Capstone Project

April 17, 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 [1]: import tensorflow as tf
        from scipy.io import loadmat

        import numpy as np
        import matplotlib.pyplot as plt
        import math

        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, BatchNormaliz
        from tensorflow.keras import regularizers
        from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStoppi
```

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

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

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

```
In [2]: # Run this cell to load the dataset

        train = loadmat('data/train_32x32.mat')
        test = loadmat('data/test_32x32.mat')
```

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

1.2 1. Inspect and preprocess the dataset

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

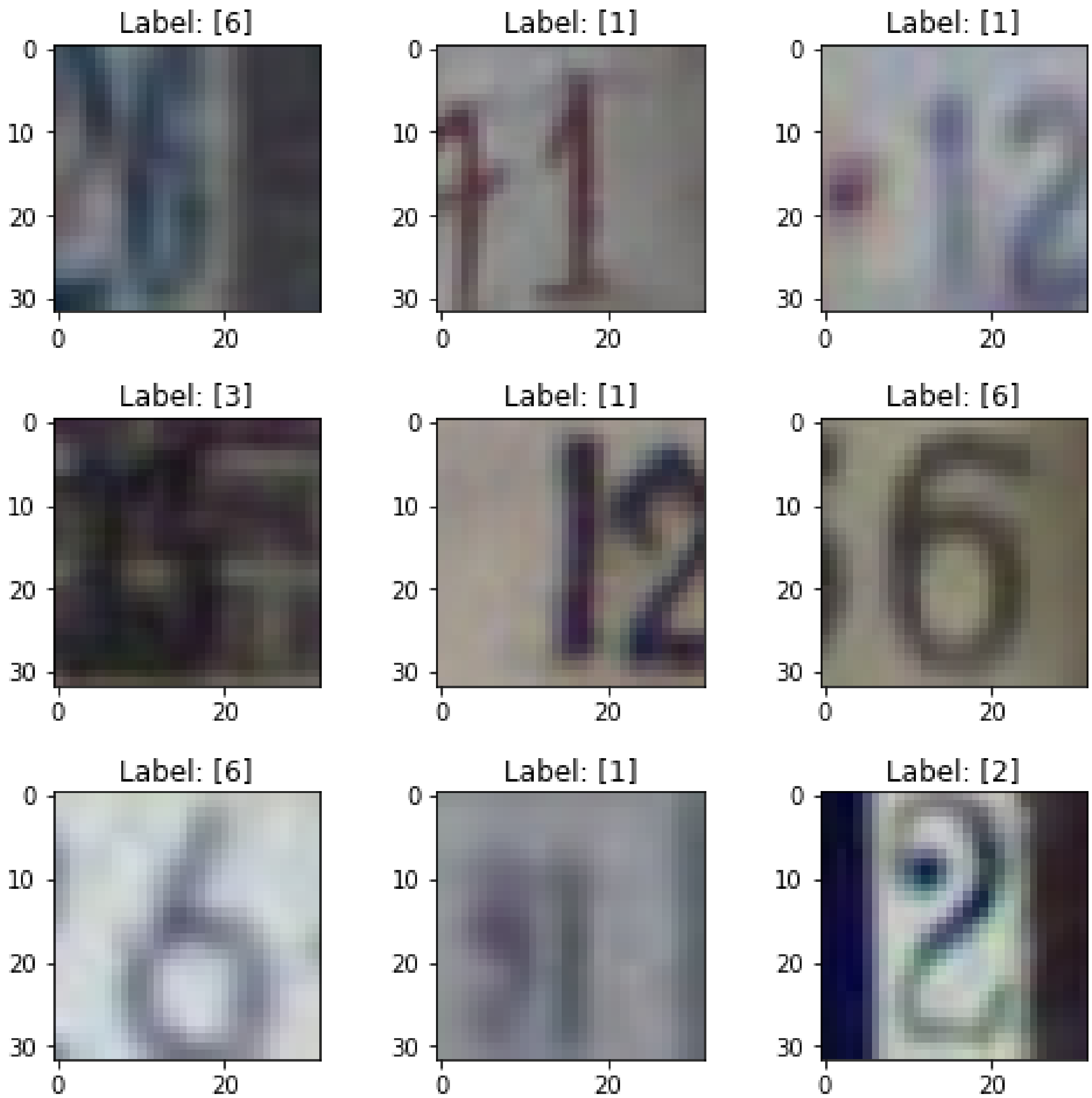
```
In [3]: """
        print(train.keys()) #dict_keys(['__header__', '__version__', '__globals__', 'X', 'y'])
        """
```

```
train_features = train["X"]
train_labels   = train["y"]

test_features  = test["X"]
test_labels     = test["y"]

"""
print(train_features.shape) #(32, 32, 3, 73257)
print(train_labels.shape) #(73257, 1)
"""

train_features = np.moveaxis(train_features,3,0) #move last axis on first place for it
test_features = np.moveaxis(test_features,3,0) #move last axis on first place for iter
"""
print(train_features.shape) #(73257, 32, 32, 3)
print(train_labels.shape) #(73257, 1)
"""
```

Out[3]: '\nprint(train_features.shape) #(73257, 32, 32, 3)\nprint(train_labels.shape) #(73257,

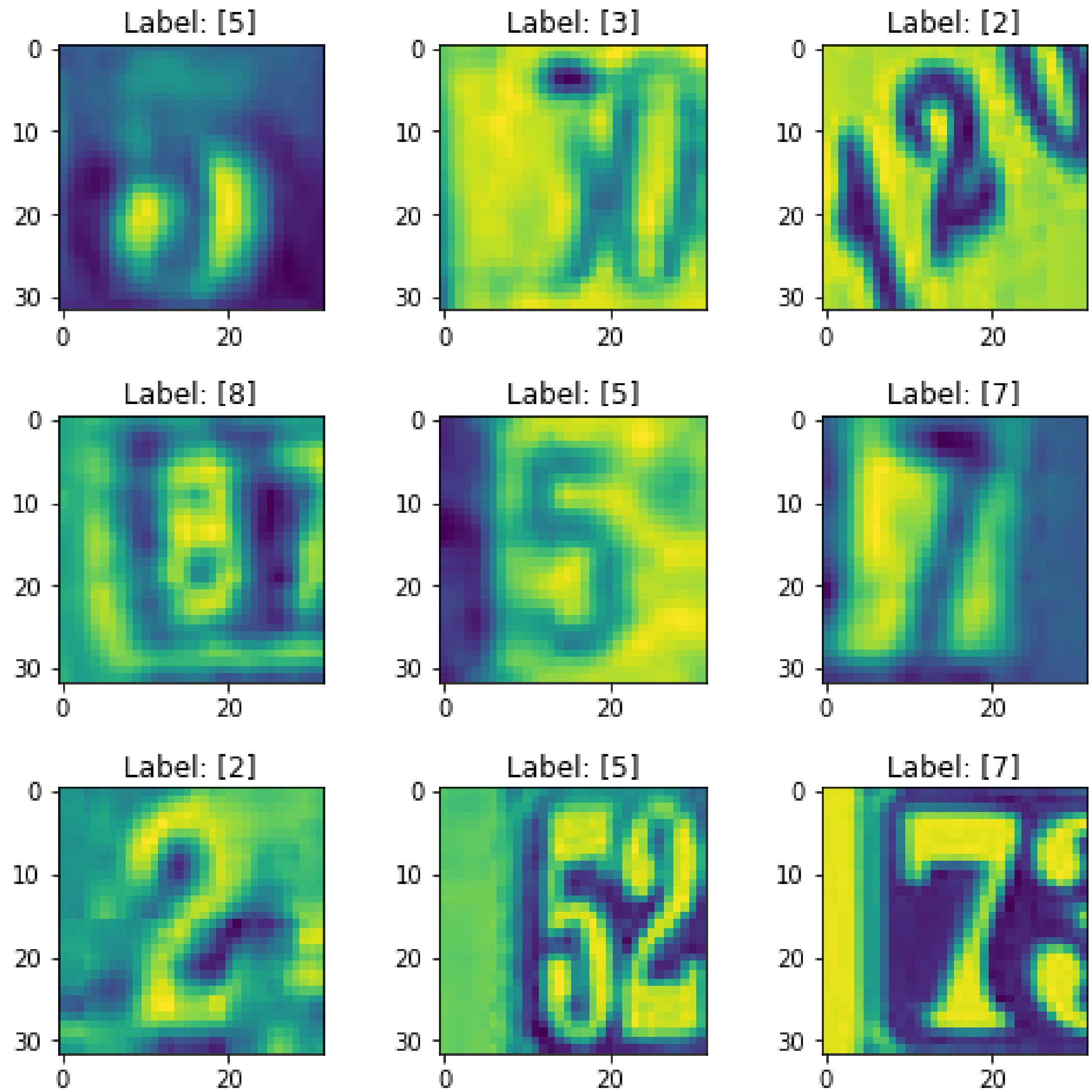
In [4]:
```
def plot2square(features, labels, N):
    #pick number of samples to show
    n_samples = 9
    random_inx = np.random.choice(features.shape[0], n_samples)

    #random samples with their labels
    random_test_images = features[random_inx] #move last axis on first place for itera
    random_test_labels = labels[random_inx]

    #make square like plot field
    x_len = int(math.sqrt(n_samples))
    y_len = math.ceil(n_samples/x_len)

    #prepare fig
    fig, axes = plt.subplots(x_len, y_len, figsize=(8, 8))
    fig.subplots_adjust(hspace=0.4, wspace= 0.4)
    for i, (image, label) in enumerate(zip(random_test_images, random_test_labels)):
        axes[int(i / y_len), i % y_len].imshow(np.squeeze(image))
        axes[int(i / y_len), i % y_len].set_title(f"Label: {label}")
```

In [5]:
```
plot2square(train_features, train_labels, 9)
```



In [6]:

```
#matlab's (NTSC/PAL) implementation:
def rgb2gray(features):
    RGB_weights = (0.2989, 0.5870, 0.1140)
    return np.average(features, axis=3, weights=RGB_weights).reshape((features.shape[0

train_features_gray = rgb2gray(train_features)

#image test
plot2square(train_features_gray, train_labels, 9)
```



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 [7]: def get_model(shape, wd1, wd2):
            return Sequential([
                Flatten(input_shape=shape),
                Dense(512, activation='relu', kernel_regularizer=regularizers.l1_l2(wd1, wd2),
                Dense(256, activation='relu', kernel_regularizer=regularizers.l1_l2(wd1, wd2),
                Dense(128, activation='relu', kernel_regularizer=regularizers.l1_l2(wd1, wd2),
                Dense(64, activation='relu', kernel_regularizer=regularizers.l1_l2(wd1, wd2), B
                Dense(10, activation='softmax')
            ])

        mlp_model = get_model(train_features_gray[0].shape, 1e-5, 1e-3)
        mlp_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 512)               524800
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_4 (Dense)              (None, 10)                650
=================================================================
Total params: 697,930
Trainable params: 697,930
Non-trainable params: 0
_________________________________________________________________
```

```
In [8]: opt = tf.keras.optimizers.Adam(learning_rate=0.0025)
        mlp_model.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accu

In [9]: reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                                      patience=2, min_lr=0.0001, verbose=2)
        early_stopping = EarlyStopping(monitor='accuracy', patience=3)
        save_best_mlp = ModelCheckpoint(filepath='./checkpoints/mlp/model_best_epoch', monitor=

        history = mlp_model.fit(train_features_gray, train_labels-1, epochs=30,
                                validation_split=0.2, batch_size=128, verbose=2,
```

```
                           callbacks=[reduce_lr, early_stopping, save_best_mlp]
                          )

Train on 58605 samples, validate on 14652 samples
Epoch 1/30

Epoch 00001: loss improved from inf to 33.53691, saving model to ./checkpoints/mlp/model_best_e
58605/58605 - 41s - loss: 33.5369 - accuracy: 0.1514 - val_loss: 5.8613 - val_accuracy: 0.1292
Epoch 2/30

Epoch 00002: loss improved from 33.53691 to 3.76455, saving model to ./checkpoints/mlp/model_be
58605/58605 - 37s - loss: 3.7645 - accuracy: 0.2858 - val_loss: 3.5614 - val_accuracy: 0.3615
Epoch 3/30

Epoch 00003: loss improved from 3.76455 to 3.13827, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 37s - loss: 3.1383 - accuracy: 0.4746 - val_loss: 3.2197 - val_accuracy: 0.4808
Epoch 4/30

Epoch 00004: loss improved from 3.13827 to 2.87923, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 37s - loss: 2.8792 - accuracy: 0.5454 - val_loss: 2.6728 - val_accuracy: 0.6068
Epoch 5/30

Epoch 00005: loss improved from 2.87923 to 2.81623, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 37s - loss: 2.8162 - accuracy: 0.5428 - val_loss: 2.6388 - val_accuracy: 0.5775
Epoch 6/30

Epoch 00006: loss improved from 2.81623 to 2.64597, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 37s - loss: 2.6460 - accuracy: 0.5638 - val_loss: 2.4996 - val_accuracy: 0.5937
Epoch 7/30

Epoch 00007: loss improved from 2.64597 to 2.41388, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 37s - loss: 2.4139 - accuracy: 0.6013 - val_loss: 2.3858 - val_accuracy: 0.5868
Epoch 8/30

Epoch 00008: loss improved from 2.41388 to 2.25281, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 37s - loss: 2.2528 - accuracy: 0.6241 - val_loss: 2.1773 - val_accuracy: 0.6356
Epoch 9/30

Epoch 00009: loss improved from 2.25281 to 2.14230, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 37s - loss: 2.1423 - accuracy: 0.6287 - val_loss: 2.0962 - val_accuracy: 0.6225
Epoch 10/30

Epoch 00010: loss improved from 2.14230 to 2.02571, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 38s - loss: 2.0257 - accuracy: 0.6407 - val_loss: 1.9734 - val_accuracy: 0.6372
Epoch 11/30

Epoch 00011: loss improved from 2.02571 to 1.92652, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 37s - loss: 1.9265 - accuracy: 0.6420 - val_loss: 1.8744 - val_accuracy: 0.6353
```

```
Epoch 12/30

Epoch 00012: loss improved from 1.92652 to 1.76827, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 37s - loss: 1.7683 - accuracy: 0.6618 - val_loss: 1.7071 - val_accuracy: 0.6695
Epoch 13/30

Epoch 00013: loss improved from 1.76827 to 1.69317, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 36s - loss: 1.6932 - accuracy: 0.6573 - val_loss: 1.6439 - val_accuracy: 0.6637
Epoch 14/30

Epoch 00014: loss improved from 1.69317 to 1.59199, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 36s - loss: 1.5920 - accuracy: 0.6642 - val_loss: 1.5367 - val_accuracy: 0.6740
Epoch 15/30

Epoch 00015: loss improved from 1.59199 to 1.50660, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 36s - loss: 1.5066 - accuracy: 0.6702 - val_loss: 1.5764 - val_accuracy: 0.6329
Epoch 16/30

Epoch 00016: loss improved from 1.50660 to 1.45933, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 36s - loss: 1.4593 - accuracy: 0.6691 - val_loss: 1.4280 - val_accuracy: 0.6752
Epoch 17/30

Epoch 00017: loss improved from 1.45933 to 1.43822, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 36s - loss: 1.4382 - accuracy: 0.6652 - val_loss: 1.5226 - val_accuracy: 0.6374
Epoch 18/30

Epoch 00018: loss improved from 1.43822 to 1.38218, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 37s - loss: 1.3822 - accuracy: 0.6733 - val_loss: 1.3928 - val_accuracy: 0.6683
Epoch 19/30

Epoch 00019: loss improved from 1.38218 to 1.34554, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 37s - loss: 1.3455 - accuracy: 0.6795 - val_loss: 1.3878 - val_accuracy: 0.6600
Epoch 20/30

Epoch 00020: loss improved from 1.34554 to 1.30072, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 37s - loss: 1.3007 - accuracy: 0.6819 - val_loss: 1.3114 - val_accuracy: 0.6833
Epoch 21/30

Epoch 00021: loss improved from 1.30072 to 1.28560, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 37s - loss: 1.2856 - accuracy: 0.6863 - val_loss: 1.3636 - val_accuracy: 0.6567
Epoch 22/30

Epoch 00022: loss improved from 1.28560 to 1.25140, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 36s - loss: 1.2514 - accuracy: 0.6882 - val_loss: 1.2847 - val_accuracy: 0.6860
Epoch 23/30

Epoch 00023: loss did not improve from 1.25140
58605/58605 - 36s - loss: 1.2701 - accuracy: 0.6863 - val_loss: 1.2037 - val_accuracy: 0.7069
```

```
Epoch 24/30

Epoch 00024: loss improved from 1.25140 to 1.24867, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 36s - loss: 1.2487 - accuracy: 0.6919 - val_loss: 1.2987 - val_accuracy: 0.6775
Epoch 25/30

Epoch 00025: ReduceLROnPlateau reducing learning rate to 0.0004999999888241291.

Epoch 00025: loss did not improve from 1.24867
58605/58605 - 36s - loss: 1.2492 - accuracy: 0.6907 - val_loss: 1.2344 - val_accuracy: 0.6982
Epoch 26/30

Epoch 00026: loss improved from 1.24867 to 1.06543, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 36s - loss: 1.0654 - accuracy: 0.7393 - val_loss: 1.0745 - val_accuracy: 0.7314
Epoch 27/30

Epoch 00027: loss improved from 1.06543 to 0.99996, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 36s - loss: 1.0000 - accuracy: 0.7457 - val_loss: 1.0362 - val_accuracy: 0.7273
Epoch 28/30

Epoch 00028: loss improved from 0.99996 to 0.96631, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 36s - loss: 0.9663 - accuracy: 0.7480 - val_loss: 1.0228 - val_accuracy: 0.7316
Epoch 29/30

Epoch 00029: loss improved from 0.96631 to 0.94263, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 36s - loss: 0.9426 - accuracy: 0.7524 - val_loss: 1.0036 - val_accuracy: 0.7358
Epoch 30/30

Epoch 00030: loss improved from 0.94263 to 0.93334, saving model to ./checkpoints/mlp/model_bes
58605/58605 - 36s - loss: 0.9333 - accuracy: 0.7483 - val_loss: 1.0017 - val_accuracy: 0.7275
```

In [10]:
```
mlp_model.evaluate(rgb2gray(test_features), test_labels-1, verbose=2)
```

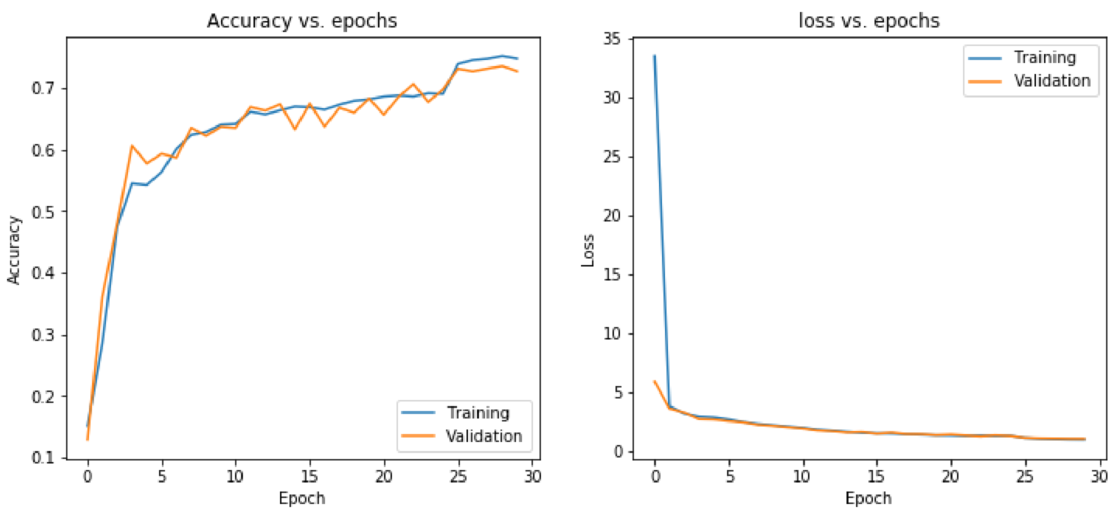
```
26032/1 - 13s - loss: 1.2140 - accuracy: 0.7071
```

Out[10]: [1.0998752706242838, 0.7070529]

In [11]:
```
def plot_history(history):
    fig = plt.figure(figsize=(12, 5))

    fig.add_subplot(121)

    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Accuracy vs. epochs')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
```

```
    plt.legend(['Training', 'Validation'], loc='lower right')

    fig.add_subplot(122)

    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 [12]: plot_history(history)
```



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 [13]: def get_model(shape, dropout_rate):
             return Sequential([
                 Conv2D(32, (4,4), activation="relu", input_shape=(32,32,1), padding="same"),
                 MaxPooling2D((4,4)),
                 BatchNormalization(),
                 Flatten(),
                 Dense(256, activation="relu", kernel_initializer='he_uniform'),
                 Dropout(dropout_rate),
                 Dense(128, activation="relu", kernel_initializer='he_uniform'),
                 Dense(10, activation="softmax")
             ])
```

```
In [14]: cnn_model = get_model(train_features_gray[0].shape, 0.3)
         cnn_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        544
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 8, 8, 32)          0
_________________________________________________________________
batch_normalization (BatchNo (None, 8, 8, 32)          128
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               524544
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290
=================================================================
Total params: 559,402
Trainable params: 559,338
Non-trainable params: 64
_________________________________________________________________
```

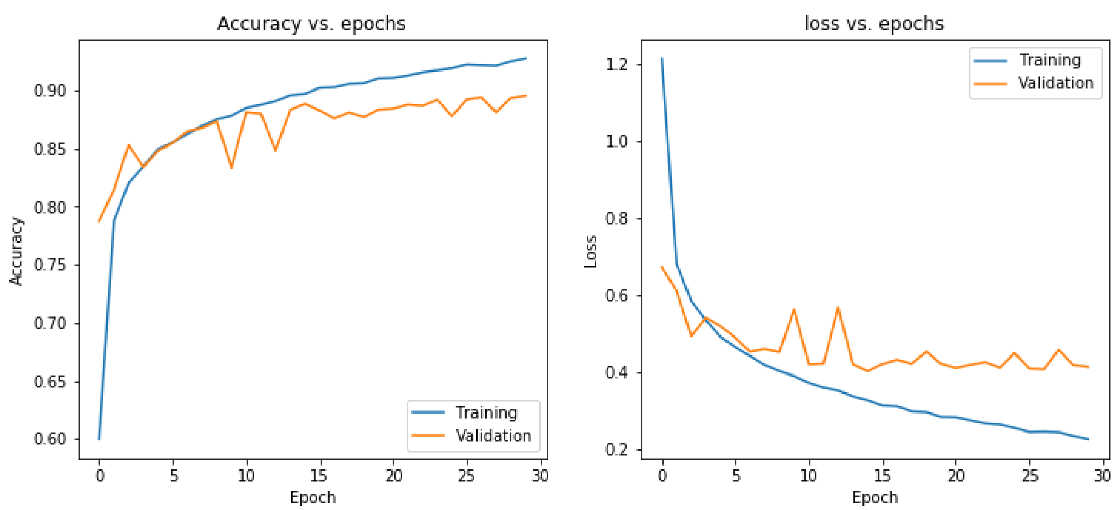
```
In [15]: save_best_cnn = ModelCheckpoint(filepath='./checkpoints/cnn/model_best_epoch', monito

         cnn_model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=['
```

```
In [16]: history = cnn_model.fit(train_features_gray, train_labels-1, epochs=30,
                                 validation_split=0.2, batch_size=128, verbose=2,
                                 callbacks=[early_stopping, save_best_cnn]
                                 )
```

```
Train on 58605 samples, validate on 14652 samples
Epoch 1/30
58605/58605 - 154s - loss: 1.2151 - accuracy: 0.5998 - val_loss: 0.6731 - val_accuracy: 0.7875
Epoch 2/30
58605/58605 - 149s - loss: 0.6820 - accuracy: 0.7875 - val_loss: 0.6115 - val_accuracy: 0.8143
Epoch 3/30
58605/58605 - 149s - loss: 0.5849 - accuracy: 0.8200 - val_loss: 0.4933 - val_accuracy: 0.8529
Epoch 4/30
58605/58605 - 148s - loss: 0.5347 - accuracy: 0.8344 - val_loss: 0.5413 - val_accuracy: 0.8339
Epoch 5/30
58605/58605 - 146s - loss: 0.4916 - accuracy: 0.8494 - val_loss: 0.5197 - val_accuracy: 0.8479
Epoch 6/30
58605/58605 - 146s - loss: 0.4648 - accuracy: 0.8550 - val_loss: 0.4894 - val_accuracy: 0.8546
Epoch 7/30
58605/58605 - 147s - loss: 0.4419 - accuracy: 0.8623 - val_loss: 0.4531 - val_accuracy: 0.8647
Epoch 8/30
58605/58605 - 147s - loss: 0.4185 - accuracy: 0.8693 - val_loss: 0.4600 - val_accuracy: 0.8675
Epoch 9/30
58605/58605 - 148s - loss: 0.4030 - accuracy: 0.8750 - val_loss: 0.4520 - val_accuracy: 0.8733
Epoch 10/30
58605/58605 - 148s - loss: 0.3897 - accuracy: 0.8781 - val_loss: 0.5633 - val_accuracy: 0.8333
Epoch 11/30
58605/58605 - 149s - loss: 0.3719 - accuracy: 0.8848 - val_loss: 0.4203 - val_accuracy: 0.8812
Epoch 12/30
58605/58605 - 152s - loss: 0.3598 - accuracy: 0.8874 - val_loss: 0.4222 - val_accuracy: 0.8799
Epoch 13/30
58605/58605 - 150s - loss: 0.3522 - accuracy: 0.8906 - val_loss: 0.5682 - val_accuracy: 0.8481
Epoch 14/30
58605/58605 - 149s - loss: 0.3372 - accuracy: 0.8956 - val_loss: 0.4205 - val_accuracy: 0.8831
Epoch 15/30
58605/58605 - 148s - loss: 0.3270 - accuracy: 0.8969 - val_loss: 0.4026 - val_accuracy: 0.8883
Epoch 16/30
58605/58605 - 145s - loss: 0.3134 - accuracy: 0.9021 - val_loss: 0.4206 - val_accuracy: 0.8825
Epoch 17/30
58605/58605 - 147s - loss: 0.3114 - accuracy: 0.9026 - val_loss: 0.4322 - val_accuracy: 0.8758
Epoch 18/30
58605/58605 - 147s - loss: 0.2978 - accuracy: 0.9055 - val_loss: 0.4213 - val_accuracy: 0.8810
Epoch 19/30
58605/58605 - 146s - loss: 0.2960 - accuracy: 0.9060 - val_loss: 0.4536 - val_accuracy: 0.8769
Epoch 20/30
58605/58605 - 153s - loss: 0.2838 - accuracy: 0.9102 - val_loss: 0.4214 - val_accuracy: 0.8833
Epoch 21/30
58605/58605 - 157s - loss: 0.2830 - accuracy: 0.9106 - val_loss: 0.4107 - val_accuracy: 0.8841
```

```
Epoch 22/30
58605/58605 - 148s - loss: 0.2748 - accuracy: 0.9127 - val_loss: 0.4188 - val_accuracy: 0.8877
Epoch 23/30
58605/58605 - 146s - loss: 0.2668 - accuracy: 0.9154 - val_loss: 0.4258 - val_accuracy: 0.8866
Epoch 24/30
58605/58605 - 149s - loss: 0.2638 - accuracy: 0.9172 - val_loss: 0.4116 - val_accuracy: 0.8917
Epoch 25/30
58605/58605 - 149s - loss: 0.2551 - accuracy: 0.9189 - val_loss: 0.4495 - val_accuracy: 0.8778
Epoch 26/30
58605/58605 - 147s - loss: 0.2445 - accuracy: 0.9221 - val_loss: 0.4093 - val_accuracy: 0.8920
Epoch 27/30
58605/58605 - 148s - loss: 0.2454 - accuracy: 0.9215 - val_loss: 0.4073 - val_accuracy: 0.8939
Epoch 28/30
58605/58605 - 147s - loss: 0.2439 - accuracy: 0.9211 - val_loss: 0.4579 - val_accuracy: 0.8811
Epoch 29/30
58605/58605 - 150s - loss: 0.2341 - accuracy: 0.9248 - val_loss: 0.4183 - val_accuracy: 0.8933
Epoch 30/30
58605/58605 - 148s - loss: 0.2259 - accuracy: 0.9273 - val_loss: 0.4136 - val_accuracy: 0.8952
```

```
In [17]: plot_history(history)
```




```
In [ ]:
```

1.5 4. Get model predictions

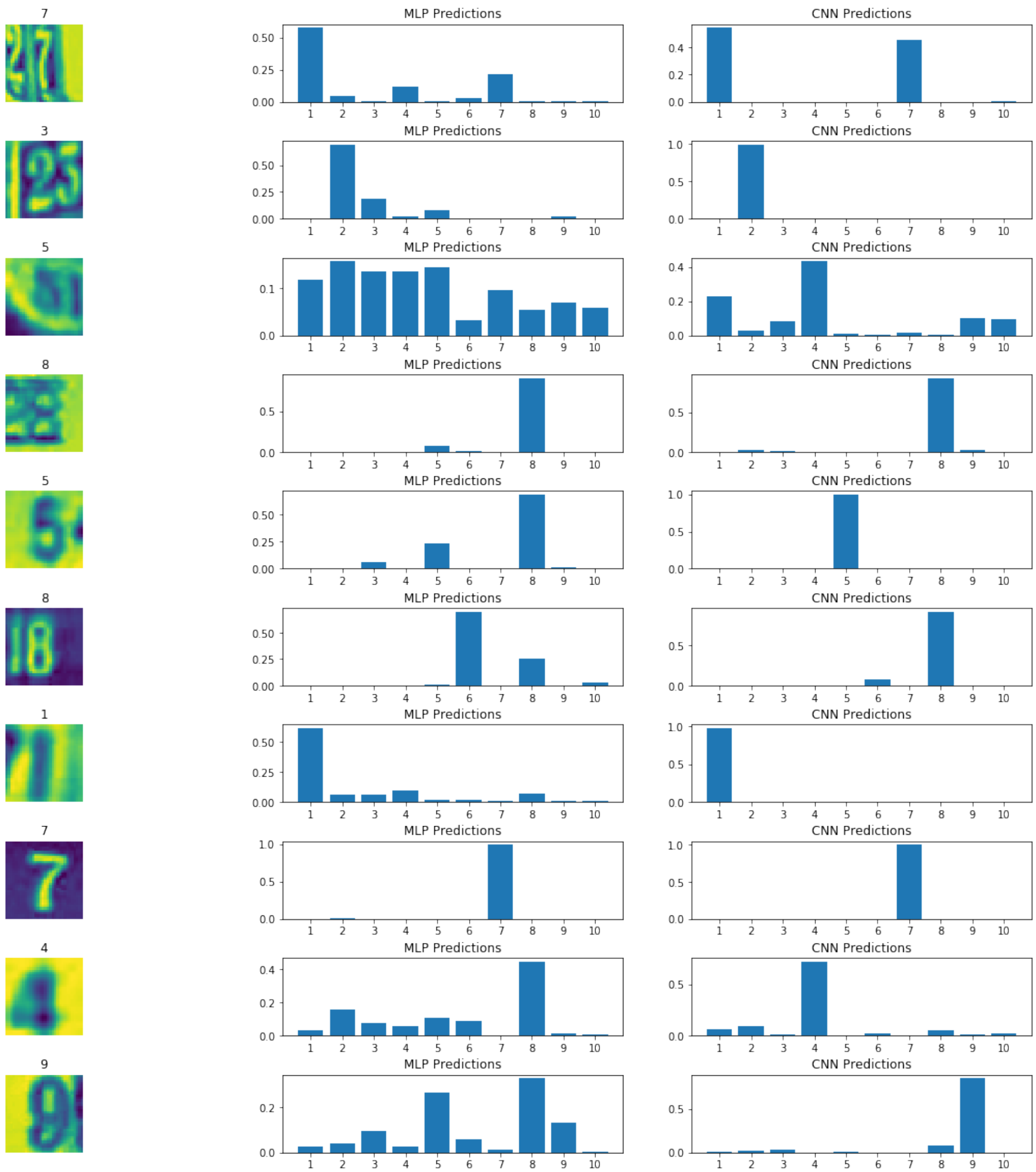
- Load the best weights for the MLP and CNN models that you saved during the training run.
- Randomly select 5 images and corresponding labels from the test set and display the images with their labels.
- Alongside the image and label, show each model's predictive distribution as a bar chart, and the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

```
In [18]: mlp_model.load_weights(tf.train.latest_checkpoint('./checkpoints/mlp/'))
         cnn_model.load_weights(tf.train.latest_checkpoint('./checkpoints/cnn/'))
```

```
Out[18]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f4ed8375b70>
```

```
In [19]: images = []
         labels = []
         predictions_mlp = []
         predictions_cnn = []
         n_tests = 10

         for i in range(0, n_tests):
             rdm_num = np.random.randint(rgb2gray(test_features).shape[0], size=1)
             img = rgb2gray(test_features)[rdm_num, :,:,:]
             images.append(img)
             labels.append(str(test_labels[rdm_num]).strip('[]'))
             predictions_mlp.append(mlp_model.predict(img))
             predictions_cnn.append(cnn_model.predict(img))
```

```
In [20]: fig, axs = plt.subplots(nrows=n_tests, ncols=3, figsize=(20,20))
         fig.subplots_adjust(hspace=0.5, wspace=0.2)

         x = np.arange(1,11)

         for i, (image, label, mlp_prediction, cnn_prediction) in enumerate(zip(images, labels
             axs[i, 0].imshow(np.squeeze(image))
             axs[i, 0].set_title(label)
             axs[i, 0].axis('off')
             axs[i, 1].bar(x, mlp_prediction.reshape(10,))
             axs[i, 1].set_title('MLP Predictions')
             axs[i, 1].set_xticks(x)
             axs[i, 2].bar(x, cnn_prediction.reshape(10,))
             axs[i, 2].set_title('CNN Predictions')
             axs[i, 2].set_xticks(x)
```


7
MLP Predictions
CNN Predictions
3
MLP Predictions
CNN Predictions
5
MLP Predictions
CNN Predictions
8
MLP Predictions
CNN Predictions
5
MLP Predictions
CNN Predictions
8
MLP Predictions
CNN Predictions
1
MLP Predictions
CNN Predictions
7
MLP Predictions
CNN Predictions
4
MLP Predictions
CNN Predictions
9
MLP Predictions
CNN Predictions

In [ ]:

In [ ]:

In [ ]: