Capstone Project

September 25, 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 [1]: import tensorflow as tf
        from scipy.io import loadmat
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
        import seaborn as sns
        sns.set_style('white')
        from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
        from tensorflow.keras.models import Sequential, load_model
        from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout, Bat
```

 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]: # Load the training and testing data and labels from the dictionary
        train_x, train_y, test_x, test_y = train['X'], train['y'], test['X'], test['y']
        print(f'Training data shape: {train_x.shape}\n\
```

```
        Training labels shape:{train_y.shape}\n\
        Test data shape:{test_x.shape}\n\
        Test labels shape:{test_y.shape}')
```

```
Training data shape: (32, 32, 3, 73257)
Training labels shape:(73257, 1)
Test data shape:(32, 32, 3, 26032)
Test labels shape:(26032, 1)
```

In [4]:
```python
# Rearrange the train and test data to have number of examples 1st

train_x = np.moveaxis(train_x, -1, 0)
test_x = np.moveaxis(test_x, -1, 0)

# Print the shape of the data again
print(f'Training data shape: {train_x.shape}\n\
Training labels shape:{train_y.shape}\n\
Test data shape:{test_x.shape}\n\
Test labels shape:{test_y.shape}')
```

```
Training data shape: (73257, 32, 32, 3)
Training labels shape:(73257, 1)
Test data shape:(26032, 32, 32, 3)
Test labels shape:(26032, 1)
```

In [5]:
```python
# Replace the class label 10 to 0
test_y[test_y==10] = 0
train_y[train_y==10] = 0
```

In [6]:
```python
# Helper function to convert color images to gray and normalize it.
def color_to_normalized_gray(arr):
    """Convert array of colored images to gray and normalize it's values
       by dividing by 255.
       baram: arr: numpy array of shape(num_images, height, width, 3)
       return normailized array of gray images.
    """
    gray_images = np.mean(arr, 3, keepdims=True) / 255
    return gray_images
```

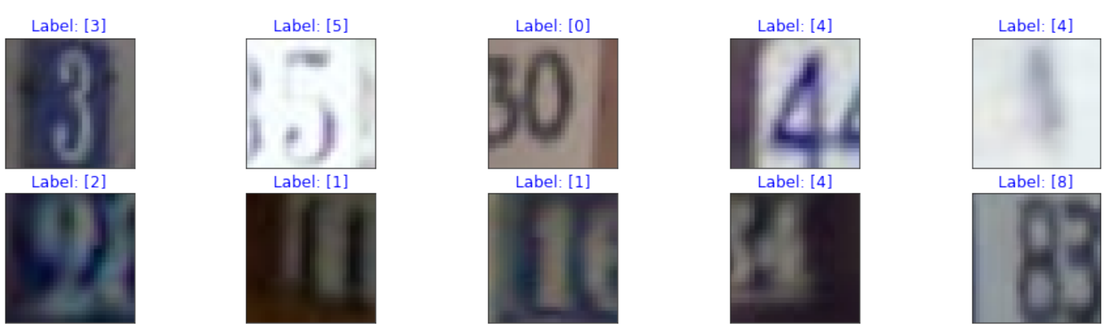
In [7]:
```python
train_x_gray = color_to_normalized_gray(train_x)
test_x_gray = color_to_normalized_gray(test_x)
# Print the shape of the data again
print(f'New training data shape: {train_x_gray.shape}\n\
New est data shape:{test_x_gray.shape}\n')
```

```
New training data shape: (73257, 32, 32, 1)
New est data shape:(26032, 32, 32, 1)
```

```
In [8]: def plot_images(images, nrows, ncols, true_labels, predictions=None):
            """ Helper function for plotting nrows * ncols images
                param: images: array of images to plot
                param: nrows: number of rows in the subplot
                param: ncols: number of columns in the subplot
                param: true_labels: true label values of the images
                param: predictions: predicted labels of the images
                return None
            """
            fig, axes = plt.subplots(nrows, ncols, figsize=(16, 2*nrows))
            for i, ax in enumerate(axes.flat):
                if images[i].shape == (32, 32, 3):
                    ax.imshow(images[i])
                else:
                    ax.imshow(images[i,:,:,0], cmap="gray")

                if predictions is None:
                    title = f"Label: {str(true_labels[i])}"
                    ax.set_title(title, color='b')
                elif predictions[i] == true_labels[i]:
                    title = f"Label: {str(true_labels[i])}, Pred: [{str(predictions[i])}]"
                    ax.set_title(title, color='g')
                else:
                    title = f"Label: {str(true_labels[i])}, Pred: [{str(predictions[i])}]"
                    ax.set_title(title, color='r')

                ax.set_xticks([]); ax.set_yticks([])
```

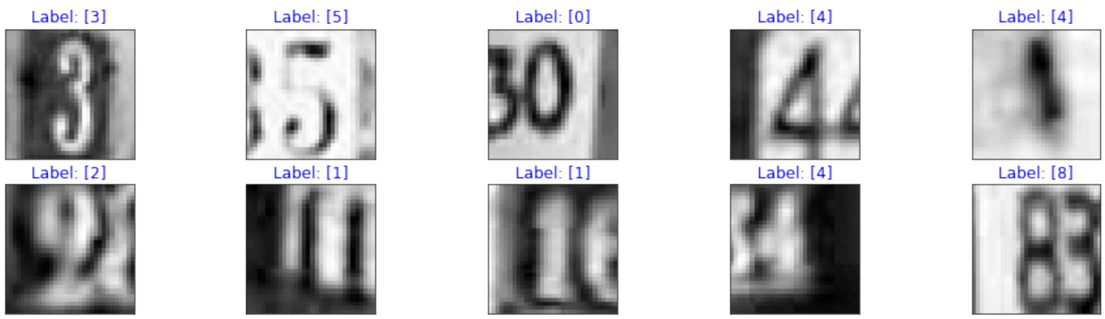
```
In [9]: # Get random indices for color images
        np.random.seed(0)
        idx_train = np.random.choice(train_x.shape[0]-1, 10)
        print('Color Images Examples')
        plot_images(train_x[idx_train], 2, 5, train_y[idx_train])
```

```
Color Images Examples
```



```
In [10]: # get random indices for Gray preprocessed images
         np.random.seed(0)
         idx_test = np.random.choice(train_x_gray.shape[0]-1, 10)
         print('Gray Images Examples')
         plot_images(train_x_gray[idx_test], 2, 5, train_y[idx_test])
```

```
Gray Images Examples
```



1.3 2. MLP neural network classifier

- Build an MLP classifier model using the Sequential API. Your model should use only Flatten and Dense layers, with the final layer having a 10-way softmax output.
- You should design and build the model yourself. Feel free to experiment with different MLP architectures. *Hint: to achieve a reasonable accuracy you won't need to use more than 4 or 5 layers.*
- Print out the model summary (using the summary() method)
- Compile and train the model (we recommend a maximum of 30 epochs), making use of both training and validation sets during the training run.
- Your model should track at least one appropriate metric, and use at least two callbacks during training, one of which should be a ModelCheckpoint callback.
- As a guide, you should aim to achieve a final categorical cross entropy training loss of less than 1.0 (the validation loss might be higher).
- Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and validation sets.
- Compute and display the loss and accuracy of the trained model on the test set.

```
In [11]: # Create MLP model
         def get_mlp_model():
             mlp_model = Sequential([
                                         Flatten(input_shape=train_x_gray[0].shape),
                                         Dense(512, activation='relu'),
                                         Dense(256, activation='relu'),
                                         Dense(128, activation='relu'),
                                         Dense(10, activation='softmax')
             ])
             return mlp_model
```

```
In [12]: mlp_model = get_mlp_model()
```

```
In [13]: # Print model summary
         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, 10)                1290
=================================================================
Total params: 690,314
Trainable params: 690,314
Non-trainable params: 0
_________________________________________________________________
```

```
In [14]: # Compile the model
         mlp_model.compile(optimizer='adam',
                           loss='sparse_categorical_crossentropy',
                           metrics=['accuracy'])
```

```
In [15]: # Create call backs
         checkpoint = ModelCheckpoint(filepath='mlp_model.h5',
                                      save_best_only=True,
                                      save_weights_only=True,
                                      monitor='val_loss',
                                      mode='min',
                                      verbose=1)

         early_stop = EarlyStopping(patience=5,
                                    monitor='loss',
                                    mode='min')
```

```
In [16]: # Train the model
         history = mlp_model.fit(train_x_gray,
                                 train_y,
                                 epochs=30,
                                 validation_data=(test_x_gray, test_y),
                                 batch_size=128,
                                 callbacks=[checkpoint, early_stop])
```

```
Train on 73257 samples, validate on 26032 samples
Epoch 1/30
73088/73257 [============================>.] - ETA: 0s - loss: 1.9441 - accuracy: 0.3117
Epoch 00001: val_loss improved from inf to 1.54935, saving model to mlp_model.h5
73257/73257 [==============================] - 45s 608us/sample - loss: 1.9426 - accuracy: 0.31
Epoch 2/30
73088/73257 [============================>.] - ETA: 0s - loss: 1.2301 - accuracy: 0.60 - ETA: 0
Epoch 00002: val_loss improved from 1.54935 to 1.18767, saving model to mlp_model.h5
73257/73257 [==============================] - 43s 591us/sample - loss: 1.2292 - accuracy: 0.60
Epoch 3/30
73088/73257 [============================>.] - ETA: 0s - loss: 1.0423 - accuracy: 0.6748
Epoch 00003: val_loss improved from 1.18767 to 1.06973, saving model to mlp_model.h5
73257/73257 [==============================] - 42s 577us/sample - loss: 1.0422 - accuracy: 0.67
Epoch 4/30
73088/73257 [============================>.] - ETA: 0s - loss: 0.9467 - accuracy: 0.7053
Epoch 00004: val_loss improved from 1.06973 to 1.01101, saving model to mlp_model.h5
73257/73257 [==============================] - 42s 579us/sample - loss: 0.9467 - accuracy: 0.70
Epoch 5/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.8780 - accuracy: 0.7276
Epoch 00005: val_loss improved from 1.01101 to 1.00514, saving model to mlp_model.h5
73257/73257 [==============================] - 43s 584us/sample - loss: 0.8778 - accuracy: 0.72
Epoch 6/30
73088/73257 [============================>.] - ETA: 0s - loss: 0.8209 - accuracy: 0.7443
Epoch 00006: val_loss improved from 1.00514 to 0.95072, saving model to mlp_model.h5
73257/73257 [==============================] - 43s 583us/sample - loss: 0.8205 - accuracy: 0.74
Epoch 7/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.7957 - accuracy: 0.7516
Epoch 00007: val_loss did not improve from 0.95072
73257/73257 [==============================] - 42s 580us/sample - loss: 0.7960 - accuracy: 0.75
Epoch 8/30
73088/73257 [============================>.] - ETA: 0s - loss: 0.7536 - accuracy: 0.7672
Epoch 00008: val_loss improved from 0.95072 to 0.84198, saving model to mlp_model.h5
73257/73257 [==============================] - 43s 583us/sample - loss: 0.7536 - accuracy: 0.76
Epoch 9/30
73088/73257 [============================>.] - ETA: 0s - loss: 0.7283 - accuracy: 0.7732
Epoch 00009: val_loss did not improve from 0.84198
73257/73257 [==============================] - 43s 594us/sample - loss: 0.7282 - accuracy: 0.77
Epoch 10/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.7011 - accuracy: 0.7806
Epoch 00010: val_loss improved from 0.84198 to 0.80137, saving model to mlp_model.h5
73257/73257 [==============================] - 43s 583us/sample - loss: 0.7011 - accuracy: 0.78
Epoch 11/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.6746 - accuracy: 0.7904
Epoch 00011: val_loss improved from 0.80137 to 0.77974, saving model to mlp_model.h5
73257/73257 [==============================] - 43s 591us/sample - loss: 0.6747 - accuracy: 0.79
Epoch 12/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.6607 - accuracy: 0.7941 ETA: 1
Epoch 00012: val_loss did not improve from 0.77974
```

```
73257/73257 [==============================] - 43s 588us/sample - loss: 0.6607 - accuracy: 0.79
Epoch 13/30
73088/73257 [============================>.] - ETA: 0s - loss: 0.6357 - accuracy: 0.8009
Epoch 00013: val_loss improved from 0.77974 to 0.77085, saving model to mlp_model.h5
73257/73257 [==============================] - 43s 586us/sample - loss: 0.6357 - accuracy: 0.80
Epoch 14/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.6239 - accuracy: 0.8049
Epoch 00014: val_loss improved from 0.77085 to 0.76856, saving model to mlp_model.h5
73257/73257 [==============================] - 43s 587us/sample - loss: 0.6238 - accuracy: 0.80
Epoch 15/30
73088/73257 [============================>.] - ETA: 0s - loss: 0.5983 - accuracy: 0.8133
Epoch 00015: val_loss improved from 0.76856 to 0.74679, saving model to mlp_model.h5
73257/73257 [==============================] - 43s 586us/sample - loss: 0.5984 - accuracy: 0.81
Epoch 16/30
73088/73257 [============================>.] - ETA: 0s - loss: 0.5932 - accuracy: 0.8132
Epoch 00016: val_loss improved from 0.74679 to 0.73286, saving model to mlp_model.h5
73257/73257 [==============================] - 43s 589us/sample - loss: 0.5932 - accuracy: 0.81
Epoch 17/30
73088/73257 [============================>.] - ETA: 0s - loss: 0.5783 - accuracy: 0.8185 ETA:
Epoch 00017: val_loss did not improve from 0.73286
73257/73257 [==============================] - 43s 588us/sample - loss: 0.5784 - accuracy: 0.81
Epoch 18/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.5626 - accuracy: 0.8248
Epoch 00018: val_loss did not improve from 0.73286
73257/73257 [==============================] - 43s 590us/sample - loss: 0.5625 - accuracy: 0.82
Epoch 19/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.5508 - accuracy: 0.8273 ETA:
Epoch 00019: val_loss improved from 0.73286 to 0.73007, saving model to mlp_model.h5
73257/73257 [==============================] - 43s 584us/sample - loss: 0.5508 - accuracy: 0.82
Epoch 20/30
73088/73257 [============================>.] - ETA: 0s - loss: 0.5422 - accuracy: 0.8301
Epoch 00020: val_loss improved from 0.73007 to 0.71365, saving model to mlp_model.h5
73257/73257 [==============================] - 43s 587us/sample - loss: 0.5425 - accuracy: 0.83
Epoch 21/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.5259 - accuracy: 0.8333
Epoch 00021: val_loss improved from 0.71365 to 0.69172, saving model to mlp_model.h5
73257/73257 [==============================] - 43s 587us/sample - loss: 0.5259 - accuracy: 0.83
Epoch 22/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.5236 - accuracy: 0.8354
Epoch 00022: val_loss did not improve from 0.69172
73257/73257 [==============================] - 42s 580us/sample - loss: 0.5236 - accuracy: 0.83
Epoch 23/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.5099 - accuracy: 0.8381
Epoch 00023: val_loss did not improve from 0.69172
73257/73257 [==============================] - 43s 583us/sample - loss: 0.5099 - accuracy: 0.83
Epoch 24/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.5006 - accuracy: 0.8411 ETA:
Epoch 00024: val_loss did not improve from 0.69172
```

```
73257/73257 [==============================] - 43s 580us/sample - loss: 0.5005 - accuracy: 0.84
Epoch 25/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.4894 - accuracy: 0.8459
Epoch 00025: val_loss did not improve from 0.69172
73257/73257 [==============================] - 43s 582us/sample - loss: 0.4893 - accuracy: 0.84
Epoch 26/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.4831 - accuracy: 0.8471
Epoch 00026: val_loss did not improve from 0.69172
73257/73257 [==============================] - 43s 586us/sample - loss: 0.4831 - accuracy: 0.84
Epoch 27/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.4728 - accuracy: 0.8519
Epoch 00027: val_loss improved from 0.69172 to 0.69095, saving model to mlp_model.h5
73257/73257 [==============================] - 42s 578us/sample - loss: 0.4727 - accuracy: 0.85
Epoch 28/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.4678 - accuracy: 0.8523
Epoch 00028: val_loss did not improve from 0.69095
73257/73257 [==============================] - 43s 584us/sample - loss: 0.4679 - accuracy: 0.85
Epoch 29/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.4602 - accuracy: 0.8552
Epoch 00029: val_loss did not improve from 0.69095
73257/73257 [==============================] - 43s 591us/sample - loss: 0.4601 - accuracy: 0.85
Epoch 30/30
73088/73257 [============================>.] - ETA: 0s - loss: 0.4507 - accuracy: 0.8567
Epoch 00030: val_loss did not improve from 0.69095
73257/73257 [==============================] - 42s 579us/sample - loss: 0.4508 - accuracy: 0.85
```

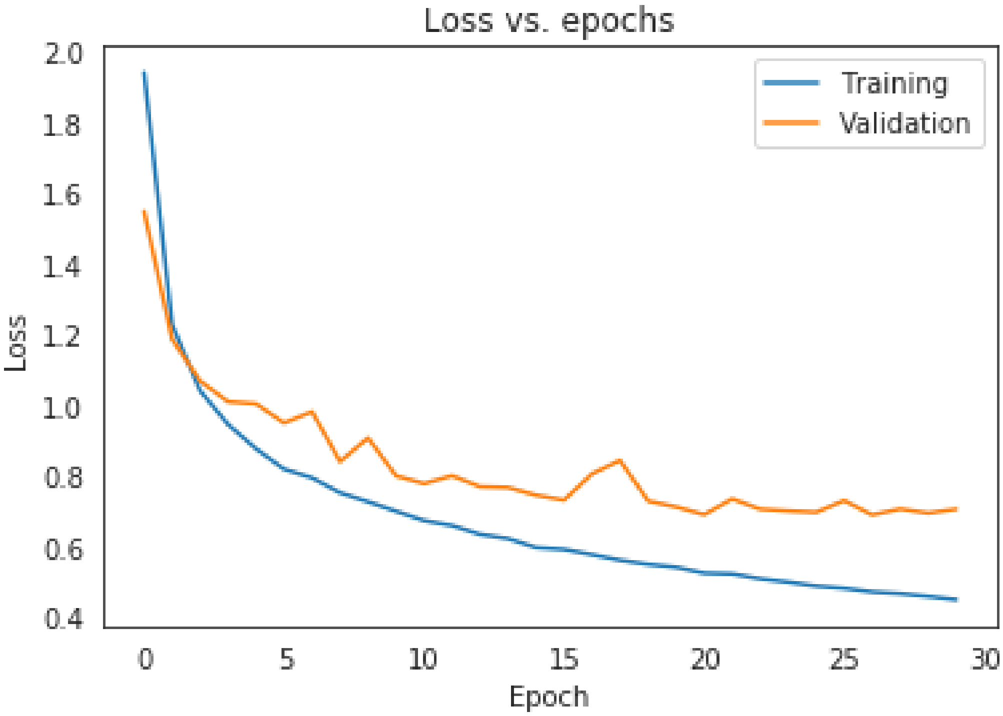
In [17]:
```
df = pd.DataFrame(history.history)
df.tail(10)
```

Out[17]:

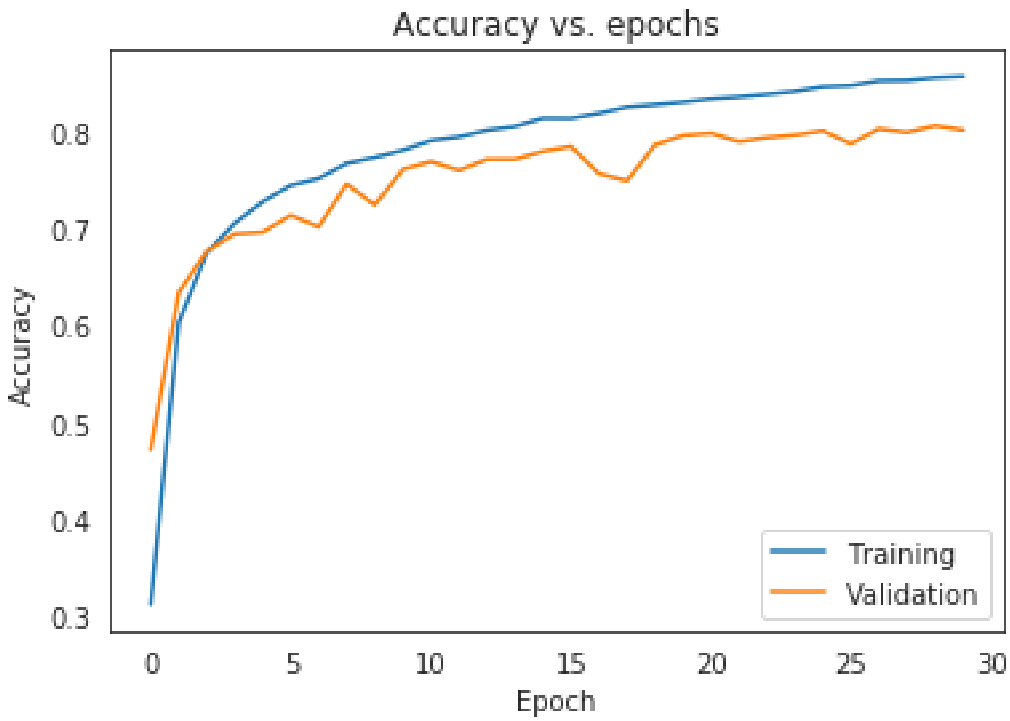
	loss	accuracy	val_loss	val_accuracy
20	0.525938	0.833354	0.691724	0.797941
21	0.523641	0.835374	0.736311	0.789183
22	0.509937	0.838091	0.707300	0.793331
23	0.500456	0.841135	0.702182	0.795943
24	0.489301	0.845912	0.698412	0.800323
25	0.483087	0.847073	0.731530	0.786993
26	0.472740	0.851878	0.690947	0.802359
27	0.467890	0.852274	0.707651	0.798748
28	0.460082	0.855222	0.696088	0.805624
29	0.450791	0.856642	0.707214	0.801206

In [18]:
```
# Plot the loss for training and validation sets
plt.plot(df.loss)
plt.plot(df.val_loss)
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
```

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




```
In [19]: # Plot the accuracy for training and validation sets
         plt.plot(df.accuracy)
         plt.plot(df.val_accuracy)
         plt.title('Accuracy vs. epochs')
         plt.ylabel('Accuracy')
         plt.xlabel('Epoch')
         plt.legend(['Training', 'Validation'], loc='lower right')
         plt.show()
```




```
In [20]: test_loss, test_acc = mlp_model.evaluate(test_x_gray, test_y, verbose=0)
         print("Test loss: {:.3f}\nTest accuracy: {:.2f}%".format(test_loss, 100 * test_acc))
```

```
Test loss: 0.707
Test accuracy: 80.12%
```

```
In [21]: # Load the best model
         mlp_best = get_mlp_model()
         mlp_best.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_5 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_6 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290
```

```
=================================================================
Total params: 690,314
Trainable params: 690,314
Non-trainable params: 0
_________________________________________________________________
```

```
In [22]: # Compile the model
         mlp_best.compile(optimizer='adam',
                          loss='sparse_categorical_crossentropy',
                          metrics=['accuracy'])
         mlp_best.load_weights('mlp_model.h5')
```

```
In [23]: test_loss, test_acc = mlp_best.evaluate(test_x_gray, test_y, verbose=0)
         print("Test loss: {:.3f}\nTest accuracy: {:.2f}%".format(test_loss, 100 * test_acc))
```

```
Test loss: 0.691
Test accuracy: 80.24%
```

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 [24]: cnn_model = Sequential([
                                Conv2D(16, 3, padding='SAME', activation='relu', input_shape=
                                MaxPooling2D(2),
                                BatchNormalization(),
                                Dropout(0.3),
                                Conv2D(32, 3, padding='SAME', activation='relu'),
                                MaxPooling2D(2),
                                BatchNormalization(),
                                Conv2D(64, 3, padding='SAME', activation='relu'),
                                MaxPooling2D(2),
                                Dropout(0.3),
```

```
                                Flatten(),
                                Dense(64, activation='relu'),
                                Dense(10, activation='softmax')
        ])

        cnn_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 16)        160
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 16)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 16)        64
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 32)          128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 64)          18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                65600
_________________________________________________________________
dense_11 (Dense)             (None, 10)                650
=================================================================
Total params: 89,738
Trainable params: 89,642
Non-trainable params: 96
_________________________________________________________________
```

```
In [25]: cnn_model.compile(optimizer='adam',
                           loss='sparse_categorical_crossentropy',
                           metrics=['accuracy'])
```

```
In [26]: checkpoint_cnn = ModelCheckpoint(filepath='CNN.h5',
                                          save_best_only=True,
```

```
                                        save_weights_only=False,
                                        monitor='val_accuracy',
                                        mode='max',
                                        verbose=1)
early_stop_cnn = EarlyStopping(monitor='loss', patience=5, verbose=1)
```

```
In [27]: history_cnn = cnn_model.fit(train_x_gray,
                                     train_y,
                                     callbacks=[checkpoint_cnn, early_stop_cnn],
                                     batch_size=128,
                                     validation_data=(test_x_gray, test_y),
                                     epochs=30)
```

```
Train on 73257 samples, validate on 26032 samples
Epoch 1/30
73216/73257 [============================>.] - ETA: 0s - loss: 1.2097 - accuracy: 0.5945
Epoch 00001: val_accuracy improved from -inf to 0.76721, saving model to CNN.h5
73257/73257 [==============================] - 367s 5ms/sample - loss: 1.2094 - accuracy: 0.594
Epoch 2/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.5900 - accuracy: 0.8188
Epoch 00002: val_accuracy improved from 0.76721 to 0.86497, saving model to CNN.h5
73257/73257 [==============================] - 352s 5ms/sample - loss: 0.5899 - accuracy: 0.818
Epoch 3/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.4903 - accuracy: 0.8516
Epoch 00003: val_accuracy improved from 0.86497 to 0.87846, saving model to CNN.h5
73257/73257 [==============================] - 349s 5ms/sample - loss: 0.4903 - accuracy: 0.851
Epoch 4/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.4396 - accuracy: 0.8665
Epoch 00004: val_accuracy improved from 0.87846 to 0.88591, saving model to CNN.h5
73257/73257 [==============================] - 350s 5ms/sample - loss: 0.4397 - accuracy: 0.866
Epoch 5/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.4047 - accuracy: 0.8769
Epoch 00005: val_accuracy improved from 0.88591 to 0.88791, saving model to CNN.h5
73257/73257 [==============================] - 352s 5ms/sample - loss: 0.4048 - accuracy: 0.876
Epoch 6/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.3834 - accuracy: 0.8840
Epoch 00006: val_accuracy improved from 0.88791 to 0.90005, saving model to CNN.h5
73257/73257 [==============================] - 352s 5ms/sample - loss: 0.3834 - accuracy: 0.884
Epoch 7/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.3641 - accuracy: 0.8891
Epoch 00007: val_accuracy improved from 0.90005 to 0.90154, saving model to CNN.h5
73257/73257 [==============================] - 350s 5ms/sample - loss: 0.3639 - accuracy: 0.889
Epoch 8/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.3508 - accuracy: 0.8934
Epoch 00008: val_accuracy did not improve from 0.90154
73257/73257 [==============================] - 350s 5ms/sample - loss: 0.3508 - accuracy: 0.893
Epoch 9/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.3388 - accuracy: 0.8955
```

```
Epoch 00009: val_accuracy improved from 0.90154 to 0.91065, saving model to CNN.h5
73257/73257 [==============================] - 346s 5ms/sample - loss: 0.3388 - accuracy: 0.895
Epoch 10/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.3283 - accuracy: 0.8984
Epoch 00010: val_accuracy did not improve from 0.91065
73257/73257 [==============================] - 354s 5ms/sample - loss: 0.3282 - accuracy: 0.898
Epoch 11/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.3184 - accuracy: 0.9039
Epoch 00011: val_accuracy did not improve from 0.91065
73257/73257 [==============================] - 348s 5ms/sample - loss: 0.3184 - accuracy: 0.903
Epoch 12/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.3134 - accuracy: 0.9054
Epoch 00012: val_accuracy did not improve from 0.91065
73257/73257 [==============================] - 352s 5ms/sample - loss: 0.3133 - accuracy: 0.905
Epoch 13/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.3023 - accuracy: 0.9082
Epoch 00013: val_accuracy improved from 0.91065 to 0.91276, saving model to CNN.h5
73257/73257 [==============================] - 350s 5ms/sample - loss: 0.3023 - accuracy: 0.908
Epoch 14/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2967 - accuracy: 0.9102
Epoch 00014: val_accuracy improved from 0.91276 to 0.91457, saving model to CNN.h5
73257/73257 [==============================] - 348s 5ms/sample - loss: 0.2966 - accuracy: 0.910
Epoch 15/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2868 - accuracy: 0.9129
Epoch 00015: val_accuracy did not improve from 0.91457
73257/73257 [==============================] - 352s 5ms/sample - loss: 0.2869 - accuracy: 0.912
Epoch 16/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2823 - accuracy: 0.9134
Epoch 00016: val_accuracy improved from 0.91457 to 0.91726, saving model to CNN.h5
73257/73257 [==============================] - 354s 5ms/sample - loss: 0.2823 - accuracy: 0.913
Epoch 17/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2774 - accuracy: 0.9142
Epoch 00017: val_accuracy improved from 0.91726 to 0.92356, saving model to CNN.h5
73257/73257 [==============================] - 354s 5ms/sample - loss: 0.2774 - accuracy: 0.914
Epoch 18/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2681 - accuracy: 0.9179
Epoch 00018: val_accuracy did not improve from 0.92356
73257/73257 [==============================] - 358s 5ms/sample - loss: 0.2682 - accuracy: 0.917
Epoch 19/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2676 - accuracy: 0.9170
Epoch 00019: val_accuracy did not improve from 0.92356
73257/73257 [==============================] - 359s 5ms/sample - loss: 0.2676 - accuracy: 0.917
Epoch 20/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2607 - accuracy: 0.9194
Epoch 00020: val_accuracy did not improve from 0.92356
73257/73257 [==============================] - 355s 5ms/sample - loss: 0.2608 - accuracy: 0.919
Epoch 21/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2553 - accuracy: 0.9221
```

```
Epoch 00021: val_accuracy did not improve from 0.92356
73257/73257 [==============================] - 354s 5ms/sample - loss: 0.2553 - accuracy: 0.922
Epoch 22/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2512 - accuracy: 0.9227
Epoch 00022: val_accuracy did not improve from 0.92356
73257/73257 [==============================] - 359s 5ms/sample - loss: 0.2511 - accuracy: 0.922
Epoch 23/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2481 - accuracy: 0.9236
Epoch 00023: val_accuracy did not improve from 0.92356
73257/73257 [==============================] - 347s 5ms/sample - loss: 0.2481 - accuracy: 0.923
Epoch 24/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2463 - accuracy: 0.9247
Epoch 00024: val_accuracy did not improve from 0.92356
73257/73257 [==============================] - 348s 5ms/sample - loss: 0.2463 - accuracy: 0.924
Epoch 25/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2413 - accuracy: 0.9248
Epoch 00025: val_accuracy did not improve from 0.92356
73257/73257 [==============================] - 354s 5ms/sample - loss: 0.2412 - accuracy: 0.924
Epoch 26/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2370 - accuracy: 0.9260
Epoch 00026: val_accuracy did not improve from 0.92356
73257/73257 [==============================] - 354s 5ms/sample - loss: 0.2370 - accuracy: 0.926
Epoch 27/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2357 - accuracy: 0.9264
Epoch 00027: val_accuracy did not improve from 0.92356
73257/73257 [==============================] - 360s 5ms/sample - loss: 0.2357 - accuracy: 0.926
Epoch 28/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2348 - accuracy: 0.9259
Epoch 00028: val_accuracy improved from 0.92356 to 0.92594, saving model to CNN.h5
73257/73257 [==============================] - 353s 5ms/sample - loss: 0.2347 - accuracy: 0.925
Epoch 29/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2291 - accuracy: 0.9279
Epoch 00029: val_accuracy did not improve from 0.92594
73257/73257 [==============================] - 354s 5ms/sample - loss: 0.2291 - accuracy: 0.927
Epoch 30/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.2263 - accuracy: 0.9292
Epoch 00030: val_accuracy did not improve from 0.92594
73257/73257 [==============================] - 352s 5ms/sample - loss: 0.2262 - accuracy: 0.929
```

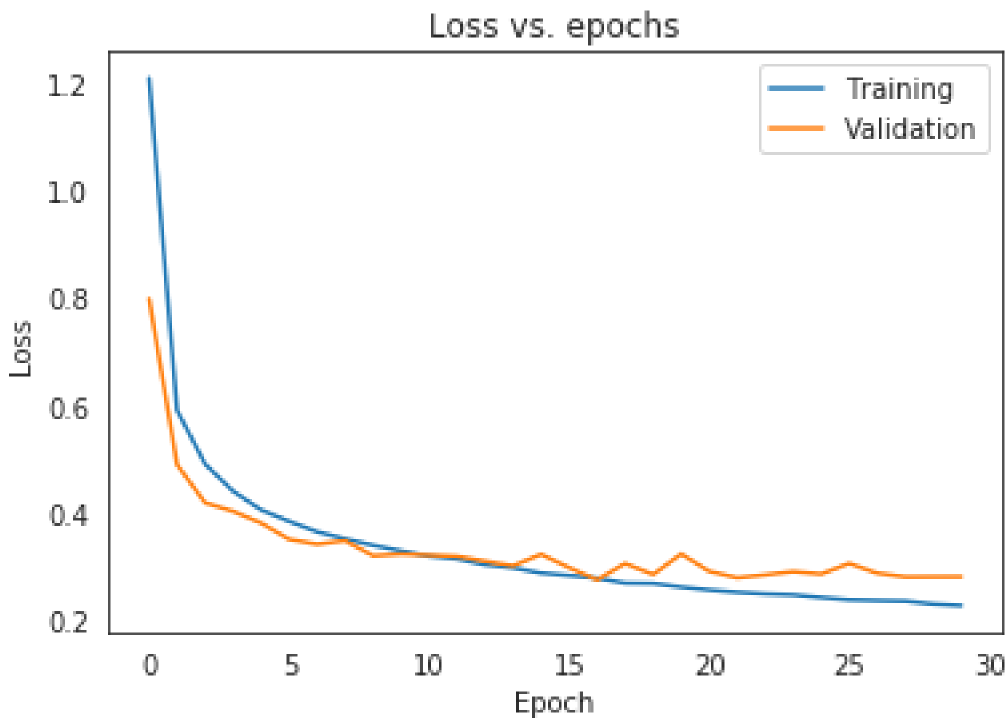
In [28]:
```
df_cnn = pd.DataFrame(history_cnn.history)
df_cnn.tail(10)
```

Out[28]:

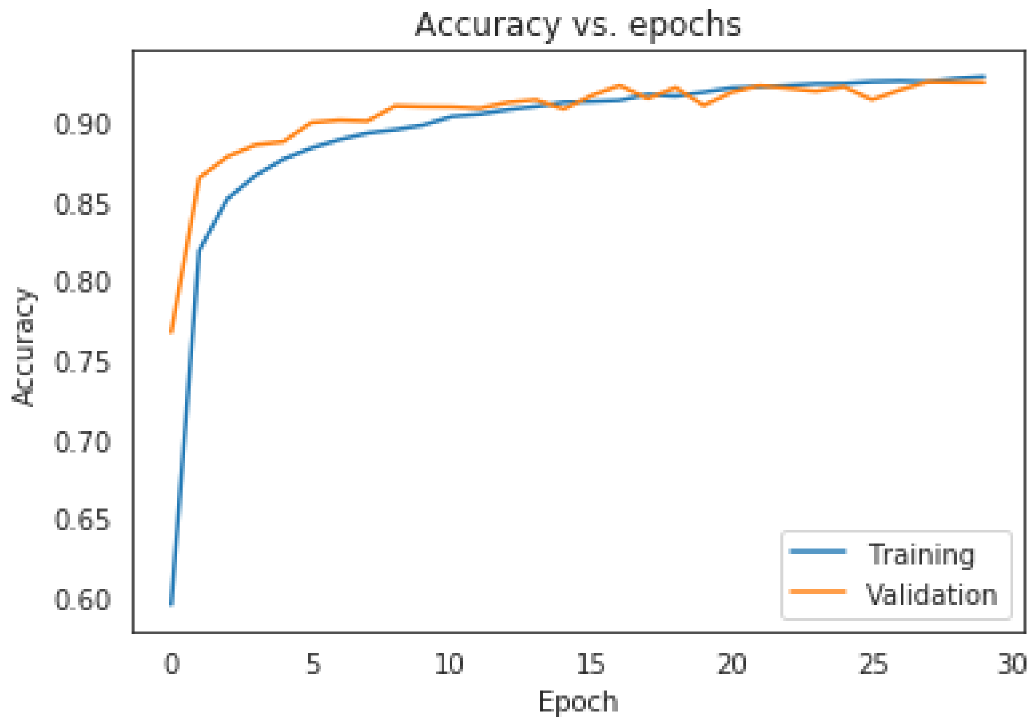
	loss	accuracy	val_loss	val_accuracy
20	0.255262	0.922042	0.290167	0.919292
21	0.251078	0.922765	0.278119	0.923517
22	0.248052	0.923611	0.283485	0.921827
23	0.246316	0.924676	0.289625	0.919945

24	0.241195	0.924826	0.285024	0.922711
25	0.236968	0.926028	0.304888	0.914605
26	0.235690	0.926423	0.286659	0.920713
27	0.234709	0.925946	0.279802	0.925937
28	0.229093	0.927884	0.280062	0.925668
29	0.226215	0.929181	0.280152	0.925668

In [29]:
```
# Plot the loss for training and validation sets
plt.plot(df_cnn.loss)
plt.plot(df_cnn.val_loss)
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```



In [30]:
```
# Plot the accuracy for training and validation sets
plt.plot(df_cnn.accuracy)
plt.plot(df_cnn.val_accuracy)
plt.title('Accuracy vs. epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='lower right')
plt.show()
```




```
In [31]: test_loss, test_acc = cnn_model.evaluate(test_x_gray, test_y, verbose=0)
         print("Test loss: {:.3f}\nTest accuracy: {:.2f}%".format(test_loss, 100 * test_acc))
```

```
Test loss: 0.280
Test accuracy: 92.57%
```

```
In [25]: cnn_best = load_model('CNN.h5')
         cnn_best.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 16)        160
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 16)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 16)        64
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 32)        4640
_________________________________________________________________
```

```
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 32)          128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 64)          18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                65600
_________________________________________________________________
dense_11 (Dense)             (None, 10)                650
=================================================================
Total params: 89,738
Trainable params: 89,642
Non-trainable params: 96
_________________________________________________________________
```

In [26]:
```
test_loss, test_acc = cnn_best.evaluate(test_x_gray, test_y, verbose=0)
print("Test loss: {:.3f}\nTest accuracy: {:.2f}%".format(test_loss, 100 * test_acc))
```

```
Test loss: 0.280
Test accuracy: 92.59%
```

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]:
```
# load MLP best model

# Create a new model with the same archetecture
mlp_best = get_mlp_model()

# Compile the model
mlp_best.compile(optimizer='adam',
                 loss='sparse_categorical_crossentropy',
                 metrics=['accuracy'])

# Load the model weights
mlp_best.load_weights('mlp_model.h5')
```

In [27]:
```
# Load CNN best model
cnn_best = load_model('CNN.h5')
```

In [28]:
```
# Get random indices for test
np.random.seed(0)
idx = np.random.choice(test_x_gray.shape[0]-1, 5)

# Get test images and labels
test_images = test_x_gray[idx]
test_labels = test_y[idx]

# Get MLP and CNN predictions for test images
mlp_predictions = mlp_best.predict(test_images)
cnn_predictions = cnn_best.predict(test_images)
```

In [29]:
```
# Show the test images and the MLP model predictions
print('MLP Test Examples')
plot_images(test_images, 1, 5, test_labels, np.argmax(mlp_predictions, 1))
```

```
MLP Test Examples
```

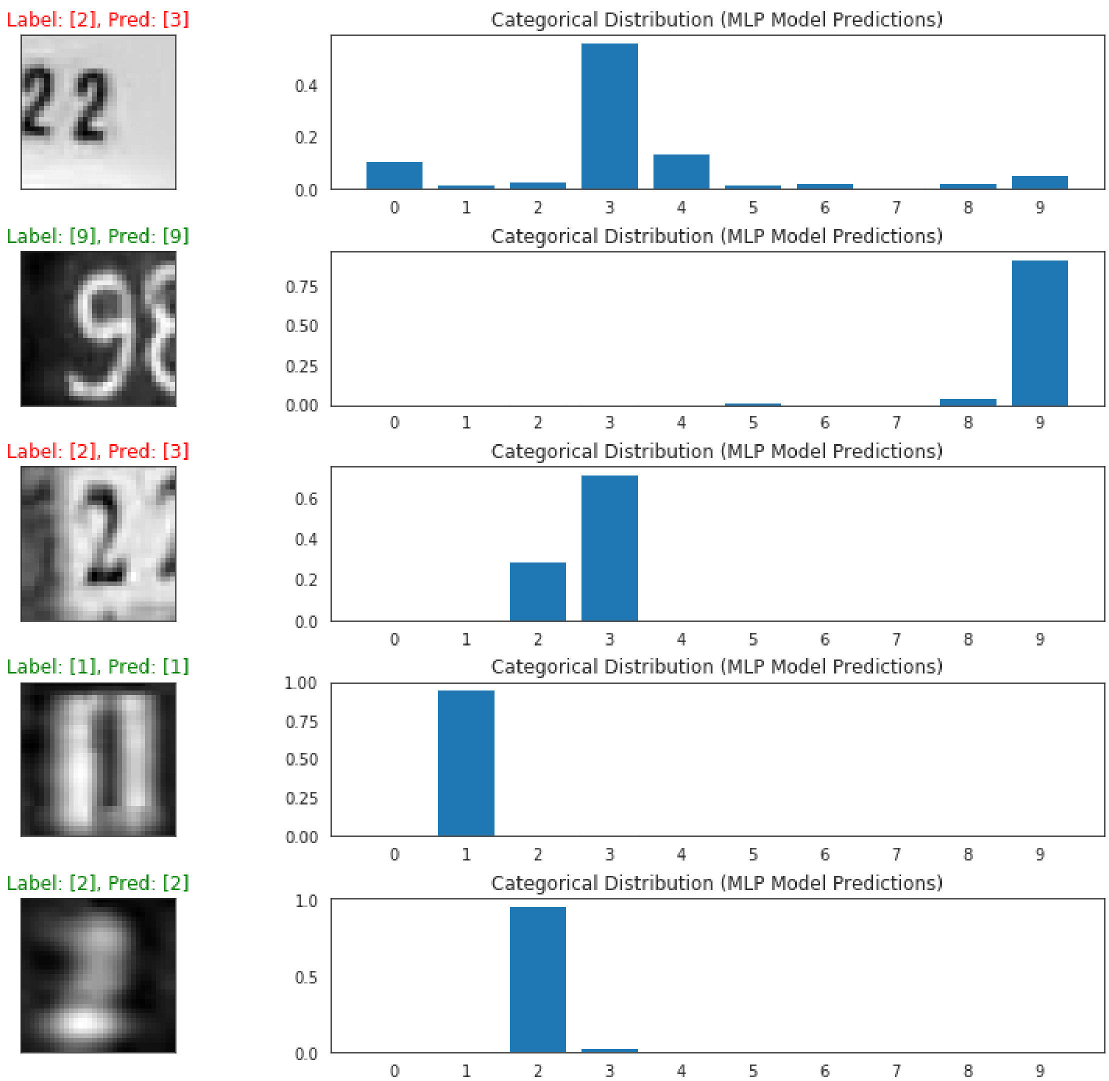
In [30]:
```
# Show the test images and the CNN model predictions
print('CNN Test Examples')
plot_images(test_images, 1, 5, test_labels, np.argmax(cnn_predictions, 1))
```

```
CNN Test Examples
```

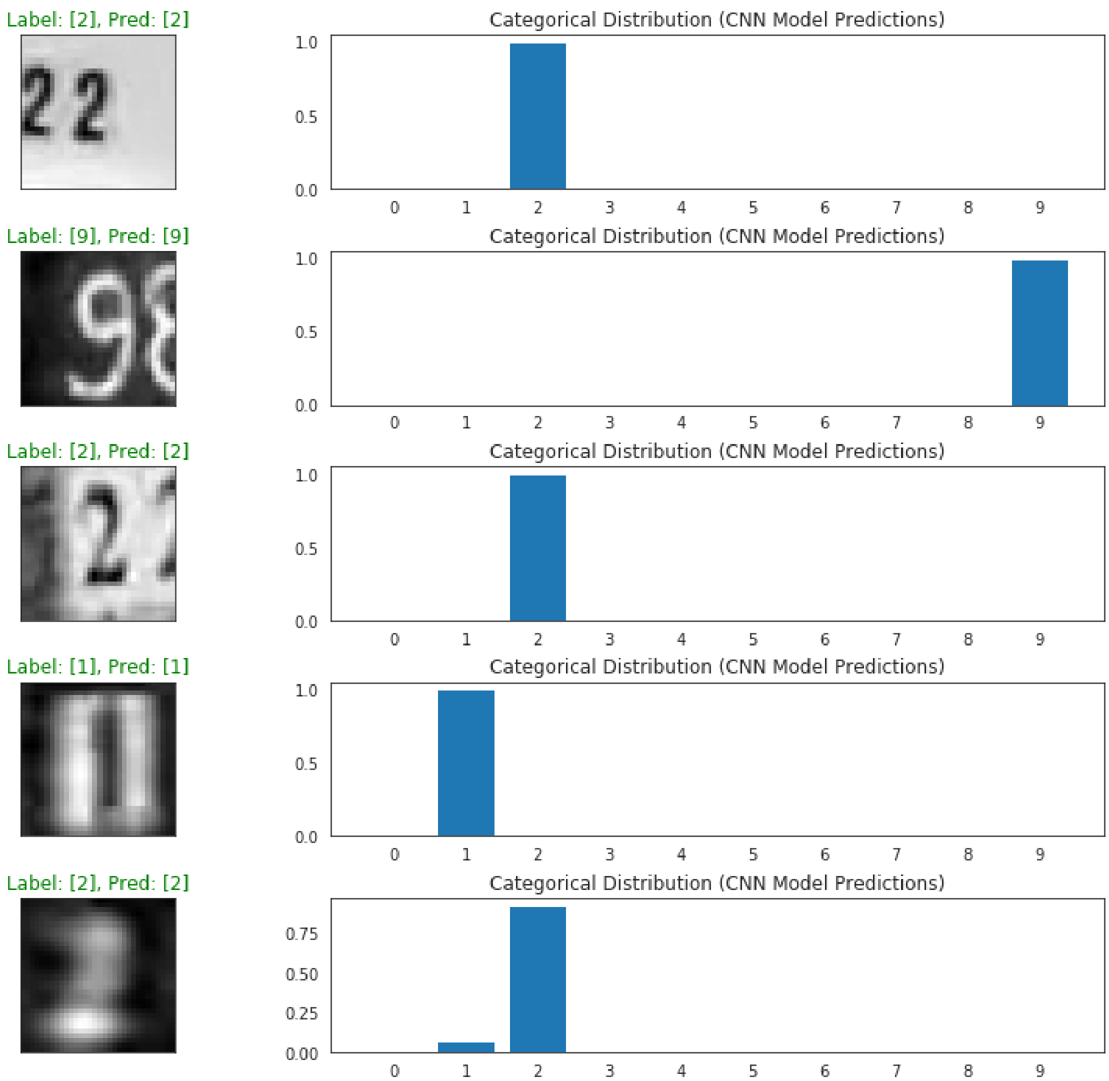
In [31]:
```
def plot_predictions(predictions, test_images, test_labels, title):
    """Function to plot the predictions and the categorical probabilty disturibution.
```

```
    param: predictions: array of shape (num_images, categories)
    param: test_images: test images to plot
    param: test_labels: true labels
    param: title: the kind of model for plot tutle
"""
# Show the images and a bar chart with the predictions probabilty
fig, axes = plt.subplots(5, 2, figsize=(16, 12))
fig.subplots_adjust(hspace=0.4, wspace=-0.2)

for i, (prediction, image, label) in enumerate(zip(predictions, test_images, test_
    predicted_label = np.argmax(prediction)
    image_title = f"Label: {label}, Pred: [{predicted_label}]"
    axes[i, 0].imshow(image[:, :, 0], cmap='gray')
    axes[i, 0].get_xaxis().set_visible(False)
    axes[i, 0].get_yaxis().set_visible(False)
    if label == predicted_label:
        axes[i, 0].set_title(image_title, color='g')
    else:
        axes[i, 0].set_title(image_title, color='r')
    axes[i, 1].bar(np.arange(10), prediction)
    axes[i, 1].set_xticks(np.arange(10))
    axes[i, 1].set_title(f"Categorical Distribution ({title} Model Predictions)")

plt.show()
```

In [32]:
```
plot_predictions(mlp_predictions, test_images, test_labels, 'MLP')
```




```
In [33]: plot_predictions(cnn_predictions, test_images, test_labels, 'CNN')
```


Label: [2], Pred: [2]
Categorical Distribution (CNN Model Predictions)
Label: [9], Pred: [9]
Categorical Distribution (CNN Model Predictions)
Label: [2], Pred: [2]
Categorical Distribution (CNN Model Predictions)
Label: [1], Pred: [1]
Categorical Distribution (CNN Model Predictions)
Label: [2], Pred: [2]
Categorical Distribution (CNN Model Predictions)