EfficientNetV2M

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

```
Mounted at /content/drive
```

```
import os
import shutil
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
import sklearn
from sklearn.metrics import classification_report
from keras.applications import *
from keras.layers import *
from keras.models import Model, load_model
from keras.optimizers import Adam
from sklearn.utils import class_weight
from tqdm import tqdm
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.efficientnet_v2 import preprocess_input as base_preprocess
from keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve, RocCurveDisplay,
auc
from sklearn.utils.multiclass import unique_labels
from collections import Counter
from pathlib import Path
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from PIL import Image
from sklearn.preprocessing import LabelEncoder
from sklearn.utils.class_weight import compute_class_weight
import itertools
```

Load data

```
data = np.load('/content/drive/MyDrive/SC Lab/Dataset/data.npy',mmap_mode='r')
labels = np.load('/content/drive/MyDrive/SC Lab/Dataset/labels.npy',mmap_mode='r')
```

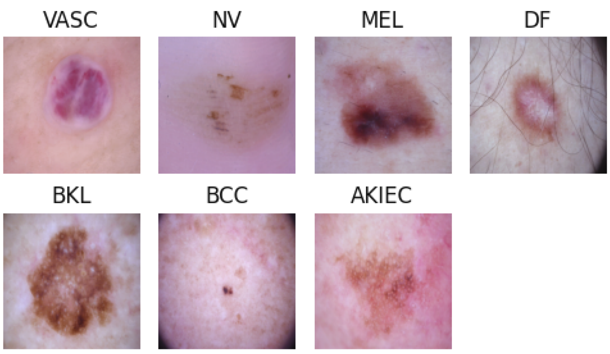
```
print("Data shape:", data.shape)
print("Labels shape:", labels.shape)
```

```
Data shape: (10015, 224, 224, 3)
Labels shape: (10015, 7)
```

Loading Images from the data

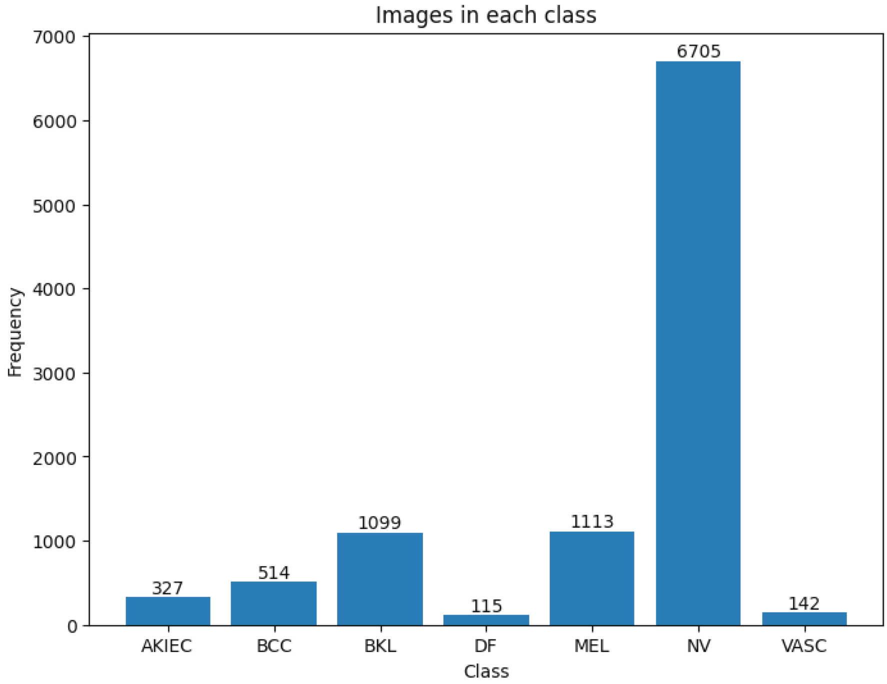
```
# Get unique class labels and their corresponding indices in the data array
unique_classes = np.unique(labels, axis=1)

# Create a dictionary to store one data sample from each class
```

```
class_samples = {}

# Map class indices to their corresponding names
class_names = {0: "AKIEC", 1: "BCC", 2: "BKL", 3: "DF", 4: "MEL", 5: "NV", 6: "VASC"}

# Select one data sample from each class
for class_label in unique_classes:
    class_indices = np.where(np.all(labels == class_label, axis=1))[0]
    class_samples[tuple(class_label)] = data[class_indices[0]]

# Plot the images in 2 rows
plt.figure(figsize=(5, 3))
for i, (class_label, image_data) in enumerate(class_samples.items()):
    class_index = np.argmax(class_label)  # Get the index of the class
    class_name = class_names[class_index]  # Get the corresponding class name
    plt.subplot(2, 4, i + 1)
    plt.imshow(image_data)
    plt.title(f'{class_name}')
    plt.axis('off')

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



Frequency of the data

```
# Sum the one-hot encoded labels along the rows to get the frequency of each class
class_counts = np.sum(labels, axis=0)

# Map class indices to their corresponding names
class_names = {0: "AKIEC", 1: "BCC", 2: "BKL", 3: "DF", 4: "MEL", 5: "NV", 6: "VASC"}

# Plot the class frequencies
plt.figure(figsize=(8, 6))
plt.bar([class_names[class_idx] for class_idx in range(len(class_names))], class_counts)
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.title('Images in each class')

# Annotate the bars with the class frequencies (integer format)
for i, count in enumerate(class_counts):
    plt.text(i, count, str(int(count)), ha='center', va='bottom')

plt.show()
```



Split data

```
# Split the data into train, test, and validation sets
train_data, test_data, train_labels, test_labels = train_test_split(data, labels, test_size=0.1,stratify=labels,
random_state=42)
train_data, val_data, train_labels, val_labels = train_test_split(train_data, train_labels,
test_size=0.1,stratify=train_labels, random_state=42)
```

Augmentation

```
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)
```

Model

```
lr_reduce = ReduceLROnPlateau(monitor = 'val_accuracy', factor = 0.5, patience = 5,mode='max', min_lr = 1e-
4,verbose = 1)
saved_model = '/content/drive/MyDrive/SC Lab/Saved Model/EfficientNetV2M.h5'
model_chkpt = ModelCheckpoint(saved_model ,save_best_only = True, monitor = 'val_accuracy',verbose = 1)
callback_list = [model_chkpt, lr_reduce]
```

```
base_model = EfficientNetV2M(weights='imagenet',
                       include_top=False,
                       input_shape=(224, 224, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/efficientnet_v2/efficientnetv2-m_notop.h5
214201816/214201816 [==============================] - 7s 0us/step
```

```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(7, activation='softmax')(x)

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


# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
epochs = 50
batch_size = 16
```

```
history = model.fit(datagen.flow(train_data, train_labels, batch_size=batch_size),
                    validation_data=(val_data, val_labels),
                    epochs=epochs,
                    callbacks=callback_list)
```

```
Epoch 1/50
507/507 [==============================] - ETA: 0s - loss: 1.0207 - accuracy: 0.6561
Epoch 1: val_accuracy improved from -inf to 0.70288, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 304s 515ms/step - loss: 1.0207 - accuracy: 0.6561 - val_loss: 0.8485
- val_accuracy: 0.7029 - lr: 0.0010
Epoch 2/50
507/507 [==============================] - ETA: 0s - loss: 0.9174 - accuracy: 0.6809
Epoch 2: val_accuracy did not improve from 0.70288
507/507 [==============================] - 247s 486ms/step - loss: 0.9174 - accuracy: 0.6809 - val_loss: 1.0370
- val_accuracy: 0.6918 - lr: 0.0010
Epoch 3/50
507/507 [==============================] - ETA: 0s - loss: 0.8436 - accuracy: 0.6917
Epoch 3: val_accuracy improved from 0.70288 to 0.72506, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 267s 525ms/step - loss: 0.8436 - accuracy: 0.6917 - val_loss: 0.7658
- val_accuracy: 0.7251 - lr: 0.0010
Epoch 4/50
507/507 [==============================] - ETA: 0s - loss: 0.8079 - accuracy: 0.7047
Epoch 4: val_accuracy improved from 0.72506 to 0.73171, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 307s 606ms/step - loss: 0.8079 - accuracy: 0.7047 - val_loss: 0.7274
- val_accuracy: 0.7317 - lr: 0.0010
Epoch 5/50
507/507 [==============================] - ETA: 0s - loss: 0.7706 - accuracy: 0.7110
Epoch 5: val_accuracy did not improve from 0.73171
507/507 [==============================] - 245s 484ms/step - loss: 0.7706 - accuracy: 0.7110 - val_loss: 0.7952
- val_accuracy: 0.7140 - lr: 0.0010
Epoch 6/50
507/507 [==============================] - ETA: 0s - loss: 0.7487 - accuracy: 0.7275
Epoch 6: val_accuracy improved from 0.73171 to 0.73725, saving model to /content/drive/MyDrive/4.2/SC
```

```
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 369s 729ms/step - loss: 0.7487 - accuracy: 0.7275 - val_loss: 1.8249
- val_accuracy: 0.7373 - lr: 0.0010
Epoch 7/50
507/507 [==============================] - ETA: 0s - loss: 0.7571 - accuracy: 0.7299
Epoch 7: val_accuracy improved from 0.73725 to 0.74501, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 419s 828ms/step - loss: 0.7571 - accuracy: 0.7299 - val_loss: 0.7299
- val_accuracy: 0.7450 - lr: 0.0010
Epoch 8/50
507/507 [==============================] - ETA: 0s - loss: 0.7274 - accuracy: 0.7381
Epoch 8: val_accuracy improved from 0.74501 to 0.77051, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 423s 834ms/step - loss: 0.7274 - accuracy: 0.7381 - val_loss: 0.6681
- val_accuracy: 0.7705 - lr: 0.0010
Epoch 9/50
507/507 [==============================] - ETA: 0s - loss: 0.7057 - accuracy: 0.7420
Epoch 9: val_accuracy did not improve from 0.77051
507/507 [==============================] - 240s 474ms/step - loss: 0.7057 - accuracy: 0.7420 - val_loss: 0.6533
- val_accuracy: 0.7661 - lr: 0.0010
Epoch 10/50
507/507 [==============================] - ETA: 0s - loss: 0.6908 - accuracy: 0.7477
Epoch 10: val_accuracy did not improve from 0.77051
507/507 [==============================] - 238s 470ms/step - loss: 0.6908 - accuracy: 0.7477 - val_loss: 0.7102
- val_accuracy: 0.7517 - lr: 0.0010
Epoch 11/50
507/507 [==============================] - ETA: 0s - loss: 0.6814 - accuracy: 0.7476
Epoch 11: val_accuracy did not improve from 0.77051
507/507 [==============================] - 244s 482ms/step - loss: 0.6814 - accuracy: 0.7476 - val_loss: 0.6477
- val_accuracy: 0.7705 - lr: 0.0010
Epoch 12/50
507/507 [==============================] - ETA: 0s - loss: 0.6616 - accuracy: 0.7597
Epoch 12: val_accuracy improved from 0.77051 to 0.77716, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 435s 858ms/step - loss: 0.6616 - accuracy: 0.7597 - val_loss: 0.6594
- val_accuracy: 0.7772 - lr: 0.0010
Epoch 13/50
507/507 [==============================] - ETA: 0s - loss: 0.6540 - accuracy: 0.7618
Epoch 13: val_accuracy did not improve from 0.77716
507/507 [==============================] - 246s 484ms/step - loss: 0.6540 - accuracy: 0.7618 - val_loss: 0.6534
- val_accuracy: 0.7528 - lr: 0.0010
Epoch 14/50
507/507 [==============================] - ETA: 0s - loss: 0.6365 - accuracy: 0.7688
Epoch 14: val_accuracy did not improve from 0.77716
507/507 [==============================] - 242s 476ms/step - loss: 0.6365 - accuracy: 0.7688 - val_loss: 0.7057
- val_accuracy: 0.7705 - lr: 0.0010
Epoch 15/50
507/507 [==============================] - ETA: 0s - loss: 0.6188 - accuracy: 0.7714
Epoch 15: val_accuracy improved from 0.77716 to 0.78936, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 452s 893ms/step - loss: 0.6188 - accuracy: 0.7714 - val_loss: 0.5798
- val_accuracy: 0.7894 - lr: 0.0010
```

```
Epoch 16/50
507/507 [==============================] - ETA: 0s - loss: 0.6234 - accuracy: 0.7786
Epoch 16: val_accuracy did not improve from 0.78936
507/507 [==============================] - 248s 489ms/step - loss: 0.6234 - accuracy: 0.7786 - val_loss: 0.6436
- val_accuracy: 0.7517 - lr: 0.0010
Epoch 17/50
507/507 [==============================] - ETA: 0s - loss: 0.6076 - accuracy: 0.7748
Epoch 17: val_accuracy did not improve from 0.78936
507/507 [==============================] - 242s 478ms/step - loss: 0.6076 - accuracy: 0.7748 - val_loss: 0.6094
- val_accuracy: 0.7816 - lr: 0.0010
Epoch 18/50
507/507 [==============================] - ETA: 0s - loss: 0.5913 - accuracy: 0.7836
Epoch 18: val_accuracy improved from 0.78936 to 0.79047, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 418s 825ms/step - loss: 0.5913 - accuracy: 0.7836 - val_loss: 0.6671
- val_accuracy: 0.7905 - lr: 0.0010
Epoch 19/50
507/507 [==============================] - ETA: 0s - loss: 0.5853 - accuracy: 0.7893
Epoch 19: val_accuracy did not improve from 0.79047
507/507 [==============================] - 247s 486ms/step - loss: 0.5853 - accuracy: 0.7893 - val_loss: 0.6973
- val_accuracy: 0.7616 - lr: 0.0010
Epoch 20/50
507/507 [==============================] - ETA: 0s - loss: 0.5761 - accuracy: 0.7925
Epoch 20: val_accuracy improved from 0.79047 to 0.79601, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 418s 825ms/step - loss: 0.5761 - accuracy: 0.7925 - val_loss: 0.5472
- val_accuracy: 0.7960 - lr: 0.0010
Epoch 21/50
507/507 [==============================] - ETA: 0s - loss: 0.5605 - accuracy: 0.7945
Epoch 21: val_accuracy improved from 0.79601 to 0.80710, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 420s 829ms/step - loss: 0.5605 - accuracy: 0.7945 - val_loss: 0.6051
- val_accuracy: 0.8071 - lr: 0.0010
Epoch 22/50
507/507 [==============================] - ETA: 0s - loss: 0.5484 - accuracy: 0.7985
Epoch 22: val_accuracy did not improve from 0.80710
507/507 [==============================] - 240s 474ms/step - loss: 0.5484 - accuracy: 0.7985 - val_loss: 0.6721
- val_accuracy: 0.7605 - lr: 0.0010
Epoch 23/50
507/507 [==============================] - ETA: 0s - loss: 0.5309 - accuracy: 0.8082
Epoch 23: val_accuracy improved from 0.80710 to 0.81596, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 445s 879ms/step - loss: 0.5309 - accuracy: 0.8082 - val_loss: 0.5370
- val_accuracy: 0.8160 - lr: 0.0010
Epoch 24/50
507/507 [==============================] - ETA: 0s - loss: 0.5181 - accuracy: 0.8116
Epoch 24: val_accuracy did not improve from 0.81596
507/507 [==============================] - 240s 473ms/step - loss: 0.5181 - accuracy: 0.8116 - val_loss: 0.5938
- val_accuracy: 0.8126 - lr: 0.0010
Epoch 25/50
507/507 [==============================] - ETA: 0s - loss: 0.5117 - accuracy: 0.8117
Epoch 25: val_accuracy did not improve from 0.81596
```

```
507/507 [==============================] - 244s 481ms/step - loss: 0.5117 - accuracy: 0.8117 - val_loss: 0.5634
- val_accuracy: 0.8004 - lr: 0.0010
Epoch 26/50
507/507 [==============================] - ETA: 0s - loss: 0.5027 - accuracy: 0.8157
Epoch 26: val_accuracy did not improve from 0.81596
507/507 [==============================] - 246s 484ms/step - loss: 0.5027 - accuracy: 0.8157 - val_loss: 0.5145
- val_accuracy: 0.8071 - lr: 0.0010
Epoch 27/50
507/507 [==============================] - ETA: 0s - loss: 0.4856 - accuracy: 0.8222
Epoch 27: val_accuracy did not improve from 0.81596
507/507 [==============================] - 244s 481ms/step - loss: 0.4856 - accuracy: 0.8222 - val_loss: 0.5999
- val_accuracy: 0.7971 - lr: 0.0010
Epoch 28/50
507/507 [==============================] - ETA: 0s - loss: 0.4963 - accuracy: 0.8209
Epoch 28: val_accuracy did not improve from 0.81596

Epoch 28: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
507/507 [==============================] - 245s 482ms/step - loss: 0.4963 - accuracy: 0.8209 - val_loss: 0.6489
- val_accuracy: 0.7794 - lr: 0.0010
Epoch 29/50
507/507 [==============================] - ETA: 0s - loss: 0.4225 - accuracy: 0.8468
Epoch 29: val_accuracy did not improve from 0.81596
507/507 [==============================] - 245s 482ms/step - loss: 0.4225 - accuracy: 0.8468 - val_loss: 0.5090
- val_accuracy: 0.8115 - lr: 5.0000e-04
Epoch 30/50
507/507 [==============================] - ETA: 0s - loss: 0.3959 - accuracy: 0.8543
Epoch 30: val_accuracy improved from 0.81596 to 0.82483, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 426s 840ms/step - loss: 0.3959 - accuracy: 0.8543 - val_loss: 0.5096
- val_accuracy: 0.8248 - lr: 5.0000e-04
Epoch 31/50
507/507 [==============================] - ETA: 0s - loss: 0.3757 - accuracy: 0.8664
Epoch 31: val_accuracy did not improve from 0.82483
507/507 [==============================] - 244s 482ms/step - loss: 0.3757 - accuracy: 0.8664 - val_loss: 0.4972
- val_accuracy: 0.8237 - lr: 5.0000e-04
Epoch 32/50
507/507 [==============================] - ETA: 0s - loss: 0.3642 - accuracy: 0.8625
Epoch 32: val_accuracy improved from 0.82483 to 0.84257, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 435s 859ms/step - loss: 0.3642 - accuracy: 0.8625 - val_loss: 0.4549
- val_accuracy: 0.8426 - lr: 5.0000e-04
Epoch 33/50
507/507 [==============================] - ETA: 0s - loss: 0.3578 - accuracy: 0.8735
Epoch 33: val_accuracy did not improve from 0.84257
507/507 [==============================] - 244s 481ms/step - loss: 0.3578 - accuracy: 0.8735 - val_loss: 0.5940
- val_accuracy: 0.8137 - lr: 5.0000e-04
Epoch 34/50
507/507 [==============================] - ETA: 0s - loss: 0.3421 - accuracy: 0.8740
Epoch 34: val_accuracy did not improve from 0.84257
507/507 [==============================] - 239s 471ms/step - loss: 0.3421 - accuracy: 0.8740 - val_loss: 0.4796
- val_accuracy: 0.8415 - lr: 5.0000e-04
Epoch 35/50
```

```
507/507 [==============================] - ETA: 0s - loss: 0.3326 - accuracy: 0.8811
Epoch 35: val_accuracy did not improve from 0.84257
507/507 [==============================] - 244s 480ms/step - loss: 0.3326 - accuracy: 0.8811 - val_loss: 0.4896
- val_accuracy: 0.8348 - lr: 5.0000e-04
Epoch 36/50
507/507 [==============================] - ETA: 0s - loss: 0.3234 - accuracy: 0.8802
Epoch 36: val_accuracy did not improve from 0.84257
507/507 [==============================] - 239s 471ms/step - loss: 0.3234 - accuracy: 0.8802 - val_loss: 0.4775
- val_accuracy: 0.8348 - lr: 5.0000e-04
Epoch 37/50
507/507 [==============================] - ETA: 0s - loss: 0.3114 - accuracy: 0.8835
Epoch 37: val_accuracy did not improve from 0.84257

Epoch 37: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
507/507 [==============================] - 244s 480ms/step - loss: 0.3114 - accuracy: 0.8835 - val_loss: 0.4948
- val_accuracy: 0.8293 - lr: 5.0000e-04
Epoch 38/50
507/507 [==============================] - ETA: 0s - loss: 0.2701 - accuracy: 0.9043
Epoch 38: val_accuracy did not improve from 0.84257
507/507 [==============================] - 242s 477ms/step - loss: 0.2701 - accuracy: 0.9043 - val_loss: 0.4944
- val_accuracy: 0.8293 - lr: 2.5000e-04
Epoch 39/50
507/507 [==============================] - ETA: 0s - loss: 0.2470 - accuracy: 0.9106
Epoch 39: val_accuracy did not improve from 0.84257
507/507 [==============================] - 248s 488ms/step - loss: 0.2470 - accuracy: 0.9106 - val_loss: 0.4989
- val_accuracy: 0.8282 - lr: 2.5000e-04
Epoch 40/50
507/507 [==============================] - ETA: 0s - loss: 0.2381 - accuracy: 0.9138
Epoch 40: val_accuracy improved from 0.84257 to 0.84368, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 424s 837ms/step - loss: 0.2381 - accuracy: 0.9138 - val_loss: 0.5161
- val_accuracy: 0.8437 - lr: 2.5000e-04
Epoch 41/50
507/507 [==============================] - ETA: 0s - loss: 0.2327 - accuracy: 0.9143
Epoch 41: val_accuracy did not improve from 0.84368
507/507 [==============================] - 248s 488ms/step - loss: 0.2327 - accuracy: 0.9143 - val_loss: 0.5415
- val_accuracy: 0.8370 - lr: 2.5000e-04
Epoch 42/50
507/507 [==============================] - ETA: 0s - loss: 0.2185 - accuracy: 0.9184
Epoch 42: val_accuracy did not improve from 0.84368
507/507 [==============================] - 246s 485ms/step - loss: 0.2185 - accuracy: 0.9184 - val_loss: 0.5712
- val_accuracy: 0.8392 - lr: 2.5000e-04
Epoch 43/50
507/507 [==============================] - ETA: 0s - loss: 0.2150 - accuracy: 0.9228
Epoch 43: val_accuracy did not improve from 0.84368
507/507 [==============================] - 248s 489ms/step - loss: 0.2150 - accuracy: 0.9228 - val_loss: 0.5433
- val_accuracy: 0.8370 - lr: 2.5000e-04
Epoch 44/50
507/507 [==============================] - ETA: 0s - loss: 0.2104 - accuracy: 0.9237
Epoch 44: val_accuracy did not improve from 0.84368
507/507 [==============================] - 243s 479ms/step - loss: 0.2104 - accuracy: 0.9237 - val_loss: 0.5693
- val_accuracy: 0.8215 - lr: 2.5000e-04
```

```
Epoch 45/50
507/507 [==============================] - ETA: 0s - loss: 0.2090 - accuracy: 0.9254
Epoch 45: val_accuracy improved from 0.84368 to 0.85809, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 452s 893ms/step - loss: 0.2090 - accuracy: 0.9254 - val_loss: 0.5159
- val_accuracy: 0.8581 - lr: 2.5000e-04
Epoch 46/50
507/507 [==============================] - ETA: 0s - loss: 0.1924 - accuracy: 0.9319
Epoch 46: val_accuracy did not improve from 0.85809
507/507 [==============================] - 241s 475ms/step - loss: 0.1924 - accuracy: 0.9319 - val_loss: 0.5985
- val_accuracy: 0.8237 - lr: 2.5000e-04
Epoch 47/50
507/507 [==============================] - ETA: 0s - loss: 0.1893 - accuracy: 0.9349
Epoch 47: val_accuracy did not improve from 0.85809
507/507 [==============================] - 247s 487ms/step - loss: 0.1893 - accuracy: 0.9349 - val_loss: 0.5397
- val_accuracy: 0.8448 - lr: 2.5000e-04
Epoch 48/50
507/507 [==============================] - ETA: 0s - loss: 0.1949 - accuracy: 0.9297
Epoch 48: val_accuracy did not improve from 0.85809
507/507 [==============================] - 247s 487ms/step - loss: 0.1949 - accuracy: 0.9297 - val_loss: 0.5648
- val_accuracy: 0.8271 - lr: 2.5000e-04
Epoch 49/50
507/507 [==============================] - ETA: 0s - loss: 0.1852 - accuracy: 0.9313
Epoch 49: val_accuracy did not improve from 0.85809
507/507 [==============================] - 249s 491ms/step - loss: 0.1852 - accuracy: 0.9313 - val_loss: 0.5471
- val_accuracy: 0.8537 - lr: 2.5000e-04
Epoch 50/50
507/507 [==============================] - ETA: 0s - loss: 0.1704 - accuracy: 0.9377
Epoch 50: val_accuracy did not improve from 0.85809

Epoch 50: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
507/507 [==============================] - 243s 479ms/step - loss: 0.1704 - accuracy: 0.9377 - val_loss: 0.5821
- val_accuracy: 0.8348 - lr: 2.5000e-04
```

```
model= load_model('/content/drive/MyDrive/4.2/SC Lab/Project/Saved Model/EfficientNetV2M.h5')
```

```
history = model.fit(datagen.flow(train_data, train_labels, batch_size=batch_size),
                    validation_data=(val_data, val_labels),
                    epochs=10,
                    callbacks=callback_list)
```

```
Epoch 1/10
507/507 [==============================] - ETA: 0s - loss: 0.1978 - accuracy: 0.9278
Epoch 1: val_accuracy improved from -inf to 0.85144, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 322s 573ms/step - loss: 0.1978 - accuracy: 0.9278 - val_loss: 0.4915
- val_accuracy: 0.8514 - lr: 2.5000e-04
Epoch 2/10
507/507 [==============================] - ETA: 0s - loss: 0.1893 - accuracy: 0.9343
Epoch 2: val_accuracy did not improve from 0.85144
507/507 [==============================] - 248s 488ms/step - loss: 0.1893 - accuracy: 0.9343 - val_loss: 0.5404
- val_accuracy: 0.8404 - lr: 2.5000e-04
```

```
Epoch 3/10
507/507 [==============================] - ETA: 0s - loss: 0.1936 - accuracy: 0.9290
Epoch 3: val_accuracy did not improve from 0.85144
507/507 [==============================] - 247s 487ms/step - loss: 0.1936 - accuracy: 0.9290 - val_loss: 0.4909
- val_accuracy: 0.8503 - lr: 2.5000e-04
Epoch 4/10
507/507 [==============================] - ETA: 0s - loss: 0.1887 - accuracy: 0.9344
Epoch 4: val_accuracy did not improve from 0.85144
507/507 [==============================] - 247s 487ms/step - loss: 0.1887 - accuracy: 0.9344 - val_loss: 0.5190
- val_accuracy: 0.8459 - lr: 2.5000e-04
Epoch 5/10
507/507 [==============================] - ETA: 0s - loss: 0.1757 - accuracy: 0.9403
Epoch 5: val_accuracy did not improve from 0.85144
507/507 [==============================] - 243s 480ms/step - loss: 0.1757 - accuracy: 0.9403 - val_loss: 0.5448
- val_accuracy: 0.8415 - lr: 2.5000e-04
Epoch 6/10
507/507 [==============================] - ETA: 0s - loss: 0.1683 - accuracy: 0.9388
Epoch 6: val_accuracy did not improve from 0.85144

Epoch 6: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
507/507 [==============================] - 243s 479ms/step - loss: 0.1683 - accuracy: 0.9388 - val_loss: 0.5250
- val_accuracy: 0.8437 - lr: 2.5000e-04
Epoch 7/10
507/507 [==============================] - ETA: 0s - loss: 0.1525 - accuracy: 0.9464
Epoch 7: val_accuracy did not improve from 0.85144
507/507 [==============================] - 247s 488ms/step - loss: 0.1525 - accuracy: 0.9464 - val_loss: 0.5411
- val_accuracy: 0.8448 - lr: 1.2500e-04
Epoch 8/10
507/507 [==============================] - ETA: 0s - loss: 0.1361 - accuracy: 0.9528
Epoch 8: val_accuracy did not improve from 0.85144
507/507 [==============================] - 242s 476ms/step - loss: 0.1361 - accuracy: 0.9528 - val_loss: 0.5861
- val_accuracy: 0.8470 - lr: 1.2500e-04
Epoch 9/10
507/507 [==============================] - ETA: 0s - loss: 0.1313 - accuracy: 0.9520
Epoch 9: val_accuracy improved from 0.85144 to 0.85366, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M.h5
507/507 [==============================] - 385s 759ms/step - loss: 0.1313 - accuracy: 0.9520 - val_loss: 0.6024
- val_accuracy: 0.8537 - lr: 1.2500e-04
Epoch 10/10
507/507 [==============================] - ETA: 0s - loss: 0.1226 - accuracy: 0.9543
Epoch 10: val_accuracy did not improve from 0.85366
507/507 [==============================] - 241s 476ms/step - loss: 0.1226 - accuracy: 0.9543 - val_loss: 0.5882
- val_accuracy: 0.8437 - lr: 1.2500e-04
```

```
model= load_model('/content/drive/MyDrive/4.2/SC Lab/Project/Saved Model/EfficientNetV2M.h5')
```

```
history = model.fit(datagen.flow(train_data, train_labels, batch_size=batch_size),
                    validation_data=(val_data, val_labels),
                    epochs=20,
                    callbacks=callback_list)
```

```
Epoch 1/20
507/507 [==============================] - ETA: 0s - loss: 0.1267 - accuracy: 0.9538
Epoch 1: val_accuracy improved from -inf to 0.83259, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M1.h5
507/507 [==============================] - 288s 504ms/step - loss: 0.1267 - accuracy: 0.9538 - val_loss: 0.7024
- val_accuracy: 0.8326 - lr: 1.2500e-04
Epoch 2/20
507/507 [==============================] - ETA: 0s - loss: 0.1220 - accuracy: 0.9567
Epoch 2: val_accuracy improved from 0.83259 to 0.85698, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M1.h5
507/507 [==============================] - 266s 525ms/step - loss: 0.1220 - accuracy: 0.9567 - val_loss: 0.6050
- val_accuracy: 0.8570 - lr: 1.2500e-04
Epoch 3/20
507/507 [==============================] - ETA: 0s - loss: 0.1262 - accuracy: 0.9567
Epoch 3: val_accuracy did not improve from 0.85698
507/507 [==============================] - 241s 475ms/step - loss: 0.1262 - accuracy: 0.9567 - val_loss: 0.6831
- val_accuracy: 0.8237 - lr: 1.2500e-04
Epoch 4/20
507/507 [==============================] - ETA: 0s - loss: 0.1326 - accuracy: 0.9534
Epoch 4: val_accuracy did not improve from 0.85698
507/507 [==============================] - 235s 463ms/step - loss: 0.1326 - accuracy: 0.9534 - val_loss: 0.6828
- val_accuracy: 0.8370 - lr: 1.2500e-04
Epoch 5/20
507/507 [==============================] - ETA: 0s - loss: 0.1164 - accuracy: 0.9586
Epoch 5: val_accuracy did not improve from 0.85698
507/507 [==============================] - 240s 473ms/step - loss: 0.1164 - accuracy: 0.9586 - val_loss: 0.5935
- val_accuracy: 0.8404 - lr: 1.2500e-04
Epoch 6/20
507/507 [==============================] - ETA: 0s - loss: 0.1088 - accuracy: 0.9628
Epoch 6: val_accuracy did not improve from 0.85698
507/507 [==============================] - 239s 472ms/step - loss: 0.1088 - accuracy: 0.9628 - val_loss: 0.6388
- val_accuracy: 0.8404 - lr: 1.2500e-04
Epoch 7/20
507/507 [==============================] - ETA: 0s - loss: 0.1025 - accuracy: 0.9636
Epoch 7: val_accuracy did not improve from 0.85698

Epoch 7: ReduceLROnPlateau reducing learning rate to 0.0001.
507/507 [==============================] - 240s 473ms/step - loss: 0.1025 - accuracy: 0.9636 - val_loss: 0.6753
- val_accuracy: 0.8370 - lr: 1.2500e-04
Epoch 8/20
507/507 [==============================] - ETA: 0s - loss: 0.1045 - accuracy: 0.9623
Epoch 8: val_accuracy did not improve from 0.85698
507/507 [==============================] - 235s 463ms/step - loss: 0.1045 - accuracy: 0.9623 - val_loss: 0.6443
- val_accuracy: 0.8492 - lr: 1.0000e-04
Epoch 9/20
507/507 [==============================] - ETA: 0s - loss: 0.1043 - accuracy: 0.9639
Epoch 9: val_accuracy did not improve from 0.85698
507/507 [==============================] - 239s 470ms/step - loss: 0.1043 - accuracy: 0.9639 - val_loss: 0.6061
- val_accuracy: 0.8503 - lr: 1.0000e-04
Epoch 10/20
507/507 [==============================] - ETA: 0s - loss: 0.1039 - accuracy: 0.9623
Epoch 10: val_accuracy did not improve from 0.85698
```

```
507/507 [==============================] - 235s 463ms/step - loss: 0.1039 - accuracy: 0.9623 - val_loss: 0.5961
- val_accuracy: 0.8525 - lr: 1.0000e-04
Epoch 11/20
507/507 [==============================] - ETA: 0s - loss: 0.0962 - accuracy: 0.9639
Epoch 11: val_accuracy did not improve from 0.85698
507/507 [==============================] - 240s 474ms/step - loss: 0.0962 - accuracy: 0.9639 - val_loss: 0.6111
- val_accuracy: 0.8537 - lr: 1.0000e-04
Epoch 12/20
507/507 [==============================] - ETA: 0s - loss: 0.0950 - accuracy: 0.9683
Epoch 12: val_accuracy did not improve from 0.85698
507/507 [==============================] - 235s 463ms/step - loss: 0.0950 - accuracy: 0.9683 - val_loss: 0.6259
- val_accuracy: 0.8514 - lr: 1.0000e-04
Epoch 13/20
507/507 [==============================] - ETA: 0s - loss: 0.0874 - accuracy: 0.9704
Epoch 13: val_accuracy did not improve from 0.85698
507/507 [==============================] - 239s 472ms/step - loss: 0.0874 - accuracy: 0.9704 - val_loss: 0.6628
- val_accuracy: 0.8537 - lr: 1.0000e-04
Epoch 14/20
507/507 [==============================] - ETA: 0s - loss: 0.0859 - accuracy: 0.9704
Epoch 14: val_accuracy did not improve from 0.85698
507/507 [==============================] - 239s 472ms/step - loss: 0.0859 - accuracy: 0.9704 - val_loss: 0.6706
- val_accuracy: 0.8470 - lr: 1.0000e-04
Epoch 15/20
507/507 [==============================] - ETA: 0s - loss: 0.0829 - accuracy: 0.9708
Epoch 15: val_accuracy did not improve from 0.85698
507/507 [==============================] - 235s 463ms/step - loss: 0.0829 - accuracy: 0.9708 - val_loss: 0.6956
- val_accuracy: 0.8481 - lr: 1.0000e-04
Epoch 16/20
507/507 [==============================] - ETA: 0s - loss: 0.0838 - accuracy: 0.9715
Epoch 16: val_accuracy did not improve from 0.85698
507/507 [==============================] - 239s 472ms/step - loss: 0.0838 - accuracy: 0.9715 - val_loss: 0.6653
- val_accuracy: 0.8481 - lr: 1.0000e-04
Epoch 17/20
507/507 [==============================] - ETA: 0s - loss: 0.0858 - accuracy: 0.9692
Epoch 17: val_accuracy improved from 0.85698 to 0.86475, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M1.h5
507/507 [==============================] - 296s 584ms/step - loss: 0.0858 - accuracy: 0.9692 - val_loss: 0.6045
- val_accuracy: 0.8647 - lr: 1.0000e-04
Epoch 18/20
507/507 [==============================] - ETA: 0s - loss: 0.0818 - accuracy: 0.9715
Epoch 18: val_accuracy improved from 0.86475 to 0.86696, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/EfficientNetV2M1.h5
507/507 [==============================] - 374s 738ms/step - loss: 0.0818 - accuracy: 0.9715 - val_loss: 0.6174
- val_accuracy: 0.8670 - lr: 1.0000e-04
Epoch 19/20
507/507 [==============================] - ETA: 0s - loss: 0.0894 - accuracy: 0.9672
Epoch 19: val_accuracy did not improve from 0.86696
507/507 [==============================] - 241s 474ms/step - loss: 0.0894 - accuracy: 0.9672 - val_loss: 0.6234
- val_accuracy: 0.8492 - lr: 1.0000e-04
Epoch 20/20
507/507 [==============================] - ETA: 0s - loss: 0.0816 - accuracy: 0.9720
Epoch 20: val_accuracy did not improve from 0.86696
```

```
507/507 [==============================] - 239s 472ms/step - loss: 0.0816 - accuracy: 0.9720 - val_loss: 0.7090
- val_accuracy: 0.8537 - lr: 1.0000e-04
```

```
model= load_model('/content/drive/MyDrive/SC Lab/Saved Model/EfficientNetV2M.h5')
```

Test Accuracy

```
test_loss, test_accuracy = model.evaluate(test_data, test_labels)
print("Test Accuracy:", test_accuracy)
```

```
32/32 [==============================] - 13s 230ms/step - loss: 0.6494 - accuracy: 0.8503
Test Accuracy: 0.8502994179725647
```

Classification Report

```
# Make predictions on the test data
predictions = model.predict(test_data)

# Convert predictions and true labels to integer format
predicted_labels = np.argmax(predictions, axis=1)
true_labels = np.argmax(test_labels, axis=1)

# Calculate the classification report
report = classification_report(true_labels, predicted_labels)

# Calculate the confusion matrix
conf_matrix = confusion_matrix(true_labels, predicted_labels)

# Print the classification report
print("Classification Report:")
print(report)
```

```
32/32 [==============================] - 21s 206ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.42      0.58        33
           1       0.76      0.76      0.76        51
           2       0.67      0.73      0.70       110
           3       0.56      0.42      0.48        12
           4       0.70      0.56      0.62       111
           5       0.91      0.95      0.93       671
           6       0.93      0.93      0.93        14

    accuracy                           0.85      1002
   macro avg       0.78      0.68      0.71      1002
weighted avg       0.85      0.85      0.84      1002
```

Confusion Matrix

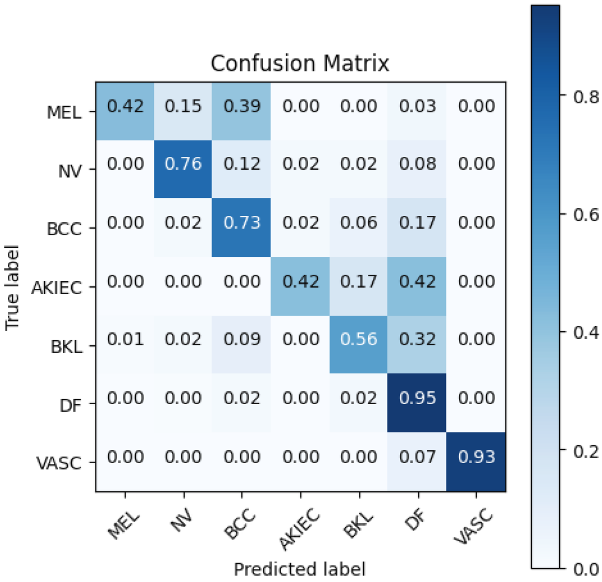
```
# Calculate the confusion matrix
cm = confusion_matrix(true_labels, np.round(predicted_labels))
```

```
cm
```

```
array([[ 14,   5,  13,   0,   0,   1,   0],
       [  0,  39,   6,   1,   1,   4,   0],
       [  0,   2,  80,   2,   7,  19,   0],
       [  0,   0,   0,   5,   2,   5,   0],
       [  1,   2,  10,   0,  62,  36,   0],
       [  0,   3,  11,   1,  16, 639,   1],
       [  0,   0,   0,   0,   0,   1,  13]])
```

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    # print(cm)

    plt.figure(figsize=(5, 5))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()


cm_plot_labels = ["MEL", "NV", "BCC", "AKIEC", "BKL", "DF", "VASC"]

plot_confusion_matrix(cm, cm_plot_labels, title='Confusion Matrix', normalize=True)
```

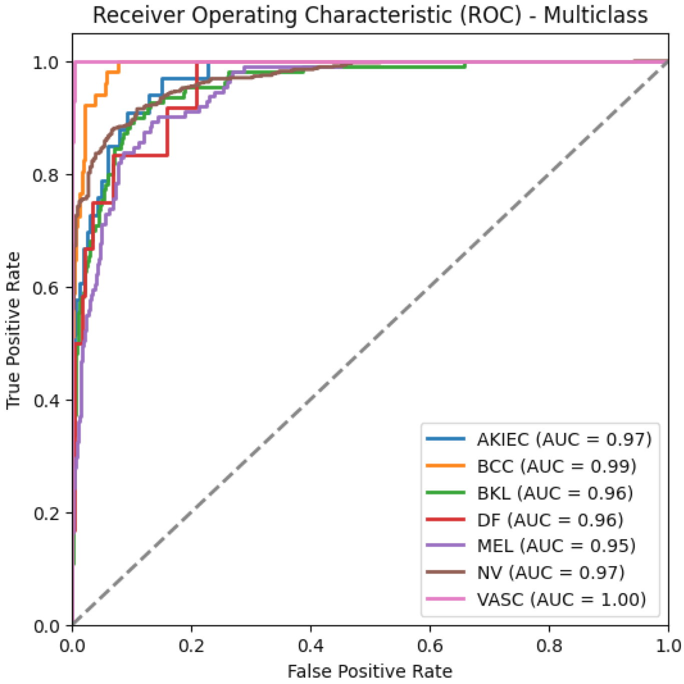
```
Normalized confusion matrix
```



ROC-AUC curve

```
# Define class names
class_names = ['AKIEC', 'BCC', 'BKL', 'DF', 'MEL', 'NV', 'VASC']

# Make predictions on the test data
predictions = model.predict(test_data)

# Get the number of classes
num_classes = test_labels.shape[1]

# Initialize a figure to plot ROC curves
plt.figure(figsize=(6, 6))

# Loop through each class
for class_index in range(num_classes):
    # Compute ROC curve and ROC AUC for the current class
    fpr, tpr, thresholds = roc_curve(test_labels[:, class_index], predictions[:, class_index])
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve for the current class
    plt.plot(fpr, tpr, lw=2, label=f'{class_names[class_index]} (AUC = {roc_auc:.2f})')

# Plot the diagonal line (random chance)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)

# Set plot properties
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) - Multiclass')
plt.legend(loc='lower right')

# Display the plot
plt.show()
```

```
32/32 [==============================] - 6s 179ms/step
```



Misclass Classification

```
!pip install tf-keras-vis
```

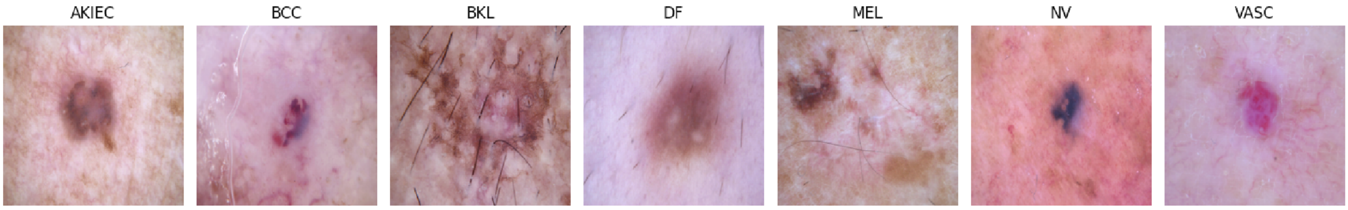
```
Collecting tf-keras-vis
  Downloading tf_keras_vis-0.8.5-py3-none-any.whl (52 kB)
 [2K     [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ [0m [32m52.1/52.1 kB [0m [31m1.9 MB/s [0m eta
 [36m0:00:00 [0m
 [?25hRequirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis)
(1.10.1)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis) (9.4.0)
Collecting deprecated (from tf-keras-vis)
  Downloading Deprecated-1.2.14-py2.py3-none-any.whl (9.6 kB)
Requirement already satisfied: imageio in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis) (2.31.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis) (23.1)
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.10/dist-packages (from deprecated->tf-
keras-vis) (1.14.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from imageio->tf-keras-vis)
(1.23.5)
Installing collected packages: deprecated, tf-keras-vis
Successfully installed deprecated-1.2.14 tf-keras-vis-0.8.5
```

```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
from tensorflow.python.client import device_lib
device_list = device_lib.list_local_devices()
gpus = [device.name for device in device_list if device.device_type == 'GPU']
print('TensorFlow recognized {} GPUs'.format(len(gpus)))
```

```
TensorFlow recognized 1 GPUs
```

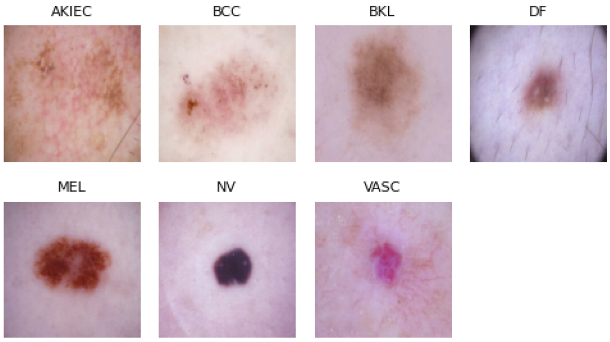
```
image_titles = ['AKIEC', 'BCC', 'BKL', 'DF', 'MEL', 'NV', 'VASC']
num_images = len(image_titles)
```

```python
# Convert one-hot encoded labels to integer labels
# test_labels_int = np.argmax(test_labels, axis=1)
# # Find the indices of the first image from each class
# class_indices = [np.where(test_labels_int == i)[0][0] for i in range(len(image_titles))]

# # Create an array to store the images
# image_array = []

# # Create subplots with 2 rows
# num_rows = 2
# num_cols = (num_images + 1) // num_rows
# fig, ax = plt.subplots(num_rows, num_cols, figsize=(5, 3))

# for i, title in enumerate(image_titles):
#     row = i // num_cols
#     col = i % num_cols
#     ax[row, col].set_title(title, fontsize=8)

#     # Display the image from test data
#     img = test_data[class_indices[i]]
#     image_array.append(img)  # Store the image in the array

#     ax[row, col].imshow(img)
#     ax[row, col].axis('off')

# # Remove any empty subplots
# for i in range(len(image_titles), num_rows * num_cols):
#     row = i // num_cols
#     col = i % num_cols
#     fig.delaxes(ax[row, col])

# plt.tight_layout()
# plt.show()

# X = base_preprocess(np.array(image_array))

test_labels_int = np.argmax(test_labels, axis=1)
# Find the indices of the first image from each class
class_indices = [np.where(test_labels_int == i)[0][0] for i in range(len(image_titles))]

# Create an array to store the images
image_array = []

# Create a single row of plots
num_images = len(image_titles)
fig, ax = plt.subplots(1, num_images, figsize=(15, 3))

for i, title in enumerate(image_titles):
    ax[i].set_title(title, fontsize=12)
    ax[i].axis('off')

    # Display the image from test data
    img = test_data[class_indices[i]]
    image_array.append(img)  # Store the image in the array

    ax[i].imshow(img)

plt.tight_layout()
plt.show()

X = base_preprocess(np.array(image_array))
```



Random Image

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

image_titles = ['AKIEC', 'BCC', 'BKL', 'DF', 'MEL', 'NV', 'VASC']
num_images = len(image_titles)

# Convert one-hot encoded labels to integer labels
test_labels_int = np.argmax(test_labels, axis=1)
# Create an array to store the images
image_array = []

# Create subplots with 2 rows
num_rows = 2
num_cols = (num_images + 1) // num_rows
fig, ax = plt.subplots(num_rows, num_cols, figsize=(5, 3))

for i, title in enumerate(image_titles):
    row = i // num_cols
    col = i % num_cols
    ax[row, col].set_title(title, fontsize=8)

    # Find indices of images for the current class
    class_indices = np.where(test_labels_int == i)[0]
    random_index = np.random.choice(class_indices)  # Choose a random index

    # Display the image from test data
    img = test_data[random_index]
    image_array.append(img)  # Store the image in the array

    ax[row, col].imshow(img)
    ax[row, col].axis('off')

# Remove any empty subplots
for i in range(len(image_titles), num_rows * num_cols):
    row = i // num_cols
    col = i % num_cols
    fig.delaxes(ax[row, col])

plt.tight_layout()
plt.show()

X = base_preprocess(np.array(image_array))
```




```
from tf_keras_vis.utils.model_modifiers import ReplaceToLinear

replace2linear = ReplaceToLinear()

# Instead of using the ReplaceToLinear instance above,
# you can also define the function from scratch as follows:
def model_modifier_function(cloned_model):
    cloned_model.layers[-1].activation = tf.keras.activations.linear
```

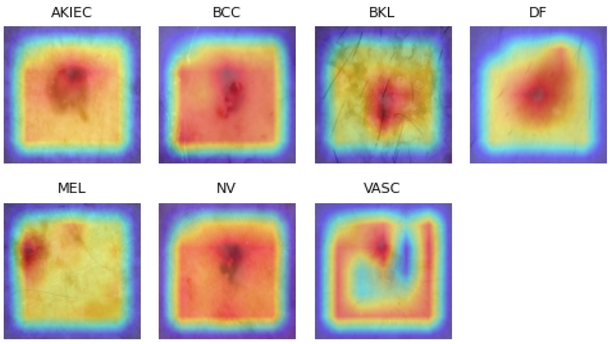
```
from tf_keras_vis.utils.scores import CategoricalScore

# 1 is the imagenet index corresponding to Goldfish, 294 to Bear and 413 to Assault Rifle.
score = CategoricalScore([0, 1, 2, 3, 4, 5, 6])

# Instead of using CategoricalScore object,
# you can also define the function from scratch as follows:
def score_function(output):
    # The `output` variable refers to the output of the model,
    # so, in this case, `output` shape is `(3, 1000)` i.e., (samples, classes).
    return (output[0][1][2][3][4][5][6])
```

Faster Scorecam

```
%%time
from matplotlib import cm
from tf_keras_vis.scorecam import Scorecam

# Create ScoreCAM object
scorecam = Scorecam(model, model_modifier=replace2linear)

# Generate heatmap with Faster-ScoreCAM
cam = scorecam(score,
               X,
               penultimate_layer=-1,
               max_N=10)

## Since v0.6.0, calling `normalize()` is NOT necessary.
# cam = normalize(cam)

# Calculate the number of rows and columns for subplots
num_rows = 2
num_cols = (num_images + 1) // num_rows

# Render
f, ax = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(5, 3))

for i, title in enumerate(image_titles):
    row = i // num_cols
    col = i % num_cols
```

```
    heatmap = np.uint8(cm.jet(cam[i])[..., :3] * 255)
    ax[row, col].set_title(title, fontsize=8)
    ax[row, col].imshow(image_array[i])
    ax[row, col].imshow(heatmap, cmap='jet', alpha=0.5)
    ax[row, col].axis('off')

# Remove any empty subplots
for i in range(len(image_titles), num_rows * num_cols):
    row = i // num_cols
    col = i % num_cols
    f.delaxes(ax[row, col])

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

```
4/4 [==============================] - 12s 750ms/step
```

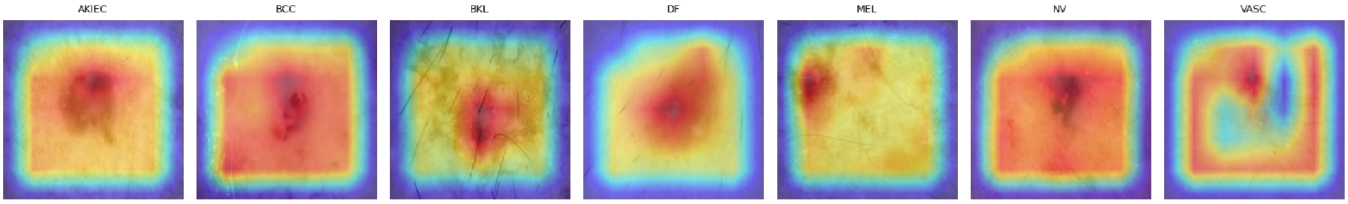



```
CPU times: user 19.7 s, sys: 434 ms, total: 20.1 s
Wall time: 30.5 s
```

```
%%time
from matplotlib import pyplot as plt, cm
from tf_keras_vis.scorecam import Scorecam

# Assuming you have already defined model and replace2linear

# Create ScoreCAM object
scorecam = Scorecam(model, model_modifier=replace2linear)

# Generate heatmaps with Faster-ScoreCAM
cam = scorecam(score, X, penultimate_layer=-1, max_N=10)

# Calculate the number of images
num_images = len(image_titles)

# Create a single row plot
fig, axes = plt.subplots(1, num_images, figsize=(15, 5))

for i, title in enumerate(image_titles):
    heatmap = np.uint8(cm.jet(cam[i])[..., :3] * 255)

    combined_image = cv2.addWeighted(image_array[i], 0.5, heatmap, 0.5, 0)

    axes[i].set_title(title, fontsize=8)
    axes[i].imshow(combined_image)
    axes[i].axis('off')
```

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

```
4/4 [==============================] - 13s 610ms/step
```

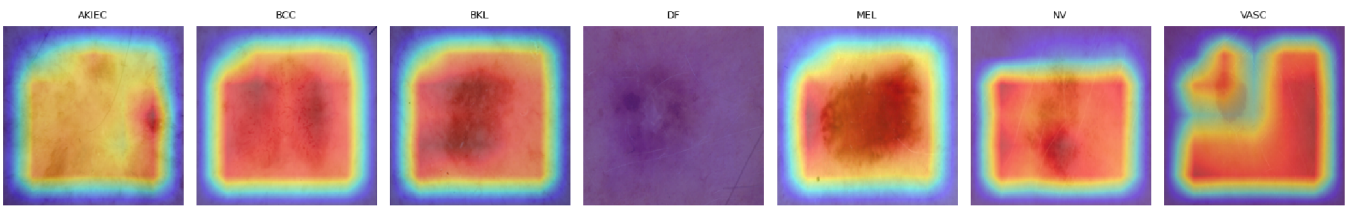



```
CPU times: user 20 s, sys: 924 ms, total: 20.9 s
Wall time: 31.5 s
```

Faster Score-CAM for Random Image

```
%%time
from matplotlib import pyplot as plt, cm
from tf_keras_vis.scorecam import Scorecam

# Assuming you have already defined model and replace2linear

# Create ScoreCAM object
scorecam = Scorecam(model, model_modifier=replace2linear)

# Generate heatmaps with Faster-ScoreCAM
cam = scorecam(score, X, penultimate_layer=-1, max_N=10)

# Calculate the number of images
num_images = len(image_titles)

# Create a single row plot
fig, axes = plt.subplots(1, num_images, figsize=(15, 5))

for i, title in enumerate(image_titles):
    heatmap = np.uint8(cm.jet(cam[i])[..., :3] * 255)

    combined_image = cv2.addWeighted(image_array[i], 0.5, heatmap, 0.5, 0)

    axes[i].set_title(title, fontsize=8)
    axes[i].imshow(combined_image)
    axes[i].axis('off')

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

```
4/4 [==============================] - 6s 193ms/step
```



```
CPU times: user 14.7 s, sys: 591 ms, total: 15.3 s
Wall time: 15.8 s
```