Capstone Project

February 24, 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 (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 [20]: import tensorflow as tf
         from scipy.io import loadmat
         import matplotlib.pyplot as plt
         %matplotlib inline
         import numpy as np
```

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 [3]:
```
# 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.

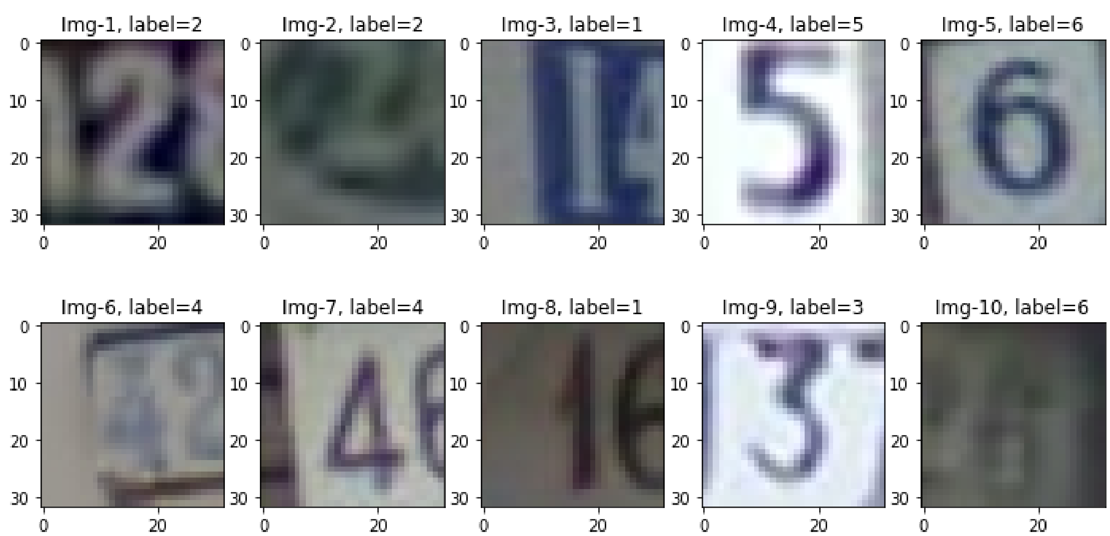
In [4]:
```
x_train = train['X']
y_train = train['y']
x_test = test['X']
y_test = test['y']
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[4]: ((32, 32, 3, 73257), (73257, 1), (32, 32, 3, 26032), (26032, 1))

In [5]:
```
x_train = np.transpose(x_train, (3, 0, 1, 2))
x_test = np.transpose(x_test, (3, 0, 1, 2))
x_train.shape, x_test.shape
```

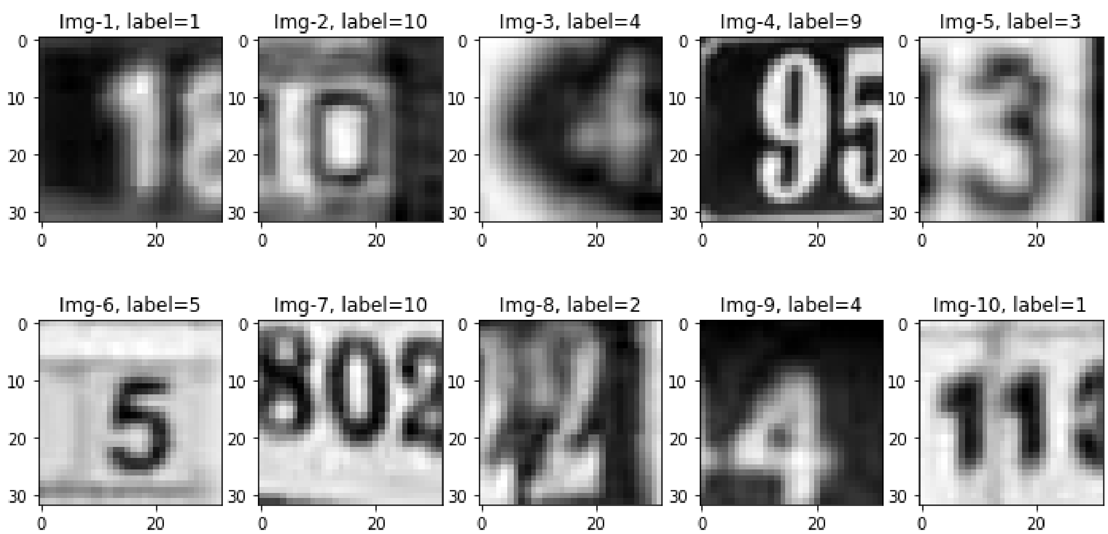
Out[5]: ((73257, 32, 32, 3), (26032, 32, 32, 3))

In [6]:
```
fig=plt.figure(figsize=(12,6))
columns = 5
rows = 2
for id in range(1, columns*rows +1):
    train_set = True if np.random.randint(2) == 1 else False
    if train_set:
        n = np.random.randint(x_train.shape[0])
        ax = fig.add_subplot(rows, columns, id)
        ax.title.set_text(f"Img-{id}, label={y_train[n][0]}")
        ax.imshow(x_train[n])
    else:
        n = np.random.randint(x_test.shape[0])
        ax = fig.add_subplot(rows, columns, id)
        ax.title.set_text(f"Img-{id}, label={y_test[n][0]}")
        ax.imshow(x_test[n])

plt.show()
```



In [7]:
```
x_train = np.mean(x_train, axis=3) / 255
x_test = np.mean(x_test, axis=3) / 255
```

In [8]:
```
fig=plt.figure(figsize=(12,6))
columns = 5
rows = 2
for id in range(1, columns*rows +1):
    train_set = True if np.random.randint(2) == 1 else False
    if train_set:
        n = np.random.randint(x_train.shape[0])
        ax = fig.add_subplot(rows, columns, id)
        ax.title.set_text(f"Img-{id}, label={y_train[n][0]}")
        ax.imshow(x_train[n], cmap='gray')
    else:
        n = np.random.randint(x_test.shape[0])
        ax = fig.add_subplot(rows, columns, id)
        ax.title.set_text(f"Img-{id}, label={y_test[n][0]}")
        ax.imshow(x_test[n], cmap='gray')

plt.show()
```



In [9]:
```
x_train = x_train.reshape(x_train.shape + (1,))
x_test = x_test.reshape(x_test.shape + (1,))
x_train.shape, x_test.shape
```

Out[9]: ((73257, 32, 32, 1), (26032, 32, 32, 1))

In [10]:
```
y_train= y_train.reshape(y_train.shape[0])
y_train= y_train-1
y_train[0:10]
```

Out[10]: array([0, 8, 1, 2, 1, 4, 8, 2, 2, 0], dtype=uint8)

In [11]:
```
y_test= y_test.reshape(y_test.shape[0])
y_test= y_test-1
y_test[0:10]
```

Out[11]:
```
array([4, 1, 0, 9, 5, 0, 8, 0, 0, 7], dtype=uint8)
```

In [12]:
```
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)
```

In [13]:
```
y_train.shape, y_test.shape
```

Out[13]:
```
((73257, 10), (26032, 10))
```

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 [14]:
```
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Activation
```

In [15]:
```
def get_model(input_shape):
    """
    This function should build a Sequential model according to the above specificatio
    weights are initialised by providing the input_shape argument in the first layer,
    function argument.
    Your function should return the model.
    """
    model = Sequential()
    model.add(keras.Input(shape=input_shape))
    model.add(Flatten())
    model.add(Dense(units=1024,activation='relu'))
    model.add(Dense(units=256,activation='relu'))
    model.add(Dense(units=128,activation='relu'))
    model.add(Dense(units=64,activation='relu'))
    model.add(Dense(units=32,activation='relu'))
```

```
    model.add(Dense(10, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics=['accuracy'])

    return model

model = get_model(x_train[0].shape)
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              1049600
_________________________________________________________________
dense_1 (Dense)              (None, 256)               262400
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_5 (Dense)              (None, 10)                330
=================================================================
Total params: 1,355,562
Trainable params: 1,355,562
Non-trainable params: 0
_________________________________________________________________
```

```
In [16]: early_stopping =  tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=5

         checkpoint_path = "checkpoints_best_only/checkpoint"
         checkpoint_best_only = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                                   save_freq='epoch',
                                                                   save_weights_only=True,
                                                                   save_best_only=True,
                                                                   monitor='val_accuracy',
                                                                   verbose=1)
         callbacks = [checkpoint_best_only, early_stopping]
```

```
In [16]: history = model.fit(x_train,y_train, validation_split=0.15, epochs=60, verbose=1, cal
```

```
Train on 62268 samples, validate on 10989 samples
Epoch 1/60
```

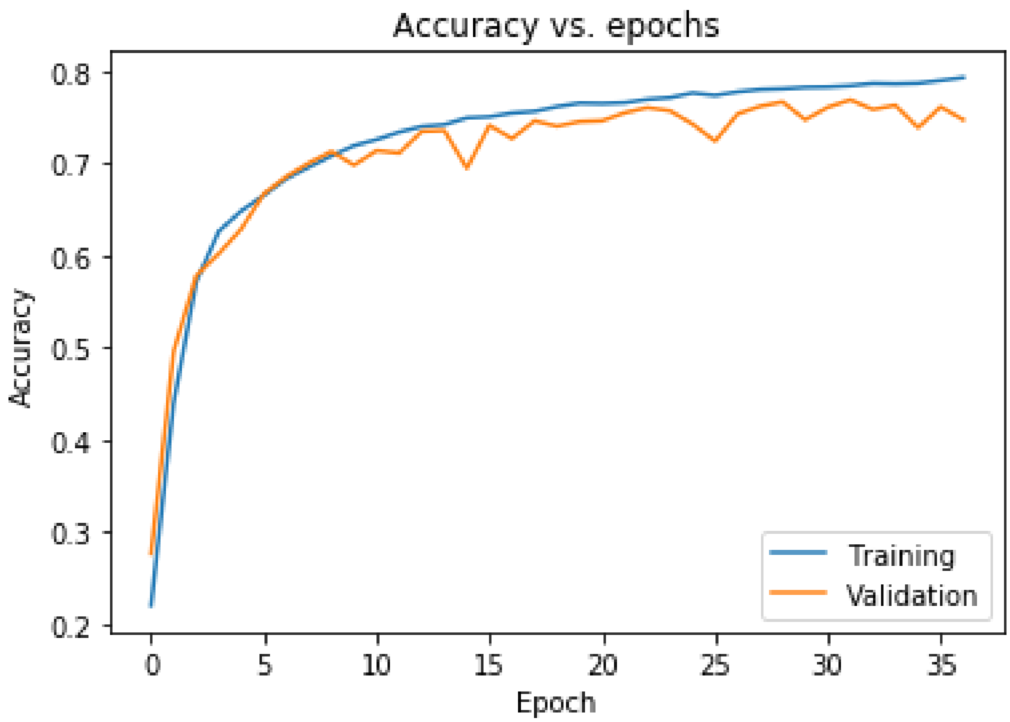
```
62208/62268 [============================>.] - ETA: 0s - loss: 2.1469 - accuracy: 0.2195
Epoch 00001: val_accuracy improved from -inf to 0.27609, saving model to checkpoints_best_only/
62268/62268 [==============================] - 109s 2ms/sample - loss: 2.1467 - accuracy: 0.219
Epoch 2/60
62208/62268 [============================>.] - ETA: 0s - loss: 1.5968 - accuracy: 0.4392
Epoch 00002: val_accuracy improved from 0.27609 to 0.49559, saving model to checkpoints_best_o
62268/62268 [==============================] - 106s 2ms/sample - loss: 1.5965 - accuracy: 0.439
Epoch 3/60
62240/62268 [============================>.] - ETA: 0s - loss: 1.2903 - accuracy: 0.5709
Epoch 00003: val_accuracy improved from 0.49559 to 0.57776, saving model to checkpoints_best_o
62268/62268 [==============================] - 108s 2ms/sample - loss: 1.2902 - accuracy: 0.570
Epoch 4/60
62208/62268 [============================>.] - ETA: 0s - loss: 1.1602 - accuracy: 0.6259
Epoch 00004: val_accuracy improved from 0.57776 to 0.60151, saving model to checkpoints_best_o
62268/62268 [==============================] - 107s 2ms/sample - loss: 1.1603 - accuracy: 0.625
Epoch 5/60
62176/62268 [============================>.] - ETA: 0s - loss: 1.0987 - accuracy: 0.6480
Epoch 00005: val_accuracy improved from 0.60151 to 0.62845, saving model to checkpoints_best_o
62268/62268 [==============================] - 107s 2ms/sample - loss: 1.0987 - accuracy: 0.648
Epoch 6/60
62240/62268 [============================>.] - ETA: 0s - loss: 1.0489 - accuracy: 0.6650
Epoch 00006: val_accuracy improved from 0.62845 to 0.66667, saving model to checkpoints_best_o
62268/62268 [==============================] - 107s 2ms/sample - loss: 1.0488 - accuracy: 0.665
Epoch 7/60
62208/62268 [============================>.] - ETA: 0s - loss: 0.9993 - accuracy: 0.6831
Epoch 00007: val_accuracy improved from 0.66667 to 0.68541, saving model to checkpoints_best_o
62268/62268 [==============================] - 107s 2ms/sample - loss: 0.9994 - accuracy: 0.683
Epoch 8/60
62208/62268 [============================>.] - ETA: 0s - loss: 0.9567 - accuracy: 0.6957
Epoch 00008: val_accuracy improved from 0.68541 to 0.70015, saving model to checkpoints_best_o
62268/62268 [==============================] - 107s 2ms/sample - loss: 0.9568 - accuracy: 0.695
Epoch 9/60
62208/62268 [============================>.] - ETA: 0s - loss: 0.9209 - accuracy: 0.7078
Epoch 00009: val_accuracy improved from 0.70015 to 0.71262, saving model to checkpoints_best_o
62268/62268 [==============================] - 103s 2ms/sample - loss: 0.9210 - accuracy: 0.707
Epoch 10/60
62208/62268 [============================>.] - ETA: 0s - loss: 0.8911 - accuracy: 0.7190
Epoch 00010: val_accuracy did not improve from 0.71262
62268/62268 [==============================] - 104s 2ms/sample - loss: 0.8913 - accuracy: 0.719
Epoch 11/60
62176/62268 [============================>.] - ETA: 0s - loss: 0.8667 - accuracy: 0.7255
Epoch 00011: val_accuracy improved from 0.71262 to 0.71353, saving model to checkpoints_best_o
62268/62268 [==============================] - 104s 2ms/sample - loss: 0.8668 - accuracy: 0.725
Epoch 12/60
62240/62268 [============================>.] - ETA: 0s - loss: 0.8404 - accuracy: 0.7337
Epoch 00012: val_accuracy did not improve from 0.71353
62268/62268 [==============================] - 104s 2ms/sample - loss: 0.8405 - accuracy: 0.733
Epoch 13/60
```

```
62176/62268 [============================>.] - ETA: 0s - loss: 0.8264 - accuracy: 0.7398
Epoch 00013: val_accuracy improved from 0.71353 to 0.73446, saving model to checkpoints_best_or
62268/62268 [==============================] - 104s 2ms/sample - loss: 0.8264 - accuracy: 0.739
Epoch 14/60
62176/62268 [============================>.] - ETA: 0s - loss: 0.8162 - accuracy: 0.7418
Epoch 00014: val_accuracy improved from 0.73446 to 0.73537, saving model to checkpoints_best_or
62268/62268 [==============================] - 99s 2ms/sample - loss: 0.8162 - accuracy: 0.7418
Epoch 15/60
62240/62268 [============================>.] - ETA: 0s - loss: 0.7924 - accuracy: 0.7492
Epoch 00015: val_accuracy did not improve from 0.73537
62268/62268 [==============================] - 99s 2ms/sample - loss: 0.7925 - accuracy: 0.7492
Epoch 16/60
62208/62268 [============================>.] - ETA: 0s - loss: 0.7879 - accuracy: 0.7503
Epoch 00016: val_accuracy improved from 0.73537 to 0.74101, saving model to checkpoints_best_or
62268/62268 [==============================] - 99s 2ms/sample - loss: 0.7878 - accuracy: 0.7503
Epoch 17/60
62240/62268 [============================>.] - ETA: 0s - loss: 0.7739 - accuracy: 0.7544
Epoch 00017: val_accuracy did not improve from 0.74101
62268/62268 [==============================] - 98s 2ms/sample - loss: 0.7738 - accuracy: 0.7544
Epoch 18/60
62240/62268 [============================>.] - ETA: 0s - loss: 0.7679 - accuracy: 0.7563
Epoch 00018: val_accuracy improved from 0.74101 to 0.74584, saving model to checkpoints_best_or
62268/62268 [==============================] - 98s 2ms/sample - loss: 0.7679 - accuracy: 0.7562
Epoch 19/60
62240/62268 [============================>.] - ETA: 0s - loss: 0.7571 - accuracy: 0.7614
Epoch 00019: val_accuracy did not improve from 0.74584
62268/62268 [==============================] - 98s 2ms/sample - loss: 0.7571 - accuracy: 0.7614
Epoch 20/60
62208/62268 [============================>.] - ETA: 0s - loss: 0.7429 - accuracy: 0.7652
Epoch 00020: val_accuracy did not improve from 0.74584
62268/62268 [==============================] - 98s 2ms/sample - loss: 0.7432 - accuracy: 0.7652
Epoch 21/60
62240/62268 [============================>.] - ETA: 0s - loss: 0.7423 - accuracy: 0.7648
Epoch 00021: val_accuracy improved from 0.74584 to 0.74602, saving model to checkpoints_best_or
62268/62268 [==============================] - 99s 2ms/sample - loss: 0.7424 - accuracy: 0.7647
Epoch 22/60
62208/62268 [============================>.] - ETA: 0s - loss: 0.7328 - accuracy: 0.7656
Epoch 00022: val_accuracy improved from 0.74602 to 0.75466, saving model to checkpoints_best_or
62268/62268 [==============================] - 98s 2ms/sample - loss: 0.7327 - accuracy: 0.7656
Epoch 23/60
62176/62268 [============================>.] - ETA: 0s - loss: 0.7231 - accuracy: 0.7693
Epoch 00023: val_accuracy improved from 0.75466 to 0.76031, saving model to checkpoints_best_or
62268/62268 [==============================] - 98s 2ms/sample - loss: 0.7232 - accuracy: 0.7693
Epoch 24/60
62208/62268 [============================>.] - ETA: 0s - loss: 0.7176 - accuracy: 0.7710
Epoch 00024: val_accuracy did not improve from 0.76031
62268/62268 [==============================] - 98s 2ms/sample - loss: 0.7174 - accuracy: 0.7711
Epoch 25/60
```

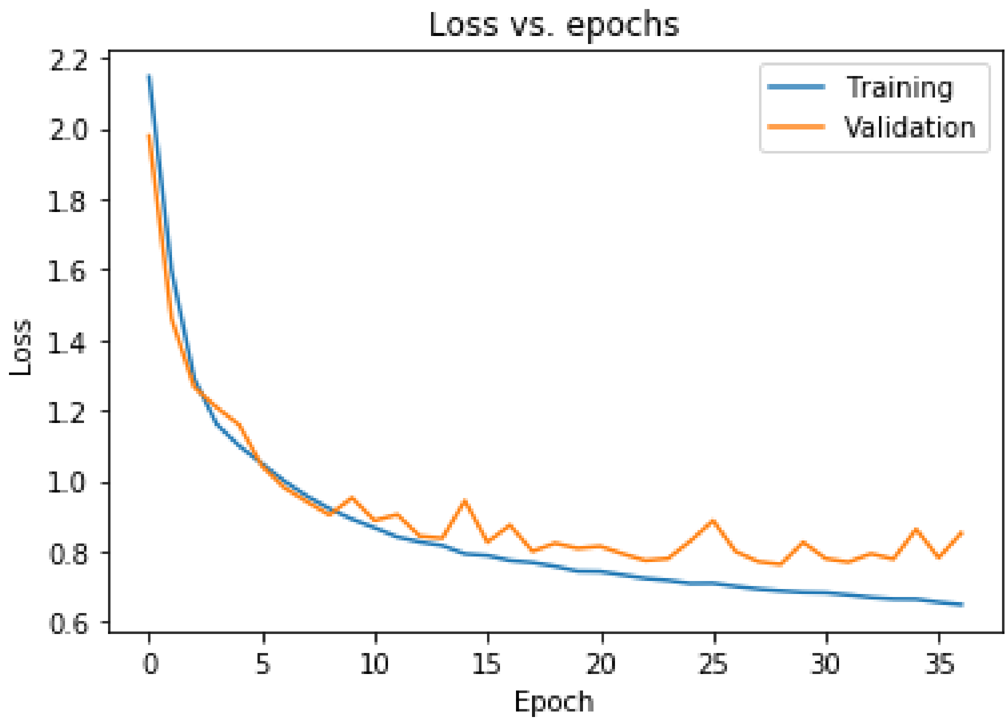
```
62240/62268 [============================>.] - ETA: 0s - loss: 0.7090 - accuracy: 0.7762
Epoch 00025: val_accuracy did not improve from 0.76031
62268/62268 [==============================] - 98s 2ms/sample - loss: 0.7089 - accuracy: 0.7762
Epoch 26/60
62208/62268 [============================>.] - ETA: 0s - loss: 0.7093 - accuracy: 0.7735
Epoch 00026: val_accuracy did not improve from 0.76031
62268/62268 [==============================] - 99s 2ms/sample - loss: 0.7094 - accuracy: 0.7735
Epoch 27/60
62240/62268 [============================>.] - ETA: 0s - loss: 0.7006 - accuracy: 0.7775
Epoch 00027: val_accuracy did not improve from 0.76031
62268/62268 [==============================] - 98s 2ms/sample - loss: 0.7006 - accuracy: 0.7775
Epoch 28/60
62208/62268 [============================>.] - ETA: 0s - loss: 0.6930 - accuracy: 0.7800
Epoch 00028: val_accuracy improved from 0.76031 to 0.76176, saving model to checkpoints_best_or
62268/62268 [==============================] - 99s 2ms/sample - loss: 0.6931 - accuracy: 0.7800
Epoch 29/60
62240/62268 [============================>.] - ETA: 0s - loss: 0.6878 - accuracy: 0.7808
Epoch 00029: val_accuracy improved from 0.76176 to 0.76649, saving model to checkpoints_best_or
62268/62268 [==============================] - 99s 2ms/sample - loss: 0.6878 - accuracy: 0.7808
Epoch 30/60
62176/62268 [============================>.] - ETA: 0s - loss: 0.6838 - accuracy: 0.7822
Epoch 00030: val_accuracy did not improve from 0.76649
62268/62268 [==============================] - 104s 2ms/sample - loss: 0.6838 - accuracy: 0.782
Epoch 31/60
62240/62268 [============================>.] - ETA: 0s - loss: 0.6827 - accuracy: 0.7826
Epoch 00031: val_accuracy did not improve from 0.76649
62268/62268 [==============================] - 102s 2ms/sample - loss: 0.6826 - accuracy: 0.782
Epoch 32/60
62208/62268 [============================>.] - ETA: 0s - loss: 0.6764 - accuracy: 0.7840
Epoch 00032: val_accuracy improved from 0.76649 to 0.76877, saving model to checkpoints_best_or
62268/62268 [==============================] - 104s 2ms/sample - loss: 0.6763 - accuracy: 0.783
Epoch 33/60
62240/62268 [============================>.] - ETA: 0s - loss: 0.6689 - accuracy: 0.7864
Epoch 00033: val_accuracy did not improve from 0.76877
62268/62268 [==============================] - 103s 2ms/sample - loss: 0.6691 - accuracy: 0.786
Epoch 34/60
62240/62268 [============================>.] - ETA: 0s - loss: 0.6648 - accuracy: 0.7860
Epoch 00034: val_accuracy did not improve from 0.76877
62268/62268 [==============================] - 100s 2ms/sample - loss: 0.6648 - accuracy: 0.786
Epoch 35/60
62208/62268 [============================>.] - ETA: 0s - loss: 0.6643 - accuracy: 0.7867
Epoch 00035: val_accuracy did not improve from 0.76877
62268/62268 [==============================] - 99s 2ms/sample - loss: 0.6643 - accuracy: 0.7866
Epoch 36/60
62176/62268 [============================>.] - ETA: 0s - loss: 0.6555 - accuracy: 0.7896
Epoch 00036: val_accuracy did not improve from 0.76877
62268/62268 [==============================] - 98s 2ms/sample - loss: 0.6555 - accuracy: 0.7896
Epoch 37/60
```

```
62240/62268 [============================>.] - ETA: 0s - loss: 0.6488 - accuracy: 0.7928
Epoch 00037: val_accuracy did not improve from 0.76877
62268/62268 [==============================] - 98s 2ms/sample - loss: 0.6488 - accuracy: 0.792
```

In [17]:
```python
try:
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
except KeyError:
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
plt.title('Accuracy vs. epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='lower right')
plt.show()
```



In [18]:
```python
plt.plot(history.history['loss'])
plt.plot(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 [19]: test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
         print("Test loss: {:.3f}\nTest accuracy: {:.2f}%".format(test_loss, 100 * test_acc))
```

```
Test loss: 0.955
Test accuracy: 72.27%
```

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 [17]:
```python
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, BatchNormalization
```

In [18]:
```python
def get_cnn_model(input_shape):
    """
    This function should build a Sequential model according to the above specificatio
    weights are initialised by providing the input_shape argument in the first layer,
    function argument.
    Your function should return the model.
    """
    model = Sequential([
        Conv2D(name="conv_1", filters=32, kernel_size=(3,3), activation='relu', paddi
        MaxPooling2D(name="pool_1", pool_size=(2,2)),
        Conv2D(name="conv_2", filters=16, kernel_size=(3,3), activation='relu', paddi
        MaxPooling2D(name="pool_2", pool_size=(4,4)),
        Flatten(name="flatten"),
        Dense(name="dense_1", units=32, activation='relu'),
        Dense(name="dense_2", units=10, activation='softmax')
    ])

    model.compile(loss='categorical_crossentropy',
              optimizer="adam",
              metrics=['accuracy'])

    return model


cnn_model = get_cnn_model(x_train[0].shape)
cnn_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_1 (Conv2D)              (None, 32, 32, 32)        320
_________________________________________________________________
pool_1 (MaxPooling2D)        (None, 16, 16, 32)        0
_________________________________________________________________
conv_2 (Conv2D)              (None, 16, 16, 16)        4624
_________________________________________________________________
pool_2 (MaxPooling2D)        (None, 4, 4, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                8224
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
=================================================================
Total params: 13,498
```

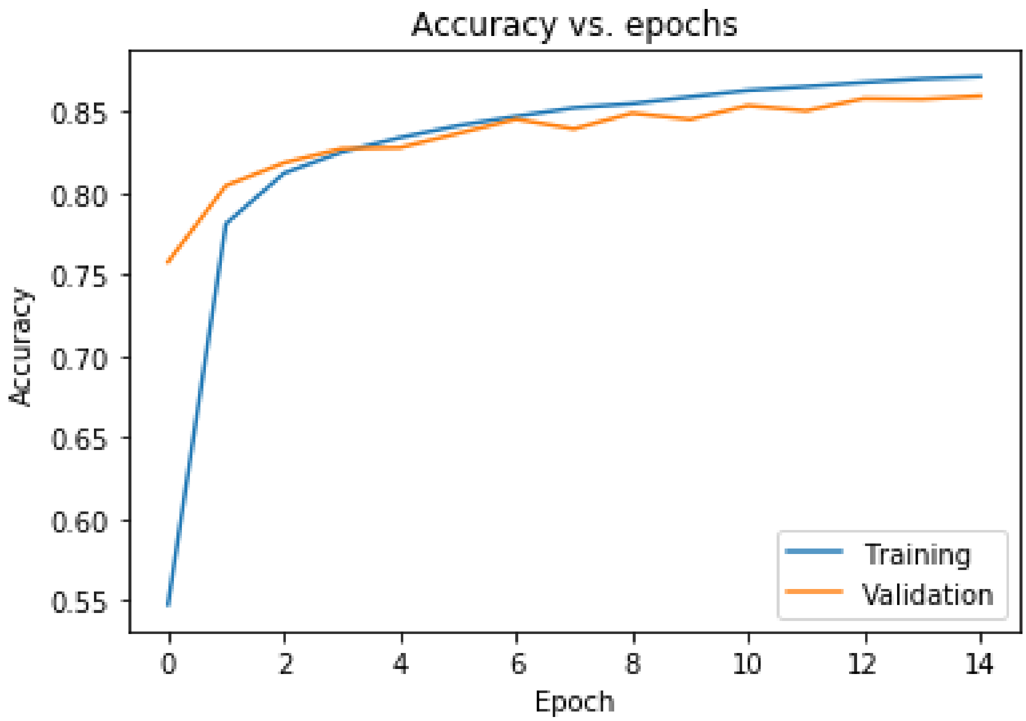
```
Trainable params: 13,498
Non-trainable params: 0
_________________________________________________________________
```

```
In [19]: early_stopping =  tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=5)

         cnn_checkpoint_path = "cnn_checkpoints_best_only/checkpoint"
         cnn_checkpoint_best_only = tf.keras.callbacks.ModelCheckpoint(filepath=cnn_checkpoint_
                                                                       save_freq='epoch',
                                                                       save_weights_only=True,
                                                                       save_best_only=True,
                                                                       monitor='val_accuracy',
                                                                       verbose=1)
         callbacks = [cnn_checkpoint_best_only, early_stopping]

         cnn_history = cnn_model.fit(x_train, y_train, epochs=15, validation_split=0.15, callba
```

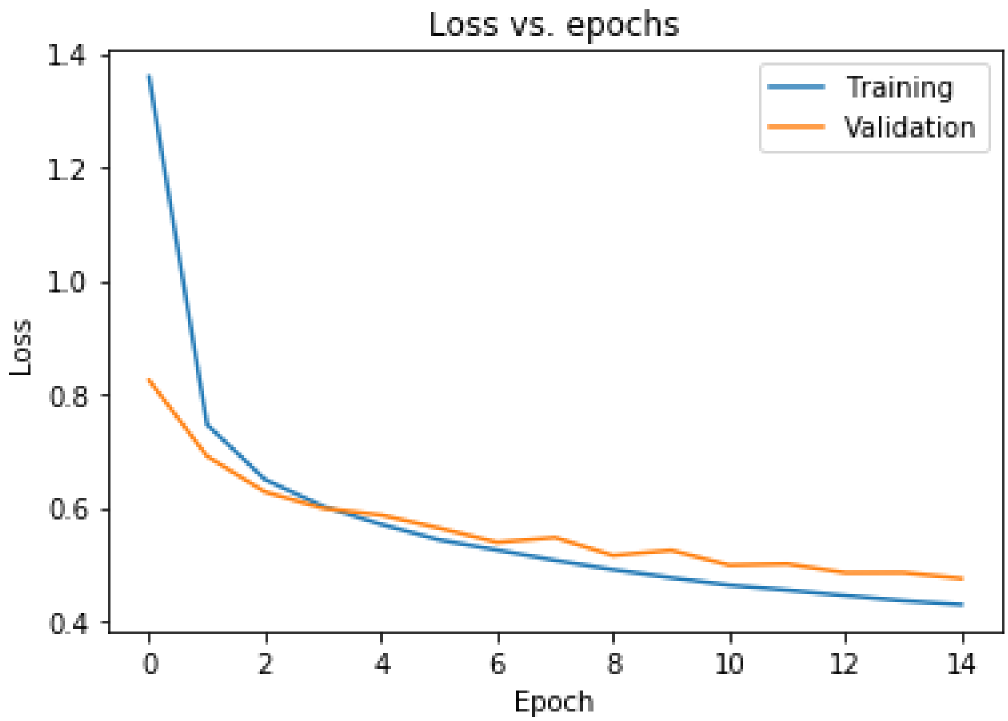
```
Train on 62268 samples, validate on 10989 samples
Epoch 1/15
62240/62268 [============================>.] - ETA: 0s - loss: 1.3598 - accuracy: 0.5474
Epoch 00001: val_accuracy improved from -inf to 0.75758, saving model to cnn_checkpoints_best_
62268/62268 [==============================] - 254s 4ms/sample - loss: 1.3595 - accuracy: 0.54
Epoch 2/15
62240/62268 [============================>.] - ETA: 0s - loss: 0.7463 - accuracy: 0.7809
Epoch 00002: val_accuracy improved from 0.75758 to 0.80435, saving model to cnn_checkpoints_bes
62268/62268 [==============================] - 250s 4ms/sample - loss: 0.7463 - accuracy: 0.78
Epoch 3/15
62240/62268 [============================>.] - ETA: 0s - loss: 0.6497 - accuracy: 0.8119
Epoch 00003: val_accuracy improved from 0.80435 to 0.81836, saving model to cnn_checkpoints_bes
62268/62268 [==============================] - 248s 4ms/sample - loss: 0.6497 - accuracy: 0.81
Epoch 4/15
62240/62268 [============================>.] - ETA: 0s - loss: 0.6025 - accuracy: 0.8250
Epoch 00004: val_accuracy improved from 0.81836 to 0.82710, saving model to cnn_checkpoints_bes
62268/62268 [==============================] - 260s 4ms/sample - loss: 0.6025 - accuracy: 0.82
Epoch 5/15
62240/62268 [============================>.] - ETA: 0s - loss: 0.5707 - accuracy: 0.8337
Epoch 00005: val_accuracy improved from 0.82710 to 0.82765, saving model to cnn_checkpoints_bes
62268/62268 [==============================] - 248s 4ms/sample - loss: 0.5706 - accuracy: 0.83
Epoch 6/15
62240/62268 [============================>.] - ETA: 0s - loss: 0.5430 - accuracy: 0.8411
Epoch 00006: val_accuracy improved from 0.82765 to 0.83629, saving model to cnn_checkpoints_bes
62268/62268 [==============================] - 248s 4ms/sample - loss: 0.5432 - accuracy: 0.84
Epoch 7/15
62240/62268 [============================>.] - ETA: 0s - loss: 0.5251 - accuracy: 0.8468
Epoch 00007: val_accuracy improved from 0.83629 to 0.84503, saving model to cnn_checkpoints_bes
62268/62268 [==============================] - 247s 4ms/sample - loss: 0.5252 - accuracy: 0.84
Epoch 8/15
```

```
62240/62268 [============================>.] - ETA: 0s - loss: 0.5070 - accuracy: 0.8521
Epoch 00008: val_accuracy did not improve from 0.84503
62268/62268 [==============================] - 246s 4ms/sample - loss: 0.5070 - accuracy: 0.852
Epoch 9/15
62240/62268 [============================>.] - ETA: 0s - loss: 0.4906 - accuracy: 0.8546
Epoch 00009: val_accuracy improved from 0.84503 to 0.84876, saving model to cnn_checkpoints_bes
62268/62268 [==============================] - 248s 4ms/sample - loss: 0.4905 - accuracy: 0.854
Epoch 10/15
62240/62268 [============================>.] - ETA: 0s - loss: 0.4767 - accuracy: 0.8587
Epoch 00010: val_accuracy did not improve from 0.84876
62268/62268 [==============================] - 271s 4ms/sample - loss: 0.4766 - accuracy: 0.858
Epoch 11/15
62240/62268 [============================>.] - ETA: 0s - loss: 0.4634 - accuracy: 0.8628
Epoch 00011: val_accuracy improved from 0.84876 to 0.85340, saving model to cnn_checkpoints_bes
62268/62268 [==============================] - 250s 4ms/sample - loss: 0.4635 - accuracy: 0.862
Epoch 12/15
62240/62268 [============================>.] - ETA: 0s - loss: 0.4548 - accuracy: 0.8651
Epoch 00012: val_accuracy did not improve from 0.85340
62268/62268 [==============================] - 249s 4ms/sample - loss: 0.4548 - accuracy: 0.865
Epoch 13/15
62240/62268 [============================>.] - ETA: 0s - loss: 0.4452 - accuracy: 0.8676
Epoch 00013: val_accuracy improved from 0.85340 to 0.85777, saving model to cnn_checkpoints_bes
62268/62268 [==============================] - 249s 4ms/sample - loss: 0.4451 - accuracy: 0.867
Epoch 14/15
62240/62268 [============================>.] - ETA: 0s - loss: 0.4361 - accuracy: 0.8698
Epoch 00014: val_accuracy did not improve from 0.85777
62268/62268 [==============================] - 246s 4ms/sample - loss: 0.4361 - accuracy: 0.869
Epoch 15/15
62240/62268 [============================>.] - ETA: 0s - loss: 0.4293 - accuracy: 0.8711
Epoch 00015: val_accuracy improved from 0.85777 to 0.85922, saving model to cnn_checkpoints_bes
62268/62268 [==============================] - 266s 4ms/sample - loss: 0.4292 - accuracy: 0.871
```

In [21]:
```python
try:
    plt.plot(cnn_history.history['accuracy'])
    plt.plot(cnn_history.history['val_accuracy'])
except KeyError:
    plt.plot(cnn_.history['acc'])
    plt.plot(cnn_.history['val_acc'])
plt.title('Accuracy vs. epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='lower right')
plt.show()
```




```
In [22]: 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 [23]: cnn_test_loss, cnn_test_acc = cnn_model.evaluate(x_test, y_test, verbose=0)
         print("Test loss: {:.3f}\nTest accuracy: {:.2f}%".format(cnn_test_loss, 100 * cnn_tes
```

```
Test loss: 0.496
Test accuracy: 85.79%
```

1.5 4. Get model predictions

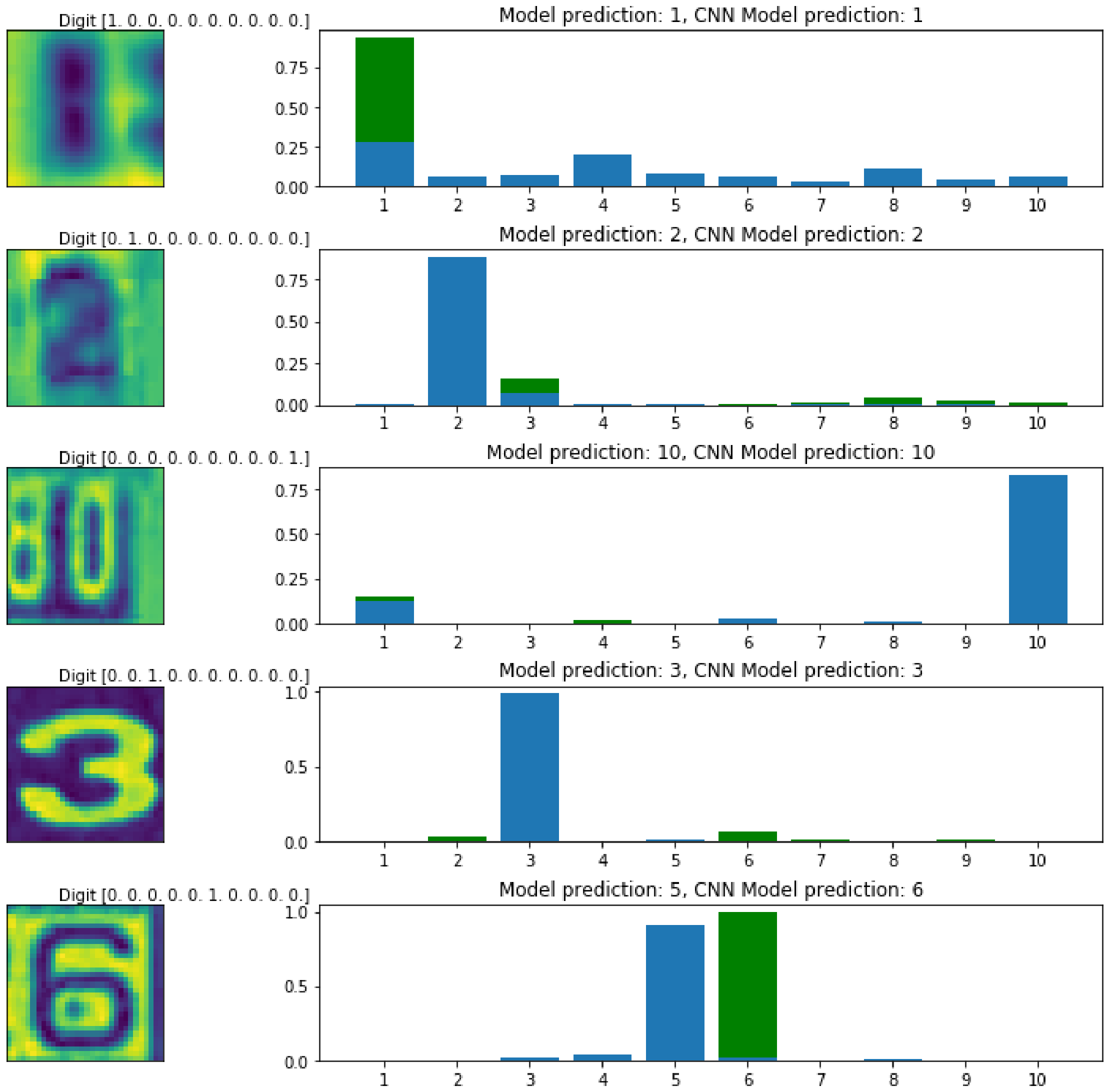
- Load the best weights for the MLP and CNN models that you saved during the training run.
- Randomly select 5 images and corresponding labels from the test set and display the images with their labels.
- Alongside the image and label, show each model's predictive distribution as a bar chart, and the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

```
In [24]: model.load_weights(checkpoint_path)
         cnn_model.load_weights(cnn_checkpoint_path)
```

```
Out[24]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fa6100d99e8>
```

```
In [57]: 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.predict(random_test_images)
cnn_predictions = cnn_model.predict(random_test_images)

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

for i, (cnn_prediction, prediction, image, label) in enumerate(zip(cnn_predictions, pr
    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(cnn_prediction))+1, cnn_prediction, color="green")
    axes[i, 1].bar(np.arange(len(prediction))+1, prediction)
    axes[i, 1].set_xticks(np.arange(len(prediction))+1)
    axes[i, 1].set_title(f"Model prediction: {np.argmax(prediction)+1}, CNN Model pred

plt.show()
```


Digit [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Model prediction: 1, CNN Model prediction: 1
Digit [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
Model prediction: 2, CNN Model prediction: 2
Digit [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
Model prediction: 10, CNN Model prediction: 10
Digit [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
Model prediction: 3, CNN Model prediction: 3
Digit [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
Model prediction: 5, CNN Model prediction: 6

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```