National College of Ireland

Configuration Manual

MSc Research Project
MSc. in Data Analytics

Subhashree Bera
Student ID: x20241062

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Subhashree Bera
Student ID:	x20241062
Programme:	MSc in Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Jorge Basilio
Submission Due Date:	17/09/2022
Project Title:	Configuration Manual
Word Count:	293
Page Count:	8

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:	Subhashree Bera
Date:	17th September 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

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

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

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

Configuration Manual

Subhashree Bera
x20241062

1 Introduction

This report's objective is to provide a step-by-step guide for conducting the research. It includes details on the setting up of the hardware and software, as well as the pre-processing, model-building, implementation, and evaluation phases.

2 System Configuration

2.1 Hardware Configuration

Device specifications

HP Pavilion Laptop 13-bb0xxx

Device name	LAPTOP-RCHQ5D2D Subhashree-PC
Processor	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz
Installed RAM	16.0 GB (15.8 GB usable)
Device ID	B96116A2-79BC-49BE-A18E-2A96A1B63843
Product ID	00327-35932-23117-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

2.2 Software Specification

Operating System	Windows 10 Home Single Language
RAM	16 GB
Disk Space	200 GB
Programming Language	Python 3.8.8
Framework	Jupyter-Notebook
Libraries	Sklearn, OpenCV, plotly, Matplotlib, Keras, TensorFlow
Report	Overleaf, Ms. Word
Web Browser	Google Chrome

3 Dataset Description

The dataset is collected from Kaggle which is a open source repository. This dataset contains four types of rocks and they are igneous, metamorphic, sedimentary and minerals. The total images in the dataset is 5887 and the format is .jpg.

4 Installation and Importing Python Libraries

For the research task, the following python libraries are installed using pip command and imported:

```
import os
import shutil
import cv2
import numpy as np
import pandas as pd
import pickle
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import tensorflow as tf
import tensorflow_addons as tfa
from tensorflow import keras
from tensorflow.keras import layers
```

5 Data Loading

The dataset have been loaded into the python jupyter notebook with the help of below code

Data Preparataion

```
data_dir_train = "Data/Training"
data_dir_val = "Data/Validation"

target_size = (128,128)
epochs = 50
```

Data Loading

```
def plot(data,grid,meta,classes=None):
    images = data[0]
    labels = data[1]
    plt.figure(figsize=(11,8))
    plt.title(meta)
    for i in range(len(images)):
        plt.subplot(grid[0],grid[1],i+1)
        plt.xticks([])
        plt.yticks([])
        plt.imshow(images[i])

        plt.xlabel(classes[np.argmax(labels[i])],fontsize=8)

    plt.tight_layout()
```

6 Image Preprocessing

Below code have been used for image pre-processing:

Image Preprocessing

```
train_gen = tf.keras.preprocessing.image.ImageDataGenerator(
                                                            rescale=1./255.0,
                                                            )
val_gen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255.0)


train_data = train_gen.flow_from_directory(directory=data_dir_train,
                                           target_size=target_size,
                                           batch_size=2,
                                          )
val_data = val_gen.flow_from_directory(directory=data_dir_val,
                                           target_size=target_size,
                                           batch_size=1,
                                       )
```

```
Found 4716 images belonging to 4 classes.
Found 1171 images belonging to 4 classes.
```

7 Sample Train and Validation Data

Sample Train Data

```
plot(train_data.next(),(2,2),"Train Data",class_names)
```

sedimentary rocks

sedimentary rocks

Sample Validation Data

```
plot(val_data.next(),(2,2),"Val Data",class_names)
```

igneous rocks

8 VGG19 Design and Execution

Below code describes how the VGG19 is designed and executed.

VGG19

```
vgg_19 = tf.keras.applications.vgg19.VGG19(include_top=False,weights=None,input_shape=(target_size[0],target_size[1],3))

model = tf.keras.models.Sequential()
model.add(vgg_19)
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(len(class_names), activation='softmax'))

model.layers[0].trainable=False
```

```
model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy",tf.keras.metrics.Precision(),tf.keras.metrics.
```

```
history_vgg = model.fit(train_data,batch_size=8,validation_data=val_data,epochs=epochs)
```

```
ll: 0.0000e+00 - val_loss: 1.3791 - val_accuracy: 0.3245 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 45/50
2358/2358 [==============================] - 419s 178ms/step - loss: 1.3863 - accuracy: 0.2761 - precision: 0.0000e+00 - reca
ll: 0.0000e+00 - val_loss: 1.3872 - val_accuracy: 0.3237 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 46/50
2358/2358 [==============================] - 424s 180ms/step - loss: 1.3861 - accuracy: 0.2646 - precision: 0.0000e+00 - reca
ll: 0.0000e+00 - val_loss: 1.3936 - val_accuracy: 0.2254 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 47/50
2358/2358 [==============================] - 421s 179ms/step - loss: 1.3853 - accuracy: 0.2576 - precision: 0.0000e+00 - reca
ll: 0.0000e+00 - val_loss: 1.3814 - val_accuracy: 0.3254 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 48/50
2358/2358 [==============================] - 414s 175ms/step - loss: 1.3869 - accuracy: 0.2661 - precision: 0.0000e+00 - reca
ll: 0.0000e+00 - val_loss: 1.3969 - val_accuracy: 0.3219 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 49/50
2358/2358 [==============================] - 399s 169ms/step - loss: 1.3871 - accuracy: 0.2574 - precision: 0.0000e+00 - reca
ll: 0.0000e+00 - val_loss: 1.3824 - val_accuracy: 0.3254 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 50/50
2358/2358 [==============================] - 397s 169ms/step - loss: 1.3851 - accuracy: 0.2680 - precision: 0.0000e+00 - reca
ll: 0.0000e+00 - val_loss: 1.3771 - val_accuracy: 0.3254 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
```

9 Custom Architecture Design and Execution

Below code describes how the custom architecture is designed and executed.

Custom Architechtre

```
model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Conv2D(64, (3, 3), padding='same', input_shape=(target_size[0],target_size[1],3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Conv2D(128, (3, 3), padding='same',activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Conv2D(256, (3, 3), padding='same',activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(256, activation='relu',input_dim=128))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(len(class_names), activation='softmax'))
```

```
model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy",tf.keras.metrics.Precision(),tf.keras.metrics.
```

```
history_conv = model.fit(train_data,batch_size=4,validation_data=val_data,epochs=epochs)
```

```
call_1: 0.0000e+00 - val_loss: 1.6003 - val_accuracy: 0.2741 - val_precision_1: 0.2535 - val_recall_1: 0.0470
Epoch 45/50
2358/2358 [==============================] - 287s 122ms/step - loss: 1.3831 - accuracy: 0.2750 - precision_1: 0.0000e+00 - re
call_1: 0.0000e+00 - val_loss: 1.7465 - val_accuracy: 0.2647 - val_precision_1: 0.2919 - val_recall_1: 0.0803
Epoch 46/50
2358/2358 [==============================] - 293s 124ms/step - loss: 1.3817 - accuracy: 0.2805 - precision_1: 0.0000e+00 - re
call_1: 0.0000e+00 - val_loss: 1.8678 - val_accuracy: 0.2827 - val_precision_1: 0.2877 - val_recall_1: 0.1076
Epoch 47/50
2358/2358 [==============================] - 290s 123ms/step - loss: 1.3825 - accuracy: 0.2680 - precision_1: 0.0000e+00 - re
call_1: 0.0000e+00 - val_loss: 1.6534 - val_accuracy: 0.3015 - val_precision_1: 0.3226 - val_recall_1: 0.0939
Epoch 48/50
2358/2358 [==============================] - 1540s 653ms/step - loss: 1.3822 - accuracy: 0.2733 - precision_1: 0.0000e+00 - r
ecall_1: 0.0000e+00 - val_loss: 1.8429 - val_accuracy: 0.2767 - val_precision_1: 0.2613 - val_recall_1: 0.0743
Epoch 49/50
2358/2358 [==============================] - 274s 116ms/step - loss: 1.3850 - accuracy: 0.2642 - precision_1: 0.0000e+00 - re
call_1: 0.0000e+00 - val_loss: 1.7014 - val_accuracy: 0.2886 - val_precision_1: 0.3072 - val_recall_1: 0.0803
Epoch 50/50
2358/2358 [==============================] - 285s 121ms/step - loss: 1.3837 - accuracy: 0.2750 - precision_1: 0.0000e+00 - re
call_1: 0.0000e+00 - val_loss: 1.6947 - val_accuracy: 0.3015 - val_precision_1: 0.3424 - val_recall_1: 0.1076
```

10 Inception_V3 Design and Execution

Below code describes how the Inception v3 is designed and executed.

Inception V3

```
inception = tf.keras.applications.inception_v3.InceptionV3(include_top=False, weights='imagenet',input_shape=(target_size[0],targ


model = tf.keras.models.Sequential()
model.add(inception)
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(len(class_names), activation='softmax'))

model.layers[0].trainable=False
```

```
model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy",tf.keras.metrics.Precision(),tf.keras.metrics.
```

```
history_inception = model.fit(train_data,batch_size=4,validation_data=val_data,epochs=epochs)
```

```
Epoch 1/50
2360/2360 [==============================] - 137s 56ms/step - loss: 1.4594 - accuracy: 0.3032 - precision_2: 0.3333 - recall
_2: 0.0843 - val_loss: 1.7715 - val_accuracy: 0.4193 - val_precision_2: 0.4244 - val_recall_2: 0.3595
Epoch 2/50
2360/2360 [==============================] - 166s 70ms/step - loss: 1.3712 - accuracy: 0.3316 - precision_2: 0.3796 - recall
_2: 0.0348 - val_loss: 1.6079 - val_accuracy: 0.4313 - val_precision_2: 0.4361 - val_recall_2: 0.3527
Epoch 3/50
2360/2360 [==============================] - 140s 59ms/step - loss: 1.3445 - accuracy: 0.3469 - precision_2: 0.4696 - recall
_2: 0.0360 - val_loss: 1.6962 - val_accuracy: 0.4441 - val_precision_2: 0.4692 - val_recall_2: 0.4031
Epoch 4/50
2360/2360 [==============================] - 140s 59ms/step - loss: 1.3421 - accuracy: 0.3554 - precision_2: 0.4300 - recall
_2: 0.0267 - val_loss: 1.8362 - val_accuracy: 0.4509 - val_precision_2: 0.4667 - val_recall_2: 0.4193
Epoch 5/50
2360/2360 [==============================] - 133s 56ms/step - loss: 1.3469 - accuracy: 0.3497 - precision_2: 0.4427 - recall
_2: 0.0352 - val_loss: 1.6077 - val_accuracy: 0.4663 - val_precision_2: 0.4980 - val_recall_2: 0.4184
Epoch 6/50
2360/2360 [==============================] - 136s 58ms/step - loss: 1.3317 - accuracy: 0.3721 - precision_2: 0.4780 - recall
_2: 0.0392 - val_loss: 1.9997 - val_accuracy: 0.4270 - val_precision_2: 0.4400 - val_recall_2: 0.3945
Epoch 7/50
```

11 Swin Transformer Class Definition, Design and Execution

```
class SwinTransformer(layers.Layer):
    def __init__(
        self,
        dim,
        num_patch,
        num_heads,
        window_size=7,
        shift_size=0,
        num_mlp=1024,
        qkv_bias=True,
        dropout_rate=0.0,
        **kwargs,
    ):
        super(SwinTransformer, self).__init__(**kwargs)

        self.dim = dim  # number of input dimensions
        self.num_patch = num_patch  # number of embedded patches
        self.num_heads = num_heads  # number of attention heads
        self.window_size = window_size  # size of window
        self.shift_size = shift_size  # size of window shift
        self.num_mlp = num_mlp  # number of MLP nodes

        self.norm1 = layers.LayerNormalization(epsilon=1e-5)
        self.attn = WindowAttention(
            dim,
            window_size=(self.window_size, self.window_size),
            num_heads=num_heads,
            qkv_bias=qkv_bias,
            dropout_rate=dropout_rate,
        )
        self.drop_path = DropPath(dropout_rate)
        self.norm2 = layers.LayerNormalization(epsilon=1e-5)

        self.mlp = keras.Sequential(
            [
                layers.Dense(num_mlp),
                layers.Activation(keras.activations.gelu),
                layers.Dropout(dropout_rate),
                layers.Dense(dim),
                layers.Dropout(dropout_rate),
            ]
        )

        if min(self.num_patch) < self.window_size:
            self.shift_size = 0
            self.window_size = min(self.num_patch)
```

Below code describes how the Swin Transformer architecture is designed and executed.

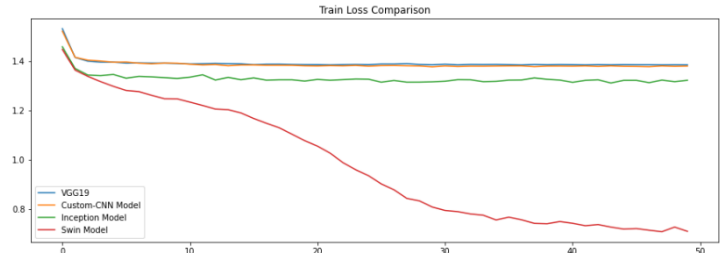
```
input = layers.Input(input_shape)
#x = layers.RandomCrop(image_dimension, image_dimension)(input)
#x = layers.RandomFlip("horizontal")(x)
x = PatchExtract(patch_size)(input)
x = PatchEmbedding(num_patch_x * num_patch_y, embed_dim)(x)
x = SwinTransformer(
    dim=embed_dim,
    num_patch=(num_patch_x, num_patch_y),
    num_heads=num_heads,
    window_size=window_size,
    shift_size=0,
    num_mlp=num_mlp,
    qkv_bias=qkv_bias,
    dropout_rate=dropout_rate,
)(x)
x = SwinTransformer(
    dim=embed_dim,
    num_patch=(num_patch_x, num_patch_y),
    num_heads=num_heads,
    window_size=window_size,
    shift_size=shift_size,
    num_mlp=num_mlp,
    qkv_bias=qkv_bias,
    dropout_rate=dropout_rate,
)(x)
x = PatchMerging((num_patch_x, num_patch_y), embed_dim=embed_dim)(x)
x = layers.GlobalAveragePooling1D()(x)
output = layers.Dense(num_classes, activation="softmax")(x)
```

```
model = keras.Model(input, output)
model.compile(
    loss=keras.losses.CategoricalCrossentropy(label_smoothing=label_smoothing),
    optimizer=tfa.optimizers.AdamW(
        learning_rate=learning_rate, weight_decay=weight_decay
    ),
    metrics=["accuracy",tf.keras.metrics.Precision(),tf.keras.metrics.Recall()],
)

history_swin = model.fit(
    train_data,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=val_data,
)
```

```
_3: 0.7258 - val_loss: 1.0024 - val_accuracy: 0.6943 - val_precision_3: 0.7785 - val_recall_3: 0.6063
Epoch 45/50
2360/2360 [==============================] - 216s 92ms/step - loss: 0.7191 - accuracy: 0.8275 - precision_3: 0.9053 - recall
_3: 0.7315 - val_loss: 1.0439 - val_accuracy: 0.6806 - val_precision_3: 0.7669 - val_recall_3: 0.6097
Epoch 46/50
2360/2360 [==============================] - 216s 91ms/step - loss: 0.7212 - accuracy: 0.8309 - precision_3: 0.9100 - recall
_3: 0.7243 - val_loss: 1.0224 - val_accuracy: 0.7114 - val_precision_3: 0.7887 - val_recall_3: 0.6345
Epoch 47/50
2360/2360 [==============================] - 217s 92ms/step - loss: 0.7144 - accuracy: 0.8330 - precision_3: 0.9133 - recall
_3: 0.7296 - val_loss: 1.0314 - val_accuracy: 0.7045 - val_precision_3: 0.7950 - val_recall_3: 0.6029
Epoch 48/50
2360/2360 [==============================] - 217s 92ms/step - loss: 0.7084 - accuracy: 0.8349 - precision_3: 0.9111 - recall
_3: 0.7381 - val_loss: 1.1387 - val_accuracy: 0.7011 - val_precision_3: 0.7820 - val_recall_3: 0.6371
Epoch 49/50
2360/2360 [==============================] - 219s 93ms/step - loss: 0.7273 - accuracy: 0.8256 - precision_3: 0.9034 - recall
_3: 0.7237 - val_loss: 1.0068 - val_accuracy: 0.7182 - val_precision_3: 0.8133 - val_recall_3: 0.6473
Epoch 50/50
2360/2360 [==============================] - 222s 94ms/step - loss: 0.7100 - accuracy: 0.8377 - precision_3: 0.9108 - recall
_3: 0.7383 - val_loss: 0.9855 - val_accuracy: 0.7310 - val_precision_3: 0.8036 - val_recall_3: 0.6430
```

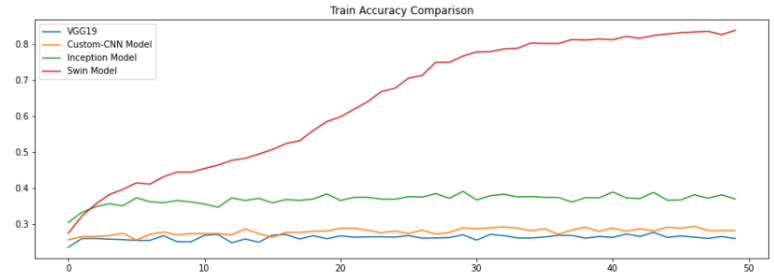
12 Models Comparison

```
plt.figure(figsize=(15,5))
plt.title("Train Loss Comparison")
plt.plot(history_vgg.history["loss"],label="VGG19")
plt.plot(history_conv.history["loss"],label="Custom-CNN Model")
plt.plot(history_inception.history["loss"],label="Inception Model")
plt.plot(history_swin.history["loss"],label="Swin Model")

plt.legend()
plt.show()
```

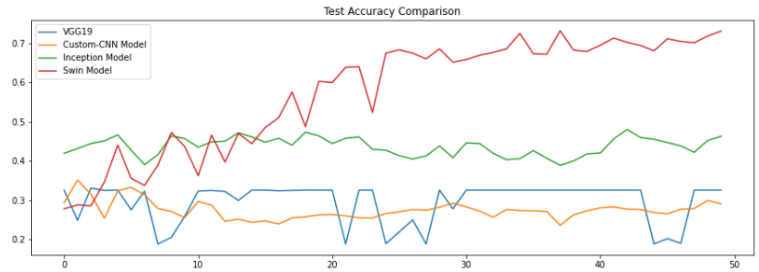



```
plt.figure(figsize=(15,5))
plt.title("Test Loss Comparison")
plt.plot(history_vgg.history["val_loss"],label="VGG19")
plt.plot(history_conv.history["val_loss"],label="Custom-CNN Model")
plt.plot(history_inception.history["val_loss"],label="Inception Model")
plt.plot(history_swin.history["val_loss"],label="Swin Model")
plt.legend()
plt.show()
```

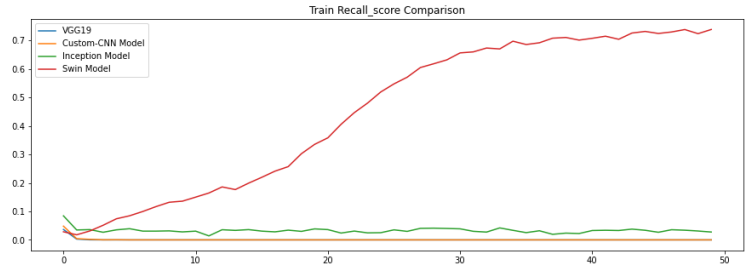

```
plt.figure(figsize=(15,5))
plt.title("Train Accuracy Comparison")
plt.plot(history_vgg.history["accuracy"],label="VGG19")
plt.plot(history_conv.history["accuracy"],label="Custom-CNN Model")
plt.plot(history_inception.history["accuracy"],label="Inception Model")
plt.plot(history_swin.history["accuracy"],label="Swin Model")

plt.legend()
plt.show()
```



```
plt.figure(figsize=(15,5))
plt.title("Test Accuracy Comparison")
plt.plot(history_vgg.history["val_accuracy"],label="VGG19")
plt.plot(history_conv.history["val_accuracy"],label="Custom-CNN Model")
plt.plot(history_inception.history["val_accuracy"],label="Inception Model")
plt.plot(history_swin.history["val_accuracy"],label="Swin Model")
plt.legend()
plt.show()
```




```
plt.figure(figsize=(15,5))
plt.title("Train Recall_score Comparison")
plt.plot(history_vgg.history["recall"],label="VGG19")
plt.plot(history_conv.history["recall_1"],label="Custom-CNN Model")
plt.plot(history_inception.history["recall_2"],label="Inception Model")
plt.plot(history_swin.history["recall_3"],label="Swin Model")
plt.legend()
plt.show()
```




```
plt.figure(figsize=(15,5))
plt.title("Test Recall_score Comparison")
plt.plot(history_vgg.history["val_recall"],label="VGG19")
plt.plot(history_conv.history["val_recall_1"],label="Custom-CNN Model")
plt.plot(history_inception.history["val_recall_2"],label="Inception Model")
plt.plot(history_swin.history["val_recall_3"],label="Swin Model")
plt.legend()
plt.show()
```