Checking sytem and python version and importing necessary libraries

```
#Checking the python version on the system

import sys

assert sys.version_info >= (3,7)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
#Checking the sklearn updated version of packages

from packaging import version
import sklearn

assert version.parse(sklearn.__version__)>=version.parse("1.0.1")
```

```
#Importiing tensorflow library and checking the version
import tensorflow as tf
assert version.parse(tf.__version__) >= version.parse("2.8.0")
print(tf.__version__)
```

```
2.17.0
```

```
#All the libraries that we will use forward

import numpy as np
import os
import PIL
import PIL.Image
import keras
from keras import models, Model
from tensorflow.keras import regularizers
from tensorflow.keras.regularizers import l2
os.environ['KERAS_BACKEND']='tensorflow'
from tensorflow.keras.models import Sequential
from keras.layers import Input, Conv2D, BatchNormalization, GlobalAveragePooling2D, MaxPooling2D, Flatten,Dropout, Dense,Add
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.applications.vgg16 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.metrics import ConfusionMatrixDisplay, classification_report
from sklearn.metrics import confusion_matrix as cm
import tensorflow_datasets as tfds
import skimage as ski
from IPython.display import Image, display
import matplotlib as mpl
from skimage.io import imshow, imread, imsave
from skimage import color, transform
from skimage.color import rgb2gray
from skimage.util import img_as_ubyte
from skimage import util
import cv2
import math
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
from PIL import Image
```

Uploading Data and Viusalisation

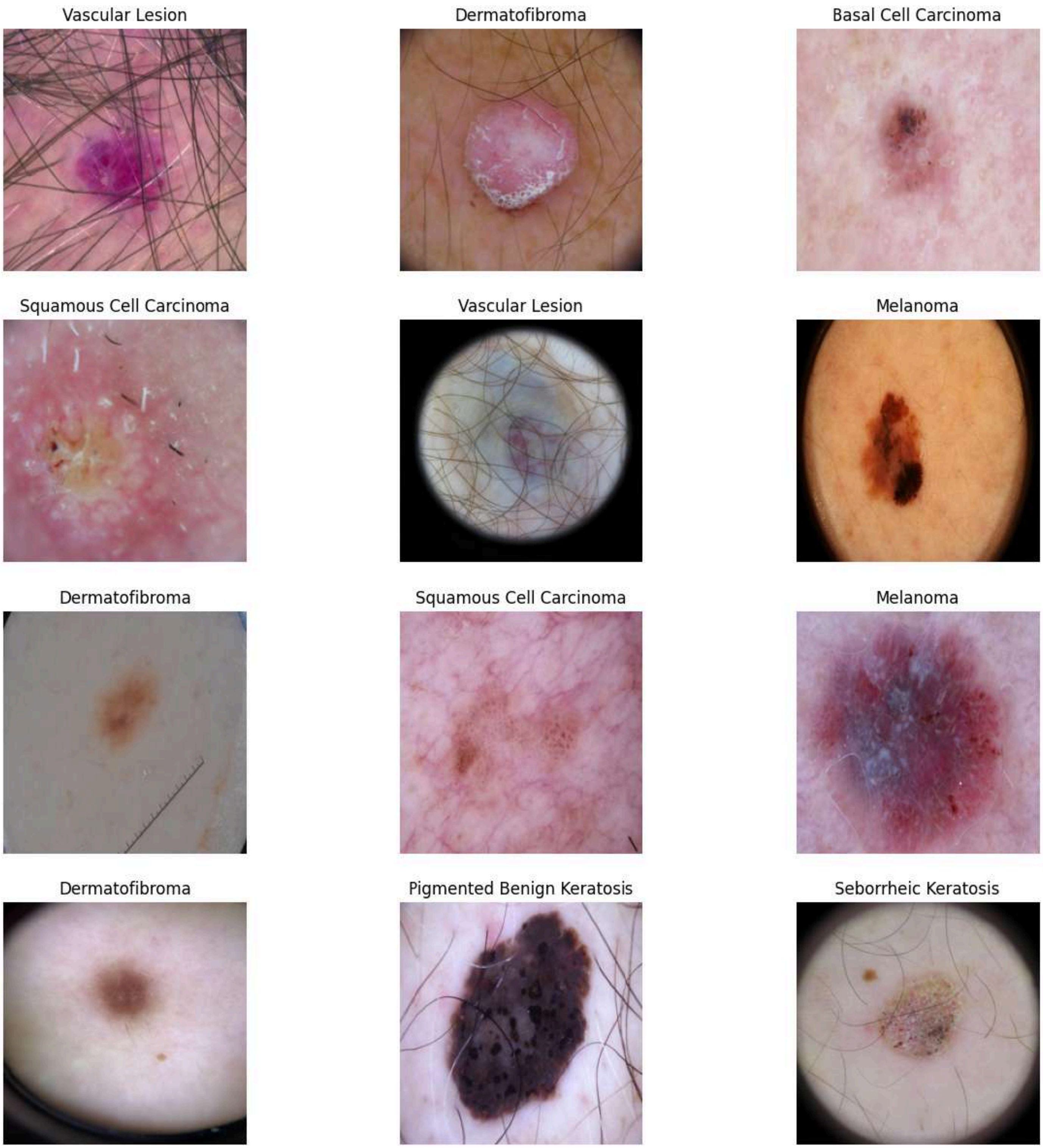
```
DATASET_DIR='/content/drive/MyDrive/Skin Cancer'
```

```
IMG_HEIGHT, IMG_WIDTH=224,224
BATCH_SIZE=32
```

```
def load_and_preprocess_data(dataset_dir, img_height, img_width):
  data=[]
  labels=[]
  class_names=os.listdir(dataset_dir)
  class_names.sort()
  class_indices={class_name: idx for idx, class_name in enumerate(class_names)}

  for class_name in class_names:
    class_dir=os.path.join(dataset_dir,class_name)
    for img_name in os.listdir(class_dir):
      img_path=os.path.join(class_dir,img_name)
      img=Image.open(img_path).convert('RGB')
      img=img.resize((img_height,img_width))
      img_array=np.array(img)/255.0
      data.append(img_array)
      labels.append(class_indices[class_name])

  data=np.array(data)
  labels=np.array(labels)
  labels=to_categorical(labels, num_classes=len(class_names))
  return data, labels, class_names

#Loading and preprocessing the data
data, labels, class_names=load_and_preprocess_data(DATASET_DIR,IMG_HEIGHT,IMG_WIDTH)

#Splitting the data into train and test
X_train,X_val,y_train,y_val=train_test_split(data, labels, test_size=0.2, random_state=52)
```

Start coding or generate with AI.

```
def display_sample_images(data, labels, class_names):
  plt.figure(figsize=(15,15))
  for i in range(12):
    plt.subplot(4,3,i+1)
    plt.imshow(data[i])
    plt.title(class_names[np.argmax(labels[i])])
    plt.axis('off')
  plt.show()

display_sample_images(X_train,y_train,class_names)
```




```
train_datagenerator=ImageDataGenerator(rotation_range=30,
                                       width_shift_range=0.2,
                                       height_shift_range=0.2,
                                       shear_range=0.3,
                                       zoom_range=0.2,
                                       horizontal_flip=True,
                                       vertical_flip=True,
                                       fill_mode='nearest')

val_datagenerator=ImageDataGenerator()
```

```
checkpoint_filepath="/content/drive/MyDrive/Skin Cancer/checkpoint.weights.h5"
model_checkpoint_callback=tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
    monitor="val_accuracy",
    mode="max",
    save_best_only=True
)
```

MODEL 1

```
#Define the input shape
input_shape=(224,224,3)
#Define the input layer
inputs=Input(shape=input_shape)
#Define the convolution layers
l=Conv2D(32,(3,3),activation='relu')(inputs)
l=BatchNormalization()(l)
l=MaxPooling2D((2,2))(l)
l=Conv2D(64,(3,3),activation='relu')(l)
l=BatchNormalization()(l)
l=MaxPooling2D((2,2))(l)
l=Conv2D(128,(3,3),activation='relu')(l)
l=BatchNormalization()(l)
l=MaxPooling2D((2,2))(l)
l=Conv2D(256,(3,3),activation='relu')(l)
l=BatchNormalization()(l)
l=MaxPooling2D((2,2))(l)
l=Conv2D(512,(3,3),activation='relu')(l)
l=BatchNormalization()(l)
l=MaxPooling2D((2,2))(l)
l=Flatten()(l)
l=Dense(512,activation='relu')(l)
l=Dropout(0.5)(l)
l=Dense(256,activation='relu')(l)
l=Dropout(0.5)(l)
l=Dense(128,activation='relu')(l)
l=Dropout(0.5)(l)
l=Dense(10,activation='softmax')(l)
#Define the output layer
outputs=l
#Create Model
model=Model(inputs=inputs,outputs=outputs)
#Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),loss='categorical_crossentropy',metrics=['accuracy'])
```

```
early_stopping=EarlyStopping(monitor='val_loss',patience=10,restore_best_weights=True)
reduce_lr=ReduceLROnPlateau(monitor='val_loss',factor=0.5,patience=5,min_lr=1e-6)
```

```
model.summary()
```

Model: "functional"

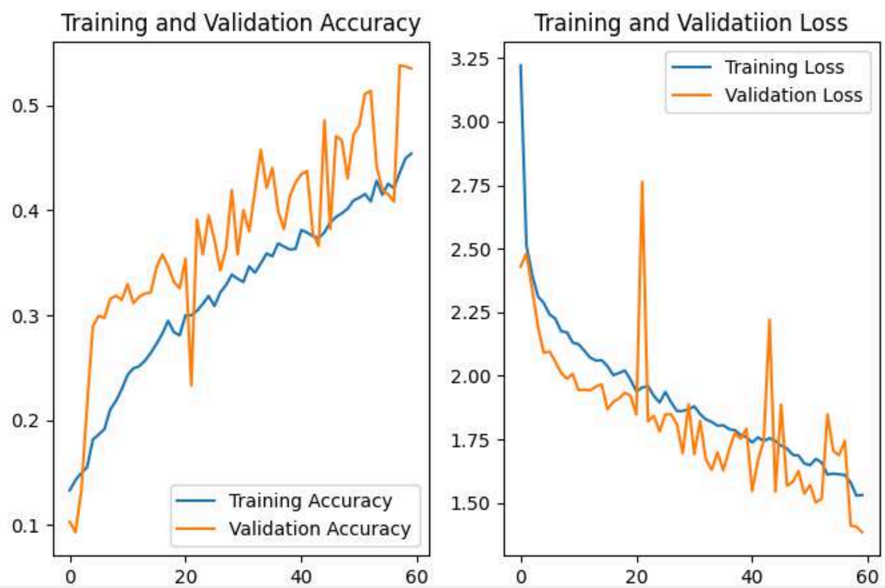
Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
conv2d (Conv2D)	(None, 222, 222, 32)	896
batch_normalization (BatchNormalization)	(None, 222, 222, 32)	128
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 109, 109, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_2 (Conv2D)	(None, 52, 52, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 52, 52, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 128)	0
conv2d_3 (Conv2D)	(None, 24, 24, 256)	295,168
batch_normalization_3 (BatchNormalization)	(None, 24, 24, 256)	1,024
max_pooling2d_3 (MaxPooling2D)	(None, 12, 12, 256)	0
conv2d_4 (Conv2D)	(None, 10, 10, 512)	1,180,160
batch_normalization_4 (BatchNormalization)	(None, 10, 10, 512)	2,048
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 512)	0
flatten (Flatten)	(None, 12800)	0
dense (Dense)	(None, 512)	6,554,112
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32,896
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1,290

```
model_history=model.fit(train_datagenerator.flow(X_train,y_train,batch_size=BATCH_SIZE),epochs=60,validation_data=(X_val,y_val),
                        callbacks=[early_stopping,reduce_lr,model_checkpoint_callback])
```

```
Epoch 1/60
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` cla
  self._warn_if_super_not_called()
124/124 ━━━━━━━━━━━━━━━━━━━━ 57s 339ms/step - accuracy: 0.1175 - loss: 3.7928 - val_accuracy: 0.1028 - val_loss: 2.4305 - learning_ra
Epoch 2/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 39s 298ms/step - accuracy: 0.1389 - loss: 2.5912 - val_accuracy: 0.0927 - val_loss: 2.4814 - learning_ra
Epoch 3/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 306ms/step - accuracy: 0.1471 - loss: 2.4238 - val_accuracy: 0.1321 - val_loss: 2.3390 - learning_ra
Epoch 4/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 307ms/step - accuracy: 0.1546 - loss: 2.3261 - val_accuracy: 0.2137 - val_loss: 2.1926 - learning_ra
Epoch 5/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 309ms/step - accuracy: 0.1680 - loss: 2.3109 - val_accuracy: 0.2893 - val_loss: 2.0909 - learning_ra
Epoch 6/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 309ms/step - accuracy: 0.1872 - loss: 2.2377 - val_accuracy: 0.2994 - val_loss: 2.0957 - learning_ra
Epoch 7/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 39s 293ms/step - accuracy: 0.1838 - loss: 2.2495 - val_accuracy: 0.2974 - val_loss: 2.0559 - learning_ra
Epoch 8/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 307ms/step - accuracy: 0.1999 - loss: 2.1841 - val_accuracy: 0.3155 - val_loss: 2.0114 - learning_ra
Epoch 9/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 308ms/step - accuracy: 0.2145 - loss: 2.1665 - val_accuracy: 0.3185 - val_loss: 1.9889 - learning_ra
Epoch 10/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 39s 296ms/step - accuracy: 0.2384 - loss: 2.1210 - val_accuracy: 0.3145 - val_loss: 2.0075 - learning_ra
Epoch 11/60
```

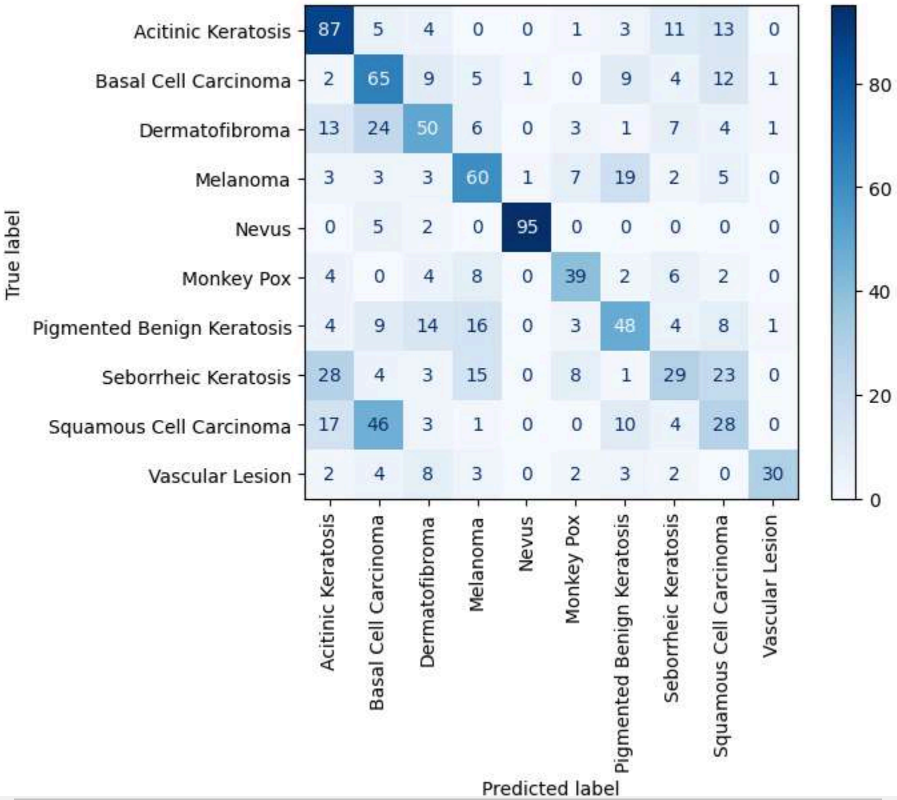
```
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 307ms/step - accuracy: 0.2399 - loss: 2.1310 - val_accuracy: 0.3296 - val_loss: 1.9434 - learning_ra
Epoch 12/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 39s 294ms/step - accuracy: 0.2407 - loss: 2.0998 - val_accuracy: 0.3115 - val_loss: 1.9459 - learning_ra
Epoch 13/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 39s 293ms/step - accuracy: 0.2323 - loss: 2.0963 - val_accuracy: 0.3175 - val_loss: 1.9425 - learning_ra
Epoch 14/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 39s 292ms/step - accuracy: 0.2574 - loss: 2.0601 - val_accuracy: 0.3206 - val_loss: 1.9576 - learning_ra
Epoch 15/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 38s 292ms/step - accuracy: 0.2613 - loss: 2.0820 - val_accuracy: 0.3216 - val_loss: 1.9670 - learning_ra
Epoch 16/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 308ms/step - accuracy: 0.2737 - loss: 2.0425 - val_accuracy: 0.3458 - val_loss: 1.8687 - learning_ra
Epoch 17/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 310ms/step - accuracy: 0.2851 - loss: 2.0151 - val_accuracy: 0.3579 - val_loss: 1.8976 - learning_ra
Epoch 18/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 39s 294ms/step - accuracy: 0.2831 - loss: 2.0246 - val_accuracy: 0.3468 - val_loss: 1.9112 - learning_ra
Epoch 19/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 39s 294ms/step - accuracy: 0.2857 - loss: 2.0050 - val_accuracy: 0.3317 - val_loss: 1.9344 - learning_ra
Epoch 20/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 39s 293ms/step - accuracy: 0.2791 - loss: 1.9841 - val_accuracy: 0.3256 - val_loss: 1.9207 - learning_ra
Epoch 21/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 39s 293ms/step - accuracy: 0.3015 - loss: 1.9544 - val_accuracy: 0.3538 - val_loss: 1.8482 - learning_ra
Epoch 22/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 38s 291ms/step - accuracy: 0.3217 - loss: 1.9173 - val_accuracy: 0.2329 - val_loss: 2.7623 - learning_ra
Epoch 23/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 306ms/step - accuracy: 0.3089 - loss: 1.9666 - val_accuracy: 0.3911 - val_loss: 1.8206 - learning_ra
Epoch 24/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 39s 294ms/step - accuracy: 0.3023 - loss: 1.9424 - val_accuracy: 0.3579 - val_loss: 1.8431 - learning_ra
Epoch 25/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 306ms/step - accuracy: 0.3170 - loss: 1.9096 - val_accuracy: 0.3952 - val_loss: 1.7814 - learning_ra
Epoch 26/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 38s 292ms/step - accuracy: 0.2959 - loss: 1.9555 - val_accuracy: 0.3720 - val_loss: 1.8487 - learning_ra
Epoch 27/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 38s 292ms/step - accuracy: 0.3233 - loss: 1.9332 - val_accuracy: 0.3427 - val_loss: 1.8488 - learning_ra
```

```
def plot_training_model_history(history):
  accuracy=history.history['accuracy']
  val_accuracy=history.history['val_accuracy']
  loss=history.history['loss']
  val_loss=history.history['val_loss']
  epochs_range=range(60)

  plt.figure(figsize=(8,5))
  plt.subplot(1,2,1)
  plt.plot(epochs_range, accuracy, label='Training Accuracy')
  plt.plot(epochs_range, val_accuracy,label='Validation Accuracy')
  plt.legend(loc='lower right')
  plt.title('Training and Validation Accuracy')

  plt.subplot(1,2,2)
  plt.plot(epochs_range,loss,label='Training Loss')
  plt.plot(epochs_range,val_loss,label='Validation Loss')
  plt.legend(loc='upper right')
  plt.title('Training and Validatiion Loss')
  plt.show()

plot_training_model_history(model_history)
```



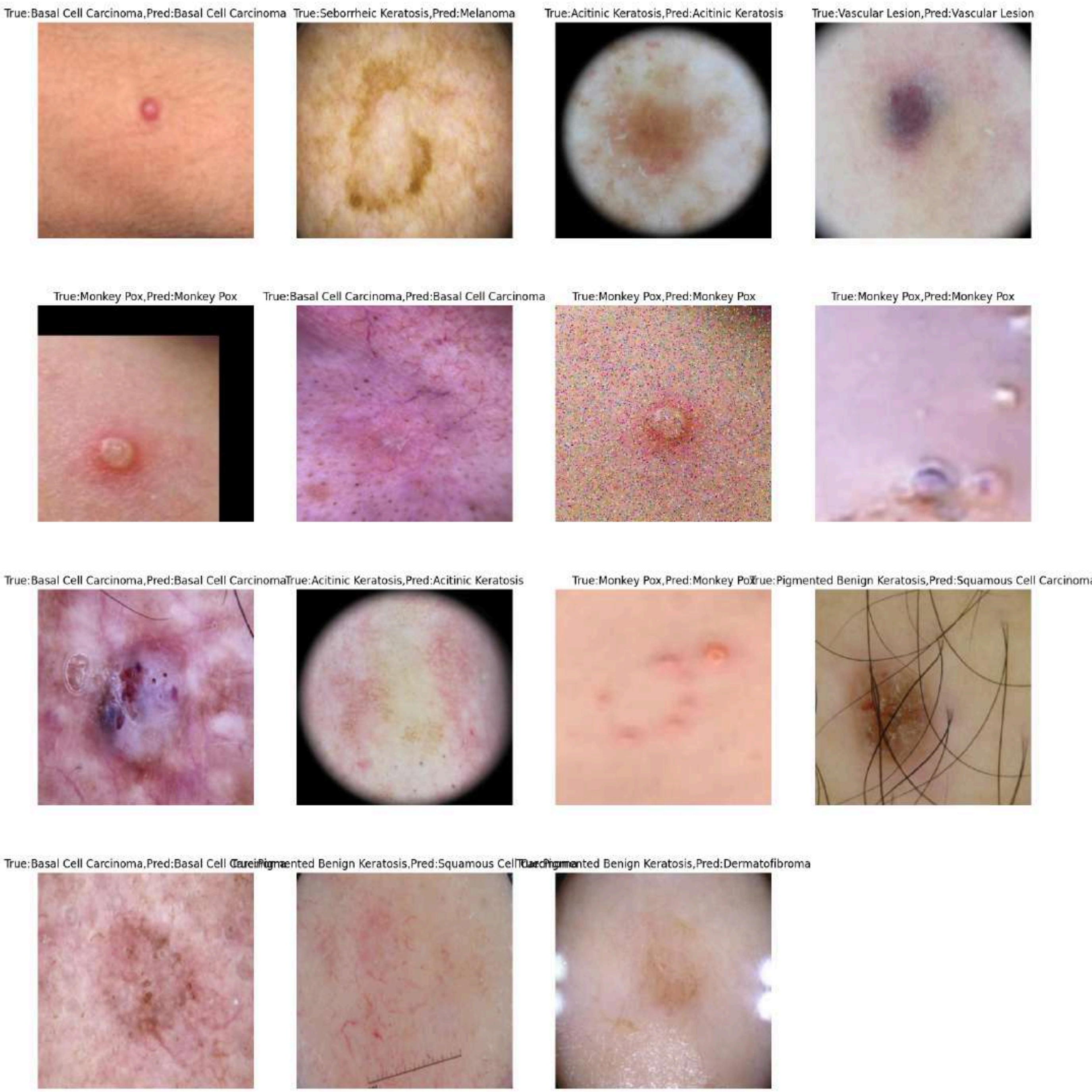

```
y_predict=model.predict(X_val)

predict= []
for i in y_predict:
    predict.append(np.argmax(i))

val=[]
for i in y_val:
  val.append(np.argmax(i))

label=['Acitinic Keratosis','Basal Cell Carcinoma','Dermatofibroma','Melanoma','Nevus','Monkey Pox','Pigmented Benign Keratosis','Seborrheic
c_m=cm(val,predict)
display=ConfusionMatrixDisplay(confusion_matrix=c_m,display_labels=label)
display.plot(cmap=plt.cm.Blues)
plt.xticks(rotation=90)
plt.show()
```

31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step




```
test_loss, test_accuracy=model.evaluate(val_datagenerator.flow(X_val,y_val,batch_size=BATCH_SIZE))
print(f'Test accuracy:{test_accuracy}')
import random
#Make predictions and compare with true labels
def check_random_sample(model_history,X_val,y_val,class_names,num_samples=15):
  indices=random.sample(range(len(X_val)),num_samples)
  plt.figure(figsize=(20,45))
  for i, idx in enumerate(indices):
    img=X_val[idx]
    true_label=np.argmax(y_val[idx])
    prediction=model.predict(np.expand_dims(img,axis=0))
    predicted_label=np.argmax(prediction)

    plt.subplot(num_samples // 2+1,4,i+1)
    plt.imshow(img)
    plt.title(f'True:{class_names[true_label]},Pred:{class_names[predicted_label]}')
    plt.axis('off')
  plt.show()

#Check random samples
check_random_sample(model_history,X_val,y_val,class_names)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class
  self._warn_if_super_not_called()
31/31 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - accuracy: 0.5456 - loss: 1.3435
Test accuracy:0.5352822542190552
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 786ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```



```
report=classification_report(val,predict,target_names=label)
print(report)
```

```
                             precision    recall  f1-score   support

        Acitinic Keratosis       0.54      0.70      0.61       124
      Basal Cell Carcinoma       0.39      0.60      0.48       108
            Dermatofibroma       0.50      0.46      0.48       109
                  Melanoma       0.53      0.58      0.55       103
                     Nevus       0.98      0.93      0.95       102
                Monkey Pox       0.62      0.60      0.61        65
Pigmented Benign Keratosis       0.50      0.45      0.47       107
      Seborrheic Keratosis       0.42      0.26      0.32       111
   Squamous Cell Carcinoma       0.29      0.26      0.27       109
           Vascular Lesion       0.91      0.56      0.69        54

                  accuracy                           0.54       992
                 macro avg       0.57      0.54      0.54       992
              weighted avg       0.54      0.54      0.53       992
```

MODEL 2

```
from functools import partial

StandardConv2D=partial(tf.keras.layers.Conv2D, kernel_size=3, strides=1,
                       padding='same',kernel_initializer='he_normal', use_bias=False)

class ResidualLayer(tf.keras.layers.Layer):
  def __init__(self, filters, strides=1, activation='relu', **kwargs):
    super().__init__(**kwargs)
    self.activation=tf.keras.activations.get(activation)
    self.main_layers=[
        StandardConv2D(filters, strides=strides),
        tf.keras.layers.BatchNormalization(),
        self.activation,
        StandardConv2D(filters),
        tf.keras.layers.BatchNormalization()
    ]
    self.skip_layers=[]
    if strides > 1:
          self.skip_layers =[StandardConv2D(filters, kernel_size=1,strides=strides),
                             tf.keras.layers.BatchNormalization()]

  def call(self, inputs):
    Z= inputs
    for layer in self.main_layers:
      Z=layer(Z)
    skip_Z=inputs
    for layer in self.skip_layers:
      skip_Z=layer(skip_Z)
    return self.activation(Z + skip_Z)


model1=tf.keras.Sequential([StandardConv2D(32, kernel_size=7, strides=2, input_shape=[IMG_HEIGHT, IMG_WIDTH, 3]),
                           tf.keras.layers.BatchNormalization(),
                           tf.keras.layers.Activation('relu'),
                           tf.keras.layers.MaxPool2D(pool_size=3, strides=2, padding='same'),
])
prev_filters=32
```

```
for filters in [32]*3+ [64]*3 + [128]*4 + [256]*6 + [512]*3:
  strides = 1 if filters == prev_filters else 2
  model1.add(ResidualLayer(filters, strides=strides))
  prev_filters = filters

model1.add(tf.keras.layers.GlobalAvgPool2D())
model1.add(tf.keras.layers.Flatten())
model1.add(tf.keras.layers.Dense(512, activation='relu'))
model1.add(tf.keras.layers.Dropout(0.5))
model1.add(tf.keras.layers.Dense(256, activation='relu'))
model1.add(tf.keras.layers.Dropout(0.5))
model1.add(tf.keras.layers.Dense(len(class_names), activation='softmax'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model1.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),loss='categorical_crossentropy', metrics=['accuracy'])
model1.summary()
```

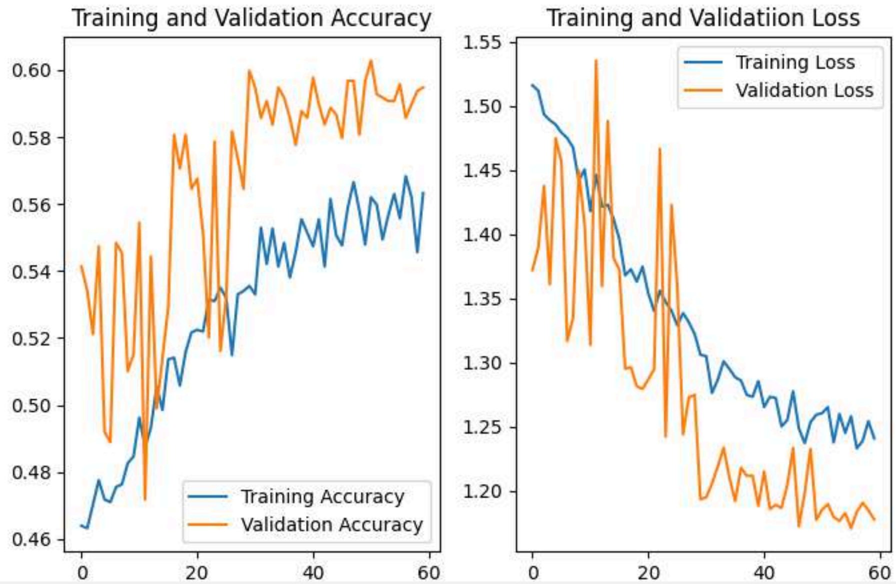
Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 112, 112, 32)	4,704
batch_normalization_5 (BatchNormalization)	(None, 112, 112, 32)	128
activation (Activation)	(None, 112, 112, 32)	0
max_pooling2d_5 (MaxPooling2D)	(None, 56, 56, 32)	0
residual_layer (ResidualLayer)	(None, 56, 56, 32)	18,688
residual_layer_1 (ResidualLayer)	(None, 56, 56, 32)	18,688
residual_layer_2 (ResidualLayer)	(None, 56, 56, 32)	18,688
residual_layer_3 (ResidualLayer)	(None, 28, 28, 64)	58,112
residual_layer_4 (ResidualLayer)	(None, 28, 28, 64)	74,240
residual_layer_5 (ResidualLayer)	(None, 28, 28, 64)	74,240
residual_layer_6 (ResidualLayer)	(None, 14, 14, 128)	230,912
residual_layer_7 (ResidualLayer)	(None, 14, 14, 128)	295,936
residual_layer_8 (ResidualLayer)	(None, 14, 14, 128)	295,936
residual_layer_9 (ResidualLayer)	(None, 14, 14, 128)	295,936
residual_layer_10 (ResidualLayer)	(None, 7, 7, 256)	920,576
residual_layer_11 (ResidualLayer)	(None, 7, 7, 256)	1,181,696
residual_layer_12 (ResidualLayer)	(None, 7, 7, 256)	1,181,696
residual_layer_13 (ResidualLayer)	(None, 7, 7, 256)	1,181,696
residual_layer_14 (ResidualLayer)	(None, 7, 7, 256)	1,181,696
residual_layer_15 (ResidualLayer)	(None, 7, 7, 256)	1,181,696
residual_layer_16 (ResidualLayer)	(None, 4, 4, 512)	3,676,160
residual_layer_17 (ResidualLayer)	(None, 4, 4, 512)	4,722,688
residual_layer_18 (ResidualLayer)	(None, 4, 4, 512)	4,722,688
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
flatten_1 (Flatten)	(None, 512)	0
dense_4 (Dense)	(None, 512)	262,656
dropout_3 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 256)	131,328
dropout_4 (Dropout)	(None, 256)	0

```
model_history1=model.fit(train_datagenerator.flow(X_train,y_train,batch_size=BATCH_SIZE),epochs=60,validation_data=(X_val,y_val),
                         callbacks=[early_stopping,reduce_lr,model_checkpoint_callback])
```

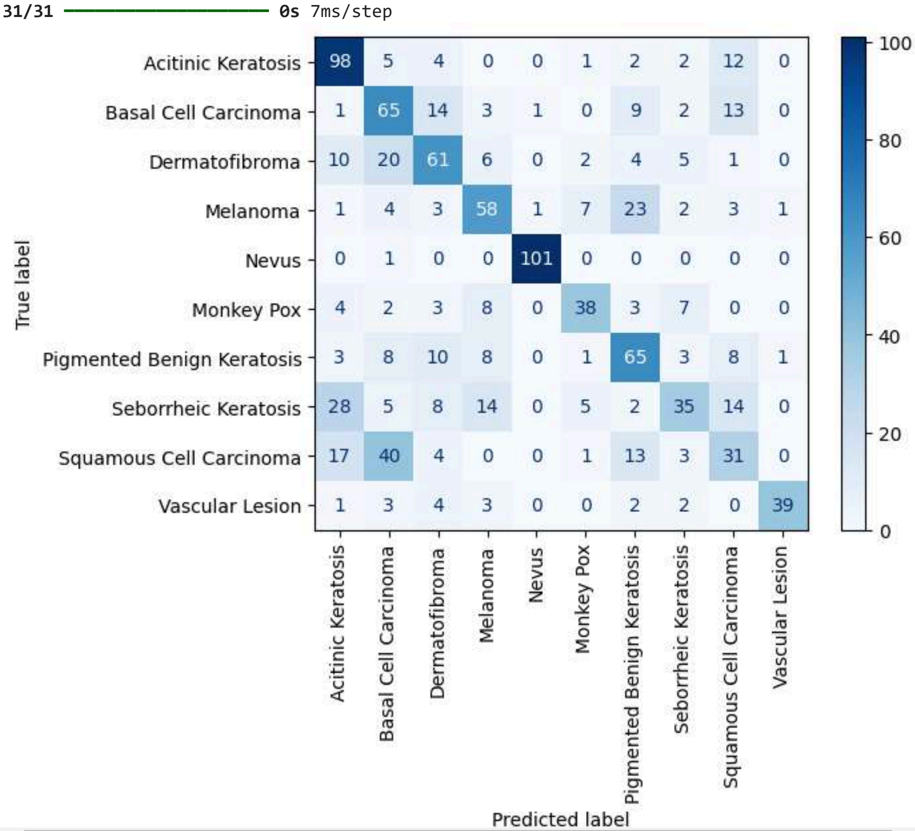
```
Epoch 40/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 304ms/step - accuracy: 0.5480 - loss: 1.2777 - val_accuracy: 0.5857 - val_loss: 1.1883 - learning_ra
Epoch 41/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 305ms/step - accuracy: 0.5485 - loss: 1.2534 - val_accuracy: 0.5978 - val_loss: 1.2151 - learning_ra
Epoch 42/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 308ms/step - accuracy: 0.5561 - loss: 1.3046 - val_accuracy: 0.5897 - val_loss: 1.1859 - learning_ra
Epoch 43/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 306ms/step - accuracy: 0.5515 - loss: 1.2806 - val_accuracy: 0.5837 - val_loss: 1.1891 - learning_ra
Epoch 44/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 306ms/step - accuracy: 0.5703 - loss: 1.2588 - val_accuracy: 0.5887 - val_loss: 1.1867 - learning_ra
Epoch 45/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 303ms/step - accuracy: 0.5466 - loss: 1.2474 - val_accuracy: 0.5867 - val_loss: 1.2048 - learning_ra
Epoch 46/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 307ms/step - accuracy: 0.5424 - loss: 1.2810 - val_accuracy: 0.5796 - val_loss: 1.2335 - learning_ra
Epoch 47/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 308ms/step - accuracy: 0.5679 - loss: 1.2339 - val_accuracy: 0.5968 - val_loss: 1.1724 - learning_ra
Epoch 48/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 307ms/step - accuracy: 0.5653 - loss: 1.2388 - val_accuracy: 0.5968 - val_loss: 1.1985 - learning_ra
Epoch 49/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 305ms/step - accuracy: 0.5518 - loss: 1.2836 - val_accuracy: 0.5806 - val_loss: 1.2327 - learning_ra
Epoch 50/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 304ms/step - accuracy: 0.5460 - loss: 1.2673 - val_accuracy: 0.5968 - val_loss: 1.1775 - learning_ra
Epoch 51/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 321ms/step - accuracy: 0.5615 - loss: 1.2703 - val_accuracy: 0.6028 - val_loss: 1.1853 - learning_ra
Epoch 52/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 306ms/step - accuracy: 0.5766 - loss: 1.2323 - val_accuracy: 0.5927 - val_loss: 1.1898 - learning_ra
Epoch 53/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 309ms/step - accuracy: 0.5554 - loss: 1.2295 - val_accuracy: 0.5917 - val_loss: 1.1798 - learning_ra
Epoch 54/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 308ms/step - accuracy: 0.5495 - loss: 1.2757 - val_accuracy: 0.5907 - val_loss: 1.1766 - learning_ra
Epoch 55/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 308ms/step - accuracy: 0.5682 - loss: 1.2541 - val_accuracy: 0.5907 - val_loss: 1.1828 - learning_ra
Epoch 56/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 305ms/step - accuracy: 0.5595 - loss: 1.2805 - val_accuracy: 0.5958 - val_loss: 1.1709 - learning_ra
Epoch 57/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 309ms/step - accuracy: 0.5717 - loss: 1.2263 - val_accuracy: 0.5857 - val_loss: 1.1838 - learning_ra
Epoch 58/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 307ms/step - accuracy: 0.5668 - loss: 1.2339 - val_accuracy: 0.5897 - val_loss: 1.1908 - learning_ra
Epoch 59/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 305ms/step - accuracy: 0.5477 - loss: 1.2323 - val_accuracy: 0.5938 - val_loss: 1.1848 - learning_ra
Epoch 60/60
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 304ms/step - accuracy: 0.5591 - loss: 1.2468 - val_accuracy: 0.5948 - val_loss: 1.1778 - learning_ra
```

```
def plot_training_model_history(history):
  accuracy=history.history['accuracy']
  val_accuracy=history.history['val_accuracy']
  loss=history.history['loss']
  val_loss=history.history['val_loss']
  epochs_range=range(60)

  plt.figure(figsize=(8,5))
  plt.subplot(1,2,1)
  plt.plot(epochs_range, accuracy, label='Training Accuracy')
  plt.plot(epochs_range, val_accuracy,label='Validation Accuracy')
  plt.legend(loc='lower right')
  plt.title('Training and Validation Accuracy')

  plt.subplot(1,2,2)
  plt.plot(epochs_range,loss,label='Training Loss')
  plt.plot(epochs_range,val_loss,label='Validation Loss')
  plt.legend(loc='upper right')
  plt.title('Training and Validatiion Loss')
  plt.show()

plot_training_model_history(model_history1)
```



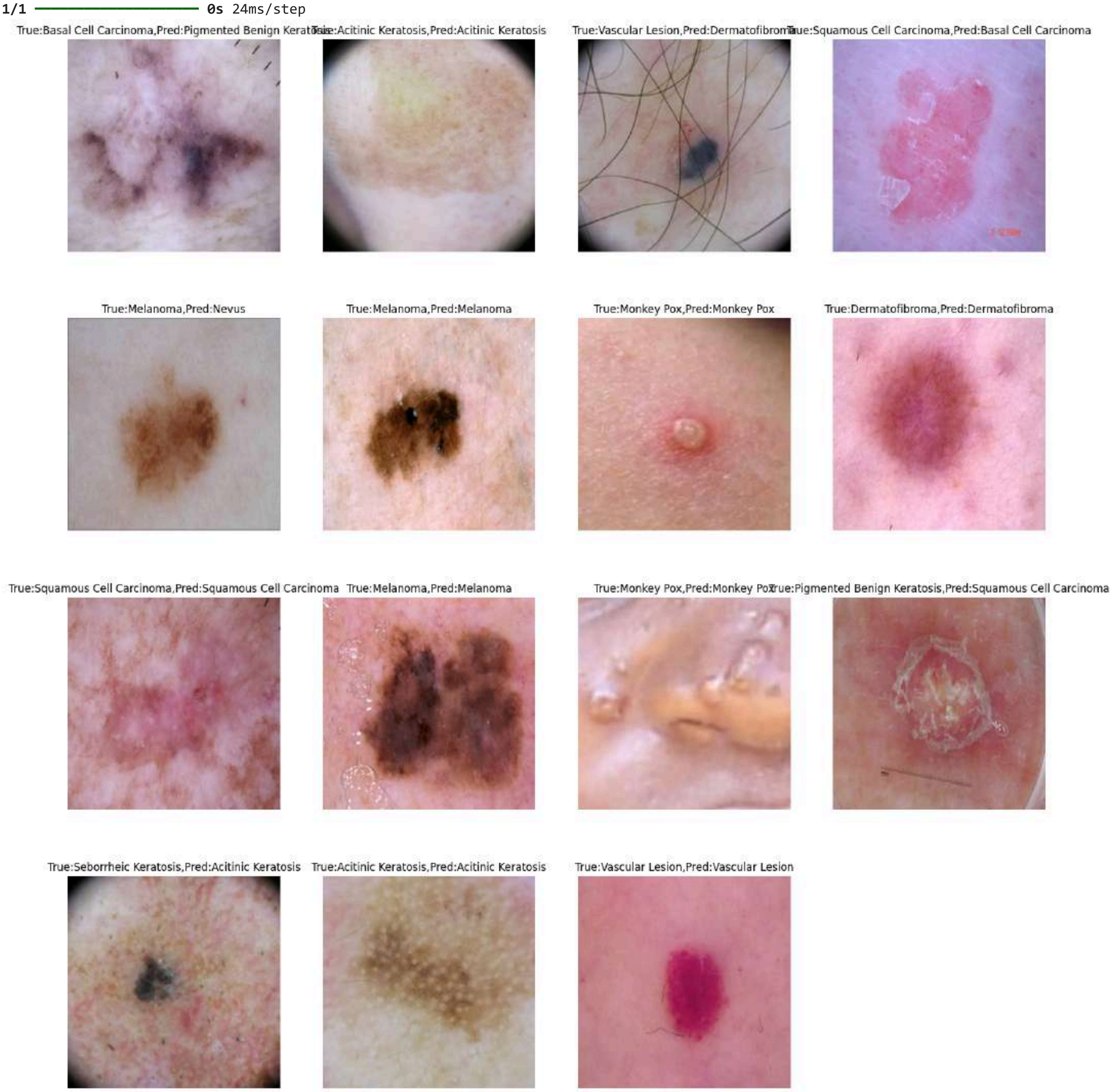

```
y_predict1=model.predict(X_val)

predict1= []
for i in y_predict1:
    predict1.append(np.argmax(i))

val1=[]
for i in y_val:
  val1.append(np.argmax(i))

label=['Acitinic Keratosis','Basal Cell Carcinoma','Dermatofibroma','Melanoma','Nevus','Monkey Pox','Pigmented Benign Keratosis','Seborrheic
confusion_matrix1=cm(val1,predict1)
display1=ConfusionMatrixDisplay(confusion_matrix=confusion_matrix1,display_labels=label)
display1.plot(cmap=plt.cm.Blues)
plt.xticks(rotation=90)
plt.show()
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
```



```
test_loss, test_accuracy=model.evaluate(val_datagenerator.flow(X_val,y_val,batch_size=BATCH_SIZE))
print(f'Test accuracy:{test_accuracy}')
import random
#Make predictions and compare with true labels
def check_random_sample(model_history,X_val,y_val,class_names,num_samples=15):
  indices=random.sample(range(len(X_val)),num_samples)
  plt.figure(figsize=(20,45))
  for i, idx in enumerate(indices):
    img=X_val[idx]
    true_label=np.argmax(y_val[idx])
    prediction=model.predict(np.expand_dims(img,axis=0))
    predicted_label=np.argmax(prediction)

    plt.subplot(num_samples // 2+1,4,i+1)
    plt.imshow(img)
    plt.title(f'True:{class_names[true_label]},Pred:{class_names[predicted_label]}')
    plt.axis('off')
  plt.show()

#Check random samples
check_random_sample(model_history1,X_val,y_val,class_names)
```

```
8/31 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.5769 - loss: 1.2105/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data
  self._warn_if_super_not_called()
31/31 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.5851 - loss: 1.1849
Test accuracy:0.5957661271095276
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
```



```
report1=classification_report(val1,predict1,target_names=label)
print(report1)
```

```
                            precision    recall  f1-score   support

        Acitinic Keratosis       0.60      0.79      0.68       124
      Basal Cell Carcinoma       0.42      0.60      0.50       108
            Dermatofibroma       0.55      0.56      0.55       109
                  Melanoma       0.58      0.56      0.57       103
                     Nevus       0.98      0.99      0.99       102
                Monkey Pox       0.69      0.58      0.63        65
Pigmented Benign Keratosis       0.53      0.61      0.57       107
      Seborrheic Keratosis       0.57      0.32      0.41       111
   Squamous Cell Carcinoma       0.38      0.28      0.32       109
           Vascular Lesion       0.95      0.72      0.82        54

                  accuracy                           0.60       992
                 macro avg       0.63      0.60      0.60       992
              weighted avg       0.60      0.60      0.59       992
```

MODEL 3

```
def residual_block(x,filters,kernel_size=3,stride=1):
  shortcut=x
  if x.shape[-1]!=filters:
    shortcut=Conv2D(filters,kernel_size=1,strides=stride,padding='same')(x)
    shortcut=BatchNormalization()(shortcut)

  x=Conv2D(filters,kernel_size=kernel_size,strides=stride,padding='same',activation='relu')(x)
  x=BatchNormalization()(x)
  x=Conv2D(filters,kernel_size=kernel_size,strides=1,padding='same',activation='relu')(x)
  x=BatchNormalization()(x)
  x=Add()([shortcut,x])
  x=tf.keras.layers.Activation('relu')(x)
  return x


def create_model(input_shape,num_classes):
  inputs=Input(shape=input_shape)
  x=Conv2D(256,(3,3),activation='relu',padding='same')(inputs)
  x=BatchNormalization()(x)
  x=MaxPooling2D((2,2))(x)

  x=residual_block(x,32)
  x=MaxPooling2D((2,2))(x)
  x=residual_block(x,64)
  x=MaxPooling2D((2,2))(x)
  x=residual_block(x,128)
  x=MaxPooling2D((2,2))(x)

  x=Conv2D(512,(3,3),activation='relu',padding='same')(x)
  x=BatchNormalization()(x)
  x=MaxPooling2D((2,2))(x)

  x=GlobalAveragePooling2D()(x)
  x=Dense(512,activation='relu')(x)
  x=Dropout(0.5)(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(num_classes,activation='softmax')(x)

  model=Model(inputs,outputs)
  return model


model2=create_model((IMG_HEIGHT,IMG_WIDTH,3),len(class_names))
model2.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),loss='categorical_crossentropy',metrics=['accuracy'])
model2.summary()
```

Model: "functional_29"

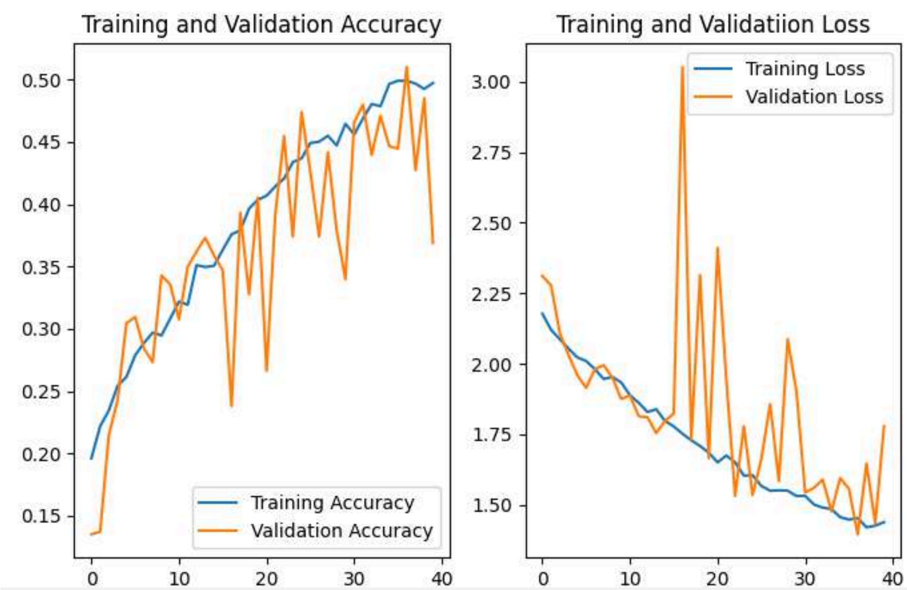
Layer (type)	Output Shape	Param #	Connected to
input_layer_3 (InputLayer)	(None, 224, 224, 3)	0	-
conv2d_59 (Conv2D)	(None, 224, 224, 256)	7,168	input_layer_3[0][0]
batch_normalization_59 (BatchNormalization)	(None, 224, 224, 256)	1,024	conv2d_59[0][0]
max_pooling2d_11 (MaxPooling2D)	(None, 112, 112, 256)	0	batch_normalization_5…
conv2d_61 (Conv2D)	(None, 112, 112, 32)	73,760	max_pooling2d_11[0][0]
batch_normalization_61 (BatchNormalization)	(None, 112, 112, 32)	128	conv2d_61[0][0]
conv2d_60 (Conv2D)	(None, 112, 112, 32)	8,224	max_pooling2d_11[0][0]
conv2d_62 (Conv2D)	(None, 112, 112, 32)	9,248	batch_normalization_6…
batch_normalization_60 (BatchNormalization)	(None, 112, 112, 32)	128	conv2d_60[0][0]
batch_normalization_62 (BatchNormalization)	(None, 112, 112, 32)	128	conv2d_62[0][0]
add_3 (Add)	(None, 112, 112, 32)	0	batch_normalization_6… batch_normalization_6…
activation_4 (Activation)	(None, 112, 112, 32)	0	add_3[0][0]
max_pooling2d_12 (MaxPooling2D)	(None, 56, 56, 32)	0	activation_4[0][0]
conv2d_64 (Conv2D)	(None, 56, 56, 64)	18,496	max_pooling2d_12[0][0]
batch_normalization_64 (BatchNormalization)	(None, 56, 56, 64)	256	conv2d_64[0][0]
conv2d_63 (Conv2D)	(None, 56, 56, 64)	2,112	max_pooling2d_12[0][0]
conv2d_65 (Conv2D)	(None, 56, 56, 64)	36,928	batch_normalization_6…
batch_normalization_63 (BatchNormalization)	(None, 56, 56, 64)	256	conv2d_63[0][0]
batch_normalization_65 (BatchNormalization)	(None, 56, 56, 64)	256	conv2d_65[0][0]
add_4 (Add)	(None, 56, 56, 64)	0	batch_normalization_6… batch_normalization_6…
activation_5 (Activation)	(None, 56, 56, 64)	0	add_4[0][0]
max_pooling2d_13 (MaxPooling2D)	(None, 28, 28, 64)	0	activation_5[0][0]
conv2d_67 (Conv2D)	(None, 28, 28, 128)	73,856	max_pooling2d_13[0][0]
batch_normalization_67 (BatchNormalization)	(None, 28, 28, 128)	512	conv2d_67[0][0]
conv2d_66 (Conv2D)	(None, 28, 28, 128)	8,320	max_pooling2d_13[0][0]
conv2d_68 (Conv2D)	(None, 28, 28, 128)	147,584	batch_normalization_6…
batch_normalization_66 (BatchNormalization)	(None, 28, 28, 128)	512	conv2d_66[0][0]
batch_normalization_68 (BatchNormalization)	(None, 28, 28, 128)	512	conv2d_68[0][0]
add_5 (Add)	(None, 28, 28, 128)	0	batch_normalization_6… batch_normalization_6…
activation_6 (Activation)	(None, 28, 28, 128)	0	add_5[0][0]
max_pooling2d_14 (MaxPooling2D)	(None, 14, 14, 128)	0	activation_6[0][0]
conv2d_69 (Conv2D)	(None, 14, 14, 512)	590,336	max_pooling2d_14[0][0]

batch_normalization_69 (BatchNormalization)	(None, 14, 14, 512)	2,048	conv2d_69[0][0]
max_pooling2d_15 (MaxPooling2D)	(None, 7, 7, 512)	0	batch_normalization_6…
global_average_pooling2d… (GlobalAveragePooling2D)	(None, 512)	0	max_pooling2d_15[0][0]
dense_11 (Dense)	(None, 512)	262,656	global_average_poolin…
dropout_8 (Dropout)	(None, 512)	0	dense_11[0][0]
dense_12 (Dense)	(None, 256)	131,328	dropout_8[0][0]
dropout_9 (Dropout)	(None, 256)	0	dense_12[0][0]
dense_13 (Dense)	(None, 128)	32,896	dropout_9[0][0]

```
model_history2=model2.fit(train_datagenerator.flow(X_train,y_train,batch_size=BATCH_SIZE),epochs=40,validation_data=(X_val,y_val),
                          callbacks=[model_checkpoint_callback])
```

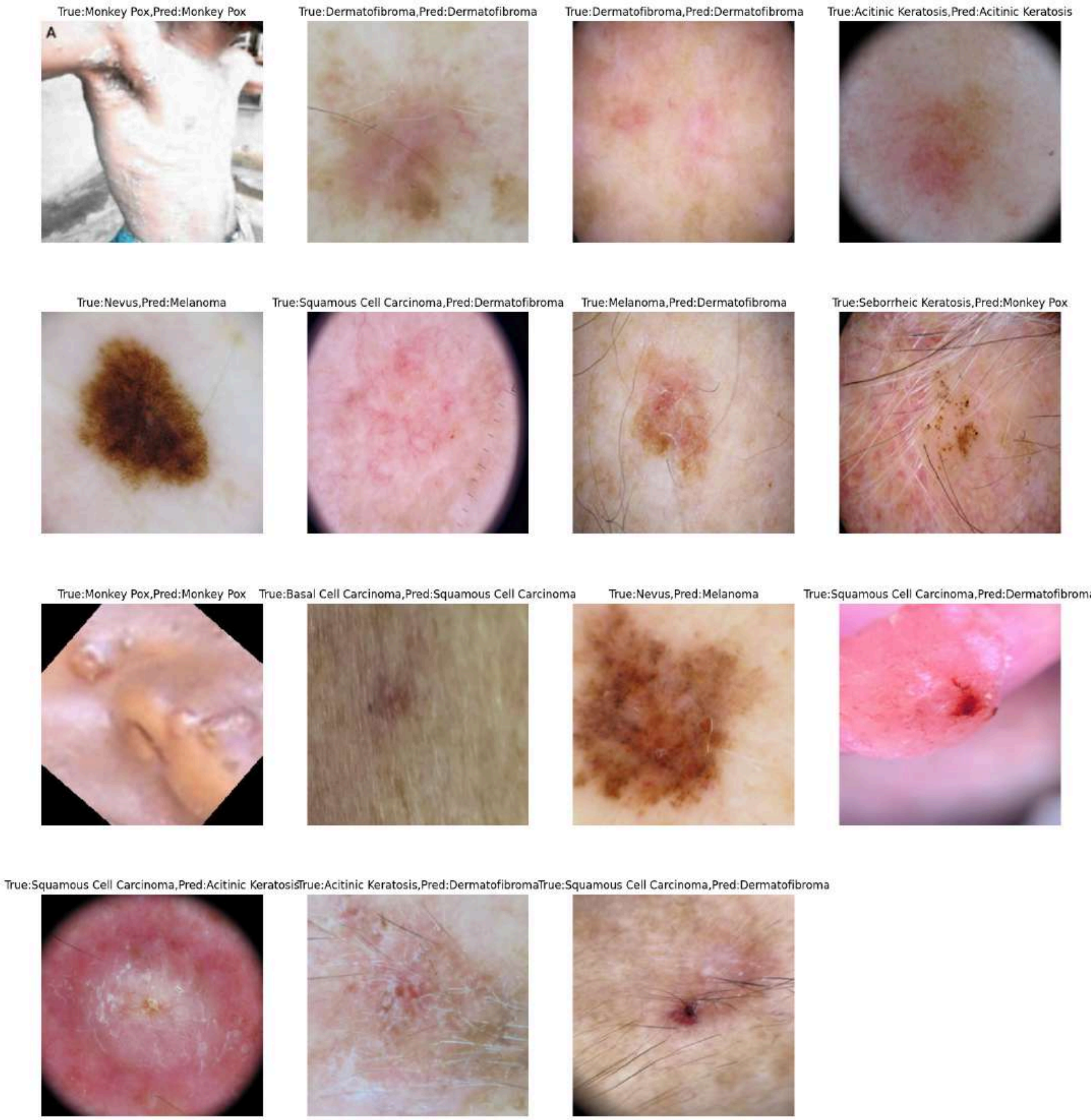
```
Epoch 1/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 43s 327ms/step - accuracy: 0.1953 - loss: 2.1830 - val_accuracy: 0.1351 - val_loss: 2.3110
Epoch 2/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 317ms/step - accuracy: 0.2127 - loss: 2.1377 - val_accuracy: 0.1371 - val_loss: 2.2769
Epoch 3/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 320ms/step - accuracy: 0.2225 - loss: 2.0941 - val_accuracy: 0.2147 - val_loss: 2.1071
Epoch 4/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 317ms/step - accuracy: 0.2449 - loss: 2.0769 - val_accuracy: 0.2429 - val_loss: 2.0301
Epoch 5/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 318ms/step - accuracy: 0.2626 - loss: 2.0092 - val_accuracy: 0.3044 - val_loss: 1.9591
Epoch 6/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 319ms/step - accuracy: 0.2634 - loss: 2.0275 - val_accuracy: 0.3095 - val_loss: 1.9148
Epoch 7/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 316ms/step - accuracy: 0.2885 - loss: 1.9856 - val_accuracy: 0.2843 - val_loss: 1.9808
Epoch 8/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 43s 324ms/step - accuracy: 0.2923 - loss: 1.9573 - val_accuracy: 0.2732 - val_loss: 1.9944
Epoch 9/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 316ms/step - accuracy: 0.3033 - loss: 1.9366 - val_accuracy: 0.3427 - val_loss: 1.9505
Epoch 10/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 318ms/step - accuracy: 0.3033 - loss: 1.9474 - val_accuracy: 0.3357 - val_loss: 1.8748
Epoch 11/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 316ms/step - accuracy: 0.3281 - loss: 1.8771 - val_accuracy: 0.3075 - val_loss: 1.8875
Epoch 12/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 317ms/step - accuracy: 0.3276 - loss: 1.8798 - val_accuracy: 0.3498 - val_loss: 1.8134
Epoch 13/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 312ms/step - accuracy: 0.3618 - loss: 1.8217 - val_accuracy: 0.3619 - val_loss: 1.8099
Epoch 14/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 315ms/step - accuracy: 0.3502 - loss: 1.8377 - val_accuracy: 0.3730 - val_loss: 1.7541
Epoch 15/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 318ms/step - accuracy: 0.3320 - loss: 1.8362 - val_accuracy: 0.3589 - val_loss: 1.7966
Epoch 16/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.3547 - loss: 1.7722 - val_accuracy: 0.3468 - val_loss: 1.8230
Epoch 17/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 321ms/step - accuracy: 0.3806 - loss: 1.7439 - val_accuracy: 0.2379 - val_loss: 3.0527
Epoch 18/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.3654 - loss: 1.7375 - val_accuracy: 0.3931 - val_loss: 1.7322
Epoch 19/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 315ms/step - accuracy: 0.3997 - loss: 1.7133 - val_accuracy: 0.3276 - val_loss: 2.3140
Epoch 20/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 322ms/step - accuracy: 0.3981 - loss: 1.6750 - val_accuracy: 0.4052 - val_loss: 1.6642
Epoch 21/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.4065 - loss: 1.6486 - val_accuracy: 0.2661 - val_loss: 2.4108
Epoch 22/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 318ms/step - accuracy: 0.4312 - loss: 1.6485 - val_accuracy: 0.3921 - val_loss: 1.9438
Epoch 23/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 315ms/step - accuracy: 0.4155 - loss: 1.6704 - val_accuracy: 0.4546 - val_loss: 1.5308
Epoch 24/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 315ms/step - accuracy: 0.4422 - loss: 1.5735 - val_accuracy: 0.3740 - val_loss: 1.7782
Epoch 25/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 316ms/step - accuracy: 0.4450 - loss: 1.5864 - val_accuracy: 0.4738 - val_loss: 1.5339
Epoch 26/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.4567 - loss: 1.5556 - val_accuracy: 0.4254 - val_loss: 1.6621
Epoch 27/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.4464 - loss: 1.5451 - val_accuracy: 0.3740 - val_loss: 1.8567
Epoch 28/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 324ms/step - accuracy: 0.4571 - loss: 1.5435 - val_accuracy: 0.4415 - val_loss: 1.5838
Epoch 29/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 314ms/step - accuracy: 0.4363 - loss: 1.5666 - val_accuracy: 0.3790 - val_loss: 2.0867
```

```
def plot_training_model_history(history):
  accuracy=history.history['accuracy']
  val_accuracy=history.history['val_accuracy']
  loss=history.history['loss']
  val_loss=history.history['val_loss']
  epochs_range=range(40)

  plt.figure(figsize=(8,5))
  plt.subplot(1,2,1)
  plt.plot(epochs_range, accuracy, label='Training Accuracy')
  plt.plot(epochs_range, val_accuracy,label='Validation Accuracy')
  plt.legend(loc='lower right')
  plt.title('Training and Validation Accuracy')

  plt.subplot(1,2,2)
  plt.plot(epochs_range,loss,label='Training Loss')
  plt.plot(epochs_range,val_loss,label='Validation Loss')
  plt.legend(loc='upper right')
  plt.title('Training and Validatiion Loss')
  plt.show()

plot_training_model_history(model_history2)
```



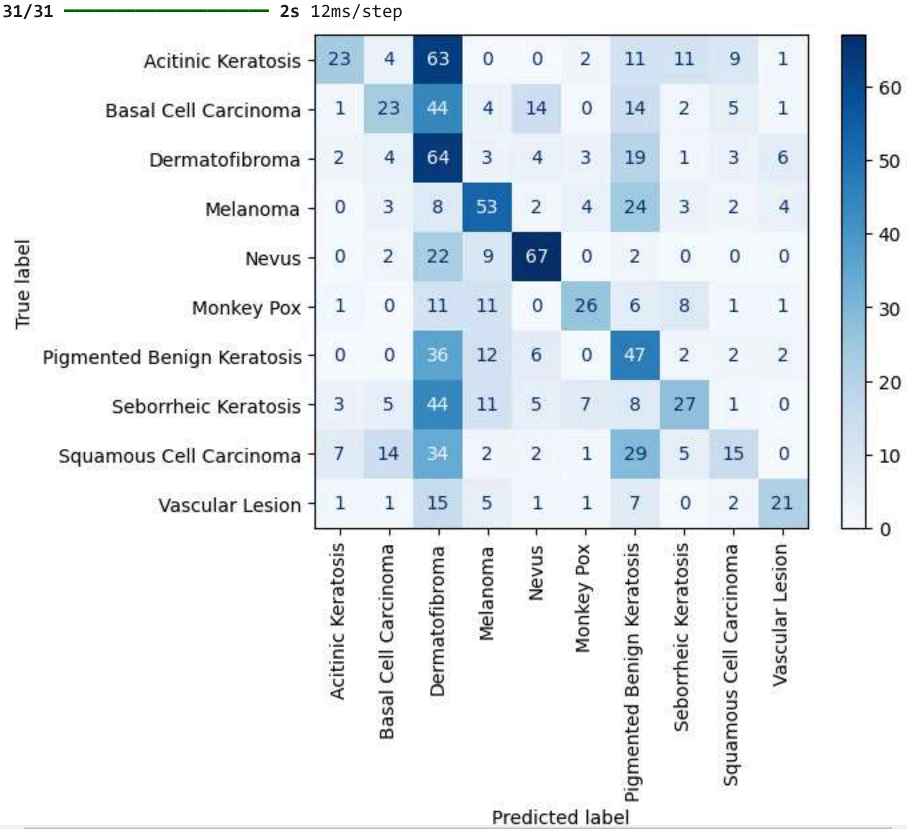

```
test_loss, test_accuracy=model2.evaluate(val_datagenerator.flow(X_val,y_val,batch_size=BATCH_SIZE))
print(f'Test accuracy:{test_accuracy}')
import random
#Make predictions and compare with true labels
def check_random_sample(model_history,X_val,y_val,class_names,num_samples=15):
  indices=random.sample(range(len(X_val)),num_samples)
  plt.figure(figsize=(20,45))
  for i, idx in enumerate(indices):
    img=X_val[idx]
    true_label=np.argmax(y_val[idx])
    prediction=model2.predict(np.expand_dims(img,axis=0))
    predicted_label=np.argmax(prediction)

    plt.subplot(num_samples // 2+1,4,i+1)
    plt.imshow(img)
    plt.title(f'True:{class_names[true_label]},Pred:{class_names[predicted_label]}')
    plt.axis('off')
  plt.show()

#Check random samples
check_random_sample(model_history2,X_val,y_val,class_names)
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.3444 - loss: 1.8128
Test accuracy:0.36895161867141724
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```



```
y_predict2=model2.predict(X_val)

predict2= []
for i in y_predict2:
    predict2.append(np.argmax(i))

val2=[]
for i in y_val:
  val2.append(np.argmax(i))

label=['Acitinic Keratosis','Basal Cell Carcinoma','Dermatofibroma','Melanoma','Nevus','Monkey Pox','Pigmented Benign Keratosis','Seborrheic
confusion_matrix2=cm(val2,predict2)
display2=ConfusionMatrixDisplay(confusion_matrix=confusion_matrix2,display_labels=label)
display2.plot(cmap=plt.cm.Blues)
plt.xticks(rotation=90)
plt.show()
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step
```




```
report2=classification_report(val2,predict2,target_names=label)
print(report2)
```

```
                            precision    recall  f1-score   support

        Acitinic Keratosis       0.61      0.19      0.28       124
      Basal Cell Carcinoma       0.41      0.21      0.28       108
            Dermatofibroma       0.19      0.59      0.28       109
                  Melanoma       0.48      0.51      0.50       103
                     Nevus       0.66      0.66      0.66       102
                Monkey Pox       0.59      0.40      0.48        65
```

```
Pigmented Benign Keratosis       0.28      0.44      0.34       107
      Seborrheic Keratosis       0.46      0.24      0.32       111
   Squamous Cell Carcinoma       0.38      0.14      0.20       109
           Vascular Lesion       0.58      0.39      0.47        54

                  accuracy                           0.37       992
                 macro avg       0.46      0.38      0.38       992
              weighted avg       0.45      0.37      0.37       992
```

Start coding or generate with AI.

Transfer Learning

1.Transfer learning is the process of using a model that has been trained on one task to another that is similar. In situation where there is insufficient data for the new task, this heps save time and enhance performance.

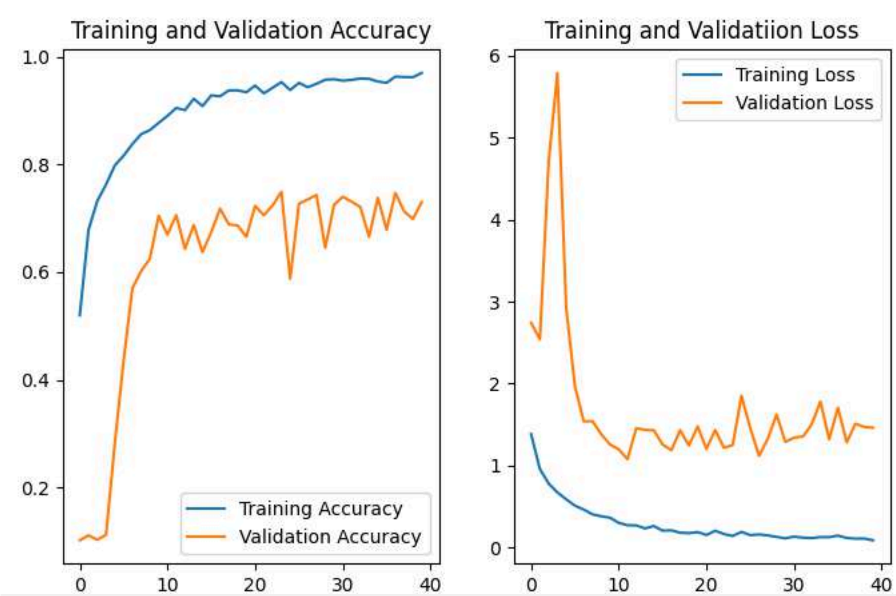
EfficientNetB0

```
efficientnetb0_model=keras.applications.EfficientNetB0(include_top=False,input_shape=(IMG_HEIGHT,IMG_WIDTH,3),weights='imagenet')

model_enb0=Sequential([efficientnetb0_model,
                       BatchNormalization(),
                       GlobalAveragePooling2D(),
                       Dense(512,activation='relu'),
                       Dropout(0.25),
                       Dense(len(class_names),activation='softmax')])

model_enb0.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),loss='categorical_crossentropy',metrics=['accuracy'])


model_history2=model_enb0.fit(train_datagenerator.flow(X_train,y_train,batch_size=BATCH_SIZE),epochs=40,validation_data=(X_val,y_val))
```

```
Epoch 1/40
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` cla
  self._warn_if_super_not_called()
124/124 ━━━━━━━━━━━━━━━━━━━━ 121s 371ms/step - accuracy: 0.4204 - loss: 1.6676 - val_accuracy: 0.1018 - val_loss: 2.7421
Epoch 2/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 316ms/step - accuracy: 0.6871 - loss: 0.9370 - val_accuracy: 0.1109 - val_loss: 2.5382
Epoch 3/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 312ms/step - accuracy: 0.7389 - loss: 0.7469 - val_accuracy: 0.1028 - val_loss: 4.7005
Epoch 4/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 316ms/step - accuracy: 0.7718 - loss: 0.6626 - val_accuracy: 0.1119 - val_loss: 5.7890
Epoch 5/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 310ms/step - accuracy: 0.8049 - loss: 0.5673 - val_accuracy: 0.2823 - val_loss: 2.9258
Epoch 6/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 314ms/step - accuracy: 0.8198 - loss: 0.5000 - val_accuracy: 0.4355 - val_loss: 1.9677
Epoch 7/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 312ms/step - accuracy: 0.8481 - loss: 0.4275 - val_accuracy: 0.5696 - val_loss: 1.5391
Epoch 8/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.8711 - loss: 0.3691 - val_accuracy: 0.6018 - val_loss: 1.5424
Epoch 9/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 316ms/step - accuracy: 0.8643 - loss: 0.3795 - val_accuracy: 0.6250 - val_loss: 1.3829
Epoch 10/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 43s 324ms/step - accuracy: 0.8799 - loss: 0.3469 - val_accuracy: 0.7046 - val_loss: 1.2577
Epoch 11/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 314ms/step - accuracy: 0.9023 - loss: 0.2721 - val_accuracy: 0.6694 - val_loss: 1.1987
Epoch 12/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 317ms/step - accuracy: 0.9117 - loss: 0.2532 - val_accuracy: 0.7056 - val_loss: 1.0788
Epoch 13/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.9011 - loss: 0.2710 - val_accuracy: 0.6431 - val_loss: 1.4576
Epoch 14/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 316ms/step - accuracy: 0.9221 - loss: 0.2261 - val_accuracy: 0.6875 - val_loss: 1.4354
Epoch 15/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 323ms/step - accuracy: 0.9161 - loss: 0.2426 - val_accuracy: 0.6371 - val_loss: 1.4313
Epoch 16/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 320ms/step - accuracy: 0.9311 - loss: 0.2093 - val_accuracy: 0.6744 - val_loss: 1.2571
Epoch 17/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 322ms/step - accuracy: 0.9371 - loss: 0.1897 - val_accuracy: 0.7177 - val_loss: 1.1883
Epoch 18/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 315ms/step - accuracy: 0.9312 - loss: 0.1843 - val_accuracy: 0.6885 - val_loss: 1.4338
Epoch 19/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.9420 - loss: 0.1659 - val_accuracy: 0.6865 - val_loss: 1.2449
Epoch 20/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 316ms/step - accuracy: 0.9347 - loss: 0.1858 - val_accuracy: 0.6653 - val_loss: 1.4780
```

```
Epoch 21/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 312ms/step - accuracy: 0.9542 - loss: 0.1391 - val_accuracy: 0.7228 - val_loss: 1.2047
Epoch 22/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 312ms/step - accuracy: 0.9447 - loss: 0.1787 - val_accuracy: 0.7056 - val_loss: 1.4342
Epoch 23/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.9491 - loss: 0.1491 - val_accuracy: 0.7238 - val_loss: 1.2181
Epoch 24/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 314ms/step - accuracy: 0.9541 - loss: 0.1449 - val_accuracy: 0.7490 - val_loss: 1.2514
Epoch 25/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.9436 - loss: 0.1682 - val_accuracy: 0.5877 - val_loss: 1.8535
Epoch 26/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 322ms/step - accuracy: 0.9615 - loss: 0.1286 - val_accuracy: 0.7268 - val_loss: 1.4561
Epoch 27/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 320ms/step - accuracy: 0.9432 - loss: 0.1637 - val_accuracy: 0.7349 - val_loss: 1.1206
```

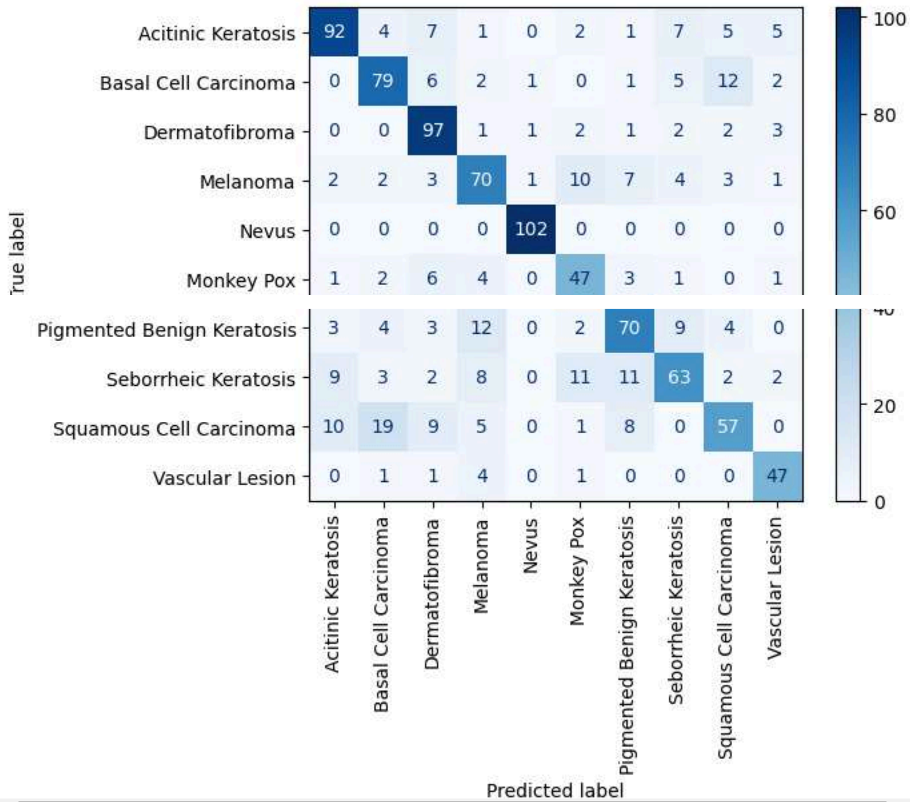
```
def plot_training_model_history(history):
  accuracy=history.history['accuracy']
  val_accuracy=history.history['val_accuracy']
  loss=history.history['loss']
  val_loss=history.history['val_loss']
  epochs_range=range(40)

  plt.figure(figsize=(8,5))
  plt.subplot(1,2,1)
  plt.plot(epochs_range, accuracy, label='Training Accuracy')
  plt.plot(epochs_range, val_accuracy,label='Validation Accuracy')
  plt.legend(loc='lower right')
  plt.title('Training and Validation Accuracy')

  plt.subplot(1,2,2)
  plt.plot(epochs_range,loss,label='Training Loss')
  plt.plot(epochs_range,val_loss,label='Validation Loss')
  plt.legend(loc='upper right')
  plt.title('Training and Validatiion Loss')
  plt.show()

plot_training_model_history(model_history2)
```




```
y_predict3=model_enb0.predict(X_val)

predict3= []
for i in y_predict3:
    predict3.append(np.argmax(i))

val3=[]
for i in y_val:
  val3.append(np.argmax(i))

label=['Acitinic Keratosis','Basal Cell Carcinoma','Dermatofibroma','Melanoma','Nevus','Monkey Pox','Pigmented Benign Keratosis','Seborrheic
confusion_matrix3=cm(val3,predict3)
display3=ConfusionMatrixDisplay(confusion_matrix=confusion_matrix3,display_labels=label)
```

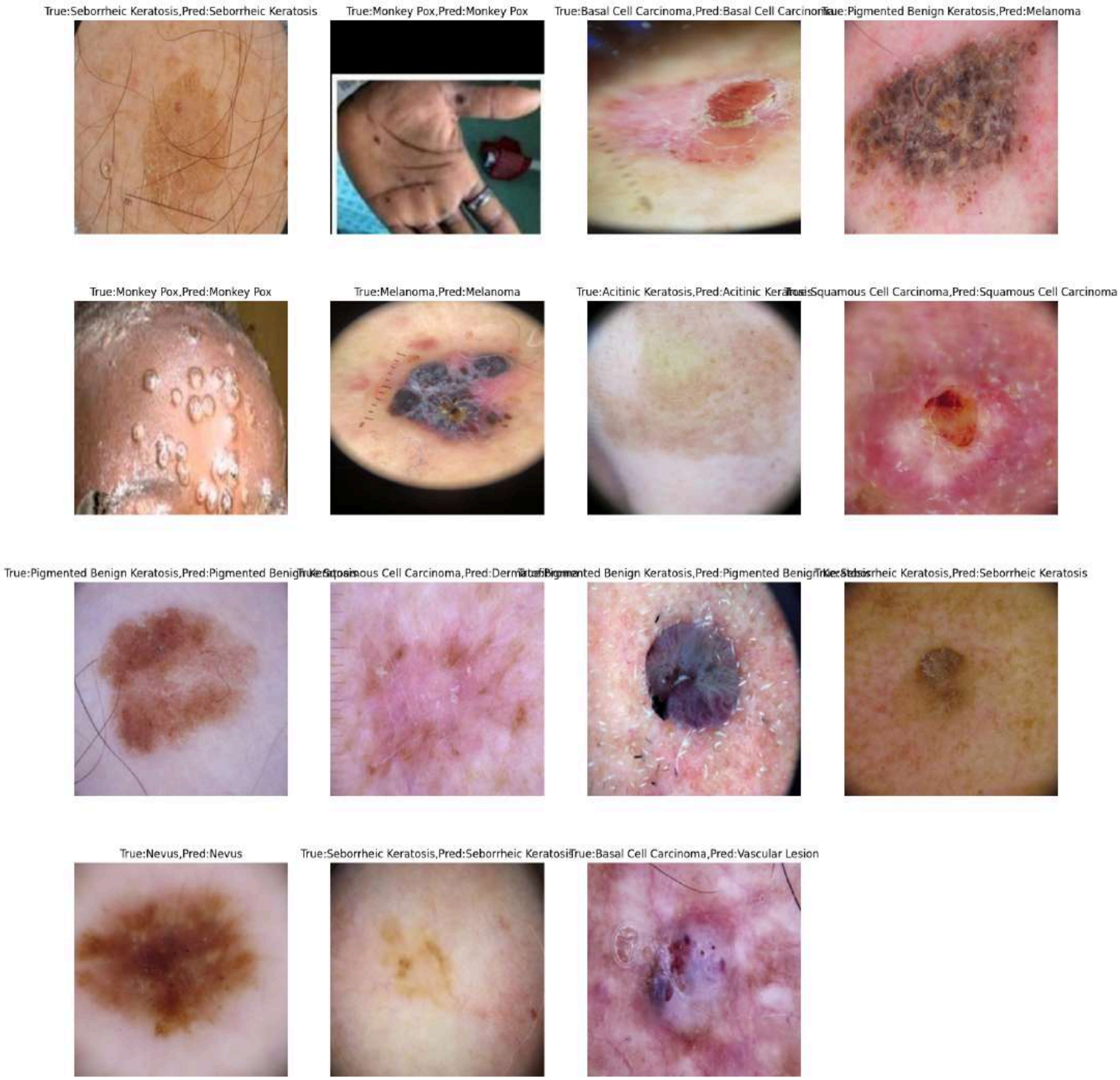
```
display3.plot(cmap=plt.cm.Blues)
plt.xticks(rotation=90)
plt.show()
```

31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step




```
test_loss, test_accuracy=model_enb0.evaluate(val_datagenerator.flow(X_val,y_val,batch_size=BATCH_SIZE))
print(f'Test accuracy:{test_accuracy}')
import random
#Make predictions and compare with true labels
def check_random_sample(model_history,X_val,y_val,class_names,num_samples=15):
  indices=random.sample(range(len(X_val)),num_samples)
  plt.figure(figsize=(20,45))
  for i, idx in enumerate(indices):
    img=X_val[idx]
    true_label=np.argmax(y_val[idx])
    prediction=model_enb0.predict(np.expand_dims(img,axis=0))
    predicted_label=np.argmax(prediction)

    plt.subplot(num_samples // 2+1,4,i+1)
    plt.imshow(img)
    plt.title(f'True:{class_names[true_label]},Pred:{class_names[predicted_label]}')
    plt.axis('off')
  plt.show()

#Check random samples
check_random_sample(model_history2,X_val,y_val,class_names)
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 9s 17ms/step - accuracy: 0.7326 - loss: 1.3403
Test accuracy:0.7298387289047241
1/1 ━━━━━━━━━━━━━━━━━━━━ 8s 8s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
```



```
report3=classification_report(val3,predict3,target_names=label)
print(report3)
```

```
                            precision    recall  f1-score   support

        Acitinic Keratosis       0.79      0.74      0.76       124
      Basal Cell Carcinoma       0.69      0.73      0.71       108
            Dermatofibroma       0.72      0.89      0.80       109
                  Melanoma       0.65      0.68      0.67       103
                     Nevus       0.97      1.00      0.99       102
               Monkey Pox       0.62      0.72      0.67        65
Pigmented Benign Keratosis       0.69      0.65      0.67       107
      Seborrheic Keratosis       0.69      0.57      0.62       111
   Squamous Cell Carcinoma       0.67      0.52      0.59       109
           Vascular Lesion       0.77      0.87      0.82        54

                  accuracy                           0.73       992
                 macro avg       0.73      0.74      0.73       992
              weighted avg       0.73      0.73      0.73       992
```

EfficientNetB0 1

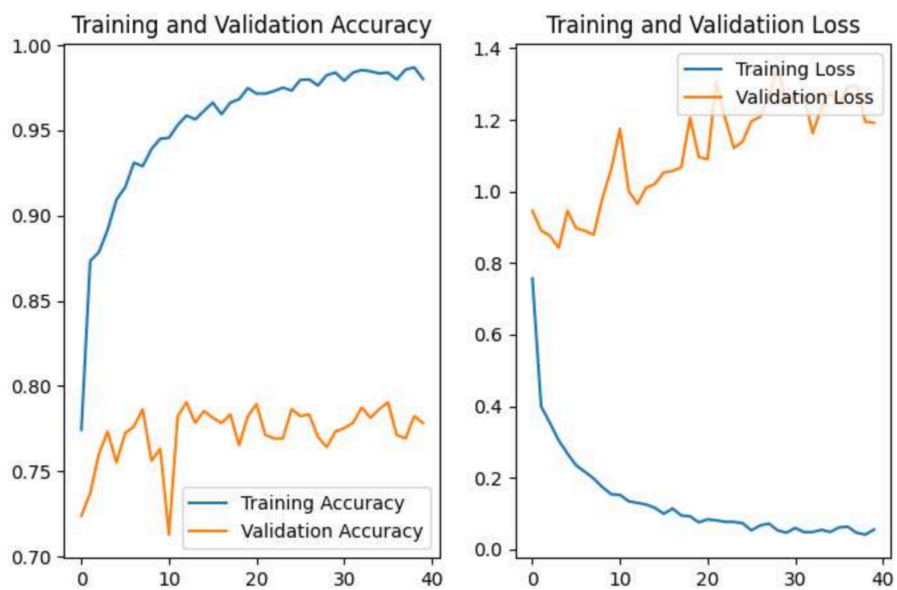
```
efficientnetb0_model1=keras.applications.EfficientNetB0(include_top=False,input_shape=(IMG_HEIGHT,IMG_WIDTH,3),weights='imagenet')

efficientnetb0_model1.trainable=True
set_trainable=False
for layer in efficientnetb0_model1.layers:
  if layer.name in ['block6a_expand_conv','block7a_expand_conv']:
    set_trainable=True
  if set_trainable:
    layer.trainable=True
  else:
    layer.trainable=False

model_enb01=Sequential([efficientnetb0_model,
                       GlobalAveragePooling2D(),
                       Dropout(0.35),
                       Dense(256,activation='relu'),
                       Dense(len(class_names),activation='softmax')])

model_enb01.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0003),loss='categorical_crossentropy',metrics=['accuracy'])
```

```
model_history3=model_enb01.fit(train_datagenerator.flow(X_train,y_train,batch_size=BATCH_SIZE),epochs=40,validation_data=(X_val,y_val))
```

```
Epoch 1/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 120s 365ms/step - accuracy: 0.6562 - loss: 1.1706 - val_accuracy: 0.7238 - val_loss: 0.9458
Epoch 2/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.8636 - loss: 0.4177 - val_accuracy: 0.7369 - val_loss: 0.8908
Epoch 3/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.8799 - loss: 0.3424 - val_accuracy: 0.7601 - val_loss: 0.8763
Epoch 4/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.8962 - loss: 0.2924 - val_accuracy: 0.7732 - val_loss: 0.8419
Epoch 5/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 309ms/step - accuracy: 0.9074 - loss: 0.2687 - val_accuracy: 0.7550 - val_loss: 0.9453
Epoch 6/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.9192 - loss: 0.2374 - val_accuracy: 0.7722 - val_loss: 0.8971
Epoch 7/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.9336 - loss: 0.2023 - val_accuracy: 0.7762 - val_loss: 0.8896
Epoch 8/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 308ms/step - accuracy: 0.9332 - loss: 0.1836 - val_accuracy: 0.7863 - val_loss: 0.8791
Epoch 9/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 310ms/step - accuracy: 0.9451 - loss: 0.1623 - val_accuracy: 0.7560 - val_loss: 0.9806
Epoch 10/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 312ms/step - accuracy: 0.9471 - loss: 0.1562 - val_accuracy: 0.7631 - val_loss: 1.0621
Epoch 11/40
```

```
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 309ms/step - accuracy: 0.9439 - loss: 0.1514 - val_accuracy: 0.7127 - val_loss: 1.1748
Epoch 12/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.9552 - loss: 0.1257 - val_accuracy: 0.7823 - val_loss: 1.0006
Epoch 13/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.9559 - loss: 0.1292 - val_accuracy: 0.7903 - val_loss: 0.9652
Epoch 14/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 310ms/step - accuracy: 0.9599 - loss: 0.1194 - val_accuracy: 0.7782 - val_loss: 1.0106
Epoch 15/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 309ms/step - accuracy: 0.9553 - loss: 0.1296 - val_accuracy: 0.7853 - val_loss: 1.0207
Epoch 16/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.9626 - loss: 0.1119 - val_accuracy: 0.7812 - val_loss: 1.0529
Epoch 17/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 314ms/step - accuracy: 0.9642 - loss: 0.1027 - val_accuracy: 0.7782 - val_loss: 1.0568
Epoch 18/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 309ms/step - accuracy: 0.9653 - loss: 0.0871 - val_accuracy: 0.7833 - val_loss: 1.0672
Epoch 19/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.9657 - loss: 0.0960 - val_accuracy: 0.7651 - val_loss: 1.2056
Epoch 20/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 309ms/step - accuracy: 0.9748 - loss: 0.0691 - val_accuracy: 0.7823 - val_loss: 1.0959
Epoch 21/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 315ms/step - accuracy: 0.9738 - loss: 0.0744 - val_accuracy: 0.7893 - val_loss: 1.0893
Epoch 22/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.9703 - loss: 0.0747 - val_accuracy: 0.7712 - val_loss: 1.3048
Epoch 23/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.9751 - loss: 0.0716 - val_accuracy: 0.7692 - val_loss: 1.2048
Epoch 24/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 312ms/step - accuracy: 0.9749 - loss: 0.0739 - val_accuracy: 0.7692 - val_loss: 1.1212
Epoch 25/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 312ms/step - accuracy: 0.9766 - loss: 0.0695 - val_accuracy: 0.7863 - val_loss: 1.1392
Epoch 26/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 314ms/step - accuracy: 0.9817 - loss: 0.0513 - val_accuracy: 0.7823 - val_loss: 1.1954
Epoch 27/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 312ms/step - accuracy: 0.9804 - loss: 0.0667 - val_accuracy: 0.7833 - val_loss: 1.2081
Epoch 28/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 314ms/step - accuracy: 0.9761 - loss: 0.0746 - val_accuracy: 0.7702 - val_loss: 1.2657
Epoch 29/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 310ms/step - accuracy: 0.9836 - loss: 0.0512 - val_accuracy: 0.7641 - val_loss: 1.3461
```

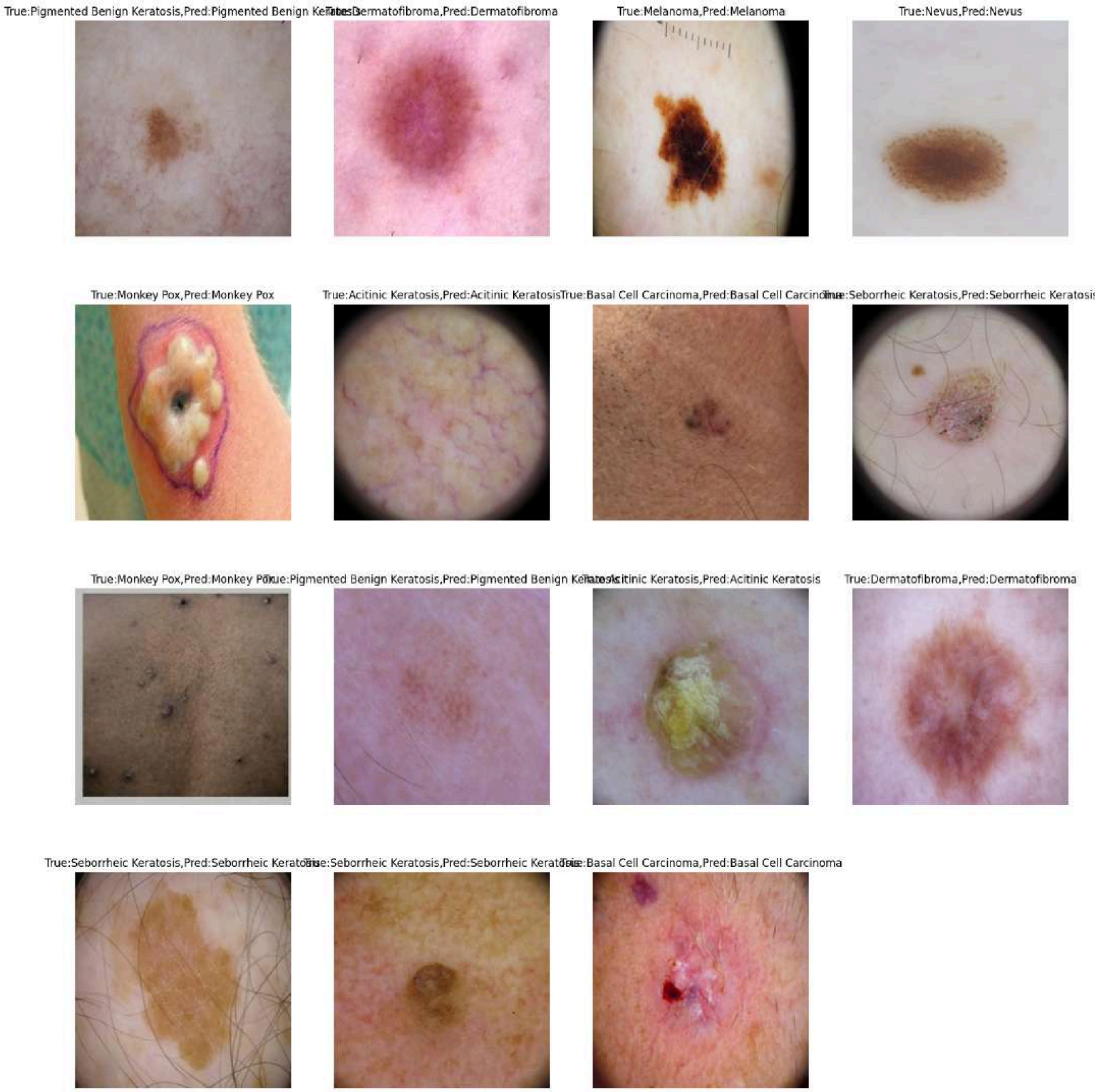
```
plot_training_model_history(model_history3)
```




```
test_loss, test_accuracy=model_enb01.evaluate(val_datagenerator.flow(X_val,y_val,batch_size=BATCH_SIZE))
print(f'Test accuracy:{test_accuracy}')
import random
#Make predictions and compare with true labels
def check_random_sample(model_history,X_val,y_val,class_names,num_samples=15):
  indices=random.sample(range(len(X_val)),num_samples)
  plt.figure(figsize=(20,45))
  for i, idx in enumerate(indices):
    img=X_val[idx]
    true_label=np.argmax(y_val[idx])
    prediction=model_enb01.predict(np.expand_dims(img,axis=0))
    predicted_label=np.argmax(prediction)

    plt.subplot(num_samples // 2+1,4,i+1)
    plt.imshow(img)
```

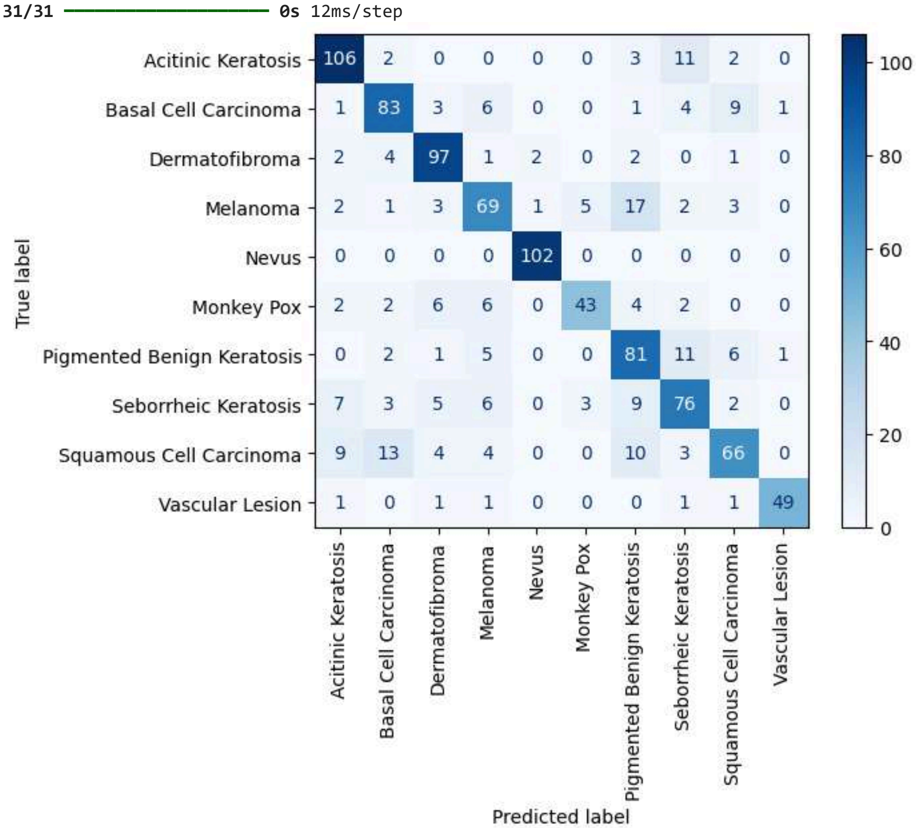
```
    plt.title(f'True:{class_names[true_label]},Pred:{class_names[predicted_label]}')
    plt.axis('off')
  plt.show()

#Check random samples
check_random_sample(model_history3,X_val,y_val,class_names)
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 7s 16ms/step - accuracy: 0.7692 - loss: 1.2010
Test accuracy:0.7782257795333862
1/1 ━━━━━━━━━━━━━━━━━━━━ 6s 6s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
```



```
y_predict4=model_enb01.predict(X_val)

predict4= []
for i in y_predict4:
    predict4.append(np.argmax(i))

val4=[]
for i in y_val:
  val4.append(np.argmax(i))

label=['Acitinic Keratosis','Basal Cell Carcinoma','Dermatofibroma','Melanoma','Nevus','Monkey Pox','Pigmented Benign Keratosis','Seborrheic
confusion_matrix4=cm(val4,predict4)
display4=ConfusionMatrixDisplay(confusion_matrix=confusion_matrix4,display_labels=label)
display4.plot(cmap=plt.cm.Blues)
plt.xticks(rotation=90)
plt.show()
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
```




```
report4=classification_report(val4,predict4,target_names=label)
print(report4)
```

```
                            precision    recall  f1-score   support

        Acitinic Keratosis       0.82      0.85      0.83       124
      Basal Cell Carcinoma       0.75      0.77      0.76       108
            Dermatofibroma       0.81      0.89      0.85       109
                  Melanoma       0.70      0.67      0.69       103
                     Nevus       0.97      1.00      0.99       102
                Monkey Pox       0.84      0.66      0.74        65
```

```
Pigmented Benign Keratosis       0.64      0.76      0.69       107
     Seborrheic Keratosis       0.69      0.68      0.69       111
  Squamous Cell Carcinoma       0.73      0.61      0.66       109
          Vascular Lesion       0.96      0.91      0.93        54

                 accuracy                           0.78       992
                macro avg       0.79      0.78      0.78       992
             weighted avg       0.78      0.78      0.78       992
```

VGG19

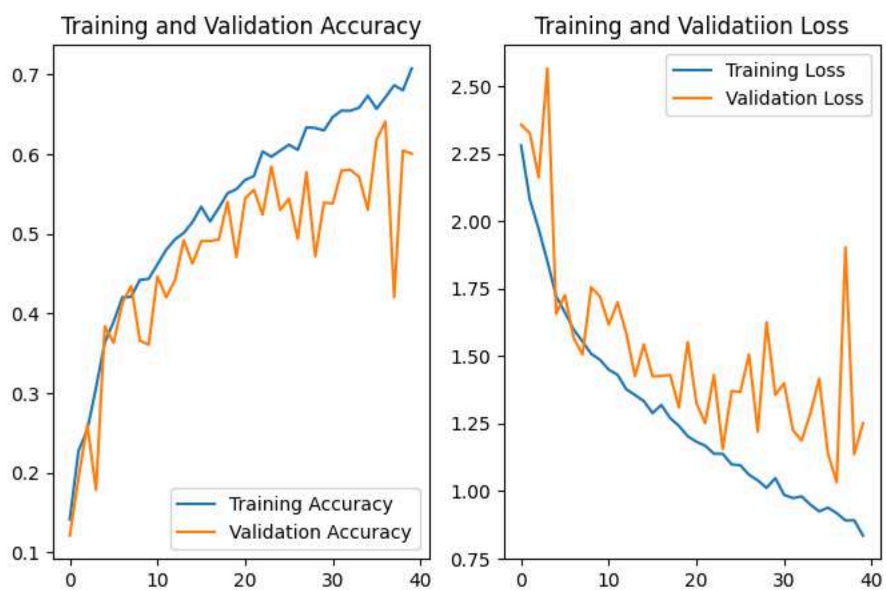
```
vgg19_model=keras.applications.VGG19(include_top=False,input_shape=(IMG_HEIGHT,IMG_WIDTH,3),weights='imagenet')

model_vgg19=Sequential([vgg19_model,
                        BatchNormalization(),
                        GlobalAveragePooling2D(),
                        Dropout(0.35),
                        Dense(256,activation='relu'),
                        Dense(len(class_names),activation='softmax')])
model_vgg19.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),loss='categorical_crossentropy',metrics=['accuracy'])
```

```
model_history4=model_vgg19.fit(train_datagenerator.flow(X_train,y_train,batch_size=BATCH_SIZE),epochs=40,validation_data=(X_val,y_val))
```

```
Epoch 1/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 52s 337ms/step - accuracy: 0.1182 - loss: 2.3057 - val_accuracy: 0.1210 - val_loss: 2.3573
Epoch 2/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 316ms/step - accuracy: 0.2229 - loss: 2.1282 - val_accuracy: 0.1935 - val_loss: 2.3251
Epoch 3/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 318ms/step - accuracy: 0.2533 - loss: 1.9920 - val_accuracy: 0.2601 - val_loss: 2.1620
Epoch 4/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 318ms/step - accuracy: 0.3043 - loss: 1.8640 - val_accuracy: 0.1784 - val_loss: 2.5655
Epoch 5/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 319ms/step - accuracy: 0.3514 - loss: 1.7341 - val_accuracy: 0.3841 - val_loss: 1.6565
Epoch 6/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 314ms/step - accuracy: 0.3780 - loss: 1.6864 - val_accuracy: 0.3629 - val_loss: 1.7262
Epoch 7/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 319ms/step - accuracy: 0.4267 - loss: 1.5819 - val_accuracy: 0.4133 - val_loss: 1.5675
Epoch 8/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 312ms/step - accuracy: 0.4175 - loss: 1.5533 - val_accuracy: 0.4345 - val_loss: 1.5061
Epoch 9/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 317ms/step - accuracy: 0.4529 - loss: 1.4923 - val_accuracy: 0.3659 - val_loss: 1.7554
Epoch 10/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 318ms/step - accuracy: 0.4417 - loss: 1.4991 - val_accuracy: 0.3609 - val_loss: 1.7207
Epoch 11/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 315ms/step - accuracy: 0.4551 - loss: 1.4596 - val_accuracy: 0.4466 - val_loss: 1.6165
Epoch 12/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 318ms/step - accuracy: 0.4782 - loss: 1.4262 - val_accuracy: 0.4204 - val_loss: 1.6995
Epoch 13/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 315ms/step - accuracy: 0.4912 - loss: 1.3970 - val_accuracy: 0.4415 - val_loss: 1.5845
Epoch 14/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 317ms/step - accuracy: 0.4991 - loss: 1.3770 - val_accuracy: 0.4919 - val_loss: 1.4258
Epoch 15/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 316ms/step - accuracy: 0.5039 - loss: 1.3436 - val_accuracy: 0.4627 - val_loss: 1.5436
Epoch 16/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.5405 - loss: 1.2854 - val_accuracy: 0.4909 - val_loss: 1.4251
Epoch 17/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 316ms/step - accuracy: 0.5048 - loss: 1.3246 - val_accuracy: 0.4909 - val_loss: 1.4263
Epoch 18/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 315ms/step - accuracy: 0.5524 - loss: 1.2376 - val_accuracy: 0.4929 - val_loss: 1.4308
Epoch 19/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.5665 - loss: 1.2321 - val_accuracy: 0.5403 - val_loss: 1.3100
Epoch 20/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 320ms/step - accuracy: 0.5528 - loss: 1.1980 - val_accuracy: 0.4708 - val_loss: 1.5520
Epoch 21/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 314ms/step - accuracy: 0.5725 - loss: 1.1875 - val_accuracy: 0.5444 - val_loss: 1.3271
Epoch 22/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 317ms/step - accuracy: 0.5758 - loss: 1.1674 - val_accuracy: 0.5554 - val_loss: 1.2516
Epoch 23/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 318ms/step - accuracy: 0.6118 - loss: 1.1330 - val_accuracy: 0.5242 - val_loss: 1.4305
Epoch 24/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 314ms/step - accuracy: 0.6051 - loss: 1.1263 - val_accuracy: 0.5847 - val_loss: 1.1569
Epoch 25/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 318ms/step - accuracy: 0.5957 - loss: 1.1273 - val_accuracy: 0.5302 - val_loss: 1.3707
Epoch 26/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 317ms/step - accuracy: 0.6213 - loss: 1.0874 - val_accuracy: 0.5444 - val_loss: 1.3678
Epoch 27/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 315ms/step - accuracy: 0.5880 - loss: 1.0657 - val_accuracy: 0.4940 - val_loss: 1.5062
Epoch 28/40
```

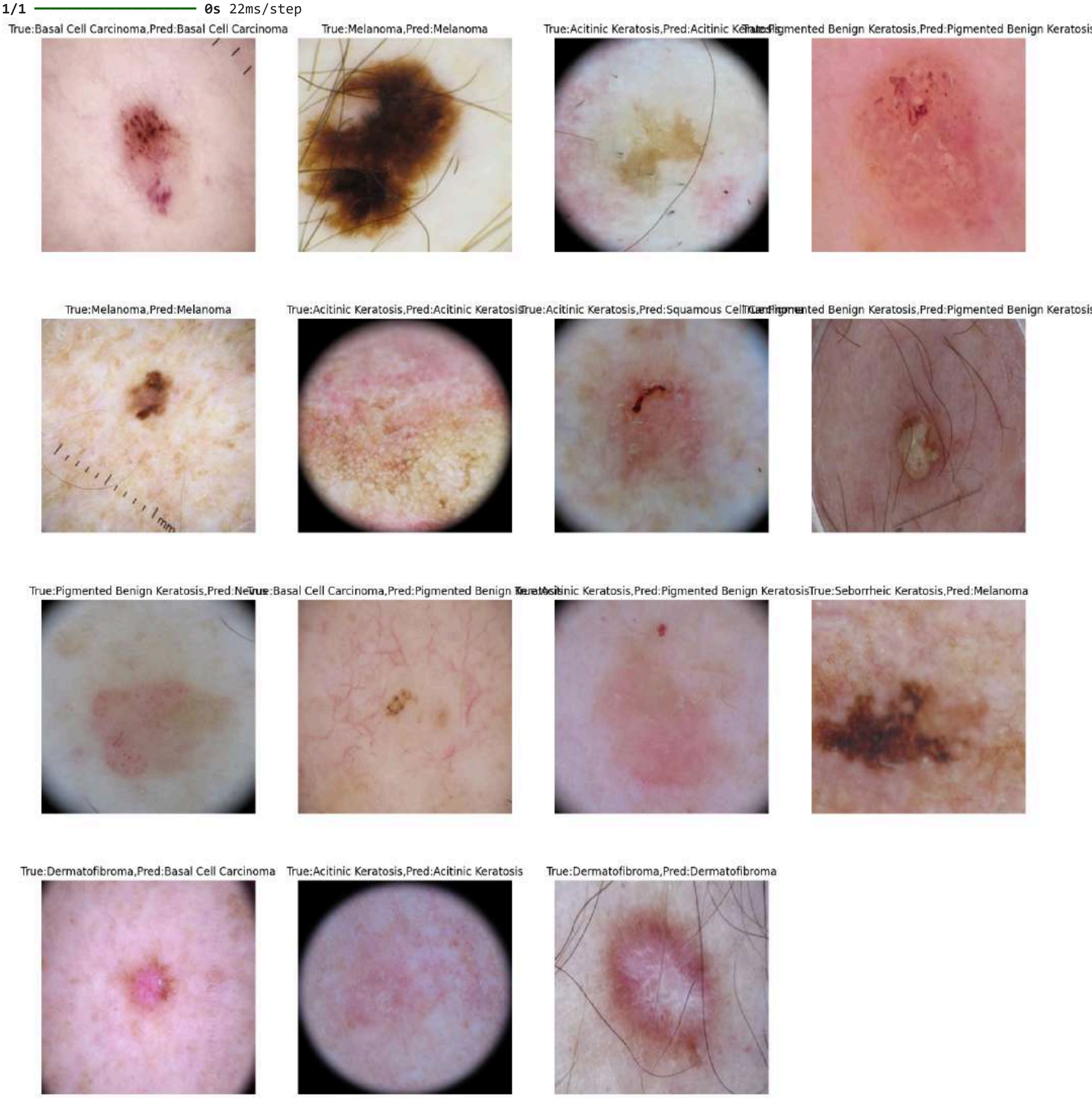
```
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 316ms/step - accuracy: 0.6332 - loss: 1.0372 - val_accuracy: 0.5776 - val_loss: 1.2202
Epoch 29/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 317ms/step - accuracy: 0.6302 - loss: 1.0261 - val_accuracy: 0.4718 - val_loss: 1.6256
```

```
def plot_training_model_history(history):
  accuracy=history.history['accuracy']
  val_accuracy=history.history['val_accuracy']
  loss=history.history['loss']
  val_loss=history.history['val_loss']
  epochs_range=range(40)

  plt.figure(figsize=(8,5))
  plt.subplot(1,2,1)
  plt.plot(epochs_range, accuracy, label='Training Accuracy')
  plt.plot(epochs_range, val_accuracy,label='Validation Accuracy')
  plt.legend(loc='lower right')
  plt.title('Training and Validation Accuracy')

  plt.subplot(1,2,2)
  plt.plot(epochs_range,loss,label='Training Loss')
  plt.plot(epochs_range,val_loss,label='Validation Loss')
  plt.legend(loc='upper right')
  plt.title('Training and Validatiion Loss')
  plt.show()

plot_training_model_history(model_history4)
```



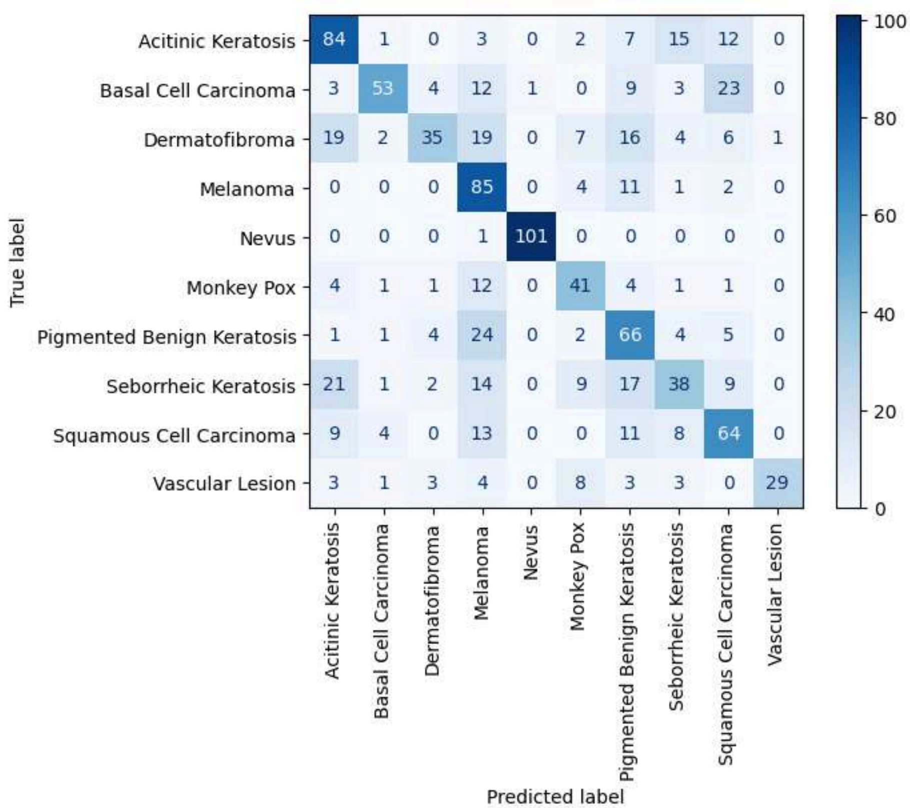

```
test_loss, test_accuracy=model_vgg19.evaluate(val_datagenerator.flow(X_val,y_val,batch_size=BATCH_SIZE))
print(f'Test accuracy:{test_accuracy}')
import random
#Make predictions and compare with true labels
def check_random_sample(model_history,X_val,y_val,class_names,num_samples=15):
  indices=random.sample(range(len(X_val)),num_samples)
  plt.figure(figsize=(20,45))
  for i, idx in enumerate(indices):
    img=X_val[idx]
    true_label=np.argmax(y_val[idx])
    prediction=model_vgg19.predict(np.expand_dims(img,axis=0))
    predicted_label=np.argmax(prediction)

    plt.subplot(num_samples // 2+1,4,i+1)
    plt.imshow(img)
    plt.title(f'True:{class_names[true_label]},Pred:{class_names[predicted_label]}')
    plt.axis('off')
  plt.show()

#Check random samples
check_random_sample(model_history4,X_val,y_val,class_names)
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 2s 23ms/step - accuracy: 0.5700 - loss: 1.2722
Test accuracy:0.600806474685669
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```



```
y_predict5=model_vgg19.predict(X_val)

predict5= []
for i in y_predict5:
    predict5.append(np.argmax(i))

val5=[]
for i in y_val:
  val5.append(np.argmax(i))

label=['Acitinic Keratosis','Basal Cell Carcinoma','Dermatofibroma','Melanoma','Nevus','Monkey Pox','Pigmented Benign Keratosis','Seborrheic
confusion_matrix5=cm(val5,predict5)
display5=ConfusionMatrixDisplay(confusion_matrix=confusion_matrix5,display_labels=label)
display5.plot(cmap=plt.cm.Blues)
plt.xticks(rotation=90)
plt.show()
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step
```




```
report5=classification_report(val5,predict5,target_names=label)
print(report5)
```

```
                            precision    recall  f1-score   support

        Acitinic Keratosis       0.58      0.68      0.63       124
      Basal Cell Carcinoma       0.83      0.49      0.62       108
            Dermatofibroma       0.71      0.32      0.44       109
                  Melanoma       0.45      0.83      0.59       103
                     Nevus       0.99      0.99      0.99       102
                Monkey Pox       0.56      0.63      0.59        65
Pigmented Benign Keratosis       0.46      0.62      0.53       107
```

```
         Seborrheic Keratosis       0.49      0.34      0.40       111
      Squamous Cell Carcinoma       0.52      0.59      0.55       109
              Vascular Lesion       0.97      0.54      0.69        54

                     accuracy                           0.60       992
                    macro avg       0.66      0.60      0.60       992
                 weighted avg       0.64      0.60      0.60       992
```

VGG19 1

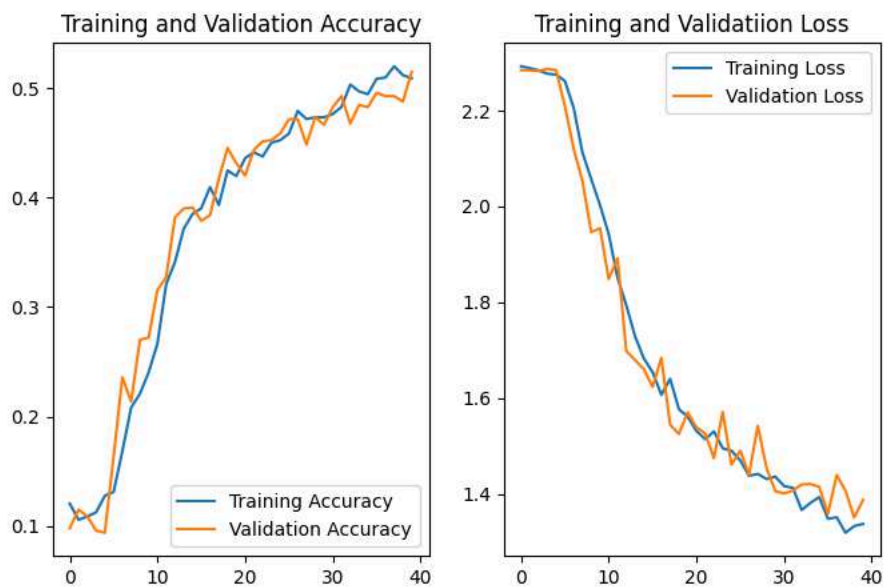
```
vgg19_model1=keras.applications.VGG19(include_top=False,input_shape=(IMG_HEIGHT,IMG_WIDTH,3),weights='imagenet')

vgg19_model1.trainable=True
set_trainable=False
for layer in vgg19_model1.layers:
  if layer.name in ['block3_conv1','block4_conv1','block5_conv1']:
    set_trainable=True
  if set_trainable:
    layer.trainable=True
  else:
    layer.trainable=False

model_vgg19_1=Sequential([vgg19_model1,
                        GlobalAveragePooling2D(),
                        Dropout(0.35),
                        Dense(256,activation='relu'),
                        Dense(len(class_names),activation='softmax')])
model_vgg19_1.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0003),loss='categorical_crossentropy',metrics=['accuracy'])
```

```
model_history5=model_vgg19_1.fit(train_datagenerator.flow(X_train,y_train,batch_size=BATCH_SIZE),epochs=40,validation_data=(X_val,y_val))
```

```
Epoch 1/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 45s 341ms/step - accuracy: 0.1141 - loss: 2.3129 - val_accuracy: 0.0978 - val_loss: 2.2843
Epoch 2/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 315ms/step - accuracy: 0.1091 - loss: 2.2892 - val_accuracy: 0.1149 - val_loss: 2.2839
Epoch 3/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 315ms/step - accuracy: 0.1151 - loss: 2.2858 - val_accuracy: 0.1089 - val_loss: 2.2828
Epoch 4/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.1061 - loss: 2.2776 - val_accuracy: 0.0958 - val_loss: 2.2871
Epoch 5/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 315ms/step - accuracy: 0.1234 - loss: 2.2760 - val_accuracy: 0.0938 - val_loss: 2.2843
Epoch 6/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 310ms/step - accuracy: 0.1097 - loss: 2.2757 - val_accuracy: 0.1633 - val_loss: 2.2090
Epoch 7/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.1639 - loss: 2.2245 - val_accuracy: 0.2359 - val_loss: 2.1200
Epoch 8/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.2108 - loss: 2.1103 - val_accuracy: 0.2137 - val_loss: 2.0530
Epoch 9/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 315ms/step - accuracy: 0.2174 - loss: 2.0536 - val_accuracy: 0.2702 - val_loss: 1.9464
Epoch 10/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 310ms/step - accuracy: 0.2448 - loss: 2.0060 - val_accuracy: 0.2722 - val_loss: 1.9544
Epoch 11/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 315ms/step - accuracy: 0.2637 - loss: 1.9496 - val_accuracy: 0.3155 - val_loss: 1.8483
Epoch 12/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 310ms/step - accuracy: 0.3017 - loss: 1.8963 - val_accuracy: 0.3276 - val_loss: 1.8919
Epoch 13/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.3363 - loss: 1.7993 - val_accuracy: 0.3821 - val_loss: 1.6985
Epoch 14/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 312ms/step - accuracy: 0.3614 - loss: 1.7675 - val_accuracy: 0.3901 - val_loss: 1.6794
Epoch 15/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 312ms/step - accuracy: 0.3792 - loss: 1.6750 - val_accuracy: 0.3911 - val_loss: 1.6605
Epoch 16/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.4002 - loss: 1.6556 - val_accuracy: 0.3790 - val_loss: 1.6234
Epoch 17/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 312ms/step - accuracy: 0.4072 - loss: 1.6086 - val_accuracy: 0.3841 - val_loss: 1.6840
Epoch 18/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 314ms/step - accuracy: 0.4057 - loss: 1.6092 - val_accuracy: 0.4173 - val_loss: 1.5444
Epoch 19/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.4206 - loss: 1.5991 - val_accuracy: 0.4456 - val_loss: 1.5251
Epoch 20/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.4126 - loss: 1.5736 - val_accuracy: 0.4315 - val_loss: 1.5703
Epoch 21/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 314ms/step - accuracy: 0.4402 - loss: 1.5296 - val_accuracy: 0.4204 - val_loss: 1.5391
Epoch 22/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.4304 - loss: 1.5612 - val_accuracy: 0.4435 - val_loss: 1.5261
Epoch 23/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.4564 - loss: 1.4972 - val_accuracy: 0.4516 - val_loss: 1.4746
```

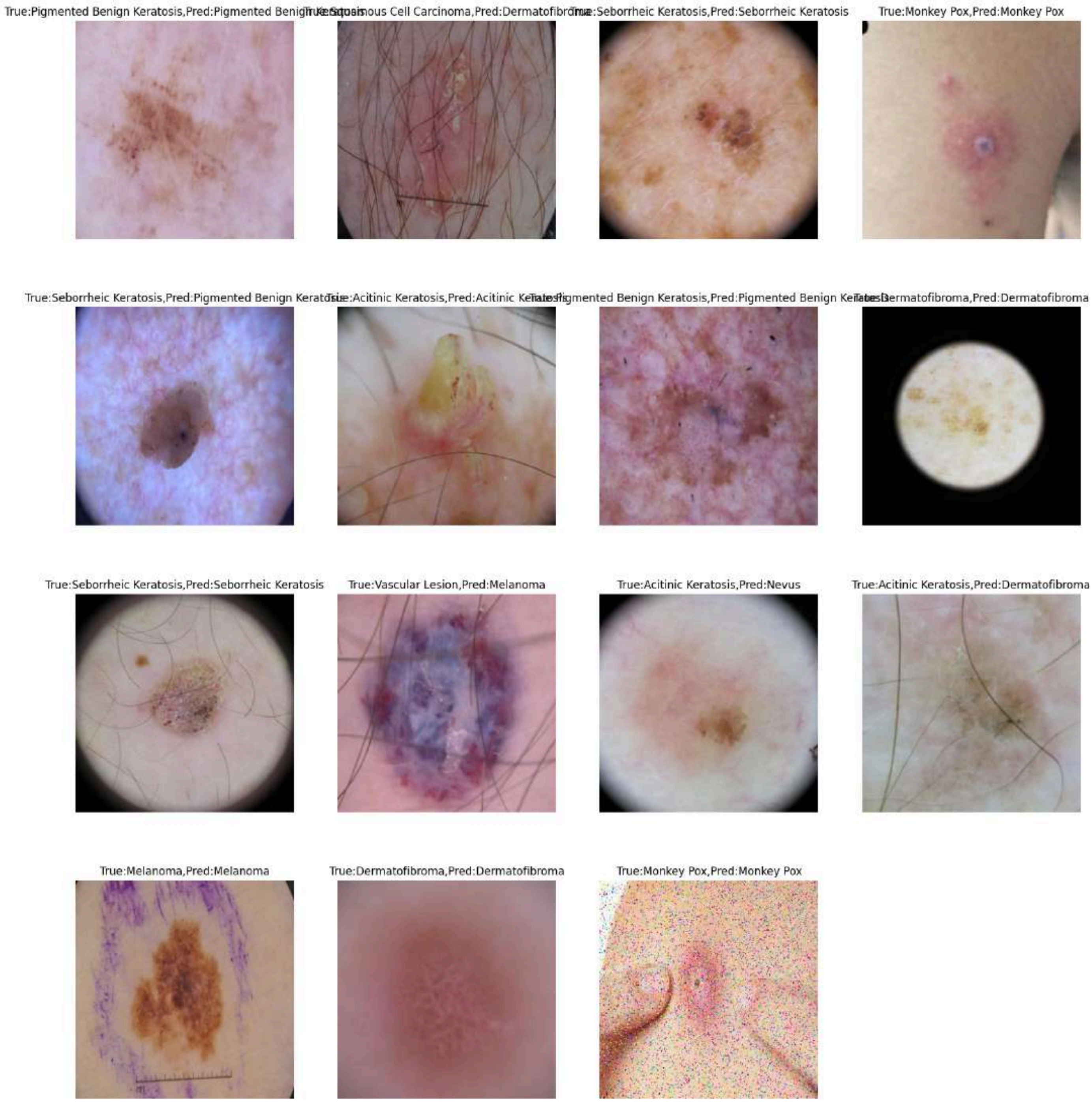
```
Epoch 24/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.4319 - loss: 1.5188 - val_accuracy: 0.4526 - val_loss: 1.5713
Epoch 25/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.4579 - loss: 1.4930 - val_accuracy: 0.4587 - val_loss: 1.4611
Epoch 26/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 42s 317ms/step - accuracy: 0.4622 - loss: 1.4593 - val_accuracy: 0.4718 - val_loss: 1.4903
Epoch 27/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.4758 - loss: 1.4433 - val_accuracy: 0.4718 - val_loss: 1.4398
Epoch 28/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 311ms/step - accuracy: 0.4719 - loss: 1.4370 - val_accuracy: 0.4486 - val_loss: 1.5422
Epoch 29/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 41s 313ms/step - accuracy: 0.4800 - loss: 1.4261 - val_accuracy: 0.4738 - val_loss: 1.4543
```

```
plot_training_model_history(model_history5)
```



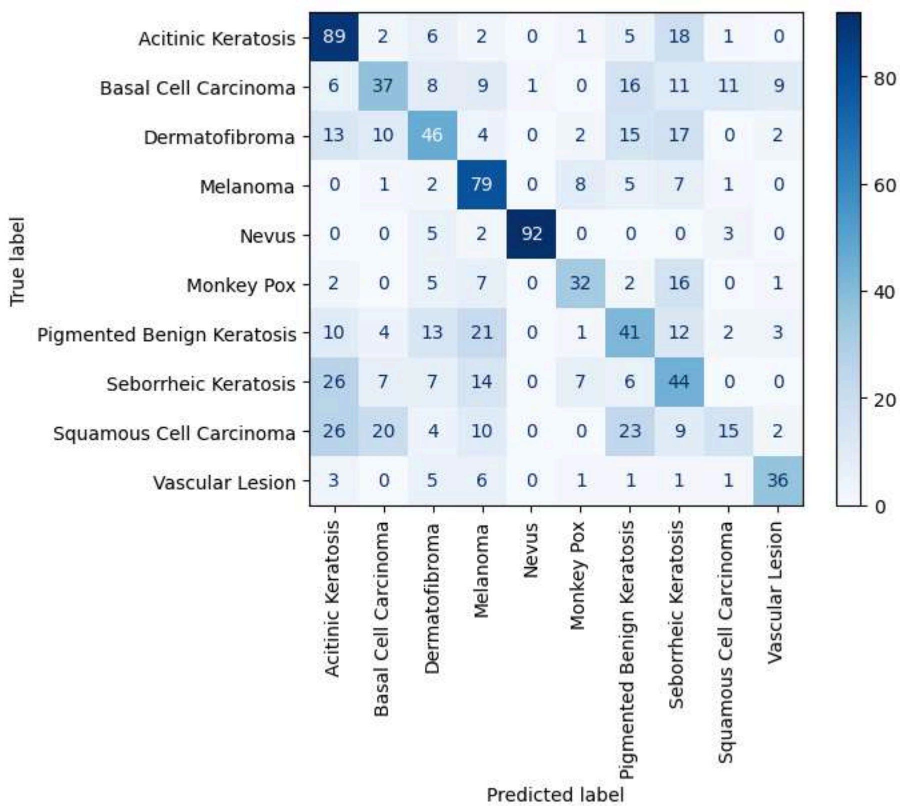

```
test_loss, test_accuracy=model_vgg19_1.evaluate(val_datagenerator.flow(X_val,y_val,batch_size=BATCH_SIZE))
print(f'Test accuracy:{test_accuracy}')
import random
#Make predictions and compare with true labels
def check_random_sample(model_history,X_val,y_val,class_names,num_samples=15):
  indices=random.sample(range(len(X_val)),num_samples)
  plt.figure(figsize=(20,45))
  for i, idx in enumerate(indices):
    img=X_val[idx]
    true_label=np.argmax(y_val[idx])
    prediction=model_vgg19_1.predict(np.expand_dims(img,axis=0))
    predicted_label=np.argmax(prediction)

    plt.subplot(num_samples // 2+1,4,i+1)
    plt.imshow(img)
    plt.title(f'True:{class_names[true_label]},Pred:{class_names[predicted_label]}')
    plt.axis('off')
  plt.show()

#Check random samples
check_random_sample(model_history5,X_val,y_val,class_names)
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 2s 22ms/step - accuracy: 0.5247 - loss: 1.3508
Test accuracy:0.5151209831237793
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 599ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```



```
y_predict6=model_vgg19_1.predict(X_val)

predict6= []
for i in y_predict6:
    predict6.append(np.argmax(i))

val6=[]
for i in y_val:
  val6.append(np.argmax(i))

label=['Acitinic Keratosis','Basal Cell Carcinoma','Dermatofibroma','Melanoma','Nevus','Monkey Pox','Pigmented Benign Keratosis','Seborrheic
confusion_matrix6=cm(val6,predict6)
display6=ConfusionMatrixDisplay(confusion_matrix=confusion_matrix6,display_labels=label)
display6.plot(cmap=plt.cm.Blues)
plt.xticks(rotation=90)
plt.show()
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step
```




```
report6=classification_report(val6,predict6,target_names=label)
print(report6)
```

```
                            precision    recall  f1-score   support

        Acitinic Keratosis       0.51      0.72      0.60       124
      Basal Cell Carcinoma       0.46      0.34      0.39       108
            Dermatofibroma       0.46      0.42      0.44       109
                  Melanoma       0.51      0.77      0.61       103
                     Nevus       0.99      0.90      0.94       102
                Monkey Pox       0.62      0.49      0.55        65
Pigmented Benign Keratosis       0.36      0.38      0.37       107
```

```
  Seborrheic Keratosis       0.33      0.40      0.36       111
Squamous Cell Carcinoma       0.44      0.14      0.21       109
        Vascular Lesion       0.68      0.67      0.67        54

               accuracy                           0.52       992
              macro avg       0.53      0.52      0.51       992
           weighted avg       0.52      0.52      0.50       992
```

InceptionV3

```
inception_v3=keras.applications.InceptionV3(include_top=False,input_shape=(IMG_HEIGHT,IMG_WIDTH,3),weights='imagenet')

inception_v3_1=Sequential([inception_v3,
                           GlobalAveragePooling2D(),
                           BatchNormalization(),
                           Dropout(0.3),
                           Dense(256,activation='relu'),
                           Dense(len(class_names),activation='softmax')])

inception_v3_1.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),loss='categorical_crossentropy',metrics=['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_
87910968/87910968 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

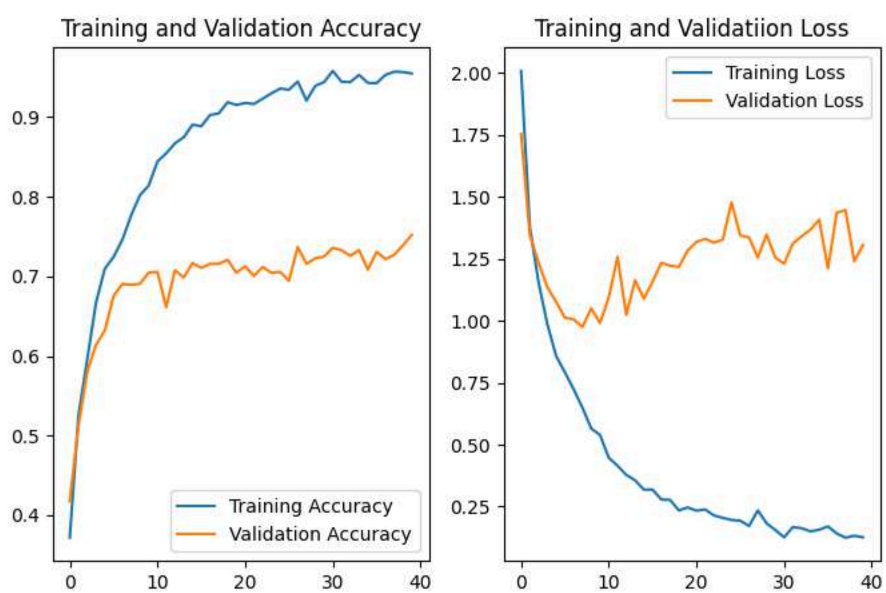
```
model_history6=inception_v3_1.fit(train_datagenerator.flow(X_train,y_train,batch_size=BATCH_SIZE),epochs=40,validation_data=(X_val,y_val))
```

```
Epoch 1/40
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` cla
  self._warn_if_super_not_called()
124/124 ━━━━━━━━━━━━━━━━━━━━ 127s 356ms/step - accuracy: 0.2729 - loss: 2.3530 - val_accuracy: 0.4173 - val_loss: 1.7542
Epoch 2/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 304ms/step - accuracy: 0.5220 - loss: 1.3615 - val_accuracy: 0.5131 - val_loss: 1.3457
Epoch 3/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 305ms/step - accuracy: 0.5798 - loss: 1.2059 - val_accuracy: 0.5806 - val_loss: 1.2341
Epoch 4/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 306ms/step - accuracy: 0.6599 - loss: 1.0237 - val_accuracy: 0.6139 - val_loss: 1.1352
Epoch 5/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 304ms/step - accuracy: 0.7083 - loss: 0.8579 - val_accuracy: 0.6321 - val_loss: 1.0763
Epoch 6/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 308ms/step - accuracy: 0.7337 - loss: 0.7554 - val_accuracy: 0.6754 - val_loss: 1.0122
Epoch 7/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 307ms/step - accuracy: 0.7449 - loss: 0.7269 - val_accuracy: 0.6905 - val_loss: 1.0058
Epoch 8/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 305ms/step - accuracy: 0.7860 - loss: 0.6257 - val_accuracy: 0.6895 - val_loss: 0.9743
Epoch 9/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 304ms/step - accuracy: 0.8161 - loss: 0.5293 - val_accuracy: 0.6905 - val_loss: 1.0505
Epoch 10/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 304ms/step - accuracy: 0.8123 - loss: 0.5315 - val_accuracy: 0.7046 - val_loss: 0.9911
Epoch 11/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 304ms/step - accuracy: 0.8529 - loss: 0.4169 - val_accuracy: 0.7056 - val_loss: 1.0936
Epoch 12/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 303ms/step - accuracy: 0.8636 - loss: 0.3849 - val_accuracy: 0.6613 - val_loss: 1.2581
Epoch 13/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 306ms/step - accuracy: 0.8728 - loss: 0.3563 - val_accuracy: 0.7077 - val_loss: 1.0234
Epoch 14/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 305ms/step - accuracy: 0.8786 - loss: 0.3305 - val_accuracy: 0.6986 - val_loss: 1.1635
Epoch 15/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 308ms/step - accuracy: 0.9073 - loss: 0.2873 - val_accuracy: 0.7167 - val_loss: 1.0884
Epoch 16/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 305ms/step - accuracy: 0.8886 - loss: 0.3105 - val_accuracy: 0.7107 - val_loss: 1.1579
Epoch 17/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 307ms/step - accuracy: 0.9062 - loss: 0.2574 - val_accuracy: 0.7157 - val_loss: 1.2337
Epoch 18/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 305ms/step - accuracy: 0.9032 - loss: 0.2718 - val_accuracy: 0.7157 - val_loss: 1.2218
Epoch 19/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 304ms/step - accuracy: 0.9239 - loss: 0.2261 - val_accuracy: 0.7208 - val_loss: 1.2172
Epoch 20/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 304ms/step - accuracy: 0.9240 - loss: 0.2230 - val_accuracy: 0.7046 - val_loss: 1.2836
Epoch 21/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 305ms/step - accuracy: 0.9285 - loss: 0.2068 - val_accuracy: 0.7127 - val_loss: 1.3179
Epoch 22/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 302ms/step - accuracy: 0.9212 - loss: 0.2285 - val_accuracy: 0.7006 - val_loss: 1.3306
Epoch 23/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 306ms/step - accuracy: 0.9336 - loss: 0.1936 - val_accuracy: 0.7117 - val_loss: 1.3152
Epoch 24/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 307ms/step - accuracy: 0.9360 - loss: 0.1895 - val_accuracy: 0.7046 - val_loss: 1.3270
```

```
Epoch 25/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 303ms/step - accuracy: 0.9369 - loss: 0.1801 - val_accuracy: 0.7056 - val_loss: 1.4779
Epoch 26/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 307ms/step - accuracy: 0.9371 - loss: 0.1879 - val_accuracy: 0.6946 - val_loss: 1.3446
Epoch 27/40
124/124 ━━━━━━━━━━━━━━━━━━━━ 40s 306ms/step - accuracy: 0.9473 - loss: 0.1638 - val_accuracy: 0.7369 - val_loss: 1.3365
```

```
def plot_training_model_history(history):
  accuracy=history.history['accuracy']
  val_accuracy=history.history['val_accuracy']
  loss=history.history['loss']
  val_loss=history.history['val_loss']
  epochs_range=range(40)

  plt.figure(figsize=(8,5))
  plt.subplot(1,2,1)
  plt.plot(epochs_range, accuracy, label='Training Accuracy')
  plt.plot(epochs_range, val_accuracy,label='Validation Accuracy')
  plt.legend(loc='lower right')
  plt.title('Training and Validation Accuracy')

  plt.subplot(1,2,2)
  plt.plot(epochs_range,loss,label='Training Loss')
  plt.plot(epochs_range,val_loss,label='Validation Loss')
  plt.legend(loc='upper right')
  plt.title('Training and Validatiion Loss')
  plt.show()
```

```
plot_training_model_history(model_history6)
```




```
test_loss, test_accuracy=inception_v3_1.evaluate(val_datagenerator.flow(X_val,y_val,batch_size=BATCH_SIZE)
print(f'Test accuracy:{test_accuracy}')
import random
#Make predictions and compare with true labels
def check_random_sample(model_history,X_val,y_val,class_names,num_samples=15):
  indices=random.sample(range(len(X_val)),num_samples)
  plt.figure(figsize=(20,45))
  for i, idx in enumerate(indices):
    img=X_val[idx]
    true_label=np.argmax(y_val[idx])
    prediction=inception_v3_1.predict(np.expand_dims(img,axis=0))
    predicted_label=np.argmax(prediction)

    plt.subplot(num_samples // 2+1,4,i+1)
    plt.imshow(img)
    plt.title(f'True:{class_names[true_label]},Pred:{class_names[predicted_label]}')
    plt.axis('off')
  plt.show()
```

```
#Check random samples
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: U erWarning: Your `PyDataset` class
  self._warn_if_super_not_called()
31/31 ━━━━━━━━━━━━━━━━━━━━ 6s 23ms/step - accuracy: 0.7335 - loss: 1.4417
Test accuracy:0.7520161271095276
1/1 ━━━━━━━━━━━━━━━━━━━━ 8s 8s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
```