(N = 10, images = None, labels = None):
    import random
    from matplotlib import pyplot as plt
    random_indices = random.sample(range(images.shape[3]),N)
    for index in random_indices:
        plt.figure()
        if images.shape[2] == 3:
            plt.imshow(images[:,:,:,index])
        elif images.shape[2] == 1:
            plt.imshow(images[:,:,0,index])
        else:
            raise ValueError("unexpected shapes")
        plt.title(f"label={labels[index][0]}")
        plt.show()
```

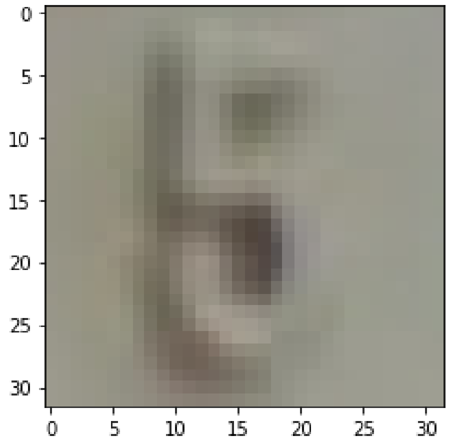
In [5]:

```
random_show( 10, train_images, train_labels)
random_show( 10, test_images, test_labels)
```

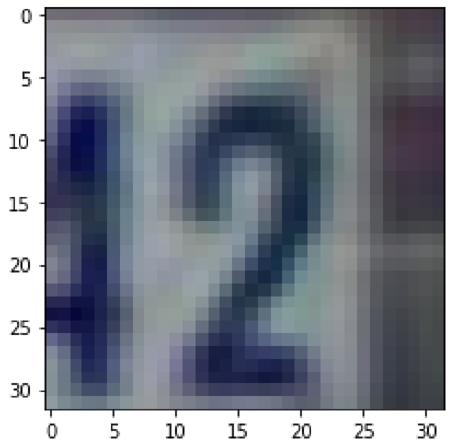




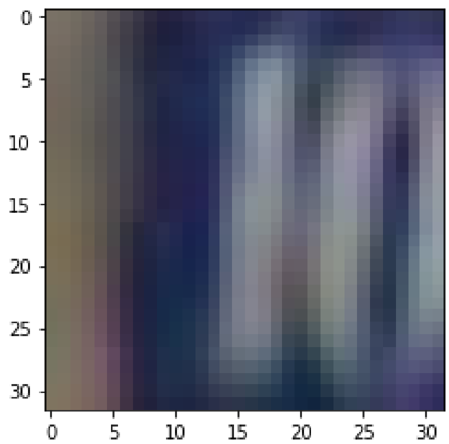
label=5



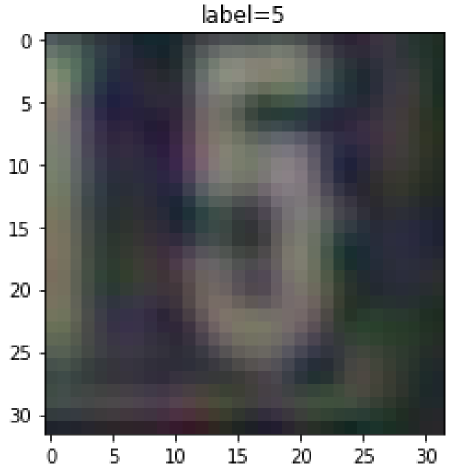
label=2



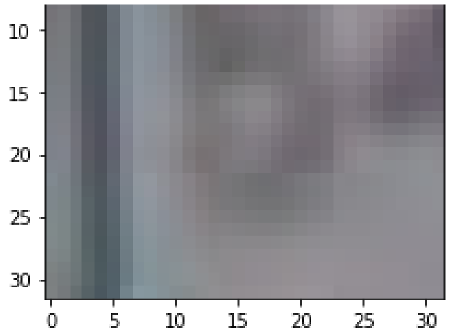
label=1



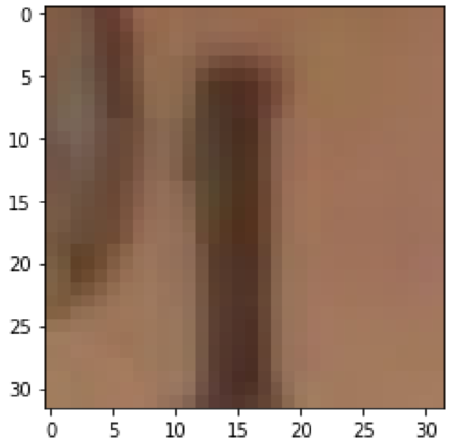
label=5



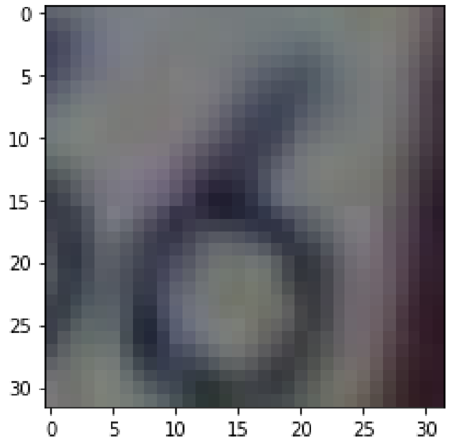
label=5



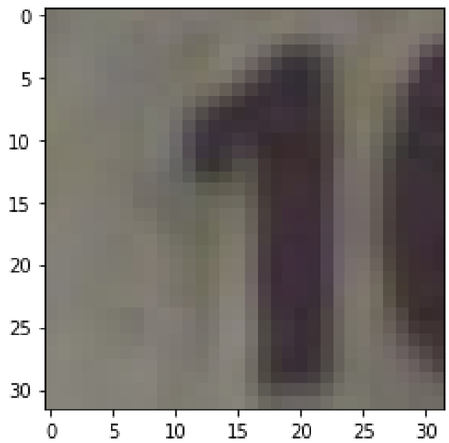
label=1



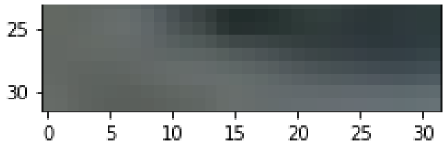
label=6



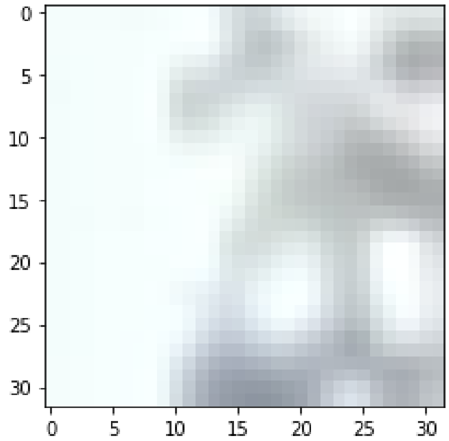
label=1



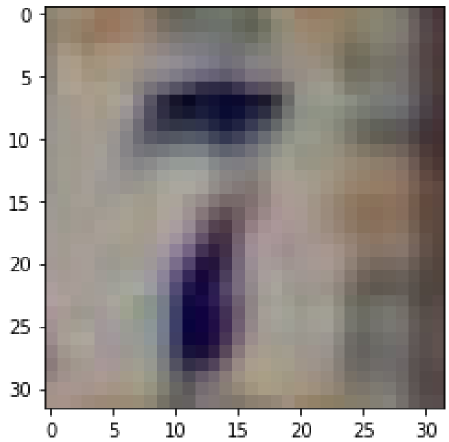
label=2



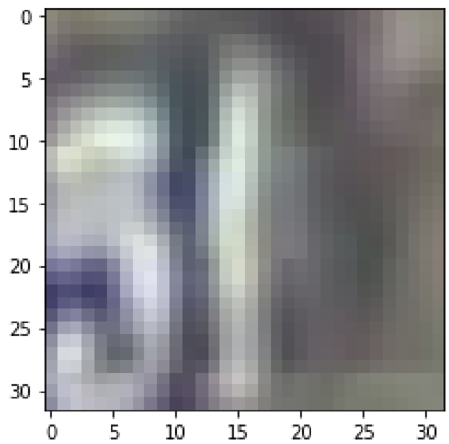
label=2



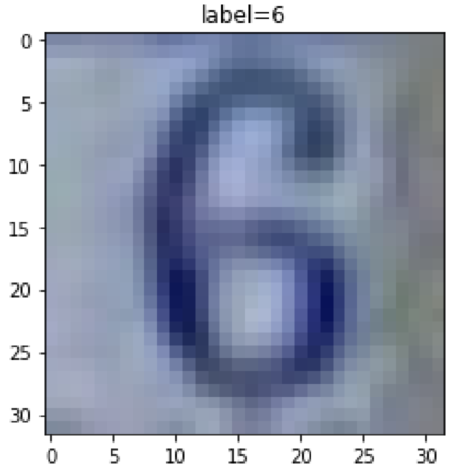
label=7



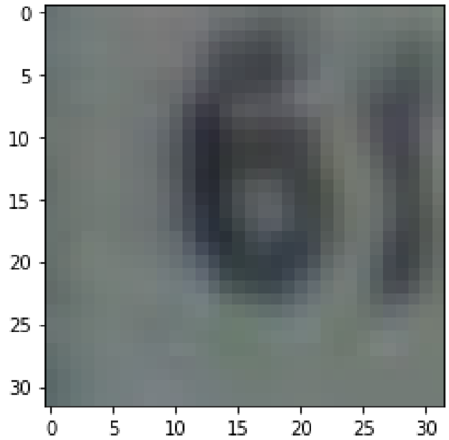
label=1



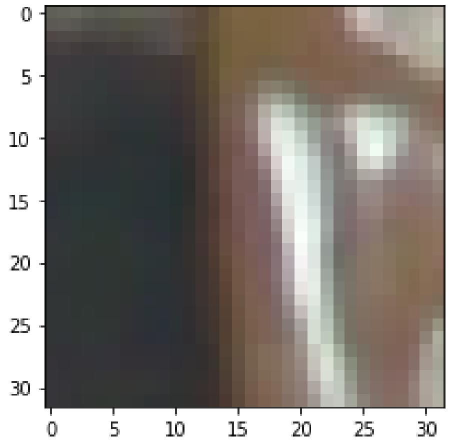
label=6



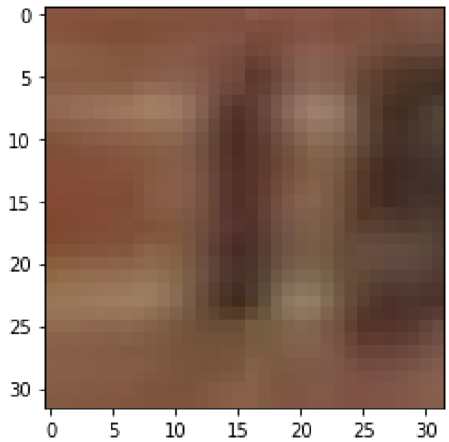
label=6



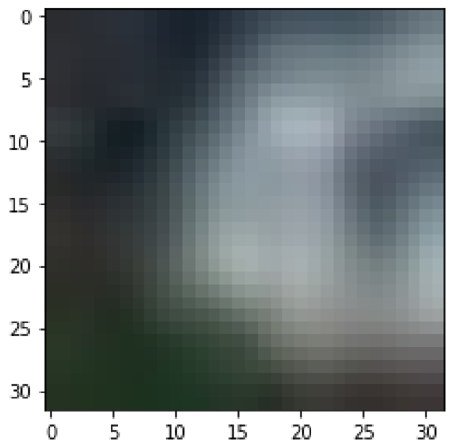
label=1



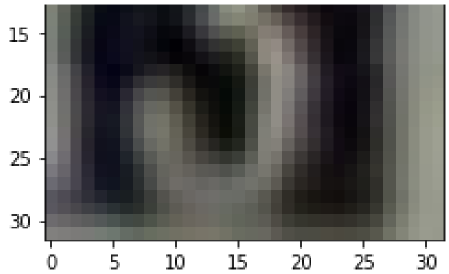
label=1



label=4



label=3



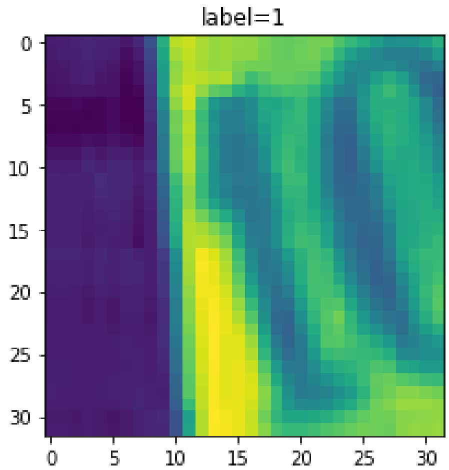
In [6]:

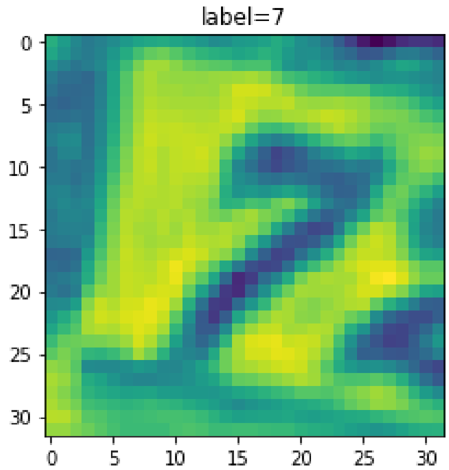
```
# Convert the training and test images to grayscale by taking the average across all colo
ur channels for each pixel. _Hint: retain the channel dimension, which will now have size
1._
import numpy as np
train_images = np.mean(train_images,axis = 2).reshape(32,32,1,-1)
test_images = np.mean(test_images,axis = 2).reshape(32,32,1,-1)
print(train_images.shape)
print(test_images.shape)
```

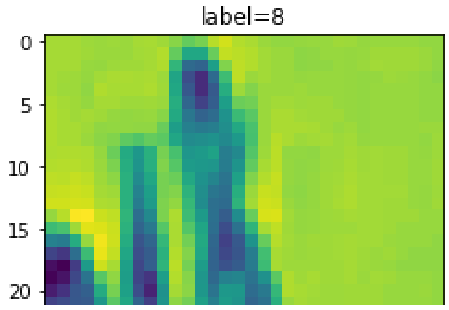
```
(32, 32, 1, 73257)
(32, 32, 1, 26032)
```

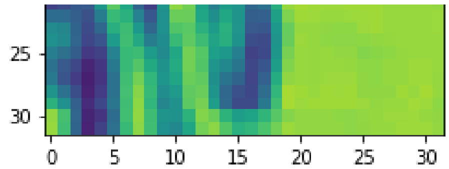
In [7]:

```
random_show( 10, train_images, train_labels)
random_show( 10, test_images, test_labels)
```

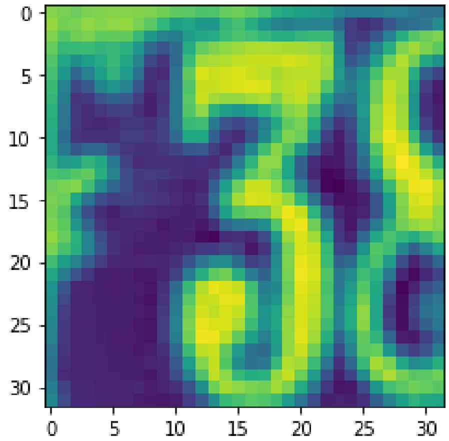




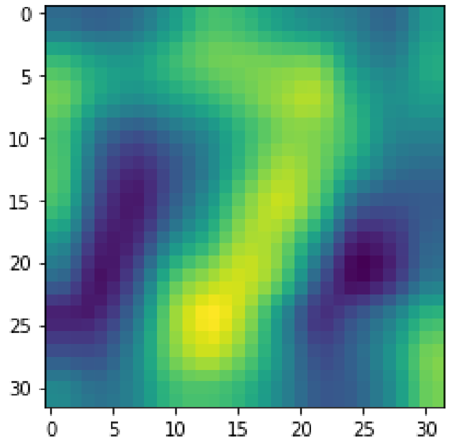




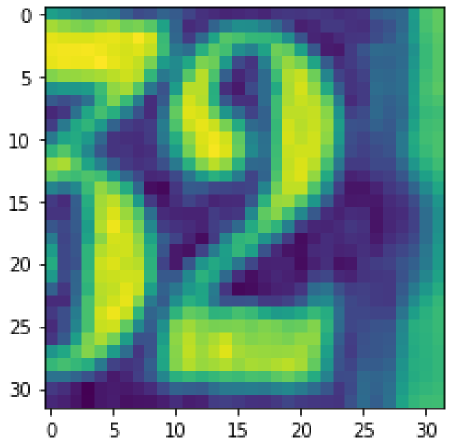
label=3



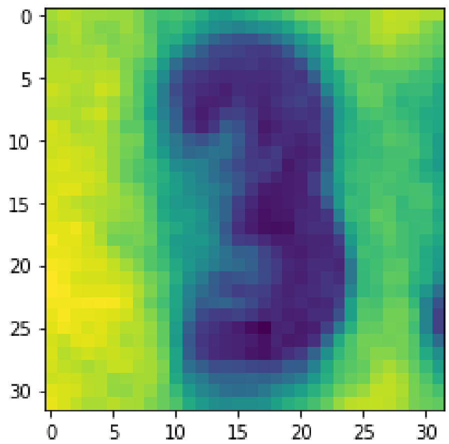
label=2



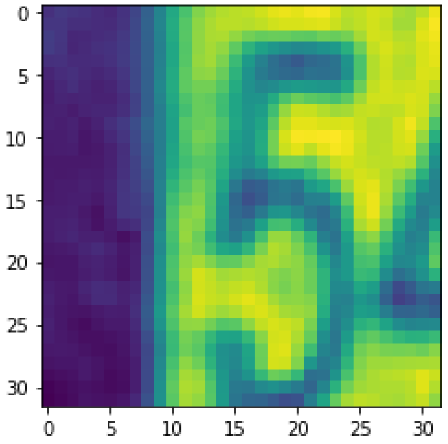
label=2



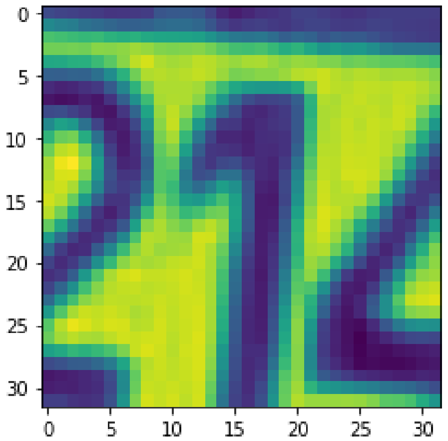
label=3



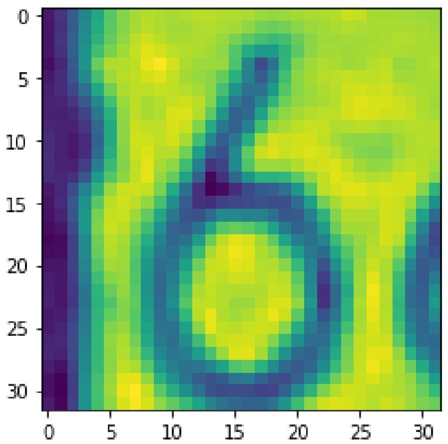
label=5



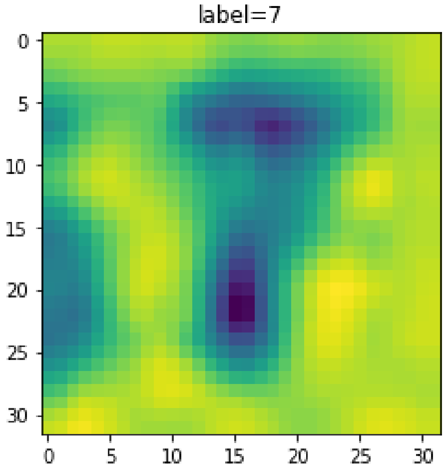
label=1



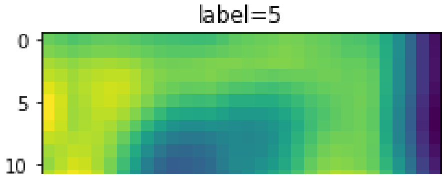
label=6

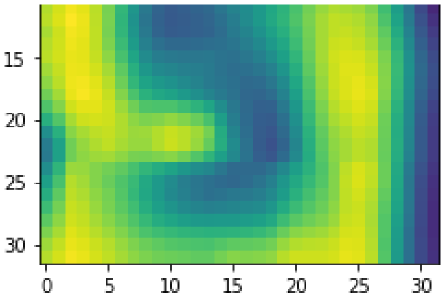


label=7

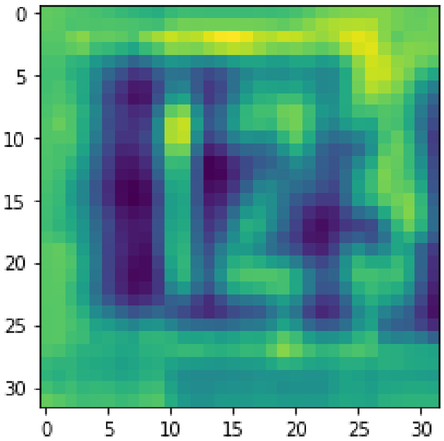


label=5

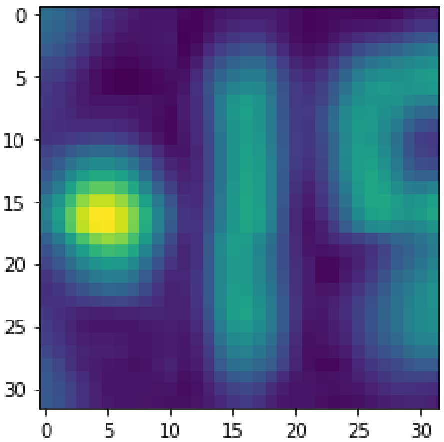




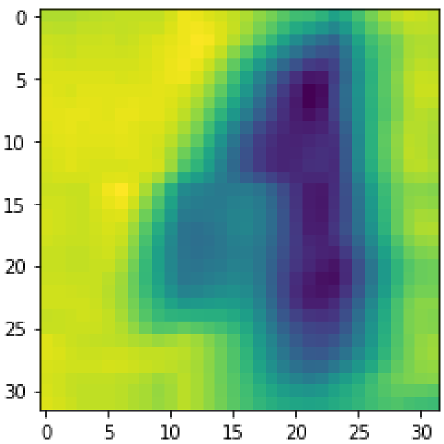
label=2



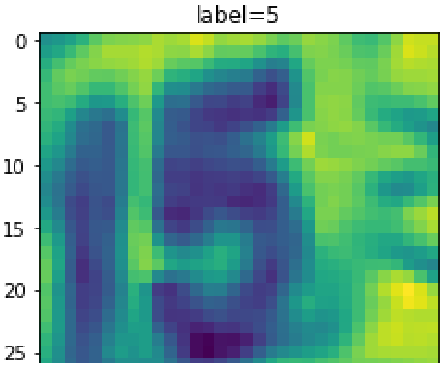
label=1

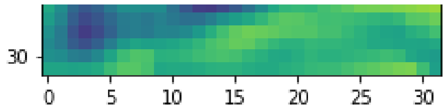


label=4

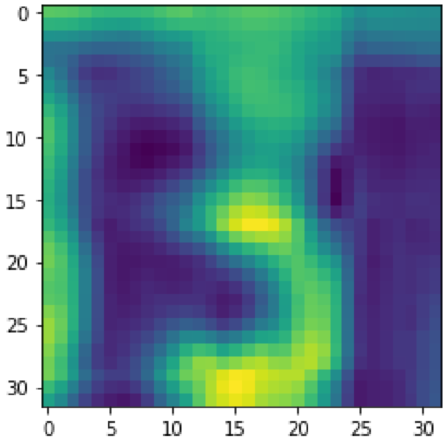


label=5

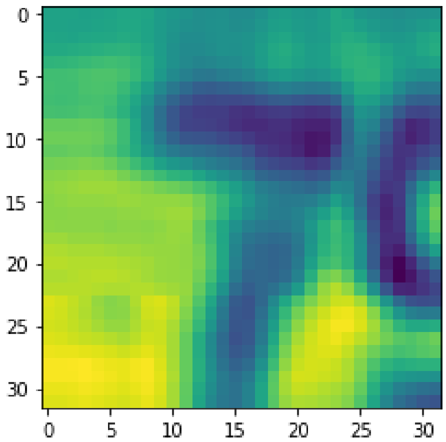




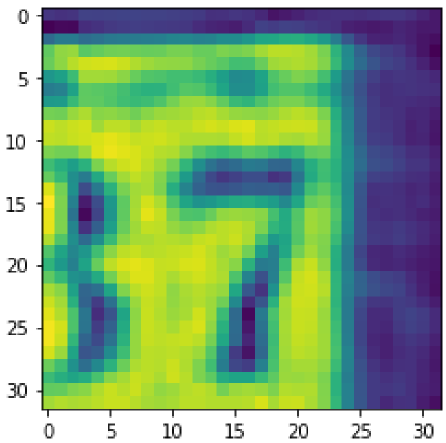
label=3



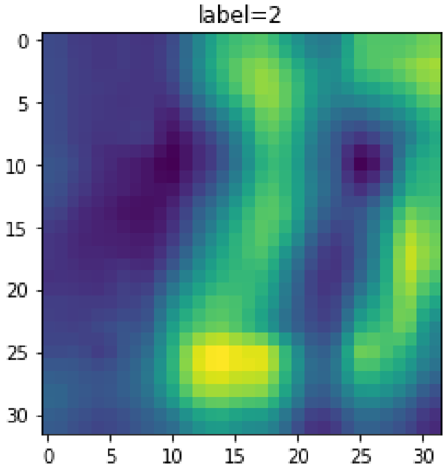
label=7



label=7



label=2



In [8]:

```
train_images = np.moveaxis(train_images, -1, 0)
test_images = np.moveaxis(test_images, -1, 0)
```

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 [9]:

```
#Build an MLP classifier model using the Sequential API. Your model should use only Flatt
en and Dense layers, with the final layer having a 10-way softmax output.
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

mlp_model = Sequential([
    Flatten(input_shape = (32,32,1)),
    Dense(units = 512, activation = "relu"),
    Dense(units = 512, activation = "relu"),
    Dense(units = 512, activation = "relu"),
    Dense(units = 512, activation = "relu"),
    Dense(units = 10, activation = "softmax"),
])
```

In [10]:

```
#Print out the model summary (using the summary() method)
mlp_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten (Flatten)            (None, 1024)              0         
_________________________________________________________________
dense (Dense)                (None, 512)               524800    
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656    
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656    
_________________________________________________________________
dense_3 (Dense)              (None, 512)               262656    
_________________________________________________________________
dense_4 (Dense)              (None, 10)                5130      
=================================================================
Total params: 1,317,898
Trainable params: 1,317,898
Non-trainable params: 0
_________________________________________________________________
```

In [11]:

```
# Compile and train the model (we recommend a maximum of 30 epochs), making use of both t
raining 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

callback_best_model_checkpoint = tf.keras.callbacks.ModelCheckpoint(
    filepath="./mlp_best_model_checkpoint/checkpoint",
    save_freq="epoch",
    save_best_only=True)

callback_early_stop = tf.keras.callbacks.EarlyStopping(monitor="val_loss",
                                                       patience=5)

callbacks = [callback_best_model_checkpoint, callback_early_stop]

mlp_model.compile(loss=  "categorical_crossentropy",
                  optimizer= tf.keras.optimizers.Adam(learning_rate=0.0001),
                  metrics = ['accuracy'])

mlp_history = mlp_model.fit(x = train_images,
              y = tf.keras.utils.to_categorical(train_labels-1, num_classes=10),
              epochs=30,
              validation_split = 0.15,
              batch_size = 1000,
              callbacks=callbacks)
```

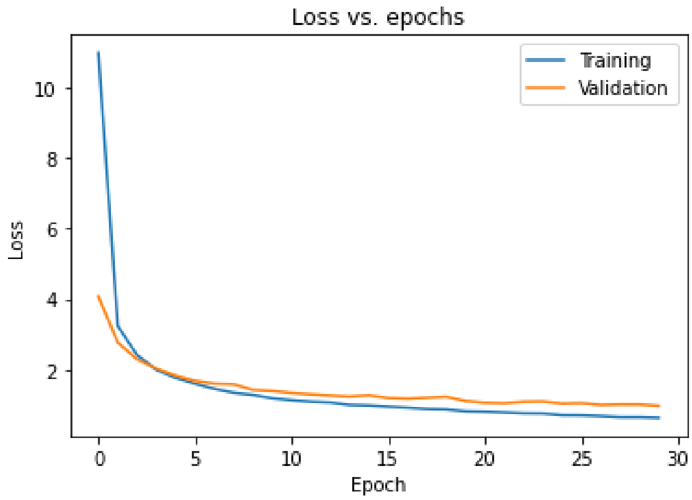
```
Epoch 1/30
62/63 [============================>.] - ETA: 0s - loss: 10.9917 - accuracy: 0.1778WARNIN
G:tensorflow:From /home/zhentao/anaconda3/lib/python3.8/site-packages/tensorflow/python/t
raining/tracking/tracking.py:111: Model.state_updates (from tensorflow.python.keras.engin
e.training) is deprecated and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatically.
WARNING:tensorflow:From /home/zhentao/anaconda3/lib/python3.8/site-packages/tensorflow/py
thon/training/tracking/tracking.py:111: Layer.updates (from tensorflow.python.keras.engin
e.base_layer) is deprecated and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatically.
INFO:tensorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 17ms/step - loss: 10.9628 - accuracy: 0.1782
- val_loss: 4.0762 - val_accuracy: 0.2569
Epoch 2/30
62/63 [============================>.] - ETA: 0s - loss: 3.2384 - accuracy: 0.3382INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 14ms/step - loss: 3.2364 - accuracy: 0.3386 -
val_loss: 2.7718 - val_accuracy: 0.3986
Epoch 3/30
53/63 [========================>.....] - ETA: 0s - loss: 2.4483 - accuracy: 0.4275INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 14ms/step - loss: 2.4195 - accuracy: 0.4319 -
val_loss: 2.2976 - val_accuracy: 0.4530
Epoch 4/30
63/63 [==============================] - ETA: 0s - loss: 1.9986 - accuracy: 0.4962INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 14ms/step - loss: 1.9986 - accuracy: 0.4962 -
val_loss: 2.0347 - val_accuracy: 0.4884
Epoch 5/30
52/63 [=======================>......] - ETA: 0s - loss: 1.7905 - accuracy: 0.5351INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 15ms/step - loss: 1.7708 - accuracy: 0.5394 -
val_loss: 1.8335 - val_accuracy: 0.5202
Epoch 6/30
51/63 [=======================>......] - ETA: 0s - loss: 1.6098 - accuracy: 0.5689INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 14ms/step - loss: 1.6049 - accuracy: 0.5705 -
val_loss: 1.6881 - val_accuracy: 0.5575
Epoch 7/30
54/63 [========================>.....] - ETA: 0s - loss: 1.4634 - accuracy: 0.5993INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 14ms/step - loss: 1.4603 - accuracy: 0.6005 -
val_loss: 1.6061 - val_accuracy: 0.5735
Epoch 8/30
53/63 [========================>.....] - ETA: 0s - loss: 1.3461 - accuracy: 0.6232INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 14ms/step - loss: 1.3498 - accuracy: 0.6242 -
```

```
val_loss: 1.5876 - val_accuracy: 0.5822
Epoch 9/30
53/63 [========================>.....] - ETA: 0s - loss: 1.2972 - accuracy: 0.6362INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 15ms/step - loss: 1.2871 - accuracy: 0.6388 -
val_loss: 1.4279 - val_accuracy: 0.6127
Epoch 10/30
53/63 [========================>.....] - ETA: 0s - loss: 1.1944 - accuracy: 0.6581INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 14ms/step - loss: 1.1934 - accuracy: 0.6581 -
val_loss: 1.3998 - val_accuracy: 0.6228
Epoch 11/30
54/63 [========================>.....] - ETA: 0s - loss: 1.1374 - accuracy: 0.6724INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 14ms/step - loss: 1.1383 - accuracy: 0.6730 -
val_loss: 1.3425 - val_accuracy: 0.6423
Epoch 12/30
52/63 [=======================>......] - ETA: 0s - loss: 1.0995 - accuracy: 0.6843INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 14ms/step - loss: 1.0975 - accuracy: 0.6843 -
val_loss: 1.3041 - val_accuracy: 0.6523
Epoch 13/30
51/63 [=======================>......] - ETA: 0s - loss: 1.0858 - accuracy: 0.6851INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 14ms/step - loss: 1.0748 - accuracy: 0.6891 -
val_loss: 1.2694 - val_accuracy: 0.6543
Epoch 14/30
52/63 [=======================>......] - ETA: 0s - loss: 1.0049 - accuracy: 0.7072INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 16ms/step - loss: 1.0043 - accuracy: 0.7065 -
val_loss: 1.2418 - val_accuracy: 0.6666
Epoch 15/30
63/63 [==============================] - 0s 5ms/step - loss: 0.9880 - accuracy: 0.7099 -
val_loss: 1.2760 - val_accuracy: 0.6661
Epoch 16/30
52/63 [=======================>......] - ETA: 0s - loss: 0.9497 - accuracy: 0.7204INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 15ms/step - loss: 0.9546 - accuracy: 0.7189 -
val_loss: 1.2022 - val_accuracy: 0.6734
Epoch 17/30
53/63 [========================>.....] - ETA: 0s - loss: 0.9249 - accuracy: 0.7248INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 15ms/step - loss: 0.9259 - accuracy: 0.7251 -
val_loss: 1.1833 - val_accuracy: 0.6802
Epoch 18/30
63/63 [==============================] - 0s 5ms/step - loss: 0.8895 - accuracy: 0.7358 -
val_loss: 1.2102 - val_accuracy: 0.6655
Epoch 19/30
63/63 [==============================] - 0s 5ms/step - loss: 0.8783 - accuracy: 0.7388 -
val_loss: 1.2365 - val_accuracy: 0.6853
Epoch 20/30
52/63 [=======================>......] - ETA: 0s - loss: 0.8252 - accuracy: 0.7534INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 15ms/step - loss: 0.8224 - accuracy: 0.7540 -
val_loss: 1.1172 - val_accuracy: 0.7010
Epoch 21/30
53/63 [========================>.....] - ETA: 0s - loss: 0.8142 - accuracy: 0.7553INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 14ms/step - loss: 0.8116 - accuracy: 0.7567 -
val_loss: 1.0685 - val_accuracy: 0.7093
Epoch 22/30
54/63 [========================>.....] - ETA: 0s - loss: 0.7906 - accuracy: 0.7617INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 15ms/step - loss: 0.7910 - accuracy: 0.7617 -
val_loss: 1.0547 - val_accuracy: 0.7146
Epoch 23/30
63/63 [==============================] - 0s 5ms/step - loss: 0.7672 - accuracy: 0.7694 -
val_loss: 1.0955 - val_accuracy: 0.7091
Epoch 24/30
63/63 [==============================] - 0s 5ms/step - loss: 0.7630 - accuracy: 0.7706 -
val_loss: 1.1035 - val_accuracy: 0.7022
Epoch 25/30
```

```
51/63 [=======================>......] - ETA: 0s - loss: 0.7140 - accuracy: 0.7845INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 15ms/step - loss: 0.7142 - accuracy: 0.7841 -
val_loss: 1.0452 - val_accuracy: 0.7193
Epoch 26/30
63/63 [==============================] - 0s 5ms/step - loss: 0.7112 - accuracy: 0.7851 -
val_loss: 1.0581 - val_accuracy: 0.7126
Epoch 27/30
53/63 [========================>.....] - ETA: 0s - loss: 0.6942 - accuracy: 0.7890INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 14ms/step - loss: 0.6888 - accuracy: 0.7907 -
val_loss: 1.0055 - val_accuracy: 0.7270
Epoch 28/30
63/63 [==============================] - 0s 5ms/step - loss: 0.6592 - accuracy: 0.7993 -
val_loss: 1.0208 - val_accuracy: 0.7256
Epoch 29/30
63/63 [==============================] - 0s 5ms/step - loss: 0.6593 - accuracy: 0.7981 -
val_loss: 1.0196 - val_accuracy: 0.7239
Epoch 30/30
52/63 [=======================>......] - ETA: 0s - loss: 0.6393 - accuracy: 0.8071INFO:te
nsorflow:Assets written to: ./mlp_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 1s 15ms/step - loss: 0.6419 - accuracy: 0.8060 -
val_loss: 0.9789 - val_accuracy: 0.7414
```

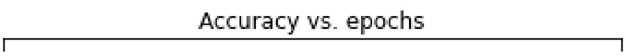
In [12]:

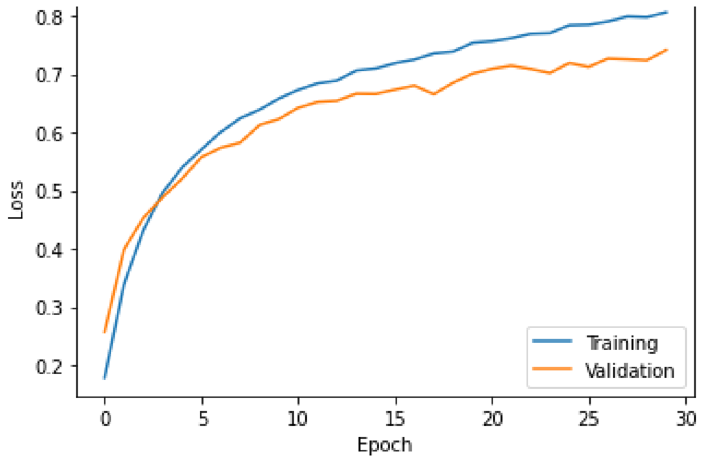
```python
# * As a guide, you should aim to achieve a final categorical cross entropy training loss
of less than 1.0 (the validation loss might be higher).
# * Plot the learning curves for loss vs epoch and accuracy vs epoch for both training an
d validation sets.
from matplotlib import pyplot as plt
plt.plot(mlp_history.history['loss'])
plt.plot(mlp_history.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```



In [13]:

```python
try:
    plt.plot(mlp_history.history['accuracy'])
    plt.plot(mlp_history.history['val_accuracy'])
except KeyError:
    plt.plot(mlp_history.history['acc'])
    plt.plot(mlp_history.history['val_acc'])
plt.title('Accuracy vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='lower right')
plt.show()
```





In [14]:

```
# Evaluate the model on the test set
test_loss, test_acc = mlp_model.evaluate(test_images,
                                         tf.keras.utils.to_categorical(test_labels-1, n
um_classes=10),
                                         verbose=0)
print("Test loss: {:.3f}\nTest accuracy: {:.2f}%".format(test_loss, 100 * test_acc))
```

```
Test loss: 1.182
Test accuracy: 72.44%
```

In [ ]:

```

```

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 [15]:

```
#You should design and build the model yourself. Feel free to experiment with different C
NN 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.)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, Dropout, BatchNormalization,
MaxPool2D

DROPOUT_RATE = 0.3

cnn_model = Sequential([
    Conv2D(input_shape = (32,32,1), filters = 8, activation="relu",padding='same', kerne
l_size = (3,3)),
    Dropout(rate = DROPOUT_RATE),
    BatchNormalization(),
    MaxPool2D(pool_size=(2,2)),
```

```
    Conv2D( filters = 8, activation="relu",padding='same', kernel_size = (3,3)),
    Dropout(rate = DROPOUT_RATE),
    BatchNormalization(),
    MaxPool2D(pool_size=(2,2)),
    Flatten(),
    Dense(units=128,activation="relu"),
    Dense(units=10,activation="softmax")
])

cnn_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 32, 32, 8)         80        
_________________________________________________________________
dropout (Dropout)            (None, 32, 32, 8)         0         
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 8)         32        
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 8)         0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 8)         584       
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 8)         0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 8)         32        
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 8)           0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 128)               65664     
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290      
=================================================================
Total params: 67,682
Trainable params: 67,650
Non-trainable params: 32
_________________________________________________________________
```

In [16]:

```
#Compile and train the model (we recommend a maximum of 30 epochs), making use of both tr
aining 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.

callback_best_model_checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath="./cnn_best_
model_checkpoint/checkpoint",
                                                                     save_freq="epoch",
                                                                     save_best_only=True
)
callback_early_stop = tf.keras.callbacks.EarlyStopping(monitor="val_loss",patience=5)
callbacks = [callback_best_model_checkpoint, callback_early_stop]

cnn_model.compile(loss=  "categorical_crossentropy",
                  optimizer= tf.keras.optimizers.Adam(learning_rate=0.0001),
                  metrics = ['accuracy'])

cnn_history = cnn_model.fit(x = train_images,
              y = tf.keras.utils.to_categorical(train_labels-1, num_classes=10),
              epochs=30,
              validation_split = 0.15,
              batch_size = 1000,
              callbacks=callbacks)
```

```
Epoch 1/30
61/63 [============================>.] - ETA: 0s - loss: 2.7503 - accuracy: 0.1250INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 34ms/step - loss: 2.7448 - accuracy: 0.1255 -
```

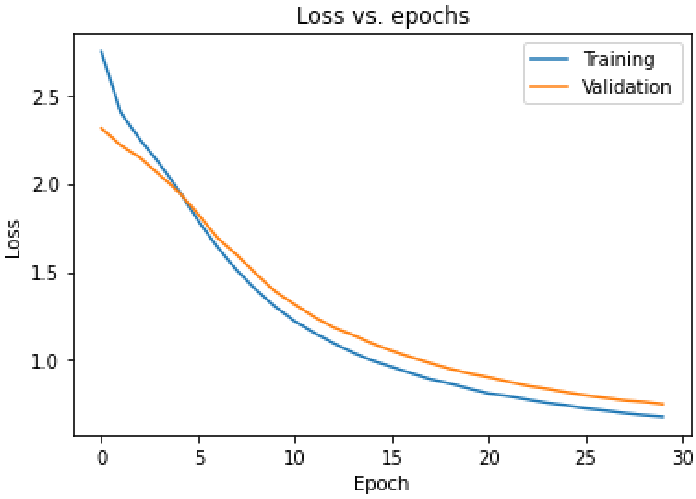
```
val_loss: 2.3120 - val_accuracy: 0.1713
Epoch 2/30
61/63 [============================>.] - ETA: 0s - loss: 2.4028 - accuracy: 0.1709INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 2.4009 - accuracy: 0.1713 -
val_loss: 2.2157 - val_accuracy: 0.2094
Epoch 3/30
61/63 [============================>.] - ETA: 0s - loss: 2.2482 - accuracy: 0.2155INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 30ms/step - loss: 2.2467 - accuracy: 0.2164 -
val_loss: 2.1454 - val_accuracy: 0.2339
Epoch 4/30
62/63 [============================>.] - ETA: 0s - loss: 2.1136 - accuracy: 0.2685INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 2.1136 - accuracy: 0.2685 -
val_loss: 2.0499 - val_accuracy: 0.2575
Epoch 5/30
62/63 [============================>.] - ETA: 0s - loss: 1.9606 - accuracy: 0.3316INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 1.9599 - accuracy: 0.3320 -
val_loss: 1.9513 - val_accuracy: 0.2850
Epoch 6/30
62/63 [============================>.] - ETA: 0s - loss: 1.7900 - accuracy: 0.3974INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 30ms/step - loss: 1.7895 - accuracy: 0.3975 -
val_loss: 1.8266 - val_accuracy: 0.3389
Epoch 7/30
62/63 [============================>.] - ETA: 0s - loss: 1.6395 - accuracy: 0.4565INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 1.6395 - accuracy: 0.4564 -
val_loss: 1.6914 - val_accuracy: 0.4168
Epoch 8/30
61/63 [============================>.] - ETA: 0s - loss: 1.5102 - accuracy: 0.5038INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 30ms/step - loss: 1.5087 - accuracy: 0.5044 -
val_loss: 1.5963 - val_accuracy: 0.4592
Epoch 9/30
61/63 [============================>.] - ETA: 0s - loss: 1.3978 - accuracy: 0.5463INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 1.3975 - accuracy: 0.5465 -
val_loss: 1.4879 - val_accuracy: 0.5212
Epoch 10/30
61/63 [============================>.] - ETA: 0s - loss: 1.3033 - accuracy: 0.5821INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 30ms/step - loss: 1.3014 - accuracy: 0.5831 -
val_loss: 1.3879 - val_accuracy: 0.5731
Epoch 11/30
62/63 [============================>.] - ETA: 0s - loss: 1.2208 - accuracy: 0.6096INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 1.2204 - accuracy: 0.6097 -
val_loss: 1.3150 - val_accuracy: 0.6023
Epoch 12/30
61/63 [============================>.] - ETA: 0s - loss: 1.1566 - accuracy: 0.6325INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 1.1550 - accuracy: 0.6334 -
val_loss: 1.2450 - val_accuracy: 0.6349
Epoch 13/30
62/63 [============================>.] - ETA: 0s - loss: 1.0968 - accuracy: 0.6554INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 30ms/step - loss: 1.0965 - accuracy: 0.6556 -
val_loss: 1.1858 - val_accuracy: 0.6615
Epoch 14/30
62/63 [============================>.] - ETA: 0s - loss: 1.0440 - accuracy: 0.6735INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 1.0435 - accuracy: 0.6737 -
val_loss: 1.1426 - val_accuracy: 0.6749
Epoch 15/30
61/63 [============================>.] - ETA: 0s - loss: 0.9994 - accuracy: 0.6904INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 30ms/step - loss: 0.9979 - accuracy: 0.6910 -
val_loss: 1.0949 - val_accuracy: 0.6977
Epoch 16/30
```

```
62/63 [============================>.] - ETA: 0s - loss: 0.9631 - accuracy: 0.7015INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 0.9630 - accuracy: 0.7015 -
val_loss: 1.0539 - val_accuracy: 0.7135
Epoch 17/30
61/63 [============================>.] - ETA: 0s - loss: 0.9280 - accuracy: 0.7133INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 0.9272 - accuracy: 0.7134 -
val_loss: 1.0177 - val_accuracy: 0.7235
Epoch 18/30
61/63 [============================>.] - ETA: 0s - loss: 0.8942 - accuracy: 0.7241INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 31ms/step - loss: 0.8934 - accuracy: 0.7244 -
val_loss: 0.9834 - val_accuracy: 0.7371
Epoch 19/30
60/63 [===========================>..] - ETA: 0s - loss: 0.8702 - accuracy: 0.7345INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 0.8689 - accuracy: 0.7347 -
val_loss: 0.9524 - val_accuracy: 0.7456
Epoch 20/30
61/63 [============================>.] - ETA: 0s - loss: 0.8403 - accuracy: 0.7434INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 30ms/step - loss: 0.8400 - accuracy: 0.7432 -
val_loss: 0.9259 - val_accuracy: 0.7517
Epoch 21/30
61/63 [============================>.] - ETA: 0s - loss: 0.8129 - accuracy: 0.7520INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 0.8131 - accuracy: 0.7521 -
val_loss: 0.9044 - val_accuracy: 0.7649
Epoch 22/30
61/63 [============================>.] - ETA: 0s - loss: 0.7983 - accuracy: 0.7555INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 0.7986 - accuracy: 0.7552 -
val_loss: 0.8799 - val_accuracy: 0.7685
Epoch 23/30
61/63 [============================>.] - ETA: 0s - loss: 0.7782 - accuracy: 0.7627INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 30ms/step - loss: 0.7789 - accuracy: 0.7625 -
val_loss: 0.8564 - val_accuracy: 0.7723
Epoch 24/30
61/63 [============================>.] - ETA: 0s - loss: 0.7609 - accuracy: 0.7686INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 0.7602 - accuracy: 0.7689 -
val_loss: 0.8393 - val_accuracy: 0.7795
Epoch 25/30
61/63 [============================>.] - ETA: 0s - loss: 0.7458 - accuracy: 0.7740INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 30ms/step - loss: 0.7459 - accuracy: 0.7742 -
val_loss: 0.8200 - val_accuracy: 0.7860
Epoch 26/30
61/63 [============================>.] - ETA: 0s - loss: 0.7295 - accuracy: 0.7780INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 0.7287 - accuracy: 0.7784 -
val_loss: 0.8026 - val_accuracy: 0.7890
Epoch 27/30
61/63 [============================>.] - ETA: 0s - loss: 0.7161 - accuracy: 0.7824INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 0.7167 - accuracy: 0.7822 -
val_loss: 0.7883 - val_accuracy: 0.7952
Epoch 28/30
62/63 [============================>.] - ETA: 0s - loss: 0.7025 - accuracy: 0.7860INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 30ms/step - loss: 0.7027 - accuracy: 0.7859 -
val_loss: 0.7742 - val_accuracy: 0.8001
Epoch 29/30
61/63 [============================>.] - ETA: 0s - loss: 0.6906 - accuracy: 0.7909INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
63/63 [==============================] - 2s 28ms/step - loss: 0.6915 - accuracy: 0.7908 -
val_loss: 0.7655 - val_accuracy: 0.8042
Epoch 30/30
62/63 [============================>.] - ETA: 0s - loss: 0.6830 - accuracy: 0.7937INFO:te
nsorflow:Assets written to: ./cnn_best_model_checkpoint/checkpoint/assets
```

```
63/63 [==============================] - 2s 30ms/step - loss: 0.6825 - accuracy: 0.7938 -
val_loss: 0.7528 - val_accuracy: 0.8064
```

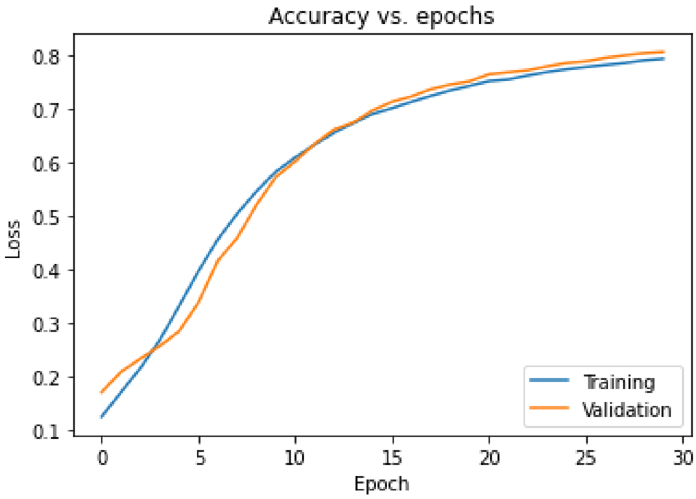
In [17]:

```
#Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and v
alidation sets.
from matplotlib import pyplot as plt
plt.plot(cnn_history.history['loss'])
plt.plot(cnn_history.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```



In [18]:

```
try:
    plt.plot(cnn_history.history['accuracy'])
    plt.plot(cnn_history.history['val_accuracy'])
except KeyError:
    plt.plot(cnn_history.history['acc'])
    plt.plot(cnn_history.history['val_acc'])
plt.title('Accuracy vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='lower right')
plt.show()
```



In [19]:

```
#Compute and display the loss and accuracy of the trained model on the test set.
test_loss, test_acc = cnn_model.evaluate(test_images,
                                          tf.keras.utils.to_categorical(test_labels-1, n
um_classes=10),
```

```
                                                        verbose=0)
print("Test loss: {:.3f}\nTest accuracy: {:.2f}%".format(test_loss, 100 * test_acc))
```

```
Test loss: 0.793
Test accuracy: 79.08%
```

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 [20]:

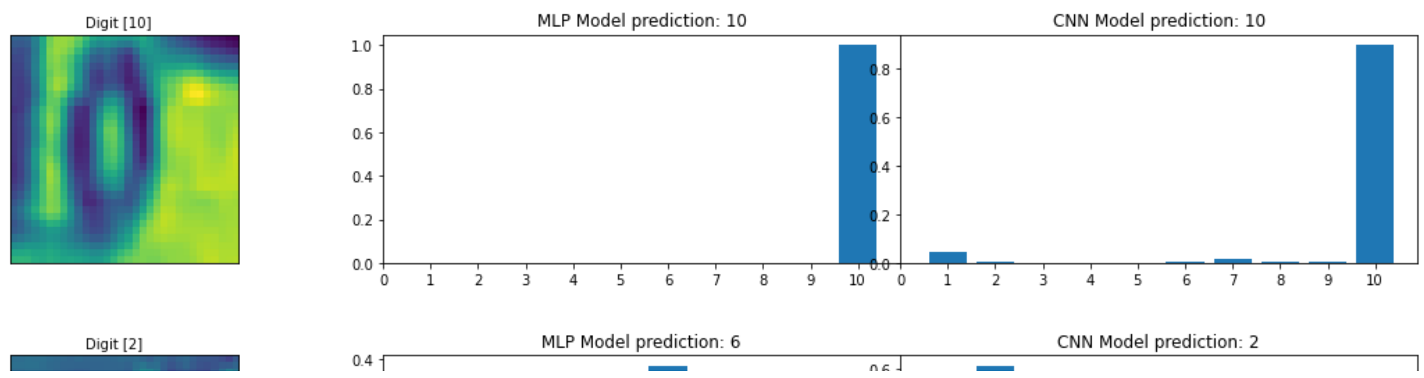
```
from tensorflow.keras.models import load_model
best_mlp_model = load_model("mlp_best_model_checkpoint/checkpoint")
```

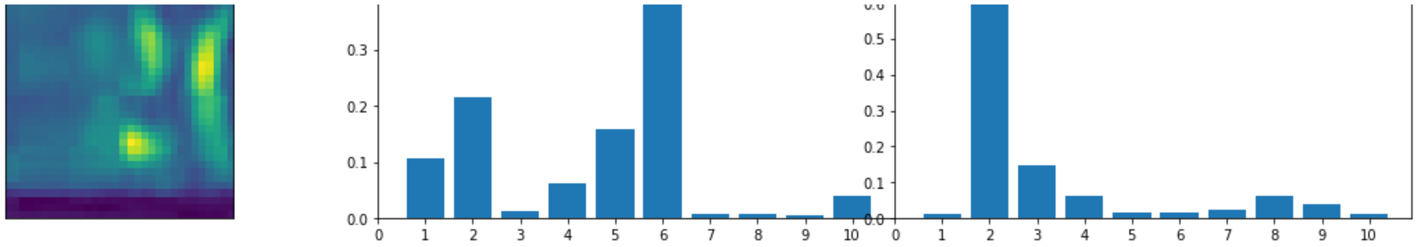
In [21]:

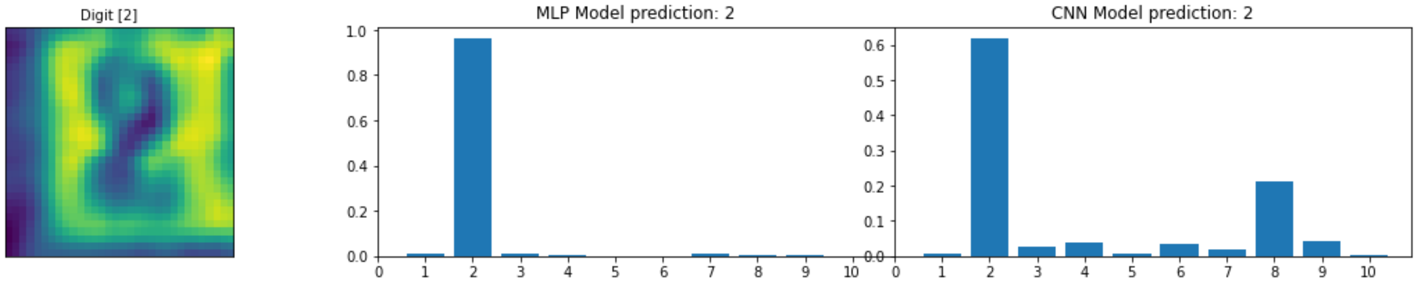
```
from tensorflow.keras.models import load_model
best_cnn_model = load_model("cnn_best_model_checkpoint/checkpoint")
```

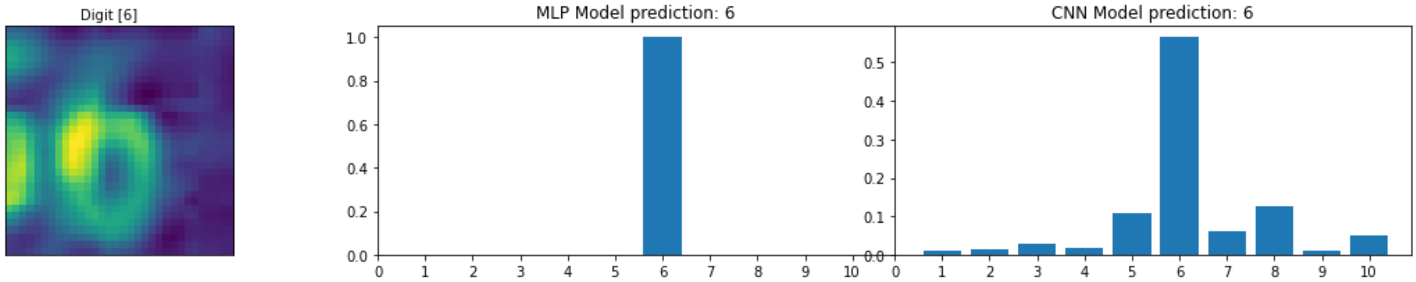
In [23]:

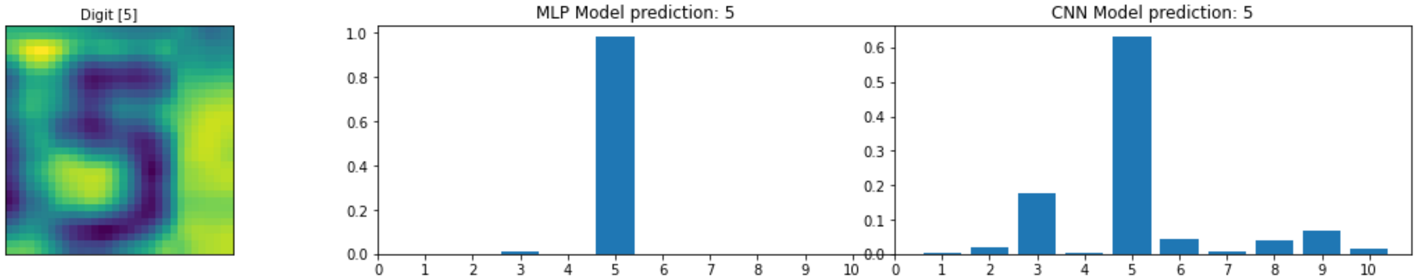
```
num_test_images = test_images.shape[0]

random_inx = np.random.choice(num_test_images, 5)
random_test_images = test_images[random_inx, ...]
random_test_labels = test_labels[random_inx, ...]


mlp_predictions = best_mlp_model.predict(random_test_images)
cnn_predictions = best_cnn_model.predict(random_test_images)

fig, axes = plt.subplots(5, 3, figsize=(20,20))
fig.subplots_adjust(hspace=0.4, wspace=0.)

for i, (mlp_prediction, cnn_prediction, image, label) in enumerate(zip(mlp_predictions,c
nn_predictions, random_test_images, random_test_labels)):
    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(len(mlp_prediction))+1, mlp_prediction)
    axes[i, 1].set_xticks(np.arange(len(mlp_prediction) + 1))
    axes[i, 1].set_title(f"MLP Model prediction: {np.argmax(mlp_prediction) + 1}")

    axes[i, 2].bar(np.arange(len(cnn_prediction))+1, cnn_prediction)
    axes[i, 2].set_xticks(np.arange(len(cnn_prediction) + 1))
    axes[i, 2].set_title(f"CNN Model prediction: {np.argmax(cnn_prediction) + 1}")

plt.show()
```






Digit [2]
MLP Model prediction: 2
CNN Model prediction: 2


Digit [6]
MLP Model prediction: 6
CNN Model prediction: 6


Digit [5]
MLP Model prediction: 5
CNN Model prediction: 5

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```