```
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).
```

```
IMG_SAVE_PATH = '/content/drive/MyDrive/train'
```

```
Str_to_Int = {
    'Actinic keratosis': 0,
    'Atopic Dermatitis': 1,
    'Benign keratosis': 2,
    'Dermatofibroma': 3,
    'Melanocytic nevus': 4,
    'Melanoma':5,
    'Squamous cell carcinoma': 6,
    'Tinea Ringworm Candidiasis':7,
    'Vascular lesion': 8
}

NUM_CLASSES = 9


def str_to_Int_mapper(val):
    return Str_to_Int[val]
```

This Load and resize images, is computationally high(16m in this case)

```
# import os
# from PIL import Image  # Used ONLY for decoding image formats like JPG/PNG

# # Resize manually (nearest-neighbor)
# def resize_image(image, old_w, old_h, new_w=240, new_h=240):
#     resized = []
#     for i in range(new_h):
#         row = []
#         for j in range(new_w):
#             x = int(j * old_w / new_w)
#             y = int(i * old_h / new_h)
#             row.append(image[y][x])  # Nearest-neighbor pixel
#         resized.append(row)
#     return resized

# # Convert image to normalized pixel list
# def read_rgb_image(filepath):
#     try:
#         img = Image.open(filepath)
#         img = img.convert('RGB')  # Ensure it's in RGB mode
#         width, height = img.size
#         raw_pixels = list(img.getdata())  # Flat list of (r,g,b)

#         # Flat list converted to 2D normalized RGB matrix
#         rgb_image = []
#         for y in range(height):
#             row = []
#             for x in range(width):
#                 idx = y * width + x
#                 r, g, b = raw_pixels[idx]
#                 row.append([r / 255.0, g / 255.0, b / 255.0])  # Normalize
#             rgb_image.append(row)

#         return rgb_image, width, height
#     except Exception as e:
#         print(f"Error reading image {filepath}: {e}")
#         return None, None, None

# # Dataset creation
# IMG_SAVE_PATH = "/content/drive/MyDrive/train"  # Training data location

# dataset = []
# for directory in os.listdir(IMG_SAVE_PATH):
#     path = os.path.join(IMG_SAVE_PATH, directory)
#     if not os.path.isdir(path):
#         continue
#     for image in os.listdir(path):
#         new_path = os.path.join(path, image)
#         img, w, h = read_rgb_image(new_path)
#         if img is None:
#             continue
```

```
#           continue
#         resized_img = resize_image(img, w, h, 240, 240)
#         dataset.append([resized_img, directory])
```

For Faster Load and resize we use libraries(7 seconds)

```
import os
import numpy as np
from PIL import Image
import cv2

dataset = []
for directory in os.listdir(IMG_SAVE_PATH):
    path = os.path.join(IMG_SAVE_PATH, directory)
    for image in os.listdir(path):
        new_path = os.path.join(path, image)
        try:
            imgpath = Image.open(new_path)
            imgpath = imgpath.convert('RGB')                # Ensure 3 channels
            img = np.asarray(imgpath)
            img = cv2.resize(img, (240, 240))               # Resize to 240x240
            img = img / 255.0                               # Normalize
            dataset.append([img, directory])                # Append to dataset
        except FileNotFoundError:
            print(f'File not found: {new_path}')
        except Exception as e:
            print(f"Error processing {new_path}: {e}")
```

```
data = []
labels = []
for item in dataset:
    data.append(item[0])
    labels.append(item[1])

temp = []
for label in labels:
    temp.append(str_to_Int_mapper(label))
```

```
def to_categorical_manual(labels):
    num_classes = max(labels) + 1
    one_hot = []
    for label in labels:
        vec = [0] * num_classes
        vec[label] = 1
        one_hot.append(vec)
    return one_hot

labels = to_categorical_manual(temp)
```

DenseNet

```
from keras.applications import DenseNet121
from keras.layers import GlobalAveragePooling2D, Dropout, Dense, Input
from keras.models import Model
from tensorflow.keras.optimizers import Adam

# Load pretrained DenseNet (no top)
base_model = DenseNet121(weights='imagenet', include_top=False, input_shape=(240, 240, 3))

# Build custom model manually using Functional API
x = base_model.output
x = GlobalAveragePooling2D()(x)       # Use built-in layer
x = Dropout(0.5)(x)
outputs = Dense(9, activation='softmax')(x)

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

# Compile for training
model.compile(
    loss='categorical_crossentropy',
    optimizer=Adam(learning_rate=0.00005),
    metrics=['accuracy']
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_k
29084464/29084464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
# Define your model as a list of layers with their info
model_layers = [
    {"name": "DenseNet121", "output_shape": (None, 7, 7, 1024), "params": 7037504},
    {"name": "GlobalAveragePooling2D", "output_shape": (None, 1024), "params": 0},
    {"name": "Dropout", "output_shape": (None, 1024), "params": 0},
    {"name": "Dense", "output_shape": (None, 9), "params": 9225},
]

def print_model_summary(layers):
    print("Layer (type)                Output Shape              Param #")
    print("=================================================================")
    total_params = 0
    for layer in layers:
        print(f"{layer['name']:<25} {str(layer['output_shape']):<25} {layer['params']}")
        total_params += layer['params']
    print("=================================================================")
    print(f"Total params: {total_params}")

# Usage
print_model_summary(model_layers)
```

```
Layer (type)                Output Shape              Param #
=================================================================
DenseNet121               (None, 7, 7, 1024)        7037504
GlobalAveragePooling2D    (None, 1024)              0
Dropout                   (None, 1024)              0
Dense                     (None, 9)                 9225
=================================================================
Total params: 7046729
```

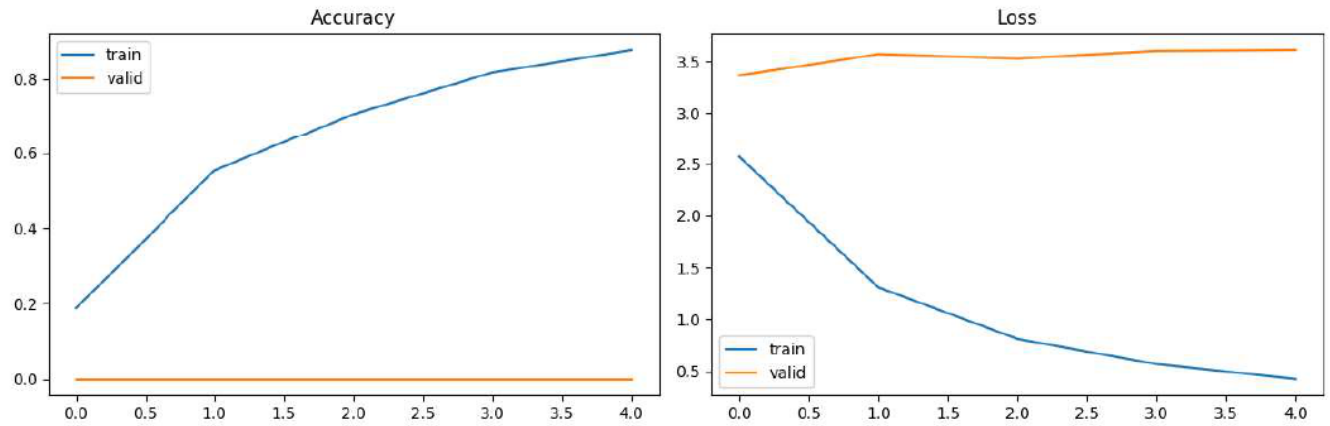
```
history=model.fit(np.array(data), np.array(labels), epochs = 5, shuffle = True, validation_split = 0.2)
```

```
Epoch 1/5
18/18 ━━━━━━━━━━━━━━━━━━━━ 575s 27s/step - accuracy: 0.1515 - loss: 2.8300 - val_accuracy: 0.0000e+00 - val_loss: 3.3649
Epoch 2/5
18/18 ━━━━━━━━━━━━━━━━━━━━ 474s 26s/step - accuracy: 0.5180 - loss: 1.4254 - val_accuracy: 0.0000e+00 - val_loss: 3.5687
Epoch 3/5
18/18 ━━━━━━━━━━━━━━━━━━━━ 486s 27s/step - accuracy: 0.6647 - loss: 0.8923 - val_accuracy: 0.0000e+00 - val_loss: 3.5272
Epoch 4/5
18/18 ━━━━━━━━━━━━━━━━━━━━ 494s 27s/step - accuracy: 0.8107 - loss: 0.5757 - val_accuracy: 0.0000e+00 - val_loss: 3.5989
Epoch 5/5
18/18 ━━━━━━━━━━━━━━━━━━━━ 486s 26s/step - accuracy: 0.8813 - loss: 0.4186 - val_accuracy: 0.0000e+00 - val_loss: 3.6077
```

```
from matplotlib import pyplot as plt

def plot_acc(history_dict):
    fig = plt.figure(figsize=(12, 4))

    ax1 = plt.subplot(1, 2, 1)
    ax1.plot(range(len(history_dict['accuracy'])), history_dict['accuracy'], label='train')
    ax1.plot(range(len(history_dict['val_accuracy'])), history_dict['val_accuracy'], label='valid')
    ax1.set_title('Accuracy')
    ax1.legend()
```

```
    ax2 = plt.subplot(1, 2, 2)
    ax2.plot(range(len(history_dict['loss'])), history_dict['loss'], label='train')
    ax2.plot(range(len(history_dict['val_loss'])), history_dict['val_loss'], label='valid')
    ax2.set_title('Loss')
    ax2.legend()

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

```
plot_acc(history.history)
```



Mobile Net

```
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.layers import Input, Dropout, Conv2D, Activation, GlobalAveragePooling2D
from tensorflow.keras.models import Model

def mobilenet():
    # Load pretrained MobileNet base without top layer
    base_model = MobileNet(input_shape=(240, 240, 3), include_top=False, weights='imagenet')

    # Add custom layers manually
    x = base_model.output
    x = Dropout(0.5)(x)
    x = Conv2D(9, (1, 1), padding='valid')(x)
    x = Activation('relu')(x)
    x = GlobalAveragePooling2D()(x)
    outputs = Activation('softmax')(x)

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

```
from tensorflow.keras.optimizers import Adam

model_mobile = mobilenet()
model_mobile.compile(
    optimizer=Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

```
/tmp/ipython-input-14-2030572540.py:7: UserWarning: `input_shape` is undefined or non-square, or `rows` is not in [128, 160, 192, 22
  base_model = MobileNet(input_shape=(240, 240, 3), include_top=False, weights='imagenet')
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_1_0_224_tf_no_top.h5
17225924/17225924 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
# #define summary function
# def custom_summary(model):
#     print("Layer (type)                 Output Shape              Param #")
#     print("=============================================================")
#     total_params = 0

#     for layer in model.layers:
```

```
#         name = layer.name
#         try:
#             output_shape = layer.output_shape
#         except AttributeError:
#             output_shape = "Not built"
#         try:
#             params = layer.count_params()
#         except:
#             params = 0
#         print(f"{name:<25} {str(output_shape):<25} {params}")
#         total_params += params

#     print("==============================================================")
#     print(f"Total params: {total_params}")
#     try:
#         trainable = sum([tf.keras.backend.count_params(p) for p in model.trainable_weights])
#         non_trainable = sum([tf.keras.backend.count_params(p) for p in model.non_trainable_weights])
#         print(f"Trainable params: {trainable}")
#         print(f"Non-trainable params: {non_trainable}")
#     except:
#         print("Trainable/non-trainable params not available.")


# #use summary function
# custom_summary(model_mobile)
```

```
# model_mobile.summary()
```

```
history_mobile = model_mobile.fit(
    np.array(data), np.array(labels),
    epochs=5, shuffle=True, validation_split=0.3
)
```

```
Epoch 1/5
16/16 ━━━━━━━━━━━━━━━━━━━━ 132s 7s/step - accuracy: 0.1977 - loss: 2.1175 - val_accuracy: 0.0476 - val_loss: 2.3381
Epoch 2/5
16/16 ━━━━━━━━━━━━━━━━━━━━ 137s 7s/step - accuracy: 0.7849 - loss: 0.8214 - val_accuracy: 0.0000e+00 - val_loss: 3.3249
Epoch 3/5
16/16 ━━━━━━━━━━━━━━━━━━━━ 110s 7s/step - accuracy: 0.9018 - loss: 0.4234 - val_accuracy: 0.0048 - val_loss: 3.5844
Epoch 4/5
16/16 ━━━━━━━━━━━━━━━━━━━━ 140s 7s/step - accuracy: 0.9760 - loss: 0.1628 - val_accuracy: 0.0381 - val_loss: 3.2609
Epoch 5/5
16/16 ━━━━━━━━━━━━━━━━━━━━ 142s 7s/step - accuracy: 0.9953 - loss: 0.0953 - val_accuracy: 0.0952 - val_loss: 2.9802
```

```
from matplotlib import pyplot as plt

def plot_acc(history_obj):
    if hasattr(history_obj, 'history'):
        history_dict = history_obj.history
    else:
        history_dict = history_obj

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

    # Plot Accuracy
    ax1.plot(range(len(history_dict['accuracy'])), history_dict['accuracy'], label='train')
    ax1.plot(range(len(history_dict['val_accuracy'])), history_dict['val_accuracy'], label='valid')
    ax1.set_title('Accuracy')
    ax1.legend()

    # Plot Loss
    ax2.plot(range(len(history_dict['loss'])), history_dict['loss'], label='train')
    ax2.plot(range(len(history_dict['val_loss'])), history_dict['val_loss'], label='valid')
    ax2.set_title('Loss')
    ax2.legend()

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

CNN

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten

def cnn():
    model=Sequential()
```

```
    model.add(Conv2D(8, kernel_size=(3,3), activation='relu', input_shape=(240,240,3)))
    model.add(Conv2D(16, kernel_size=(3,3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(9, activation='softmax'))

    return model
```

```
from keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten
from keras.optimizers import Adam

model_cnn = cnn()
model_cnn.compile(
    optimizer=Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# #define summary function
# def print_model_summary(layers):
#     print("Layer (type)                Output Shape              Param #")
#     print("=============================================================")
#     total_params = 0
#     for layer in layers:
#         print(f"{layer['name']:<25} {str(layer['output_shape']):<25} {layer['params']}")
#         total_params += layer['params']
#     print("=============================================================")
#     print(f"Total params: {total_params}")

# # Example usage:
# model_layers = [
#     {"name": "Conv2D", "output_shape": (None, 238, 238, 8), "params": 224},
#     {"name": "MaxPooling2D", "output_shape": (None, 119, 119, 8), "params": 0},
#     # ... add other layers manually
# ]

# #summary function
# print_model_summary(model_layers)
```

```
model_cnn.summary()
```

Model: "sequential"

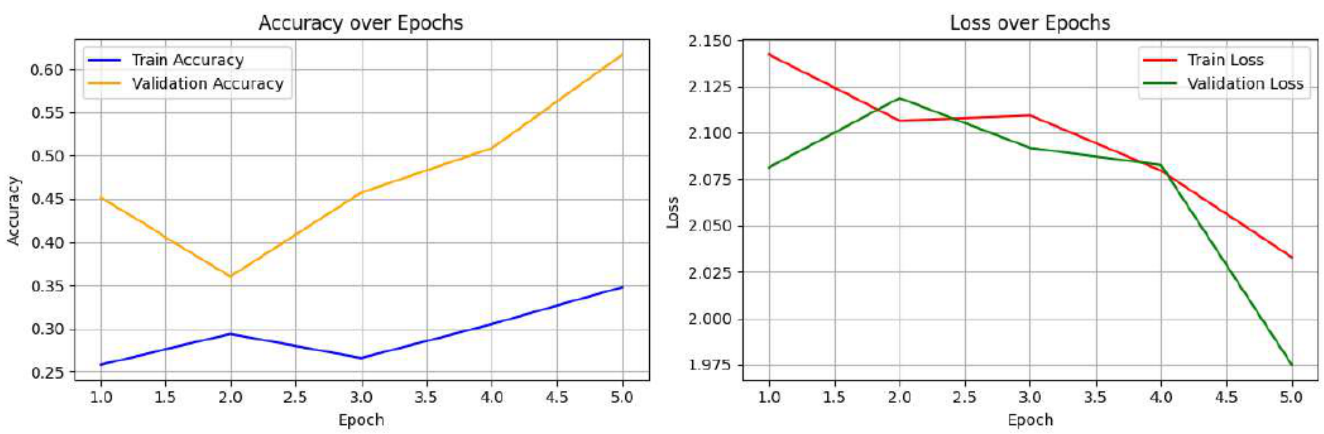
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 238, 238, 8)	224
conv2d_2 (Conv2D)	(None, 236, 236, 16)	1,168
max_pooling2d (MaxPooling2D)	(None, 118, 118, 16)	0
dropout_2 (Dropout)	(None, 118, 118, 16)	0
conv2d_3 (Conv2D)	(None, 116, 116, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 58, 58, 32)	0
conv2d_4 (Conv2D)	(None, 56, 56, 32)	9,248
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 32)	0
dropout_3 (Dropout)	(None, 28, 28, 32)	0
flatten (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 64)	1,605,696
dropout_4 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 9)	585

Total params: 1,621,561 (6.19 MB)
Trainable params: 1,621,561 (6.19 MB)

```
import numpy as np

epochs = 5
batch_size = 32
num_samples = len(data)
indices = np.arange(num_samples)

# Initialize custom history dictionary
history_manual = {
    'accuracy': [],
    'val_accuracy': [],
    'loss': [],
    'val_loss': []
}

for epoch in range(epochs):
    np.random.shuffle(indices)
    epoch_loss = 0
    epoch_acc = 0
    steps = 0

    for start_idx in range(0, num_samples, batch_size):
        end_idx = min(start_idx + batch_size, num_samples)
        batch_indices = indices[start_idx:end_idx]

        batch_data = np.array(data)[batch_indices]
        batch_labels = np.array(labels)[batch_indices]

        # One training step
        loss, acc = model_cnn.train_on_batch(batch_data, batch_labels)
        epoch_loss += loss
        epoch_acc += acc
        steps += 1

    # Average training metrics for the epoch
    avg_loss = epoch_loss / steps
    avg_acc = epoch_acc / steps

    # Validation step
    val_split = int(num_samples * 0.75)
    val_data = np.array(data)[val_split:]
    val_labels = np.array(labels)[val_split:]
    val_loss, val_acc = model_cnn.evaluate(val_data, val_labels, verbose=0)

    # Log values
    history_manual['accuracy'].append(avg_acc)
    history_manual['loss'].append(avg_loss)
    history_manual['val_accuracy'].append(val_acc)
    history_manual['val_loss'].append(val_loss)

    print(f"Epoch {epoch+1}/{epochs} -> Loss: {avg_loss:.4f}, Accuracy: {avg_acc:.4f}, Val_Loss: {val_loss:.4f}, Val_Accuracy: {val_acc
```

```
Epoch 1/5 -> Loss: 2.1422, Accuracy: 0.2581, Val_Loss: 2.0813, Val_Accuracy: 0.4514
Epoch 2/5 -> Loss: 2.1065, Accuracy: 0.2936, Val_Loss: 2.1187, Val_Accuracy: 0.3600
Epoch 3/5 -> Loss: 2.1094, Accuracy: 0.2658, Val_Loss: 2.0918, Val_Accuracy: 0.4571
Epoch 4/5 -> Loss: 2.0796, Accuracy: 0.3049, Val_Loss: 2.0826, Val_Accuracy: 0.5086
Epoch 5/5 -> Loss: 2.0329, Accuracy: 0.3480, Val_Loss: 1.9751, Val_Accuracy: 0.6171
```

```
import matplotlib.pyplot as plt

def plot_acc_minimal(history_dict):
    epochs = range(1, len(history_dict['accuracy']) + 1)

    plt.figure(figsize=(12, 4))

    # Accuracy plot
    plt.subplot(1, 2, 1)
    plt.plot(epochs, history_dict['accuracy'], label='Train Accuracy', color='blue')
    plt.plot(epochs, history_dict['val_accuracy'], label='Validation Accuracy', color='orange')
    plt.title('Accuracy over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    # Loss plot
    plt.subplot(1, 2, 2)
    plt.plot(epochs, history_dict['loss'], label='Train Loss', color='red')
    plt.plot(epochs, history_dict['val_loss'], label='Validation Loss', color='green')
    plt.title('Loss over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()

# Call the plot
plot_acc_minimal(history_manual)
```



Save Model

```
model.save('/content/drive/MyDrive/final_project/SkinDiseasePredictionLater.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is 
```

Evaluate Model

```
IMG_SAVE_PATH_TESTING = '/content/drive/MyDrive/val'
```

```
import os
from PIL import Image
import numpy as np

dataset_testing = []
```

```
IMG_SAVE_PATH_TESTING = '/content/drive/MyDrive/val'

for directory in os.listdir(IMG_SAVE_PATH_TESTING):
    path = os.path.join(IMG_SAVE_PATH_TESTING, directory)
    for image in os.listdir(path):
        new_path = os.path.join(path, image)
        try:
            imgpath = Image.open(new_path)
            imgpath = imgpath.convert('RGB')                # Ensure 3-channel image
            imgpath = imgpath.resize((240, 240))            # Resize using PIL instead of cv2
            img = np.asarray(imgpath) / 255.0               # Normalize
            dataset_testing.append([img, directory])        # Store image and label
        except Exception as e:
            print(f"Error processing {new_path}: {e}")
```

```
# Unpack images and labels from the testing dataset
testing_data, testing_labels = zip(*dataset_testing)

# Map class names (strings) to integers using your dictionary
testing_temp = [Str_to_Int[label] for label in testing_labels]
```

```
import numpy as np

def to_one_hot(labels, num_classes):
    one_hot = np.zeros((len(labels), num_classes))
    for idx, val in enumerate(labels):
        one_hot[idx][val] = 1
    return one_hot

# Convert testing_temp to one-hot labels
NUM_CLASSES = 9
testing_labels = to_one_hot(testing_temp, NUM_CLASSES)
```

```
model.evaluate(np.array(testing_data), np.array(testing_labels))
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 44s 6s/step - accuracy: 0.4708 - loss: 1.7223
[1.5184286832809448, 0.5359116196632385]
```

```
model_mobile.evaluate(np.array(testing_data), np.array(testing_labels))
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 9s 1s/step - accuracy: 0.3951 - loss: 1.8256
[1.7074254751205444, 0.4364641010761261]
```

```
import numpy as np

num_classes = 9
testing_labels_one_hot = np.zeros((len(testing_temp), num_classes))
for i, label in enumerate(testing_temp):
    testing_labels_one_hot[i, label] = 1
```

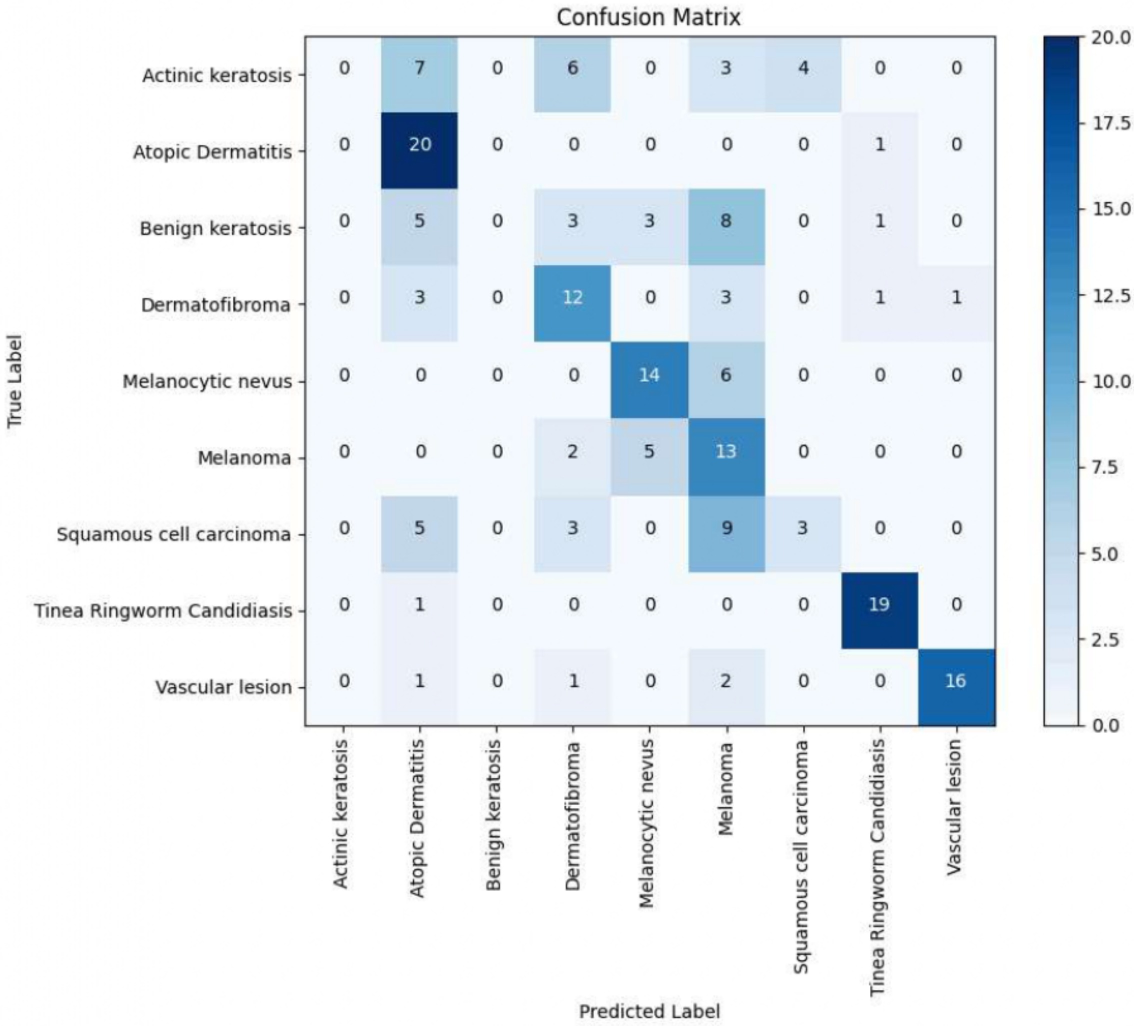
Prediction

```
import numpy as np

# Assuming testing_labels is one-hot encoded (shape: [samples, num_classes])
true_labels = np.argmax(testing_labels, axis=1)

# Predict probabilities with your model, then get predicted classes
pred_probs = model.predict(np.array(testing_data))
pred_labels = np.argmax(pred_probs, axis=1)

# Now call confusion matrix function
def confusion_matrix_manual(true_labels, pred_labels, num_classes):
    cm = np.zeros((num_classes, num_classes), dtype=int)
    for t, p in zip(true_labels, pred_labels):
        cm[t, p] += 1
    return cm

num_classes = 9
cm_manual = confusion_matrix_manual(true_labels, pred_labels, num_classes)

# Print confusion matrix
print("Confusion Matrix:")
for i in range(num_classes):
    print(cm_manual[i])
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 39s 6s/step
Confusion Matrix:
[0 7 0 6 0 3 4 0 0]
[ 0 20  0  0  0  0  0  1  0]
[0 5 0 3 3 8 0 1 0]
[ 0  3  0 12  0  3  0  1  1]
[ 0  0  0  0 14  6  0  0  0]
[ 0  0  0  2  5 13  0  0  0]
[0 5 0 3 0 9 3 0 0]
[ 0  1  0  0  0  0  0 19  0]
[ 0  1  0  1  0  2  0  0 16]
```

Double-click (or enter) to edit

```
import numpy as np
import matplotlib.pyplot as plt

# Compute confusion matrix manually
def confusion_matrix_manual(true_labels, pred_labels, num_classes):
    cm = np.zeros((num_classes, num_classes), dtype=int)
    for t, p in zip(true_labels, pred_labels):
        cm[t, p] += 1
    return cm

num_classes = 9
cm = confusion_matrix_manual(true_labels, pred_labels, num_classes)

# Plot using matplotlib
plt.figure(figsize=(10,8))
plt.imshow(cm, interpolation='nearest', cmap='Blues')
plt.title('Confusion Matrix')
plt.colorbar()

classes = list(Str_to_Int.keys())
tick_marks = np.arange(num_classes)
plt.xticks(tick_marks, classes, rotation=90)
plt.yticks(tick_marks, classes)

# Label each cell with counts
thresh = cm.max() / 2
for i in range(num_classes):
    for j in range(num_classes):
        plt.text(j, i, format(cm[i, j], 'd'),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.tight_layout()
plt.show()
```




```
def make_preds(model, data):
    preds_probs = model.predict(data)
    preds = preds_probs.argmax(axis=1)
    return preds
```

```
testing_data, testing_labels = zip(*dataset_testing)
testing_data = np.array(testing_data)
```

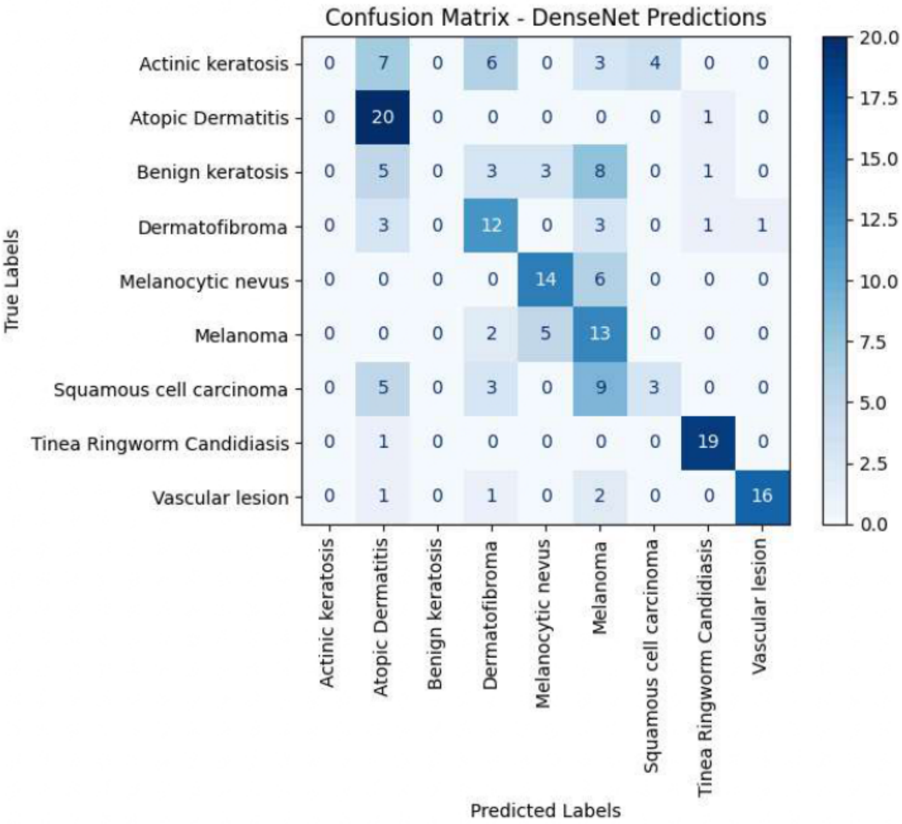
```
pred_labels_densenet = make_preds(model, testing_data)
pred_labels_mobilenet = make_preds(model_mobile, testing_data)
pred_labels_cnn = make_preds(model_cnn, testing_data)
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 31s 5s/step
6/6 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step
6/6 ━━━━━━━━━━━━━━━━━━━━ 3s 464ms/step
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

def plot_confusion_matrix(true_labels, pred_labels, label_map, title):
    cm = confusion_matrix(true_labels, pred_labels)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=list(label_map.keys())
    disp.plot(cmap=plt.cm.Blues, xticks_rotation='vertical')
    plt.title(title)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.show()

# Example usage (assuming you have predictions and true labels):

# true_labels = [0, 1, 2, 1, 0, ...]  # numeric labels
# pred_labels = [0, 2, 2, 1, 0, ...]
# Str_to_Int = {
#     'Actinic keratosis': 0,
#     'Atopic Dermatitis': 1,
#     'Benign keratosis': 2,
#     'Dermatofibroma': 3,
```

```
#     'Melanocytic nevus': 4,
#     'Melanoma':5,
#     'Squamous cell carcinoma': 6,
#     'Tinea Ringworm Candidiasis':7,
#     'Vascular lesion': 8
# }

# Then call:
# plot_confusion_matrix(true_labels, pred_labels, Str_to_Int, "DenseNet Confusion Matrix")
```

```
plot_confusion_matrix(true_labels, pred_labels_densenet, Str_to_Int, "Confusion Matrix - DenseNet Predictions")
```




```
plot_confusion_matrix(true_labels, pred_labels_mobilenet, Str_to_Int, "Confusion Matrix - MobileNet Predictions")
```