Configuration Manual

MSc Research Project

MSc Cybersecurity

Sahana Nagaraj

Student ID: X23325704

School of Computing

National College of Ireland

Supervisor: Michael Prior

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sahana Nagaraj
Student ID:	X23325704
Programme:	MSc Cybersecurity **Year:** 2025
Module:	MSc (Research) Practicum
Lecturer:	Michael Prior
Submission Due Date:	15/09/2025
Project Title:	Biometrics Security: Securing Facial Recognition Systems against Deepfakes and Spoofing Attacks
Word Count:	1507 **Page Count:** 12

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: Sahana Nagaraj

Date: 15/09/2025

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):	

Contents

Deepfake Detection Using DenseNet121: Configuration Manual

1. Overview

The present document includes code and configuration walkthrough of the codebase in the Deepfake Detection Using DenseNet121 project. This project carries out and contrasts three deep learning designs on recognizing deepfake images:

- DenseNet121: major construction with dense connectivity nature to reuse the features and optimize the parameters.
- ResNet50: Baseline ResNet with addition of residual neighbourhoods to increase flow of gradients.
- EfficientNet-B0: Large modern architecture with computationally-efficient compound scaling.

In this manual, I will discuss the setting of these models, implementation of their code, training protocols and assessment by transfer learning on the 140k real and fake faces.

2. The Requirements of System

2.1. Hardware

- **GPU:** NVIDIA RTX 3080/4090 or comparable and with CUDA enabled.
- RAM: 16GB + to carry out batch processing with efficiency.
- Storage: 50GB+ to store dataset and models.
- CPU: multicore data pre-processing unit.

2.2. Software

- Python 3.8+
- Jupyter Notebook
- TensorFlow 2.x
- Keras
- NumPy
- Pandas
- scikit-learn
- Matplotlib
- Seaborn
- Pillow (PIL)
- OpenCV

3. Installation

In order to install the required Python libraries, run the following commands

```
pip install tensorflow
pip install keras
pip install numpy pandas
pip install scikit-learn
pip install matplotlib seaborn
pip install pillow opencv-python
pip install jupyter
```

For GPU support:

```
pip install tensorflow-gpu
```

4. Code Description

The following section gives an account of the Python code employed in this project.

4.1. Imports and Model Loading

```
import tensorflow as tf
from tensorflow.keras.applications import DenseNet121, ResNet50,
EfficientNetB0
from tensorflow.keras.layers import Input, Dense,
GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import pandas as pd
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

This cell imports the necessary libraries and deep learning frameworks required for model implementation, data processing, and evaluation.

```
height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        brightness_range=[0.8, 1.2],
        fill_mode='nearest'
    )

    val_test_datagen = ImageDataGenerator(rescale=1./255)

    return train_datagen, val_test_datagen
```

4.2. Data Processing

```
def preprocess_image(image_path, target_size=(224, 224)):
    """
    Preprocess individual images for model input
    """
    from PIL import Image
    import numpy as np

    img = Image.open(image_path).convert('RGB')
    img = img.resize(target_size)
    img_array = np.array(img) / 255.0
    return img_array

def create_data_generators(train_dir, val_dir, test_dir):
    """
    Create data generators with augmentation for training and validation
    """
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
        width_shift_range=0.2,
```

These functions handles the image preprocessing, normalization, and data augmentation to improve model generalization and robustness.

```
height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        brightness_range=[0.8, 1.2],
        fill_mode='nearest'
    )

    val_test_datagen = ImageDataGenerator(rescale=1./255)

    return train_datagen, val_test_datagen
```

4.3. Model Architecture

```
def create_densenet_model():
    """
    Create DenseNet121 model with custom classification head
    """
    input_tensor = Input(shape=(224, 224, 3))
    base_model = DenseNet121(
        input_tensor=input_tensor,
        weights='imagenet',
        include_top=False
    )

    # Freeze base model layers for transfer learning
    for layer in base_model.layers:
        layer.trainable = False

    # Custom classification head
    x = GlobalAveragePooling2D()(base_model.output)
    x = Dense(512, activation='relu', name='dense_1')(x)
    x = Dropout(0.5)(x)
    x = Dense(256, activation='relu', name='dense_2')(x)
    x = Dropout(0.3)(x)
    final_output = Dense(1, activation='sigmoid', name='predictions')(x)

    model = Model(input_tensor, final_output)
    return model

def create_resnet_model():
    """
    Create ResNet50 model with custom classification head
    """
    input_tensor = Input(shape=(224, 224, 3))
    base_model = ResNet50(
        input_tensor=input_tensor,
        weights='imagenet',
        include_top=False
    )

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

    x = GlobalAveragePooling2D()(base_model.output)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x
```

```
)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.3)(x)
    final_output = Dense(1, activation='sigmoid')(x)

    model = Model(input_tensor, final_output)
    return model

def create_efficientnet_model():
    """
    Create EfficientNet-B0 model with custom classification head
    """
    input_tensor = Input(shape=(224, 224, 3))
    base_model = EfficientNetB0(
        input_tensor=input_tensor,
        weights='imagenet',
        include_top=False
    )

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

    x = GlobalAveragePooling2D()(base_model.output)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.3)(x)
    final_output = Dense(1, activation='sigmoid')(x)

    model = Model(input_tensor, final_output)
    return model
```

4.4. Data Generators

```
def setup_data_flow(train_dir, val_dir, test_dir, batch_size=32):
    """
    Setup data flow for training, validation, and testing
    """
    train_datagen, val_test_datagen = create_data_generators(train_dir,
val_dir, test_dir)

    train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(224, 224),
        batch_size=batch_size,
        class_mode='binary',
        shuffle=True
    )

    validation_generator = val_test_datagen.flow_from_directory(
        val_dir,
        target_size=(224, 224),
        batch_size=batch_size,
        class_mode='binary',
        shuffle=False
    )

    test_generator = val_test_datagen.flow_from_directory(
        test_dir,
        target_size=(224, 224),
        batch_size=batch_size
```

```
,
        class_mode='binary',
        shuffle=False
    )

    return train_generator, validation_generator, test_generator
```

4.5. Training Configuration

```
def train_model(model_name, model_func, train_gen, val_gen):
    """
    Train model with specified configuration and callbacks
    """
    # Create and compile model
    model = model_func()
    model.compile(
        optimizer=Adam(learning_rate=0.0001),
        loss='binary_crossentropy',
        metrics=['accuracy', 'precision', 'recall']
    )

    # Training callbacks
    callbacks = [
        EarlyStopping(
            monitor='val_accuracy',
            patience=5,
            restore_best_weights=True,
            verbose=1
        ),
        ReduceLROnPlateau(
            monitor='val_loss',
            factor=0.2,
            patience=3,
            min_lr=1e-7,
            verbose=1
        )
    ]

    # Model training
    history = model.fit(
        train_gen,
        epochs=50,
        validation_data=val_gen,
        callbacks=callbacks,
        verbose=1
    )

    return model, history
```

4.6. Evaluation Functions

```
def evaluate_model(model, test_generator):
    """
    Comprehensive model evaluation with multiple metrics
    """
    # Generate predictions
    predictions = model.predict(test_generator)
    y_pred = (predictions > 0.5).astype(int)
    y_true = test_generator.classes

    # Calculate metrics
    from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score

    metrics = {
        'accuracy': accuracy_score(y_true, y_pred),
        'precision': precision_score(y_true, y_pred),
        'recall': recall_score(y_true, y_pred),
        'f1_score': f1_score(y_true, y_pred),
        'auc_roc': roc_auc_score(y_true, predictions)
    }

    return metrics, predictions

def plot_confusion_matrix(y_true, y_pred, model_name):
    """
    Plot confusion matrix for model evaluation
    """
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix - {model_name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

def plot_training_curves(history, model_name):
    """
    Plot training and validation curves
    """
    fig, axes = plt.subplots(2, 2, figsize=(15, 10))

    # Accuracy
    axes[0, 0].plot(history.history['accuracy'], label='Training
Accuracy')
    axes[0, 0].plot(history.history['val_accuracy'], label='Validation
Accuracy')
    axes[0, 0].set_title(f'{model_name} - Accuracy')
    axes[0, 0].legend()

    # Loss
    axes[0, 1].plot(history.history['loss'], label='Training Loss')
    axes[0, 1].plot(history.history['val_loss'], label='Validation
Loss')
    axes[0, 1].set_title(f'{model_name} - Loss')
    axes[0, 1].legend()

    # Precision
    axes[1, 0].plot(history.history['precision'], label='Training
Precision')
    axes[1, 0].plot(history.history['val_precision'], label='Validation
Precision')
    axes[1, 0].set_title(f'{model_name} - Precision')
    axes[1, 0].legend()
```

```
# Recall
    axes[1, 1].plot(history.history['recall'], label='Training Recall')
    axes[1, 1].plot(history.history['val_recall'], label='Validation
Recall')
    axes[1, 1].set_title(f'{model_name} - Recall')
    axes[1, 1].legend()

    plt.tight_layout()
    plt.show()
```

5. Running the Code

The following code demonstrates how to train and evaluate the models:

```
# Setup data paths
train_dir = "path/to/140k_faces/train"
val_dir = "path/to/140k_faces/validation"
test_dir = "path/to/140k_faces/test"

# Create data generators
train_gen, val_gen, test_gen = setup_data_flow(train_dir, val_dir,
test_dir)

# Train models
models_config = {
    'DenseNet121': create_densenet_model,
    'ResNet50': create_resnet_model,
    'EfficientNet-B0': create_efficientnet_model
}

results = {}
for model_name, model_func in models_config.items():
    print(f"\nTraining {model_name}...")
    model, history = train_model(model_name, model_func, train_gen,
val_gen)

    # Evaluate model
    metrics, predictions = evaluate_model(model, test_gen)
    results[model_name] = {
        'model': model,
        'history': history,
        'metrics': metrics,
        'predictions': predictions
    }

    # Plot results
    plot_training_curves(history, model_name)
    y_pred = (predictions > 0.5).astype(int)
    plot_confusion_matrix(test_gen.classes, y_pred, model_name)

    print(f"{model_name} Results:")
    for metric, value in metrics.items():
        print(f"{metric}: {value:.4f}")
```

6. Results

The following are the results obtained from the evaluation of the models on the "140k Real and Fake Faces" dataset.

6.1. Performance Metrics

Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC	Inference Time (ms)
DenseNet121	**94.2%**	**93.8%**	**94.6%**	**94.2%**	**0.987**	**15.3**
ResNet50	91.7%	90.9%	92.5%	91.7%	0.973	12.8
EfficientNet-B0	92.3%	91.7%	93.0%	92.3%	0.979	18.7

6.2. Confusion Matrix

The confusion matrix for the best performing model (DenseNet121) shows excellent classification performance:

```
                Predicted
Actual    Real     Fake
Real      9,847     653
Fake      567     9,933

Classification Accuracy: 94.2%
False Positive Rate: 6.2%
False Negative Rate: 5.4%
```

6.3. Training Curves

Analysis of Training Performance:

DenseNet121: Converged at epoch 23 and very little overfitting

- ResNet50:Training holding on well, converged at 21 epoch.
- EfficientNet-B0: Lasting longer on training and got good generalization.

Key Observations:

1. There was a consistent improvement of all the models in relation to the training.

2. The training accuracy was closely followed by the validation accuracy signifying sound generalization.

3. Overfitting was able to be avoided with early stopping.

4. The scaling back of learning rate assisted in refinement of the model outfits.

The findings testify to the fact that Deepfake detection tasks can reach their state due to the use of DenseNet121 with correct transfer learning and data augmentation.