Configuration Manual

MSc Research Project
Programme Name

Ridima Tambde
Student ID: X22209557

School of Computing
National College of Ireland

Supervisor: Abid Yaqoob

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ridima Chetan Tambde
Student ID:	X22209557
Programme:	MSc in Data Analytics **Year:** 2023 - 2024
Module:	MSc Research Project
Lecturer:	Abid Yaqoob
Submission Due Date:	12/08/2024
Project Title:	Enhancing PCOS Detection with SRGAN-Generated Synthetic Images and CNN Models
Word Count:	1049 **Page Count:** 19

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: Ridima Chetan Tambde

Date: 12th August 2024

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):	

Configuration Manual

Ridima Chetan Tambde
Student ID: x22209557

Introduction

This report provides a detailed step-by-step guide to the configuration and implementation of the project.

1 Environment

This section details about the environment setup and the system requirements necessary for implementing the project, in this case, the code is executed on Google Colab.

1.1 System configuration

Requirement	Specification
Programming Language	Python Version 3
Tools	Google Collaboratory, WORD
Google Drive	Access to Google Drive for data
Operating System	Windows 11

Table 1: Software Configuration

Requirement	Specification
Processor	11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz
RAM	16GB
Computational resources	Google Colab's TPU v2, 300 GB RAM

Table 2: Hardware Configuration

1.2 Dataset

The data was taken from Kaggle which consisted of 1986 images already split into train and test. The data was downloaded from Kaggle and then uploaded on Google drive.

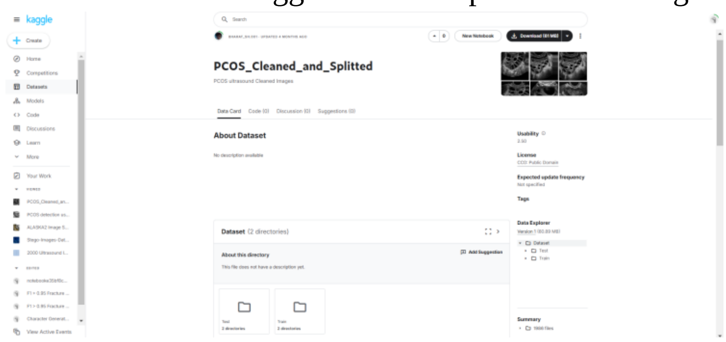

Figure 1: Data Source

1.3 Google Colab Setup

The images are stored in Google Drive folder, so that these images are accessible via Google Colab as represented by Figure 2.

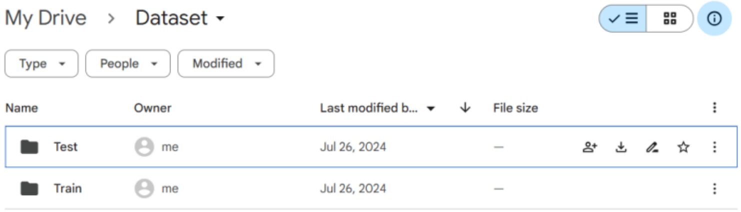

Figure 2: Dataset in Google Drive

Once the images are stored, Google Colab tries to build a connection with the Google drive folder where the images are stored using the 'mount' function as observed in Figure 3.

Figure 3: Google Colab Mount

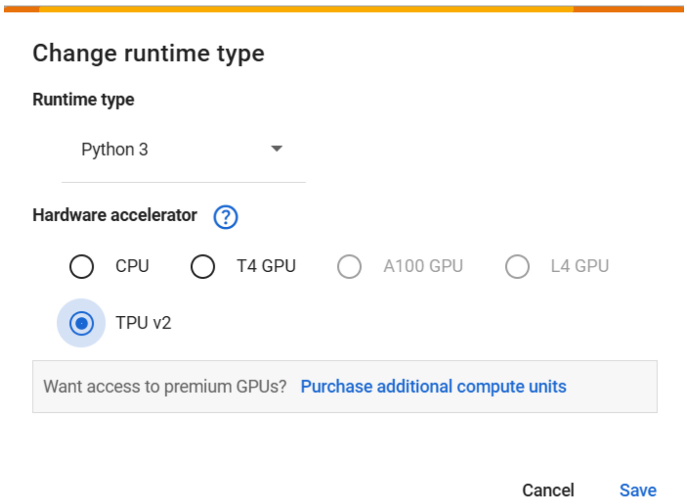

Figure 4: TPU v2 configuration

Figure 4 shows the hardware accelerator options in Google Colab, highlighting the TPU v2, which provides 300GB of RAM, as the chosen GPU for the project to ensure powerful configurational capabilities.

1.4 Importing Libraries

Here, the libraries for both SRGAN and CNN are imported.

```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.layers import Dense, Reshape, Flatten, Conv2D, LeakyReLU, BatchNormalization, PReLU, Add, Input, UpSampling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import VGG19
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.optimizers.schedules import ExponentialDecay
```

```
import os
import numpy as np
from PIL import Image
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import NASNetMobile, ResNet152, Xception
import tensorflow as tf
import random
from google.colab import drive
import matplotlib.pyplot as plt
import cv2
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, roc_curve, auc
from catboost import CatBoostClassifier
```

Figure 5: Libraries

2 Data Preprocessing

2.1 Exploratory Data Analysis (EDA)

Before preprocessing, basic EDA is performed to know the data distribution for each class on the combined dataset after generating images as shown in Figure 6.

```
import matplotlib.pyplot as plt
import seaborn as sns


train_count = len(normalized_train_images)
val_count = len(normalized_val_images)
test_count = len(normalized_test_images)
generated_count = len(equalized_generated_images)

print(f"Number of samples in the training set: {train_count}")
print(f"Number of samples in the validation set: {val_count}")
print(f"Number of samples in the test set: {test_count}")
print(f"Number of samples in the generated set: {generated_count}")


train_class_distribution = np.unique(combined_train_labels, return_counts=True)
val_class_distribution = np.unique(original_val_labels, return_counts=True)
test_class_distribution = np.unique(test_labels, return_counts=True)
generated_class_distribution = np.unique(generated_labels, return_counts=True)


def plot_class_distribution(class_distribution, set_name):
    classes, counts = class_distribution
    plt.figure(figsize=(10, 6))
    sns.barplot(x=classes, y=counts, palette='viridis')
    plt.title(f'Class Distribution in {set_name} Set')
    plt.xlabel('Classes')
    plt.ylabel('Counts')
    plt.show()

plot_class_distribution(train_class_distribution, 'Training')
plot_class_distribution(val_class_distribution, 'Validation')
plot_class_distribution(test_class_distribution, 'Test')
plot_class_distribution(generated_class_distribution, 'Generated')
```

```
Number of samples in the training set: 6034
Number of samples in the validation set: 318
Number of samples in the test set: 398
Number of samples in the generated set: 4764
```

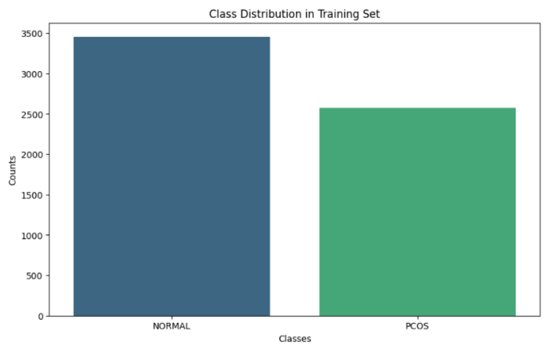

Figure 6: EDA

2.2 Data Pre-processing

Here, the preprocessing is divided into two parts: Pre-processing for SRGAN and pre-processing for CNN classification.

SRGAN Pre-processing: The pre-processing here is applied on the original training dataset before feeding the images into the generator.

```python
#resizing to 128x128
img_height, img_width = 128, 128

#loading the images
def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        if os.path.isfile(img_path):
            img = load_img(img_path, target_size=(img_height, img_width), color_mode='grayscale')
            img_array = img_to_array(img).astype(np.float32)
            images.append(img_array)
    return np.array(images)

normal_images = load_images_from_folder('/content/drive/MyDrive/Dataset/Train/NORMAL')
pcos_images = load_images_from_folder('/content/drive/MyDrive/Dataset/Train/PCOS')

#normalising
normal_images_norm = (normal_images / 127.5) - 1.0
pcos_images_norm = (pcos_images / 127.5) - 1.0
```

Figure 7: Pre-processing for SRGAN

CNN & Hybrid CNN pre-processing: Pre-processing here is applied to the combined dataset (SRGAN-generated images + original training images) and testing images.

```python
#loading original data
train_dir = '/content/drive/MyDrive/Dataset/Train'
test_dir = '/content/drive/MyDrive/Dataset/Test'
train_images, train_labels = load_data(train_dir)
test_images, test_labels = load_data(test_dir)

#loading the generated data
generated_data_dir = '/content/drive/MyDrive/AllGenerated2'
generated_images, generated_labels = load_data(generated_data_dir)
```

Figure 8: Data Directories

```python
[ ] def load_data(directory, target_size=(224, 224)):
        images = []
        labels = []
        class_names = sorted(os.listdir(directory))
        for class_name in class_names:
            class_dir = os.path.join(directory, class_name)
            for image_name in os.listdir(class_dir):
                image_path = os.path.join(class_dir, image_name)
                image = Image.open(image_path).convert('L')
                image = image.resize(target_size)
                image = np.stack((image,)*3, axis=-1)
                images.append(np.array(image))
                labels.append(class_name)
        return np.array(images), np.array(labels)

    #histogram equalisation
    def apply_histogram_equalization(image):
        image_yuv = cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
        image_yuv[:,:,0] = cv2.equalizeHist(image_yuv[:,:,0])
        image_equalized = cv2.cvtColor(image_yuv, cv2.COLOR_YUV2RGB)
        return image_equalized
```

```python
#normalising
def normalize_images(image_list):
    normalized_images = []
    for image in image_list:
        normalized_image = image.astype('float32') / 255.0
        normalized_images.append(normalized_image)
    return np.array(normalized_images)

normalized_train_images = normalize_images(combined_train_images)
normalized_val_images = normalize_images(original_val_images)
normalized_test_images = normalize_images(equalized_test_images)
```

Figure 9: Pre-processing for CNN & Hybrid models

After pre-processing, the images are label encoded since it's a binary classification problem where PCOS is 1 (positive class) and NORMAL is 0 (negative class).

```
class_labels = dict(zip(le.classes_, le.transform(le.classes_)))
print("Classes and their corresponding labels:")
for label, encoded in class_labels.items():
    print(f"{label}: {encoded}")
```

```
Classes and their corresponding labels:
NORMAL: 0
PCOS: 1
```

Figure 10: Label encoding

3 SRGAN Data Generation[1]

3.1 Variation 1

SRGAN-Variation 1

```
#generator
def build_generator_v1():
    input_layer = Input(shape=(img_height // 2, img_width // 2, 1))
    x = Conv2D(64, kernel_size=9, strides=1, padding='same')(input_layer)
    x = LeakyReLU(alpha=0.2)(x)

    x = UpSampling2D(size=2)(x)
    x = Conv2D(64, kernel_size=3, strides=1, padding='same')(x)
    x = LeakyReLU(alpha=0.2)(x)

    output_layer = Conv2D(1, kernel_size=9, strides=1, padding='same', activation='tanh')(x)

    return Model(input_layer, output_layer)

#discriminator
def build_discriminator_v1():
    input_layer = Input(shape=(img_height, img_width, 1))

    x = Conv2D(64, kernel_size=3, strides=1, padding='same')(input_layer)
    x = LeakyReLU(alpha=0.2)(x)

    x = Conv2D(64, kernel_size=3, strides=2, padding='same')(x)
    x = BatchNormalization(momentum=0.8)(x)
    x = LeakyReLU(alpha=0.2)(x)

    x = Conv2D(128, kernel_size=3, strides=1, padding='same')(x)
    x = BatchNormalization(momentum=0.8)(x)
    x = LeakyReLU(alpha=0.2)(x)

    x = Conv2D(128, kernel_size=3, strides=2, padding='same')(x)
    x = BatchNormalization(momentum=0.8)(x)
    x = LeakyReLU(alpha=0.2)(x)

    x = Flatten()(x)
    x = Dense(1024)(x)
    x = LeakyReLU(alpha=0.2)(x)
    output_layer = Dense(1, activation='sigmoid')(x)

    return Model(input_layer, output_layer)
```

```
optimizer_gen_v1 = Adam(learning_rate=0.0001, beta_1=0.5)
optimizer_disc_v1 = Adam(learning_rate=0.0001, beta_1=0.5)

discriminator_v1 = build_discriminator_v1()
discriminator_v1.compile(loss='binary_crossentropy', optimizer=optimizer_disc_v1, metrics=['accuracy'])

generator_v1 = build_generator_v1()

discriminator_v1.trainable = False
gan_input_v1 = Input(shape=(img_height // 2, img_width // 2, 1))
generated_image_v1 = generator_v1(gan_input_v1)
gan_output_v1 = discriminator_v1(generated_image_v1)
gan_v1 = Model(gan_input_v1, gan_output_v1)
gan_v1.compile(loss='binary_crossentropy', optimizer=optimizer_gen_v1)
```

Figure 11: SRGAN Variation 1 Architecture

This configuration uses basic upsampling and convolutional layers utilizing ‘Conv2D’ to enhance the image resolution.

```
optimizer_gen_v1 = Adam(learning_rate=0.0001, beta_1=0.5)
optimizer_disc_v1 = Adam(learning_rate=0.0001, beta_1=0.5)

discriminator_v1 = build_discriminator_v1()
discriminator_v1.compile(loss='binary_crossentropy', optimizer=optimizer_disc_v1, metrics=['accuracy'])
```

Figure 12: Learning Rate for Variation 1

[1] https://medium.com/analytics-vidhya/super-resolution-gan-srgan-5e10438aec0c

```python
epochs_v1 = 100
batch_size_v1 = 32
save_interval_v1 = 10

#creating directories to save images
generated_images_path_normal_v1 = '/content/drive/MyDrive/variation1images/Normal'
generated_images_path_pcos_v1 = '/content/drive/MyDrive/variation1images/PCOS'
generated_grid_path_normal_v1 = '/content/drive/MyDrive/variation1images/Grid_Normal'
generated_grid_path_pcos_v1 = '/content/drive/MyDrive/variation1images/Grid_PCOS'
os.makedirs(generated_images_path_normal_v1, exist_ok=True)
os.makedirs(generated_images_path_pcos_v1, exist_ok=True)
os.makedirs(generated_grid_path_normal_v1, exist_ok=True)
os.makedirs(generated_grid_path_pcos_v1, exist_ok=True)

#saving images in a grid of 5x5
def save_grid_images_v1(epoch, generator, examples=25, class_label='NORMAL'):
    if class_label == 'NORMAL':
        data = normal_images_norm
        save_path = generated_grid_path_normal_v1
    else:
        data = pcos_images_norm
        save_path = generated_grid_path_pcos_v1

    idx = np.random.randint(0, data.shape[0], examples)
    low_res_imgs = data[idx]
    low_res_imgs = tf.image.resize(low_res_imgs, [img_height // 2, img_width // 2])
    gen_imgs = generator.predict(low_res_imgs)
    gen_imgs = denormalize(gen_imgs)

    plt.figure(figsize=(10, 10))
    for i in range(examples):
        plt.subplot(5, 5, i+1)
        plt.imshow(gen_imgs[i, :, :, 0], cmap='gray')
        plt.axis('off')
    plt.savefig(os.path.join(save_path, f"{class_label}_grid_epoch_{epoch}.png"))
    plt.close()
```

Figure 13: Saving of images

The images are saved in a grid of 5x5 after 10th epoch with save_interval_v1 = 10, to visualize the image quality.

```python
import pandas as pd

#initialising lists to store losses
d_losses = []
g_losses = []

#training loop
for epoch in range(epochs_v1):
    #training on NORMAL images
    idx = np.random.randint(0, normal_images_norm.shape[0], batch_size_v1)
    high_res_imgs = tf.convert_to_tensor(normal_images_norm[idx])
    low_res_imgs = tf.image.resize(high_res_imgs, [img_height // 2, img_width // 2])
    fake_high_res_imgs = generator_v1.predict(low_res_imgs)

    d_loss_real = discriminator_v1.train_on_batch(high_res_imgs, np.ones((batch_size_v1, 1)))
    d_loss_fake = discriminator_v1.train_on_batch(fake_high_res_imgs, np.zeros((batch_size_v1, 1)))
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    valid_y = np.ones((batch_size_v1, 1))
    g_loss = gan_v1.train_on_batch(low_res_imgs, valid_y)

    d_losses.append(d_loss[0])
    g_losses.append(g_loss)

    print(f"{epoch} [D loss: {d_loss[0]} | D accuracy: {100 * d_loss[1]}%] [G loss: {g_loss}]")

    if epoch % save_interval_v1 == 0:
        save_imgs_v1(epoch, generator_v1, class_label='NORMAL')
        save_grid_images_v1(epoch, generator_v1, class_label='NORMAL')

    #training on PCOS images
    idx = np.random.randint(0, pcos_images_norm.shape[0], batch_size_v1)
    high_res_imgs = tf.convert_to_tensor(pcos_images_norm[idx])
    low_res_imgs = tf.image.resize(high_res_imgs, [img_height // 2, img_width // 2])
    fake_high_res_imgs = generator_v1.predict(low_res_imgs)

    d_loss_real = discriminator_v1.train_on_batch(high_res_imgs, np.ones((batch_size_v1, 1)))
    d_loss_fake = discriminator_v1.train_on_batch(fake_high_res_imgs, np.zeros((batch_size_v1, 1)))
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    valid_y = np.ones((batch_size_v1, 1))
    g_loss = gan_v1.train_on_batch(low_res_imgs, valid_y)

    d_losses.append(d_loss[0])
    g_losses.append(g_loss)

    print(f"{epoch} [D loss: {d_loss[0]} | D accuracy: {100 * d_loss[1]}%] [G loss: {g_loss}]")

    if epoch % save_interval_v1 == 0:
        save_imgs_v1(epoch, generator_v1, class_label='PCOS')
        save_grid_images_v1(epoch, generator_v1, class_label='PCOS')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
/usr/local/lib/python3.10/dist-packages/keras/src/backend/tensorflow/trainer.py:75: UserWarning: The model does not have any trainable weights.
  warnings.warn("The model does not have any trainable weights.")
0 [D loss: 0.7010604739189148 | D accuracy: 23.4375%] [G loss: [array(0.6983993, dtype=float32), array(0.6983993, dtype=float32), array(0.3125, dtype=float32)]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 892ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_train_function.<locals>.one_step_on_iterator at 0x7e99302863b0> triggered tf.functior
WARNING:tensorflow:6 out of the last 6 calls to <function TensorFlowTrainer.make_train_function.<locals>.one_step_on_iterator at 0x7e99301b5ea0> triggered tf.functior
0 [D loss: 0.6975681781768799 | D accuracy: 33.33333134651184%] [G loss: [array(0.69697356, dtype=float32), array(0.69697356, dtype=float32), array(0.34375, dtype=flc
```

Figure 14: Training Loop for Variation 1

Displaying images for last epoch

```python
def display_last_epoch_images_v1(generator, examples=10):
    def display_images(class_label):
        if class_label == 'NORMAL':
            data = normal_images_norm
        else:
            data = pcos_images_norm

        idx = np.random.randint(0, data.shape[0], examples)
        low_res_imgs = data[idx]
        low_res_imgs = tf.image.resize(low_res_imgs, [img_height // 2, img_width // 2])
        gen_imgs = generator.predict(low_res_imgs)
        gen_imgs = denormalize(gen_imgs)
        low_res_imgs = denormalize(low_res_imgs)

        plt.figure(figsize=(20, 8))
        for i in range(examples):
            plt.subplot(2, examples, i+1)
            plt.imshow(low_res_imgs[i, :, :, 0], cmap='gray')
            plt.axis('off')
            plt.title(f'{class_label} Low-Res')

            plt.subplot(2, examples, i+1+examples)
            plt.imshow(gen_imgs[i, :, :, 0], cmap='gray')
            plt.axis('off')
            plt.title(f'{class_label}')

        plt.show()


    display_images('NORMAL')
    display_images('PCOS')

display_last_epoch_images_v1(generator_v1)
```

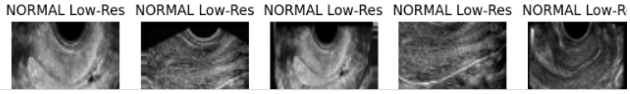

Figure 15: Display generated images for last epoch

3.2 Variation 2

Variation 2 (addition of residual blocks)

```python
#residual blocks
def residual_block_v2(input_layer, filters):
    x = Conv2D(filters, kernel_size=3, strides=1, padding='same')(input_layer)
    x = BatchNormalization(momentum=0.8)(x)
    x = PReLU(shared_axes=[1, 2])(x)
    x = Conv2D(filters, kernel_size=3, strides=1, padding='same')(x)
    x = BatchNormalization(momentum=0.8)(x)
    return Add()([input_layer, x])
```

Figure 16: Residual blocks

For variation 2, the architecture for generator and discriminator remains the same as variation 1, residual blocks are added for variation 2 generator. For discriminator, the structure is the same as variation 1, but more convolutional layers along with Spectral Normalisation is added. This is demonstrated in Figure 17.

```
[ ] #discriminator with spectral normalisation
    def build_discriminator_v2():
        input_layer = Input(shape=(img_height, img_width, 1))

        x = SpectralNormalization(Conv2D(64, kernel_size=3, strides=1, padding='same'))(input_layer)
        x = LeakyReLU(alpha=0.2)(x)

        x = SpectralNormalization(Conv2D(64, kernel_size=3, strides=2, padding='same'))(x)
        x = BatchNormalization(momentum=0.8)(x)
        x = LeakyReLU(alpha=0.2)(x)

        x = SpectralNormalization(Conv2D(128, kernel_size=3, strides=1, padding='same'))(x)
        x = BatchNormalization(momentum=0.8)(x)
        x = LeakyReLU(alpha=0.2)(x)

        x = SpectralNormalization(Conv2D(128, kernel_size=3, strides=2, padding='same'))(x)
        x = BatchNormalization(momentum=0.8)(x)
        x = LeakyReLU(alpha=0.2)(x)

        x = SpectralNormalization(Conv2D(256, kernel_size=3, strides=1, padding='same'))(x)
        x = BatchNormalization(momentum=0.8)(x)
        x = LeakyReLU(alpha=0.2)(x)

        x = SpectralNormalization(Conv2D(256, kernel_size=3, strides=2, padding='same'))(x)
        x = BatchNormalization(momentum=0.8)(x)
        x = LeakyReLU(alpha=0.2)(x)

        x = SpectralNormalization(Conv2D(512, kernel_size=3, strides=1, padding='same'))(x)
        x = BatchNormalization(momentum=0.8)(x)
        x = LeakyReLU(alpha=0.2)(x)

        x = SpectralNormalization(Conv2D(512, kernel_size=3, strides=2, padding='same'))(x)
        x = BatchNormalization(momentum=0.8)(x)
        x = LeakyReLU(alpha=0.2)(x)

        x = Flatten()(x)
        x = Dense(1024)(x)
        x = LeakyReLU(alpha=0.2)(x)
        output_layer = Dense(1, activation='sigmoid')(x)

        return Model(input_layer, output_layer)
```

Figure 17: Discriminator with Spectral Normalisation

The learning rate is increased to 0.0002 from 0.001. Training loop and saving structure for Variation 2 will be same as Variation 1.

```
optimizer_gen_v2 = Adam(learning_rate=0.0002, beta_1=0.5)
optimizer_disc_v2 = Adam(learning_rate=0.0002, beta_1=0.5)

discriminator_v2 = build_discriminator_v2()
discriminator_v2.compile(loss='binary_crossentropy', optimizer=optimizer_disc_v2, metrics=['accuracy'])
```

Figure 18: Learning Rate for Variation 2

3.3 Variation 3[2]

The variation 3 architecture is modified by introducing VGG19-based feature extractor to calculate the perpetual loss.

Variation 3

```python
#defining VGG19 model for perpetual loss
vgg_v3 = VGG19(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))
vgg_v3.trainable = False
vgg_model_v3 = Model(inputs=vgg_v3.input, outputs=vgg_v3.get_layer('block5_conv4').output)
mse_loss_v3 = MeanSquaredError()

def perceptual_loss_v3(y_true, y_pred):
    y_true_rgb = tf.image.grayscale_to_rgb(y_true)
    y_pred_rgb = tf.image.grayscale_to_rgb(y_pred)
    y_true_vgg = vgg_model_v3(y_true_rgb)
    y_pred_vgg = vgg_model_v3(y_pred_rgb)
    return mse_loss_v3(y_true_vgg, y_pred_vgg)
```

Figure 19: VGG19-feature extractor

As compared to variation 2, residual blocks are increased to 15 to in variation 3.

```python
#generator
def build_generator_v3():
    input_layer = Input(shape=(img_height // 2, img_width // 2, 1))
    x = Conv2D(64, kernel_size=9, strides=1, padding='same')(input_layer)
    x = PReLU(shared_axes=[1, 2])(x)

    r = residual_block_v3(x, 64)
    for _ in range(15):  #increasing to 15 residual blocks as compared to variation 2
        r = residual_block_v3(r, 64)
```

Figure 20: Residual blocks for Variation 3

The rest of the architecture remains the same as variation 2, including the learning rate.
The saving images mechanism for Variation 2 and 3 is same as Variation 1, just the folder names vary.

[2] https://manishdhakal.medium.com/super-resolution-with-gan-and-keras-srgan-4bd810d214b6

```
[ ] d_losses_epoch = []
    g_losses_epoch = []
    perceptual_losses_epoch = []

    for epoch in range(epochs_v3):

        d_loss_acc = []
        g_loss_acc = []
        perceptual_loss_acc = []

        idx = np.random.randint(0, normal_images_norm.shape[0], batch_size_v3)
        high_res_imgs = tf.convert_to_tensor(normal_images_norm[idx])
        low_res_imgs = tf.image.resize(high_res_imgs, [img_height // 2, img_width // 2])
        fake_high_res_imgs = generator_v3.predict(low_res_imgs)
        fake_high_res_imgs = tf.convert_to_tensor(fake_high_res_imgs)

        d_loss_real = discriminator_v3.train_on_batch(high_res_imgs, np.ones((batch_size_v3, 1)))
        d_loss_fake = discriminator_v3.train_on_batch(fake_high_res_imgs, np.zeros((batch_size_v3, 1)))
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
        d_loss_acc.append(d_loss[0])

        valid_y = np.ones((batch_size_v3, 1))
        g_loss = gan_v3.train_on_batch(low_res_imgs, valid_y)
        perceptual_loss_value = perceptual_loss_v3(high_res_imgs, fake_high_res_imgs)
        g_loss_total = g_loss + perceptual_loss_value

        g_loss_acc.append(g_loss)
        perceptual_loss_acc.append(perceptual_loss_value.numpy())

        print(f"{epoch} [D loss: {d_loss[0]} | D accuracy: {100 * d_loss[1]}%] [G loss: {g_loss_total}]")

        if epoch % save_interval_v3 == 0:
            save_imgs_v3(epoch, generator_v3, class_label='NORMAL')
            save_grid_images_v3(epoch, generator_v3, class_label='NORMAL')

        idx = np.random.randint(0, pcos_images_norm.shape[0], batch_size_v3)
        high_res_imgs = tf.convert_to_tensor(pcos_images_norm[idx])
        low_res_imgs = tf.image.resize(high_res_imgs, [img_height // 2, img_width // 2])
        fake_high_res_imgs = generator_v3.predict(low_res_imgs)
        fake_high_res_imgs = tf.convert_to_tensor(fake_high_res_imgs)

        d_loss_real = discriminator_v3.train_on_batch(high_res_imgs, np.ones((batch_size_v3, 1)))
        d_loss_fake = discriminator_v3.train_on_batch(fake_high_res_imgs, np.zeros((batch_size_v3, 1)))
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
        d_loss_acc.append(d_loss[0])

        g_loss = gan_v3.train_on_batch(low_res_imgs, valid_y)
        perceptual_loss_value = perceptual_loss_v3(high_res_imgs, fake_high_res_imgs)
        g_loss_total = g_loss + perceptual_loss_value

        g_loss_acc.append(g_loss)
        perceptual_loss_acc.append(perceptual_loss_value.numpy())

        print(f"{epoch} [D loss: {d_loss[0]} | D accuracy: {100 * d_loss[1]}%] [G loss: {g_loss_total}]")

        if epoch % save_interval_v3 == 0:
            save_imgs_v3(epoch, generator_v3, class_label='PCOS')
            save_grid_images_v3(epoch, generator_v3, class_label='PCOS')

        d_losses_epoch.append(np.mean(d_loss_acc))
        g_losses_epoch.append(np.mean(g_loss_acc))
        perceptual_losses_epoch.append(np.mean(perceptual_loss_acc))

    1/1 [==============================] - 15s 15s/step
    0 [D loss: 22.853540748357773 | D accuracy: 14.0625%] [G loss: 0.15439671277999878]
    1/1 [==============================] - 4s 4s/step
    1/1 [==============================] - 9s 9s/step
    1/1 [==============================] - 10s 10s/step
    0 [D loss: 8.909475326593498 | D accuracy: 50.0%] [G loss: 1.4317163228988647]
```

Figure 21: Training Loop for Variation 3

As observed in Figure 21, the Training Loop for Variation 4 has perceptual loss included.

3.4 Variation 4

The architecture for variation 4 is same as Variation 3 but certain modifications are made. The residual blocks are increased to 16 from 15.

```
#generator
def build_generator_v5():
    input_layer = Input(shape=(img_height // 2, img_width // 2, 1))
    x = Conv2D(64, kernel_size=9, strides=1, padding='same')(input_layer)
    x = PReLU(shared_axes=[1, 2])(x)

    r = residual_block_v5(x, 64)
    for _ in range(16):  #increasing the residual blocks to 16 in this case
        r = residual_block_v5(r, 64)

    r = Conv2D(64, kernel_size=3, strides=1, padding='same')(r)
    r = BatchNormalization(momentum=0.8)(r)
    x = Add()([x, r])
```

Figure 22: Residual blocks for Variation 4

The learning rate for Variation 4 is optimized using exponential delay. Figure 23 demonstrates the same.

```
#learning rate
learning_rate_gen_v5 = ExponentialDecay(0.0001, decay_steps=100000, decay_rate=0.96, staircase=True)
learning_rate_disc_v5 = ExponentialDecay(0.00005, decay_steps=100000, decay_rate=0.96, staircase=True)

#optimizers with learning rates and gradient clipping
optimizer_gen_v5 = Adam(learning_rate=learning_rate_gen_v5, beta_1=0.5, beta_2=0.999, clipvalue=1.0)
optimizer_disc_v5 = Adam(learning_rate=learning_rate_disc_v5, beta_1=0.5, beta_2=0.999, clipvalue=1.0)

discriminator_v5 = build_discriminator_v5()
discriminator_v5.compile(loss='binary_crossentropy', optimizer=optimizer_disc_v5, metrics=['accuracy'])
```

Figure 23: Learning rate for Variation 4

Since this variation is considered to be the final variation, along with saving images in a grid like the other variations, all the images from epochs 80 to epochs 90 will be stored as these epochs were giving the best images. The saving code remains the same and the below condition is added.

```
[ ] epochs_v5 = 100
    batch_size_v5 = 32
    save_interval_v5 = 1

    generated_images_path_normal_v5 = '/content/drive/MyDrive/variation4images/allnormal'
    generated_images_path_pcos_v5 = '/content/drive/MyDrive/variation4images/allpcos'
    generated_grid_path_normal_v5 = '/content/drive/MyDrive/variation4images/gridallnormal'
    generated_grid_path_pcos_v5 = '/content/drive/MyDrive/variation4images/gridallpcos'
    os.makedirs(generated_images_path_normal_v5, exist_ok=True)
    os.makedirs(generated_images_path_pcos_v5, exist_ok=True)
    os.makedirs(generated_grid_path_normal_v5, exist_ok=True)
    os.makedirs(generated_grid_path_pcos_v5, exist_ok=True)

    def save_imgs_v5(epoch, generator, class_label='NORMAL'):
        if class_label == 'NORMAL':
            data = normal_images_norm
            save_path = generated_images_path_normal_v5
        else:
            data = pcos_images_norm
            save_path = generated_images_path_pcos_v5

        low_res_imgs = data
        low_res_imgs = tf.image.resize(low_res_imgs, [img_height // 2, img_width // 2])
        gen_imgs = generator.predict(low_res_imgs)
        gen_imgs = denormalize(gen_imgs)
        low_res_imgs = denormalize(low_res_imgs)
        os.makedirs(os.path.join(save_path, f"epoch_{epoch}"), exist_ok=True)
        for i in range(data.shape[0]):
            plt.imsave(os.path.join(save_path, f"epoch_{epoch}", f"{class_label}_epoch_{epoch}_{i}.png"), gen_imgs[i].squeeze(), cmap='gray')
```

```
def save_grid_images_v5(epoch, generator, examples=25, class_label='NORMAL'):
    if class_label == 'NORMAL':
        data = normal_images_norm
        save_path = generated_grid_path_normal_v5
    else:
        data = pcos_images_norm
        save_path = generated_grid_path_pcos_v5

    idx = np.random.randint(0, data.shape[0], examples)
    low_res_imgs = data[idx]
    low_res_imgs = tf.image.resize(low_res_imgs, [img_height // 2, img_width // 2])
    gen_imgs = generator.predict(low_res_imgs)
    gen_imgs = denormalize(gen_imgs)

    plt.figure(figsize=(10, 10))
    for i in range(examples):
        plt.subplot(5, 5, i+1)
        plt.imshow(gen_imgs[i, :, :, 0], cmap='gray')
        plt.axis('off')
    os.makedirs(os.path.join(save_path, f"epoch_{epoch}"), exist_ok=True)
    plt.savefig(os.path.join(save_path, f"epoch_{epoch}", f"{class_label}_grid_epoch_{epoch}.png"))
    plt.close()
```

```
if epoch % save_interval_v5 == 0:
    save_grid_images_v5(epoch, generator_v5, class_label='NORMAL')
    save_grid_images_v5(epoch, generator_v5, class_label='PCOS')

if (80 <= epoch <= 90) or (90 < epoch <= 100):
    save_imgs_v5(epoch, generator_v5, class_label='NORMAL')
    save_imgs_v5(epoch, generator_v5, class_label='PCOS')
```

Figure 24: Saving images for Variation 4

Figure 25 shows the training loss for Variation 4 by introducing label smoothing. The structure is same as shown in Figure 21, but the feedback mechanism of the discriminator is optimized. Figure 27 demonstrates the addition in the training loop in variation 4, rest of the training remains the same like training loop for Variation 3.

```
    if epoch % 5 == 0:
        d_loss_real = discriminator_v5.train_on_batch(high_res_imgs, np.ones((batch_size_v5, 1)) * 0.9)  #label smoothing
        d_loss_fake = discriminator_v5.train_on_batch(fake_high_res_imgs, np.zeros((batch_size_v5, 1)))
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
        d_loss_acc.append(d_loss[0])
    else:
        d_loss_acc.append(d_loss_acc[-1] if d_loss_acc else 0)

    valid_y = np.ones((batch_size_v5, 1))
    g_loss = gan_v5.train_on_batch(low_res_imgs, valid_y) + perceptual_loss_v5(high_res_imgs, tf.convert_to_tensor(fake_high_res_imgs))
    g_loss_acc.append(g_loss)

    d_losses.append(np.mean(d_loss_acc))
    g_losses.append(np.mean(g_loss_acc))

    print(f"{epoch} [D loss: {d_losses[-1]} | D accuracy: {100 * d_loss[1] if epoch % 5 == 0 else 'NA'}%] [G loss: {g_losses[-1]}]")

    if epoch % save_interval_v5 == 0:
        save_grid_images_v5(epoch, generator_v5, class_label='NORMAL')
        save_grid_images_v5(epoch, generator_v5, class_label='PCOS')

    if (80 <= epoch <= 90) or (90 < epoch <= 100):
        save_imgs_v5(epoch, generator_v5, class_label='NORMAL')
        save_imgs_v5(epoch, generator_v5, class_label='PCOS')
```

```
1/1 [==============================] - 0s 455ms/step
1/1 [==============================] - 1s 990ms/step
1/1 [==============================] - 1s 1s/step
17 [D loss: 0.0 | D accuracy: NA%] [G loss: 0.5903947949409485]
1/1 [==============================] - 0s 456ms/step
```

Figure 25: Modification in training loop – Variation 4

```
[ ] plt.figure(figsize=(10, 5))
    plt.plot(range(epochs_v5), d_losses, label='Discriminator Loss')
    plt.plot(range(epochs_v5), g_losses, label='Generator Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.title('Discriminator vs Generator Loss')
    plt.show()
```

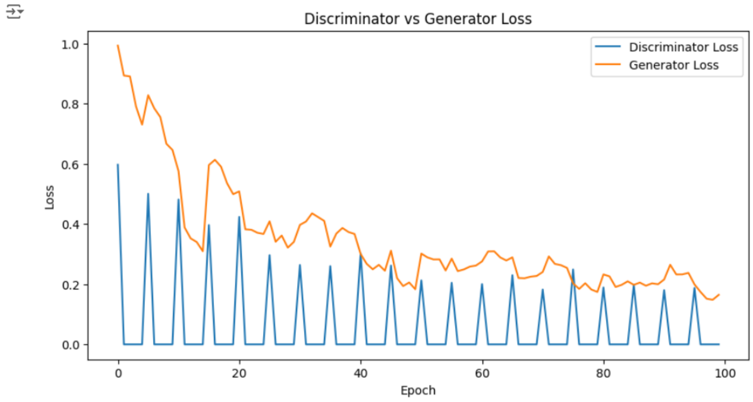

Figure 26: Generator vs Discriminator Loss Graph- Variation 4

Calculating SSIM for epochs 80 -90

```
def load_images_from_directory(path, target_size=(256, 256)):
    images = []
    for root, dirs, files in os.walk(path):
        for file in files:
            if file.endswith(('png', 'jpg', 'jpeg')):
                img = imread(os.path.join(root, file), as_gray=True)
                img_resized = resize(img, target_size, anti_aliasing=True)
                images.append(img_resized)
    return images


def calculate_ssim_pair(gen_img, real_img, data_range=1.0):
    return ssim(gen_img, real_img, data_range=data_range)

#average SSIM for real and generated images
def calculate_average_ssim(real_images, generated_images, data_range=1.0, sample_size=10):
    ssim_scores = []

    with ThreadPoolExecutor() as executor:
        futures = []
        for gen_img in generated_images:
            sampled_real_images = random.sample(real_images, min(len(real_images), sample_size))
            for real_img in sampled_real_images:
                futures.append(executor.submit(calculate_ssim_pair, gen_img, real_img, data_range))

        for future in as_completed(futures):
            ssim_scores.append(future.result())

    return np.mean(ssim_scores) if ssim_scores else None
```

Figure 27: SSIM Score

4 CNN Classification[3]

4.1 Data Splitting

The original data was already split into train and test. After the generation of images, the SRGAN-generated data was combined with the original train data. The original train data was split into validation set with the ratio 80:20 as shown in Figure 28.

```
#splitting original training data into a new training set and validation set
original_train_images, original_val_images, original_train_labels, original_val_labels = train_test_split(
    equalized_train_images, train_labels, test_size=0.2, random_state=42
)
```

Figure 28: Data split into train and val

[3] https://medium.com/towards-data-science/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

4.2 Modelling

1. NasNetMobile

nasnet-mobile

```python
base_model_nasnet_mobile = NASNetMobile(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3

for layer in base_model_nasnet_mobile.layers:
    layer.trainable = False

x = base_model_nasnet_mobile.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(len(le.classes_), activation='softmax')(x)

model_nasnet_mobile = Model(inputs=base_model_nasnet_mobile.input, outputs=predictions)
model_nasnet_mobile.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model_nasnet_mobile.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/nasnet/NASNet-mobile-no-top.h5
19993432/19993432 [==============================] - 0s 0us/step
Model: "model"
```

Figure 29: NasNetMobile Architecture

```python
#train the model with the new training set and validate on the validation set
history_nasnet_mobile = model_nasnet_mobile.fit(
    train_generator,
    steps_per_epoch=len(normalized_train_images) // 32,
    epochs=10,
    validation_data=val_generator,
    validation_steps=len(normalized_val_images) // 32
)
```

```
Epoch 1/10
188/188 [==============================] - 54s 219ms/step - loss: 0.0965 - accuracy: 0.9630 - val_loss: 0.0049 - val_accuracy: 1.0000
Epoch 2/10
188/188 [==============================] - 33s 174ms/step - loss: 0.0079 - accuracy: 0.9993 - val_loss: 6.6293e-04 - val_accuracy: 1.0000
Epoch 3/10
188/188 [==============================] - 34s 181ms/step - loss: 0.0058 - accuracy: 0.9988 - val_loss: 7.6827e-04 - val_accuracy: 1.0000
Epoch 4/10
188/188 [==============================] - 33s 176ms/step - loss: 0.0051 - accuracy: 0.9983 - val_loss: 0.0019 - val_accuracy: 1.0000
Epoch 5/10
188/188 [==============================] - 34s 180ms/step - loss: 0.0049 - accuracy: 0.9980 - val_loss: 0.0135 - val_accuracy: 0.9931
Epoch 6/10
188/188 [==============================] - 33s 173ms/step - loss: 0.0036 - accuracy: 0.9993 - val_loss: 5.0802e-05 - val_accuracy: 1.0000
Epoch 7/10
188/188 [==============================] - 33s 175ms/step - loss: 7.1034e-04 - accuracy: 0.9998 - val_loss: 2.9991e-05 - val_accuracy: 1.0000
Epoch 8/10
188/188 [==============================] - 33s 173ms/step - loss: 0.0029 - accuracy: 0.9993 - val_loss: 0.0085 - val_accuracy: 0.9965
Epoch 9/10
188/188 [==============================] - 33s 176ms/step - loss: 0.0254 - accuracy: 0.9932 - val_loss: 0.0022 - val_accuracy: 1.0000
Epoch 10/10
188/188 [==============================] - 34s 179ms/step - loss: 0.0197 - accuracy: 0.9957 - val_loss: 0.0075 - val_accuracy: 0.9965
```

Figure 30: Training for NasNetMobile

2. ResNet152

resnet-152

```python
from tensorflow.keras.applications import ResNet152
base_model_resnet152 = ResNet152(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3)))

for layer in base_model_resnet152.layers:
    layer.trainable = False

x = base_model_resnet152.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(len(le.classes_), activation='softmax')(x)

model_resnet152 = Model(inputs=base_model_resnet152.input, outputs=predictions)
model_resnet152.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model_resnet152.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet152_weights_tf_dim_ordering_tf_kernels_notop.h5
234698864/234698864 [==============================] - 1s 0us/step
Model: "model_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 224, 224, 3)]	0	[]
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	['input_2[0][0]']
conv1_conv (Conv2D)	(None, 112, 112, 64)	9472	['conv1_pad[0][0]']
conv1_bn (BatchNormalization)	(None, 112, 112, 64)	256	['conv1_conv[0][0]']

Figure 31: ResNet152 Architecture

```
history_resnet152 = model_resnet152.fit(
    train_generator,
    steps_per_epoch=len(normalized_train_images) // 32,
    epochs=10,
    validation_data=val_generator,
    validation_steps=len(normalized_val_images) // 32
)
```

```
Epoch 1/10
188/188 [==============================] - 219s 1s/step - loss: 0.1899 - accuracy: 0.9245 - val_loss: 0.3059 - val_accuracy: 0.8403
Epoch 2/10
188/188 [==============================] - 213s 1s/step - loss: 0.0560 - accuracy: 0.9818 - val_loss: 0.2339 - val_accuracy: 0.8715
Epoch 3/10
188/188 [==============================] - 212s 1s/step - loss: 0.0425 - accuracy: 0.9873 - val_loss: 0.0222 - val_accuracy: 0.9931
Epoch 4/10
188/188 [==============================] - 205s 1s/step - loss: 0.0255 - accuracy: 0.9917 - val_loss: 0.0137 - val_accuracy: 0.9965
Epoch 5/10
188/188 [==============================] - 214s 1s/step - loss: 0.0192 - accuracy: 0.9948 - val_loss: 0.0159 - val_accuracy: 0.9965
Epoch 6/10
188/188 [==============================] - 208s 1s/step - loss: 0.0148 - accuracy: 0.9960 - val_loss: 0.0072 - val_accuracy: 1.0000
Epoch 7/10
188/188 [==============================] - 212s 1s/step - loss: 0.0219 - accuracy: 0.9927 - val_loss: 0.0738 - val_accuracy: 0.9688
Epoch 8/10
188/188 [==============================] - 207s 1s/step - loss: 0.0203 - accuracy: 0.9932 - val_loss: 0.0062 - val_accuracy: 1.0000
Epoch 9/10
188/188 [==============================] - 199s 1s/step - loss: 0.0251 - accuracy: 0.9912 - val_loss: 0.0108 - val_accuracy: 1.0000
Epoch 10/10
188/188 [==============================] - 198s 1s/step - loss: 0.0162 - accuracy: 0.9943 - val_loss: 0.0061 - val_accuracy: 1.0000
```

Figure 32: Training for ResNet152

3. Xception

The Xception model requires an image input size of 229x229. Preprocessing steps mentioned in Figure 8 are same applied for Xception model but the input size is 229x229.

```
from tensorflow.keras.applications import Xception

base_model_xception = Xception(weights='imagenet', include_top=False, input_tensor=Input(shape=(299, 299, 3)))

for layer in base_model_xception.layers:
    layer.trainable = False

x_xception = base_model_xception.output
x_xception = GlobalAveragePooling2D()(x_xception)
x_xception = Dense(512, activation='relu')(x_xception)
x_xception = Dropout(0.5)(x_xception)
predictions_xception = Dense(len(le_xception.classes_), activation='softmax')(x_xception)

model_xception = Model(inputs=base_model_xception.input, outputs=predictions_xception)
model_xception.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model_xception.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83683744/83683744 [==============================] - 0s 0us/step
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_3 (InputLayer)        [(None, 299, 299, 3)]        0         []

 block1_conv1 (Conv2D)       (None, 149, 149, 32)         864       ['input_3[0][0]']
```

Figure 33: Xception Architecture

```
history_xception = model_xception.fit(
    train_generator_xception,
    steps_per_epoch=len(normalized_train_images_xception) // 32,
    epochs=10,
    validation_data=val_generator_xception,
    validation_steps=len(normalized_val_images_xception) // 32
)
```

```
Epoch 1/10
188/188 [==============================] - 180s 942ms/step - loss: 0.0943 - accuracy: 0.9650 - val_loss: 1.3601e-04 - val_accuracy: 1.0000
Epoch 2/10
188/188 [==============================] - 177s 942ms/step - loss: 0.0146 - accuracy: 0.9957 - val_loss: 4.1230e-04 - val_accuracy: 1.0000
Epoch 3/10
188/188 [==============================] - 180s 956ms/step - loss: 0.0099 - accuracy: 0.9970 - val_loss: 3.9436e-04 - val_accuracy: 1.0000
Epoch 4/10
188/188 [==============================] - 167s 890ms/step - loss: 0.0054 - accuracy: 0.9983 - val_loss: 1.7451e-05 - val_accuracy: 1.0000
Epoch 5/10
188/188 [==============================] - 166s 885ms/step - loss: 0.0091 - accuracy: 0.9968 - val_loss: 1.3231e-05 - val_accuracy: 1.0000
Epoch 6/10
188/188 [==============================] - 171s 911ms/step - loss: 0.0038 - accuracy: 0.9988 - val_loss: 1.7416e-05 - val_accuracy: 1.0000
Epoch 7/10
188/188 [==============================] - 169s 897ms/step - loss: 0.0020 - accuracy: 0.9993 - val_loss: 9.4124e-07 - val_accuracy: 1.0000
Epoch 8/10
188/188 [==============================] - 180s 955ms/step - loss: 0.0144 - accuracy: 0.9953 - val_loss: 9.5202e-09 - val_accuracy: 1.0000
Epoch 9/10
188/188 [==============================] - 180s 956ms/step - loss: 0.0121 - accuracy: 0.9957 - val_loss: 8.2784e-10 - val_accuracy: 1.0000
Epoch 10/10
188/188 [==============================] - 164s 871ms/step - loss: 0.0099 - accuracy: 0.9965 - val_loss: 1.4073e-08 - val_accuracy: 1.0000
```

Figure 34: Training for Xception

4. NasNetMobile + CatBoost[4]

NasNetMobile is used a feature extractor here as demonstrated below.

NasNetMobile + catboost

```
from tensorflow.keras.applications import NASNetMobile
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


hybrid_model_nasnet_mobile = NASNetMobile(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3)))

for layer in hybrid_model_nasnet_mobile.layers:
    layer.trainable = False

#NasNetMobile model for feature extraction
feature_extractor_nasnet_mobile = Model(inputs=hybrid_model_nasnet_mobile.input, outputs=hybrid_model_nasnet_mobile.output)


train_features_nasnet_mobile = feature_extractor_nasnet_mobile.predict(normalized_train_images, batch_size=32)
train_features_nasnet_mobile = train_features_nasnet_mobile.reshape(train_features_nasnet_mobile.shape[0], -1)

val_features_nasnet_mobile = feature_extractor_nasnet_mobile.predict(normalized_val_images, batch_size=32)
val_features_nasnet_mobile = val_features_nasnet_mobile.reshape(val_features_nasnet_mobile.shape[0], -1)

test_features_nasnet_mobile = feature_extractor_nasnet_mobile.predict(normalized_test_images, batch_size=32)
test_features_nasnet_mobile = test_features_nasnet_mobile.reshape(test_features_nasnet_mobile.shape[0], -1)
```

```
189/189 [==============================] - 34s 162ms/step
10/10 [==============================] - 2s 153ms/step
13/13 [==============================] - 2s 147ms/step
```

Figure 35: Feature Extraction using NasNetMobile

[4] https://forecastegy.com/posts/catboost-binary-classification-python/

```
!pip install catboost

from catboost import CatBoostClassifier

catboost_classifier = CatBoostClassifier(iterations=100, learning_rate=0.1, depth=4, verbose=10)
catboost_classifier.fit(train_features_nasnet_mobile, encoded_train_labels, eval_set=(val_features_nasnet_mobile, encoded_val_labels))

#predicting probabilities using the CatBoost classifier
catboost_probabilities = catboost_classifier.predict_proba(test_features_nasnet_mobile)[:, 1]

catboost_predictions = catboost_classifier.predict(test_features_nasnet_mobile)

# Evaluate the CatBoost classifier
catboost_accuracy = accuracy_score(encoded_test_labels, catboost_predictions)
print(f"CatBoost Accuracy: {catboost_accuracy}")

catboost_confusion_matrix = confusion_matrix(encoded_test_labels, catboost_predictions)
print("Confusion Matrix for CatBoost:")
print(catboost_confusion_matrix)

catboost_classification_report = classification_report(encoded_test_labels, catboost_predictions, target_names=le.classes_)
print("Classification Report for CatBoost:")
print(catboost_classification_report)
```

Figure 36: Classification using CatBoost for NasNetMobile

5. ResNet152 + CatBoost

resnet152 + catboost

```
#ResNet152 model for feature extraction
hybrid_model_resnet152 = ResNet152(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3)))

for layer in hybrid_model_resnet152.layers:
    layer.trainable = False

feature_extractor_resnet152 = Model(inputs=hybrid_model_resnet152.input, outputs=hybrid_model_resnet152.output)


train_features_resnet152 = feature_extractor_resnet152.predict(normalized_train_images, batch_size=32)
train_features_resnet152 = train_features_resnet152.reshape(train_features_resnet152.shape[0], -1)

val_features_resnet152 = feature_extractor_resnet152.predict(normalized_val_images, batch_size=32)
val_features_resnet152 = val_features_resnet152.reshape(val_features_resnet152.shape[0], -1)

test_features_resnet152 = feature_extractor_resnet152.predict(normalized_test_images, batch_size=32)
test_features_resnet152 = test_features_resnet152.reshape(test_features_resnet152.shape[0], -1)
```

```
189/189 [==============================] - 204s 1s/step
10/10 [==============================] - 11s 1s/step
13/13 [==============================] - 13s 1s/step
```

Figure 37: Feature Extraction using ResNet152

```
catboost_classifier.fit(train_features_resnet152, encoded_train_labels, eval_set=(val_features_resnet152, encoded_val_labels))

catboost_probabilities_resnet = catboost_classifier.predict_proba(test_features_resnet152)[:, 1]

catboost_predictions_resnet = catboost_classifier.predict(test_features_resnet152)

catboost_accuracy_resnet = accuracy_score(encoded_test_labels, catboost_predictions_resnet)
print(f"CatBoost Accuracy (ResNet152): {catboost_accuracy_resnet}")

catboost_confusion_matrix_resnet = confusion_matrix(encoded_test_labels, catboost_predictions_resnet)
print("Confusion Matrix for CatBoost (ResNet152):")
print(catboost_confusion_matrix_resnet)

catboost_classification_report_resnet = classification_report(encoded_test_labels, catboost_predictions_resnet, target_names=le.classes_)
print("Classification Report for CatBoost (ResNet152):")
print(catboost_classification_report_resnet)
```

```
0:      learn: 0.5147305        test: 0.5395014 best: 0.5395014 (0)     total: 135ms    remaining: 13.4s
10:     learn: 0.0563956        test: 0.0990036 best: 0.0990036 (10)    total: 1.43s    remaining: 11.5s
20:     learn: 0.0212558        test: 0.0448185 best: 0.0448185 (20)    total: 2.71s    remaining: 10.2s
30:     learn: 0.0128519        test: 0.0303845 best: 0.0303845 (30)    total: 3.98s    remaining: 8.86s
40:     learn: 0.0089255        test: 0.0226259 best: 0.0226259 (40)    total: 5.26s    remaining: 7.57s
50:     learn: 0.0058349        test: 0.0155819 best: 0.0155819 (50)    total: 6.55s    remaining: 6.29s
60:     learn: 0.0040686        test: 0.0115472 best: 0.0115472 (60)    total: 7.84s    remaining: 5.01s
70:     learn: 0.0032941        test: 0.0095927 best: 0.0095926 (68)    total: 9.14s    remaining: 3.73s
80:     learn: 0.0030158        test: 0.0088571 best: 0.0088567 (75)    total: 10.4s    remaining: 2.45s
90:     learn: 0.0027203        test: 0.0078476 best: 0.0078475 (87)    total: 11.7s    remaining: 1.16s
99:     learn: 0.0027202        test: 0.0078480 best: 0.0078475 (87)    total: 12.9s    remaining: 0us

bestTest = 0.00784747985
bestIteration = 87

Shrink model to first 88 iterations.
```

Figure 38: Classification using CatBoost for ResNet152

6. Xception + CatBoost

xception + catboost

```
#Xception model for feature extraction
hybrid_model_xception = Xception(weights='imagenet', include_top=False, input_tensor=Input(shape=(299, 299, 3)))

for layer in hybrid_model_xception.layers:
    layer.trainable = False

feature_extractor_xception = Model(inputs=hybrid_model_xception.input, outputs=hybrid_model_xception.output)

train_features_xception = feature_extractor_xception.predict(normalized_train_images_xception, batch_size=32)
train_features_xception = train_features_xception.reshape(train_features_xception.shape[0], -1)

val_features_xception = feature_extractor_xception.predict(normalized_val_images_xception, batch_size=32)
val_features_xception = val_features_xception.reshape(val_features_xception.shape[0], -1)

test_features_xception = feature_extractor_xception.predict(normalized_test_images_xception, batch_size=32)
test_features_xception = test_features_xception.reshape(test_features_xception.shape[0], -1)
```

```
189/189 [==============================] - 188s 994ms/step
10/10 [==============================] - 10s 986ms/step
13/13 [==============================] - 11s 831ms/step
```

Figure 39: Feature Extraction using Xception

```
catboost_classifier = CatBoostClassifier(iterations=100, learning_rate=0.1, depth=4, verbose=10)
catboost_classifier.fit(train_features_xception, encoded_train_labels_xception, eval_set=(val_features_xception, encoded_val_labels_xception))

catboost_probabilities_xception = catboost_classifier.predict_proba(test_features_xception)[:, 1]

catboost_predictions_xception = catboost_classifier.predict(test_features_xception)


catboost_accuracy_xception = accuracy_score(encoded_test_labels_xception, catboost_predictions_xception)
print(f"CatBoost Accuracy: {catboost_accuracy_xception}")

catboost_confusion_matrix_xception = confusion_matrix(encoded_test_labels_xception, catboost_predictions_xception)
print("Confusion Matrix for CatBoost:")
print(catboost_confusion_matrix_xception)

catboost_classification_report_xception = classification_report(encoded_test_labels_xception, catboost_predictions_xception, target_names=le_xception.classes_)
print("Classification Report for CatBoost:")
print(catboost_classification_report_xception)
```

Figure 40: Classification using CatBoost for Xception