AMERICAN SIGN LANGUAGE RECOGNITION

Setting up the environment and kaggle API

Importing tensorflow and checking tensorflow:

```
import tensorflow as tf

print(tf.__version__)
```

Installing kaggle so as to download the dataset using kaggle API:

```
!pip install -q kaggle
```

Setting up the kaggle.json authentication file enabling me to download the dataset:

```
!mkdir -p ~/.kaggle
from google.colab import files
files.upload()
!cp kaggle.json ~/.kaggle/
```

Downloading the grassknoted/asl-alphabet dataset

Downloading the dataset using the API:

```
!kaggle datasets download -d grassknoted/asl-alphabet
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.js
Downloading asl-alphabet.zip to /content
 99% 1.02G/1.03G [00:10<00:00, 161MB/s]
100% 1.03G/1.03G [00:10<00:00, 107MB/s]
```

Extracting the contents:

```
!unzip asl-alphabet.zip
```

```
Streaming output truncated to the last 5000 lines.
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing19.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing190.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1900.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1901.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1902.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1903.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1904.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1905.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1906.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1907.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1908.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1909.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing191.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1910.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1911.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1912.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1913.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1914.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1915.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1916.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1917.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1918.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1919.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing192.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1920.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1921.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1922.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1923.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1924.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1925.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1926.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1927.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1928.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1929.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing193.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1930.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1931.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1932.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1933.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1934.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1935.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1936.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1937.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1938.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1939.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing194.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1940.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1941.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1942.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1943.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1944.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1945.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1946.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1947.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1948.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1949.jpg
  inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing195.jpg
```

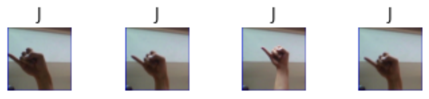
Looking at the dataset

Specifying train and test directories:

```
# Specifying the training and test directories

TRAINING_DIR = './asl_alphabet_train/asl_alphabet_train/'
TEST_DIR = './asl_alphabet_test/asl_alphabet_test/'
```

Looking at some random images from the dataset:

```
# Printing 5 random images from any training category or from a specified category
%matplotlib inline

import cv2
import os
import random
import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

number_of_rows = 4
number_of_columns = 4

categories = os.listdir(TRAINING_DIR)

random.seed(13)

category = categories[random.randint(1, 30)]
# category = 'A'

for i in range(number_of_columns):
  subplot = plt.subplot(number_of_rows, number_of_columns, i + 1)
  subplot.axis('Off')
  subplot.set_title(category)
  image_path = os.path.join(
      TRAINING_DIR,
      str(category),
      str(category) + str(random.randint(1, 1000)) + '.jpg'
  )
  image = mpimg.imread(image_path)
  plt.imshow(image)

plt.show()
```



Preparing the training set

Augmenting the data with brightness and zoom ranges:

```
# Preparing ImageDataGenerator object for training the model
from tensorflow.keras.preprocessing.image import ImageDataGenerator

IMAGE_SIZE = 200
BATCH_SIZE = 64

data_generator = ImageDataGenerator(
    samplewise_center=True,
    samplewise_std_normalization=True,
    brightness_range=[0.8, 1.0],
    zoom_range=[1.0, 1.2],
    validation_split=0.1
)

train_generator = data_generator.flow_from_directory(TRAINING_DIR, target_size=(IMAGE_SIZE, IMAGE_SIZE), shuffle=True, seed=13,
                                                     class_mode='categorical', batch_size=BATCH_SIZE, subset="training")

validation_generator = data_generator.flow_from_directory(TRAINING_DIR, target_size=(IMAGE_SIZE, IMAGE_SIZE), shuffle=True, seed=13,
                                                     class_mode='categorical', batch_size=BATCH_SIZE, subset="validation")
```

```
Found 78300 images belonging to 29 classes.
Found 8700 images belonging to 29 classes.
```

Preparing the model for training

Preparing Inception V3 Network for transfer learning:

```
# Loading inception v3 network for transfer learning
from tensorflow.keras import layers
from tensorflow.keras import Model

from tensorflow.keras.applications.inception_v3 import InceptionV3

WEIGHTS_FILE = './inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'

inception_v3_model = InceptionV3(
    input_shape = (IMAGE_SIZE, IMAGE_SIZE, 3),
    include_top = False,
    weights = 'imagenet'
)

# Enabling the top 2 inception blocks to train
for layer in inception_v3_model.layers[:249]:
    layer.trainable = False
for layer in inception_v3_model.layers[249:]:
    layer.trainable = True

# Checking model summary to pick a layer (if required)
inception_v3_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_orde
87910968/87910968 [==============================] - 0s 0us/step
Model: "inception_v3"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 200, 200, 3  0           []
                                )]

 conv2d (Conv2D)                (None, 99, 99, 32)   864         ['input_1[0][0]']

 batch_normalization (BatchNorm  (None, 99, 99, 32)  96          ['conv2d[0][0]']
 alization)

 activation (Activation)        (None, 99, 99, 32)   0           ['batch_normalization[0][0]']

 conv2d_1 (Conv2D)              (None, 97, 97, 32)   9216        ['activation[0][0]']

 batch_normalization_1 (BatchNo  (None, 97, 97, 32)  96          ['conv2d_1[0][0]']
 rmalization)

 activation_1 (Activation)      (None, 97, 97, 32)   0           ['batch_normalization_1[0][0]']
 conv2d_2 (Conv2D)              (None, 97, 97, 64)   18432       ['activation_1[0][0]']

 batch_normalization_2 (BatchNo  (None, 97, 97, 64)  192         ['conv2d_2[0][0]']
 rmalization)

 activation_2 (Activation)      (None, 97, 97, 64)   0           ['batch_normalization_2[0][0]']

 max_pooling2d (MaxPooling2D)   (None, 48, 48, 64)   0           ['activation_2[0][0]']

 conv2d_3 (Conv2D)              (None, 48, 48, 80)   5120        ['max_pooling2d[0][0]']

 batch_normalization_3 (BatchNo  (None, 48, 48, 80)  240         ['conv2d_3[0][0]']
 rmalization)

 activation_3 (Activation)      (None, 48, 48, 80)   0           ['batch_normalization_3[0][0]']

 conv2d_4 (Conv2D)              (None, 46, 46, 192)  138240      ['activation_3[0][0]']

 batch_normalization_4 (BatchNo  (None, 46, 46, 192)  576        ['conv2d_4[0][0]']
 rmalization)

 activation_4 (Activation)      (None, 46, 46, 192)  0           ['batch_normalization_4[0][0]']

 max_pooling2d_1 (MaxPooling2D)  (None, 22, 22, 192)  0          ['activation_4[0][0]']

 conv2d_8 (Conv2D)              (None, 22, 22, 64)   12288       ['max_pooling2d_1[0][0]']

 batch_normalization_8 (BatchNo  (None, 22, 22, 64)  192         ['conv2d_8[0][0]']
 rmalization)

 activation_8 (Activation)      (None, 22, 22, 64)   0           ['batch_normalization_8[0][0]']
```

Choosing the inception output layer:

```
# Choosing the output layer to be merged with our FC layers (if required)
inception_output_layer = inception_v3_model.get_layer('mixed7')
print('Inception model output shape:', inception_output_layer.output_shape)

# Not required --> inception_output = inception_output_layer.output
inception_output = inception_v3_model.output
```

```
Inception model output shape: (None, 10, 10, 768)
```

Adding our own set of fully connected layers at the end of Inception v3 network:

```
from tensorflow.keras.optimizers import RMSprop, Adam, SGD

x = layers.GlobalAveragePooling2D()(inception_output)
x = layers.Dense(1024, activation='relu')(x)
# Not required --> x = layers.Dropout(0.2)(x)
x = layers.Dense(29, activation='softmax')(x)

model = Model(inception_v3_model.input, x)

model.compile(
    optimizer=SGD(lr=0.0001, momentum=0.9),
    loss='categorical_crossentropy',
    metrics=['acc']
)
```

```
WARNING:absl:`lr` is deprecated, please use `learning_rate` instead, or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.S
```

Looking at the final model:

```
# Watch the new model summary
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 200, 200, 3  0           []
                                )]

 conv2d (Conv2D)                (None, 99, 99, 32)   864         ['input_1[0][0]']

 batch_normalization (BatchNorm  (None, 99, 99, 32)  96          ['conv2d[0][0]']
 alization)
 activation (Activation)        (None, 99, 99, 32)   0           ['batch_normalization[0][0]']

 conv2d_1 (Conv2D)              (None, 97, 97, 32)   9216        ['activation[0][0]']

 batch_normalization_1 (BatchNo  (None, 97, 97, 32)  96          ['conv2d_1[0][0]']
 rmalization)

 activation_1 (Activation)      (None, 97, 97, 32)   0           ['batch_normalization_1[0][0]']

 conv2d_2 (Conv2D)              (None, 97, 97, 64)   18432       ['activation_1[0][0]']

 batch_normalization_2 (BatchNo  (None, 97, 97, 64)  192         ['conv2d_2[0][0]']
 rmalization)

 activation_2 (Activation)      (None, 97, 97, 64)   0           ['batch_normalization_2[0][0]']

 max_pooling2d (MaxPooling2D)   (None, 48, 48, 64)   0           ['activation_2[0][0]']

 conv2d_3 (Conv2D)              (None, 48, 48, 80)   5120        ['max_pooling2d[0][0]']

 batch_normalization_3 (BatchNo  (None, 48, 48, 80)  240         ['conv2d_3[0][0]']
 rmalization)

 activation_3 (Activation)      (None, 48, 48, 80)   0           ['batch_normalization_3[0][0]']

 conv2d_4 (Conv2D)              (None, 46, 46, 192)  138240      ['activation_3[0][0]']

 batch_normalization_4 (BatchNo  (None, 46, 46, 192)  576        ['conv2d_4[0][0]']
 rmalization)

 activation_4 (Activation)      (None, 46, 46, 192)  0           ['batch_normalization_4[0][0]']

 max_pooling2d_1 (MaxPooling2D)  (None, 22, 22, 192)  0          ['activation_4[0][0]']

 conv2d_8 (Conv2D)              (None, 22, 22, 64)   12288       ['max_pooling2d_1[0][0]']

 batch_normalization_8 (BatchNo  (None, 22, 22, 64)  192         ['conv2d_8[0][0]']
 rmalization)

 activation_8 (Activation)      (None, 22, 22, 64)   0           ['batch_normalization_8[0][0]']

 conv2d_6 (Conv2D)              (None, 22, 22, 48)   9216        ['max_pooling2d_1[0][0]']

 conv2d_9 (Conv2D)              (None, 22, 22, 96)   55296       ['activation_8[0][0]']

 batch_normalization_6 (BatchNo  (None, 22, 22, 48)  144         ['conv2d_6[0][0]']
```

Setting up a callback funtion in order to stop training at a particular threshold:

```
# Creating a callback to stop model training after reaching a threshold accuracy

LOSS_THRESHOLD = 0.2
ACCURACY_THRESHOLD = 0.95

class ModelCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if logs.get('val_loss') <= LOSS_THRESHOLD and logs.get('val_acc') >= ACCURACY_THRESHOLD:
      print("\nReached", ACCURACY_THRESHOLD * 100, "accuracy, Stopping!")
      self.model.stop_training = True

callback = ModelCallback()
```

Training the model generated using Inception v3 and our own set of Fully Connected layers

Fitting the model to the training dataset:

```
history = model.fit_generator(
    train_generator,
    validation_data=validation_generator,
    steps_per_epoch=100,
    validation_steps=50,
    epochs=50,
    callbacks=[callback]
)
```

```
<ipython-input-14-bd102a896588>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please
  history = model.fit_generator(
Epoch 1/50
```

```
100/100 [==============================] - 135s 1s/step - loss: 1.0683 - acc: 0.7161 - val_loss: 0.9500 - val_acc: 0.6972
Epoch 2/50
100/100 [==============================] - ETA: 0s - loss: 0.1830 - acc: 0.9431
```

Plotting the results

Training Accuracy vs Validation Accuracy:

```
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.figure()


plt.show()
```

Training Loss vs Validation Loss

```
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend(loc=0)
plt.figure()
```

Saving the model

As we were satisfied with our results we save our model:

```
# Saving the model
MODEL_NAME = 'asl_alphabet.h5'
model.save(MODEL_NAME)
from google.colab import files

files.download('asl_alphabet.h5')
```

Testing our model

Plotting images along with their respective actual and predicted classes:

```
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt

classes = os.listdir(TRAINING_DIR)
classes.sort()

for i, test_image in enumerate(os.listdir(TEST_DIR)):
    image_location = TEST_DIR + test_image
    img = cv2.imread(image_location)
    img = cv2.resize(img, (IMAGE_SIZE, IMAGE_SIZE))
    plt.figure()
    plt.axis('Off')
    plt.imshow(img)
    img = np.array(img) / 255.
    img = img.reshape((1, IMAGE_SIZE, IMAGE_SIZE, 3))
    img = data_generator.standardize(img)
    prediction = np.array(model.predict(img))
    actual = test_image.split('_')[0]
    predicted = classes[prediction.argmax()]
    print('Actual class: {} \n Predicted class: {}'.format(actual, predicted))
    plt.show()
```

Calculating test accuracy:

```
test_images = os.listdir(TEST_DIR)
total_test_cases = len(test_images)
total_correctly_classified = 0
total_misclassified = 0
for i, test_image in enumerate(test_images):
    image_location = TEST_DIR + test_image
    img = cv2.imread(image_location)
    img = cv2.resize(img, (IMAGE_SIZE, IMAGE_SIZE))
    img = np.array(img) / 255.
    img = img.reshape((1, IMAGE_SIZE, IMAGE_SIZE, 3))
    img = data_generator.standardize(img)
    prediction = np.array(model.predict(img))
    actual = test_image.split('_')[0]
    predicted = classes[prediction.argmax()]
    print('Actual class: {} - Predicted class: {}'.format(
        actual, predicted), end=' ')
    if actual == predicted:
      print('PASS!')
      total_correctly_classified += 1
    else:
      print('FAIL!')
      total_misclassified += 1
print("=" * 20)
test_accuracy = (total_correctly_classified / total_test_cases) * 100
test_error_rate = (total_misclassified / total_test_cases) * 100

print('Test accuracy (%):', test_accuracy)
print('Test error rate (%):', test_error_rate)
print('Number of misclassified classes:', total_misclassified)
print('Number of correctly classified classes', total_correctly_classified)
```