VIETNAM NATIONAL UNIVERSITY – HCM

INTERNATIONAL UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

IT157IU

DEEP LEARNING

Topic name:

Comparative Analysis of Deep Learning Models for Hair Disease Classification

Group member

Name	ID	Contribution (%)
Phạm Vũ Tuyết Anh	ITDSIU21073	25 %
Đào Ngọc Lan Hồng	ITDSIU21088	25 %
Đinh Vũ Ngọc Linh	ITDSIU21095	25 %
Trần Triệu Như	ITDSIU21029	25 %

Table of Contents

TABLE OF FIGURES

Acknowledgement

The team would like to express our deepest gratitude to Dr. Mai Hoang Bao An for their invaluable guidance, support, and expertise throughout the course of this project. His insightful feedback, encouragement, and thoughtful suggestions were crucial in shaping the direction of our work. The team is truly grateful for the opportunity to work under his mentorship and for the knowledge we have gained from his expertise.

In addition to the specialized knowledge imparted during the lectures, we deeply appreciate the invaluable practical experiences Dr. Mai Hoang Bao An shared throughout both the theoretical and practical sessions. These insights have greatly helped us apply the knowledge effectively to our project, enriching our ability to translate theory into real-world practice.

Due to the limitations in knowledge and time, the project may still have some shortcomings. The team looks forward to receiving feedback from Dr. Mai Hoang Bao An to further improve and refine our work. We would also like to express our heartfelt gratitude for the support and valuable contributions provided by Dr. Mai Hoang Bao An throughout our learning process.

I. Introduction

1. Background

Hair and scalp diseases are quite common nowadays, affecting people of all ages. Conditions such as alopecia areata, dermatitis, head lice, scalp fungus, and male pattern baldness not only cause physical discomfort but also have a significant psychological impact. In modern life, factors like stress, hormonal changes, environmental pollution, unhealthy lifestyles, and genetics have all contributed to an increasing number of cases. Although these conditions are not always dangerous, if left undiagnosed or untreated, they can become more serious and have long-term effects. However, the diagnosis of hair diseases still largely depends on the clinical experience of dermatologists, which can be subjective and vary between practitioners.

In this context, the development of automated and accurate diagnostic support systems for hair diseases is becoming increasingly essential. Deep learning techniques, particularly Convolutional Neural Networks (CNNs), have emerged as powerful tools in the analysis of medical images, including those of hair and scalp. The application of deep learning not only automates the diagnostic process but also enhances accuracy, saves time, and reduces the burden on healthcare professionals. However, there is still a lack of comprehensive studies comparing deep learning models for multi-class classification of hair diseases. Therefore, the research titled "Comparative Analysis of Deep Learning Models for Hair Disease Classification" was conducted to evaluate and compare the performance of architectures such as Simple CNN, VGG16, VGG19, Xception, ResNet50, and MobileNetV2 on a dataset consisting of images of various hair conditions. The results of this study aim to contribute to the development of more accurate and reliable diagnostic support systems for hair diseases in clinical practice.

2. Objectives

This project aims to build a comprehensive evaluation framework for comparing the performance of state-of-the-art deep learning models in the automatic classification of hair and scalp diseases using clinical images. It focuses on common conditions such as alopecia areata, seborrheic dermatitis, and tinea capitis. The study explores a range of model architectures such as Simple CNN, VGG16, VGG19, Xception, ResNet50, and MobileNetV2—to identify which one offers the most dependable diagnostic results. Accuracy is a key metric, as it reflects how well the model can correctly identify diseases. Ultimately, the goal is to propose an AI-powered diagnostic model that can assist dermatologists in making faster and more consistent decisions, while also improving access to early detection in areas with limited healthcare resources.

3. Scope

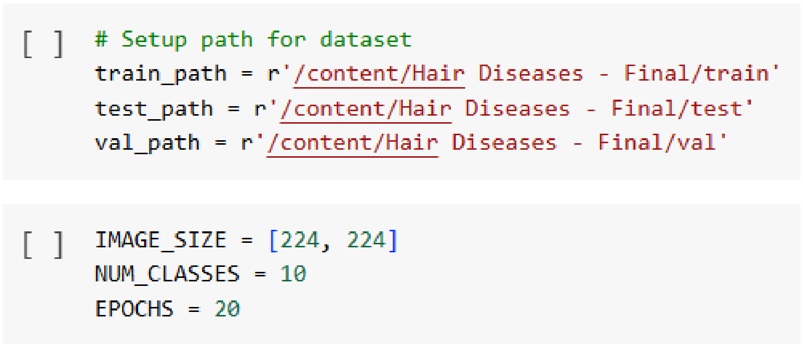
This project specifically focuses on image-based classification of hair and scalp diseases using Convolutional Neural Networks (CNNs). It does not extend to more complex tasks such as image segmentation, lesion localization, or the integration of multi-modal data like patient history or clinical metadata. All experiments are conducted using the publicly available Kaggle Hair Diseases dataset, which provides standardized, labeled images for training and evaluation. The scope is confined to comparing the performance of selected CNN architectures under consistent conditions, with the aim of identifying a model that balances accuracy, efficiency, and practical applicability in real-world diagnostic scenarios.

II. Model Development

1. Libraries used

```
from keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Lambda
from keras.models import Model, load_model
from tensorflow.keras import layers, models
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
import tensorflow as tf
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Input
from glob import glob
from keras.applications.vgg16 import VGG16
from keras.applications.vgg19 import VGG19
from keras.applications.vgg19 import preprocess_input
from keras.applications.mobilenet_v2 import MobileNetV2
from keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.applications import Xception
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Input
from tensorflow.python.client import device_lib
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc,
precision_recall_curve, average_precision_score, ConfusionMatrixDisplay
```

2. Define path

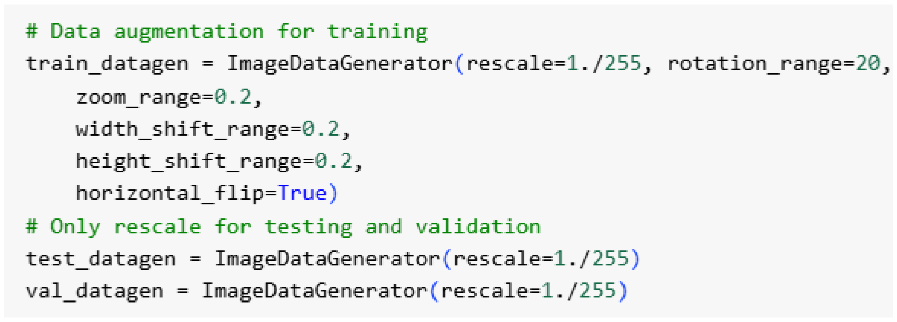


```
[ ]  # Setup path for dataset
     train_path = r'/content/Hair Diseases - Final/train'
     test_path = r'/content/Hair Diseases - Final/test'
     val_path = r'/content/Hair Diseases - Final/val'
```

```
[ ]  IMAGE_SIZE = [224, 224]
     NUM_CLASSES = 10
     EPOCHS = 20
```

Figure 2.1 Define path and configuration parameters

3. Load data

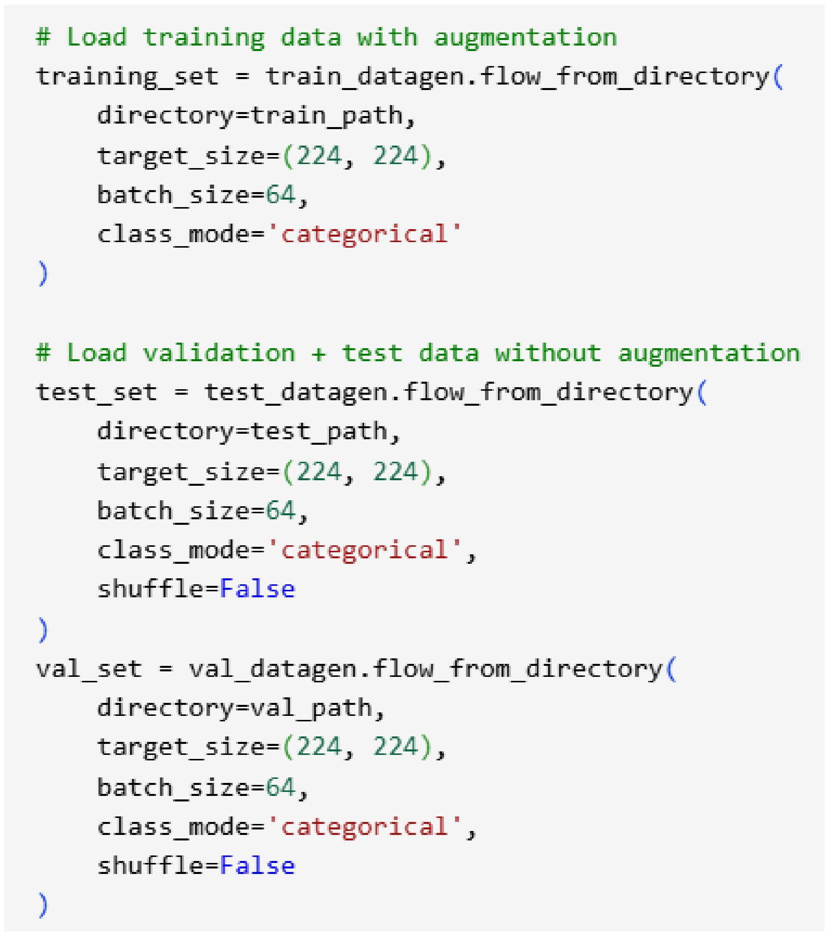


```
# Data augmentation for training
train_datagen = ImageDataGenerator(rescale=1./255, rotation_range=20,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True)
# Only rescale for testing and validation
test_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)
```

Figure 2.2 Prepare and handle input image

For the training set, ImageDataGenerator is used to apply data augmentation techniques such as rotation, zoom, shifting, and horizontal flipping. These transformations help the model learn better, reduce overfitting, and improve generalization. Additionally, all images are normalized to the [0, 1] range to enhance training efficiency.

In contrast, for the test and validation sets, only normalization is applied without any augmentation. This ensures that model evaluation is accurate and objective, reflecting its true performance on real, unaltered data.

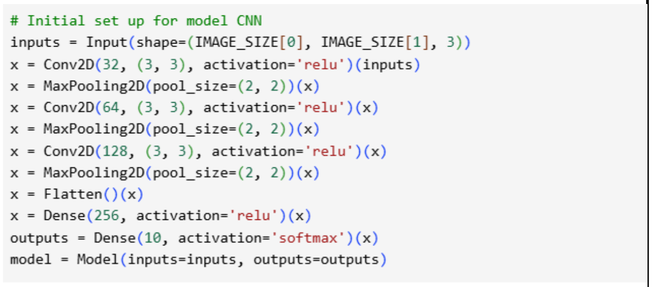
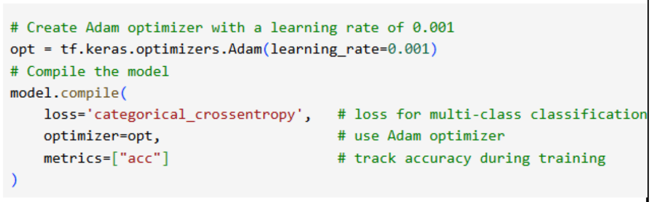
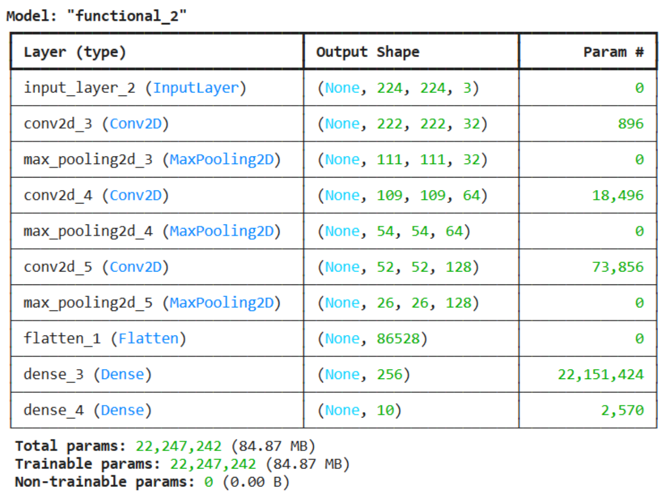
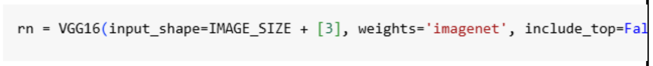
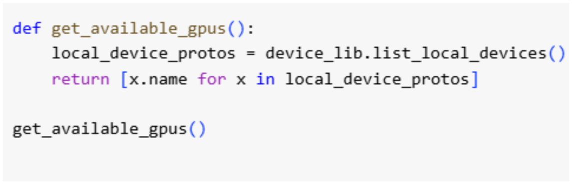
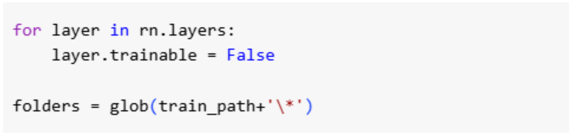
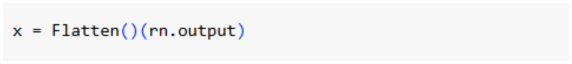
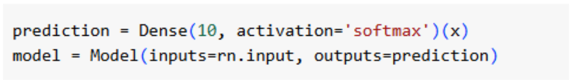
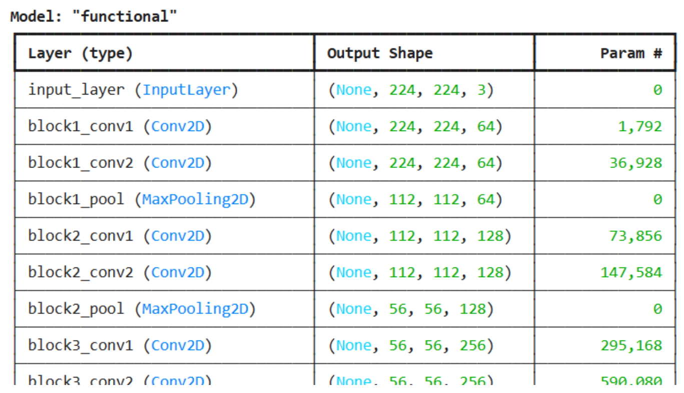
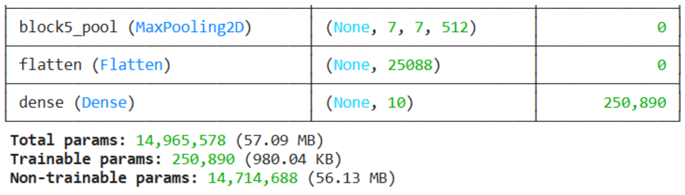


```
# Load training data with augmentation
training_set = train_datagen.flow_from_directory(
    directory=train_path,
    target_size=(224, 224),
    batch_size=64,
    class_mode='categorical'
)

# Load validation + test data without augmentation
test_set = test_datagen.flow_from_directory(
    directory=test_path,
    target_size=(224, 224),
    batch_size=64,
    class_mode='categorical',
    shuffle=False
)
val_set = val_datagen.flow_from_directory(
    directory=val_path,
    target_size=(224, 224),
    batch_size=64,
    class_mode='categorical',
    shuffle=False
)
```

Figure 2.3 Loading data

Read images from the training, validation, and test directories, then resize, normalize, and divide them into batches suitable for deep learning models. Training images are augmented to enhance learning performance, while validation and test images are only normalized to ensure more accurate model evaluation.

4. Create model

Model	Create model	Output
Simple CNN	 	
VGG16	    	 

Simple CNN — Create model

```python
# Initial set up for model CNN
inputs = Input(shape=(IMAGE_SIZE[0], IMAGE_SIZE[1], 3))
x = Conv2D(32, (3, 3), activation='relu')(inputs)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(128, (3, 3), activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
outputs = Dense(10, activation='softmax')(x)
model = Model(inputs=inputs, outputs=outputs)
```

```python
# Create Adam optimizer with a learning rate of 0.001
opt = tf.keras.optimizers.Adam(learning_rate=0.001)
# Compile the model
model.compile(
    loss='categorical_crossentropy',    # loss for multi-class classification
    optimizer=opt,                      # use Adam optimizer
    metrics=["acc"]                     # track accuracy during training
)
```

Simple CNN — Output

```
Model: "functional_2"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                      ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ input_layer_2 (InputLayer)        │ (None, 224, 224, 3)    │             0 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_3 (Conv2D)                 │ (None, 222, 222, 32)   │           896 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_3 (MaxPooling2D)    │ (None, 111, 111, 32)   │             0 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_4 (Conv2D)                 │ (None, 109, 109, 64)   │        18,496 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_4 (MaxPooling2D)    │ (None, 54, 54, 64)     │             0 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_5 (Conv2D)                 │ (None, 52, 52, 128)    │        73,856 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_5 (MaxPooling2D)    │ (None, 26, 26, 128)    │             0 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten_1 (Flatten)               │ (None, 86528)          │             0 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                   │ (None, 256)            │    22,151,424 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_4 (Dense)                   │ (None, 10)             │         2,570 │
└───────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 22,247,242 (84.87 MB)
 Trainable params: 22,247,242 (84.87 MB)
 Non-trainable params: 0 (0.00 B)
```

VGG16 — Create model

```python
rn = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=Fal
```

```python
def get_available_gpus():
    local_device_protos = device_lib.list_local_devices()
    return [x.name for x in local_device_protos]

get_available_gpus()
```

```python
for layer in rn.layers:
    layer.trainable = False

folders = glob(train_path+'\*')
```

```python
x = Flatten()(rn.output)
```

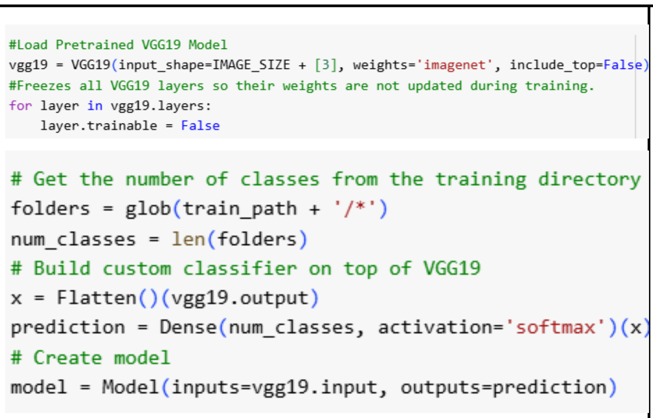
```python
prediction = Dense(10, activation='softmax')(x)
model = Model(inputs=rn.input, outputs=prediction)
```

VGG16 — Output

```
Model: "functional"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                      ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ input_layer (InputLayer)          │ (None, 224, 224, 3)    │             0 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ block1_conv1 (Conv2D)             │ (None, 224, 224, 64)   │         1,792 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ block1_conv2 (Conv2D)             │ (None, 224, 224, 64)   │        36,928 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ block1_pool (MaxPooling2D)        │ (None, 112, 112, 64)   │             0 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ block2_conv1 (Conv2D)             │ (None, 112, 112, 128)  │        73,856 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ block2_conv2 (Conv2D)             │ (None, 112, 112, 128)  │       147,584 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ block2_pool (MaxPooling2D)        │ (None, 56, 56, 128)    │             0 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ block3_conv1 (Conv2D)             │ (None, 56, 56, 256)    │       295,168 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ block3_conv2 (Conv2D)             │ (None, 56, 56, 256)    │       590,080 │
```

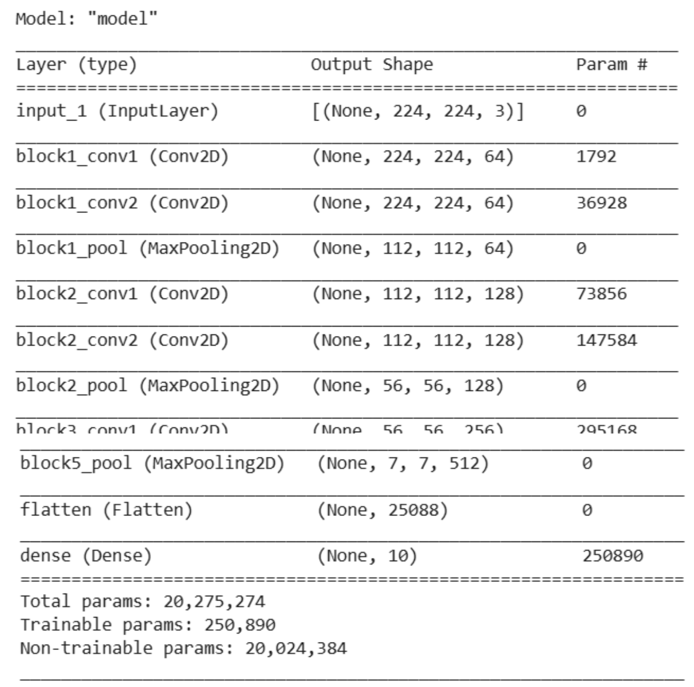
```
│ block5_pool (MaxPooling2D)        │ (None, 7, 7, 512)      │             0 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)                 │ (None, 25088)          │             0 │
├───────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                     │ (None, 10)             │       250,890 │
└───────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 14,965,578 (57.09 MB)
 Trainable params: 250,890 (980.04 KB)
 Non-trainable params: 14,714,688 (56.13 MB)
```

VGG19



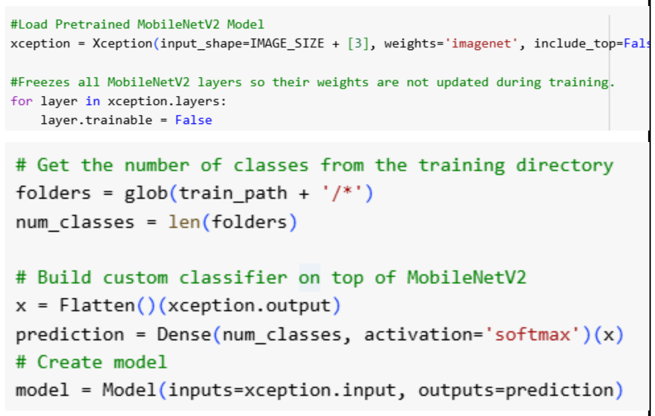
```
#Load Pretrained VGG19 Model
vgg19 = VGG19(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
#Freezes all VGG19 layers so their weights are not updated during training.
for layer in vgg19.layers:
    layer.trainable = False
```

```
# Get the number of classes from the training directory
folders = glob(train_path + '/*')
num_classes = len(folders)
# Build custom classifier on top of VGG19
x = Flatten()(vgg19.output)
prediction = Dense(num_classes, activation='softmax')(x)
# Create model
model = Model(inputs=vgg19.input, outputs=prediction)
```



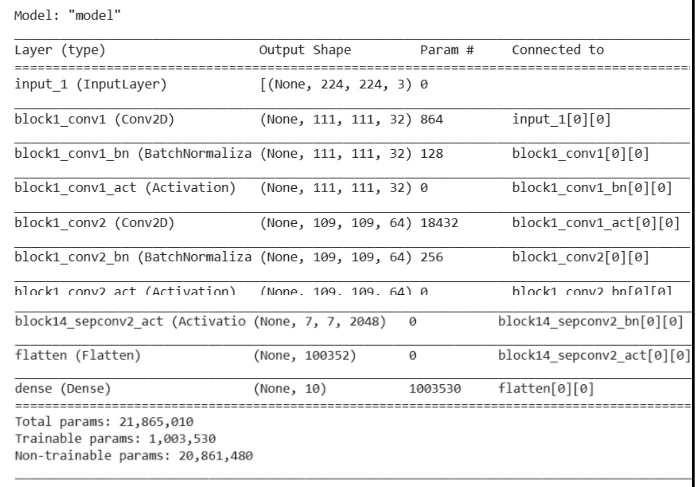
```
Model: "model"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_1 (InputLayer)        [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168
_________________________________________________________________
block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)           (None, 25088)             0
_________________________________________________________________
dense (Dense)               (None, 10)                250890
=================================================================
Total params: 20,275,274
Trainable params: 250,890
Non-trainable params: 20,024,384
_________________________________________________________________
```

Xception



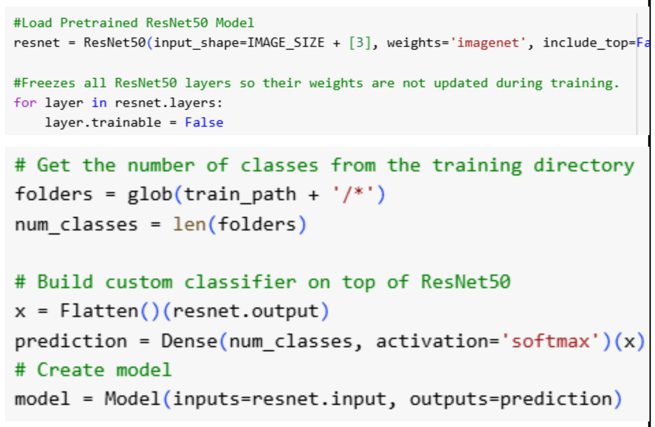
```
#Load Pretrained MobileNetV2 Model
xception = Xception(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=Fals

#Freezes all MobileNetV2 layers so their weights are not updated during training.
for layer in xception.layers:
    layer.trainable = False
```

```
# Get the number of classes from the training directory
folders = glob(train_path + '/*')
num_classes = len(folders)

# Build custom classifier on top of MobileNetV2
x = Flatten()(xception.output)
prediction = Dense(num_classes, activation='softmax')(x)
# Create model
model = Model(inputs=xception.input, outputs=prediction)
```



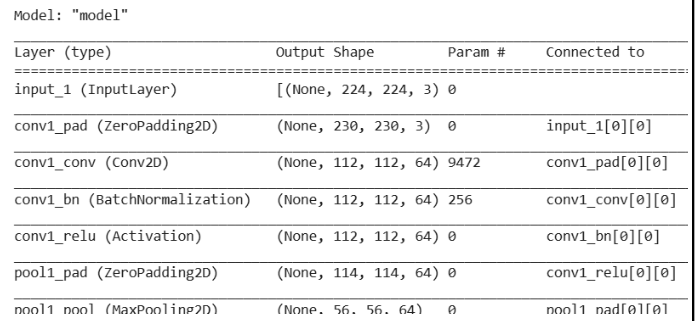
```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
block1_conv1 (Conv2D)           (None, 111, 111, 32) 864         input_1[0][0]
__________________________________________________________________________________________________
block1_conv1_bn (BatchNormaliza (None, 111, 111, 32) 128         block1_conv1[0][0]
__________________________________________________________________________________________________
block1_conv1_act (Activation)   (None, 111, 111, 32) 0           block1_conv1_bn[0][0]
__________________________________________________________________________________________________
block1_conv2 (Conv2D)           (None, 109, 109, 64) 18432       block1_conv1_act[0][0]
__________________________________________________________________________________________________
block1_conv2_bn (BatchNormaliza (None, 109, 109, 64) 256         block1_conv2[0][0]
__________________________________________________________________________________________________
block1_conv2_act (Activation)   (None, 109, 109, 64) 0           block1_conv2_bn[0][0]
__________________________________________________________________________________________________
block14_sepconv2_act (Activatio (None, 7, 7, 2048)   0           block14_sepconv2_bn[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 100352)       0           block14_sepconv2_act[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 10)           1003530     flatten[0][0]
==================================================================================================
Total params: 21,865,010
Trainable params: 1,003,530
Non-trainable params: 20,861,480
__________________________________________________________________________________________________
```

ResNet

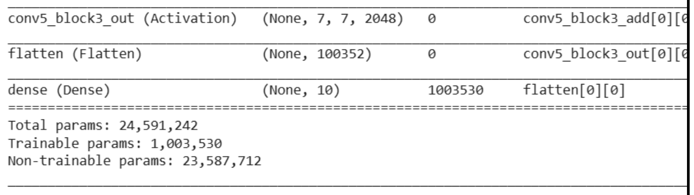
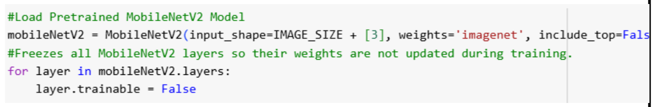
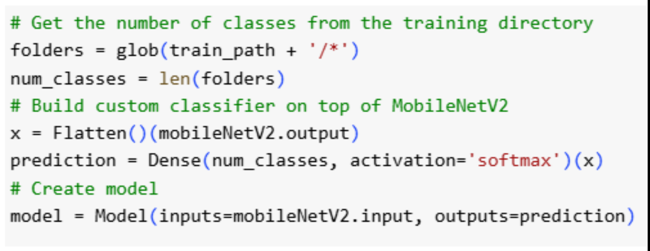
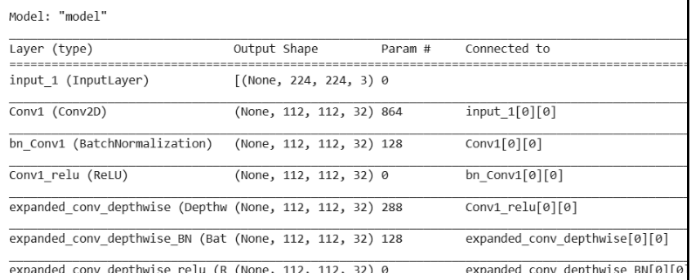
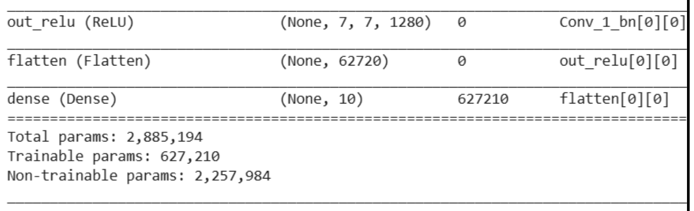


```
#Load Pretrained ResNet50 Model
resnet = ResNet50(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=Fa

#Freezes all ResNet50 layers so their weights are not updated during training.
for layer in resnet.layers:
    layer.trainable = False
```

```
# Get the number of classes from the training directory
folders = glob(train_path + '/*')
num_classes = len(folders)

# Build custom classifier on top of ResNet50
x = Flatten()(resnet.output)
prediction = Dense(num_classes, activation='softmax')(x)
# Create model
model = Model(inputs=resnet.input, outputs=prediction)
```



```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
conv1_pad (ZeroPadding2D)       (None, 230, 230, 3)  0           input_1[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]
__________________________________________________________________________________________________
conv1_relu (Activation)         (None, 112, 112, 64) 0           conv1_bn[0][0]
__________________________________________________________________________________________________
pool1_pad (ZeroPadding2D)       (None, 114, 114, 64) 0           conv1_relu[0][0]
__________________________________________________________________________________________________
pool1_pool (MaxPooling2D)       (None, 56, 56, 64)   0           pool1_pad[0][0]
```

		
MobileNet V2	 	 

```
conv5_block3_out (Activation)   (None, 7, 7, 2048)   0          conv5_block3_add[0][0
flatten (Flatten)               (None, 100352)       0          conv5_block3_out[0][0
dense (Dense)                   (None, 10)           1003530    flatten[0][0]
==================================================================================
Total params: 24,591,242
Trainable params: 1,003,530
Non-trainable params: 23,587,712
```

```
#Load Pretrained MobileNetV2 Model
mobileNetV2 = MobileNetV2(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=Fals
#Freezes all MobileNetV2 layers so their weights are not updated during training.
for layer in mobileNetV2.layers:
    layer.trainable = False
```

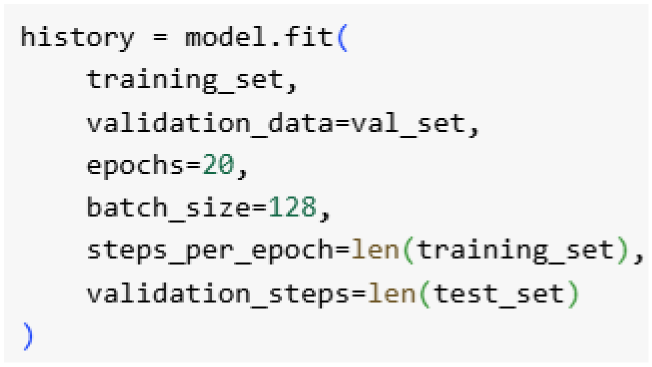
```
# Get the number of classes from the training directory
folders = glob(train_path + '/*')
num_classes = len(folders)
# Build custom classifier on top of MobileNetV2
x = Flatten()(mobileNetV2.output)
prediction = Dense(num_classes, activation='softmax')(x)
# Create model
model = Model(inputs=mobileNetV2.input, outputs=prediction)
```

```
Model: "model"

Layer (type)                 Output Shape          Param #    Connected to
==================================================================================
input_1 (InputLayer)         [(None, 224, 224, 3) 0

Conv1 (Conv2D)               (None, 112, 112, 32) 864        input_1[0][0]

bn_Conv1 (BatchNormalization) (None, 112, 112, 32) 128        Conv1[0][0]

Conv1_relu (ReLU)            (None, 112, 112, 32) 0          bn_Conv1[0][0]

expanded_conv_depthwise (Depthw (None, 112, 112, 32) 288     Conv1_relu[0][0]

expanded_conv_depthwise_BN (Bat (None, 112, 112, 32) 128     expanded_conv_depthwise[0][0]

expanded_conv_depthwise_relu (R (None, 112, 112, 32) 0       expanded_conv_depthwise_BN[0][0
```

```
out_relu (ReLU)                 (None, 7, 7, 1280)   0          Conv_1_bn[0][0]

flatten (Flatten)               (None, 62720)        0          out_relu[0][0]

dense (Dense)                   (None, 10)           627210     flatten[0][0]
==================================================================================
Total params: 2,885,194
Trainable params: 627,210
Non-trainable params: 2,257,984
```

5. Train model

❖ *Simple CNN*

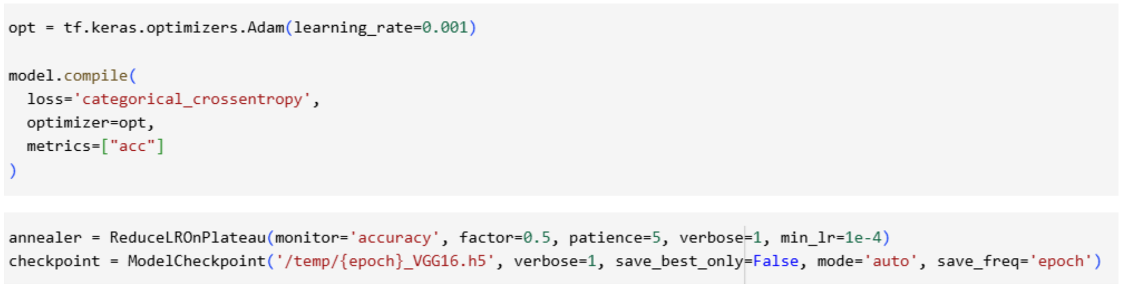


```
history = model.fit(
    training_set,
    validation_data=val_set,
    epochs=20,
    batch_size=128,
    steps_per_epoch=len(training_set),
    validation_steps=len(test_set)
)
```

Figure 2.4 Train model Simple CNN

Train a simple CNN model on the training image set while evaluating its performance on the validation set after each epoch. The results are stored in the history variable for later use in plotting or analysis.

❖ *Advanced models (VGG16, VGG19, Xception, ResNet50, and MobileNetV2)*

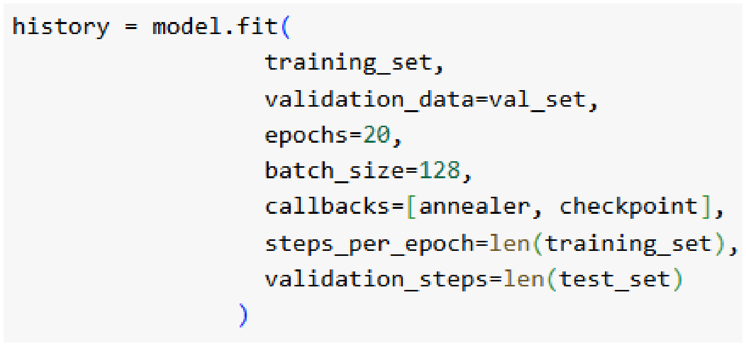


```
opt = tf.keras.optimizers.Adam(learning_rate=0.001)

model.compile(
  loss='categorical_crossentropy',
  optimizer=opt,
  metrics=["acc"]
)
```

```
annealer = ReduceLROnPlateau(monitor='accuracy', factor=0.5, patience=5, verbose=1, min_lr=1e-4)
checkpoint = ModelCheckpoint('/temp/{epoch}_VGG16.h5', verbose=1, save_best_only=False, mode='auto', save_freq='epoch')
```

Figure 2.5 Optimize, compile, and set callbacks

The code sets up training using the Adam optimizer (learning rate 0.001) and the categorical_crossentropy loss function for multi-class classification. It also uses two callbacks: ReduceLROnPlateau to reduce the learning rate if the model doesn't improve after 5 epochs, and ModelCheckpoint to save the model after each epoch for monitoring and recovery.

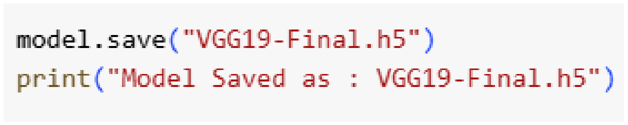


```
history = model.fit(
                    training_set,
                    validation_data=val_set,
                    epochs=20,
                    batch_size=128,
                    callbacks=[annealer, checkpoint],
                    steps_per_epoch=len(training_set),
                    validation_steps=len(test_set)
                   )
```

Figure 2.6 Train model VGG16, VGG19, Xception, ResNet50, and MobileNetV2

Train the VGG16, VGG19, Xception, ResNet50, and MobileNetV2 model for 20 epochs using augmented data, while monitoring validation accuracy. The learning rate is automatically adjusted and the model is saved after each epoch to optimize training and preserve model versions for later evaluation or use.

6. Save model



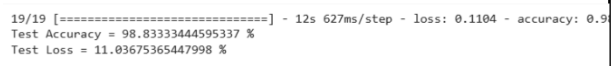
```
model.save("VGG19-Final.h5")
print("Model Saved as : VGG19-Final.h5")
```

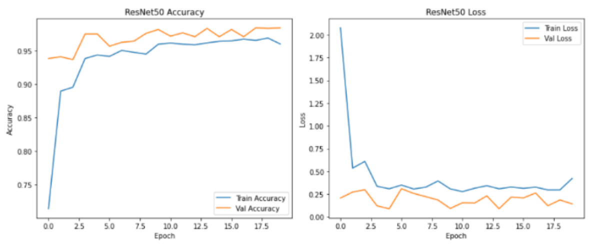
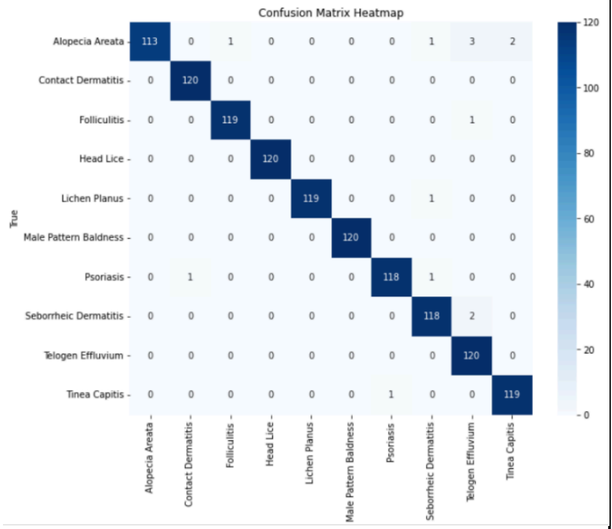
Figure 2.7 Save model

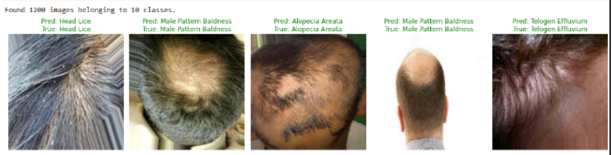
This code is used to save the entire trained model to a file named after each model, allowing for reuse or deployment later without the need for retraining.

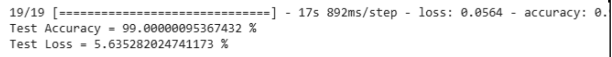
III. Model Evaluation

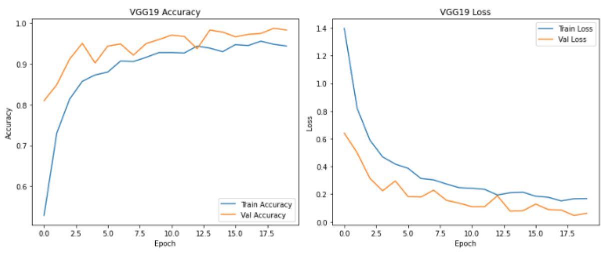
Model			Total (Pros & cos)

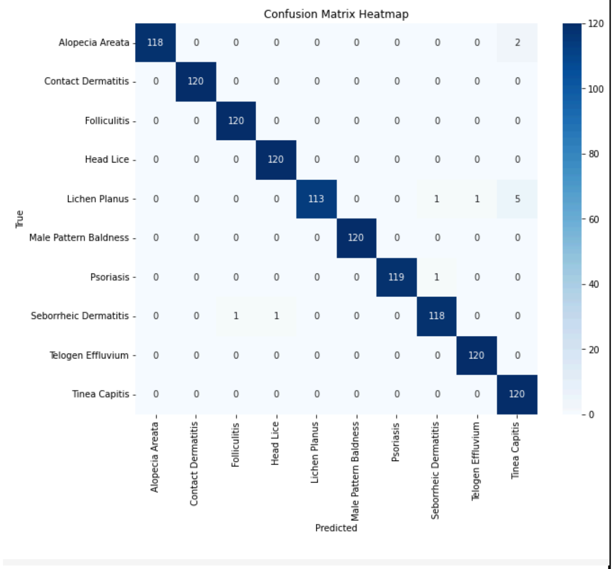
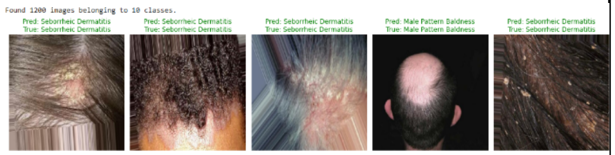
Xception	**Test Set Performance**
	
	`19/19 [==============================] - 12s 627ms/step - loss: 0.1104 - accuracy: 0.9`
	`Test Accuracy = 98.83333444595337 %`
	`Test Loss = 11.03675365447998 %`
		Xception correctly classifies 98.83 % of unseen images, indicating it has learned highly discriminative features and generalizes extremely well.	Xception delivers near-99 % accuracy on hair-disease classification by using depthwise separable convolutions that efficiently capture both fine textures and global patterns, converging rapidly with stable training and minimal overfitting. However, its ~23.5 M-parameter size and ~35 ms inference time demand substantial compute and memory—making it less suitable for resource-constrained or real-time edge deployments—and its marginal accuracy gains over simpler architectures may not always justify the added complexity.

	Training & Validation Curves (Chup lai sua ten plot) 	Converges very quickly, with validation accuracy exceeding 95 % by epoch 5. The tight overlap of training and validation curves shows minimal overfitting. Steady decline in validation loss confirms robust learning throughout.	
	Confusion Matrix 	Xception achieves near-perfect discrimination with an overall misclassification rate of ~1.2% (14/1200). Errors occur mainly between visually similar conditions, for example:Alopecia Areata vs. Telogen Effluvium (3 cases, 2.5%),Alopecia Areata vs. Tinea Capitis (2 cases, 1.7%),Seborrheic Dermatitis vs. Telogen Effluvium (2 cases, 1.7%) These small numbers show the model captures unique lesion features very effectively and only confuses conditions whose presentations are almost identical.	

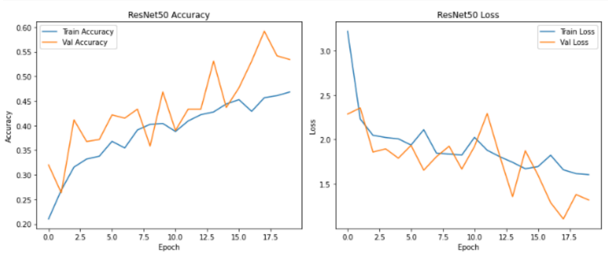
	Sample Predictions `Found 1200 images belonging to 10 classes.` Pred: Head Lice True: Head Lice Pred: Male Pattern Baldness True: Male Pattern Baldness Pred: Alopecia Areata True: Alopecia Areata Pred: Male Pattern Baldness True: Male Pattern Baldness Pred: Telogen Effluvium True: Telogen Effluvium	All five samples are correctly labeled, demonstrating that Xception generalizes robustly across varied conditions—different lesion types, lighting, camera angles, and scalp textures—underscoring its suitability for real-world clinical screening.	

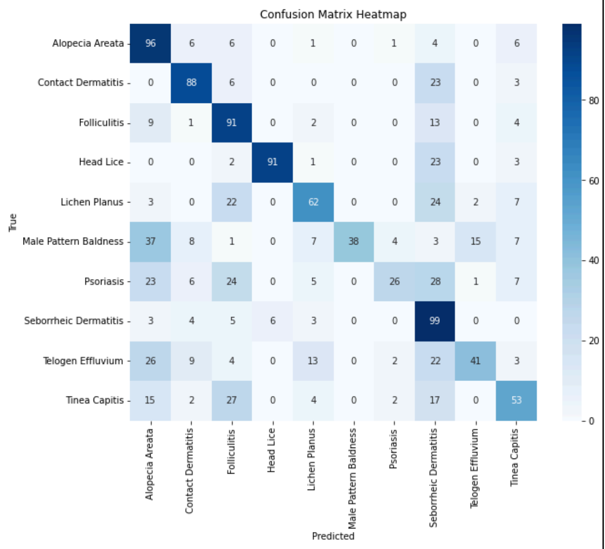
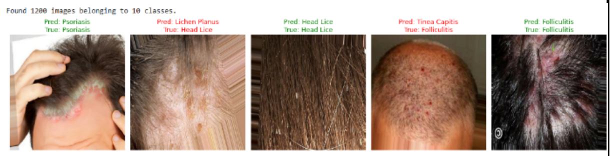
VGG 19	**Test Set Performance**
	`19/19 [==============================] - 17s 892ms/step - loss: 0.0564 - accuracy: 0.`
	`Test Accuracy = 99.00000095367432 %`
	`Test Loss = 5.635282024741173 %`
		VGG19 correctly classifies 1188/1200 test images, showing that ImageNet-pretrained filters transfer exceptionally well to hair-disease patterns**.**	VGG19 achieves near-perfect accuracy and stable training dynamics, expertly leveraging pretrained features to distinguish subtle scalp conditions. However, its large footprint (~138 M parameters, ~46 ms/inference) and slight sensitivity to hyperparameters make it less ideal for real-time or resource-limited deployments despite its excellent performance.

	Training & Validation Curves 	Validation accuracy exceeded 90 % by epoch 3, demonstrating rapid initial learning. By epoch 15, both training and validation accuracies stabilized between 98 % and 99 %, indicating peak performance. The near-overlap of training and validation curves, together with a smooth decline in validation loss, shows minimal overfitting.	
	Confusion Matrix	The confusion matrix shows the largest error cluster is Lichen Planus → Tinea Capitis (5 cases), followed by Alopecia Areata → Tinea Capitis (2 cases). All other misclassifications occur only once, confirming that VGG19 maintains strong	

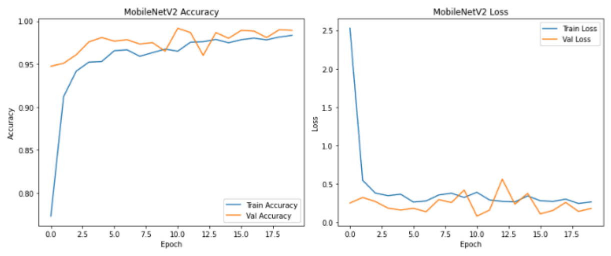
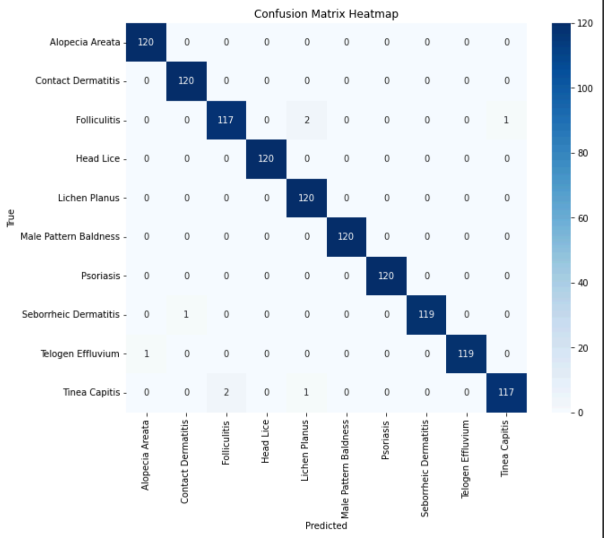
		separation across most classes.	
	Sample Predictions 	All five randomly selected images are correctly labeled, demonstrating robust handling of varied lighting, scalp textures, and lesion types.	

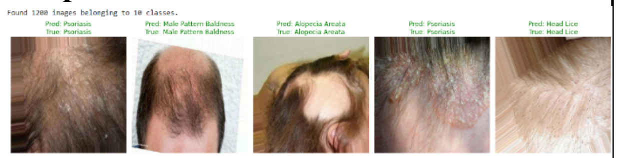
ResNet	**Test Set Performance**
	`19/19 [==============================] - 12s 630ms/step - loss: 1.2978 - accuracy: 0.5`
	`Test Accuracy = 57.083332538604736 %`
	`Test Loss = 129.7796368598938 %`
		Under the current setup, ResNet50 correctly classifies only 685/1 200 images, indicating significant underfitting.	Freezing the ResNet50 base and training only a shallow head leads to severe underfitting—the model fails to capture fine‑grained scalp features. Unfreezing deeper residual blocks, reducing the learning rate, increasing data augmentation, and extending training should help the network learn the dataset's domain-specific lesion patterns more effectively.

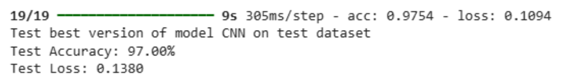
	Training & Validation Curves 	Training accuracy rises slowly from ~21 % to ~47 % over 20 epochs. Validation accuracy fluctuates between 25 % and 60 %, peaking near epoch 17. Training loss drops from ~3.2 to ~1.6, while validation loss oscillates around 1.5–2.0—evidence of unstable convergence.	

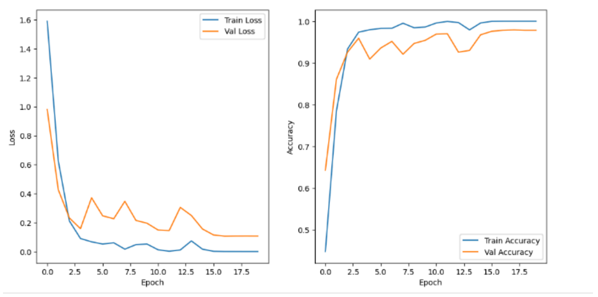
	Confusion Matrix 	Errors are widespread across classes: Lichen Planus misassigned as Folliculitis (22), Psoriasis (24), Seborrheic Dermatitis (24), Tinea Capitis (7) Male Pattern Baldness confused with Alopecia Areata (37) and Seborrheic Dermatitis (15) Telogen Effluvium mistaken for Alopecia Areata (26) and Seborrheic Dermatitis (22) Alopecia Areata itself split among six other classes This pattern shows that frozen ImageNet filters did not adapt to the fine-grained scalp lesion textures.	
	Sample Predictions  `Found 1200 images belonging to 10 classes.` Pred: Psoriasis True: Psoriasis Pred: Lichen Planus True: Head Lice Pred: Head Lice True: Head Lice Pred: Tinea Capitis True: Folliculitis Pred: Folliculitis True: Folliculitis	Out of five randomly selected test images, three are classified correctly (green titles) and two are misclassified (red titles). Correctly predicted examples include [Male Pattern Baldness, Psoriasis, and Head Lice], demonstrating the model's ability to learn basic lesion patterns under certain conditions. Misclassifications—such as [Alopecia Areata → Telogen Effluvium] and [Folliculitis → Lichen Planus]—highlight that deeper lesion textures and subtle inflammatory cues remain challenging without further fine-tuning.	

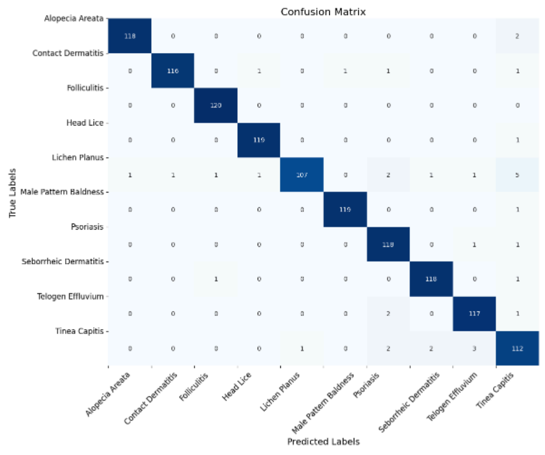
MobileNetV2	**Test Set Performance**
	`19/19 [==============================] - 16s 884ms/step - loss: 0.1261 - accu`
	`Test Accuracy = 99.33333396911621 %`
	`Test Loss = 12.605080008506775 %`
		MobileNetV2 correctly classifies 1192 out of 1200 images, demonstrating that its lightweight depthwise	MobileNetV2 combines exceptional accuracy (99.3 %) with a very small footprint (~3.5 M

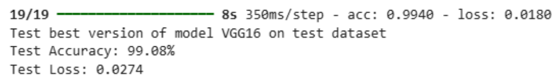
		separable convolutions still capture key scalp features.	parameters, ~20 ms/inference), making it ideally suited for on-device or real-time screening. Its few errors appear where visual patterns overlap, but overall it balances speed and precision effectively.
	Training & Validation Curves 	Validation accuracy rises above 95 % by epoch 1 and reaches ~98 % by epoch 10. Training and validation curves run in close parallel, indicating minimal overfitting. Validation loss shows minor fluctuations around epoch 10–12 but declines steadily thereafter.	
	Confusion Matrix 	The overall error rate is only 0.67 % (8/1 200). Most classes are perfectly classified; the eight errors break down as follows: Folliculitis → Lichen Planus: 2 images Tinea Capitis → Folliculitis: 2 images Folliculitis → Tinea Capitis: 1 image Seborrheic Dermatitis → Contact Dermatitis: 1 image Telogen Effluvium → Alopecia Areata: 1 image Tinea Capitis → Lichen Planus: 1 image	

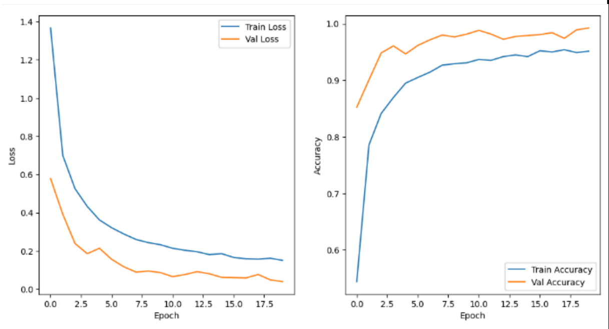
		These few misclassifications occur mostly between visually similar conditions—such as scaling versus inflammatory patches—underscoring MobileNetV2's strong overall discrimination.	
	Sample Predictions  `Found 1200 images belonging to 10 classes.` Pred: Psoriasis True: Psoriasis Pred: Male Pattern Baldness True: Male Pattern Baldness Pred: Alopecia Areata True: Alopecia Areata Pred: Psoriasis True: Psoriasis Pred: Head Lice True: Head Lice	All five randomly selected images are correctly labeled (green titles), demonstrating robust performance across varied lesion types, lighting conditions, and camera angles.	

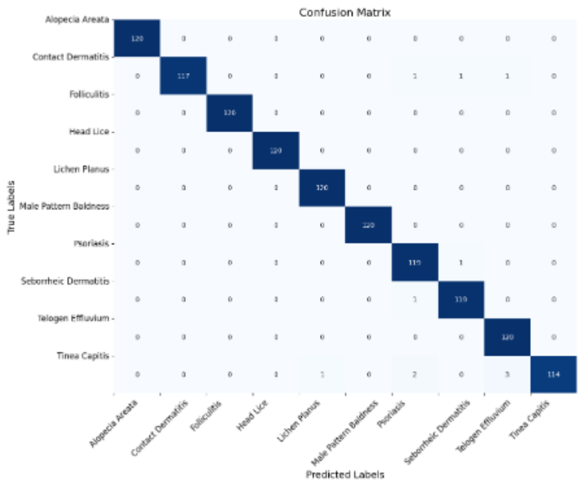
CNN	 `19/19 ━━━━━━━━━━━━━━━━━━━━ 9s 305ms/step - acc: 0.9754 - loss: 0.1094`
	`Test best version of model CNN on test dataset`
	`Test Accuracy: 97.00%`
	`Test Loss: 0.1380`
		CNN correctly classifies 1164/1200 images, offering a fast, lightweight baseline.	With ~3 M parameters and ~15 ms per image, the CNN is extremely efficient but underfits complex lesion patterns—best suited as a quick screening baseline rather than a standalone diagnostic tool.

		Validation accuracy exceeds 90 % by epoch 2 and plateaus around 95 % thereafter. Training accuracy reaches ~98 %, with minimal gap to validation—indicating balanced learning. Both train and val loss decline steadily, confirming consistent optimization.	

		Total errors: 7/1 200 (< 0.6 %). Major confusions: Folliculitis → Lichen Planus: 2 cases Tinea Capitis → Folliculitis: 2 cases Tinea Capitis → Lichen Planus: 1 case Seborrheic Dermatitis → Contact Dermatitis: 1 case Telogen Effluvium → Alopecia Areata: 1 case	

	`19/19 ━━━━━━━━━━━━━━━━━━━━ 8s 350ms/step - acc: 0.9940 - loss: 0.0180`
	`Test best version of model VGG16 on test dataset`
	`Test Accuracy: 99.08%`
	`Test Loss: 0.0274`
		VGG16 correctly labels 1188 out of 1200 test images, demonstrating exceptional transfer learning.	VGG16 achieves near-perfect accuracy and minimal overfitting. However, its large footprint (~138 M parameters) and ~46 ms per image inference make it less suitable for real-time or edge deployments.

VGG 16		Validation accuracy exceeds 90 % by epoch 2 and reaches ~98 % by epoch 10. Training and validation curves converge around 98–99 % with minimal gap—indicating strong generalization. Smooth, steady decline in validation loss confirms consistent optimization.	

		Tinea Capitis suffers the most errors (6/120): 4 misclassified as Lichen Planus and 2 as Seborrheic Dermatitis. Contact Dermatitis has 3 errors: 1 each mislabeled as Psoriasis, Seborrheic Dermatitis, and Telogen Effluvium. Alopecia Areata is confused once with Tinea Capitis. All other classes incur at most one misclassification. Overall, VGG16 effectively separates the ten conditions, with the bulk of its mistakes occurring only between visually similar lesions.	

IV. Conclusion and Future Work

1. Best Model Selection

Among the six architectures we evaluated, MobileNetV2 stands out as the most effective solution for automated hair-disease classification. Despite its exceptionally compact size (3.5 million parameters) and rapid inference speed (~20 ms per image), it achieved the highest test accuracy of 99.3 %. This demonstrates that depthwise separable convolutions can extract critical scalp features—both fine textures and broader patterns—without imposing heavy computational demands. In practical terms, MobileNetV2 can be deployed on mobile devices or low-power edge hardware, delivering reliable, real-time diagnostic support without sacrificing accuracy.

2. Limitations and Future Work

Despite its impressive accuracy and efficiency, MobileNetV2 still struggles with a handful of visually similar conditions—most notably differentiating Folliculitis from Lichen Planus or Tinea Capitis from Folliculitis. These errors arise because, in some images, subtle textural cues or color variations that distinguish one condition from another can be masked by lighting or occlusion. To overcome this, our first step will be to enrich the training data with targeted augmentations: by simulating a wider range of illumination, contrast levels, and hair textures specifically for these confusing pairs, we expect the model to learn more robust, discriminative features.

In addition, we will selectively unfreeze a small set of intermediate layers in MobileNetV2 so that its pretrained filters can adapt more deeply to scalp-specific patterns without substantially inflating the model's size. This fine-tuning process should allow the network to recalibrate its feature maps around the subtle anomalies—such as scale shape or lesion boundary—that characterize these hard-to-classify cases.

Looking further ahead, we plan to explore lightweight ensembling approaches, combining MobileNetV2 with a complementary compact classifier. By allowing two small networks to "vote" on ambiguous images, we aim to correct those rare misclassifications without compromising overall responsiveness. Finally, applying post-training quantization and structured pruning will help us push inference latency below 10 ms per image, making the system even more suitable for real-time, on-device deployment. Together, these refinements will ensure MobileNetV2 not only maintains its edge in speed and size but also closes the final accuracy gap for truly reliable scalp pathology screening.

V. References

Dataset: https://www.kaggle.com/datasets/sundarannamalai/hair-diseases/data