capstone_imperial

August 31, 2020

1 Capstone Project

1.1 Image classifier for the SVHN dataset

1.1.1 Instructions

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

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

1.1.2 How to submit

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

1.1.3 Let's get started!

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

[86]:
```
import tensorflow as tf
from scipy.io import loadmat
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, BatchNormalization,␣
↪MaxPool2D, Dense
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
```

```
import random
%matplotlib inline
```

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.

[87]:
```
# Run this cell to connect to your Drive folder

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

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call
drive.mount("/content/gdrive", force_remount=True).
```

[88]:
```
!wget --no-check-certificate \
    http://ufldl.stanford.edu/housenumbers/train_32x32.mat\
   -O /tmp/train_32x32.mat
```

```
--2020-08-31 05:03:04--  http://ufldl.stanford.edu/housenumbers/train_32x32.mat
Resolving ufldl.stanford.edu (ufldl.stanford.edu)... 171.64.68.10
Connecting to ufldl.stanford.edu (ufldl.stanford.edu)|171.64.68.10|:80...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 182040794 (174M) [text/plain]
Saving to: /tmp/train_32x32.mat

/tmp/train_32x32.ma 100%[===================>] 173.61M  10.4MB/s    in 17s

2020-08-31 05:03:21 (10.1 MB/s) - /tmp/train_32x32.mat saved
[182040794/182040794]
```

[89]:
```
!wget --no-check-certificate \
    http://ufldl.stanford.edu/housenumbers/test_32x32.mat\
    -O /tmp/test_32x32.mat
```

```
--2020-08-31 05:03:21--  http://ufldl.stanford.edu/housenumbers/test_32x32.mat
Resolving ufldl.stanford.edu (ufldl.stanford.edu)... 171.64.68.10
Connecting to ufldl.stanford.edu (ufldl.stanford.edu)|171.64.68.10|:80...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 64275384 (61M) [text/plain]
Saving to: /tmp/test_32x32.mat

/tmp/test_32x32.mat 100%[===================>]  61.30M  10.5MB/s    in 6.6s

2020-08-31 05:03:28 (9.22 MB/s) - /tmp/test_32x32.mat saved
[64275384/64275384]
```

[90]:
```
# Load the dataset from your Drive folder
train = loadmat('/tmp/train_32x32.mat')
test = loadmat('/tmp/test_32x32.mat')
```

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

1.2 1. Inspect and preprocess the dataset

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

[91]:
```
X_train = train['X']
X_test = test['X']
y_train = train['y']
y_test = test['y']
```

[92]:
```
X_train.shape, X_test.shape
```

[92]:
```
((32, 32, 3, 73257), (32, 32, 3, 26032))
```

[93]:
```
X_train = np.moveaxis(X_train, -1, 0)
X_test = np.moveaxis(X_test, -1, 0)
```

[94]:
```
plt.figure(figsize=(8,4))
for i in range(10):
    plt.subplot(2, 5, i + 1, xticks=[], yticks=[])
    plt.imshow(X_train[i, :, :, :])
    plt.xlabel(f'{y_train[i]}', color = 'green')
plt.tight_layout()
```

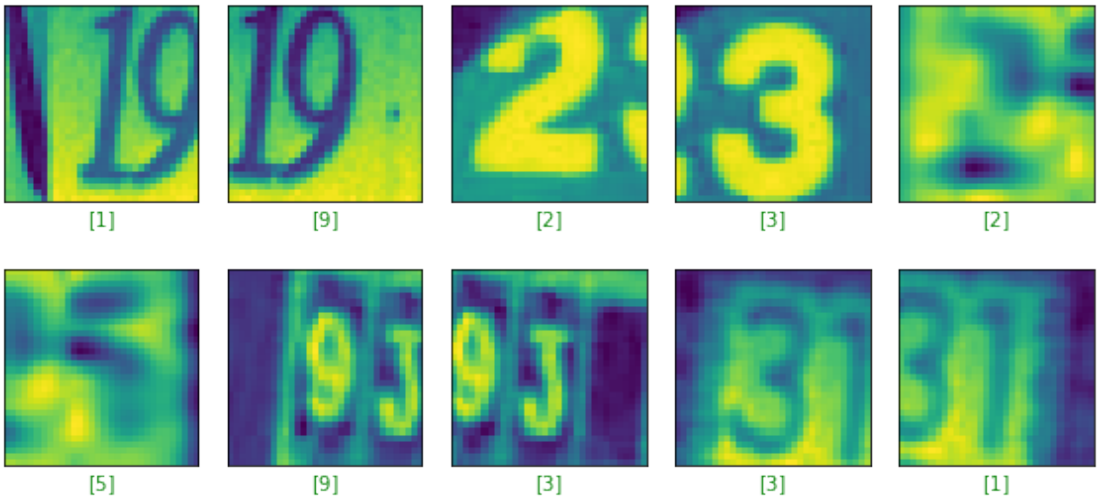
```
plt.show()
```

[95]:
```
X_train_gs = np.mean(X_train, 3).reshape(73257, 32, 32, 1)/255
X_test_gs = np.mean(X_test,3).reshape(26032, 32,32 ,1)/255
X_train_for_plotting = np.mean(X_train,3)
```

[96]:
```
X_train_gs.shape
```

[96]: (73257, 32, 32, 1)

[97]:
```
plt.figure(figsize=(8,4))
for i in range(10):
    plt.subplot(2, 5, i + 1, xticks=[], yticks=[])
    plt.imshow(X_train_for_plotting[i, :, :,  ])
    plt.xlabel(f'{y_train[i]}', color = 'green')
plt.tight_layout()
plt.show()
```



[98]:
```
X_train[0].shape
```

[98]:
```
(32, 32, 3)
```

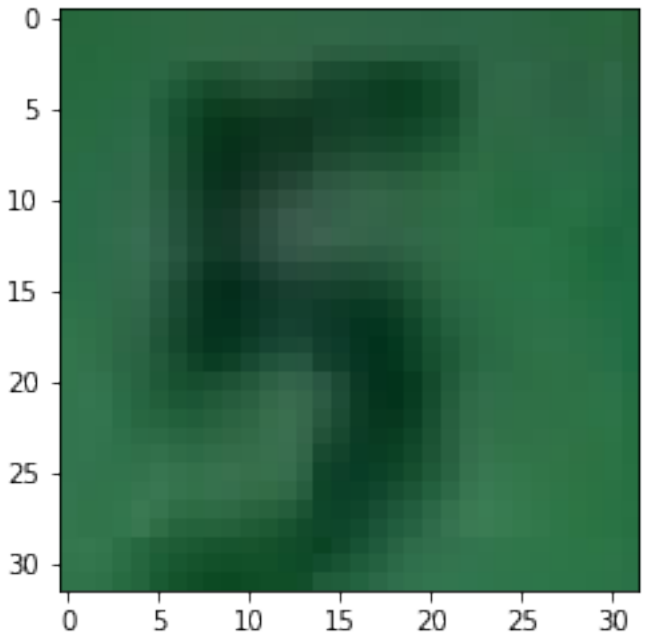
[99]:
```
enc = OneHotEncoder().fit(y_train)
y_train_oh = enc.transform(y_train).toarray()
y_test_oh = enc.transform(y_test).toarray()
```

[100]:
```
y_test_oh[0]
```

[100]:
```
array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0.])
```

[101]:
```
plt.imshow(X_test[0])
```

[101]:
```
<matplotlib.image.AxesImage at 0x7f07a4b1c668>
```



1.3 2. MLP neural network classifier

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

[102]:
```
checkpoint = ModelCheckpoint(filepath = 'checkpoints/checkpoint',␣
↪save_best_only=True, save_weights_only=True, monitor='val_loss', verbose=1)
earlystop = EarlyStopping(patience=5, monitor='loss')
```

[103]:
```
model = Sequential([
    Flatten(input_shape=X_train[0].shape),
    Dense(128*4, activation='relu'),
    Dense(64, activation='relu'),
    BatchNormalization(),
```

```
    Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    Dense(32, activation='relu'),
    Dense(10, activation='softmax')
])
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_4 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_16 (Dense)             (None, 512)               1573376
_________________________________________________________________
dense_17 (Dense)             (None, 64)                32832
_________________________________________________________________
batch_normalization_4 (Batch (None, 64)                256
_________________________________________________________________
dense_18 (Dense)             (None, 64)                4160
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_19 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_20 (Dense)             (None, 10)                330
=================================================================
Total params: 1,613,034
Trainable params: 1,612,906
Non-trainable params: 128
_________________________________________________________________
```

[104]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[105]:
```
history = model.fit(X_train, y_train_oh, callbacks=[checkpoint, earlystop],␣
↪batch_size=128, validation_data=(X_test, y_test_oh), epochs=30)
```

```
Epoch 1/30
569/573 [============================>.] - ETA: 0s - loss: 2.0406 - accuracy:
0.2683
Epoch 00001: val_loss improved from inf to 2.59939, saving model to
checkpoints/checkpoint
573/573 [==============================] - 6s 10ms/step - loss: 2.0387 -
accuracy: 0.2691 - val_loss: 2.5994 - val_accuracy: 0.3024
Epoch 2/30
573/573 [==============================] - ETA: 0s - loss: 1.6031 - accuracy:
```

```
0.4552
Epoch 00002: val_loss improved from 2.59939 to 1.77156, saving model to
checkpoints/checkpoint
573/573 [==============================] - 5s 9ms/step - loss: 1.6031 -
accuracy: 0.4552 - val_loss: 1.7716 - val_accuracy: 0.4667
Epoch 3/30
573/573 [==============================] - ETA: 0s - loss: 1.4262 - accuracy:
0.5339
Epoch 00003: val_loss improved from 1.77156 to 1.53036, saving model to
checkpoints/checkpoint
573/573 [==============================] - 5s 9ms/step - loss: 1.4262 -
accuracy: 0.5339 - val_loss: 1.5304 - val_accuracy: 0.5225
Epoch 4/30
566/573 [============================>.] - ETA: 0s - loss: 1.3788 - accuracy:
0.5539
Epoch 00004: val_loss improved from 1.53036 to 1.32510, saving model to
checkpoints/checkpoint
573/573 [==============================] - 5s 9ms/step - loss: 1.3783 -
accuracy: 0.5541 - val_loss: 1.3251 - val_accuracy: 0.5800
Epoch 5/30
568/573 [============================>.] - ETA: 0s - loss: 1.3128 - accuracy:
0.5763
Epoch 00005: val_loss did not improve from 1.32510
573/573 [==============================] - 5s 8ms/step - loss: 1.3130 -
accuracy: 0.5765 - val_loss: 1.3803 - val_accuracy: 0.5528
Epoch 6/30
573/573 [==============================] - ETA: 0s - loss: 1.2597 - accuracy:
0.5983
Epoch 00006: val_loss did not improve from 1.32510
573/573 [==============================] - 5s 8ms/step - loss: 1.2597 -
accuracy: 0.5983 - val_loss: 1.6395 - val_accuracy: 0.5391
Epoch 7/30
568/573 [============================>.] - ETA: 0s - loss: 1.2499 - accuracy:
0.6007
Epoch 00007: val_loss did not improve from 1.32510
573/573 [==============================] - 5s 8ms/step - loss: 1.2507 -
accuracy: 0.6001 - val_loss: 1.5898 - val_accuracy: 0.5067
Epoch 8/30
567/573 [============================>.] - ETA: 0s - loss: 1.2603 - accuracy:
0.5979
Epoch 00008: val_loss did not improve from 1.32510
573/573 [==============================] - 5s 8ms/step - loss: 1.2606 -
accuracy: 0.5978 - val_loss: 1.3637 - val_accuracy: 0.5841
Epoch 9/30
567/573 [============================>.] - ETA: 0s - loss: 1.2126 - accuracy:
0.6155
Epoch 00009: val_loss improved from 1.32510 to 1.27469, saving model to
checkpoints/checkpoint
```

```
573/573 [==============================] - 5s 8ms/step - loss: 1.2122 -
accuracy: 0.6156 - val_loss: 1.2747 - val_accuracy: 0.5978
Epoch 10/30
571/573 [===========================>.] - ETA: 0s - loss: 1.1906 - accuracy:
0.6246
Epoch 00010: val_loss did not improve from 1.27469
573/573 [==============================] - 4s 8ms/step - loss: 1.1907 -
accuracy: 0.6245 - val_loss: 1.4213 - val_accuracy: 0.5804
Epoch 11/30
570/573 [===========================>.] - ETA: 0s - loss: 1.1828 - accuracy:
0.6275
Epoch 00011: val_loss improved from 1.27469 to 1.23048, saving model to
checkpoints/checkpoint
573/573 [==============================] - 4s 8ms/step - loss: 1.1828 -
accuracy: 0.6274 - val_loss: 1.2305 - val_accuracy: 0.6059
Epoch 12/30
572/573 [===========================>.] - ETA: 0s - loss: 1.1624 - accuracy:
0.6342
Epoch 00012: val_loss did not improve from 1.23048
573/573 [==============================] - 4s 8ms/step - loss: 1.1623 -
accuracy: 0.6342 - val_loss: 1.3304 - val_accuracy: 0.5772
Epoch 13/30
567/573 [===========================>.] - ETA: 0s - loss: 1.1489 - accuracy:
0.6378
Epoch 00013: val_loss did not improve from 1.23048
573/573 [==============================] - 4s 8ms/step - loss: 1.1490 -
accuracy: 0.6378 - val_loss: 1.2941 - val_accuracy: 0.6116
Epoch 14/30
569/573 [===========================>.] - ETA: 0s - loss: 1.1364 - accuracy:
0.6444
Epoch 00014: val_loss improved from 1.23048 to 1.16136, saving model to
checkpoints/checkpoint
573/573 [==============================] - 5s 8ms/step - loss: 1.1366 -
accuracy: 0.6444 - val_loss: 1.1614 - val_accuracy: 0.6611
Epoch 15/30
568/573 [===========================>.] - ETA: 0s - loss: 1.1230 - accuracy:
0.6457
Epoch 00015: val_loss did not improve from 1.16136
573/573 [==============================] - 5s 8ms/step - loss: 1.1224 -
accuracy: 0.6460 - val_loss: 1.2735 - val_accuracy: 0.6429
Epoch 16/30
571/573 [===========================>.] - ETA: 0s - loss: 1.1118 - accuracy:
0.6515
Epoch 00016: val_loss did not improve from 1.16136
573/573 [==============================] - 5s 8ms/step - loss: 1.1120 -
accuracy: 0.6515 - val_loss: 1.2753 - val_accuracy: 0.6174
Epoch 17/30
571/573 [===========================>.] - ETA: 0s - loss: 1.1022 - accuracy:
```

```
0.6531
Epoch 00017: val_loss improved from 1.16136 to 1.13408, saving model to
checkpoints/checkpoint
573/573 [==============================] - 4s 8ms/step - loss: 1.1024 -
accuracy: 0.6531 - val_loss: 1.1341 - val_accuracy: 0.6478
Epoch 18/30
571/573 [============================>.] - ETA: 0s - loss: 1.0911 - accuracy:
0.6577
Epoch 00018: val_loss did not improve from 1.13408
573/573 [==============================] - 4s 8ms/step - loss: 1.0907 -
accuracy: 0.6578 - val_loss: 1.1472 - val_accuracy: 0.6529
Epoch 19/30
569/573 [============================>.] - ETA: 0s - loss: 1.0828 - accuracy:
0.6619
Epoch 00019: val_loss did not improve from 1.13408
573/573 [==============================] - 4s 8ms/step - loss: 1.0826 -
accuracy: 0.6620 - val_loss: 1.2181 - val_accuracy: 0.6457
Epoch 20/30
570/573 [============================>.] - ETA: 0s - loss: 1.0745 - accuracy:
0.6627
Epoch 00020: val_loss did not improve from 1.13408
573/573 [==============================] - 4s 8ms/step - loss: 1.0747 -
accuracy: 0.6626 - val_loss: 1.2245 - val_accuracy: 0.6388
Epoch 21/30
571/573 [============================>.] - ETA: 0s - loss: 1.0683 - accuracy:
0.6656
Epoch 00021: val_loss did not improve from 1.13408
573/573 [==============================] - 5s 8ms/step - loss: 1.0682 -
accuracy: 0.6656 - val_loss: 1.3105 - val_accuracy: 0.6329
Epoch 22/30
570/573 [============================>.] - ETA: 0s - loss: 1.0544 - accuracy:
0.6685
Epoch 00022: val_loss improved from 1.13408 to 1.12363, saving model to
checkpoints/checkpoint
573/573 [==============================] - 5s 8ms/step - loss: 1.0545 -
accuracy: 0.6685 - val_loss: 1.1236 - val_accuracy: 0.6516
Epoch 23/30
570/573 [============================>.] - ETA: 0s - loss: 1.0522 - accuracy:
0.6697
Epoch 00023: val_loss improved from 1.12363 to 1.09229, saving model to
checkpoints/checkpoint
573/573 [==============================] - 5s 9ms/step - loss: 1.0524 -
accuracy: 0.6696 - val_loss: 1.0923 - val_accuracy: 0.6575
Epoch 24/30
567/573 [============================>.] - ETA: 0s - loss: 1.0409 - accuracy:
0.6744
Epoch 00024: val_loss did not improve from 1.09229
573/573 [==============================] - 6s 10ms/step - loss: 1.0412 -
```

```
accuracy: 0.6743 - val_loss: 1.1151 - val_accuracy: 0.6604
Epoch 25/30
568/573 [============================>.] - ETA: 0s - loss: 1.0261 - accuracy:
0.6777
Epoch 00025: val_loss did not improve from 1.09229
573/573 [==============================] - 6s 10ms/step - loss: 1.0266 -
accuracy: 0.6775 - val_loss: 1.1475 - val_accuracy: 0.6374
Epoch 26/30
571/573 [============================>.] - ETA: 0s - loss: 1.0327 - accuracy:
0.6760
Epoch 00026: val_loss improved from 1.09229 to 1.03629, saving model to
checkpoints/checkpoint
573/573 [==============================] - 5s 9ms/step - loss: 1.0327 -
accuracy: 0.6760 - val_loss: 1.0363 - val_accuracy: 0.6778
Epoch 27/30
572/573 [============================>.] - ETA: 0s - loss: 1.0204 - accuracy:
0.6798
Epoch 00027: val_loss did not improve from 1.03629
573/573 [==============================] - 5s 9ms/step - loss: 1.0204 -
accuracy: 0.6798 - val_loss: 1.0407 - val_accuracy: 0.6822
Epoch 28/30
570/573 [============================>.] - ETA: 0s - loss: 1.0127 - accuracy:
0.6838
Epoch 00028: val_loss improved from 1.03629 to 0.99774, saving model to
checkpoints/checkpoint
573/573 [==============================] - 5s 9ms/step - loss: 1.0125 -
accuracy: 0.6838 - val_loss: 0.9977 - val_accuracy: 0.6945
Epoch 29/30
568/573 [============================>.] - ETA: 0s - loss: 1.0145 - accuracy:
0.6822
Epoch 00029: val_loss did not improve from 0.99774
573/573 [==============================] - 5s 9ms/step - loss: 1.0141 -
accuracy: 0.6824 - val_loss: 1.0721 - val_accuracy: 0.6614
Epoch 30/30
572/573 [============================>.] - ETA: 0s - loss: 1.0061 - accuracy:
0.6840
Epoch 00030: val_loss improved from 0.99774 to 0.98920, saving model to
checkpoints/checkpoint
573/573 [==============================] - 5s 9ms/step - loss: 1.0060 -
accuracy: 0.6840 - val_loss: 0.9892 - val_accuracy: 0.6924
```

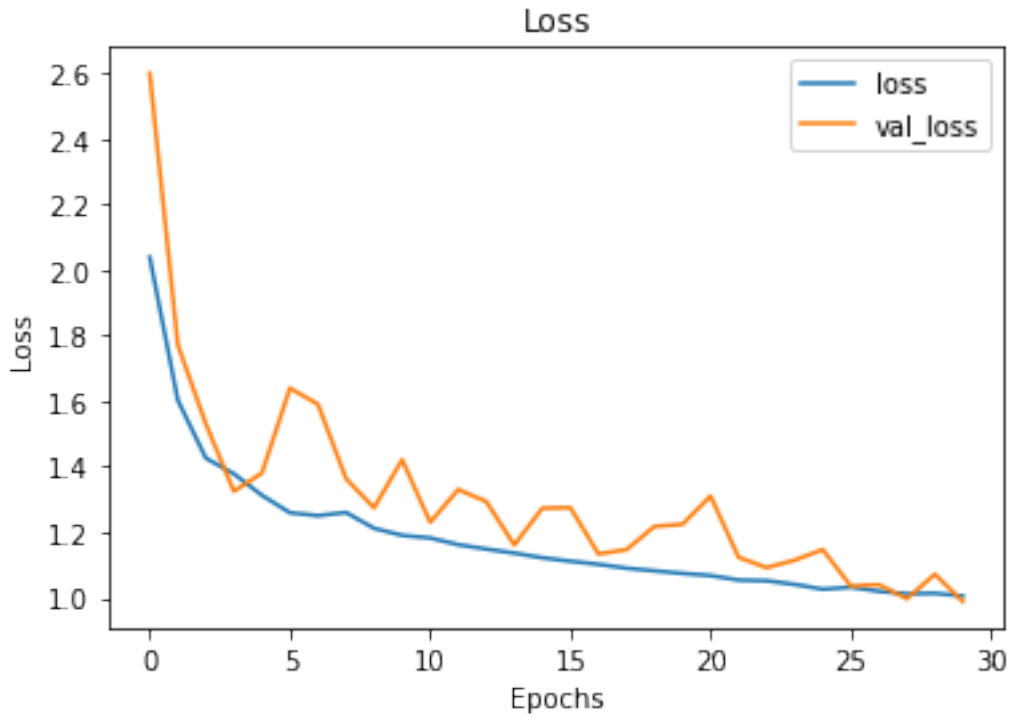
[106]:
```
! dir
```

```
checkpoints  gdrive  sample_data
```

[107]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

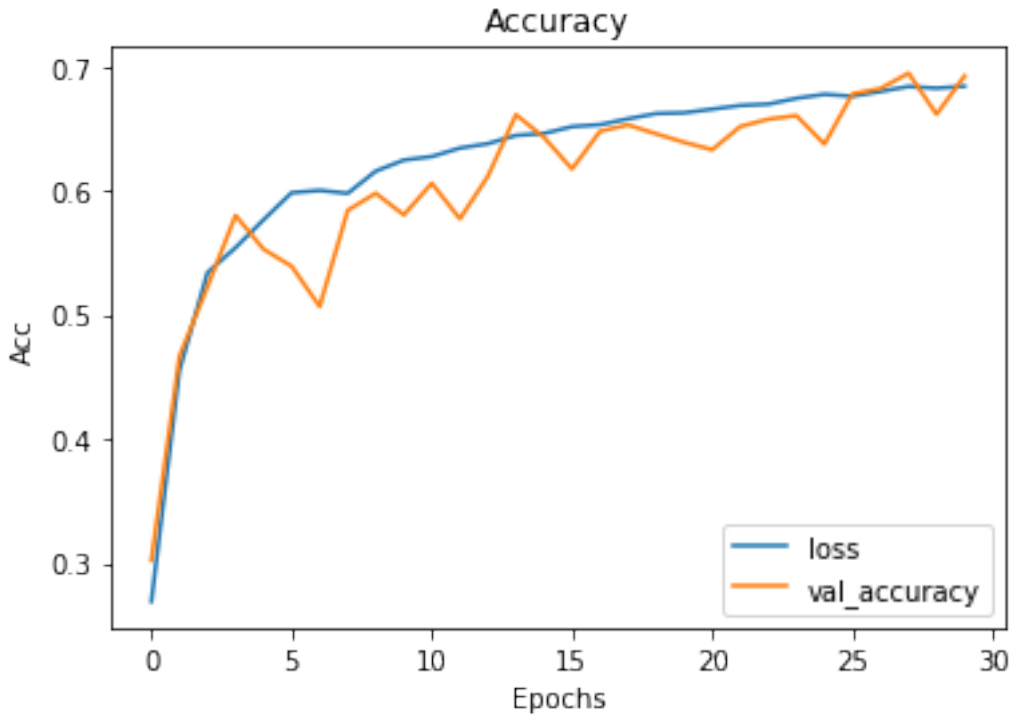
```
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['loss','val_loss'], loc='upper right')
plt.title("Loss")
```

[107]:
```
Text(0.5, 1.0, 'Loss')
```



[108]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('Epochs')
plt.ylabel('Acc')
plt.legend(['loss','val_accuracy'], loc='lower right')
plt.title("Accuracy")
```

[108]:
```
Text(0.5, 1.0, 'Accuracy')
```



1.4 3. CNN neural network classifier

- Build a CNN classifier model using the Sequential API. Your model should use the Conv2D, MaxPool2D, BatchNormalization, Flatten, Dense and Dropout layers. The final layer should again have a 10-way softmax output.
- You should design and build the model yourself. Feel free to experiment with different CNN architectures. *Hint: to achieve a reasonable accuracy you won't need to use more than 2 or 3 convolutional layers and 2 fully connected layers.)*
- The CNN model should use fewer trainable parameters than your MLP model.
- Compile and train the model (we recommend a maximum of 30 epochs), making use of both training and validation sets during the training run.
- Your model should track at least one appropriate metric, and use at least two callbacks during training, one of which should be a ModelCheckpoint callback.
- You should aim to beat the MLP model performance with fewer parameters!
- Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and validation sets.
- Compute and display the loss and accuracy of the trained model on the test set.

[109]:
```
model2 = Sequential([
    Conv2D(filters= 16, kernel_size= 3, activation='relu',␣
↪input_shape=X_train[0].shape),
    MaxPool2D(pool_size= (3,3), strides=1),
```

```
    Conv2D(filters= 32, kernel_size = 3, padding='valid', strides=1,␣
 ↪activation='relu'),
    MaxPool2D(pool_size = (1,1), strides = 3),
    BatchNormalization(),
    Conv2D(filters= 32, kernel_size = 3, padding='valid', strides=2,␣
 ↪activation='relu'),
    tf.keras.layers.Dropout(0.5),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(32, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    Dense(10, activation='softmax')
])
```

[110]:
```
model2.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 30, 30, 16)        448
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 28, 28, 16)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 26, 26, 32)        4640
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 9, 9, 32)          0
_________________________________________________________________
batch_normalization_5 (Batch (None, 9, 9, 32)          128
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 4, 4, 32)          9248
_________________________________________________________________
dropout_7 (Dropout)          (None, 4, 4, 32)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_21 (Dense)             (None, 128)               65664
_________________________________________________________________
dense_22 (Dense)             (None, 32)                4128
_________________________________________________________________
dropout_8 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_23 (Dense)             (None, 10)                330
=================================================================
Total params: 84,586
Trainable params: 84,522
Non-trainable params: 64
_________________________________________________________________
```

[111]:
```
model2.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[112]:
```
callback1 = ModelCheckpoint(filepath='CNNweights', save_best_only=True,␣
↪save_weights_only=True, save_freq=5000,monitor='val_acc')
callback2 = EarlyStopping(monitor='loss',patience=7, verbose=1)
```

[113]:
```
X_train.shape
```

[113]:
```
(73257, 32, 32, 3)
```

[114]:
```
history = model2.fit(X_train, y_train_oh, callbacks=[checkpoint, earlystop],␣
↪batch_size=256, validation_data=(X_test, y_test_oh), epochs=30)
```

```
Epoch 1/30
285/287 [============================>.] - ETA: 0s - loss: 1.8455 - accuracy:
0.3551
Epoch 00001: val_loss did not improve from 0.98920
287/287 [==============================] - 6s 20ms/step - loss: 1.8430 -
accuracy: 0.3561 - val_loss: 1.3453 - val_accuracy: 0.5624
Epoch 2/30
287/287 [==============================] - ETA: 0s - loss: 1.0501 - accuracy:
0.6603
Epoch 00002: val_loss did not improve from 0.98920
287/287 [==============================] - 5s 18ms/step - loss: 1.0501 -
accuracy: 0.6603 - val_loss: 1.5492 - val_accuracy: 0.5148
Epoch 3/30
287/287 [==============================] - ETA: 0s - loss: 0.8436 - accuracy:
0.7356
Epoch 00003: val_loss improved from 0.98920 to 0.77412, saving model to
checkpoints/checkpoint
287/287 [==============================] - 5s 18ms/step - loss: 0.8436 -
accuracy: 0.7356 - val_loss: 0.7741 - val_accuracy: 0.7603
Epoch 4/30
285/287 [============================>.] - ETA: 0s - loss: 0.7423 - accuracy:
0.7701
Epoch 00004: val_loss did not improve from 0.77412
287/287 [==============================] - 5s 18ms/step - loss: 0.7418 -
accuracy: 0.7705 - val_loss: 1.2084 - val_accuracy: 0.5930
Epoch 5/30
287/287 [==============================] - ETA: 0s - loss: 0.6800 - accuracy:
0.7920
Epoch 00005: val_loss improved from 0.77412 to 0.67430, saving model to
checkpoints/checkpoint
287/287 [==============================] - 5s 18ms/step - loss: 0.6800 -
accuracy: 0.7920 - val_loss: 0.6743 - val_accuracy: 0.7903
Epoch 6/30
285/287 [============================>.] - ETA: 0s - loss: 0.6339 - accuracy:
0.8084
```

```
Epoch 00006: val_loss improved from 0.67430 to 0.66955, saving model to
checkpoints/checkpoint
287/287 [==============================] - 5s 18ms/step - loss: 0.6344 -
accuracy: 0.8082 - val_loss: 0.6696 - val_accuracy: 0.7925
Epoch 7/30
286/287 [============================>.] - ETA: 0s - loss: 0.5992 - accuracy:
0.8188
Epoch 00007: val_loss improved from 0.66955 to 0.65319, saving model to
checkpoints/checkpoint
287/287 [==============================] - 5s 18ms/step - loss: 0.5992 -
accuracy: 0.8188 - val_loss: 0.6532 - val_accuracy: 0.8026
Epoch 8/30
286/287 [============================>.] - ETA: 0s - loss: 0.5662 - accuracy:
0.8312
Epoch 00008: val_loss improved from 0.65319 to 0.43753, saving model to
checkpoints/checkpoint
287/287 [==============================] - 5s 18ms/step - loss: 0.5660 -
accuracy: 0.8312 - val_loss: 0.4375 - val_accuracy: 0.8719
Epoch 9/30
285/287 [============================>.] - ETA: 0s - loss: 0.5467 - accuracy:
0.8358
Epoch 00009: val_loss did not improve from 0.43753
287/287 [==============================] - 5s 19ms/step - loss: 0.5472 -
accuracy: 0.8357 - val_loss: 0.4450 - val_accuracy: 0.8709
Epoch 10/30
286/287 [============================>.] - ETA: 0s - loss: 0.5269 - accuracy:
0.8428
Epoch 00010: val_loss improved from 0.43753 to 0.43540, saving model to
checkpoints/checkpoint
287/287 [==============================] - 5s 19ms/step - loss: 0.5269 -
accuracy: 0.8428 - val_loss: 0.4354 - val_accuracy: 0.8725
Epoch 11/30
283/287 [============================>.] - ETA: 0s - loss: 0.5093 - accuracy:
0.8501
Epoch 00011: val_loss improved from 0.43540 to 0.41944, saving model to
checkpoints/checkpoint
287/287 [==============================] - 5s 19ms/step - loss: 0.5089 -
accuracy: 0.8501 - val_loss: 0.4194 - val_accuracy: 0.8801
Epoch 12/30
284/287 [============================>.] - ETA: 0s - loss: 0.4945 - accuracy:
0.8547
Epoch 00012: val_loss improved from 0.41944 to 0.37640, saving model to
checkpoints/checkpoint
287/287 [==============================] - 5s 19ms/step - loss: 0.4943 -
accuracy: 0.8546 - val_loss: 0.3764 - val_accuracy: 0.8890
Epoch 13/30
286/287 [============================>.] - ETA: 0s - loss: 0.4778 - accuracy:
0.8588
```

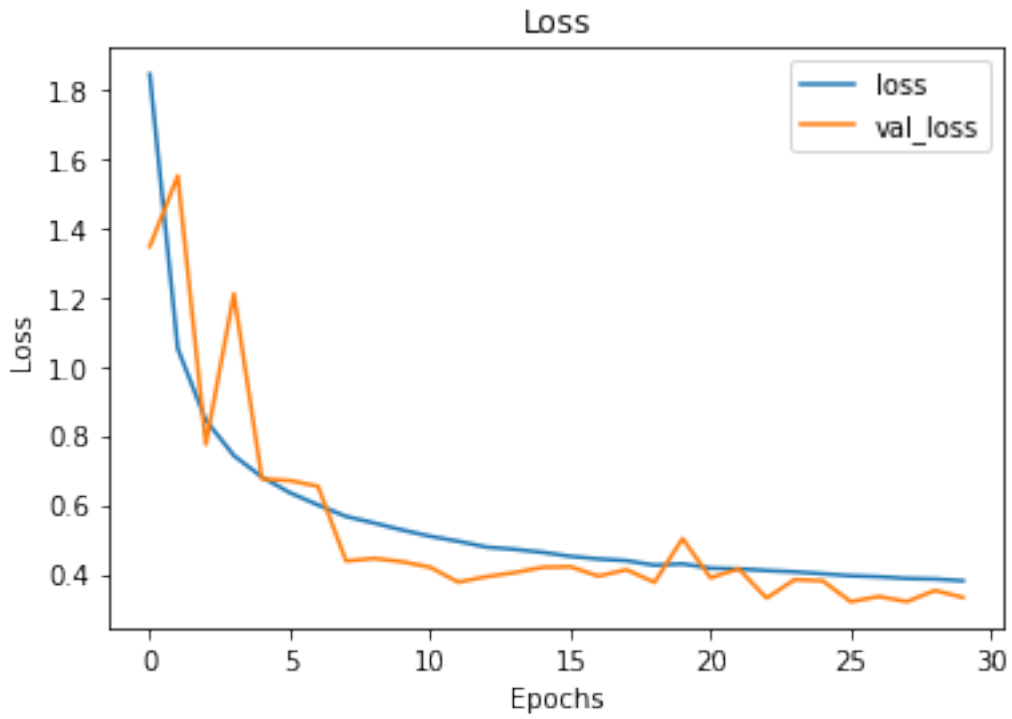
```
Epoch 00013: val_loss did not improve from 0.37640
287/287 [==============================] - 5s 18ms/step - loss: 0.4777 -
accuracy: 0.8588 - val_loss: 0.3922 - val_accuracy: 0.8865
Epoch 14/30
284/287 [============================>.] - ETA: 0s - loss: 0.4714 - accuracy:
0.8604
Epoch 00014: val_loss did not improve from 0.37640
287/287 [==============================] - 5s 18ms/step - loss: 0.4712 -
accuracy: 0.8605 - val_loss: 0.4038 - val_accuracy: 0.8792
Epoch 15/30
284/287 [============================>.] - ETA: 0s - loss: 0.4626 - accuracy:
0.8632
Epoch 00015: val_loss did not improve from 0.37640
287/287 [==============================] - 5s 17ms/step - loss: 0.4627 -
accuracy: 0.8631 - val_loss: 0.4190 - val_accuracy: 0.8789
Epoch 16/30
284/287 [============================>.] - ETA: 0s - loss: 0.4512 - accuracy:
0.8662
Epoch 00016: val_loss did not improve from 0.37640
287/287 [==============================] - 5s 17ms/step - loss: 0.4509 -
accuracy: 0.8662 - val_loss: 0.4211 - val_accuracy: 0.8765
Epoch 17/30
287/287 [==============================] - ETA: 0s - loss: 0.4433 - accuracy:
0.8697
Epoch 00017: val_loss did not improve from 0.37640
287/287 [==============================] - 5s 17ms/step - loss: 0.4433 -
accuracy: 0.8697 - val_loss: 0.3938 - val_accuracy: 0.8874
Epoch 18/30
286/287 [============================>.] - ETA: 0s - loss: 0.4383 - accuracy:
0.8711
Epoch 00018: val_loss did not improve from 0.37640
287/287 [==============================] - 5s 18ms/step - loss: 0.4384 -
accuracy: 0.8711 - val_loss: 0.4123 - val_accuracy: 0.8777
Epoch 19/30
285/287 [============================>.] - ETA: 0s - loss: 0.4251 - accuracy:
0.8764
Epoch 00019: val_loss improved from 0.37640 to 0.37589, saving model to
checkpoints/checkpoint
287/287 [==============================] - 5s 18ms/step - loss: 0.4252 -
accuracy: 0.8765 - val_loss: 0.3759 - val_accuracy: 0.8890
Epoch 20/30
284/287 [============================>.] - ETA: 0s - loss: 0.4290 - accuracy:
0.8745
Epoch 00020: val_loss did not improve from 0.37589
287/287 [==============================] - 5s 17ms/step - loss: 0.4291 -
accuracy: 0.8745 - val_loss: 0.5019 - val_accuracy: 0.8614
Epoch 21/30
285/287 [============================>.] - ETA: 0s - loss: 0.4164 - accuracy:
```

```
0.8767
Epoch 00021: val_loss did not improve from 0.37589
287/287 [==============================] - 5s 17ms/step - loss: 0.4168 -
accuracy: 0.8767 - val_loss: 0.3888 - val_accuracy: 0.8899
Epoch 22/30
284/287 [============================>.] - ETA: 0s - loss: 0.4145 - accuracy:
0.8767
Epoch 00022: val_loss did not improve from 0.37589
287/287 [==============================] - 5s 17ms/step - loss: 0.4148 -
accuracy: 0.8766 - val_loss: 0.4150 - val_accuracy: 0.8817
Epoch 23/30
285/287 [============================>.] - ETA: 0s - loss: 0.4094 - accuracy:
0.8796
Epoch 00023: val_loss improved from 0.37589 to 0.33050, saving model to
checkpoints/checkpoint
287/287 [==============================] - 5s 17ms/step - loss: 0.4099 -
accuracy: 0.8795 - val_loss: 0.3305 - val_accuracy: 0.9067
Epoch 24/30
284/287 [============================>.] - ETA: 0s - loss: 0.4067 - accuracy:
0.8814
Epoch 00024: val_loss did not improve from 0.33050
287/287 [==============================] - 5s 17ms/step - loss: 0.4062 -
accuracy: 0.8816 - val_loss: 0.3839 - val_accuracy: 0.8887
Epoch 25/30
285/287 [============================>.] - ETA: 0s - loss: 0.4002 - accuracy:
0.8832
Epoch 00025: val_loss did not improve from 0.33050
287/287 [==============================] - 5s 17ms/step - loss: 0.4004 -
accuracy: 0.8832 - val_loss: 0.3802 - val_accuracy: 0.8908
Epoch 26/30
285/287 [============================>.] - ETA: 0s - loss: 0.3949 - accuracy:
0.8843
Epoch 00026: val_loss improved from 0.33050 to 0.32007, saving model to
checkpoints/checkpoint
287/287 [==============================] - 5s 18ms/step - loss: 0.3947 -
accuracy: 0.8844 - val_loss: 0.3201 - val_accuracy: 0.9095
Epoch 27/30
285/287 [============================>.] - ETA: 0s - loss: 0.3919 - accuracy:
0.8858
Epoch 00027: val_loss did not improve from 0.32007
287/287 [==============================] - 5s 17ms/step - loss: 0.3920 -
accuracy: 0.8858 - val_loss: 0.3349 - val_accuracy: 0.9045
Epoch 28/30
286/287 [============================>.] - ETA: 0s - loss: 0.3870 - accuracy:
0.8866
Epoch 00028: val_loss did not improve from 0.32007
287/287 [==============================] - 5s 17ms/step - loss: 0.3870 -
accuracy: 0.8866 - val_loss: 0.3202 - val_accuracy: 0.9079
```

```
Epoch 29/30
286/287 [============================>.] - ETA: 0s - loss: 0.3854 - accuracy:
0.8872
Epoch 00029: val_loss did not improve from 0.32007
287/287 [==============================] - 5s 17ms/step - loss: 0.3854 -
accuracy: 0.8872 - val_loss: 0.3524 - val_accuracy: 0.8992
Epoch 30/30
286/287 [============================>.] - ETA: 0s - loss: 0.3805 - accuracy:
0.8876
Epoch 00030: val_loss did not improve from 0.32007
287/287 [==============================] - 5s 17ms/step - loss: 0.3804 -
accuracy: 0.8876 - val_loss: 0.3323 - val_accuracy: 0.9053
```

[115]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['loss','val_loss'], loc='upper right')
plt.title("Loss")
```

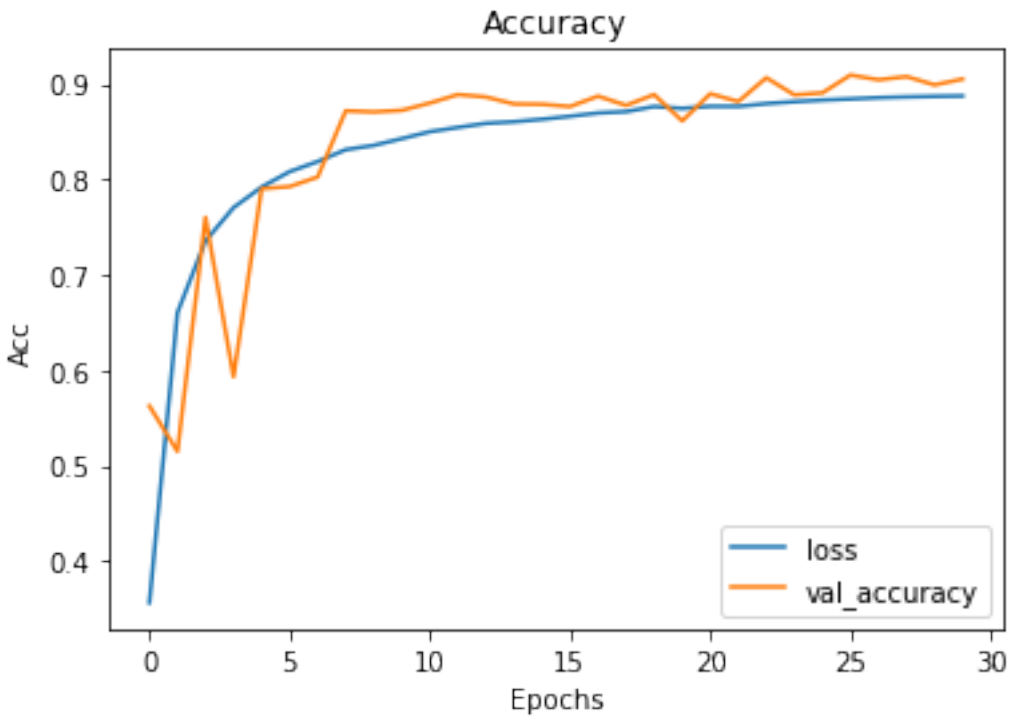
[115]:
```
Text(0.5, 1.0, 'Loss')
```



[116]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('Epochs')
```

```
plt.ylabel('Acc')
plt.legend(['loss','val_accuracy'], loc='lower right')
plt.title("Accuracy")
```

[116]:
```
Text(0.5, 1.0, 'Accuracy')
```



1.5 4. Get model predictions

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

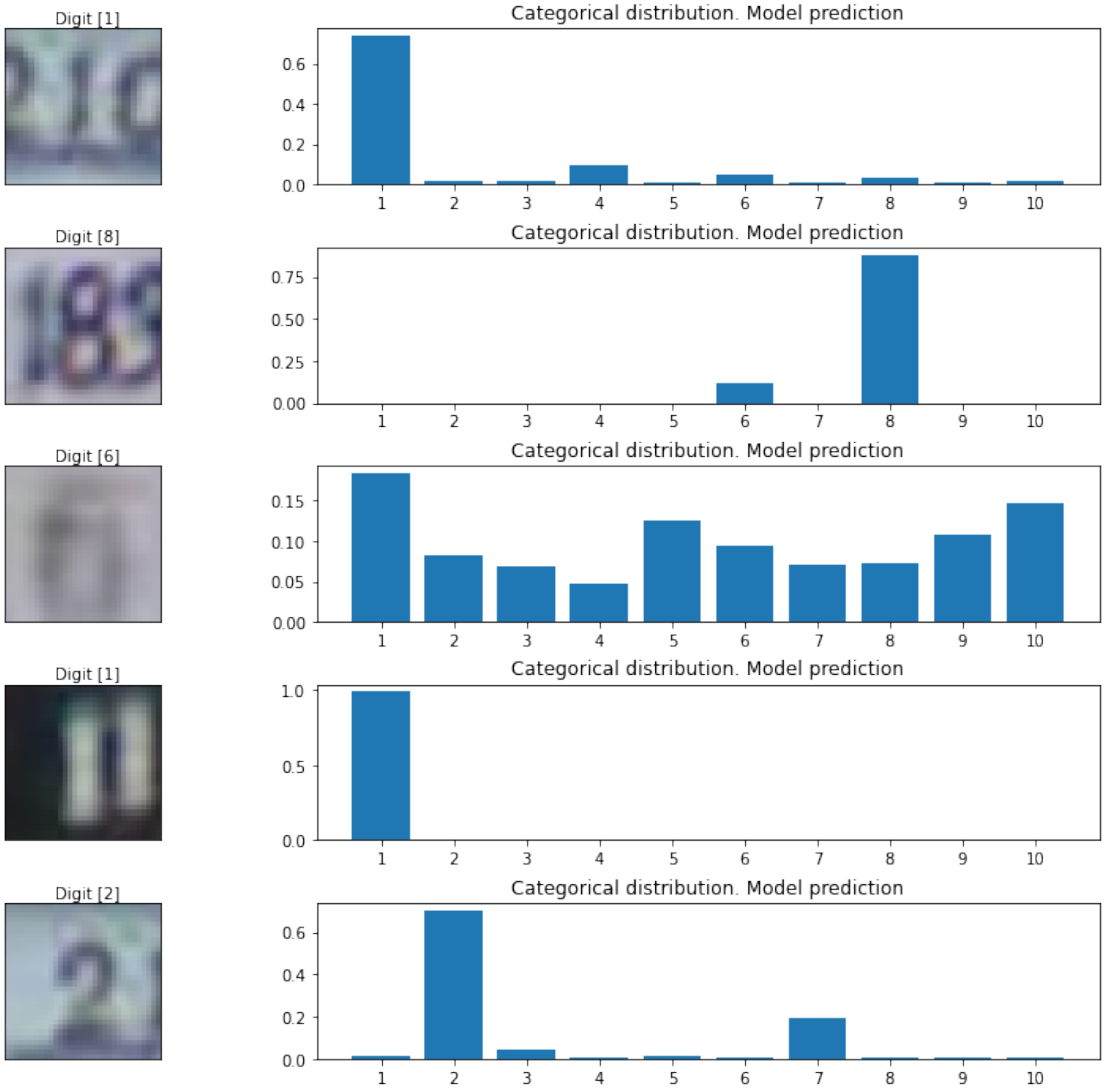
[117]:
```
! ls checkpoints
```

```
checkpoint  checkpoint.data-00000-of-00001  checkpoint.index
```

[118]:
```
random_inx = np.random.choice(10, 5)
random_test_images = X_test[random_inx, ...]
random_test_labels = y_test[random_inx, ...]

predictions = model.predict(random_test_images)
```

```
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,␣
 ↪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(1,11), prediction)
    axes[i, 1].set_xticks(np.arange(1,11))
    axes[i, 1].set_title("Categorical distribution. Model prediction")

plt.show()
```

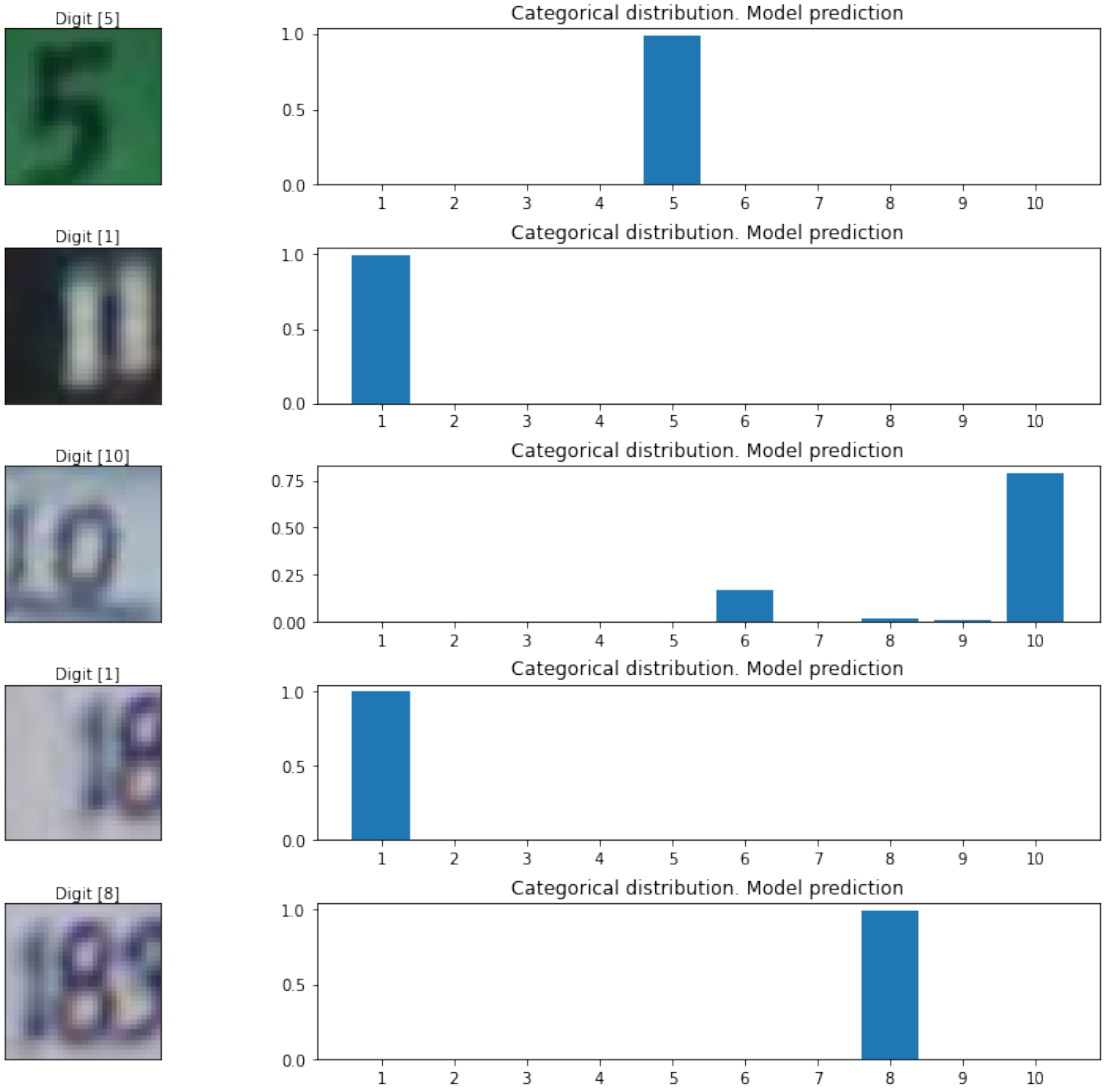
```
WARNING:tensorflow:5 out of the last 5 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x7f07a428fea0>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```



[119]:
```
num_test_images = X_test.shape[0]

random_inx = np.random.choice(10, 5)
random_test_images = X_test[random_inx, ...]
random_test_labels = y_test[random_inx, ...]

predictions = model2.predict(random_test_images)

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,␣
 ↪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(1,11), prediction)
    axes[i, 1].set_xticks(np.arange(1,11))
    axes[i, 1].set_title("Categorical distribution. Model prediction")

plt.show()
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x7f08013c6c80>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```



[120]:
```
!jupyter nbconvert --to PDF "/content/gdrive/My Drive/Colab Notebooks/
↪capstone_imperial.ipynb"
```

```
[NbConvertApp] Converting notebook /content/gdrive/My Drive/Colab
Notebooks/capstone_imperial.ipynb to PDF
[NbConvertApp] Support files will be in capstone_imperial_files/
[NbConvertApp] Making directory ./capstone_imperial_files
[NbConvertApp] Making directory ./capstone_imperial_files
[NbConvertApp] Making directory ./capstone_imperial_files
[NbConvertApp] Making directory ./capstone_imperial_files
[NbConvertApp] Making directory ./capstone_imperial_files
[NbConvertApp] Writing 84509 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: [u'xelatex', u'./notebook.tex',
```

```
'-quiet']
[NbConvertApp] Running bibtex 1 time: [u'bibtex', u'./notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 172756 bytes to /content/gdrive/My Drive/Colab
Notebooks/capstone_imperial.pdf
```

[120]: