Capstone Project

July 8, 2020

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 [85]: 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 [86]: # 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 [87]: import numpy as np
         import matplotlib.pyplot as plt
         from tensorflow.keras import Sequential
         from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Dropout, BatchNormali
         from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

```
In [88]: x_train = train['X']
         y_train = train['y']
         x_test = test['X']
         y_test = test['y']

         print('X_train shape is ', x_train.shape)
         print('y_train shape is ', y_train.shape)
         print('X_test shape is ', x_test.shape)
         print('y_test shape is ', y_test.shape)
```

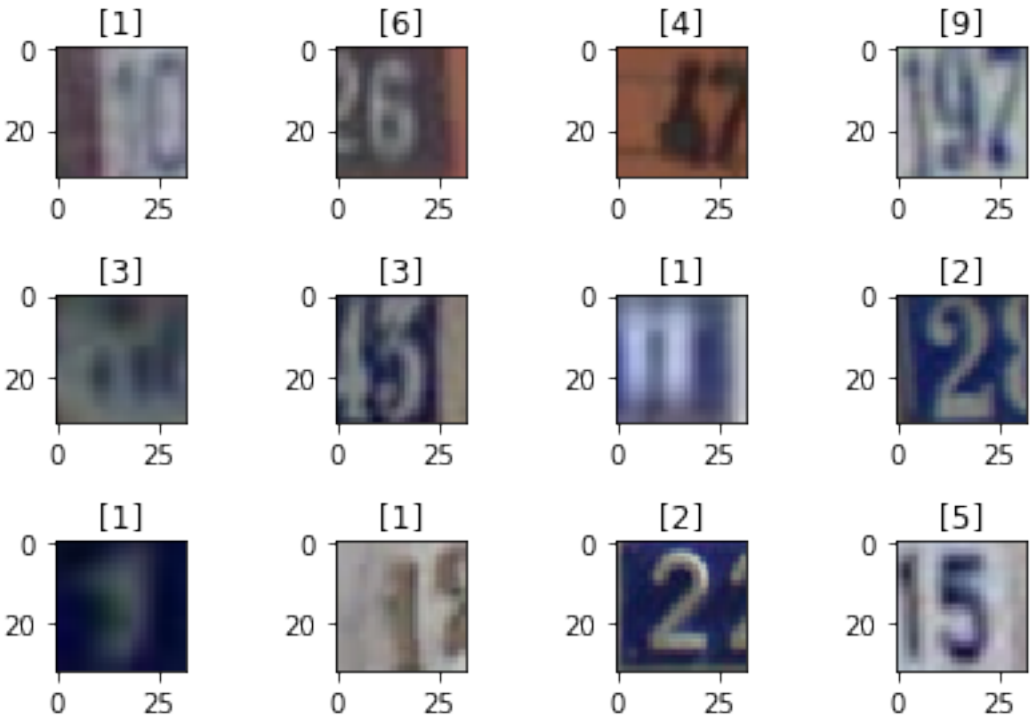
```
X_train shape is  (32, 32, 3, 73257)
y_train shape is  (73257, 1)
X_test shape is  (32, 32, 3, 26032)
y_test shape is  (26032, 1)
```

To correct shape, use `np.moveaxis` to move the last axis (axis = -1) to first position (index = 0)

```
In [89]: x_train = np.moveaxis(x_train,-1,0)
         x_test = np.moveaxis(x_test, -1, 0)

         print('X_train shape after reshaped is ', x_train.shape)
         print('X_test shape after reshaped is ', x_test.shape)
```

```
X_train shape after reshaped is  (73257, 32, 32, 3)
X_test shape after reshaped is  (26032, 32, 32, 3)
```

```
In [90]: fig = plt.figure()
         for i in range(12):
             idx = np.random.choice(x_train.shape[0])
             plt.subplot(3,4,i+1)
             plt.imshow(x_train[idx])
             plt.title(y_train[idx])
         fig.tight_layout()
         plt.show()
```

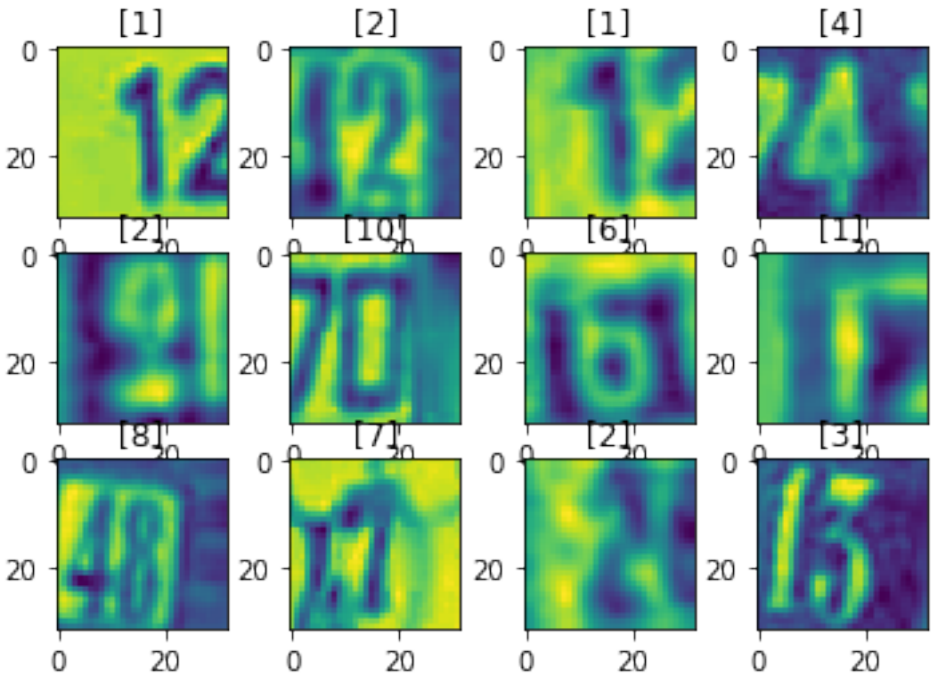


Convert the training and test images to grayscale by taking the average across all colour channels for each pixel. Hint: retain the channel dimension, which will now have size 1.

```
In [91]: x_train = np.mean(x_train, axis=3)
         x_test = np.mean(x_test, axis=3)
```

Select a random sample of the grayscale images and corresponding labels from the dataset (at least 10), and display them in a figure.

```
In [92]: plt.figure()
         for i in range(12):
             idx = np.random.choice(x_train.shape[0])
             plt.subplot(3,4,i+1)
             plt.imshow(x_train[idx])
             plt.title(y_train[idx])
         fig.tight_layout()
         plt.show()
```




```
In [93]: x_train = x_train[..., np.newaxis]/255.0
         x_test = x_test[..., np.newaxis]/255.0
         print('x_train shape is ', x_train.shape)
```

```
x_train shape is  (73257, 32, 32, 1)
```

```
In [94]: y_train = y_train - 1
         y_test = y_test - 1
```

```
In [95]: print(x_train.shape)
```

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

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 [96]: from tensorflow.keras.regularizers import l1

         def MLP_neuralnetwork(input_shape):
             model = Sequential()
             model.add(Flatten(input_shape=input_shape))
             model.add(Dense(256, activation='relu'))
             #model.add(Dense(1024, activation='relu'))
             model.add(Dense(256, activation='relu'))
             #model.add(Dense(, activation='relu'))
             model.add(Dense(10, activation='softmax'))
             return model

In [97]: model = MLP_neuralnetwork((x_train.shape[1:]))
         model.summary()

Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_4 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_12 (Dense)             (None, 256)               262400
_________________________________________________________________
dense_13 (Dense)             (None, 256)               65792
_________________________________________________________________
dense_14 (Dense)             (None, 10)                2570
=================================================================
Total params: 330,762
Trainable params: 330,762
Non-trainable params: 0
_________________________________________________________________
```

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).

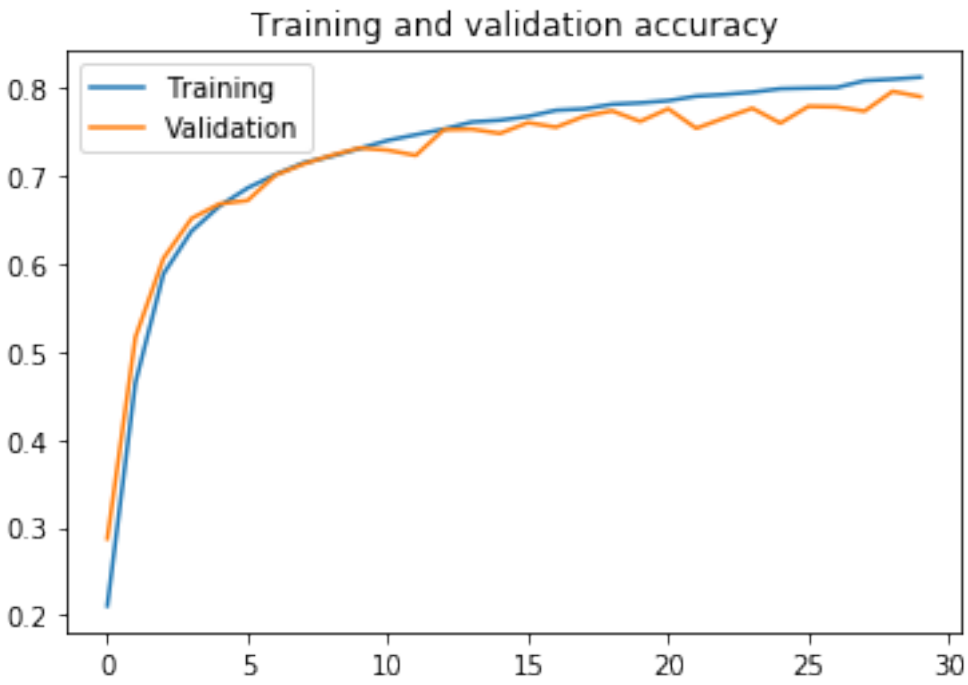
Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and validation sets.

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

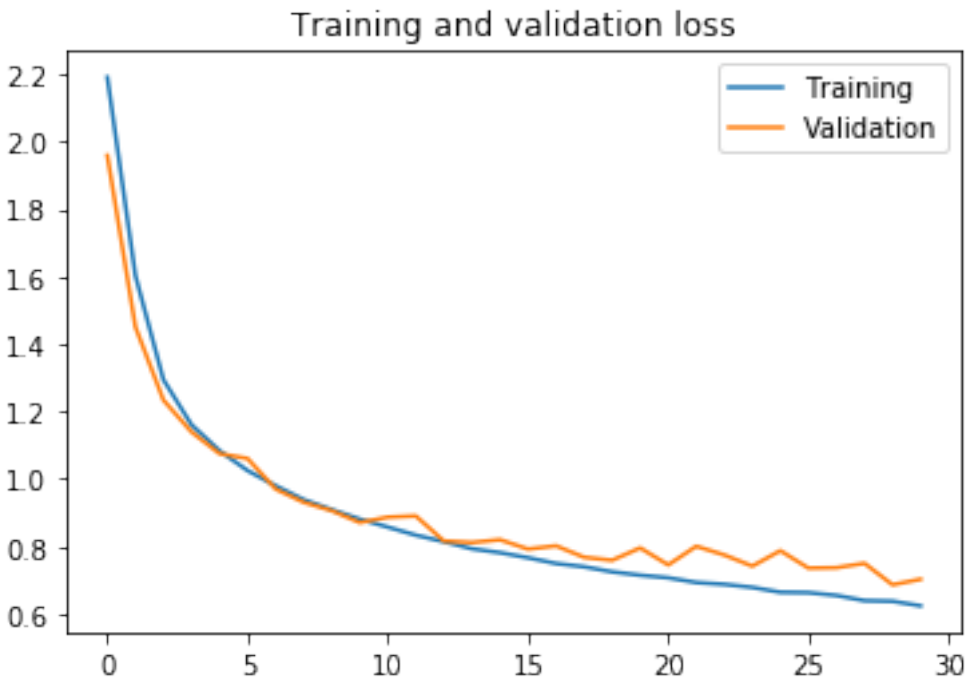
```
In [14]: model.compile(optimizer='adam',
                       #loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                       loss = 'sparse_categorical_crossentropy',
                       metrics=['accuracy'])

In [ ]: def checkpoint():
            cp = ModelCheckpoint('checkpoints/checkpoint.h5',
                                 save_best_only=True,
                                 save_weights_only=True)
            return cp

        def earlystopping():
            es = EarlyStopping(monitor='val_accuracy', patience=3)
            return es

In [16]: checkpoint = checkpoint()
         early = earlystopping()

In [17]: history = model.fit(x_train, y_train,
                             epochs=30,
                             callbacks=[checkpoint, early],
                             validation_split = 0.3,
                              batch_size=256,
                             verbose=1)

Train on 51279 samples, validate on 21978 samples
Epoch 1/30
51279/51279 [==============================] - 18s 347us/sample - loss: 2.1911 - accuracy: 0.2
Epoch 2/30
51279/51279 [==============================] - 16s 317us/sample - loss: 1.6036 - accuracy: 0.4
Epoch 3/30
51279/51279 [==============================] - 17s 328us/sample - loss: 1.2943 - accuracy: 0.5
Epoch 4/30
51279/51279 [==============================] - 17s 329us/sample - loss: 1.1608 - accuracy: 0.6
Epoch 5/30
51279/51279 [==============================] - 17s 330us/sample - loss: 1.0838 - accuracy: 0.6
Epoch 6/30
51279/51279 [==============================] - 17s 327us/sample - loss: 1.0249 - accuracy: 0.6
Epoch 7/30
51279/51279 [==============================] - 17s 331us/sample - loss: 0.9800 - accuracy: 0.7
Epoch 8/30
51279/51279 [==============================] - 17s 331us/sample - loss: 0.9387 - accuracy: 0.7
Epoch 9/30
51279/51279 [==============================] - 17s 332us/sample - loss: 0.9092 - accuracy: 0.7
Epoch 10/30
51279/51279 [==============================] - 17s 322us/sample - loss: 0.8810 - accuracy: 0.7
Epoch 11/30
51279/51279 [==============================] - 16s 306us/sample - loss: 0.8574 - accuracy: 0.7
Epoch 12/30
```

```
51279/51279 [==============================] - 16s 304us/sample - loss: 0.8334 - accuracy: 0.74
Epoch 13/30
51279/51279 [==============================] - 16s 307us/sample - loss: 0.8142 - accuracy: 0.75
Epoch 14/30
51279/51279 [==============================] - 16s 312us/sample - loss: 0.7931 - accuracy: 0.76
Epoch 15/30
51279/51279 [==============================] - 16s 306us/sample - loss: 0.7816 - accuracy: 0.76
Epoch 16/30
51279/51279 [==============================] - 16s 312us/sample - loss: 0.7665 - accuracy: 0.76
Epoch 17/30
51279/51279 [==============================] - 16s 320us/sample - loss: 0.7494 - accuracy: 0.77
Epoch 18/30
51279/51279 [==============================] - 16s 318us/sample - loss: 0.7397 - accuracy: 0.77
Epoch 19/30
51279/51279 [==============================] - 14s 277us/sample - loss: 0.7248 - accuracy: 0.78
Epoch 20/30
51279/51279 [==============================] - 14s 265us/sample - loss: 0.7147 - accuracy: 0.78
Epoch 21/30
51279/51279 [==============================] - 14s 269us/sample - loss: 0.7071 - accuracy: 0.78
Epoch 22/30
51279/51279 [==============================] - 14s 267us/sample - loss: 0.6931 - accuracy: 0.79
Epoch 23/30
51279/51279 [==============================] - 14s 263us/sample - loss: 0.6877 - accuracy: 0.79
Epoch 24/30
51279/51279 [==============================] - 14s 269us/sample - loss: 0.6786 - accuracy: 0.79
Epoch 25/30
51279/51279 [==============================] - 14s 267us/sample - loss: 0.6642 - accuracy: 0.79
Epoch 26/30
51279/51279 [==============================] - 14s 271us/sample - loss: 0.6631 - accuracy: 0.80
Epoch 27/30
51279/51279 [==============================] - 14s 265us/sample - loss: 0.6547 - accuracy: 0.80
Epoch 28/30
51279/51279 [==============================] - 14s 271us/sample - loss: 0.6393 - accuracy: 0.80
Epoch 29/30
51279/51279 [==============================] - 14s 270us/sample - loss: 0.6375 - accuracy: 0.81
Epoch 30/30
51279/51279 [==============================] - 13s 255us/sample - loss: 0.6239 - accuracy: 0.81
```

In [18]:
```
plt.figure()
plt.plot(history.epoch, history.history['accuracy'], label='Training')
plt.plot(history.epoch, history.history['val_accuracy'], label='Validation')
plt.legend()
plt.title('Training and validation accuracy')
plt.show()
```



In [19]:
```
plt.figure()
plt.plot(history.epoch, history.history['loss'], label='Training')
plt.plot(history.epoch, history.history['val_loss'], label='Validation')
plt.legend()
plt.title('Training and validation loss')
plt.show()
```




```
In [20]: model.evaluate(x_test, y_test,verbose=0)
```

```
Out[20]: [0.8059455945987092, 0.7698986]
```

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 [ ]: def CNN_model(input_shape):
            model = Sequential()
            model.add(Conv2D(16, (3,3), padding='same', activation='relu', input_shape=input_sh
```

```
    model.add(MaxPooling2D((3,3)))
    model.add(BatchNormalization())
    model.add(Dropout(0.1))

    model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
    model.add(MaxPooling2D((3,3)))
    model.add(BatchNormalization())
    model.add(Dropout(0.1))

    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    return model
```

```
In [22]: model_CNN = CNN_model(x_train.shape[1:])
         model_CNN.compile(optimizer = 'adam',
                           #loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
                            loss = 'sparse_categorical_crossentropy',
                           metrics = ['accuracy'])
         model_CNN.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 16)        160
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 10, 10, 16)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 10, 10, 16)        64
_________________________________________________________________
dropout (Dropout)            (None, 10, 10, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 3, 3, 32)          0
_________________________________________________________________
batch_normalization_1 (Batch (None, 3, 3, 32)          128
_________________________________________________________________
dropout_1 (Dropout)          (None, 3, 3, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 288)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               147968
_________________________________________________________________
dense_4 (Dense)              (None, 128)               65664
_________________________________________________________________
```

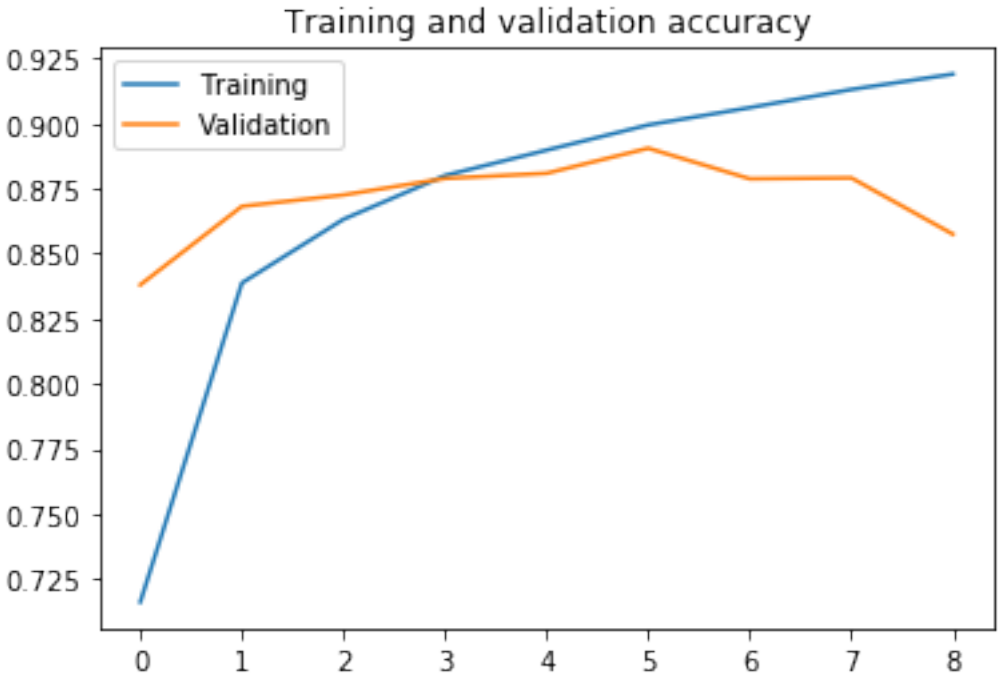
```
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 219,914
Trainable params: 219,818
Non-trainable params: 96
_________________________________________________________________
```

```
In [23]: cnn_checkpoint = ModelCheckpoint('checkpoints/checkpointcnn.h5',
                                  save_best_only=True,
                                  save_weights_only=True)
         earlycnn = earlystopping()

         cnn_history = model_CNN.fit(x_train, y_train,
                                     epochs = 30,
                                     callbacks=[
                                         cnn_checkpoint,
                                         earlycnn
                                     ],
                                     validation_split=0.2,
                                      batch_size=64,
                                     verbose=1)
```

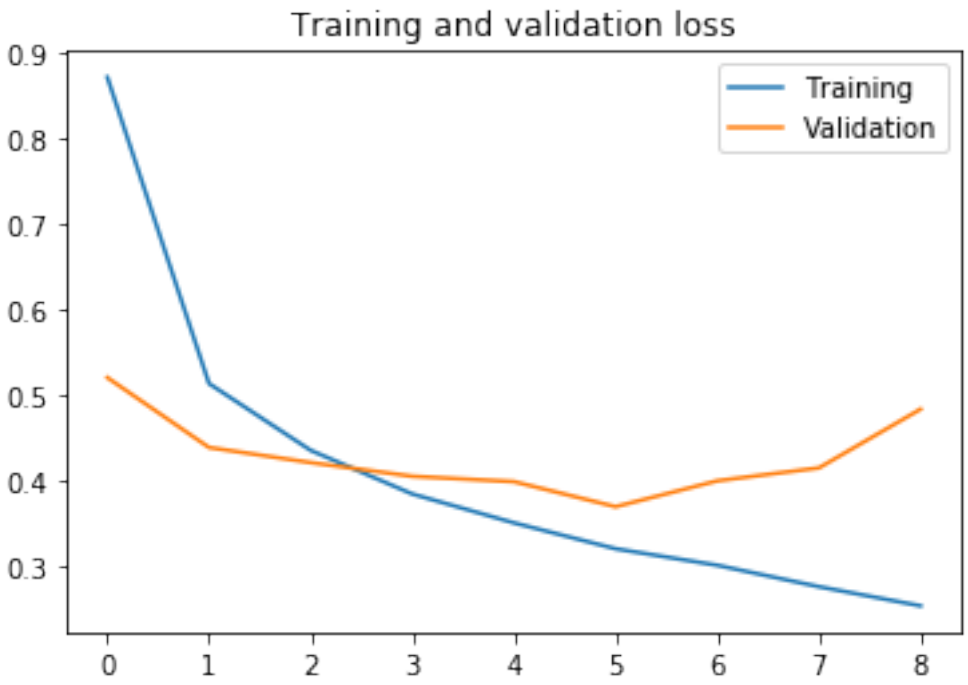
```
Train on 58605 samples, validate on 14652 samples
Epoch 1/30
58605/58605 [==============================] - 172s 3ms/sample - loss: 0.8718 - accuracy: 0.71
Epoch 2/30
58605/58605 [==============================] - 167s 3ms/sample - loss: 0.5141 - accuracy: 0.83
Epoch 3/30
58605/58605 [==============================] - 165s 3ms/sample - loss: 0.4361 - accuracy: 0.86
Epoch 4/30
58605/58605 [==============================] - 176s 3ms/sample - loss: 0.3851 - accuracy: 0.88
Epoch 5/30
58605/58605 [==============================] - 174s 3ms/sample - loss: 0.3514 - accuracy: 0.88
Epoch 6/30
58605/58605 [==============================] - 173s 3ms/sample - loss: 0.3212 - accuracy: 0.89
Epoch 7/30
58605/58605 [==============================] - 169s 3ms/sample - loss: 0.3019 - accuracy: 0.90
Epoch 8/30
58605/58605 [==============================] - 172s 3ms/sample - loss: 0.2769 - accuracy: 0.91
Epoch 9/30
58605/58605 [==============================] - 173s 3ms/sample - loss: 0.2545 - accuracy: 0.91
```

```
In [24]: plt.figure()
         plt.plot(cnn_history.epoch, cnn_history.history['accuracy'], label='Training')
         plt.plot(cnn_history.epoch, cnn_history.history['val_accuracy'], label='Validation')
         plt.legend()
```

```
plt.title('Training and validation accuracy')
plt.show()
```



In [25]:
```
plt.figure()
plt.plot(cnn_history.epoch, cnn_history.history['loss'], label='Training')
plt.plot(cnn_history.epoch, cnn_history.history['val_loss'], label='Validation')
plt.legend()
plt.title('Training and validation loss')
plt.show()
```




```
In [26]: model_CNN.evaluate(x_test, y_test, verbose=0)
```

```
Out[26]: [0.4801437336614913, 0.85863554]
```

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.

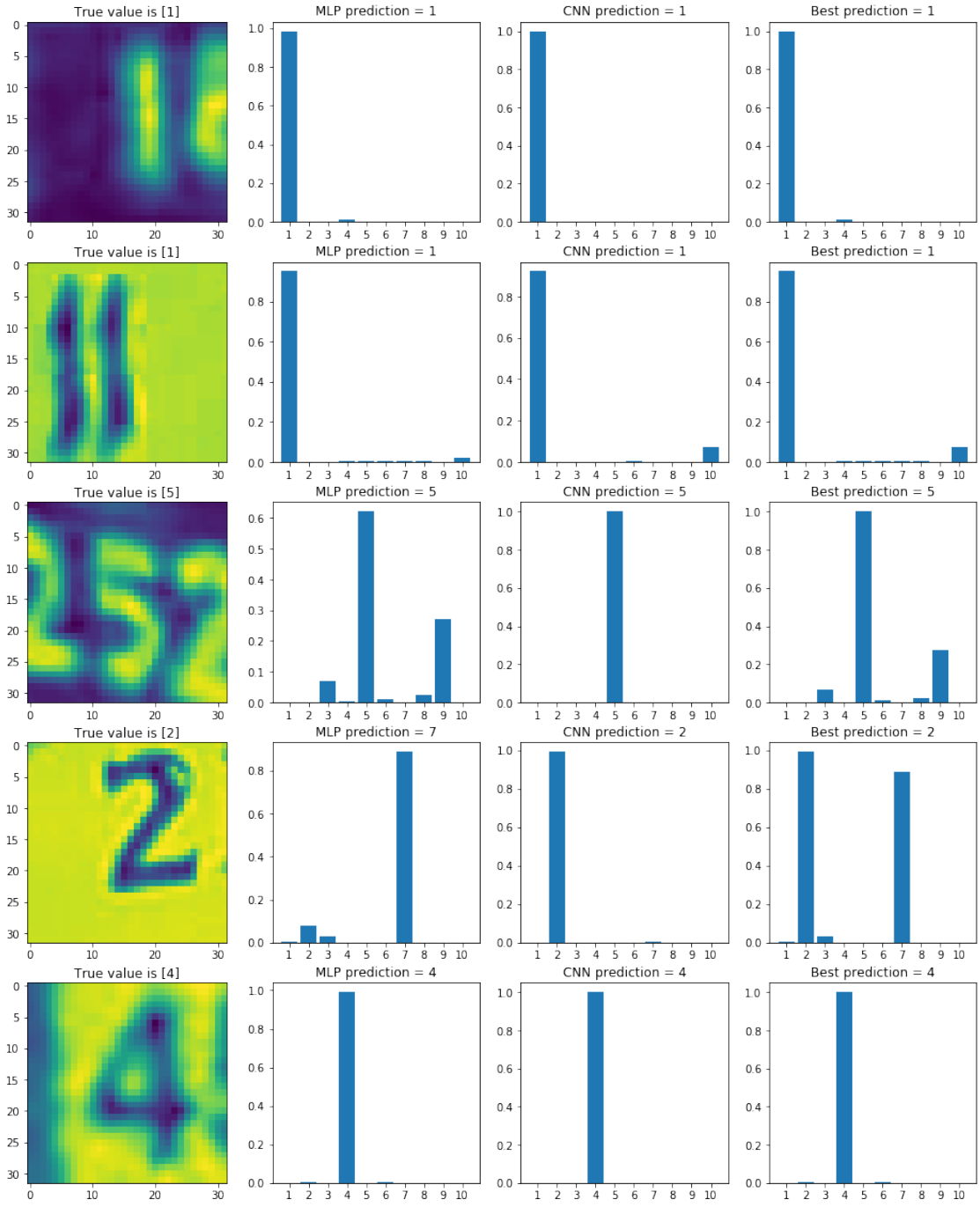
Load MLP & CNN model.

```
In [104]: model_MLP2 = MLP_neuralnetwork((x_train.shape[1:]))
          model_MLP2.compile(optimizer = 'adam',
                             loss = 'sparse_categorical_crossentropy',
                            metrics = ['accuracy'])

          model_MLP2.load_weights('checkpoints/checkpoint.h5')
```

```
In [105]: model_CNN2 = CNN_model(x_train.shape[1:])
          model_CNN2.compile(optimizer = 'adam',
                             loss = 'sparse_categorical_crossentropy',
                            metrics = ['accuracy'])

          model_CNN2.load_weights('checkpoints/checkpointcnn.h5')
```

Random select 5 images.

```
In [109]: idxs = np.random.choice(x_test.shape[0],5)
          y_prob_MLP = model_MLP2.predict_proba(x_test[idxs])
          y_prob_CNN = model_CNN2.predict_proba(x_test[idxs])
          bar_x = np.linspace(1,10,10).astype(int).astype(str)

In [110]: plt.figure(figsize=(16,20))
          for i,v in enumerate(idxs):
              plt.subplot(5,4,4*i+1)
              plt.imshow(x_test[v,:,:,0])
              plt.title('True value is {}'.format(y_test[v]+1))

              plt.subplot(5,4,4*i+2)
              plt.bar(bar_x, y_prob_MLP[i])
              plt.title('MLP prediction = {}'.format(np.argmax(y_prob_MLP[i])+1))

              plt.subplot(5,4,4*i+3)
              plt.bar(bar_x, y_prob_CNN[i])
              plt.title('CNN prediction = {}'.format(np.argmax(y_prob_CNN[i])+1))


              plt.subplot(5,4,4*i+4)
              y_max = np.vstack((y_prob_MLP[i], y_prob_CNN[i])).max(axis=0)
              plt.bar(bar_x, y_max)
              plt.title('Best prediction = {}'.format(np.argmax(y_max)+1))
          plt.show()
```


True value is [1]
MLP prediction = 1
CNN prediction = 1
Best prediction = 1
True value is [1]
MLP prediction = 1
CNN prediction = 1
Best prediction = 1
True value is [5]
MLP prediction = 5
CNN prediction = 5
Best prediction = 5
True value is [2]
MLP prediction = 7
CNN prediction = 2
Best prediction = 2
True value is [4]
MLP prediction = 4
CNN prediction = 4
Best prediction = 4

In [ ]: