# 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 (you could download the notebook with File -> Download .ipynb, open the notebook locally, and then File -> Download as -> PDF via LaTeX), and 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.

```
import tensorflow as tf
from scipy.io import loadmat
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
from sklearn.preprocessing import OneHotEncoder
```

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.

The train and test datasets required for this project can be downloaded from here and here. Once unzipped, you will have two files: `train_32x32.mat` and `test_32x32.mat`. You should store these files in Drive for use in this Colab notebook.

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.

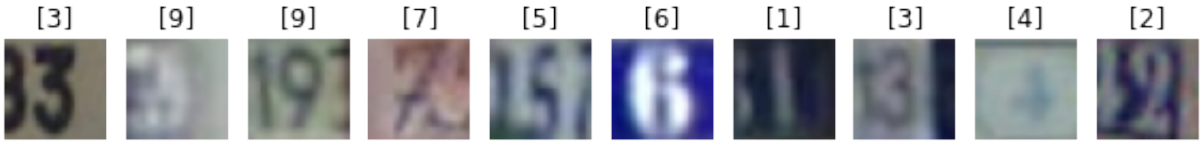
```
# Run this cell to connect to your Drive folder

#from google.colab import drive
#drive.mount('/content/gdrive')


# Load the dataset from your Drive folder

#train = loadmat('path/to/train_32x32.mat')
#test = loadmat('path/to/test_32x32.mat')


# Load the dataset from your folder - I used local instead of google colab

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

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

# 1. Inspect and preprocess the dataset

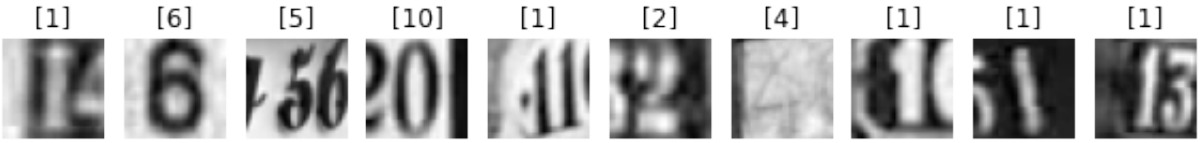
- Extract the training and testing images and labels separately from the train and test dictionaries loaded for you.
- Select a random sample of images and corresponding labels from the dataset (at least 10), and display them in a figure.
- Convert the training and test images to grayscale by taking 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.

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

#rescale images
train_images = train_images/255.
test_images = test_images/255.


# get 10 samples
sample_indices = random.sample(range(len(train_images[1][1][1])), 10)
sample_images = train_images[:,:,:,sample_indices]
sample_labels = train_labels[sample_indices]


fig, ax = plt.subplots(1, 10, figsize=(10, 1))
for i in range(10):
    ax[i].set_axis_off()
    ax[i].imshow(sample_images[:,:,:,i])
    ax[i].set_title(str(sample_labels[i]))
```




```
# grayscale by taking the average across all colour channels
train_images = np.average(train_images,axis=2)
test_images = np.average(test_images,axis=2)
```

```
# get 10 greyscale samples
sample_indices = random.sample(range(len(train_images[1][1])), 10)
sample_images_grey = train_images[:,:,sample_indices]
sample_labels_grey = train_labels[sample_indices]


fig, ax = plt.subplots(1, 10, figsize=(10, 1))
for i in range(10):
    ax[i].set_axis_off()
    ax[i].imshow(sample_images_grey[:,:,i],cmap=plt.get_cmap('gray'))
    ax[i].set_title(str(sample_labels_grey[i]))
```




```
#transpose images for model
test_images=np.transpose(test_images, (2,0,1))[...,np.newaxis]
train_images=np.transpose(train_images, (2,0,1))[...,np.newaxis]


encoder = OneHotEncoder(sparse=False)

train_labels = encoder.fit_transform(train_labels % 10)
test_labels = encoder.fit_transform(test_labels % 10)
```

# 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.

```
from tensorflow.keras.layers import Dropout,Dense,Flatten
from tensorflow.keras import regularizers
from tensorflow.keras.models import Sequential


def get_new_model(input_shape,rate,wd):

    model = Sequential([
        Flatten(input_shape=input_shape),
        Dense(128,activation = 'relu',kernel_regularizer=regularizers.l2(wd),nam
        Dense(128,activation = 'relu',kernel_regularizer=regularizers.l2(wd),nam
        Dense(128,activation = 'relu',kernel_regularizer=regularizers.l2(wd),nam
        Dense(128,activation = 'relu',kernel_regularizer=regularizers.l2(wd),nam
        Dense(64,activation = 'relu',kernel_regularizer=regularizers.l2(wd),name
        Dense(10,activation = 'softmax',name='layer_output')
    ])
    return model
```

```
model_MLP = get_new_model(train_images[1].shape,0.2,1e-5)
model_MLP.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten (Flatten)            (None, 1024)              0         
_________________________________________________________________
layer_1 (Dense)              (None, 128)               131200    
_________________________________________________________________
layer_2 (Dense)              (None, 128)               16512     
_________________________________________________________________
layer_3 (Dense)              (None, 128)               16512     
_________________________________________________________________
layer_4 (Dense)              (None, 128)               16512     
_________________________________________________________________
layer_5 (Dense)              (None, 64)                8256      
_________________________________________________________________
layer_output (Dense)         (None, 10)                650       
=================================================================
Total params: 189,642
Trainable params: 189,642
Non-trainable params: 0
_________________________________________________________________
2022-08-10 17:15:27.162631: I tensorflow/core/platform/cpu_feature_guard.cc
To enable them in non-MKL-DNN operations, rebuild TensorFlow with the appro
2022-08-10 17:15:27.167541: I tensorflow/core/common_runtime/process_util.c
```

```
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau
```

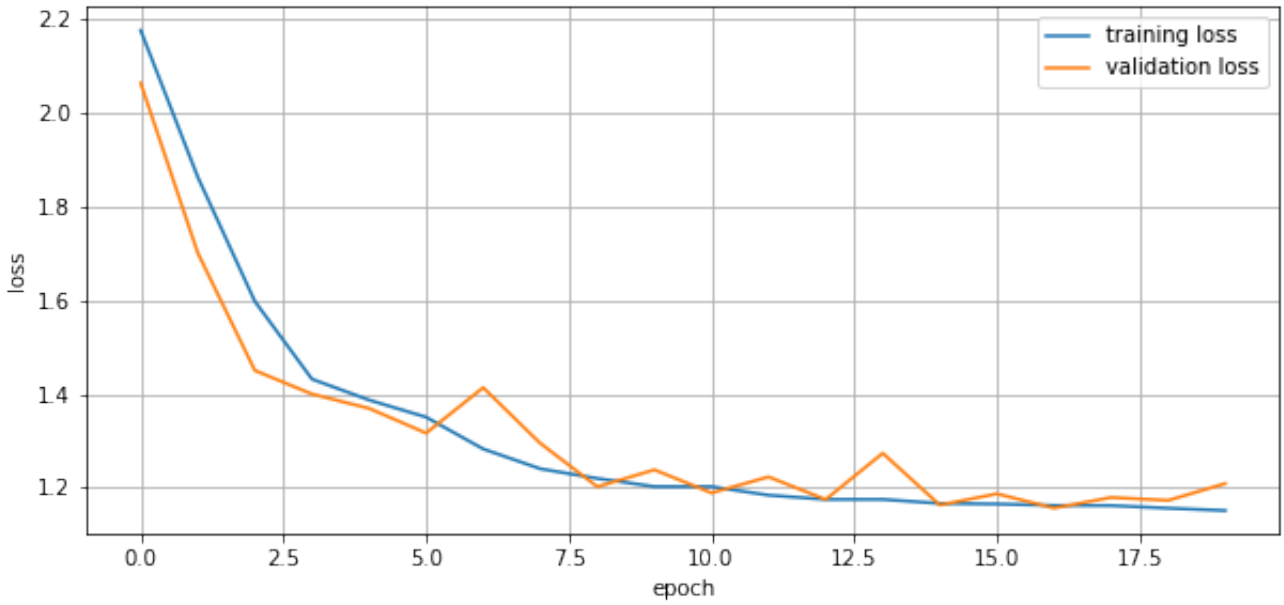
```
class CustomSaver(tf.keras.callbacks.Callback): # this is a try to save specific

    # init method or constructor
    def __init__(self, typ):
        self.typ = typ

    def on_epoch_end(self, epoch, logs={}):
        checkpoint_path = "model_"+self.typ+"_checkpoints/epoch:{}"
        if epoch % 5 == 0:  # each k-th epoch
            self.model.save(checkpoint_path.format(epoch))

def createCallbacks(typ):

    earlyStopping = EarlyStopping(monitor = 'val_loss', patience=10,verbose=1)

    saverCallback = CustomSaver(typ)

    checkpoint_best_path = 'model_'+typ+'_checkpoints_best/checkpoint'
    checkpoint_best = ModelCheckpoint(filepath = checkpoint_best_path,save_weigh
                                      save_best_only=True,verbose = 1)

    plateau = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=10, min

    return earlyStopping,saverCallback,checkpoint_best,plateau



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


earlyStopping,saverCallback,checkpoint_best_path,plateau = createCallbacks('MLP'
history_MLP= model_MLP.fit(x=train_images,
                       y=train_labels,
                       epochs = 20,
                       batch_size = 128,
                       callbacks= [earlyStopping, saverCallback, checkpoint_best
                       validation_split=0.15,
                       verbose=2)
```

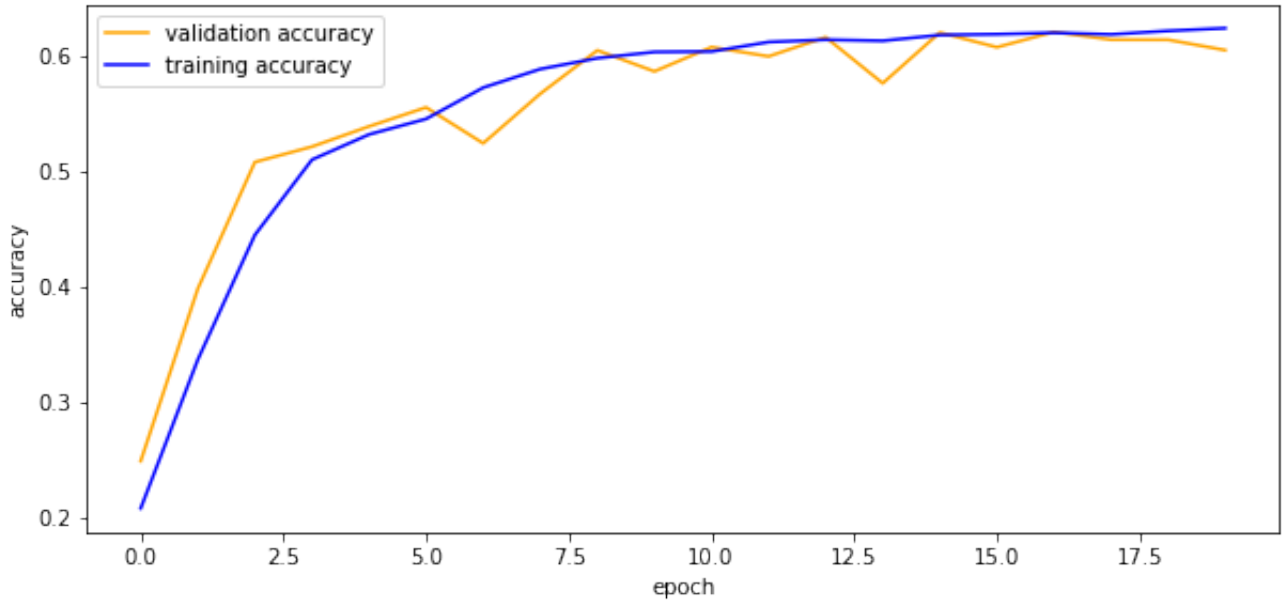
```
Train on 62268 samples, validate on 10989 samples
Epoch 1/20
2022-08-10 17:16:06.019370: W tensorflow/python/util/util.cc:299] Sets ar
WARNING:tensorflow:From /Users/metehangelgi/opt/anaconda3/envs/summerProj
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

```
INFO:tensorflow:Assets written to: model_MLP_checkpoints/epoch:0/assets

Epoch 00001: val_accuracy improved from -inf to 0.24970, saving model to
62268/62268 - 14s - loss: 2.1772 - accuracy: 0.2088 - val_loss: 2.0647 -
Epoch 2/20

Epoch 00002: val_accuracy improved from 0.24970 to 0.39904, saving model
62268/62268 - 8s - loss: 1.8639 - accuracy: 0.3376 - val_loss: 1.7019 - v
Epoch 3/20

Epoch 00003: val_accuracy improved from 0.39904 to 0.50878, saving model
62268/62268 - 9s - loss: 1.5984 - accuracy: 0.4457 - val_loss: 1.4507 - v
Epoch 4/20

Epoch 00004: val_accuracy improved from 0.50878 to 0.52216, saving model
62268/62268 - 10s - loss: 1.4324 - accuracy: 0.5109 - val_loss: 1.4005 -
Epoch 5/20

Epoch 00005: val_accuracy improved from 0.52216 to 0.53972, saving model
62268/62268 - 9s - loss: 1.3877 - accuracy: 0.5327 - val_loss: 1.3699 - v
Epoch 6/20
INFO:tensorflow:Assets written to: model_MLP_checkpoints/epoch:5/assets

Epoch 00006: val_accuracy improved from 0.53972 to 0.55619, saving model
62268/62268 - 12s - loss: 1.3512 - accuracy: 0.5462 - val_loss: 1.3170 -
Epoch 7/20

Epoch 00007: val_accuracy did not improve from 0.55619
62268/62268 - 11s - loss: 1.2832 - accuracy: 0.5733 - val_loss: 1.4142 -
Epoch 8/20

Epoch 00008: val_accuracy improved from 0.55619 to 0.56802, saving model
62268/62268 - 8s - loss: 1.2409 - accuracy: 0.5894 - val_loss: 1.2953 - v
Epoch 9/20

Epoch 00009: val_accuracy improved from 0.56802 to 0.60561, saving model
62268/62268 - 8s - loss: 1.2202 - accuracy: 0.5988 - val_loss: 1.2025 - v
Epoch 10/20

Epoch 00010: val_accuracy did not improve from 0.60561
62268/62268 - 8s - loss: 1.2028 - accuracy: 0.6042 - val_loss: 1.2386 - v
Epoch 11/20
INFO:tensorflow:Assets written to: model_MLP_checkpoints/epoch:10/assets

Epoch 00011: val_accuracy improved from 0.60561 to 0.60843, saving model
62268/62268 - 10s - loss: 1.2026 - accuracy: 0.6046 - val_loss: 1.1893 -
Epoch 12/20

Epoch 00012: val_accuracy did not improve from 0.60843
62268/62268 - 8s - loss: 1.1844 - accuracy: 0.6129 - val_loss: 1.2235 - v
Epoch 13/20

Epoch 00013: val_accuracy improved from 0.60843 to 0.61698, saving model
```

```
def evaluate_model(model):
    test_loss, test_accuracy = model.evaluate(test_images, test_labels, verbose
    print(f"Test accuracy is {test_accuracy}")
    print(f"Test loss is {test_loss}")


def plotting(history):

    fig, axs = plt.subplots(2,1, figsize=(10,10))
    axs[0].plot(history['loss'], label = 'training loss')
    axs[0].plot(history['val_loss'], label = 'validation loss')
    axs[0].set_ylabel('loss')
    axs[0].set_xlabel('epoch')
    axs[0].grid(True)

    axs[1].plot(history['val_accuracy'], label = 'validation accuracy',color = '
    axs[1].plot(history['accuracy'], label = 'training accuracy',color = 'blue')
    axs[1].set_ylabel('accuracy')
    axs[1].set_xlabel('epoch')
    axs[0].grid(True)

    axs[1].legend()
    axs[0].legend()

    plt.show()



evaluate_model(model_MLP)

     Test accuracy is 0.5831284523010254
     Test loss is 1.309795838378292
```

```
plotting(history_MLP.history)
```



# 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.

```
from tensorflow.keras.layers import Flatten, MaxPooling2D, BatchNormalization, D

def get_new_2_model(input_shape,rate,wd):
    model = Sequential([
        Conv2D(filters=16, input_shape=input_shape, kernel_size=(3, 3),
               activation='relu', name='conv_1'),
        Conv2D(filters=8, kernel_size=(3, 3), activation='relu', name='conv_2'),
        MaxPooling2D(pool_size=(4, 4), name='pool_1'),
        Flatten(name='flatten'),
        Dense(units=32, activation='relu', name='dense_1'),
        Dropout(rate),
        Dense(units=32, activation='relu', name='dense_2'),
        BatchNormalization(),
        Dense(units=10, activation='softmax', name='output')
    ])

    return model
```

```
model = get_new_2_model(train_images[1].shape,0.2,1e-5)
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv_1 (Conv2D)              (None, 30, 30, 16)        160       
_________________________________________________________________
conv_2 (Conv2D)              (None, 28, 28, 8)         1160      
_________________________________________________________________
pool_1 (MaxPooling2D)        (None, 7, 7, 8)           0         
_________________________________________________________________
flatten (Flatten)            (None, 392)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 32)                12576     
_________________________________________________________________
dropout (Dropout)            (None, 32)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056      
_________________________________________________________________
batch_normalization (BatchNo (None, 32)                128       
_________________________________________________________________
output (Dense)               (None, 10)                330       
=================================================================
Total params: 15,410
Trainable params: 15,346
Non-trainable params: 64
_________________________________________________________________
```

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

```
earlyStopping,saverCallback,checkpoint_best_path,plateau = createCallbacks('CNN'
history_CNN= model.fit(x=train_images,
                       y=train_labels,
                       epochs = 20,
                       batch_size = 128,
                       callbacks= [earlyStopping, saverCallback, checkpoint_best
                       validation_split=0.15,
                       verbose=2)
```

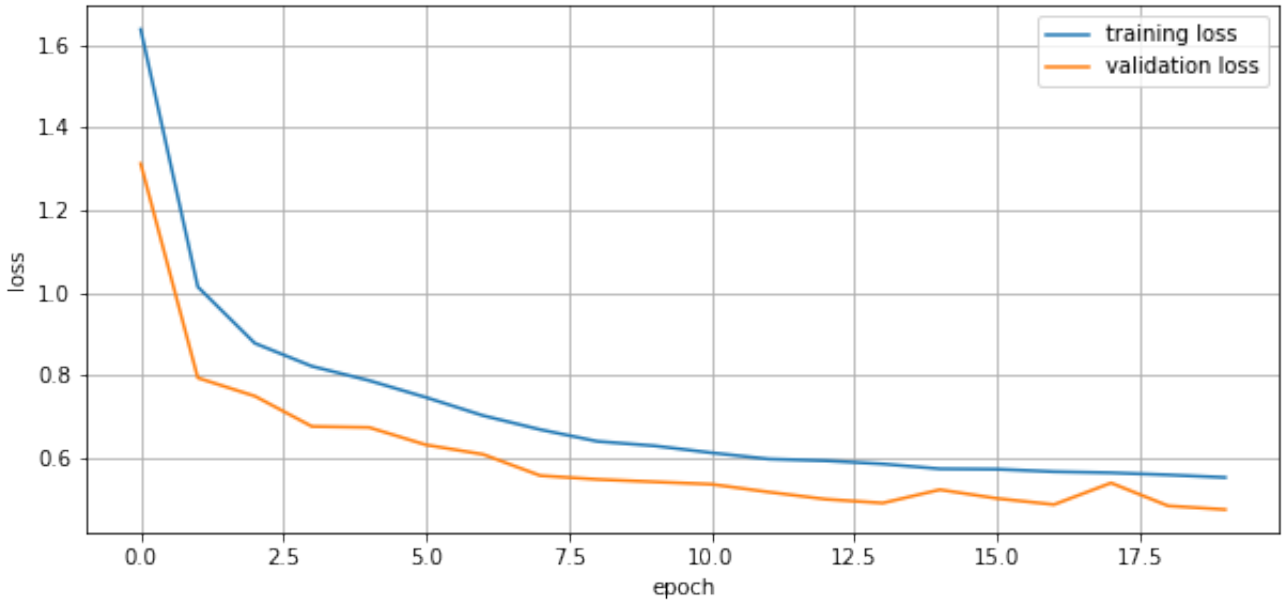
```
Train on 62268 samples, validate on 10989 samples
Epoch 1/20
INFO:tensorflow:Assets written to: model_CNN_checkpoints/epoch:0/assets

Epoch 00001: val_accuracy improved from -inf to 0.61607, saving model to
```

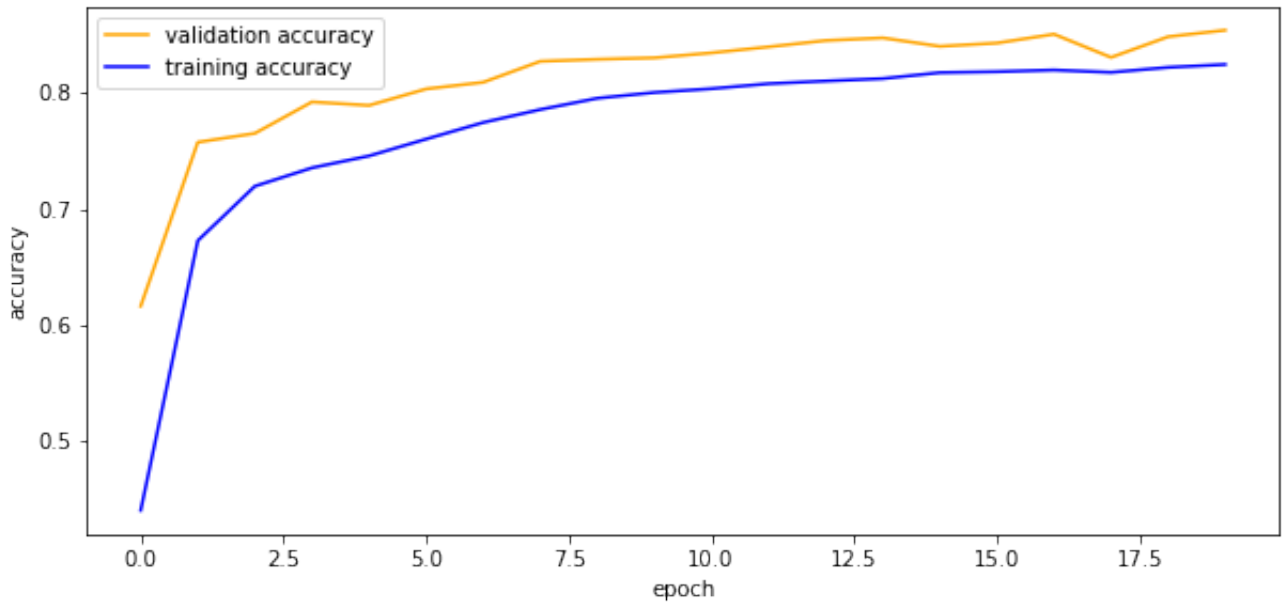
```
Epoch 00001: val_accuracy improved from -inf to 0.61607, saving model to
62268/62268 - 36s - loss: 1.6368 - accuracy: 0.4399 - val_loss: 1.3120 -
Epoch 2/20

Epoch 00002: val_accuracy improved from 0.61607 to 0.75748, saving model
62268/62268 - 29s - loss: 1.0140 - accuracy: 0.6725 - val_loss: 0.7938 -
Epoch 3/20

Epoch 00003: val_accuracy improved from 0.75748 to 0.76513, saving model
62268/62268 - 29s - loss: 0.8776 - accuracy: 0.7195 - val_loss: 0.7497 -
Epoch 4/20

Epoch 00004: val_accuracy improved from 0.76513 to 0.79225, saving model
62268/62268 - 29s - loss: 0.8219 - accuracy: 0.7354 - val_loss: 0.6764 -
Epoch 5/20

Epoch 00005: val_accuracy did not improve from 0.79225
62268/62268 - 30s - loss: 0.7875 - accuracy: 0.7457 - val_loss: 0.6740 -
Epoch 6/20
INFO:tensorflow:Assets written to: model_CNN_checkpoints/epoch:5/assets

Epoch 00006: val_accuracy improved from 0.79225 to 0.80344, saving model
62268/62268 - 33s - loss: 0.7465 - accuracy: 0.7602 - val_loss: 0.6318 -
Epoch 7/20

Epoch 00007: val_accuracy improved from 0.80344 to 0.80917, saving model
62268/62268 - 29s - loss: 0.7026 - accuracy: 0.7745 - val_loss: 0.6088 -
Epoch 8/20

Epoch 00008: val_accuracy improved from 0.80917 to 0.82728, saving model
62268/62268 - 29s - loss: 0.6688 - accuracy: 0.7857 - val_loss: 0.5574 -
Epoch 9/20

Epoch 00009: val_accuracy improved from 0.82728 to 0.82901, saving model
62268/62268 - 29s - loss: 0.6403 - accuracy: 0.7954 - val_loss: 0.5483 -
Epoch 10/20

Epoch 00010: val_accuracy improved from 0.82901 to 0.83028, saving model
62268/62268 - 36s - loss: 0.6295 - accuracy: 0.8004 - val_loss: 0.5423 -
Epoch 11/20
INFO:tensorflow:Assets written to: model_CNN_checkpoints/epoch:10/assets

Epoch 00011: val_accuracy improved from 0.83028 to 0.83456, saving model
62268/62268 - 36s - loss: 0.6126 - accuracy: 0.8037 - val_loss: 0.5365 -
Epoch 12/20

Epoch 00012: val_accuracy improved from 0.83456 to 0.83975, saving model
62268/62268 - 30s - loss: 0.5976 - accuracy: 0.8080 - val_loss: 0.5177 -
Epoch 13/20

Epoch 00013: val_accuracy improved from 0.83975 to 0.84521, saving model
62268/62268 - 29s - loss: 0.5935 - accuracy: 0.8103 - val_loss: 0.5003 -
Epoch 14/20

Epoch 00014: val_accuracy improved from 0.84521 to 0.84748, saving model
```

```
evaluate_model(model)
```

```
Test accuracy is 0.8319376111030579
Test loss is 0.5456908703738733
```

```
plotting(history_CNN.history)
```



# 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.

```
def compileModel(model):
    model.compile(
          optimizer = 'adam',
          loss = 'categorical_crossentropy',
          metrics = ['accuracy']
    )


checkpoint_best_MLP = 'model_MLP_checkpoints_best/checkpoint'
checkpoint_best_CNN = 'model_CNN_checkpoints_best/checkpoint'

model_MLP = get_new_model(train_images[1].shape,0.2,1e-5)
model_CNN = get_new_2_model(train_images[1].shape,0.2,1e-5)
compileModel(model_MLP)
compileModel(model_CNN)

model_MLP.load_weights(checkpoint_best_MLP)
model_CNN.load_weights(checkpoint_best_CNN)
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7fa49d23d450>
```

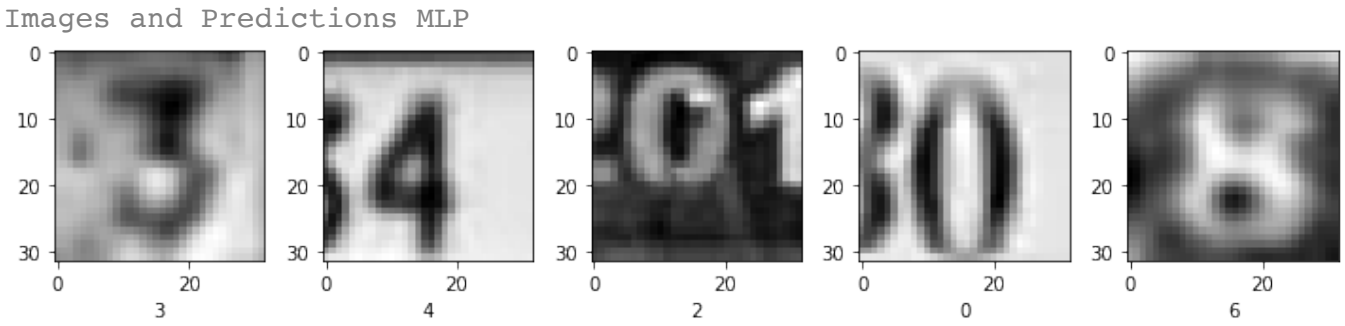
```
print('MLP accuracy:\n')
evaluate_model(model_MLP)
print('\nCNN accuracy:\n')
evaluate_model(model_CNN)
```

```
MLP accuracy:

Test accuracy is 0.5958051681518555
Test loss is 1.257245532705803

CNN accuracy:

Test accuracy is 0.8319376111030579
Test loss is 0.5456908703738733
```

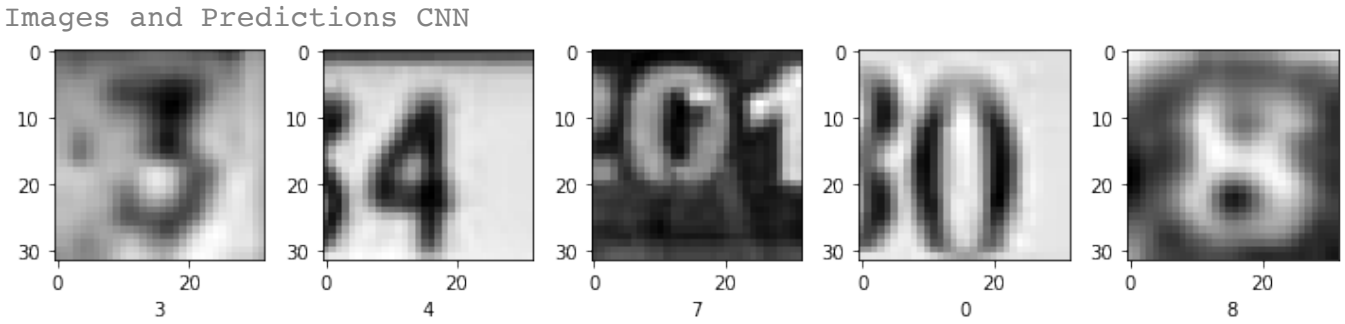
```
# get 5 test cases
sample_indices = random.sample(range(len(test_images)), 5)
new_test_images = test_images[sample_indices,:,:]
new_test_labels = test_labels[sample_indices]
new_test_labels = np.argmax(new_test_labels,axis=1)
```

```
def plotImages(model):
    fig, ax = plt.subplots(1, 5, figsize=(10, 10))

    preds = {}
    for j in range(5):
        predictions = model.predict(new_test_images[j][np.newaxis, ...])
        preds[j]=predictions
        fig.suptitle('')
        #ax[i].set_axis_off()
        ax[j].imshow(new_test_images[j,:,:],cmap=plt.get_cmap('gray'))
        ax[j].set_xlabel(str(np.argmax(predictions)))
        fig.tight_layout()
    plt.show()
    return preds
```

```
print('Images and Predictions MLP')
preds_MLP=plotImages(model_MLP)
```




```
print('Images and Predictions CNN')
preds_CNN=plotImages(model_CNN)
```



```
i = 4
interesting_result= new_test_images[i]
interesting_result_label= new_test_labels[i]
fig, ax = plt.subplots()
keys = range(10)
ax.bar(np.arange(len(keys)) - 0.2,
       preds_MLP[i][0],
       width=0.2,
       color='b',
       align='center', label = 'MLP')
ax.bar(np.arange(len(keys)) + 0.2,
       preds_CNN[i][0],
       width=0.2,
       color='g',
       align='center',label = 'CNN')
ax.legend()
ax.set_xticklabels(keys)
ax.set_xticks(np.arange(len(keys)))
ax.set_title('Prediction Bar Chart For Label: '+str(interesting_result_label))

plt.show()
```

```
/Users/metehangelgi/opt/anaconda3/envs/summerProj/lib/python3.7/site-packag
```