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
```

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')
```

```
Mounted at /content/gdrive
```

```
# Load the dataset from your Drive folder

train = loadmat('./gdrive/MyDrive/Colab Notebooks/train_32x32.mat')
test = loadmat('./gdrive/MyDrive/Colab Notebooks/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.

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

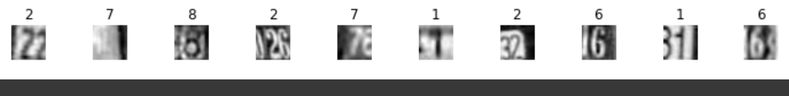
```
X_train, X_test, y_train, y_test = train['X'], test['X'], train['y'], test['y']
```

```
#Digit 0 is encoded as 10, so we change it back to 0
y_train, y_test = np.where(y_train==10,0,y_train), np.where(y_test==10,0,y_test)
```

```
#Choose 10 random images from the rgb data

idxs = np.random.choice(range(X_train.shape[3]), 10)
images = X_train[...,idxs]
targets = y_train[idxs]

fig, ax = plt.subplots(1, 10, figsize=(10, 1))
fig.tight_layout()
for i in range(10):
    ax[i].set_axis_off()
    ax[i].imshow(images[...,i])
    ax[i].title.set_text(str(targets[i][0]))
```


```
#Take average across channel axis, keep dimensions
X_train, X_test = np.mean(X_train, axis =2, keepdims=True), np.mean(X_test, axis=2, keepdims=True)
```

```
#Choose 10 random images from the grayscale data

idxs = np.random.choice(range(X_train.shape[3]), 10)
images = X_train[...,idxs]
targets = y_train[idxs]

fig, ax = plt.subplots(1, 10, figsize=(10, 1))
fig.tight_layout()
for i in range(10):
    ax[i].set_axis_off()
    ax[i].imshow(images[...,0,i], cmap='gray')
    ax[i].title.set_text(str(targets[i][0]))
```



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 Dense, Flatten
from tensorflow.keras.models import Sequential
```

```
def get_mlp_model(input_shape):
  model = Sequential([
                      Flatten(input_shape = input_shape),
                      Dense(128, activation='relu'),
                      Dense(128, activation='relu'),
                      Dense(64, activation='relu'),
                      Dense(10, activation='softmax')
  ])
  model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(), metrics= ['accuracy'])
  return model
```

```
#Reorder the feature matrices to have the samples in the first dimension and the channel in the last dimension
X_train, X_test = np.transpose(X_train, (3,0,1,2)), np.transpose(X_test, (3,0,1,2))
#Preprocess X_train and X_test
```

```
X_train, X_test = X_train/255.0 , X_test/255.0
```

```
mlp_model = get_mlp_model(X_train[0,...].shape)
print(mlp_model.summary())
```

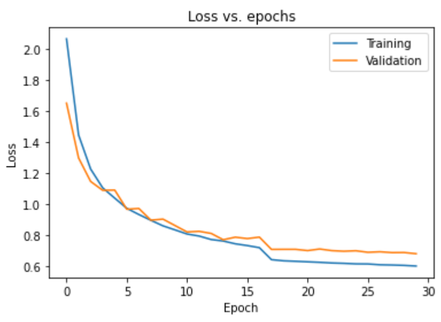
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 128)               131200
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 156,618
Trainable params: 156,618
Non-trainable params: 0
_________________________________________________________________
None
```

```
print(X_train.shape)
print(y_train.shape)
```

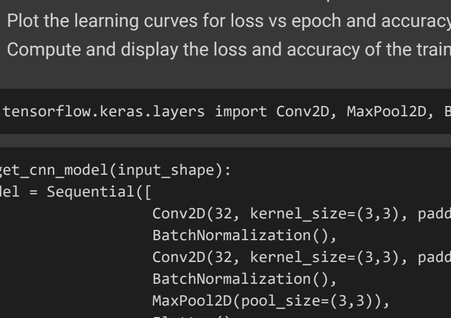
```
(73257, 32, 32, 1)
(73257, 1)
```

```
from tensorflow.keras.callbacks import  EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
filepath='mlp_model_checkpoints/checkpoint'

reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.2, patience=3)
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy', save_weights_only=True, save_best_only=True)

mlp_history = mlp_model.fit(X_train, y_train, batch_size=128, validation_split=0.2, epochs=30, callbacks=[reduce_lr, checkpoint])
```

```
plt.plot(mlp_history.history['loss'])
plt.plot(mlp_history.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```




```
plt.plot(mlp_history.history['accuracy'])
plt.plot(mlp_history.history['val_accuracy'])
plt.title('Accuracy vs. epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```




```
test_loss, test_accuracy = mlp_model.evaluate(X_test, y_test)
print(f'Test loss is {test_loss}')
print(f'Test accuracy is {test_accuracy}')
```

```
814/814 [==============================] - 1s 2ms/step - loss: 0.7902 - accuracy: 0.7753
Test loss is 0.7902397513389587
Test accuracy is 0.77531498670578
```

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 Conv2D, MaxPool2D, BatchNormalization, Dropout
```

```
def get_cnn_model(input_shape):
  model = Sequential([
                      Conv2D(32, kernel_size=(3,3), padding='SAME', activation='relu', input_shape=input_shape),
                      BatchNormalization(),
                      Conv2D(32, kernel_size=(3,3), padding='SAME', activation='relu'),
                      BatchNormalization(),
                      MaxPool2D(pool_size=(3,3)),
                      Flatten(),
                      Dense(32, activation='relu'),
                      Dropout(0.3),
                      Dense(10, activation='softmax')
  ])
  model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(),metrics= ['accuracy'])
  return model
```

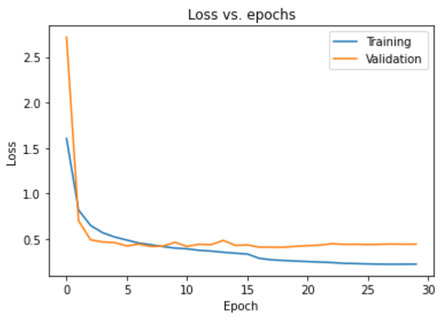
```
cnn_model = get_cnn_model(X_train[0,...].shape)
print(cnn_model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        320
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 10, 10, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                102432
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                330
=================================================================
Total params: 112,586
Trainable params: 112,458
Non-trainable params: 128
_________________________________________________________________
None
```

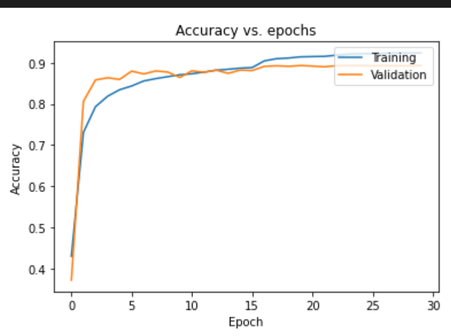
```
filepath='cnn_model_checkpoints/checkpoint'
```

```
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.2, patience=3)
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy', save_weights_only=True, save_best_only=True)


cnn_history = cnn_model.fit(X_train, y_train, batch_size=128, validation_split=0.2, epochs=30, callbacks=[reduce_lr, checkpoint])
```

```
plt.plot(cnn_history.history['loss'])
plt.plot(cnn_history.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```




```
plt.plot(cnn_history.history['accuracy'])
plt.plot(cnn_history.history['val_accuracy'])
plt.title('Accuracy vs. epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```




```
test_loss, test_accuracy = cnn_model.evaluate(X_test, y_test)
print(f'Test loss is {test_loss}')
print(f'Test accuracy is {test_accuracy}')
```

```
814/814 [==============================] - 2s 2ms/step - loss: 0.4922 - accuracy: 0.8834
Test loss is 0.49216505885124207
Test accuracy is 0.8834127187728882
```

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.

```
mlp = get_mlp_model(X_train[0].shape)
mlp.load_weights('mlp_model_checkpoints/checkpoint')
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f5e1cac18d0>
```

```
cnn = get_cnn_model(X_train[0].shape)
cnn.load_weights('cnn_model_checkpoints/checkpoint')
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f5dc034c080>
```

```
num_test_images = X_test.shape[0]

random_inx = np.random.choice(num_test_images, 5)
random_test_images = X_test[random_inx, ...]
random_test_labels = y_test[random_inx, ...]

mlp_predictions = mlp.predict(random_test_images)

cnn_predictions = cnn.predict(random_test_images)

fig, axes = plt.subplots(5, 3, figsize=(16, 12))
fig.tight_layout()
for i, (mlp_predictions, cnn_predictions, image, label) in enumerate(zip(mlp_predictions, cnn_predictions, random_test_images, random_test_labels)):
    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(10., -1.5, f'Digit {label}')
    axes[i, 1].bar(np.arange(len(mlp_predictions)), mlp_predictions)
    axes[i, 1].set_xticks(np.arange(len(mlp_predictions)))
    axes[i, 1].set_title(f"MLP model prediction: {np.argmax(mlp_predictions)}")
    axes[i, 2].bar(np.arange(len(cnn_predictions)), cnn_predictions)
    axes[i, 2].set_xticks(np.arange(len(cnn_predictions)))
    axes[i, 2].set_title(f"CNN model prediction: {np.argmax(cnn_predictions)}")

plt.show()
```