+ Code + Text

RAM Disk

```
[1]  1  from google.colab import drive
     2  drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
  1  !rm -rf sample_data
  2  !rm -rf segmented
  3  !rm -rf segmented_224
  4  !rm -rf random.png random_data.png
```

```
  1  # !unzip '/content/drive/MyDrive/Colab Notebooks/Final_Year_Project/combined_segmented.zip'
  2  # !unzip '/content/drive/MyDrive/Colab Notebooks/Final_Year_Project/png.zip'
  3  !unzip '/content/drive/MyDrive/Colab Notebooks/Final_Year_Project/bw_segmented.zip'
```

```
extracting: bw_segmented/8/2.png
extracting: bw_segmented/8/20.png
extracting: bw_segmented/8/21.png
extracting: bw_segmented/8/22.png
extracting: bw_segmented/8/23.png
extracting: bw_segmented/8/24.png
extracting: bw_segmented/8/25.png
extracting: bw_segmented/8/26.png
extracting: bw_segmented/8/27.png
extracting: bw_segmented/8/28.png
extracting: bw_segmented/8/29.png
extracting: bw_segmented/8/3.png
extracting: bw_segmented/8/30.png
extracting: bw_segmented/8/31.png
extracting: bw_segmented/8/32.png
extracting: bw_segmented/8/33.png
extracting: bw_segmented/8/34.png
extracting: bw_segmented/8/4.png
extracting: bw_segmented/8/5.png
extracting: bw_segmented/8/6.png
extracting: bw_segmented/8/7.png
extracting: bw_segmented/8/8.png
extracting: bw_segmented/8/9.png
  creating: bw_segmented/9/
extracting: bw_segmented/9/0.png
extracting: bw_segmented/9/1.png
extracting: bw_segmented/9/10.png
extracting: bw_segmented/9/11.png
extracting: bw_segmented/9/12.png
extracting: bw_segmented/9/13.png
extracting: bw_segmented/9/14.png
extracting: bw_segmented/9/15.png
extracting: bw_segmented/9/16.png
extracting: bw_segmented/9/17.png
extracting: bw_segmented/9/18.png
extracting: bw_segmented/9/19.png
extracting: bw_segmented/9/2.png
extracting: bw_segmented/9/20.png
extracting: bw_segmented/9/21.png
extracting: bw_segmented/9/22.png
extracting: bw_segmented/9/23.png
extracting: bw_segmented/9/24.png
extracting: bw_segmented/9/25.png
extracting: bw_segmented/9/26.png
extracting: bw_segmented/9/27.png
extracting: bw_segmented/9/28.png
extracting: bw_segmented/9/29.png
extracting: bw_segmented/9/3.png
extracting: bw_segmented/9/30.png
extracting: bw_segmented/9/31.png
extracting: bw_segmented/9/32.png
extracting: bw_segmented/9/33.png
extracting: bw_segmented/9/4.png
extracting: bw_segmented/9/5.png
extracting: bw_segmented/9/6.png
extracting: bw_segmented/9/7.png
extracting: bw_segmented/9/8.png
extracting: bw_segmented/9/9.png
```

```
[4]  1  import os
     2
     3  root_dir = os.listdir()
     4  # os.rename('combined_segmented', 'segmented')
     5  # os.rename('png', 'segmented')
     6  os.rename('bw_segmented', 'segmented')
```

```
[5]  1  # Loading the data and generating labels
     2  import numpy as np
     3  import os
     4  from imageio import imread
     5  from keras.applications.vgg16 import preprocess_input
     6  from keras.utils import to_categorical
     7  from skimage.transform import resize
     8  import glob
     9  import cv2
    10  import random
    11
    12  IMG_SIZE = 32
    13  no_of_classes = 55
    14  image_height, image_width = IMG_SIZE, IMG_SIZE
    15  no_of_color_channels = 3
    16  root_dir = os.getcwd()
```

```
print(root_dir)

total_no_of_images = len(list(glob.glob(root_dir+"/segmented/[0-9]*/"+"/*.png", recursive=True)))
print(total_no_of_images)

count = 0
train_size = total_no_of_images - no_of_classes*3
data = np.empty((train_size, image_height, image_width, no_of_color_channels))
labels = np.empty(train_size, dtype=int)

test_count = 0
test_data = np.empty((no_of_classes*3, image_height, image_width, no_of_color_channels))
test_labels = np.empty(no_of_classes*3, dtype=int)

print(train_size)
print(no_of_classes*3)

r = random.randrange(total_no_of_images)
for i in range(no_of_classes):
    class_dir = root_dir + "/" + "segmented" + "/" + str(i)
    class_images = glob.glob(class_dir+'/*.png')
    size = len(class_images)
    if (i % 10 == 0): print(i)
    else: print(i, end=' ')
    for j in range(size):
        image_path = class_images[j]
        image =imread(image_path)
        # if len(image.shape) > 2 and image.shape[2] == 4:
        #     image = cv2.cvtColor(image, cv2.COLOR_BGRA2BGR)
        # image = resize(image, output_shape=(IMG_SIZE, IMG_SIZE))
        if count == r:
            cv2.imwrite(root_dir+'/random.png',image)
        image = preprocess_input(image)
        # image = resize(image, output_shape=(IMG_SIZE, IMG_SIZE))
        if j < size-3:
            data[count] = image
        else:
            test_data[test_count] = image
        if count == r:
            cv2.imwrite(root_dir+'/random_data.png',image)
        if j < size-3:
            labels[count] = i
            count+=1
        else:
            test_labels[test_count] = i
            test_count+=1
print('')
```

```
/content
58482
58317
165
0
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54
```

[6]

```
# np.save(root_dir+'/data.npy', data)
# np.save(root_dir+'/labels.npy', labels)
# data = np.load(root_dir+'/data.npy')
# labels = np.load(root_dir+'/labels.npy')
```

[7]

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

print(data[random.randrange(total_no_of_images)].shape)
fig = plt.figure(figsize=(5, 5))
fig.add_subplot(1, 2, 1)
plt.imshow(cv2.imread(root_dir+'/random.png'))
plt.axis('off')
fig.add_subplot(1, 2, 2)
plt.imshow(cv2.imread(root_dir+'/random_data.png'))
plt.axis('off')
print(data.min())
print(data.max())
print(labels)
print(len(test_data), test_data[random.randrange(no_of_classes)].shape)
print(test_labels)
```

```
(32, 32, 3)
-123.68000030517578
151.06100463867188
[ 0  0  0 ... 54 54 54]
165 (32, 32, 3)
[ 0  0  0  1  1  1  2  2  2  3  3  3  4  4  4  5  5  5  6  6  6  7  7  7
  8  8  8  9  9  9 10 10 10 11 11 11 12 12 12 13 13 13 14 14 14 15 15 15
 16 16 16 17 17 17 18 18 18 19 19 19 20 20 20 21 21 21 22 22 22 23 23 23
 24 24 24 25 25 25 26 26 26 27 27 27 28 28 28 29 29 29 30 30 30 31 31 31
 32 32 32 33 33 33 34 34 34 35 35 35 36 36 36 37 37 37 38 38 38 39 39 39
 40 40 40 41 41 41 42 42 42 43 43 43 44 44 44 45 45 45 46 46 46 47 47 47
 48 48 48 49 49 49 50 50 50 51 51 51 52 52 52 53 53 53 54 54 54]
```

[8]
```
# Importing VGG16
from keras.applications.vgg16 import VGG16

IMG_SHAPE = (IMG_SIZE, IMG_SIZE, 3)
base_model = VGG16(
    input_shape=IMG_SHAPE,
    include_top=False,
    weights='imagenet'
)
base_model.trainable = False
base_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 block1_conv1 (Conv2D)       (None, 32, 32, 64)        1792

 block1_conv2 (Conv2D)       (None, 32, 32, 64)        36928

 block1_pool (MaxPooling2D)  (None, 16, 16, 64)        0

 block2_conv1 (Conv2D)       (None, 16, 16, 128)       73856

 block2_conv2 (Conv2D)       (None, 16, 16, 128)       147584

 block2_pool (MaxPooling2D)  (None, 8, 8, 128)         0

 block3_conv1 (Conv2D)       (None, 8, 8, 256)         295168

 block3_conv2 (Conv2D)       (None, 8, 8, 256)         590080

 block3_conv3 (Conv2D)       (None, 8, 8, 256)         590080

 block3_pool (MaxPooling2D)  (None, 4, 4, 256)         0

 block4_conv1 (Conv2D)       (None, 4, 4, 512)         1180160

 block4_conv2 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_conv3 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 2, 2, 512)         0

 block5_conv1 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv2 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv3 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

[9]
```
# # Modifying the model
from keras import Model
from keras.layers import Dense, Flatten
from keras import layers, models


# character_output = Dense(no_of_classes, activation='softmax')
# character_output = character_output(base_model.layers[-2].output)
# character_input = base_model.input
# character_model = Model(inputs=character_input, outputs=character_output)
# for layer in character_model.layers[:-1]:
#   layer.trainable = False


flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(50, activation='relu')
dense_layer_2 = layers.Dense(20, activation='relu')
prediction_layer = layers.Dense(55, activation='softmax')

# character_model = models.Sequential([
#     base_model,
#     flatten_layer,
#     dense_layer_1,
#     dense_layer_2,
#     prediction_layer
# ])
# character_model.summary()

# flatten_layer = layers.Flatten()
# dense_layer_1 = layers.Dense(64, activation='relu')
# prediction_layer = layers.Dense(55, activation='softmax')

```

```
character_model = models.Sequential([
    base_model,
    flatten_layer,
    dense_layer_1,
    dense_layer_2,
    prediction_layer
])
character_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 1, 1, 512)         14714688

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 50)                25650

 dense_1 (Dense)             (None, 20)                1020

 dense_2 (Dense)             (None, 55)                1155

=================================================================
Total params: 14,742,513
Trainable params: 27,825
Non-trainable params: 14,714,688
_________________________________________________________________
```

[10]
```
# Compiling the model
character_model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

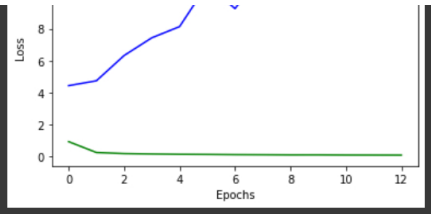
[11]
```
# EarlyStopping
from keras.callbacks import EarlyStopping

es = EarlyStopping(monitor='val_accuracy', mode='max', patience=5,  restore_best_weights=True)
```

[12]
```
# Training the model
history = character_model.fit(
    x=data,
    y=labels,
    epochs=200,
    verbose='auto',
    validation_split=0.1,
    shuffle=True,
    callbacks=[es]
)
```

```
Epoch 1/200
1641/1641 [==============================] - 25s 11ms/step - loss: 0.9031 - accuracy: 0.7915 - val_loss: 4.4255 - val_accuracy: 0.6778
Epoch 2/200
1641/1641 [==============================] - 17s 10ms/step - loss: 0.2207 - accuracy: 0.9461 - val_loss: 4.7307 - val_accuracy: 0.6756
Epoch 3/200
1641/1641 [==============================] - 17s 10ms/step - loss: 0.1540 - accuracy: 0.9625 - val_loss: 6.3199 - val_accuracy: 0.6763
Epoch 4/200
1641/1641 [==============================] - 17s 10ms/step - loss: 0.1286 - accuracy: 0.9686 - val_loss: 7.4350 - val_accuracy: 0.6806
Epoch 5/200
1641/1641 [==============================] - 17s 10ms/step - loss: 0.1080 - accuracy: 0.9735 - val_loss: 8.1363 - val_accuracy: 0.6773
Epoch 6/200
1641/1641 [==============================] - 17s 10ms/step - loss: 0.0998 - accuracy: 0.9753 - val_loss: 10.7778 - val_accuracy: 0.6809
Epoch 7/200
1641/1641 [==============================] - 17s 10ms/step - loss: 0.0870 - accuracy: 0.9788 - val_loss: 9.2709 - val_accuracy: 0.6783
Epoch 8/200
1641/1641 [==============================] - 19s 12ms/step - loss: 0.0807 - accuracy: 0.9802 - val_loss: 11.5314 - val_accuracy: 0.6812
Epoch 9/200
1641/1641 [==============================] - 17s 10ms/step - loss: 0.0725 - accuracy: 0.9815 - val_loss: 12.1827 - val_accuracy: 0.6800
Epoch 10/200
1641/1641 [==============================] - 17s 10ms/step - loss: 0.0738 - accuracy: 0.9812 - val_loss: 12.6270 - val_accuracy: 0.6766
Epoch 11/200
1641/1641 [==============================] - 17s 11ms/step - loss: 0.0660 - accuracy: 0.9829 - val_loss: 11.2724 - val_accuracy: 0.6794
Epoch 12/200
1641/1641 [==============================] - 18s 11ms/step - loss: 0.0634 - accuracy: 0.9844 - val_loss: 11.3210 - val_accuracy: 0.6754
Epoch 13/200
1641/1641 [==============================] - 17s 11ms/step - loss: 0.0604 - accuracy: 0.9850 - val_loss: 13.6748 - val_accuracy: 0.6787
```

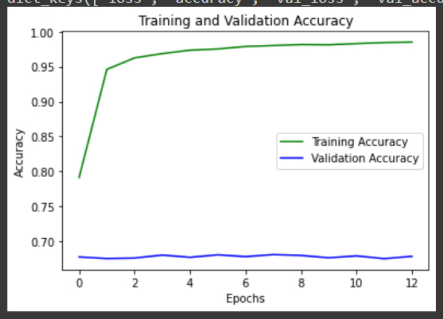
[15]
```
print(history.history.keys())
loss_train = history.history['loss']
loss_val = history.history['val_loss']
epochs = range(13)
plt.plot(epochs, loss_train, 'g', label='Training Loss')
plt.plot(epochs, loss_val, 'b', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

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



[14]
```
print(history.history.keys())
loss_train = history.history['accuracy']
loss_val = history.history['val_accuracy']
epochs = range(13)
plt.plot(epochs, loss_train, 'g', label='Training Accuracy')
plt.plot(epochs, loss_val, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

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



[16]
```
character_model.save('character_model_vgg16.h5')
```

[17]
```
predictions = character_model.predict(test_data)
print('Shape: {}'.format(predictions.shape))
```

```
6/6 [==============================] - 0s 16ms/step
Shape: (165, 55)
```

[19]
```
for i in range(no_of_classes):
  output_neuron = np.argmax(predictions[i])
  print('Most active neuron: {} ({:.2f}%)'.format(
      output_neuron,
      100 * predictions[i][output_neuron]
  ))
```

```
Most active neuron: 47 (56.50%)
Most active neuron: 0 (99.95%)
Most active neuron: 0 (95.06%)
Most active neuron: 1 (100.00%)
Most active neuron: 1 (99.98%)
Most active neuron: 1 (99.99%)
Most active neuron: 2 (99.70%)
Most active neuron: 2 (97.14%)
Most active neuron: 2 (99.72%)
Most active neuron: 3 (99.99%)
Most active neuron: 3 (99.63%)
Most active neuron: 3 (100.00%)
Most active neuron: 4 (94.47%)
Most active neuron: 4 (97.94%)
Most active neuron: 25 (45.02%)
Most active neuron: 5 (98.88%)
Most active neuron: 5 (46.30%)
Most active neuron: 5 (97.46%)
Most active neuron: 6 (96.72%)
Most active neuron: 6 (92.83%)
Most active neuron: 6 (68.48%)
Most active neuron: 7 (99.92%)
Most active neuron: 7 (100.00%)
Most active neuron: 7 (99.68%)
Most active neuron: 6 (72.34%)
Most active neuron: 9 (90.80%)
Most active neuron: 6 (76.71%)
Most active neuron: 1 (75.63%)
Most active neuron: 44 (76.63%)
Most active neuron: 9 (99.69%)
Most active neuron: 10 (99.87%)
Most active neuron: 10 (99.93%)
Most active neuron: 10 (100.00%)
Most active neuron: 11 (99.91%)
Most active neuron: 11 (99.79%)
Most active neuron: 11 (100.00%)
Most active neuron: 12 (100.00%)
Most active neuron: 12 (100.00%)
Most active neuron: 12 (99.43%)
Most active neuron: 13 (100.00%)
Most active neuron: 13 (100.00%)
Most active neuron: 13 (100.00%)
Most active neuron: 14 (100.00%)
Most active neuron: 14 (100.00%)
```

```
Most active neuron: 14 (99.99%)
Most active neuron: 15 (99.97%)
Most active neuron: 15 (99.95%)
Most active neuron: 15 (100.00%)
Most active neuron: 16 (100.00%)
Most active neuron: 16 (100.00%)
Most active neuron: 16 (100.00%)
Most active neuron: 17 (100.00%)
Most active neuron: 17 (100.00%)
Most active neuron: 17 (100.00%)
Most active neuron: 18 (87.78%)
```

```python
top1_correct = 0
top1_incorrect = 0
top5_correct = 0
top5_incorrect = 0
for i in range(no_of_classes*3):
  s = set()
  sorted_predictions = np.sort(predictions[i])[::-1]
  for j in range(5):
    x = list(np.where(predictions[i]==sorted_predictions[j]))[0][0]
    s.add(x)
  # print(list(s))
  if test_labels[i] in s: top5_correct+=1
  else: top5_incorrect+=1
  if test_labels[i] == np.argmax(predictions[i]): top1_correct+=1
  else: top1_incorrect+=1

print('Top-1 Accuracy: ', (top1_correct*100)/(top1_correct+top1_incorrect))
print('Top-5 Accuracy: ', (top5_correct*100)/(top5_correct+top5_incorrect))
```

```
Top-1 Accuracy:  80.0
Top-5 Accuracy:  85.45454545454545
```

```
[28]  1
```