Capstone Project

Image classifier for the SVHN dataset

Instructions

In this notebook, you will create a neural network that classifies real-world images digits. You will use concepts from throughout this course in building, training, testing, validating and saving your Tensorflow classifier model.

This project is peer-assessed. Within this notebook you will find instructions in each section for how to complete the project. Pay close attention to the instructions as the peer review will be carried out according to a grading rubric that checks key parts of the project instructions. Feel free to add extra cells into the notebook as required.

How to submit

When you have completed the Capstone project notebook, you will submit a pdf of the notebook for peer review. First ensure that the notebook has been fully executed from beginning to end, and all of the cell outputs are visible. This is important, as the grading rubric depends on the reviewer being able to view the outputs of your notebook. Save the notebook as a pdf (File -> Download as -> PDF via LaTeX). You should then submit this pdf for review.

Let's get started!

We'll start by running some imports, and loading the dataset. For this project you are free to make further imports throughout the notebook as you wish.

In [0]:
```
import tensorflow as tf
from scipy.io import loadmat
```

SVHN overview image
For the capstone project, you will use the [SVHN dataset (http://ufldl.stanford.edu/housenumbers/)](http://ufldl.stanford.edu/housenumbers/). 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 [4]:
```
# from google.colab import drive
# drive.mount('/content/drive')

# import sys
# import os
# path='/content/drive/My Drive/INSAID/TensorFlow/Getting started with TensorF
low 2/Week5/notebooks'
# sys.path.append(path)
# os.chdir(path)
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client
_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&
redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=ema
il%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.
googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdri
ve.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.read
only

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
# 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.

In [8]:
```
# Let's see the keys in the dictionary

print(train.keys())
print(test.keys())
```

```
dict_keys(['__header__', '__version__', '__globals__', 'X', 'y'])
dict_keys(['__header__', '__version__', '__globals__', 'X', 'y'])
```

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.

Extract the training and testing images and labels separately from the train and test dictionaries

In [9]:
```
# Extracting training and test images and corresponding labels

train_images, train_labels = train['X'], train['y']
test_images, test_labels = test['X'], test['y']

print("Train Data : ",train_images.shape, train_labels.shape)
print("Test Data : ",test_images.shape, test_labels.shape)
```

```
Train Data :  (32, 32, 3, 73257) (73257, 1)
Test Data :  (32, 32, 3, 26032) (26032, 1)
```

No. of samples is the 4th dimension, lets bring it back as the first dimesion (as usual)

In [10]:
```
train_images = train_images.transpose((3, 0, 1, 2))
test_images = test_images.transpose((3, 0, 1, 2))
print("Train Data : ",train_images.shape, train_labels.shape)
print("Test Data : ",test_images.shape, test_labels.shape)
```

```
Train Data :  (73257, 32, 32, 3) (73257, 1)
Test Data :  (26032, 32, 32, 3) (26032, 1)
```

In [0]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
In [0]: # Plot a random sample of images (alteast 10)

def plot_images(image, labels, row=1, col=10):
    '''
    Plot a random row * col images.
    '''
    fig, ax = plt.subplots(row, col)

    indexes = np.random.randint(1, image.shape[0], row*col)

    x=0;
    for i, ax in enumerate(ax.flat):
        img_index = indexes[i]

        if image[img_index].shape == (32,32,3):
            ax.imshow(image[img_index])
            ax.set_title(labels[img_index][0])
        else:
            ax.imshow(image[img_index, :, :, 0])
            ax.set_title(np.argmax(labels[img_index]))

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

#         plt.tight_layout()
```

Select a random sample of images and corresponding labels from the dataset (at least 10)

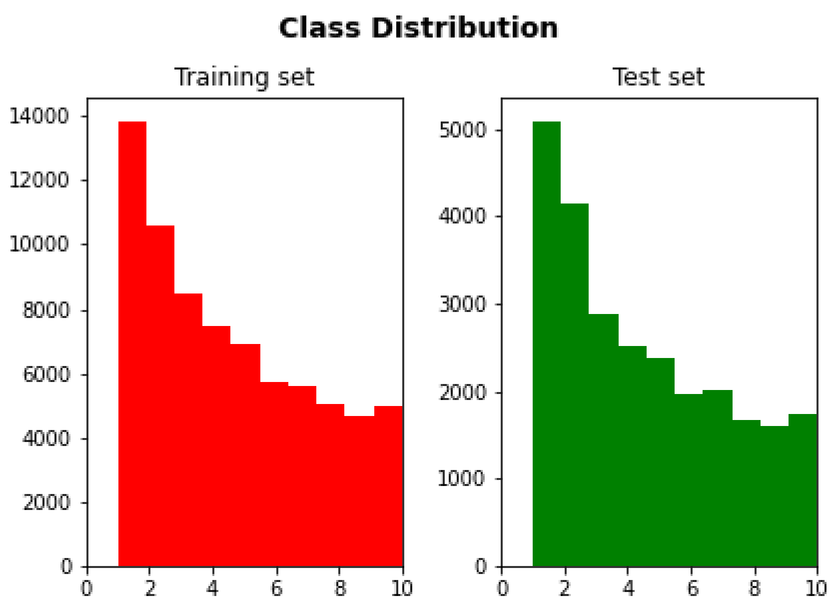
```
In [13]: # Plot 10 train images

plot_images(train_images, train_labels)
```


```
In [14]: # Plot 10 test images

plot_images(test_images, test_labels)
```

In [15]:
```python
# Lets check how may unique numbers in output

print('Train :',np.unique(train_labels))
print('Test :',np.unique(test_labels))
```

```
Train : [ 1  2  3  4  5  6  7  8  9 10]
Test : [ 1  2  3  4  5  6  7  8  9 10]
```

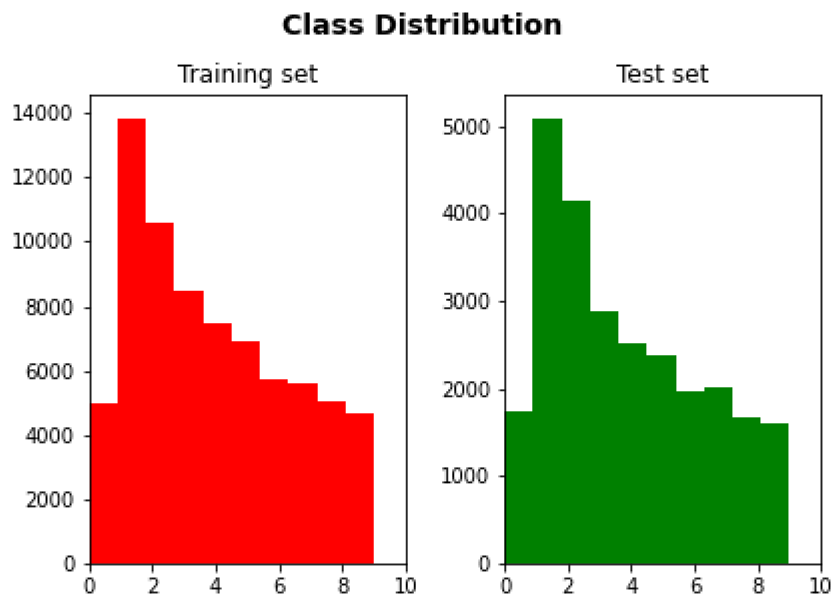
In [16]:
```python
# Let see the distribution of the labels/classes
def plot_class_distribution(ytrain, ytest):
    fig, (ax1, ax2) = plt.subplots(1, 2, sharex=True)
    fig.suptitle('Class Distribution', fontsize=14, fontweight='bold', y=1.05)

    ax1.hist(ytrain, bins=10, color='r')
    ax1.set_title("Training set")
    ax1.set_xlim(0, 10)

    ax2.hist(ytest, bins=10, color='g')
    ax2.set_title("Test set")

    fig.tight_layout()

plot_class_distribution(train_labels, test_labels)
```



Both the plots show right skewness, which means we have high number of images for lower values when compared to high values

In [0]:
```python
# For simpicity of programming, lets convert the class 10 as class 0
test_labels[test_labels == 10] = 0
train_labels[train_labels == 10] = 0
```

In [18]:
```
plot_class_distribution(train_labels, test_labels)
```



Convert the training and test images to grayscale by taking the average across all colour channels for each pixel

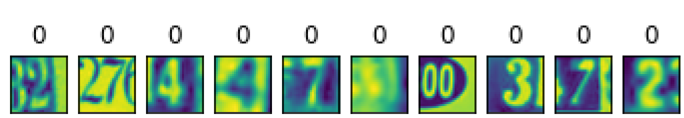
In [0]:
```
# Convert the training and test images to grayscale by taking the average acro
ss all colour channels for each pixel

def rgb2grayscale(images):
    return np.expand_dims(np.dot(images, [0.2990, 0.5870, 0.1140]), axis=image
s.ndim-1)
```

In [20]:
```
train_images_grayscale = rgb2grayscale(train_images)
test_images_grayscale = rgb2grayscale(test_images)

print("Train (grayscale) :", train_images_grayscale.shape)
print("Test (grayscale) :", test_images_grayscale.shape)
```

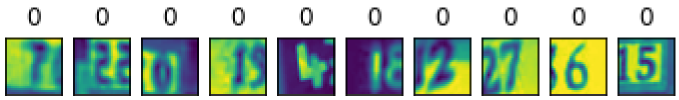
```
Train (grayscale) : (73257, 32, 32, 1)
Test (grayscale) : (26032, 32, 32, 1)
```

Select a random sample of the grayscale images and corresponding labels from the dataset (at least 10)

In [21]:
```
# Lets plot the gray scaled images

plot_images(train_images_grayscale, train_labels)
```



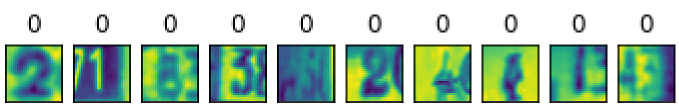
In [22]:
```
plot_images(test_images_grayscale, test_labels)
```



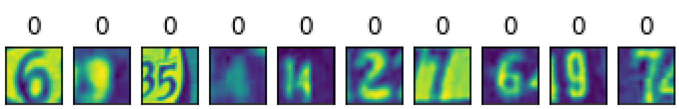
Data Normalization

In [0]:
```
# Lets normalize the data-set

# Mean and Std.Dev values
train_mean = np.mean(train_images_grayscale, axis=0)
train_std = np.std(train_images_grayscale, axis=0)

# Subt. mean and div. by std. dev
train_images_norm = (train_images_grayscale - train_mean) / train_std
test_images_norm = (test_images_grayscale - train_mean) / train_std
```

In [24]:
```
# Lets plot the gray scaled normalized images

plot_images(train_images_norm, train_labels)
```



In [25]:
```
plot_images(test_images_norm, test_labels)
```



Now, lets split the train images into train set and validation set

In [0]:
```
# Splitting the Training data into Train and Validation sets.
# 13% of train set gives around 9500 data having min. of 800 instances of each
class
# Using random state to regenrate the whole Dataset in re-run

from sklearn.model_selection import train_test_split

xtrain, xval, ytrain, yval = train_test_split(train_images_norm, train_labels,
test_size=.13, random_state=42)
xtest = test_images_norm # for same naming convention
ytest = test_labels
```

One-Hot encoding target variable

In [0]:
```
# For model prediction purpose, lets one-hot encode the target varialble.
# Apply One Hot Encoding to make label suitable for CNN Classification

from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder().fit(ytrain.reshape(-1,1))

y_train = ohe.transform(ytrain.reshape(-1,1)).toarray()
y_val = ohe.transform(yval.reshape(-1,1)).toarray()
y_test = ohe.transform(ytest.reshape(-1,1)).toarray()
```

In [28]:
```
# ytrain.reshape(-1,1)
ohe.transform(ytrain[0].reshape(-1,1)).toarray()
```

Out[28]:
```
array([[0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

In [29]:
```
print('y_train :', y_train.shape)
print('y_val :', y_val.shape)
print('y_test :', y_test.shape)
```

```
y_train : (63733, 10)
y_val : (9524, 10)
y_test : (26032, 10)
```

Store the processed data to disk.

In [0]:
```
# Storing only the Grayscale Data not the RGB

import h5py

# Create file
h5f = h5py.File('SVHN_grey.h5', 'w')

# Store the datasets
h5f.create_dataset('X_train', data=xtrain)
h5f.create_dataset('y_train', data=y_train)
h5f.create_dataset('X_test', data=xtest)
h5f.create_dataset('y_test', data=y_test)
h5f.create_dataset('X_val', data=xval)
h5f.create_dataset('y_val', data=y_val)

# Close the file
h5f.close()
```

Free-up RAM memory

In [0]:
```
# Lets delete all the data loaded into memory to free up some RAM mem.

del y_train, y_val, y_test, xtrain, xtest, xval, ytrain, ytest, yval, train_im
ages_norm, test_images_norm, \
train_images_grayscale, test_images_grayscale, train_images, test_images, trai
n_labels, test_labels, train, test
```

2. MLP neural network classifier

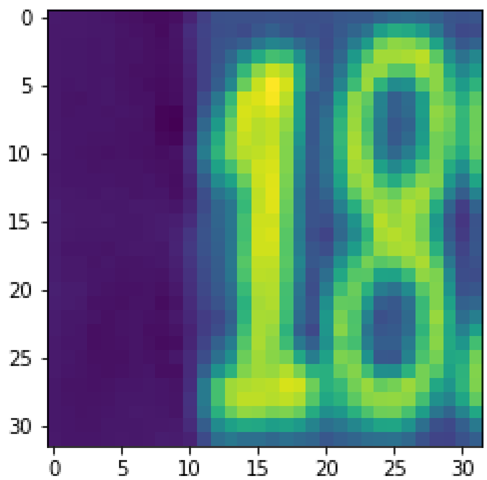
- Build an MLP classifier model using the Sequential API. Your model should use only Flatten and Dense layers, with the final layer having a 10-way softmax output.
- You should design and build the model yourself. Feel free to experiment with different MLP architectures. *Hint: to achieve a reasonable 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 [32]:
```
# Open file in read mode
import h5py
h5f = h5py.File('SVHN_grey.h5', 'r')

# Read the dataset into local variables
x_train = h5f['X_train'][:]
y_train = h5f['y_train'][:]
x_test = h5f['X_test'][:]
y_test = h5f['y_test'][:]
x_val = h5f['X_val'][:]
y_val = h5f['y_val'][:]

# Close file
h5f.close()

print('Training set', x_train.shape, y_train.shape)
print('Validation set', x_val.shape, y_val.shape)
print('Test set', x_test.shape, y_test.shape)
```

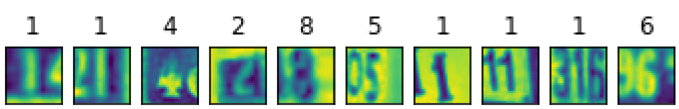
```
Training set (63733, 32, 32, 1) (63733, 10)
Validation set (9524, 32, 32, 1) (9524, 10)
Test set (26032, 32, 32, 1) (26032, 10)
```

In [33]:
```python
# Display one of the images

i = 0
labels = np.argmax(y_train[i])
img = x_train[i,:,:,0]
plt.imshow(img)
# plt.show()
print(f"label: {labels}")
```

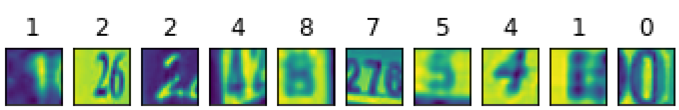
```
label: 1
```



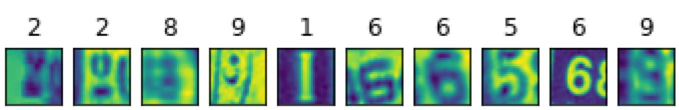
In [34]:
```python
plot_images(x_train, y_train)
```



In [35]:
```python
plot_images(x_test, y_test)
```



In [36]:
```python
plot_images(x_val, y_val)
```



In [0]:
```python
# import lib
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.callbacks import Callback, ModelCheckpoint, EarlyStoppin
g
from tensorflow.keras.optimizers import Adam
```

Build and compile the model

In [0]:
```
# Build an MLP classifier model using the Sequential API.
# Model should use only Flatten and Dense layers, with the final layer having
 a 10-way softmax output

def getModel(input_shape):
    model = Sequential([
        Flatten(input_shape=input_shape, name='Flatten'),
        Dense(128, activation='relu', name='Dense_1'),
        Dense(128, activation='relu', name='Dense_2'),
        Dense(128, activation='relu', name='Dense_3'),
        Dense(128, activation='relu', name='Dense_4'),
        Dense(128, activation='relu', name='Dense_5'),
        Dense(10, activation='softmax', name='Dense_6')
    ])

    model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_cros
sentropy', metrics=['accuracy'])
    return model
```

Print out the model summary

In [39]:
```
print(x_train[0].shape)
model = getModel(x_train[0].shape)
model.summary()
```

```
(32, 32, 1)
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
Dense_1 (Dense)              (None, 128)               131200
_________________________________________________________________
Dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
Dense_3 (Dense)              (None, 128)               16512
_________________________________________________________________
Dense_4 (Dense)              (None, 128)               16512
_________________________________________________________________
Dense_5 (Dense)              (None, 128)               16512
_________________________________________________________________
Dense_6 (Dense)              (None, 10)                1290
=================================================================
Total params: 198,538
Trainable params: 198,538
Non-trainable params: 0
_________________________________________________________________
```

In [40]:
```
print(f'Loss :{model.loss}')
print(f'Learning Rate :{model.optimizer.lr}, \nOptimizer: {model.optimizer}')
print(f'Mertrics : {model.metrics}')
```

```
Loss :categorical_crossentropy
Learning Rate :<tf.Variable 'learning_rate:0' shape=() dtype=float32, numpy=1
e-04>,
Optimizer: <tensorflow.python.keras.optimizer_v2.adam.Adam object at 0x7f7153
818b00>
Mertrics : []
```

Custom Callback

In [0]:
```
class my_callback(Callback):
    def on_train_begin(self, logs=None):
        print("Starting training....")

    def on_epoch_begin(self, epoch, logs=None):
        print(f"Starting epoch {epoch}")

    def on_epoch_end(self, epoch, logs=None):
        print(f"Finishing epoch {epoch}")

    def on_train_end(self, logs=None):
        print("Finished training:")

#  track at least one appropriate metric
def checkpoint_getBestOnly():
    checkpoint_path='model_checkpoint/checkpoint'
    checkpoint = ModelCheckpoint(filepath=checkpoint_path, save_freq='epoch',
                                 save_best_only=False, verbose=1,
                                 save_weights_only=True, monitor = 'val_accura
cy',
                                )
    return checkpoint
```

Train the model

In [42]:
```
history = model.fit(x_train, y_train, epochs=30,
                    validation_data=(x_val,y_val), batch_size=64,
                    callbacks=[my_callback(), checkpoint_getBestOnly()])
```

```
Starting training....
Starting epoch 0
Epoch 1/30
988/996 [============================>.] - ETA: 0s - loss: 1.5957 - accuracy:
0.4667Finishing epoch 0

Epoch 00001: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 1.5920 - accur
acy: 0.4681 - val_loss: 1.0949 - val_accuracy: 0.6602
Starting epoch 1
Epoch 2/30
988/996 [============================>.] - ETA: 0s - loss: 0.9828 - accuracy:
0.6974Finishing epoch 1

Epoch 00002: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.9826 - accur
acy: 0.6975 - val_loss: 0.8979 - val_accuracy: 0.7246
Starting epoch 2
Epoch 3/30
993/996 [============================>.] - ETA: 0s - loss: 0.8410 - accuracy:
0.7426Finishing epoch 2

Epoch 00003: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.8409 - accur
acy: 0.7426 - val_loss: 0.8147 - val_accuracy: 0.7517
Starting epoch 3
Epoch 4/30
991/996 [============================>.] - ETA: 0s - loss: 0.7591 - accuracy:
0.7703Finishing epoch 3

Epoch 00004: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.7590 - accur
acy: 0.7703 - val_loss: 0.7534 - val_accuracy: 0.7717
Starting epoch 4
Epoch 5/30
991/996 [============================>.] - ETA: 0s - loss: 0.7003 - accuracy:
0.7881Finishing epoch 4

Epoch 00005: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.7008 - accur
acy: 0.7880 - val_loss: 0.7375 - val_accuracy: 0.7753
Starting epoch 5
Epoch 6/30
991/996 [============================>.] - ETA: 0s - loss: 0.6564 - accuracy:
0.8000Finishing epoch 5

Epoch 00006: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.6577 - accur
acy: 0.7997 - val_loss: 0.6993 - val_accuracy: 0.7883
Starting epoch 6
Epoch 7/30
993/996 [============================>.] - ETA: 0s - loss: 0.6202 - accuracy:
0.8118Finishing epoch 6

Epoch 00007: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.6208 - accur
acy: 0.8117 - val_loss: 0.7001 - val_accuracy: 0.7870
```

```
Starting epoch 7
Epoch 8/30
985/996 [============================>.] - ETA: 0s - loss: 0.5895 - accuracy:
0.8216Finishing epoch 7

Epoch 00008: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.5893 - accur
acy: 0.8215 - val_loss: 0.6505 - val_accuracy: 0.8072
Starting epoch 8
Epoch 9/30
995/996 [============================>.] - ETA: 0s - loss: 0.5635 - accuracy:
0.8297Finishing epoch 8

Epoch 00009: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.5637 - accur
acy: 0.8296 - val_loss: 0.6427 - val_accuracy: 0.8109
Starting epoch 9
Epoch 10/30
995/996 [============================>.] - ETA: 0s - loss: 0.5427 - accuracy:
0.8346Finishing epoch 9

Epoch 00010: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.5427 - accur
acy: 0.8346 - val_loss: 0.6172 - val_accuracy: 0.8170
Starting epoch 10
Epoch 11/30
987/996 [============================>.] - ETA: 0s - loss: 0.5193 - accuracy:
0.8425Finishing epoch 10

Epoch 00011: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.5199 - accur
acy: 0.8425 - val_loss: 0.6046 - val_accuracy: 0.8236
Starting epoch 11
Epoch 12/30
996/996 [==============================] - ETA: 0s - loss: 0.5025 - accuracy:
0.8477Finishing epoch 11

Epoch 00012: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.5025 - accur
acy: 0.8477 - val_loss: 0.6068 - val_accuracy: 0.8232
Starting epoch 12
Epoch 13/30
987/996 [============================>.] - ETA: 0s - loss: 0.4850 - accuracy:
0.8528Finishing epoch 12

Epoch 00013: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.4847 - accur
acy: 0.8530 - val_loss: 0.5927 - val_accuracy: 0.8265
Starting epoch 13
Epoch 14/30
986/996 [============================>.] - ETA: 0s - loss: 0.4693 - accuracy:
0.8577Finishing epoch 13

Epoch 00014: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.4688 - accur
acy: 0.8578 - val_loss: 0.5829 - val_accuracy: 0.8322
Starting epoch 14
```

```
Epoch 15/30
988/996 [============================>.] - ETA: 0s - loss: 0.4533 - accuracy:
0.8619Finishing epoch 14

Epoch 00015: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.4537 - accur
acy: 0.8616 - val_loss: 0.5859 - val_accuracy: 0.8271
Starting epoch 15
Epoch 16/30
988/996 [============================>.] - ETA: 0s - loss: 0.4425 - accuracy:
0.8654Finishing epoch 15

Epoch 00016: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.4421 - accur
acy: 0.8656 - val_loss: 0.5847 - val_accuracy: 0.8320
Starting epoch 16
Epoch 17/30
992/996 [============================>.] - ETA: 0s - loss: 0.4300 - accuracy:
0.8688Finishing epoch 16

Epoch 00017: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.4301 - accur
acy: 0.8687 - val_loss: 0.5918 - val_accuracy: 0.8312
Starting epoch 17
Epoch 18/30
992/996 [============================>.] - ETA: 0s - loss: 0.4162 - accuracy:
0.8736Finishing epoch 17

Epoch 00018: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.4163 - accur
acy: 0.8736 - val_loss: 0.5700 - val_accuracy: 0.8394
Starting epoch 18
Epoch 19/30
990/996 [============================>.] - ETA: 0s - loss: 0.4055 - accuracy:
0.8766Finishing epoch 18

Epoch 00019: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.4057 - accur
acy: 0.8765 - val_loss: 0.5953 - val_accuracy: 0.8307
Starting epoch 19
Epoch 20/30
989/996 [============================>.] - ETA: 0s - loss: 0.3952 - accuracy:
0.8794Finishing epoch 19

Epoch 00020: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.3947 - accur
acy: 0.8795 - val_loss: 0.5711 - val_accuracy: 0.8370
Starting epoch 20
Epoch 21/30
986/996 [============================>.] - ETA: 0s - loss: 0.3852 - accuracy:
0.8815Finishing epoch 20

Epoch 00021: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.3855 - accur
acy: 0.8815 - val_loss: 0.5884 - val_accuracy: 0.8349
Starting epoch 21
Epoch 22/30
```

```
985/996 [============================>.] - ETA: 0s - loss: 0.3746 - accuracy: 
0.8849Finishing epoch 21

Epoch 00022: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.3749 - accur
acy: 0.8847 - val_loss: 0.5794 - val_accuracy: 0.8331
Starting epoch 22
Epoch 23/30
996/996 [==============================] - ETA: 0s - loss: 0.3666 - accuracy: 
0.8880Finishing epoch 22

Epoch 00023: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.3666 - accur
acy: 0.8880 - val_loss: 0.5669 - val_accuracy: 0.8402
Starting epoch 23
Epoch 24/30
984/996 [============================>.] - ETA: 0s - loss: 0.3537 - accuracy: 
0.8910Finishing epoch 23

Epoch 00024: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.3537 - accur
acy: 0.8911 - val_loss: 0.5882 - val_accuracy: 0.8347
Starting epoch 24
Epoch 25/30
996/996 [==============================] - ETA: 0s - loss: 0.3492 - accuracy: 
0.8934Finishing epoch 24

Epoch 00025: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.3492 - accur
acy: 0.8934 - val_loss: 0.5756 - val_accuracy: 0.8379
Starting epoch 25
Epoch 26/30
988/996 [============================>.] - ETA: 0s - loss: 0.3388 - accuracy: 
0.8957Finishing epoch 25

Epoch 00026: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.3387 - accur
acy: 0.8958 - val_loss: 0.5773 - val_accuracy: 0.8400
Starting epoch 26
Epoch 27/30
994/996 [============================>.] - ETA: 0s - loss: 0.3334 - accuracy: 
0.8988Finishing epoch 26

Epoch 00027: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.3333 - accur
acy: 0.8989 - val_loss: 0.5849 - val_accuracy: 0.8368
Starting epoch 27
Epoch 28/30
992/996 [============================>.] - ETA: 0s - loss: 0.3247 - accuracy: 
0.9005Finishing epoch 27

Epoch 00028: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.3247 - accur
acy: 0.9005 - val_loss: 0.6122 - val_accuracy: 0.8282
Starting epoch 28
Epoch 29/30
991/996 [============================>.] - ETA: 0s - loss: 0.3161 - accuracy: 
```

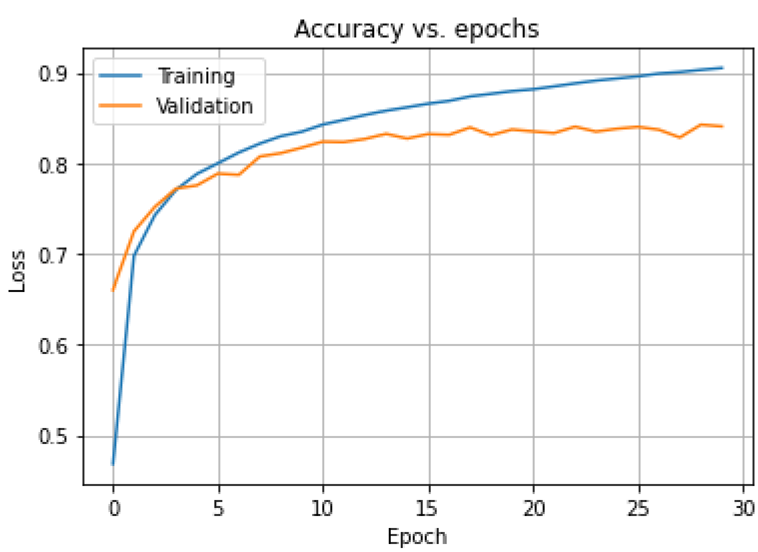
```
0.9029Finishing epoch 28

Epoch 00029: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.3158 - accur
acy: 0.9030 - val_loss: 0.5833 - val_accuracy: 0.8424
Starting epoch 29
Epoch 30/30
993/996 [============================>.] - ETA: 0s - loss: 0.3094 - accuracy:
0.9051Finishing epoch 29

Epoch 00030: saving model to model_checkpoint/checkpoint
996/996 [==============================] - 4s 4ms/step - loss: 0.3094 - accur
acy: 0.9050 - val_loss: 0.5847 - val_accuracy: 0.8405
Finished training:
```

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

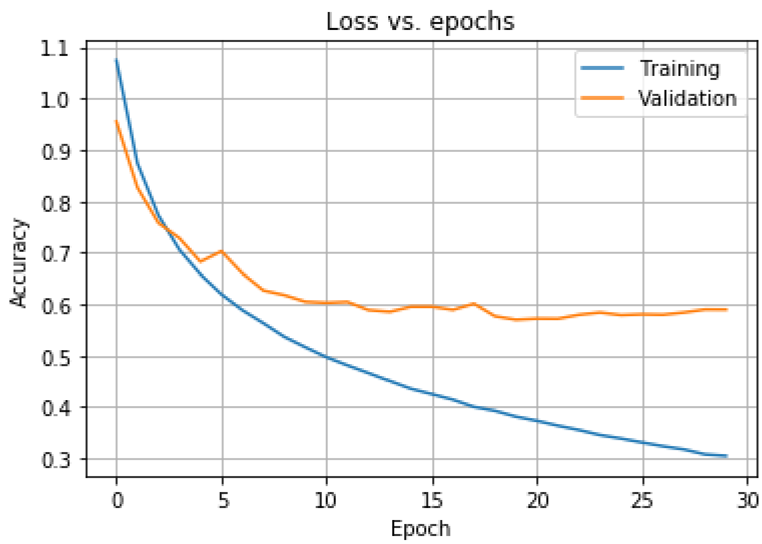
In [43]:
```python
print(history.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [44]:
```python
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Accuracy vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'])
plt.grid()
```



```
In [0]: 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'])
        plt.grid()
```




```
In [45]: import pandas as pd
         df = pd.DataFrame(history.history)
         df.head()
```

Out[45]:

	loss	accuracy	val_loss	val_accuracy
0	1.591985	0.468141	1.094862	0.660227
1	0.982615	0.697535	0.897930	0.724590
2	0.840916	0.742629	0.814694	0.751680
3	0.758958	0.770292	0.753439	0.771735
4	0.700814	0.788022	0.737491	0.775304

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

```
In [46]: loss, accuracy = model.evaluate(x_test, y_test, verbose=2)
         print(f'Loss : {loss:.3f} \nAccuracy :{accuracy:.3f}')
```

```
814/814 - 1s - loss: 0.7821 - accuracy: 0.8038
Loss : 0.782
Accuracy :0.804
```

Cleanup

In [47]:
```
!ls -lh model_checkpoint
# !rm -r model_checkpoint
```

```
total 2.3M
-rw------- 1 root root   77 May 20 20:15 checkpoint
-rw------- 1 root root 2.3M May 20 20:15 checkpoint.data-00000-of-00001
-rw------- 1 root root 2.8K May 20 20:15 checkpoint.index
```

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 [0]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Batc
hNormalization, Dropout
from tensorflow.keras.callbacks import Callback, EarlyStopping, ModelCheckpoin
t
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
```

Build a CNN classifier model using the Sequential API

In [0]:
```
def getCNNModel(inputshape, decayRate, dropRate):
    model = Sequential([
        Conv2D(16, kernel_size=3, padding='SAME', activation='relu',
               kernel_initializer='he_uniform', kernel_regularizer=l2(decayRat
e), bias_initializer='ones',
               name='conv2d_1', input_shape=(inputshape)),

        Dropout(dropRate, name='dropout_1'),

        Conv2D(16, kernel_size=3, padding='SAME', activation='relu',
               kernel_regularizer=l2(decayRate), name='conv2d_2'),

        Dropout(dropRate, name='dropout_2'),
        BatchNormalization(name='batch_norm_1'),
        MaxPooling2D(pool_size=(4,4), name='max_pool_1'),

        Dense(128, activation='relu', name='dense_1'),
        Flatten(name='flatten_1'),
        Dense(10, activation='softmax', name='dense_2')
    ])

    model.compile(
        optimizer=Adam(learning_rate=0.0001),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return model
```

In [50]:
```
model = getCNNModel(x_train[0].shape, 0.001, 0.3)
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 16)        160       
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 16)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 16)        2320      
_________________________________________________________________
dropout_2 (Dropout)          (None, 32, 32, 16)        0         
_________________________________________________________________
batch_norm_1 (BatchNormaliza (None, 32, 32, 16)        64        
_________________________________________________________________
max_pool_1 (MaxPooling2D)    (None, 8, 8, 16)          0         
_________________________________________________________________
dense_1 (Dense)              (None, 8, 8, 128)         2176      
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                81930     
=================================================================
Total params: 86,650
Trainable params: 86,618
Non-trainable params: 32
_________________________________________________________________
```

In [51]:
```
print(model.optimizer)
print(model.loss)
print(model.metrics)
print(model.optimizer.lr)
```

```
<tensorflow.python.keras.optimizer_v2.adam.Adam object at 0x7f7141554470>
categorical_crossentropy
[]
<tf.Variable 'learning_rate:0' shape=() dtype=float32, numpy=1e-04>
```

In [0]:
```
def get_early_stopping():
    return EarlyStopping(patience=4, monitor='val_accuracy')

def get_best_CNN_checkpoint():
    path = 'model_checkpoint_CNN/checkpoint'
    return ModelCheckpoint(filepath=path, save_freq='epoch',
                                 save_weights_only=True, save_best_only=True,
                                 monitor='val_accuracy', verbose=1)
```

In [53]:

```
history = model.fit(x_train, y_train, batch_size=256, epochs=30,
                   validation_data=(x_val,y_val),
                   callbacks=[get_early_stopping(), get_best_CNN_checkpoint
()])
```

```
Epoch 1/30
249/249 [==============================] - ETA: 0s - loss: 2.2910 - accuracy:
0.1977
Epoch 00001: val_accuracy improved from -inf to 0.26302, saving model to mode
l_checkpoint_CNN/checkpoint
249/249 [==============================] - 112s 449ms/step - loss: 2.2910 - a
ccuracy: 0.1977 - val_loss: 2.1746 - val_accuracy: 0.2630
Epoch 2/30
249/249 [==============================] - ETA: 0s - loss: 1.8744 - accuracy:
0.3871
Epoch 00002: val_accuracy improved from 0.26302 to 0.45842, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 112s 450ms/step - loss: 1.8744 - a
ccuracy: 0.3871 - val_loss: 1.8262 - val_accuracy: 0.4584
Epoch 3/30
249/249 [==============================] - ETA: 0s - loss: 1.3584 - accuracy:
0.5789
Epoch 00003: val_accuracy improved from 0.45842 to 0.63167, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 113s 452ms/step - loss: 1.3584 - a
ccuracy: 0.5789 - val_loss: 1.3852 - val_accuracy: 0.6317
Epoch 4/30
249/249 [==============================] - ETA: 0s - loss: 1.0989 - accuracy:
0.6722
Epoch 00004: val_accuracy improved from 0.63167 to 0.69677, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 114s 457ms/step - loss: 1.0989 - a
ccuracy: 0.6722 - val_loss: 1.1588 - val_accuracy: 0.6968
Epoch 5/30
249/249 [==============================] - ETA: 0s - loss: 0.9669 - accuracy:
0.7176
Epoch 00005: val_accuracy improved from 0.69677 to 0.74129, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 112s 452ms/step - loss: 0.9669 - a
ccuracy: 0.7176 - val_loss: 1.0100 - val_accuracy: 0.7413
Epoch 6/30
249/249 [==============================] - ETA: 0s - loss: 0.8811 - accuracy:
0.7481
Epoch 00006: val_accuracy improved from 0.74129 to 0.76480, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 112s 452ms/step - loss: 0.8811 - a
ccuracy: 0.7481 - val_loss: 0.9242 - val_accuracy: 0.7648
Epoch 7/30
249/249 [==============================] - ETA: 0s - loss: 0.8221 - accuracy:
0.7680
Epoch 00007: val_accuracy improved from 0.76480 to 0.78433, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 113s 452ms/step - loss: 0.8221 - a
ccuracy: 0.7680 - val_loss: 0.8601 - val_accuracy: 0.7843
Epoch 8/30
249/249 [==============================] - ETA: 0s - loss: 0.7811 - accuracy:
0.7825
Epoch 00008: val_accuracy improved from 0.78433 to 0.79378, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 113s 454ms/step - loss: 0.7811 - a
ccuracy: 0.7825 - val_loss: 0.8139 - val_accuracy: 0.7938
Epoch 9/30
```

```
249/249 [==============================] - ETA: 0s - loss: 0.7469 - accuracy:
0.7941
Epoch 00009: val_accuracy improved from 0.79378 to 0.80176, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 115s 460ms/step - loss: 0.7469 - a
ccuracy: 0.7941 - val_loss: 0.7867 - val_accuracy: 0.8018
Epoch 10/30
249/249 [==============================] - ETA: 0s - loss: 0.7187 - accuracy:
0.8010
Epoch 00010: val_accuracy improved from 0.80176 to 0.81279, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 113s 454ms/step - loss: 0.7187 - a
ccuracy: 0.8010 - val_loss: 0.7476 - val_accuracy: 0.8128
Epoch 11/30
249/249 [==============================] - ETA: 0s - loss: 0.6962 - accuracy:
0.8096
Epoch 00011: val_accuracy improved from 0.81279 to 0.81909, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 114s 456ms/step - loss: 0.6962 - a
ccuracy: 0.8096 - val_loss: 0.7192 - val_accuracy: 0.8191
Epoch 12/30
249/249 [==============================] - ETA: 0s - loss: 0.6773 - accuracy:
0.8168
Epoch 00012: val_accuracy improved from 0.81909 to 0.82192, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 113s 454ms/step - loss: 0.6773 - a
ccuracy: 0.8168 - val_loss: 0.7043 - val_accuracy: 0.8219
Epoch 13/30
249/249 [==============================] - ETA: 0s - loss: 0.6576 - accuracy:
0.8223
Epoch 00013: val_accuracy improved from 0.82192 to 0.82728, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 113s 454ms/step - loss: 0.6576 - a
ccuracy: 0.8223 - val_loss: 0.6822 - val_accuracy: 0.8273
Epoch 14/30
249/249 [==============================] - ETA: 0s - loss: 0.6424 - accuracy:
0.8263
Epoch 00014: val_accuracy improved from 0.82728 to 0.83547, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 115s 461ms/step - loss: 0.6424 - a
ccuracy: 0.8263 - val_loss: 0.6681 - val_accuracy: 0.8355
Epoch 15/30
249/249 [==============================] - ETA: 0s - loss: 0.6273 - accuracy:
0.8321
Epoch 00015: val_accuracy did not improve from 0.83547
249/249 [==============================] - 112s 451ms/step - loss: 0.6273 - a
ccuracy: 0.8321 - val_loss: 0.6583 - val_accuracy: 0.8323
Epoch 16/30
249/249 [==============================] - ETA: 0s - loss: 0.6183 - accuracy:
0.8339
Epoch 00016: val_accuracy improved from 0.83547 to 0.83956, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 113s 453ms/step - loss: 0.6183 - a
ccuracy: 0.8339 - val_loss: 0.6485 - val_accuracy: 0.8396
Epoch 17/30
249/249 [==============================] - ETA: 0s - loss: 0.6049 - accuracy:
0.8368
```

```
Epoch 00017: val_accuracy improved from 0.83956 to 0.84240, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 113s 454ms/step - loss: 0.6049 - a
ccuracy: 0.8368 - val_loss: 0.6307 - val_accuracy: 0.8424
Epoch 18/30
249/249 [==============================] - ETA: 0s - loss: 0.5924 - accuracy:
0.8415
Epoch 00018: val_accuracy improved from 0.84240 to 0.84912, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 113s 455ms/step - loss: 0.5924 - a
ccuracy: 0.8415 - val_loss: 0.6186 - val_accuracy: 0.8491
Epoch 19/30
249/249 [==============================] - ETA: 0s - loss: 0.5837 - accuracy:
0.8439
Epoch 00019: val_accuracy improved from 0.84912 to 0.84985, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 114s 456ms/step - loss: 0.5837 - a
ccuracy: 0.8439 - val_loss: 0.6141 - val_accuracy: 0.8499
Epoch 20/30
249/249 [==============================] - ETA: 0s - loss: 0.5765 - accuracy:
0.8480
Epoch 00020: val_accuracy did not improve from 0.84985
249/249 [==============================] - 115s 461ms/step - loss: 0.5765 - a
ccuracy: 0.8480 - val_loss: 0.6097 - val_accuracy: 0.8483
Epoch 21/30
249/249 [==============================] - ETA: 0s - loss: 0.5653 - accuracy:
0.8487
Epoch 00021: val_accuracy did not improve from 0.84985
249/249 [==============================] - 113s 454ms/step - loss: 0.5653 - a
ccuracy: 0.8487 - val_loss: 0.6037 - val_accuracy: 0.8488
Epoch 22/30
249/249 [==============================] - ETA: 0s - loss: 0.5567 - accuracy:
0.8508
Epoch 00022: val_accuracy improved from 0.84985 to 0.85300, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 113s 455ms/step - loss: 0.5567 - a
ccuracy: 0.8508 - val_loss: 0.5971 - val_accuracy: 0.8530
Epoch 23/30
249/249 [==============================] - ETA: 0s - loss: 0.5499 - accuracy:
0.8542
Epoch 00023: val_accuracy improved from 0.85300 to 0.85720, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 114s 456ms/step - loss: 0.5499 - a
ccuracy: 0.8542 - val_loss: 0.5844 - val_accuracy: 0.8572
Epoch 24/30
249/249 [==============================] - ETA: 0s - loss: 0.5422 - accuracy:
0.8559
Epoch 00024: val_accuracy did not improve from 0.85720
249/249 [==============================] - 113s 453ms/step - loss: 0.5422 - a
ccuracy: 0.8559 - val_loss: 0.5850 - val_accuracy: 0.8546
Epoch 25/30
249/249 [==============================] - ETA: 0s - loss: 0.5338 - accuracy:
0.8572
Epoch 00025: val_accuracy improved from 0.85720 to 0.85930, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 116s 464ms/step - loss: 0.5338 - a
ccuracy: 0.8572 - val_loss: 0.5718 - val_accuracy: 0.8593
```

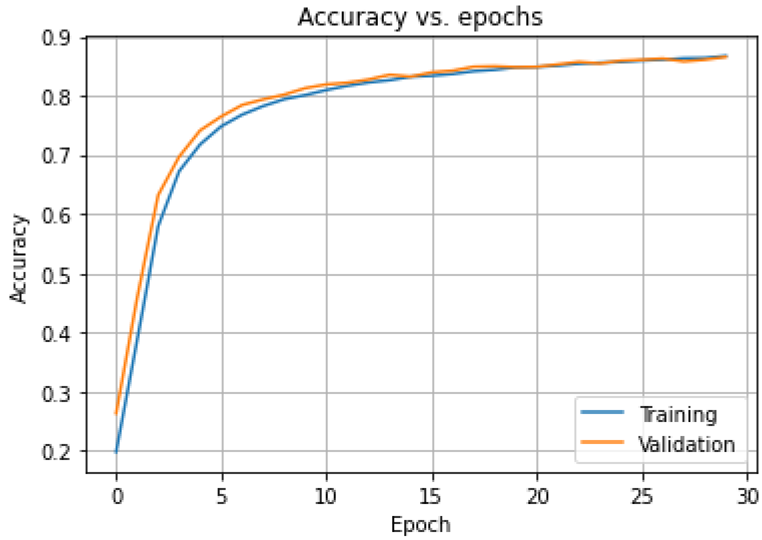
```
Epoch 26/30
249/249 [==============================] - ETA: 0s - loss: 0.5280 - accuracy:
0.8595
Epoch 00026: val_accuracy improved from 0.85930 to 0.86077, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 113s 454ms/step - loss: 0.5280 - a
ccuracy: 0.8595 - val_loss: 0.5661 - val_accuracy: 0.8608
Epoch 27/30
249/249 [==============================] - ETA: 0s - loss: 0.5240 - accuracy:
0.8612
Epoch 00027: val_accuracy improved from 0.86077 to 0.86298, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 113s 455ms/step - loss: 0.5240 - a
ccuracy: 0.8612 - val_loss: 0.5633 - val_accuracy: 0.8630
Epoch 28/30
249/249 [==============================] - ETA: 0s - loss: 0.5154 - accuracy:
0.8634
Epoch 00028: val_accuracy did not improve from 0.86298
249/249 [==============================] - 113s 454ms/step - loss: 0.5154 - a
ccuracy: 0.8634 - val_loss: 0.5630 - val_accuracy: 0.8580
Epoch 29/30
249/249 [==============================] - ETA: 0s - loss: 0.5121 - accuracy:
0.8640
Epoch 00029: val_accuracy did not improve from 0.86298
249/249 [==============================] - 113s 452ms/step - loss: 0.5121 - a
ccuracy: 0.8640 - val_loss: 0.5545 - val_accuracy: 0.8610
Epoch 30/30
249/249 [==============================] - ETA: 0s - loss: 0.5024 - accuracy:
0.8673
Epoch 00030: val_accuracy improved from 0.86298 to 0.86571, saving model to m
odel_checkpoint_CNN/checkpoint
249/249 [==============================] - 115s 461ms/step - loss: 0.5024 - a
ccuracy: 0.8673 - val_loss: 0.5488 - val_accuracy: 0.8657
```

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

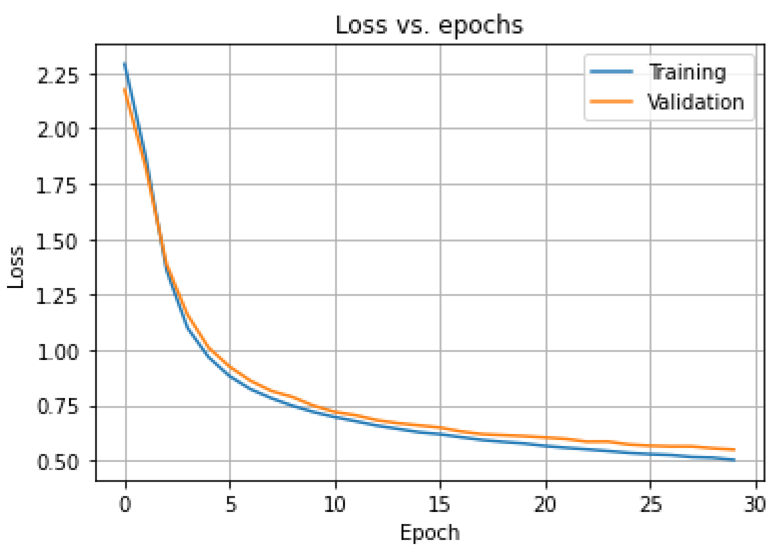
In [54]:
```
print(history.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [55]:
```
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'])
plt.grid()
```



In [56]:
```
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'])
plt.grid()
```



In [57]:
```
import pandas as pd
df = pd.DataFrame(history.history)
df.head()
```

Out[57]:

	loss	accuracy	val_loss	val_accuracy
0	2.291016	0.197715	2.174621	0.263020
1	1.874427	0.387068	1.826240	0.458421
2	1.358393	0.578947	1.385186	0.631667
3	1.098860	0.672164	1.158793	0.696766
4	0.966949	0.717572	1.009975	0.741285

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

In [58]:
```
loss, accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f'Loss : {loss:.3f} \nAccuracy :{accuracy:.3f}')
```

```
814/814 - 11s - loss: 0.6041 - accuracy: 0.8510
Loss : 0.604
Accuracy :0.851
```

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 the best weights for the MLP and CNN models

In [61]:
```
model_mlp = getModel(x_train[0].shape)
model_mlp.load_weights(filepath='model_checkpoint/checkpoint')
```

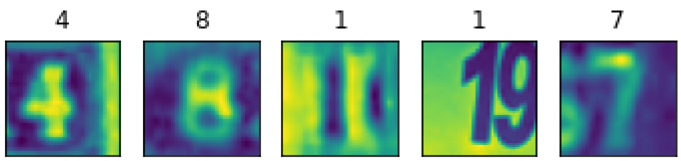
Out[61]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f7156605
7f0>
```

In [63]:
```
model_cnn = getCNNModel(x_train[0].shape, 0.001, 0.3)
model_cnn.load_weights(filepath='model_checkpoint_CNN/checkpoint')
```

Out[63]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f7156edc
1d0>
```

Randomly select 5 images and corresponding labels from the test set and display the images with their labels.

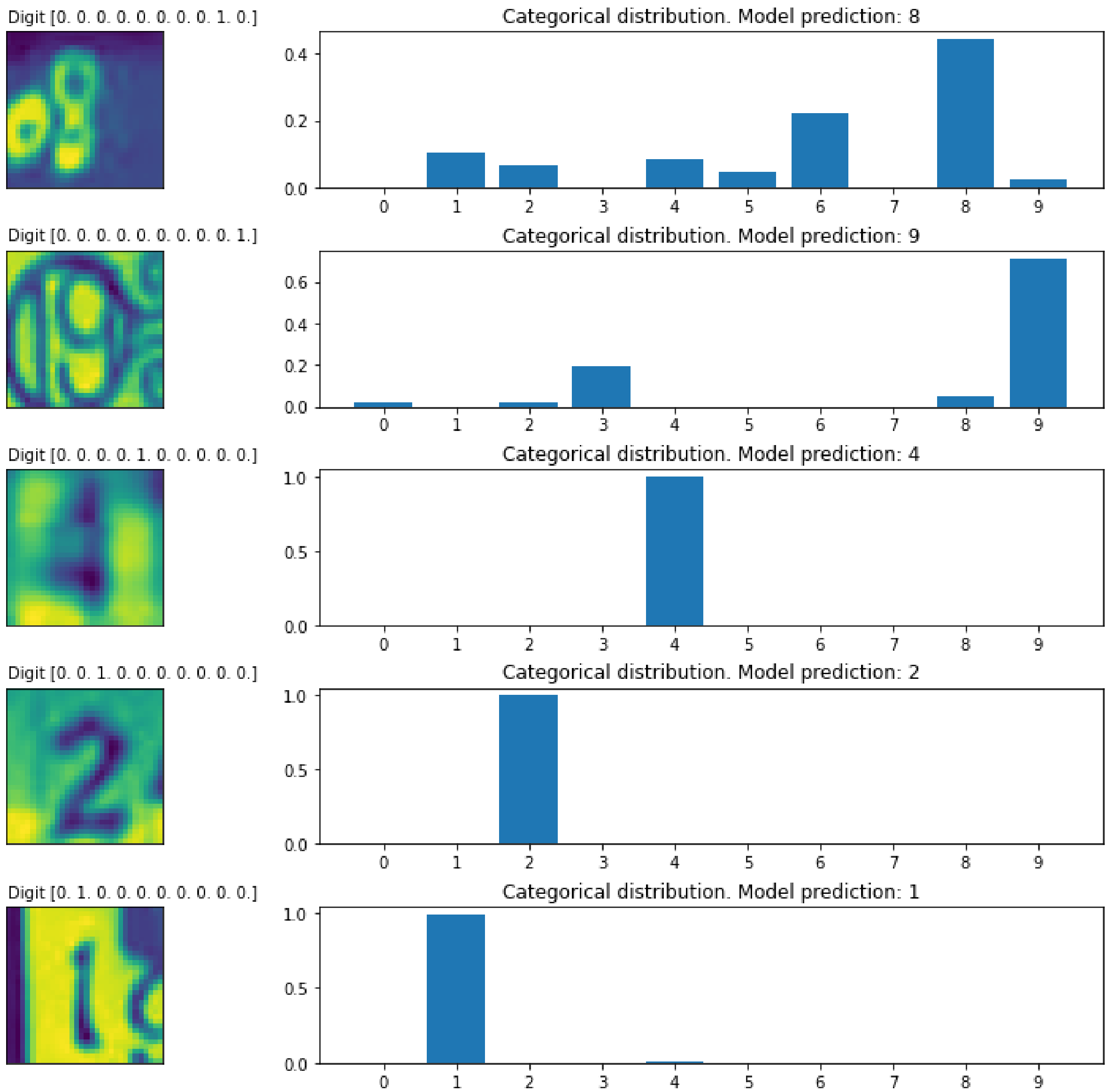
In [64]:
```
plot_images(x_test, y_test, 1, 5)
```



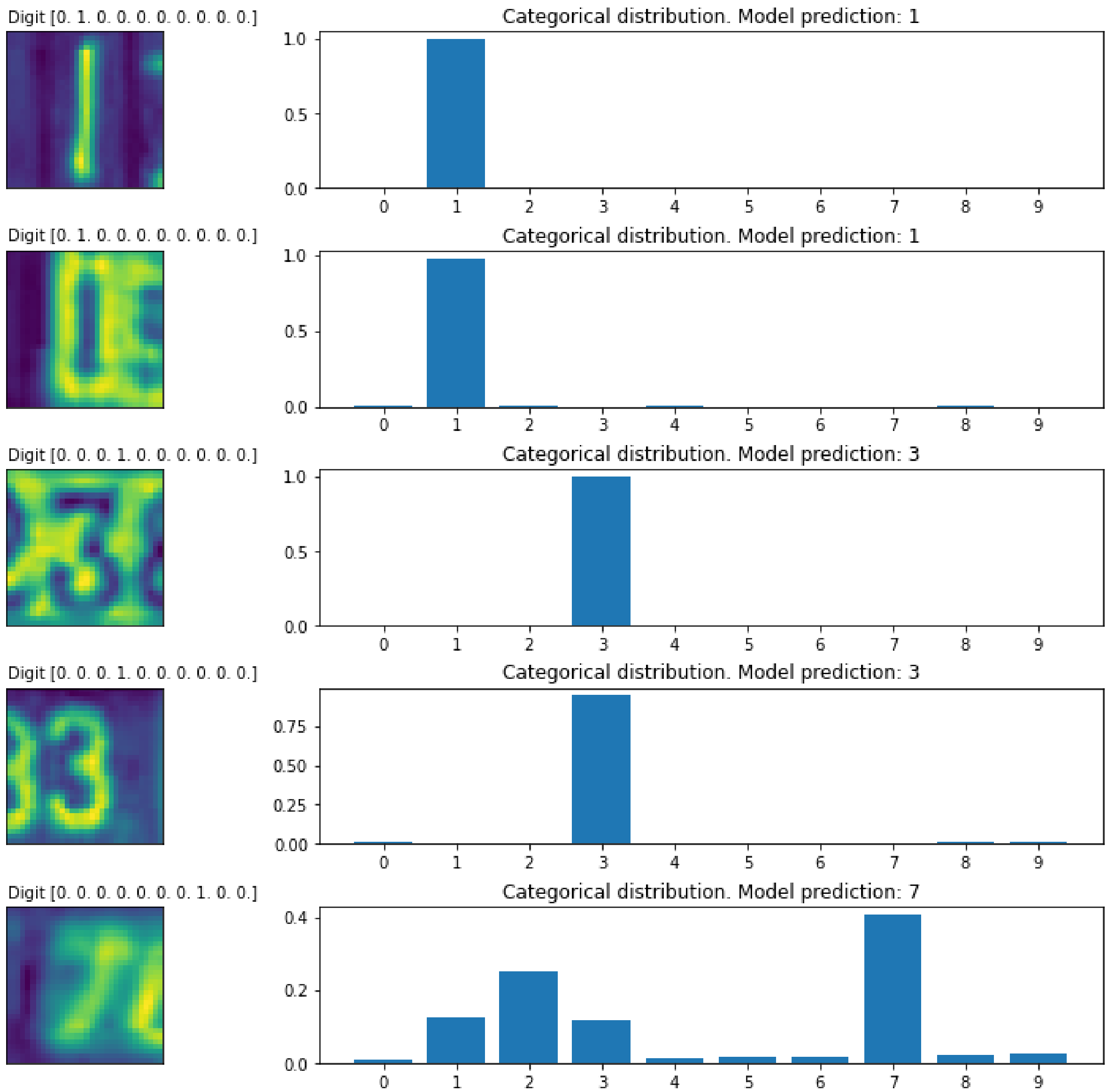
In [0]:
```
def plot_prediction_bars(predictions, images, labels):
    fig, axes = plt.subplots(5, 2, figsize=(16, 12))
    fig.subplots_adjust(hspace=0.4, wspace=-0.2)

    for i, (prediction, image, label) in enumerate(zip(predictions, images, la
bels)):
        axes[i, 0].imshow(np.squeeze(image))
        axes[i, 0].get_xaxis().set_visible(False)
        axes[i, 0].get_yaxis().set_visible(False)
        axes[i, 0].text(0., -2.5, f'Digit {label}')
        axes[i, 1].bar(np.arange(len(prediction)), prediction)
        axes[i, 1].set_xticks(np.arange(len(prediction)))
        axes[i, 1].set_title(f"Categorical distribution. Model prediction: {n
p.argmax(prediction)}")
```

In [86]:

```
num_test_images = x_test.shape[0]     # Get the total no. of images
indexes = np.random.choice(num_test_images, 5) # Choose 5 random indexes
images = x_test[indexes, ...]  # Get the random 5 images
labels = y_test[indexes, ...]  # Get correspodig labels

predictions = model_mlp.predict(images)
plot_prediction_bars(predictions, images, labels)
```



In [87]:
```
num_test_images = x_test.shape[0]     # Get the total no. of images
indexes = np.random.choice(num_test_images, 5) # Choose 5 random indexes
images = x_test[indexes, ...]  # Get the random 5 images
labels = y_test[indexes, ...]  # Get correspodig labels

predictions = model_cnn.predict(images)
plot_prediction_bars(predictions, images, labels)
```



In [0]:
```
# THANK YOU
```