XceptionNet

```
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
import itertools
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.xception 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
```

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')
```

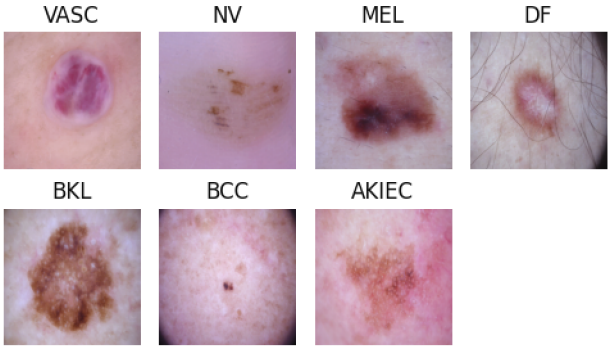
```
print("Data shape:", data.shape)
print("Labels shape:", labels.shape)
```

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

Loading Images from the data

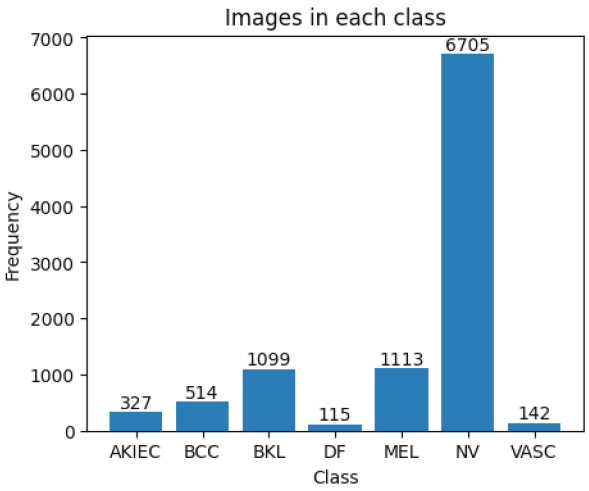
```
# Get unique class labels and their corresponding indices in the data array
unique_classes = np.unique(labels, axis=1)

# 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=(5, 4))
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)
```

Data Frequency of Train, Test, and Validation

```
print("train_data shape:", train_data.shape)
print("train_labels shape:", train_labels.shape)
print("val_data shape:", val_data.shape)
print("val_labels shape:", val_labels.shape)
print("test_data shape:", test_data.shape)
print("test_labels shape:", test_labels.shape)
```

```
train_data shape: (8111, 224, 224, 3)
train_labels shape: (8111, 7)
val_data shape: (902, 224, 224, 3)
val_labels shape: (902, 7)
test_data shape: (1002, 224, 224, 3)
test_labels shape: (1002, 7)
```

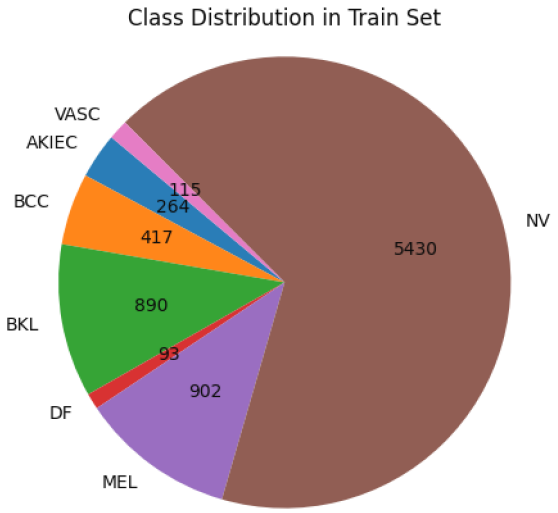
Data Frequency of Each Class

```
class_names_mapping = {
    0: "AKIEC",
    1: "BCC",
    2: "BKL",
    3: "DF",
    4: "MEL",
    5: "NV",
    6: "VASC"
}
# Calculate class distribution in each set
num_classes = train_labels.shape[1]
class_counts_train = np.sum(train_labels, axis=0)
class_counts_val = np.sum(val_labels, axis=0)
class_counts_test = np.sum(test_labels, axis=0)

class_counts_mapping = {}
for index, class_name in class_names_mapping.items():
```

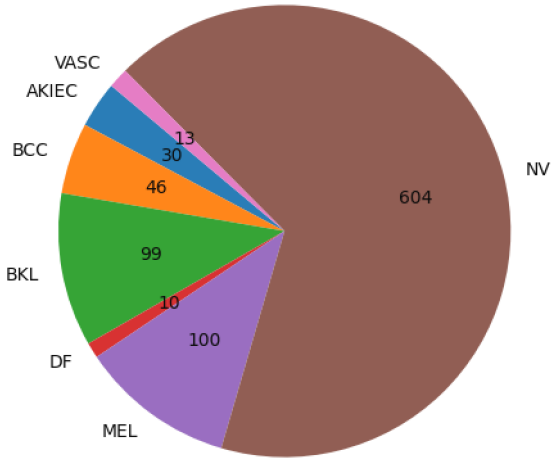
```
    class_counts_mapping[class_name] = {'Train': class_counts_train[index]}

for index, class_name in class_names_mapping.items():
    class_counts_mapping[class_name]['Validation'] = class_counts_val[index]

for index, class_name in class_names_mapping.items():
    class_counts_mapping[class_name]['Test'] = class_counts_test[index]

# Print class distribution mapping
for class_name, counts in class_counts_mapping.items():
    print(class_name)
    for set_name, count in counts.items():
        print(f" - {set_name}: {count}")
```

```
AKIEC
 - Train: 264.0
 - Validation: 30.0
 - Test: 33.0
BCC
 - Train: 417.0
 - Validation: 46.0
 - Test: 51.0
BKL
 - Train: 890.0
 - Validation: 99.0
 - Test: 110.0
DF
 - Train: 93.0
 - Validation: 10.0
 - Test: 12.0
MEL
 - Train: 902.0
 - Validation: 100.0
 - Test: 111.0
NV
 - Train: 5430.0
 - Validation: 604.0
 - Test: 671.0
VASC
 - Train: 115.0
 - Validation: 13.0
 - Test: 14.0
```

```
# Create pie charts for each set
for set_name in ['Train']:
    class_counts = [counts[set_name] for counts in class_counts_mapping.values()]
    class_labels = list(class_counts_mapping.keys())

    plt.figure(figsize=(5, 5))
    plt.pie(class_counts, labels=class_labels, startangle=140, autopct=lambda p: '{:.0f}'.format(p *
sum(class_counts) / 100))
    plt.title(f'Class Distribution in {set_name} Set')
    plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
    plt.show()
```




```
# Create pie charts for each set
for set_name in ['Validation']:
    class_counts = [counts[set_name] for counts in class_counts_mapping.values()]
    class_labels = list(class_counts_mapping.keys())

    plt.figure(figsize=(5, 5))
    plt.pie(class_counts, labels=class_labels, startangle=140, autopct=lambda p: '{:.0f}'.format(p *
sum(class_counts) / 100))
    plt.title(f'Class Distribution in {set_name} Set')
    plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
    plt.show()
```




```
# Create pie charts for each set
for set_name in ['Test']:
    class_counts = [counts[set_name] for counts in class_counts_mapping.values()]
    class_labels = list(class_counts_mapping.keys())

    plt.figure(figsize=(5, 5))
    plt.pie(class_counts, labels=class_labels, startangle=140, autopct=lambda p: '{:.0f}'.format(p *
sum(class_counts) / 100))
    plt.title(f'Class Distribution in {set_name} Set')
    plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
    plt.show()
```

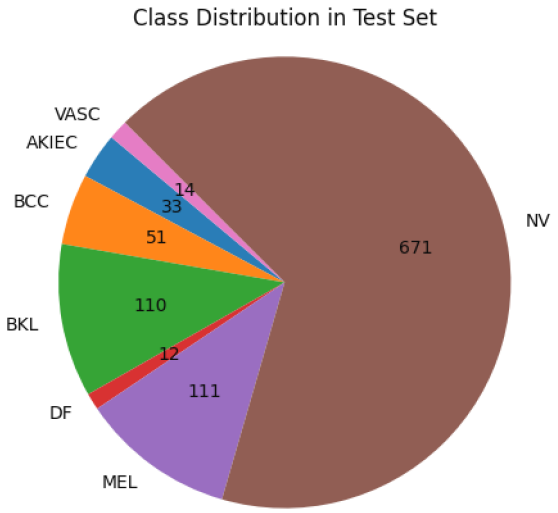


Image Augmentation

```
# Create an ImageDataGenerator for data augmentation during training
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
)
```

Callback Functions

```
lr_reduce = ReduceLROnPlateau(monitor = 'val_accuracy', factor = 0.5, patience = 5,mode='max', min_lr = 1e-
4,verbose = 1)
#early_stop = EarlyStopping(monitor = "val_loss", patience = 5, verbose=1)
saved_model = '/content/drive/MyDrive/SC Lab/Saved Model/Xception.h5'
model_chkpt = ModelCheckpoint(saved_model ,save_best_only = True, monitor = 'val_accuracy',verbose = 1)

# callback_list = [early_stop, model_chkpt, lr_reduce]
callback_list = [model_chkpt, lr_reduce]
```

XceptionNet Model

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83683744/83683744 [==============================] - 1s 0us/step
```

Model's Layers

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

# Create the final model with custom classification layers
```

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


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

Model Training

```
epochs = 30
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 1/30
507/507 [==============================] - ETA: 0s - loss: 0.9278 - accuracy: 0.6857
Epoch 1: val_accuracy improved from -inf to 0.71064, saving model to /content/drive/MyDrive/SC Lab/Saved
Model/Xception.h5
507/507 [==============================] - 181s 319ms/step - loss: 0.9278 - accuracy: 0.6857 - val_loss: 1.6198
- val_accuracy: 0.7106 - lr: 0.0010
Epoch 2/30
507/507 [==============================] - ETA: 0s - loss: 0.7764 - accuracy: 0.7173
Epoch 2: val_accuracy improved from 0.71064 to 0.72284, saving model to /content/drive/MyDrive/SC Lab/Saved
Model/Xception.h5
507/507 [==============================] - 155s 306ms/step - loss: 0.7764 - accuracy: 0.7173 - val_loss: 0.8816
- val_accuracy: 0.7228 - lr: 0.0010
Epoch 3/30
507/507 [==============================] - ETA: 0s - loss: 0.7214 - accuracy: 0.7448
Epoch 3: val_accuracy improved from 0.72284 to 0.76829, saving model to /content/drive/MyDrive/SC Lab/Saved
Model/Xception.h5
507/507 [==============================] - 164s 323ms/step - loss: 0.7214 - accuracy: 0.7448 - val_loss: 0.7394
- val_accuracy: 0.7683 - lr: 0.0010
Epoch 4/30
507/507 [==============================] - ETA: 0s - loss: 0.6765 - accuracy: 0.7608
Epoch 4: val_accuracy improved from 0.76829 to 0.76940, saving model to /content/drive/MyDrive/SC Lab/Saved
Model/Xception.h5
507/507 [==============================] - 162s 319ms/step - loss: 0.6765 - accuracy: 0.7608 - val_loss: 0.6247
- val_accuracy: 0.7694 - lr: 0.0010
Epoch 5/30
507/507 [==============================] - ETA: 0s - loss: 0.6561 - accuracy: 0.7638
Epoch 5: val_accuracy improved from 0.76940 to 0.77384, saving model to /content/drive/MyDrive/SC Lab/Saved
Model/Xception.h5
507/507 [==============================] - 166s 326ms/step - loss: 0.6561 - accuracy: 0.7638 - val_loss: 0.6217
- val_accuracy: 0.7738 - lr: 0.0010
Epoch 6/30
507/507 [==============================] - ETA: 0s - loss: 0.6145 - accuracy: 0.7851
Epoch 6: val_accuracy did not improve from 0.77384
507/507 [==============================] - 153s 301ms/step - loss: 0.6145 - accuracy: 0.7851 - val_loss: 0.7200
- val_accuracy: 0.7461 - lr: 0.0010
```

```
Epoch 7/30
507/507 [==============================] - ETA: 0s - loss: 0.5912 - accuracy: 0.7858
Epoch 7: val_accuracy improved from 0.77384 to 0.82705, saving model to /content/drive/MyDrive/SC Lab/Saved Model/Xception.h5
507/507 [==============================] - 167s 329ms/step - loss: 0.5912 - accuracy: 0.7858 - val_loss: 0.5555 - val_accuracy: 0.8271 - lr: 0.0010
Epoch 8/30
507/507 [==============================] - ETA: 0s - loss: 0.5702 - accuracy: 0.7968
Epoch 8: val_accuracy did not improve from 0.82705
507/507 [==============================] - 153s 302ms/step - loss: 0.5702 - accuracy: 0.7968 - val_loss: 0.7315 - val_accuracy: 0.7927 - lr: 0.0010
Epoch 9/30
507/507 [==============================] - ETA: 0s - loss: 0.5400 - accuracy: 0.8104
Epoch 9: val_accuracy did not improve from 0.82705
507/507 [==============================] - 151s 298ms/step - loss: 0.5400 - accuracy: 0.8104 - val_loss: 0.9652 - val_accuracy: 0.7672 - lr: 0.0010
Epoch 10/30
507/507 [==============================] - ETA: 0s - loss: 0.5184 - accuracy: 0.8205
Epoch 10: val_accuracy did not improve from 0.82705
507/507 [==============================] - 151s 297ms/step - loss: 0.5184 - accuracy: 0.8205 - val_loss: 0.5751 - val_accuracy: 0.8248 - lr: 0.0010
Epoch 11/30
507/507 [==============================] - ETA: 0s - loss: 0.4979 - accuracy: 0.8246
Epoch 11: val_accuracy did not improve from 0.82705
507/507 [==============================] - 156s 307ms/step - loss: 0.4979 - accuracy: 0.8246 - val_loss: 0.5865 - val_accuracy: 0.7960 - lr: 0.0010
Epoch 12/30
507/507 [==============================] - ETA: 0s - loss: 0.4893 - accuracy: 0.8317
Epoch 12: val_accuracy improved from 0.82705 to 0.83481, saving model to /content/drive/MyDrive/SC Lab/Saved Model/Xception.h5
507/507 [==============================] - 171s 338ms/step - loss: 0.4893 - accuracy: 0.8317 - val_loss: 0.5261 - val_accuracy: 0.8348 - lr: 0.0010
Epoch 13/30
507/507 [==============================] - ETA: 0s - loss: 0.4535 - accuracy: 0.8419
Epoch 13: val_accuracy did not improve from 0.83481
507/507 [==============================] - 154s 304ms/step - loss: 0.4535 - accuracy: 0.8419 - val_loss: 0.6706 - val_accuracy: 0.7927 - lr: 0.0010
Epoch 14/30
507/507 [==============================] - ETA: 0s - loss: 0.4452 - accuracy: 0.8413
Epoch 14: val_accuracy did not improve from 0.83481
507/507 [==============================] - 155s 305ms/step - loss: 0.4452 - accuracy: 0.8413 - val_loss: 0.5329 - val_accuracy: 0.8248 - lr: 0.0010
Epoch 15/30
507/507 [==============================] - ETA: 0s - loss: 0.4206 - accuracy: 0.8545
Epoch 15: val_accuracy improved from 0.83481 to 0.85366, saving model to /content/drive/MyDrive/SC Lab/Saved Model/Xception.h5
507/507 [==============================] - 178s 350ms/step - loss: 0.4206 - accuracy: 0.8545 - val_loss: 0.5925 - val_accuracy: 0.8537 - lr: 0.0010
Epoch 16/30
507/507 [==============================] - ETA: 0s - loss: 0.3933 - accuracy: 0.8607
Epoch 16: val_accuracy did not improve from 0.85366
507/507 [==============================] - 154s 303ms/step - loss: 0.3933 - accuracy: 0.8607 - val_loss: 0.5367
```

```
- val_accuracy: 0.8149 - lr: 0.0010
Epoch 17/30
507/507 [==============================] - ETA: 0s - loss: 0.4042 - accuracy: 0.8628
Epoch 17: val_accuracy did not improve from 0.85366
507/507 [==============================] - 153s 302ms/step - loss: 0.4042 - accuracy: 0.8628 - val_loss: 0.4775
- val_accuracy: 0.8359 - lr: 0.0010
Epoch 18/30
507/507 [==============================] - ETA: 0s - loss: 0.3847 - accuracy: 0.8718
Epoch 18: val_accuracy did not improve from 0.85366
507/507 [==============================] - 153s 302ms/step - loss: 0.3847 - accuracy: 0.8718 - val_loss: 0.4919
- val_accuracy: 0.8315 - lr: 0.0010
Epoch 19/30
507/507 [==============================] - ETA: 0s - loss: 0.3515 - accuracy: 0.8755
Epoch 19: val_accuracy did not improve from 0.85366
507/507 [==============================] - 152s 300ms/step - loss: 0.3515 - accuracy: 0.8755 - val_loss: 0.6243
- val_accuracy: 0.8060 - lr: 0.0010
Epoch 20/30
507/507 [==============================] - ETA: 0s - loss: 0.3386 - accuracy: 0.8851
Epoch 20: val_accuracy did not improve from 0.85366

Epoch 20: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
507/507 [==============================] - 154s 304ms/step - loss: 0.3386 - accuracy: 0.8851 - val_loss: 0.4621
- val_accuracy: 0.8415 - lr: 0.0010
Epoch 21/30
507/507 [==============================] - ETA: 0s - loss: 0.2586 - accuracy: 0.9122
Epoch 21: val_accuracy improved from 0.85366 to 0.87140, saving model to /content/drive/MyDrive/SC Lab/Saved
Model/Xception.h5
507/507 [==============================] - 178s 350ms/step - loss: 0.2586 - accuracy: 0.9122 - val_loss: 0.4036
- val_accuracy: 0.8714 - lr: 5.0000e-04
Epoch 22/30
507/507 [==============================] - ETA: 0s - loss: 0.2200 - accuracy: 0.9216
Epoch 22: val_accuracy did not improve from 0.87140
507/507 [==============================] - 150s 296ms/step - loss: 0.2200 - accuracy: 0.9216 - val_loss: 0.4287
- val_accuracy: 0.8692 - lr: 5.0000e-04
Epoch 23/30
507/507 [==============================] - ETA: 0s - loss: 0.2094 - accuracy: 0.9285
Epoch 23: val_accuracy did not improve from 0.87140
507/507 [==============================] - 151s 297ms/step - loss: 0.2094 - accuracy: 0.9285 - val_loss: 0.4626
- val_accuracy: 0.8692 - lr: 5.0000e-04
Epoch 24/30
507/507 [==============================] - ETA: 0s - loss: 0.1965 - accuracy: 0.9324
Epoch 24: val_accuracy improved from 0.87140 to 0.88470, saving model to /content/drive/MyDrive/SC Lab/Saved
Model/Xception.h5
507/507 [==============================] - 179s 353ms/step - loss: 0.1965 - accuracy: 0.9324 - val_loss: 0.4583
- val_accuracy: 0.8847 - lr: 5.0000e-04
Epoch 25/30
507/507 [==============================] - ETA: 0s - loss: 0.1860 - accuracy: 0.9368
Epoch 25: val_accuracy did not improve from 0.88470
507/507 [==============================] - 152s 300ms/step - loss: 0.1860 - accuracy: 0.9368 - val_loss: 0.3735
- val_accuracy: 0.8803 - lr: 5.0000e-04
Epoch 26/30
507/507 [==============================] - ETA: 0s - loss: 0.1816 - accuracy: 0.9372
```

```
Epoch 26: val_accuracy did not improve from 0.88470
507/507 [==============================] - 151s 298ms/step - loss: 0.1816 - accuracy: 0.9372 - val_loss: 0.5343
- val_accuracy: 0.8769 - lr: 5.0000e-04
Epoch 27/30
507/507 [==============================] - ETA: 0s - loss: 0.1615 - accuracy: 0.9430
Epoch 27: val_accuracy did not improve from 0.88470
507/507 [==============================] - 152s 299ms/step - loss: 0.1615 - accuracy: 0.9430 - val_loss: 0.7328
- val_accuracy: 0.8271 - lr: 5.0000e-04
Epoch 28/30
507/507 [==============================] - ETA: 0s - loss: 0.1435 - accuracy: 0.9493
Epoch 28: val_accuracy did not improve from 0.88470
507/507 [==============================] - 150s 296ms/step - loss: 0.1435 - accuracy: 0.9493 - val_loss: 0.7470
- val_accuracy: 0.8537 - lr: 5.0000e-04
Epoch 29/30
507/507 [==============================] - ETA: 0s - loss: 0.1636 - accuracy: 0.9467
Epoch 29: val_accuracy did not improve from 0.88470

Epoch 29: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
507/507 [==============================] - 152s 300ms/step - loss: 0.1636 - accuracy: 0.9467 - val_loss: 0.4584
- val_accuracy: 0.8792 - lr: 5.0000e-04
Epoch 30/30
507/507 [==============================] - ETA: 0s - loss: 0.1154 - accuracy: 0.9613
Epoch 30: val_accuracy did not improve from 0.88470
507/507 [==============================] - 151s 297ms/step - loss: 0.1154 - accuracy: 0.9613 - val_loss: 0.5222
- val_accuracy: 0.8725 - lr: 2.5000e-04
```

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

```
epochs = 10
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 1/10
507/507 [==============================] - ETA: 0s - loss: 0.1860 - accuracy: 0.9364
Epoch 1: val_accuracy improved from -inf to 0.86585, saving model to /content/drive/MyDrive/SC Lab/Saved
Model/Xception.h5
507/507 [==============================] - 199s 359ms/step - loss: 0.1860 - accuracy: 0.9364 - val_loss: 0.4807
- val_accuracy: 0.8659 - lr: 5.0000e-04
Epoch 2/10
507/507 [==============================] - ETA: 0s - loss: 0.1851 - accuracy: 0.9350
Epoch 2: val_accuracy improved from 0.86585 to 0.88359, saving model to /content/drive/MyDrive/SC Lab/Saved
Model/Xception.h5
507/507 [==============================] - 188s 371ms/step - loss: 0.1851 - accuracy: 0.9350 - val_loss: 0.3988
- val_accuracy: 0.8836 - lr: 5.0000e-04
Epoch 3/10
507/507 [==============================] - ETA: 0s - loss: 0.1714 - accuracy: 0.9467
Epoch 3: val_accuracy did not improve from 0.88359
```

```
507/507 [==============================] - 166s 327ms/step - loss: 0.1714 - accuracy: 0.9467 - val_loss: 0.5394
- val_accuracy: 0.8747 - lr: 5.0000e-04
Epoch 4/10
507/507 [==============================] - ETA: 0s - loss: 0.1614 - accuracy: 0.9454
Epoch 4: val_accuracy did not improve from 0.88359
507/507 [==============================] - 154s 304ms/step - loss: 0.1614 - accuracy: 0.9454 - val_loss: 0.5524
- val_accuracy: 0.8625 - lr: 5.0000e-04
Epoch 5/10
507/507 [==============================] - ETA: 0s - loss: 0.1521 - accuracy: 0.9460
Epoch 5: val_accuracy did not improve from 0.88359
507/507 [==============================] - 153s 302ms/step - loss: 0.1521 - accuracy: 0.9460 - val_loss: 0.6482
- val_accuracy: 0.8470 - lr: 5.0000e-04
Epoch 6/10
507/507 [==============================] - ETA: 0s - loss: 0.1366 - accuracy: 0.9518
Epoch 6: val_accuracy did not improve from 0.88359
507/507 [==============================] - 153s 300ms/step - loss: 0.1366 - accuracy: 0.9518 - val_loss: 0.6486
- val_accuracy: 0.8337 - lr: 5.0000e-04
Epoch 7/10
507/507 [==============================] - ETA: 0s - loss: 0.1337 - accuracy: 0.9534
Epoch 7: val_accuracy did not improve from 0.88359

Epoch 7: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
507/507 [==============================] - 152s 300ms/step - loss: 0.1337 - accuracy: 0.9534 - val_loss: 0.5364
- val_accuracy: 0.8636 - lr: 5.0000e-04
Epoch 8/10
507/507 [==============================] - ETA: 0s - loss: 0.0919 - accuracy: 0.9715
Epoch 8: val_accuracy improved from 0.88359 to 0.89911, saving model to /content/drive/MyDrive/SC Lab/Saved
Model/Xception.h5
507/507 [==============================] - 197s 390ms/step - loss: 0.0919 - accuracy: 0.9715 - val_loss: 0.4873
- val_accuracy: 0.8991 - lr: 2.5000e-04
Epoch 9/10
507/507 [==============================] - ETA: 0s - loss: 0.0838 - accuracy: 0.9726
Epoch 9: val_accuracy did not improve from 0.89911
507/507 [==============================] - 152s 300ms/step - loss: 0.0838 - accuracy: 0.9726 - val_loss: 0.4891
- val_accuracy: 0.8891 - lr: 2.5000e-04
Epoch 10/10
507/507 [==============================] - ETA: 0s - loss: 0.0703 - accuracy: 0.9763
Epoch 10: val_accuracy did not improve from 0.89911
507/507 [==============================] - 151s 297ms/step - loss: 0.0703 - accuracy: 0.9763 - val_loss: 0.6302
- val_accuracy: 0.8825 - lr: 2.5000e-04
```

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

```
epochs = 10
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 1/10
507/507 [==============================] - ETA: 0s - loss: 0.0761 - accuracy: 0.9718
Epoch 1: val_accuracy improved from -inf to 0.88692, saving model to /content/drive/MyDrive/SC Lab/Saved Model/Xception.h5
507/507 [==============================] - 184s 331ms/step - loss: 0.0761 - accuracy: 0.9718 - val_loss: 0.6214 - val_accuracy: 0.8869 - lr: 2.5000e-04
Epoch 2/10
507/507 [==============================] - ETA: 0s - loss: 0.0819 - accuracy: 0.9702
Epoch 2: val_accuracy did not improve from 0.88692
507/507 [==============================] - 157s 309ms/step - loss: 0.0819 - accuracy: 0.9702 - val_loss: 0.6516 - val_accuracy: 0.8792 - lr: 2.5000e-04
Epoch 3/10
507/507 [==============================] - ETA: 0s - loss: 0.0789 - accuracy: 0.9739
Epoch 3: val_accuracy improved from 0.88692 to 0.88914, saving model to /content/drive/MyDrive/SC Lab/Saved Model/Xception.h5
507/507 [==============================] - 178s 351ms/step - loss: 0.0789 - accuracy: 0.9739 - val_loss: 0.6034 - val_accuracy: 0.8891 - lr: 2.5000e-04
Epoch 4/10
507/507 [==============================] - ETA: 0s - loss: 0.0693 - accuracy: 0.9760
Epoch 4: val_accuracy did not improve from 0.88914
507/507 [==============================] - 154s 304ms/step - loss: 0.0693 - accuracy: 0.9760 - val_loss: 0.6777 - val_accuracy: 0.8780 - lr: 2.5000e-04
Epoch 5/10
507/507 [==============================] - ETA: 0s - loss: 0.0646 - accuracy: 0.9787
Epoch 5: val_accuracy improved from 0.88914 to 0.89911, saving model to /content/drive/MyDrive/SC Lab/Saved Model/Xception.h5
507/507 [==============================] - 184s 363ms/step - loss: 0.0646 - accuracy: 0.9787 - val_loss: 0.5612 - val_accuracy: 0.8991 - lr: 2.5000e-04
Epoch 6/10
507/507 [==============================] - ETA: 0s - loss: 0.0703 - accuracy: 0.9795
Epoch 6: val_accuracy did not improve from 0.89911
507/507 [==============================] - 154s 304ms/step - loss: 0.0703 - accuracy: 0.9795 - val_loss: 0.7061 - val_accuracy: 0.8825 - lr: 2.5000e-04
Epoch 7/10
507/507 [==============================] - ETA: 0s - loss: 0.0612 - accuracy: 0.9790
Epoch 7: val_accuracy did not improve from 0.89911
507/507 [==============================] - 154s 304ms/step - loss: 0.0612 - accuracy: 0.9790 - val_loss: 0.5570 - val_accuracy: 0.8880 - lr: 2.5000e-04
Epoch 8/10
507/507 [==============================] - ETA: 0s - loss: 0.0599 - accuracy: 0.9816
Epoch 8: val_accuracy did not improve from 0.89911
507/507 [==============================] - 154s 304ms/step - loss: 0.0599 - accuracy: 0.9816 - val_loss: 0.6796 - val_accuracy: 0.8780 - lr: 2.5000e-04
Epoch 9/10
507/507 [==============================] - ETA: 0s - loss: 0.0515 - accuracy: 0.9818
Epoch 9: val_accuracy did not improve from 0.89911
507/507 [==============================] - 155s 306ms/step - loss: 0.0515 - accuracy: 0.9818 - val_loss: 0.5940 - val_accuracy: 0.8980 - lr: 2.5000e-04
Epoch 10/10
507/507 [==============================] - ETA: 0s - loss: 0.0546 - accuracy: 0.9850
Epoch 10: val_accuracy did not improve from 0.89911
```

```
Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
507/507 [==============================] - 156s 308ms/step - loss: 0.0546 - accuracy: 0.9850 - val_loss: 0.6764
- val_accuracy: 0.8769 - lr: 2.5000e-04
```

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

```
history = model.fit(datagen.flow(train_data, train_labels, batch_size=batch_size),
                    validation_data=(val_data, val_labels),
                    epochs=5,
                    callbacks=callback_list)
```

```
Epoch 1/5
507/507 [==============================] - ETA: 0s - loss: 0.0580 - accuracy: 0.9805
Epoch 1: val_accuracy improved from -inf to 0.86918, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/Xception1.h5
507/507 [==============================] - 161s 302ms/step - loss: 0.0580 - accuracy: 0.9805 - val_loss: 0.8680
- val_accuracy: 0.8692 - lr: 2.5000e-04
Epoch 2/5
507/507 [==============================] - ETA: 0s - loss: 0.0648 - accuracy: 0.9792
Epoch 2: val_accuracy improved from 0.86918 to 0.88914, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/Xception1.h5
507/507 [==============================] - 151s 298ms/step - loss: 0.0648 - accuracy: 0.9792 - val_loss: 0.6459
- val_accuracy: 0.8891 - lr: 2.5000e-04
Epoch 3/5
507/507 [==============================] - ETA: 0s - loss: 0.0504 - accuracy: 0.9819
Epoch 3: val_accuracy improved from 0.88914 to 0.89690, saving model to /content/drive/MyDrive/4.2/SC
Lab/Project/Saved Model/Xception1.h5
507/507 [==============================] - 154s 304ms/step - loss: 0.0504 - accuracy: 0.9819 - val_loss: 0.6430
- val_accuracy: 0.8969 - lr: 2.5000e-04
Epoch 4/5
507/507 [==============================] - ETA: 0s - loss: 0.0594 - accuracy: 0.9814
Epoch 4: val_accuracy did not improve from 0.89690
507/507 [==============================] - 147s 289ms/step - loss: 0.0594 - accuracy: 0.9814 - val_loss: 0.6440
- val_accuracy: 0.8958 - lr: 2.5000e-04
Epoch 5/5
507/507 [==============================] - ETA: 0s - loss: 0.0608 - accuracy: 0.9800
Epoch 5: val_accuracy did not improve from 0.89690
507/507 [==============================] - 149s 294ms/step - loss: 0.0608 - accuracy: 0.9800 - val_loss: 0.6279
- val_accuracy: 0.8947 - lr: 2.5000e-04
```

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

Test Accuracy

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

```
32/32 [==============================] - 15s 147ms/step - loss: 0.6125 - accuracy: 0.8872
Test Accuracy: 0.8872255682945251
```

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)
# test_labels = test_data.classes
true_labels = np.argmax(test_labels, axis=1)

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

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

```
32/32 [==============================] - 14s 134ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.58      0.70        33
           1       0.91      0.82      0.87        51
           2       0.72      0.87      0.79       110
           3       0.70      0.58      0.64        12
           4       0.72      0.68      0.70       111
           5       0.94      0.95      0.95       671
           6       1.00      0.93      0.96        14

    accuracy                           0.89      1002
   macro avg       0.84      0.77      0.80      1002
weighted avg       0.89      0.89      0.89      1002
```

Confusion Matrix

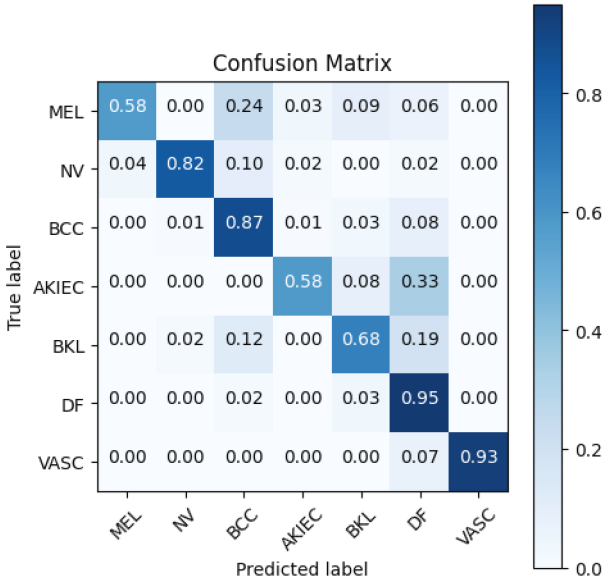
```
# Calculate the confusion matrix
cm = confusion_matrix(true_labels, np.round(predicted_labels))

cm
```

```
array([[ 19,   0,   8,   1,   3,   2,   0],
       [  2,  42,   5,   1,   0,   1,   0],
       [  0,   1,  96,   1,   3,   9,   0],
       [  0,   0,   0,   7,   1,   4,   0],
       [  0,   2,  13,   0,  75,  21,   0],
       [  0,   1,  11,   0,  22, 637,   0],
       [  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
```

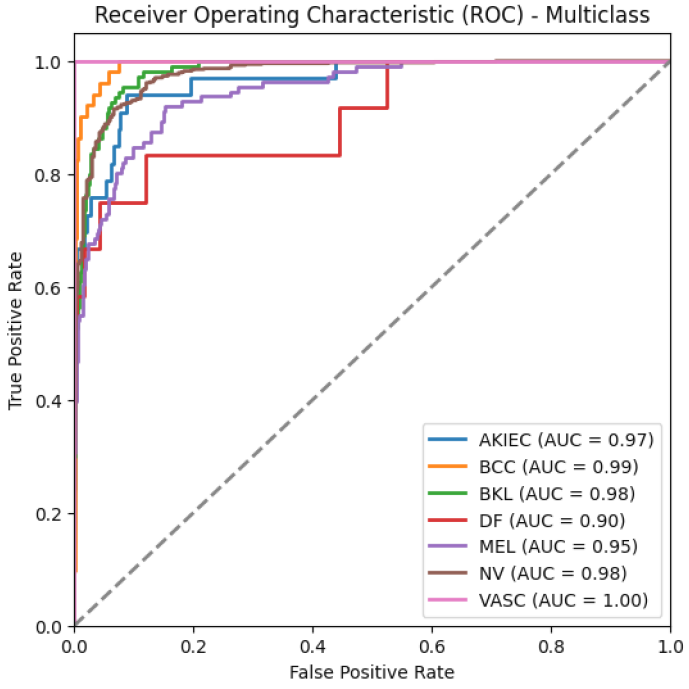


ROC-AUC curve

```
# 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 [==============================] - 4s 118ms/step
```



Explainable AI (XAI)

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

```
Collecting tf-keras-vis
  Downloading tf_keras_vis-0.8.5-py3-none-any.whl (52 kB)
[?25l     [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ [0m [32m0.0/52.1 kB [0m [31m? [0m eta [36m-:--:-- [0m
```

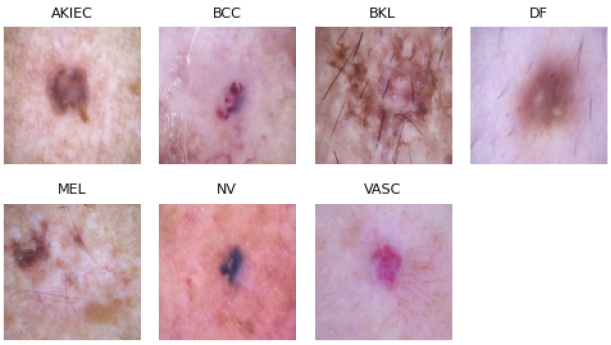
[2K [90m━━ [0m [32m52.1/52.1 kB [0m [31m1.3 MB/s [0m eta [36m0:00:00 [0m [?25hRequirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis) (1.10.1) Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis) (9.4.0) Collecting 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)))
```

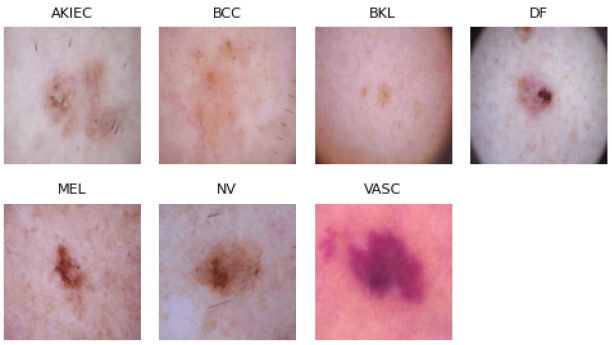
```
TensorFlow recognized 1 GPUs
```

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

# 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))
```



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))
```



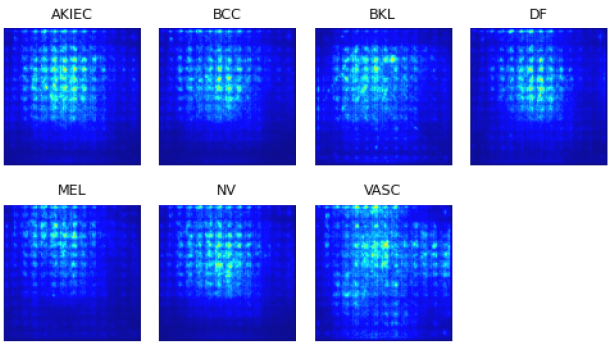

```
from tf_keras_vis.utils.model_modifiers import ReplaceToLinear

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])
```

SmoothGrad

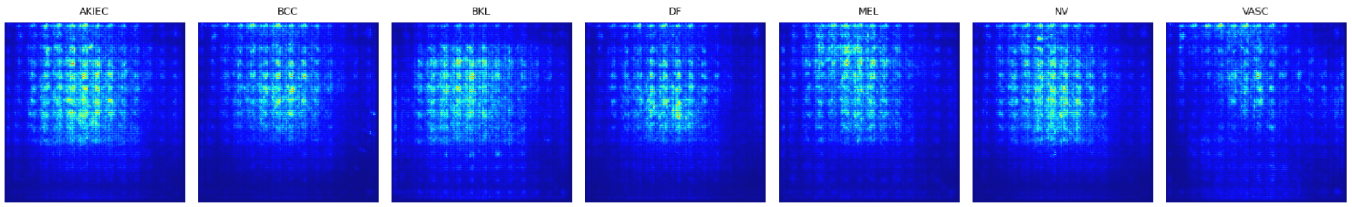
```
%%time
from keras import backend as K
from tf_keras_vis.saliency import Saliency

# Create Saliency object.
saliency = Saliency(model,
                    model_modifier=replace2linear,
                    clone=True)

# Generate saliency map with smoothing that reduce noise by adding noise
saliency_map = saliency(score,
                        X,
                        smooth_samples=20, # The number of calculating gradients iterations.
                        smooth_noise=0.20) # noise spread level.

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

# 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

    ax[row, col].set_title(title, fontsize=8)
    ax[row, col].imshow(saliency_map[i], cmap='jet')
    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()
```



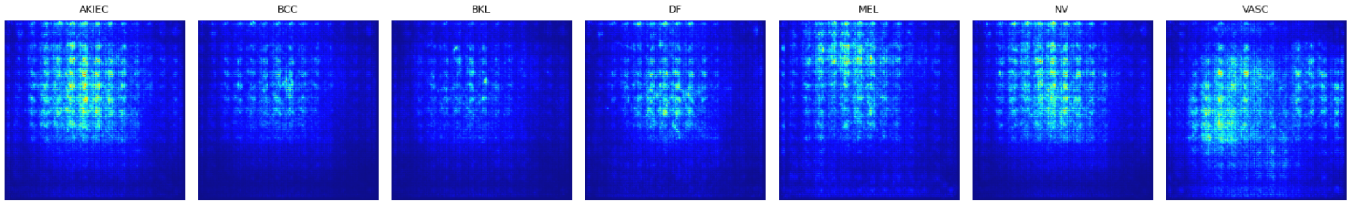

```
CPU times: user 7.24 s, sys: 377 ms, total: 7.62 s
Wall time: 9.59 s
```

```
%%time
from tensorflow.keras import backend as K
from tf_keras_vis.saliency import Saliency

# Create Saliency object.
saliency = Saliency(model, model_modifier=replace2linear, clone=True)

# Generate saliency maps with smoothing that reduces noise by adding noise
saliency_maps = saliency(score, X, smooth_samples=20, smooth_noise=0.20)

# 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):
    axes[i].set_title(title, fontsize=8)
    axes[i].imshow(saliency_maps[i], cmap='jet')
    axes[i].axis('off')

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




```
CPU times: user 5.6 s, sys: 229 ms, total: 5.83 s
Wall time: 5.66 s
```

```
%%time
from tensorflow.keras import backend as K
from tf_keras_vis.saliency import Saliency

# Create Saliency object.
saliency = Saliency(model, model_modifier=replace2linear, clone=True)

# Generate saliency maps with smoothing that reduces noise by adding noise
saliency_maps = saliency(score,
                         X,
                         smooth_samples=20,
                         smooth_noise=0.20)

# 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):
    axes[i].set_title(title, fontsize=13)
    axes[i].imshow(saliency_maps[i], cmap='jet')
    axes[i].axis('off')

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



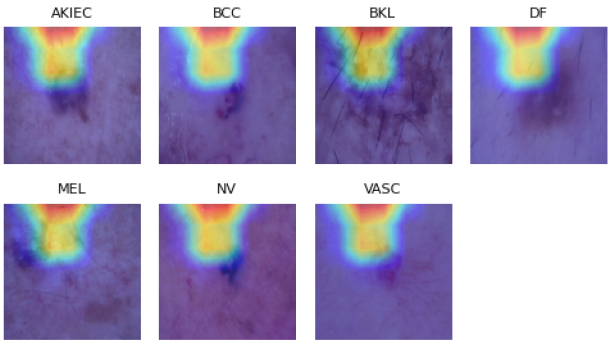

```
CPU times: user 6.79 s, sys: 237 ms, total: 7.03 s
Wall time: 6.95 s
```

Faster Score-CAM

```
%%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()
```

```
3/3 [==============================] - 6s 501ms/step
```

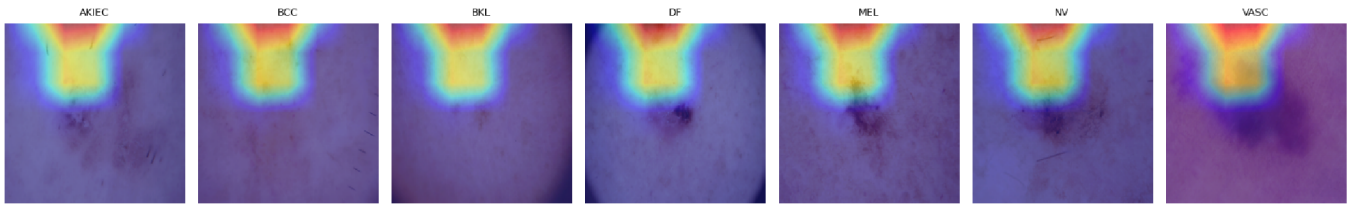



```
CPU times: user 6.87 s, sys: 946 ms, total: 7.82 s
Wall time: 17.8 s
```

```
%%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()
```

```
3/3 [==============================] - 1s 151ms/step
```




```
CPU times: user 3.31 s, sys: 273 ms, total: 3.59 s
Wall time: 3.91 s
```