Waste Classification system using Transfer Learning and Image Segmentation Configuration Manual

MSc Research Project
Data Analytics

Kalpesh Dhande
Student ID: 20185821

School of Computing
National College of Ireland

Supervisor: Mr.Jorge Basilio

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Kalpesh Dhande
Student ID:	20185821
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Mr.Jorge Basilio
Submission Due Date:	15/08/2022
Project Title:	Waste Classification system using Transfer Learning and Image Segmentation Configuration Manual
Word Count:	917
Page Count:	14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Kalpesh Dhande
Date:	16th September 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Waste Classification system using Transfer Learning and Image Segmentation Configuration Manual

Kalpesh Dhande
20185821

1 Overview

This is the setup guide for the research project "Waste classification using Transfer learning and Image Segmentation." I've provided a step-by-step instruction for running the code in this. In this, I've also mentioned the system configuration and setup that I used to run the code.

2 System Configuration

2.1 Hardware Requirement

The following system configuration is used for code development and code execution:

- Operating System: macOS Monterey
- Macbook Air M1
- Ram: 8GB
- HDD: 256GB SSD

2.2 Software Requirement

The following system setup is used for code development and code execution:

- Python Version 3.7
- Google Colab
- Overleaf

3 Steps To Environment Setup

In this part, I've described how to get started with Google Collaborate. To begin, go to the official Google Colab website, as seen in figure 1. The GPU must then be enabled, as shown in figure 2

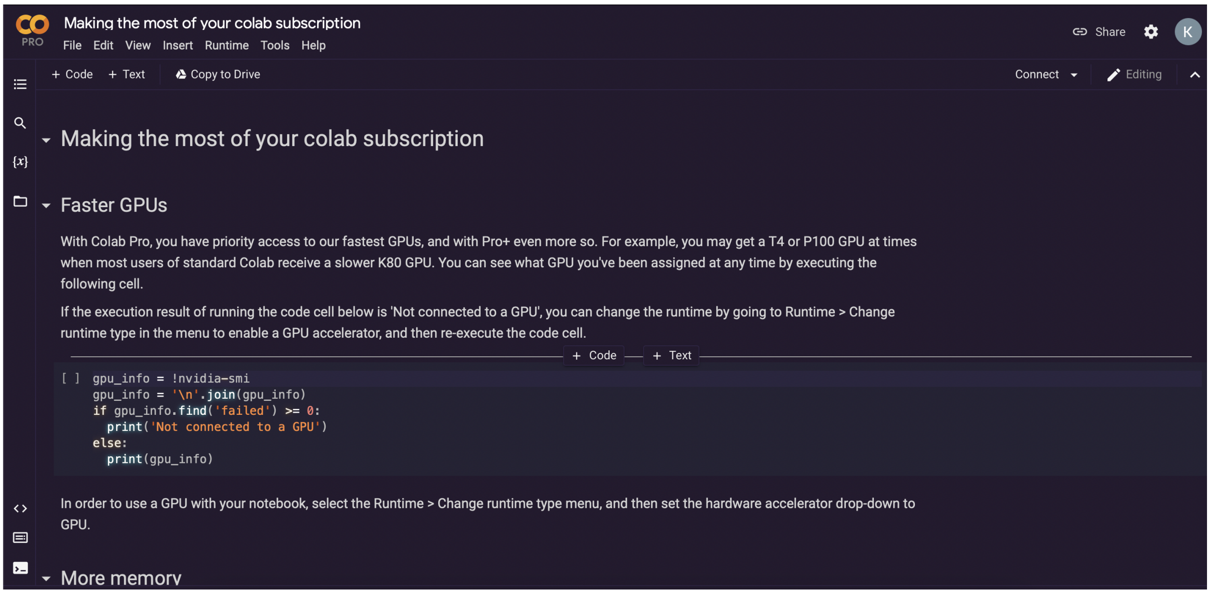


Figure 1: Offical Google Colab page

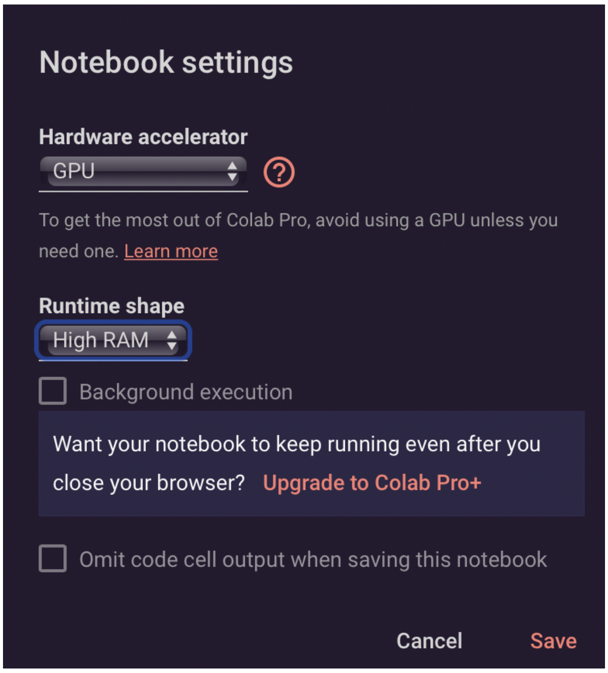


Figure 2: Enable GPU

4 Data Collection

I downloaded the dataset from the kaggle website. The collection is called "Waste Classification Data," and it is displayed in figure 3.

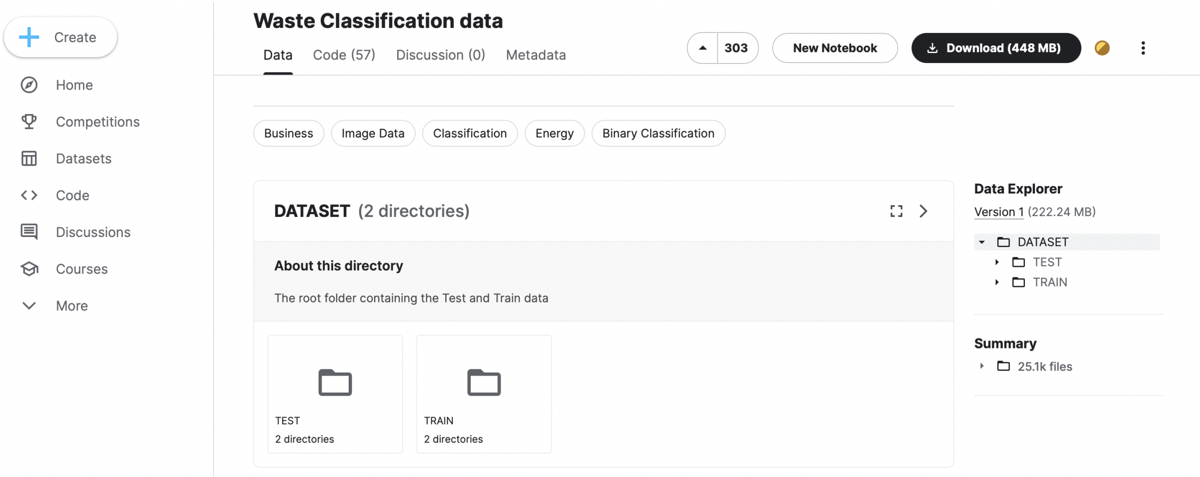


Figure 3: Waste Classification Dataset

5 Classification Model using Transfer Learning

5.1 Importing Libraries

While carrying out this experiment, I included a number of libraries, which I have listed below.

- Tensorflow
- Keras
- Numpy
- Pandas
- Sklearn
- Matplotlib
- Seaborn
- glob

```
import glob
import tensorflow as tensorflow
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tqdm import tqdm
from sklearn.metrics import confusion_matrix
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation, MaxPool2D, Conv2DTranspose, Concatenate, Input
from tensorflow.keras.models import Model, load_model
from tensorflow import keras
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer, Dense, Flatten, BatchNormalization, Dropout, Activation
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.applications import DenseNet121
from keras.utils.vis_utils import plot_model
from pylab import *
from sklearn.metrics import classification_report, confusion_matrix
from keras.applications import densenet
from sklearn.utils import class_weight
```

Figure 4: Imported Libraries

5.2 Uploading data to drive and Connecting to google Drive

The downloaded dataset must be uploaded to the same Google Drive account as the Google colab. After uploading the data, the Google colab must be linked to the Google Drive, as illustrated in Figure 5.

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

Figure 5: Connecting google drive to google colab

5.3 Reading , pre-processing and splitting the data

5.3.1 VGG16

In this step, I first augmented the dataset with the configuration shown in figure 6 and also split the training dataset in training and validation in an 8:2 ratio while augmenting using the Imagedatagenerator, and then I imported the dataset using the glob library and called the augmentation part to perform the augmentation as shown in figure 7.

```
train_datagenerator = ImageDataGenerator(rescale = 1.0 / 255.0,
                                         horizontal_flip = True,
                                         vertical_flip = True,
                                         zoom_range = 0.6,
                                         rotation_range = 12,
                                         validation_split = 0.2,)

valid_datagengenerator = ImageDataGenerator(rescale = 1.0 / 255.0,
                                      validation_split = 0.2)

test_datagengenerator  = ImageDataGenerator(rescale = 1.0 / 255.0)
```

Figure 6: Data Augmentation And splitting the dataset

```
▾ Importing Data

train  = train_datagenerator.flow_from_directory(directory = '/content/drive/MyDrive/WasteData/TRAIN',
                                                  target_size = (224,224),
                                                  shuffle = True,
                                                  class_mode = 'binary',
                                                  batch_size = 64,
                                                  subset = 'training')

Found 18052 images belonging to 2 classes.

[ ] val = valid_datagengenerator.flow_from_directory(directory = '/content/drive/MyDrive/WasteData/TRAIN',
                                                  target_size = (224,224),
                                                  shuffle = True,
                                                  class_mode = 'binary',
                                                  batch_size = 64,
                                                  subset = 'validation')

Found 4512 images belonging to 2 classes.

[ ] test = test_datagengenerator.flow_from_directory(directory = '/content/drive/MyDrive/WasteData/TEST',
                                                  target_size = (224,224),
                                                  class_mode = 'binary',
                                                  batch_size = 32,
                                                  shuffle = False)

Found 2513 images belonging to 2 classes.
```

Figure 7: Reading Dataset

5.3.2 DenseNet121

In this phase, I used the glob library to import the dataset, divided the dataset in an 8:2 ratio as shown in figure 8

```
▾ DenseNet 121

[ ] train_datagen_dense = ImageDataGenerator(validation_split = 0.2)

    valid_datagen_dense = ImageDataGenerator(validation_split = 0.2)

    test_datagen_dense  = ImageDataGenerator()

train_dense  = train_datagen_dense.flow_from_directory(directory = '/content/drive/MyDrive/WasteData/TRAIN',
                                                  target_size = (224,224),
                                                  shuffle = True,
                                                  class_mode = 'binary',
                                                  batch_size = 64,
                                                  subset = 'training')

Found 18052 images belonging to 2 classes.

[ ] val_dense = valid_datagen_dense.flow_from_directory(directory = '/content/drive/MyDrive/WasteData/TRAIN',
                                                  target_size = (224,224),
                                                  shuffle = True,
                                                  class_mode = 'binary',
                                                  batch_size = 64,
                                                  subset = 'validation')

Found 4512 images belonging to 2 classes.

[ ] test_dense = test_datagen_dense.flow_from_directory(directory = '/content/drive/MyDrive/WasteData/TEST',
                                                  target_size = (224,224),
                                                  class_mode = 'binary',
                                                  batch_size = 32,
                                                  shuffle = False)

Found 2513 images belonging to 2 classes.
```

Figure 8: Reading Dataset for densenet

5.4 Model Building, Training, Testing

5.4.1 VGG16

First, as shown in Figure 9a, import the VGG 16 model with the shape as (224,224,3),include_top as false and weights as imagenet and then freeze the layers as shown in Figure 9b.

```
[ ] #Model Definition
    vgg16_model = VGG16(input_shape=(224,224,3),
                        include_top=False,
                        weights="imagenet")
```

(a) Defining Model

```
#Freeze Layers
for layer in vgg16_model.layers:
    layer.trainable=False
```

(b) Freezing Layers

Figure 9: Pre trained VGG16

Figure 10a depicts several additional architectural layers. Finally, compile the model by setting the optimiser to adam with a learning rate of 0.001 and loss to binary crossentropy, as illustrated in Figure 10b.

```
# Defining Layers for our transfer learning model

tl_vgg16_model=Sequential()
tl_vgg16_model.add(vgg16_model)
tl_vgg16_model.add(Dropout(0.3))
tl_vgg16_model.add(Flatten())
tl_vgg16_model.add(BatchNormalization())
tl_vgg16_model.add(Dense(1024,kernel_initializer='he_uniform'))
tl_vgg16_model.add(BatchNormalization())
tl_vgg16_model.add(Activation('relu'))
tl_vgg16_model.add(Dropout(0.2))
tl_vgg16_model.add(Dense(512,kernel_initializer='he_uniform'))
tl_vgg16_model.add(BatchNormalization())
tl_vgg16_model.add(Activation('relu'))
tl_vgg16_model.add(Dropout(0.1))
tl_vgg16_model.add(Dense(1,activation='sigmoid'))
```

(a) Added layers to VGG16

```
#Model Compilation using Adam Optiminzer with learing rate 0.001
adam_opt    = tensorflow.keras.optimizers.Adam(lr=0.001)

tl_vgg16_model.compile(optimizer=adam_opt,
                       loss='binary_crossentropy',
                       metrics=[tensorflow.keras.metrics.AUC(name = 'auc')],
                      )
```

(b) Compiling VGG16 Model

Figure 10: Pre trained VGG16

As demonstrated in figure 11a, callbacks must be defined for early stopping in the event of model overfitting or the model is not improving, as well as checkpoints to store the best weights, and the model must be trained for 20 epochs, as shown in figure 11b.

```
#Callbacks
filepath = '/content/drive/MyDrive/VGG16Class/best_weights.hdf5'

earlystopping = EarlyStopping(monitor = 'val_auc',
                              mode = 'max' ,
                              patience = 5,
                              verbose = 1)

checkpoint    = ModelCheckpoint(filepath,
                                monitor = 'val_auc',
                                mode='max',
                                save_best_only=True,
                                verbose = 1)


callback_list = [earlystopping, checkpoint]
```

(a) Callbacks for VGG16

```
#Model Fitting
tl_vgg16_model_fit=tl_vgg16_model.fit(train,
                                      steps_per_epoch = len(train),
                                      validation_data=val,
                                      validation_steps = len(val),
                                      epochs = 20,
                                      callbacks = callback_list,
                                      verbose = 1)
```

(b) Training VGG16 Model

Figure 11: Callbacks and Training of VGG16

The developed model is next evaluated, as shown in Figure 12. First, calculated the auc and loss for the test data using evaluate function , then generate the confusion matrix using confusion matrix function of sklearn library, then create the classification report to assess the accuracy precision and recall, and lastly test on a single picture from the test dataset.

```
[ ] loss, AUC = tl_vgg16_model.evaluate(test)
    print("Test dataset AUC: %f and Loss: %f" % (AUC,loss))

[ ] tl_vgg_model_pred = tl_vgg16_model.predict(test)
    tl_vgg_model_pred_list = [int(i > .5) for i in tl_vgg_model_pred]

[ ] import seaborn as sns
    df_cm = pd.DataFrame(
            confusion_matrix(tl_vgg_model_pred_list,test_y), index=['Organic','Recyclable'], columns=['Organic','Recyclable'],
        )
    sns.heatmap(df_cm, annot=True, fmt='g', cmap='Blues')
    plt.ylabel('Predicted label')
    plt.xlabel('True label')

[ ] print(classification_report(tl_vgg_model_pred_list,test_y ,target_names=['Organic', 'Recyclable']))

    test_image = load_img('/content/drive/MyDrive/Test Image /Bananaimage.jpeg', target_size=(224,224))
    test_image = img_to_array(test_image)
    test_image = test_image / 255
    imshow(test_image)
    plt.axis('off')
    test_image = np.expand_dims(test_image,axis=0)
    prediction = tl_vgg16_model.predict(test_image)

    if prediction[0][0] > 0.5:
        print("Object in the image is Recycalable waste")
    else:
        print("Object in the image is Organic waste")
```

Figure 12: VGG16 Evaluation

5.5 DenseNet121

Import the DenseNet121 model with the shape (224,224,3), include top as false, pooling as average and weights as imagenet, as shown in figure 13a, and then freeze the layers as shown in figure 13b.

```
densenet121_pretrained = densenet.DenseNet121(include_top=False,weights='imagenet',
                                              input_shape=(224,224,3),pooling='avg')
```

(a) Defining Model DenseNet121

```
[ ] for layer in densenet121_pretrained.layers:
        layer.trainable=False
```

(b) Freezing Layers DenseNet121

Figure 13: Pre trained DenseNet121

Figure 14a shows various additional architectural layers. Finally, compile the model by setting the optimiser to adam and the loss to binary crossentropy, as shown in figure 14b.

```
tl_densenet121_model=Sequential()
tl_densenet121_model.add(densenet121_pretrained)
tl_densenet121_model.add(Flatten())
tl_densenet121_model.add(BatchNormalization())
tl_densenet121_model.add(Dense(128,kernel_initializer='he_uniform'))
tl_densenet121_model.add(Activation('relu'))
tl_densenet121_model.add(BatchNormalization())
tl_densenet121_model.add(Dense(64,kernel_initializer='he_uniform'))
tl_densenet121_model.add(Activation('relu'))
tl_densenet121_model.add(Dense(1,activation='sigmoid'))
```

(a) Added layers to DenseNet121

```
# Compiling the model
tl_densenet121_model.compile(optimizer='adam',
                             loss='binary_crossentropy',
                             metrics=[tensorflow.keras.metrics.AUC(name = 'auc')])
```

(b) Compiling DenseNet121 Model

Figure 14: modified architecture of model and compiling the DenseNet121 model

As demonstrated in figure 15a, callbacks must be defined for early stopping in the event of model overfitting or the model is not improving, as well as checkpoints to store the best weights, and the model must be trained for 20 epochs, as shown in figure 15b.

```
earlystopping = EarlyStopping(monitor = 'val_auc',
                              mode = 'max' ,
                              patience = 5,
                              verbose = 1)

checkpoint    = ModelCheckpoint(filepath,
                                monitor = 'val_auc',
                                mode='max',
                                save_best_only=True,
                                verbose = 1)


callback_list = [earlystopping, checkpoint]
```

(a) Callbacks for DenseNet121

```
tl_densenet121_model_fit = tl_densenet121_model.fit(train_dense,
                                        steps_per_epoch = len(train_dense),
                                        validation_data=val_dense,
                                        validation_steps = len(val_dense),
                                        epochs = 20,
                                        callbacks = callback_list,
                                        verbose = 1)
```

(b) Training DenseNet121 Model

Figure 15: Callbacks and Training of DenseNet121

The constructed model is next evaluated, as illustrated in Figure 16. To begin, compute the auc and loss for the test data using the evaluate function, then produce the confusion matrix using the confusion matrix function of the sklearn package, then create the classification report to analyze the accuracy precision and recall, and finally test on a single image from the test dataset.

```
[ ] loss, AUC = tl_densenet121_model.evaluate(test_dense)
    print("Test dataset AUC: %f and Loss: %f" % (AUC,loss))

    tl_densenet121_model_pred = tl_densenet121_model.predict(test_dense)
    tl_densenet121_model_pred_list = [int(i > .5) for i in tl_densenet121_model_pred]

[ ] import seaborn as sns
    df_cm = pd.DataFrame(
            confusion_matrix(tl_densenet121_model_pred_list,test_dense.classes), index=['Organic','Recyclable'], columns=['Organic','Recyclable'],
        )
    sns.heatmap(df_cm, annot=True, fmt='g', cmap='Blues')
    plt.ylabel('Predicted label')
    plt.xlabel('True label')

[ ] print(classification_report(tl_densenet121_model_pred_list,test_dense.classes, target_names=['Organic', 'Recyclable']))

    test_image = load_img('/content/drive/MyDrive/Test Image /Bananaimage.jpeg', target_size=(224,224))
    test_image = img_to_array(test_image)
    test_image = test_image / 255
    imshow(test_image)
    plt.axis('off')
    test_image = np.expand_dims(test_image,axis=0)
    prediction = tl_densenet121_model.predict(test_image)

    if prediction[0][0] > 0.5:
        print("Object in the image is Recycalable waste")
    else:
        print("Object in the image is Organic waste")
```

Figure 16: DenseNet121 Evaluation

6 Image Segmentation

6.1 Data Collection and Anotation

First, I extracted several photos of banana from the original collection. Then I used the website APEER.com to annotate these banana photos. It is a free and open website that anybody can use to annotate their dataset, which I have then separated into training, testing, and validation data for both photos and annotated images that are binary mask.

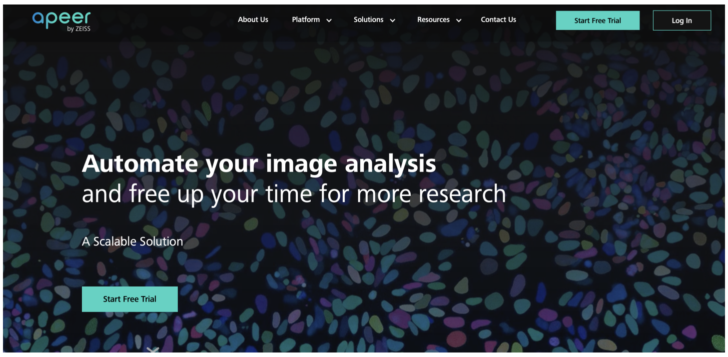


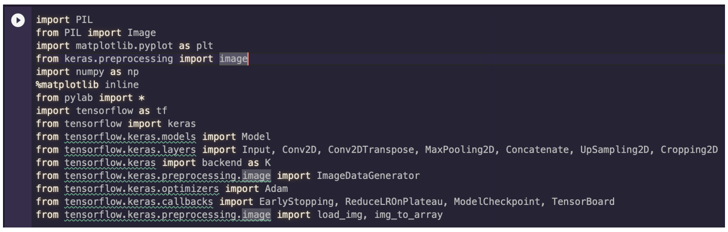
Figure 17: APEER for Annotating image

6.2 Importing Libraries

I used a variety of libraries throughout this experiment, which I've mentioned here.

- Tensorflow

- Keras
- Pylab
- Numpy
- Matplotlib
- glob

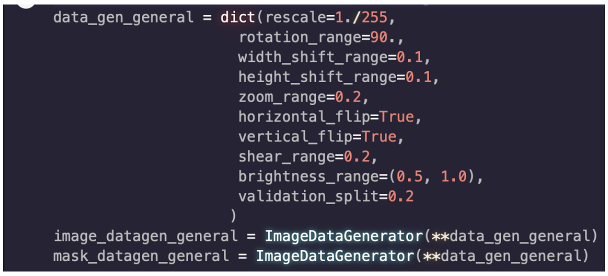


```
import PIL
from PIL import Image
import matplotlib.pyplot as plt
from keras.preprocessing import image
import numpy as np
%matplotlib inline
from pylab import *
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, Conv2DTranspose, MaxPooling2D, Concatenate, UpSampling2D, Cropping2D
from tensorflow.keras import backend as K
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint, TensorBoard
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

Figure 18: Importing Libraries for Image Segmentation

6.3 Loading Data and Data pre-processing

First, define augmentation for the data as shown in figure 19a, then load the dataset with flow and send this data for augmentation as shown in figure 19b.



```
data_gen_general = dict(rescale=1./255,
                        rotation_range=90.,
                        width_shift_range=0.1,
                        height_shift_range=0.1,
                        zoom_range=0.2,
                        horizontal_flip=True,
                        vertical_flip=True,
                        shear_range=0.2,
                        brightness_range=(0.5, 1.0),
                        validation_split=0.2
                       )
image_datagen_general = ImageDataGenerator(**data_gen_general)
mask_datagen_general = ImageDataGenerator(**data_gen_general)
```

(a) Augmenting data for Image Segmentation

```
img_size = 256

image_arguments = dict(seed=seed,
    batch_size=32,
    shuffle=True,
    class_mode=None,
    target_size=(img_size, img_size),
    color_mode='rgb')

mask_arguments = dict(seed=seed,
    batch_size=32,
    class_mode=None,
    shuffle=True,
    target_size=(img_size, img_size),
    color_mode='grayscale')

dir = '/content/drive/MyDrive/ImageSegmentation/banana/'
images = 'images'
masks = 'masks'

train_generator = zip(image_datagen_general.flow_from_directory(**image_arguments, directory=dir+'training_'+images),
                      mask_datagen_general.flow_from_directory(**mask_arguments, directory=dir+'training_'+masks))

validation_generator = zip(image_datagen_general.flow_from_directory(**image_arguments, directory=dir+'validation_'+images),
                      mask_datagen_general.flow_from_directory(**mask_arguments, directory=dir+'validation_'+masks))
```

(b) Loading data for image segmentation

Figure 19: Data Preparation for UNET

6.4 Model Building, Training, and evaluation

Begin by building the UNET class, from which three functions were created: a conv block, a decoder block, and create model function. The first conv block is used to create the Unet architecture's encoder and bridge, while the second decoder block is used to create the UNET's decoders. In generate models, I built an array of filters for which the encoder will be created for each filter and the final encoder will work as a bridge, and then filter decoders will be designed in reverse order. Figure 21 depicts the model's construction.

```python
class UNet:

    def __init__(self, img_dim=None):
        self.img_shape = (img_dim, img_dim, 3)

    def conv_block(self, x, filters, pool=True):
        conv = Conv2D(filters=filters, kernel_size=(3,3), padding='same', activation='relu', kernel_initializer='he_uniform')(x)
        res = Conv2D(filters=filters, kernel_size=(3,3), padding='same', activation='relu', kernel_initializer='he_uniform')(conv)
        if pool:
            out = MaxPooling2D()(res)
            return out, res
        else:
            return res

    def decoder_block(self, x, res, filters):
        x = UpSampling2D()(x)
        conv = Conv2D(filters=filters, kernel_size=(2,2), padding='same')(x)
        cropping_size = res.get_shape().as_list()[1] - conv.get_shape().as_list()[1]
        crop = Cropping2D(cropping=cropping_size//2)(res)
        merged = Concatenate()([conv, crop])
        conv_op_1 = Conv2D(filters=filters, kernel_size=(3,3), padding='same', activation='relu')(merged)
        out = Conv2D(filters=filters, kernel_size=(3,3), padding='same', activation='relu')(conv_op_1)
        return out

    def create_model(self):
```

(a) UNET Model Building 1

```python
def create_model(self):
    img = Input(shape=self.img_shape)
    filters = [64, 128, 256, 512, 1024]
    x = img
    residuals = []
    pool = True
    """ Encoder """
    for fil in filters:
        if fil == 1024:
            pool = False
        if pool == True:
            x, res = self.conv_block(x, fil, pool=pool)
            residuals.append(res)
        else:
            x = self.conv_block(x, fil, pool=pool)
    counter = -1
    """ Decoder """
    for fil in reversed(filters[:-1]):
        x = self.decoder_block(x, residuals[counter], fil)
        counter = counter - 1
    """ Output """
    out = Conv2D(filters=1, kernel_size=(1,1), padding='same', activation='sigmoid')(x)
    model = Model(inputs=img, outputs=out)
    return model
```

(b) UNET Model Building 2

Figure 20: UNET Model Building

The model was then compiled using the optimizer as Adam, loss as binary crossentropy, and metrics as accuracy, as shown in figure 21a, and callbacks and early stopping were defined as shown in figure 21b.

```
model.compile(optimizer=Adam(1e-4), loss='binary_crossentropy',
              metrics=['accuracy'])
```

(a) UNET Model Compiling

```
earlystopping = EarlyStopping(monitor='val_loss',
                              patience=8,
                              verbose=1,
                              min_delta=1e-4)

checkpoint    = ModelCheckpoint(monitor='val_loss',
                                filepath=filepath,
                                save_best_only=True,
                                save_weights_only=True)


callback_list = [earlystopping, checkpoint]
```

(b) UNET Callbacks

Figure 21: UNET Model Compilation and Defining Callbacks

The model was then trained on the training and validation datasets using images and masks for 10 epochs as shown in figure 22a, and it was evaluated by sending the test data to modelevaluate as shown in figure 22b.

```
unet_history = model.fit(train_generator,
                         steps_per_epoch=135,
                         epochs=10,
                         validation_data=validation_generator,
                         validation_steps=16,
                         callbacks=callback_list,
                         workers=2)
```

(a) UNET Model Training

```
loss, accuracy = model.evaluate(test_generator, steps=15)
print("Test dataset Loss: %f and accuracy: %f" % (loss,accuracy))
```

(b) UNET Model Testing

Figure 22: UNET Model Training and Testing

Finally, as shown in Figure 23, I tested the model by constructing the predicted mask using the predict function.

```
test_image = load_img('/content/drive/MyDrive/TestImageForClassificationAndSegmentation/Bananaimage.jpeg', target_size=(256,256))
test_image = img_to_array(test_image)
test_image = test_image / 255
mask_test = model.predict(test_image[np.newaxis, :])
plt.subplot(1, 2, 1) # row 1, col 2 index 1
plt.imshow(test_image)
plt.title("Original Image")

plt.subplot(1, 2, 2)
plt.imshow(mask_test[0,:,:,0])
plt.title("Predicted Mask")
```

Figure 23: UNET Prediction

7 Other Software

Overleaf, a web-based application, was utilized for report writing and configuration manual writing.

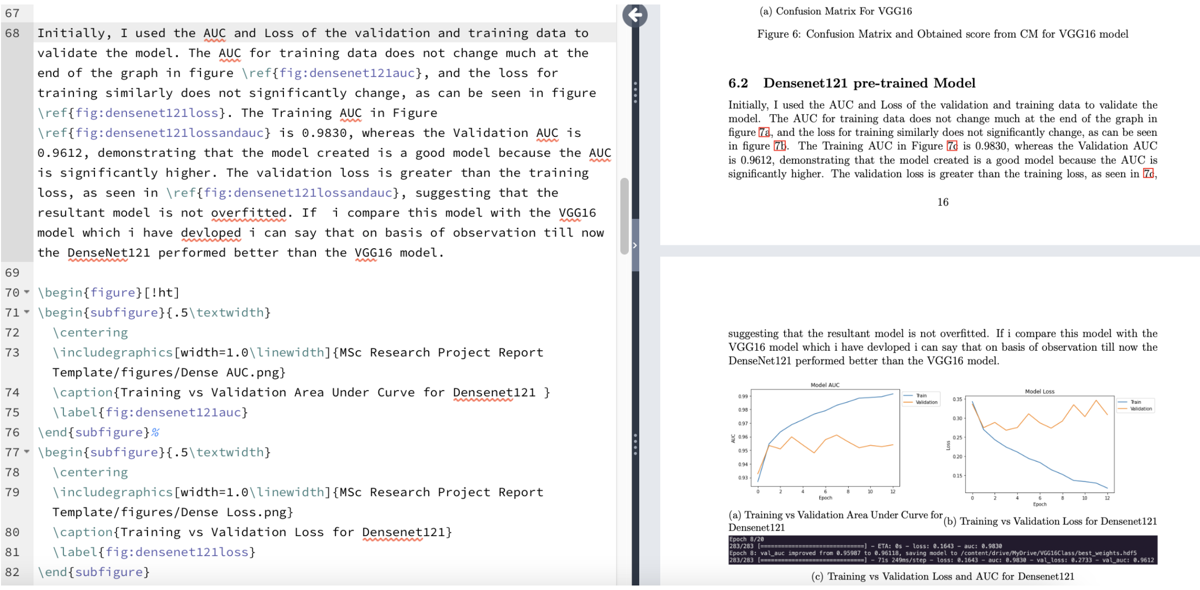


Figure 24: Overleaf