

[Link to Colab](#)

# Importing Data from Kaggle

In this step, the data is loaded from kaggle and imported onto the google drive

```
In [1]: from google.colab import drive  
drive.mount('/content/drive')
```

Mounted at /content/drive

For the next cell, keep the json file of the kaggle api key handy

```
In [2]: from google.colab import files  
files.upload()
```

Choose Files no files selected      Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Out[2]: Saving kaggle.json to kaggle.json  
{'kaggle.json': b'{"username": "kuvalbrar", "key": "0136cf88cb7c19ed1d4b6702e39e0ef"}'}
```

```
In [3]: ! pip install -q kaggle
```

```
In [4]: #Loading HAM10000 Dataset  
! mkdir ~/.kaggle  
! cp kaggle.json ~/.kaggle/  
! chmod 600 ~/.kaggle/kaggle.json  
! kaggle datasets list  
! kaggle datasets download -d kmader/skin-cancer-mnist-ham10000  
! mkdir ham_dataset  
! unzip skin-cancer-mnist-ham10000.zip -d ham_dataset
```

Streaming output truncated to the last 5000 lines.

```
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029325.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029326.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029327.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029328.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029329.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029330.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029331.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029332.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029333.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029334.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029335.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029336.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029337.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029338.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029339.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029340.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029341.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029342.jpg  
inflating: ham_dataset/ham10000_images_part_2/ISIC_0029343.jpg
```























































































































inflating: ham dataset/ham10000 images part 2/ISIC\_0032837.jpg



















































```
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034294.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034295.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034296.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034297.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034298.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034299.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034300.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034301.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034302.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034303.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034304.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034305.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034306.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034307.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034308.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034309.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034310.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034311.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034312.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034313.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034314.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034315.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034316.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034317.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034318.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034319.jpg
inflating: ham_dataset/ham10000_images_part_2/ISIC_0034320.jpg

inflating: ham_dataset/hmnist_28_28_L.csv
inflating: ham_dataset/hmnist_28_28_RGB.csv
inflating: ham_dataset/hmnist_8_8_L.csv
inflating: ham_dataset/hmnist_8_8_RGB.csv
```

## Processing Dataset (Alternative)

```
In [32]: !pip install -q albumentations==1.2.1
```

```
In [33]: import albumentations as A
from albumentations.pytorch import ToTensorV2
from torch.utils.data import ConcatDataset, DataLoader, Subset
import random
from collections import Counter

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
from glob import glob #to walk through folder structure
import seaborn as sns #for plotting confusion matrix
from PIL import Image #for displaying images and format images
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

import torch
from torch.nn.functional import F
```

```
import torch.nn.functional as F
import torch.nn as nn
import torch.optim as optim

import torchvision
from torch.utils.data.sampler import SubsetRandomSampler
import torchvision.transforms as transforms
from torch.utils.data import random_split
from torch.utils.data import DataLoader, Dataset

from scipy import stats
from sklearn.preprocessing import LabelEncoder #for formatting ouput labels
```

```
In [34]: #Loading the dataset as a class. The same thing as the other method but just
class HAM10000(Dataset):
    def __init__(self, csv_path, img_dir, transform=None):
        self.data = pd.read_csv(csv_path) #same as before. Reading the dataset.
        self.img_dir = img_dir
        self.transform = transform

        self.label_map = {label: index for index, label in enumerate(self.data[''
        self.num_classes = len(self.label_map)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        img_name = self.data.iloc[index, 1] + '.jpg' #locates the name of the image
        img = None
        for img_dir in self.img_dir: #checking for available images
            img_path = os.path.join(img_dir, img_name)
            if os.path.exists(img_path):
                img = Image.open(img_path)
                break

        if img is None:
            raise FileNotFoundError(f"Image file not found: {img_path}")

        label = self.data.iloc[index, 2] #self.data.iloc[index, 2] -> indexing
        label = self.label_map[label] #converts to integer

        if self.transform: #Applies transform onto image
            img = self.transform(img)

        label_one_hot = torch.nn.functional.one_hot(torch.tensor(label), num_cla
    return img, label_one_hot
```

```
In [35]: transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.22]
])

metadata_path= '/content/ham_dataset/HAM10000_metadata.csv'
img_dir= [ '/content/ham_dataset/HAM10000_images_part_1', '/content/ham_datas

dataset = HAM10000(csv_path=metadata_path, img_dir=img_dir, transform=transf
```

```
In [36]: class DatasetFromList(Dataset): #just a class that gets the item(dataset) fr
    def __init__(self, data_lst):
        self.data_lst = data_lst

    def __len__(self):
        return len(self.data_lst)

    def __getitem__(self, index):
        return self.data_lst[index]
```

```
In [37]: def Augment_Duplicate_HAM10000(Dataset, aug_transform, target_class_count=None):
    labels = [label.argmax().item() for _, label in dataset]

    data_counts = Counter(labels) #counts the number of appearances for each unique label
    max_count = max(data_counts.values()) #locate the max number of appearance

    if not target_class_count:
        target_class_count = {label: max_count for label in data_counts} #locate the target class count for each label

    aug_data = []

    for label, count in data_counts.items():
        indices = [i for i, (_, lbl) in enumerate(dataset) if lbl.argmax().item() == label]
        num_to_duplicate = target_class_count[label] - count #Calc the difference between target and current count

        for _ in range(num_to_duplicate):
            idx = random.choice(indices) #idx is just grabs a random index corresponding to the label
            image, label = dataset[idx] #defines the image and label from the duplicate

            if image.shape[0] == 3:
                image_np = image.permute(1, 2, 0).numpy()

            else:
                image_np = image.numpy()

            image_np = image_np.permute(1, 2, 0).numpy() #Convert to numpy
            image_np = (image_np * 255).astype(np.uint8)
            image = Image.fromarray(image_np) #Convert to PIL image mememory
            image_rgb = image.convert('RGB') #Convert to rgb image
            aug_image = aug_transform(image=np.array(image_rgb))['image'] #Applies augmentation

            if aug_image.shape[-1] == 3:
                aug_image_tensor = aug_image.permute(2, 0, 1)

            else:
                aug_image_tensor = aug_image.clone().detach()

            aug_image = aug_image.float()/255.0 #Normalizes the image

            aug_data.append((aug_image, label))

    aug_dataset = DatasetFromList(aug_data) #Adds all augmented data into one dataset
    return ConcatDataset([dataset, aug_dataset]) #Adds augmented duplicate data into original dataset
```

```
In [38]: #define a bunch of augmentations and their specs
aug_transform = A.Compose([
    A.Resize(height=224, width=224),
    A.RandomRotate90(),
    A.Flip(),
    A.OneOf([
        A.GaussNoise(var_limit=(10.0, 50.0)),
    ], p=0.2),
    A.OneOf([
        A.MotionBlur(p=0.2),
        A.MedianBlur(blur_limit=3, p=0.1),
        A.Blur(blur_limit=3, p=0.1),
    ], p=0.2),
    A.OneOf([
        A.Sharpen(),
        A.RandomBrightnessContrast(),
    ], p=0.3),
    A.LongestMaxSize(max_size=224),
    A.PadIfNeeded(min_height=224, min_width=224, border_mode=0),
    ToTensorV2()
])
```

```
In [39]: #split the og dataset into train and test
labels_og = [label for _, label in dataset]

#split the testing index from untouched dataset 80/20 test, train
train_val_index, test_index = train_test_split(np.arange(len(dataset))), test

#create a subset from original dataset (80% of remainging) to augment
remianing_dataset= Subset(dataset, train_val_index)

aug_dataset = Augment_Duplicate_HAM10000(remianing_dataset, aug_transform)
```

```
In [40]: #Dataloader functions
labels = [label for _, label in aug_dataset]

#Stratified split (60-20-20)

train_index, val_index = train_test_split(np.arange(len(aug_dataset)), test_size=0.2, random_state=42)
#Assigning the data to each index
train_dataset = Subset(aug_dataset, train_index)
val_dataset = Subset(aug_dataset, val_index)
test_dataset = Subset(dataset, test_index)

def get_dataLoader(dataset, batch_size):
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=True)

    return train_loader, val_loader

bs=128
train_loader, val_loader = get_dataLoader(aug_dataset, batch_size=bs)
test_loader = DataLoader(test_dataset, batch_size=bs, shuffle=True)
```

In [41]: *#this is a checkpoint cell to see the shapes of the loaders*

```
for images, labels in train_loader:
    print(images.shape)
    print(labels.shape)
    break

for images, labels in test_loader:
    print(images.shape)
    print(labels.shape)
    break

for images, labels in val_loader:
    print(images.shape)
    print(labels.shape)
    break
```

```
torch.Size([128, 3, 224, 224])
torch.Size([128, 7])
torch.Size([128, 3, 224, 224])
torch.Size([128, 7])
torch.Size([128, 3, 224, 224])
torch.Size([128, 7])
```

In [42]: `use_cuda = torch.cuda.is_available()`  
`device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")`

## Visualizing the Dataset:

Analyze the metadata:

image\_id = unique id assigned to all images \ dx column = classification (diagnosis) \

dx\_type = how the diagnosis was confirmed \ localization = part of body with skin lesion

In [43]:

```
#establish path to metadata.csv
metadata_path= '/content/ham_dataset/HAM10000_metadata.csv'
metadata = pd.read_csv(metadata_path)
metadata.head()
```

Out[43]:

	lesion_id	image_id	dx	dx_type	age	sex	localization
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear

Process the data, lets convert the diagnoses types into numerical catagories and save as new column

In [44]:

```
label_encoder= LabelEncoder()
label_encoder.fit(metadata['dx'])
metadata['label'] = label_encoder.transform(metadata['dx'])

#move the new column beside the dx
metadata = metadata[['lesion_id', 'image_id', 'label', 'dx', 'dx_type', 'age']]
img_path={os.path.splitext(os.path.basename(x))[0]: x for x in glob(os.path.
img_path.update({os.path.splitext(os.path.basename(x))[0]: x for x in glob(c
metadata['path'] = metadata['image_id'].map(img_path.get)
metadata.head(-5)
```

Out[44]:

	lesion_id	image_id	label	dx	dx_type	age	sex	localization
0	HAM_0000118	ISIC_0027419	2	bkl	histo	80.0	male	scalp /content
1	HAM_0000118	ISIC_0025030	2	bkl	histo	80.0	male	scalp /content
2	HAM_0002730	ISIC_0026769	2	bkl	histo	80.0	male	scalp /content
3	HAM_0002730	ISIC_0025661	2	bkl	histo	80.0	male	scalp /content
-	-----	-----	-	-	-	-	-	-

4	HAM_0001466	ISIC_0031633	2	bkl	histo	/5.0	male	ear	/conte
...	...	...	...	...	...	...	...	...	...
10005	HAM_0005579	ISIC_0028393	0	akiec	histo	80.0	male	face	/conte
10006	HAM_0004034	ISIC_0024948	0	akiec	histo	55.0	female	face	/conte
10007	HAM_0001565	ISIC_0028619	0	akiec	histo	60.0	female	face	/conte
10008	HAM_0001576	ISIC_0033705	0	akiec	histo	60.0	male	face	/conte
10009	HAM_0005705	ISIC_0031430	0	akiec	histo	75.0	female	lower extremity	/conte

10010 rows × 9 columns

Visualize the data up to this point in the processing part

In [45]:

```
#plot the sex, localization, and dx
figures= plt.figure(figsize=(15,15))
# Calculate the counts for each category
sex_counts = metadata['sex'].value_counts()
localization_counts = metadata['localization'].value_counts()
dx_counts = metadata['dx'].value_counts()
# Sort the categories based on the counts
sorted_sex = sex_counts.index
sorted_localization = localization_counts.index
sorted_dx = dx_counts.index

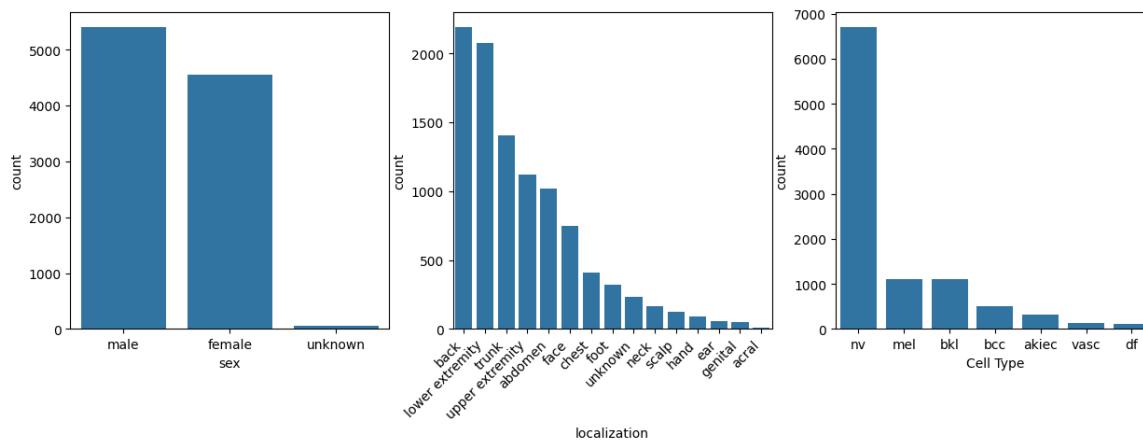
#graph 1
figures.add_subplot(3,3,1)
sns.countplot(x='sex', data=metadata, order=sorted sex)
```

```
#graph 2
figures.add_subplot(3,3,2)
sns.countplot(x='localization', data=metadata, order=sorted_localization)

plt.xticks(
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='medium'
)

#graph 3
figures.add_subplot(3,3,3)
sns.countplot(x='dx', data=metadata, order=sorted_dx)
plt.xlabel("Cell Type")
```

Out[45]: Text(0.5, 0, 'Cell Type')



From the diagnosis graph, we can see the images are not evenly spread across the different types of diagnosis. There are more lesions of nv type than df type. We must normalize and split the data evenly. Combine the methods of oversampling and class weights

```
In [46]: print(metadata['label'].value_counts())
label_map = {0: 'akiec', 1: 'bcc', 2: 'bkl', 3: 'df', 4: 'mel', 5: 'nv', 6:
```

Label	Count
5	6705
4	1113
2	1099
1	514
0	327
6	142
3	115

Name: count, dtype: int64

```
In [47]: def imshow(img, title=None):
    img = img.permute(1, 2, 0)
    img = img.detach().cpu().numpy()
    img = (img * 255).clip(0, 255).astype(np.uint8)
    plt.imshow(img)
    if title:
        plt.title(title)
    plt.pause(0.001)

def visualize_dataloader(dataloader, num_img=5, rows=3, cols=5):
    fig = plt.figure(figsize=(15, 15))

    data_iter = iter(dataloader)
    images, labels = next(data_iter)

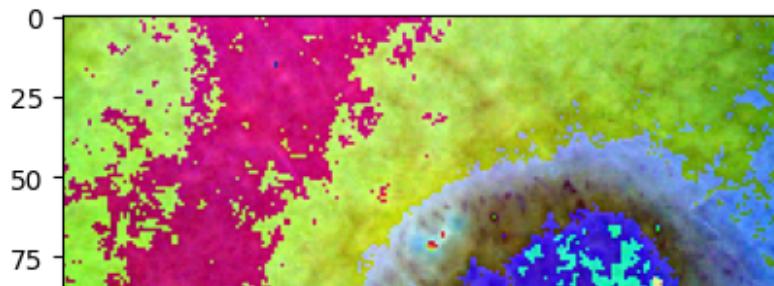
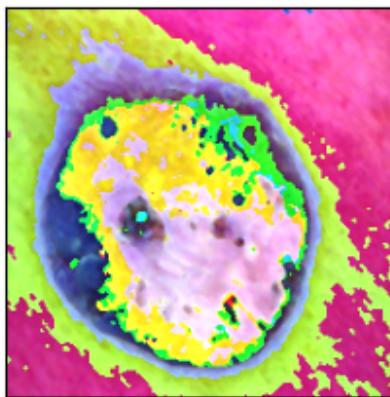
    for i in range(num_img):
        ax = fig.add_subplot(rows, cols, i+1, xticks=[], yticks=[])
        # For one-hot encoded labels
        if labels.ndim > 1:
            ax.set_title(f"Label: {labels[i].argmax().item()}")
        else:
            ax.set_title(f"Label: {labels[i].item()}")

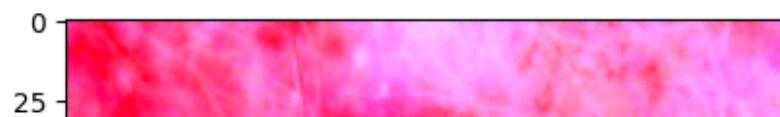
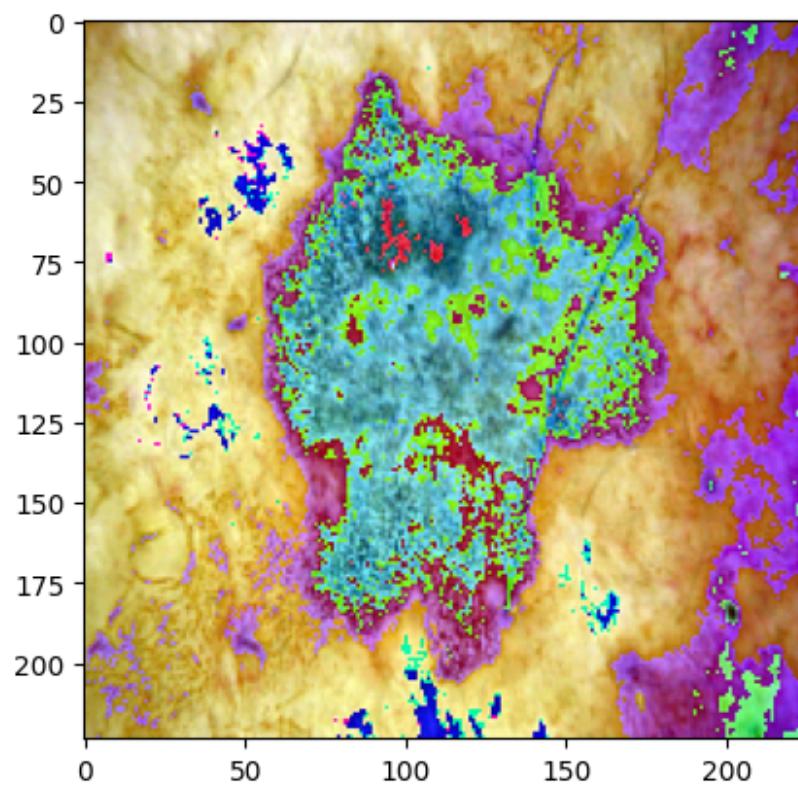
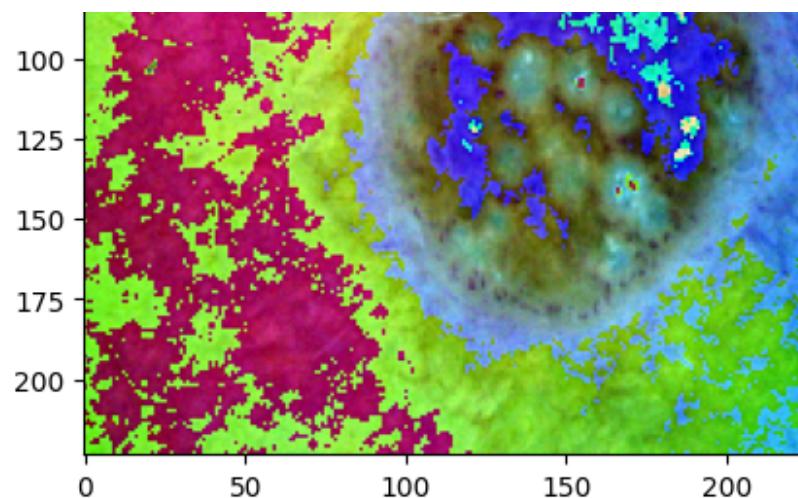
        imshow(images[i])

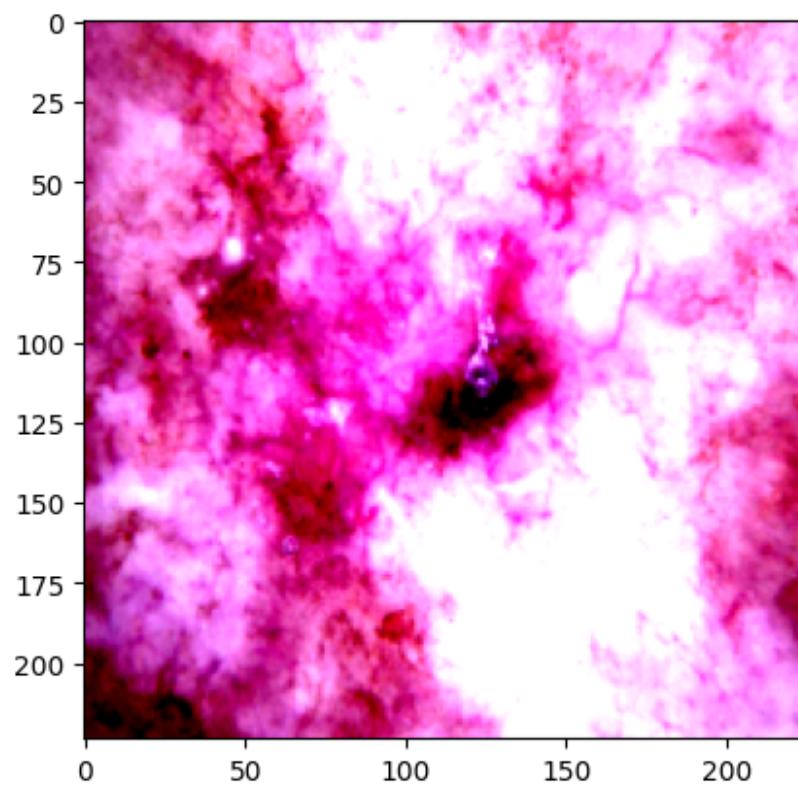
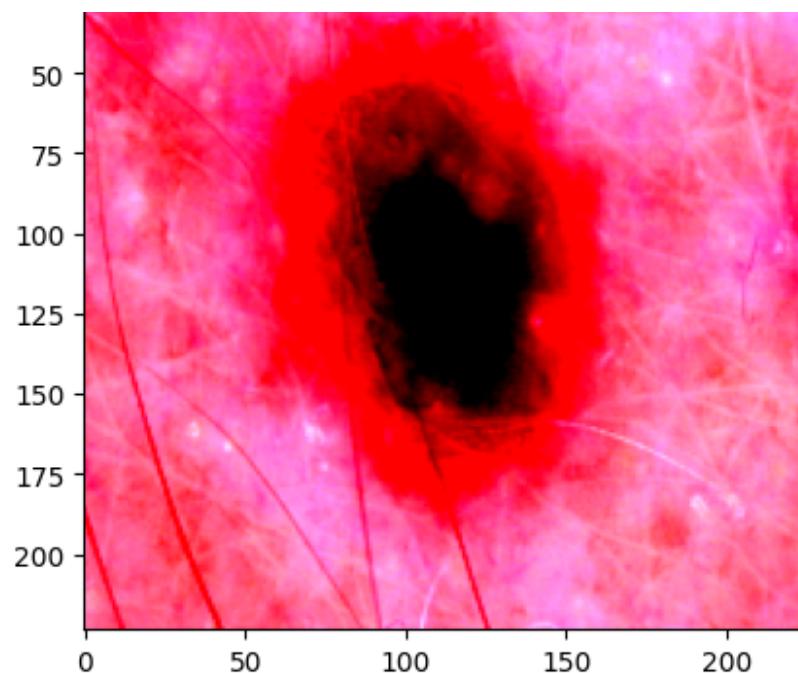
    plt.show()

# Visualize 15 images in 3 rows and 5 columns
visualize_dataloader(train_loader, num_img=5, rows=3, cols=5)
visualize_dataloader(val_loader, num_img=5, rows=3, cols=5)
visualize_dataloader(test_loader, num_img=5, rows=3, cols=5)
```

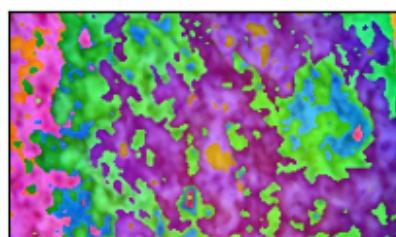
Label: 0

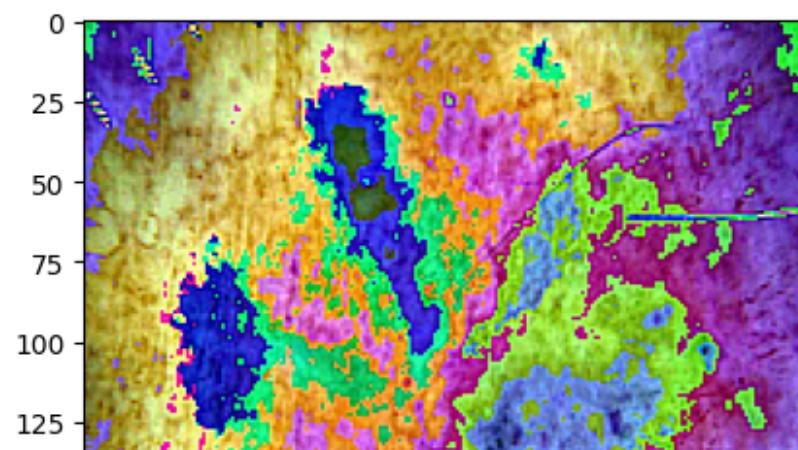
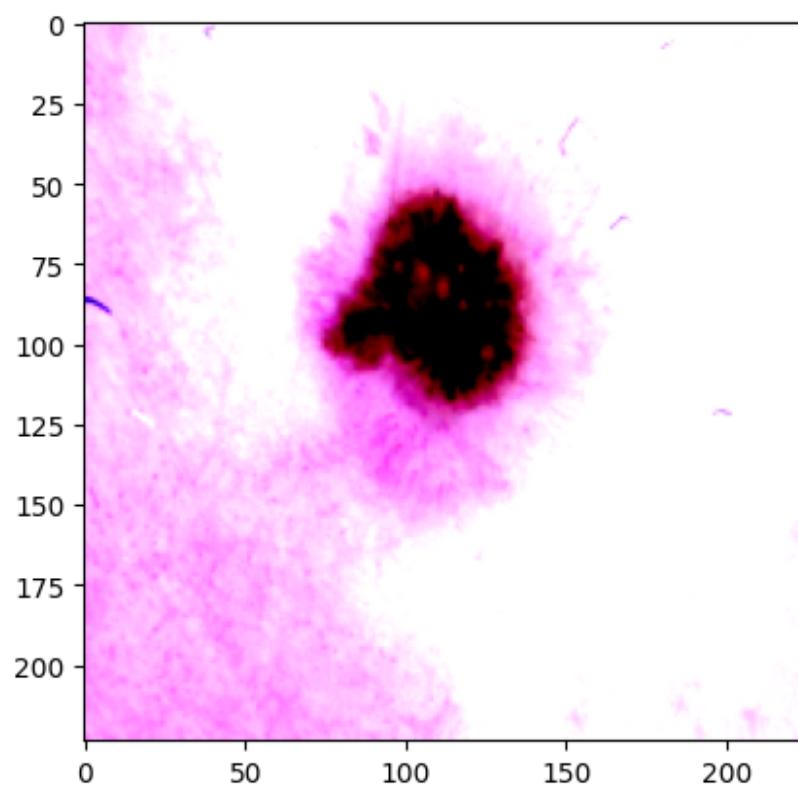
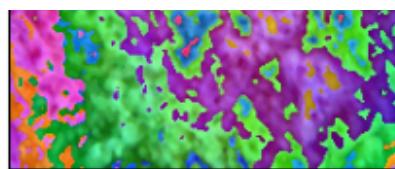


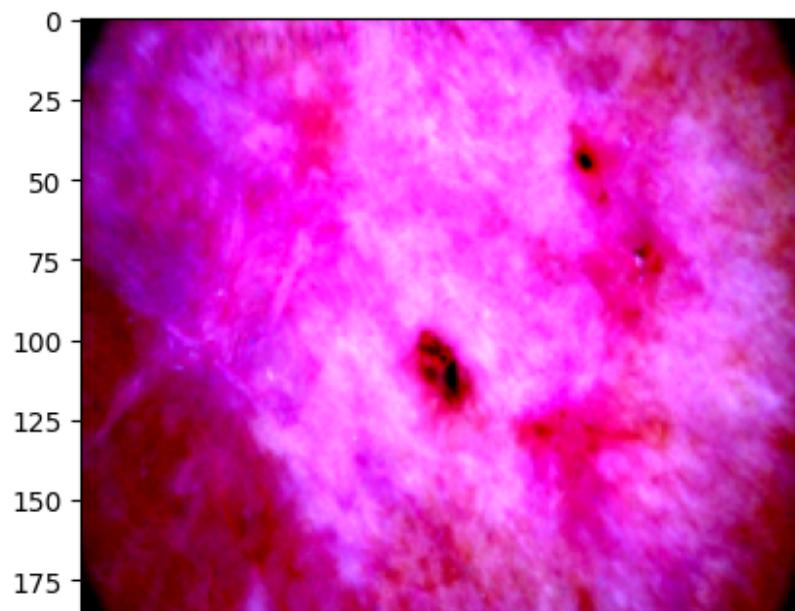
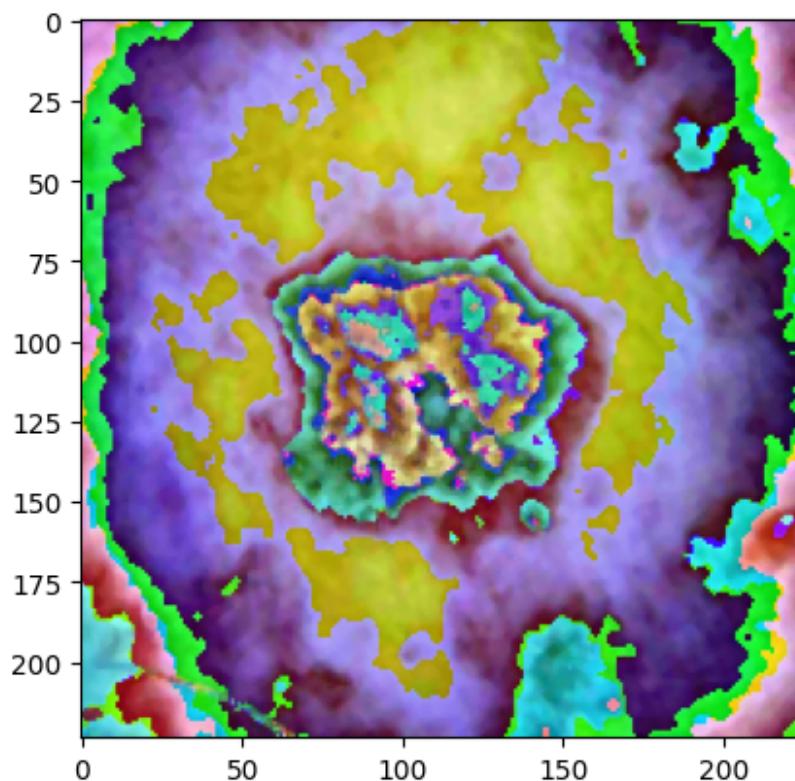
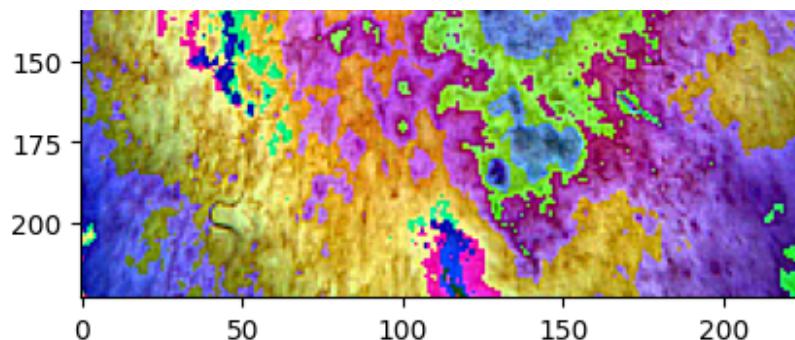


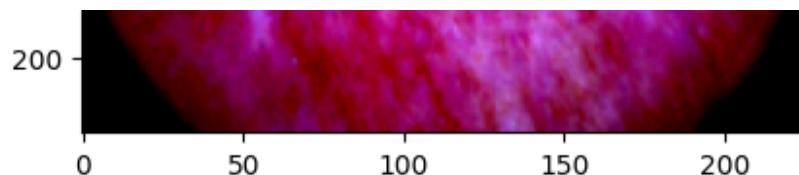


Label: 0

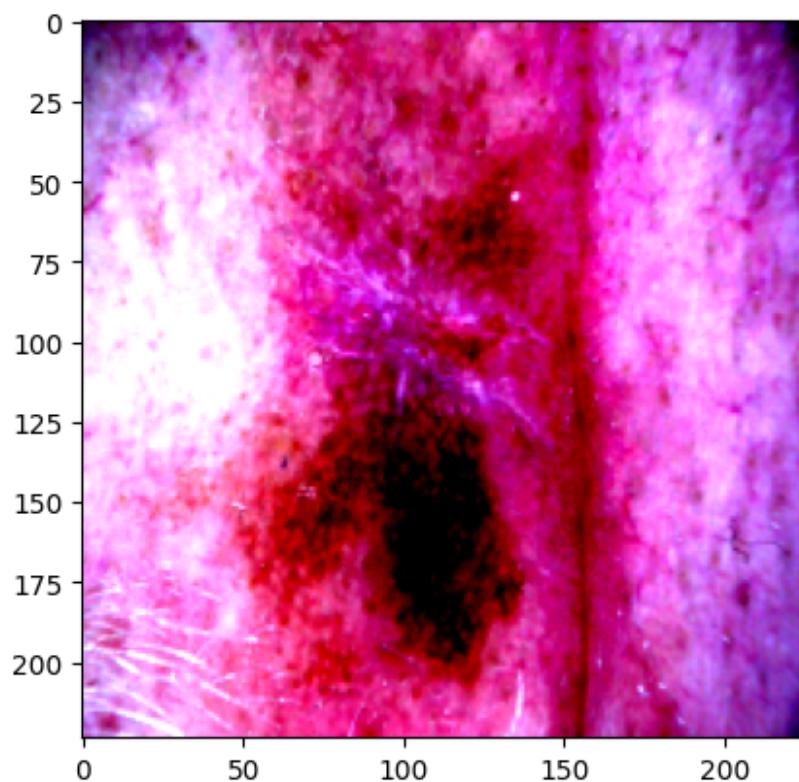
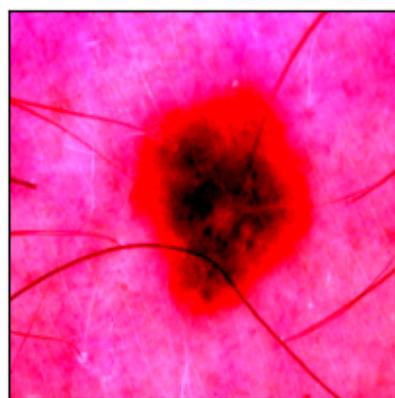


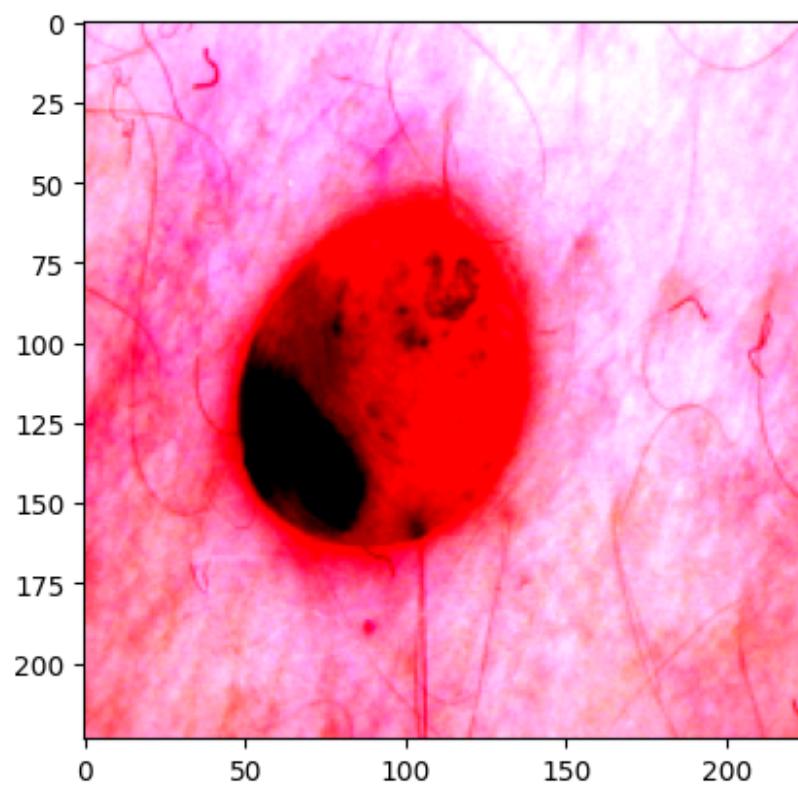
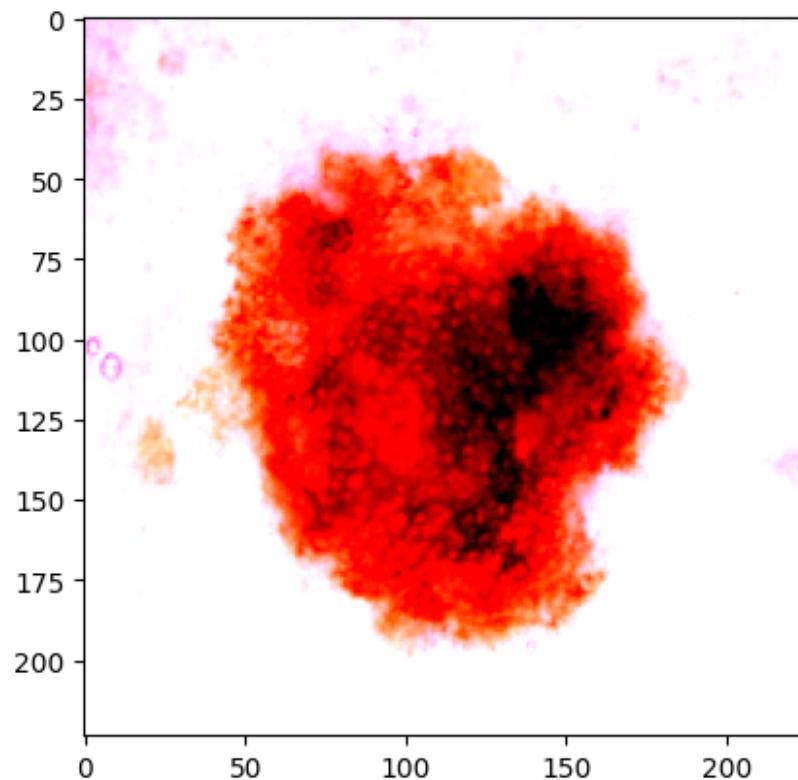


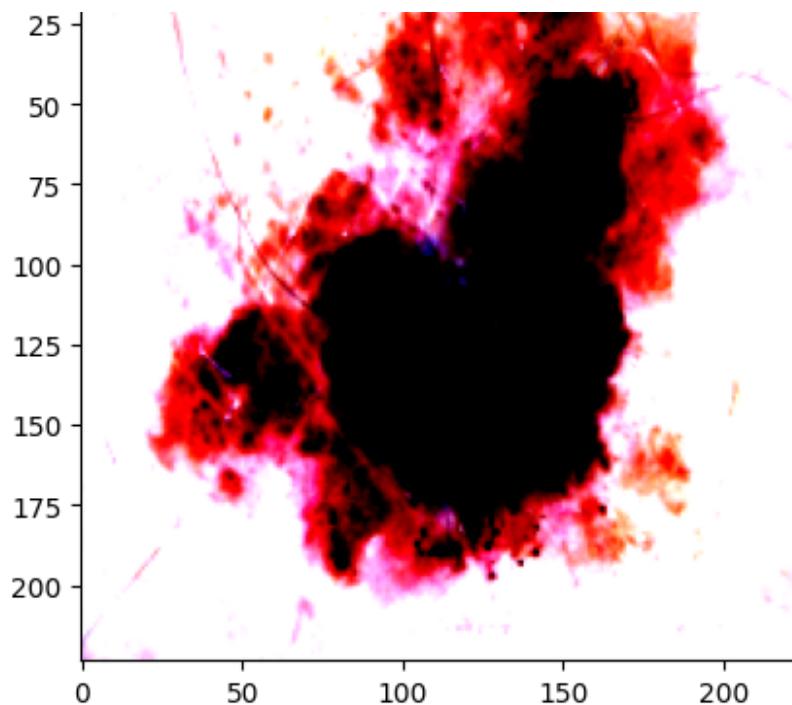




Label: 1







## Baseline Model Testing and Score

```
In [ ]: #Use Naive Bayes as it can be used for multiclassification for the different  
#First we need to extract the features and labels from the data loaders and  
from sklearn.naive_bayes import GaussianNB  
  
#Establishing Base Model Naive Bayes for HAM1000  
def NaiveBayes_model(training_dataset, training_labels, testing_dataset, tes  
model = GaussianNB() #Defining model
```

```
model.fit(training_dataset, training_labels) #Fit model to training dataset

testing_prediction = model.predict(testing_dataset) #Make predictions

#For debugging purposes: print the prediction and test labels
#print(testing_prediction)
#print(testing_labels)

#Calculate score
correct = 0
for i in range(len(testing_prediction)):
    if testing_prediction[i] == testing_labels[i]:
        correct += 1
    else:
        pass
score = 100*(correct/len(testing_prediction))

#score = 100*(1-sum(abs(testing_prediction - testing_labels))/len(testing_labels))
return print("Naive Bayes Test:", score)
```

In [ ]:

```
def loader_to_numpy(loader, device='cuda'):
    data_list = []
    labels_list = []

    # Disable gradients for faster computations
    with torch.no_grad():
        for data, labels in loader:
            data = data.to(device).view(data.size(0), -1) # Flatten the data
            labels = labels.to(device)
            data_list.append(data) # Keep the data on the GPU
            labels_list.append(labels)

    # Concatenate all batches into a single tensor
```

```

data = torch.cat(data_list, dim=0)
labels = torch.cat(labels_list, dim=0)

# Decode one-hot encoded labels to class indices
labels = torch.argmax(labels, axis=1)

# Convert to numpy if needed (this will move the data back to CPU)
data_np = data.cpu().numpy()
labels_np = labels.cpu().numpy()

return data_np, labels_np

# Run on GPU
training_dataset, training_labels = loader_to_numpy(train_loader, device='cpu')
testing_dataset, testing_labels = loader_to_numpy(test_loader, device='cuda')

```

In [ ]: NaiveBayes\_model(training\_dataset, training\_labels, testing\_dataset, testing

## Primary Model

```

In [48]: #create our model to train
#define model
class SkinCancerCNN(nn.Module):
    def __init__(self):
        super(SkinCancerCNN, self).__init__()
        self.name = 'SkinCancerCNN'

        self.conv1 = nn.Conv2d(in_channels=3, out_channels=256, kernel_size=3, padding=1)
        self.BatchNorm1 = nn.BatchNorm2d(256)

        self.conv2 = nn.Conv2d(in_channels=256, out_channels=128, kernel_size=3, padding=1)
        self.BatchNorm2 = nn.BatchNorm2d(128)

        self.conv3 = nn.Conv2d(in_channels=128, out_channels=64, kernel_size=3, padding=1)
        self.BatchNorm3 = nn.BatchNorm2d(64)

        self.conv4 = nn.Conv2d(in_channels=64, out_channels=32, kernel_size=3, padding=1)
        self.BatchNorm4 = nn.BatchNorm2d(32)

        self.conv5 = nn.Conv2d(in_channels=32, out_channels=16, kernel_size=3, padding=1)
        self.BatchNorm5 = nn.BatchNorm2d(16)

        self.dropout = nn.Dropout(p=0.4) #50% drop out
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        self.flatten = nn.Flatten()

        self.fc1 = nn.Linear(256, 64)
        self.fc2 = nn.Linear(64, 7)

    def forward(self, x):
        #print(x)

```

```

x = self.conv1(x)
x = self.BatchNorm1(x)
x = F.relu(x)
x = self.pool(x) # output is 256x64x64

x = self.conv2(x)
x = self.BatchNorm2(x)
x = F.relu(x) # output is 128x64x64
x = self.pool(x) # output is 128x32x32
x = self.dropout(x)

x = self.conv3(x)
x = self.BatchNorm3(x)
x = F.relu(x)
x = self.pool(x) # output is 64x32x32

x = self.conv4(x)
x = self.BatchNorm4(x)
x = F.relu(x)
x = self.pool(x) # output is 32x32x32
x = self.dropout(x)

x = self.conv5(x)
x = self.BatchNorm5(x)
x = F.relu(x)
x = self.pool(x) # output is 16x32x32

x = self.flatten(x)
#print(x.shape)
x = self.fc1(x)
x = F.relu(x)

x = self.dropout(x)

x = self.fc2(x)

# print(f"Final output: {x.shape}")
return x

```

SkinCancerCNN = SkinCancerCNN().to(device)

In [49]:

```

class SkinCancerCNN2(nn.Module):
    def __init__(self):
        super(SkinCancerCNN2, self).__init__()
        self.name = 'SkinCancerCNN2'

        self.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, padding=1)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv2 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv3 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)

        # New layer added
        self.conv4 = nn.Conv2d(in_channels=256, out_channels=512, kernel_size=3, padding=1)
        self.pool4 = nn.MaxPool2d(kernel_size=2, stride=2)
    
```

```
# New layer added
self.conv5 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3,
self.pool5 = nn.MaxPool2d(kernel_size=2, stride=2)

self.dropout1 = nn.Dropout(p=0.4) # Increase dropout rate to 40%

self.flatten = nn.Flatten()

self.fc1 = nn.Linear(512 * 4 * 4, 256)

self.dropout2 = nn.Dropout(p=0.5) # Additional dropout layer

self.fc2 = nn.Linear(256, 128)

self.fc3 = nn.Linear(128, 7)

def forward(self, x):
    x = self.conv1(x)
    x = F.relu(x)
    x = self.pool1(x)

    x = self.conv2(x)
    x = F.relu(x)
    x = self.pool2(x)

    x = self.conv3(x)
    x = F.relu(x)
    x = self.pool3(x)

    x = self.conv4(x)
    x = F.relu(x)
    x = self.pool4(x)

    x = self.conv5(x)
    x = F.relu(x)
    x = self.pool5(x)

    x = self.dropout1(x)

    x = self.flatten(x)

    x = self.fc1(x)
    x = F.relu(x)

    x = self.dropout2(x)

    x = self.fc2(x)
    x = F.relu(x)

    x = self.fc3(x)

    return x

SkinCancerCNN2 = SkinCancerCNN2().to(device)
```

```
In [50]: class SkinCancerCNN3(nn.Module):
    def __init__(self):
```

```
super(SkinCancerCNN3, self).__init__()
self.name = 'SkinCancerCNN3'

#3 Convolutional Layers
self.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, pa
self.conv2 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3,
self.conv3 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3,

#1 Pooling Layer
self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

1# Batch Normalization Layer
self.BatchNorm1= nn.BatchNorm2d(64)
self.BatchNorm2= nn.BatchNorm2d(128)
self.BatchNorm3= nn.BatchNorm2d(256)

#2 Dropout Layer
self.dropout1 = nn.Dropout(p=0.2) # Increase dropout rate to 20%
self.dropout2 = nn.Dropout(p=0.3) # Additional dropout layer

self.flatten = nn.Flatten()

#3 Fully-Connected Linear Layers
self.fc1 = nn.Linear(64*56*56, 512)
self.fc2 = nn.Linear(512, 256)
self.fc3 = nn.Linear(256, 7)

def forward(self, x):
    x = self.conv1(x)
    x = F.relu(x)
    x = self.BatchNorm1(x)
    x = self.pool(x)
    x = self.dropout1(x)

    x = self.conv2(x)
    x = F.relu(x)
    x = self.BatchNorm2(x)
    x = self.pool(x)
    x = self.dropout1(x)

    x = self.conv3(x)
    x = F.relu(x)
    x = self.BatchNorm3(x)
    x = self.pool(x)
    x = self.dropout1(x)

    x = self.flatten(x)

    x = self.fc1(x)
    x = F.relu(x)
    x = self.dropout2(x)

    x = self.fc2(x)
    x = F.relu(x)
    x = self.dropout2(x)

    x = self.fc3(x)
```

```
    return x

SkinCancerCNN3 = SkinCancerCNN3().to(device)
```

## Train Model

```
In [51]: #plot the accuracy curves
def plot_curves(train_acc, val_acc):
    plt.figure(figsize=(10, 6))
    plt.plot(train_acc, label='Train Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Accuracy Curves')
    plt.legend()

In [52]: #used for checkpointing, creates a name for current model configs
def get_model_name(name, batch_size, learning_rate, epoch):
    path = "model_{0}_bs{1}_lr{2}_epoch{3}".format(name, batch_size, learning_rate, epoch)
    return path

In [53]: torch.cuda.empty_cache()

In [54]: import torch.optim.lr_scheduler as lr_scheduler

# Assuming use_cuda and device are already defined
use_cuda = torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")

# Define the get_accuracy function
def get_accuracy(model, data_loader, device):
    model.eval() # Set the model to evaluation mode
    correct_predictions = 0
    total_samples = 0

    with torch.no_grad(): # Disable gradient computation
        for images, labels in data_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images.float())
            _, predicted = torch.max(outputs, 1)

            # print(predicted)
            # print(labels)

            correct_predictions += (predicted == torch.argmax(labels, dim=1)).sum()
            total_samples += labels.size(0)

    accuracy = correct_predictions / total_samples
    return accuracy

# Define the train function
def train(model, train_loader, val_loader, criterion, optimizer, num_epochs, i
```

```

for epoch in range(num_epochs):
    running_loss = 0.0
    model.train() # Set the model to train mode

    for images, labels in train_loader:
        # To Enable GPU Usage
        if use_cuda and torch.cuda.is_available():
            images, labels = images.cuda(), labels.cuda()
            model.cuda()

        images, labels = images.to(device), labels.to(device) # Move data to device
        optimizer.zero_grad() # Zero the gradients
        # Forward pass
        outputs = model(images.float())
        loss = criterion(outputs, labels.float())
        # Backward pass and optimize
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * images.size(0)

    epoch_loss = running_loss / len(train_loader.dataset)
    train_accuracy = get_accuracy(model, train_loader, device)
    val_accuracy = get_accuracy(model, val_loader, device)

    train_acc.append(train_accuracy)
    val_acc.append(val_accuracy)

    if scheduler is not None:
        scheduler.step(val_accuracy)

    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}, Train Accuracy: {train_accuracy:.4f}, Val Accuracy: {val_accuracy:.4f}')
    plot_curves(train_acc, val_acc)

```

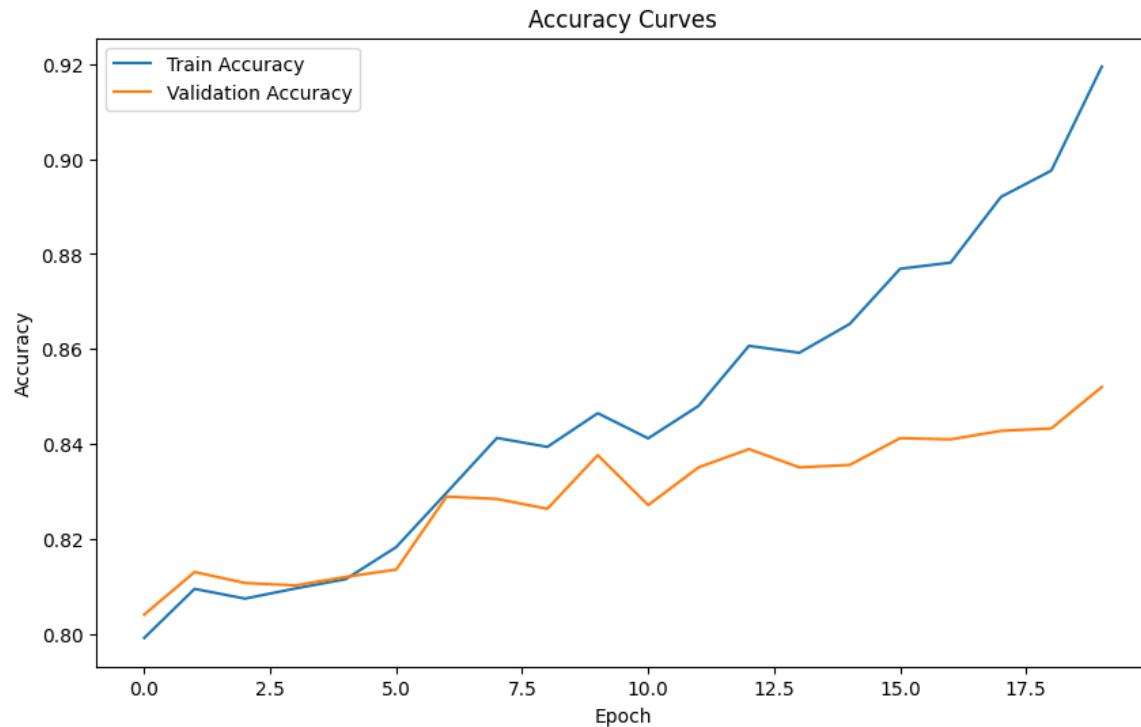
## Training and Tuning Hyperparameters

In [55]: `torch.cuda.empty_cache()`

In [56]: `#Select Model Here`  
`model = SkinCancerCNN3`

In [57]: `# Train the model`  
`lr = 1e-3`  
`num_epochs = 20`  
`optimizer = optim.Adam(model.parameters(), lr=lr)`  
`criterion = nn.CrossEntropyLoss() # Multi-classification loss function applied`  
`scheduler = lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1)`  
`train(model, train_loader, val_loader, criterion, optimizer, num_epochs=num_epochs)`

```
Epoch [1/20], Loss: 1.9592, Train Accuracy: 0.7992, Val Accuracy: 0.8041
Epoch [2/20], Loss: 0.6351, Train Accuracy: 0.8096, Val Accuracy: 0.8131
Epoch [3/20], Loss: 0.5951, Train Accuracy: 0.8075, Val Accuracy: 0.8108
Epoch [4/20], Loss: 0.5742, Train Accuracy: 0.8096, Val Accuracy: 0.8103
Epoch [5/20], Loss: 0.5538, Train Accuracy: 0.8116, Val Accuracy: 0.8121
Epoch [6/20], Loss: 0.5354, Train Accuracy: 0.8184, Val Accuracy: 0.8136
Epoch [7/20], Loss: 0.5199, Train Accuracy: 0.8298, Val Accuracy: 0.8290
Epoch [8/20], Loss: 0.5081, Train Accuracy: 0.8413, Val Accuracy: 0.8285
Epoch [9/20], Loss: 0.4941, Train Accuracy: 0.8394, Val Accuracy: 0.8264
Epoch [10/20], Loss: 0.4865, Train Accuracy: 0.8465, Val Accuracy: 0.8377
Epoch [11/20], Loss: 0.4757, Train Accuracy: 0.8412, Val Accuracy: 0.8272
Epoch [12/20], Loss: 0.4618, Train Accuracy: 0.8481, Val Accuracy: 0.8351
Epoch [13/20], Loss: 0.4440, Train Accuracy: 0.8607, Val Accuracy: 0.8390
Epoch [14/20], Loss: 0.4432, Train Accuracy: 0.8592, Val Accuracy: 0.8351
Epoch [15/20], Loss: 0.4227, Train Accuracy: 0.8653, Val Accuracy: 0.8356
Epoch [16/20], Loss: 0.4101, Train Accuracy: 0.8769, Val Accuracy: 0.8413
Epoch [17/20], Loss: 0.4056, Train Accuracy: 0.8782, Val Accuracy: 0.8410
Epoch [18/20], Loss: 0.3737, Train Accuracy: 0.8920, Val Accuracy: 0.8428
Epoch [19/20], Loss: 0.3568, Train Accuracy: 0.8976, Val Accuracy: 0.8433
Epoch [20/20], Loss: 0.3365, Train Accuracy: 0.9194, Val Accuracy: 0.8520
```



```
In [59]: #save the model
torch.save(model.state_dict(), 'model.pth')
```

In [60]:

```
#get the shape of the dataloaders
for images, labels in train_loader:
    print(images.shape)
    print(labels.shape)
    break
for images, labels in val_loader:
    print(images.shape)
    print(labels.shape)
    break
for images, labels in test_loader:
    print(images.shape)
    print(labels.shape)
    break

torch.Size([128, 3, 224, 224])
torch.Size([128, 7])
torch.Size([128, 3, 224, 224])
torch.Size([128, 7])
torch.Size([128, 3, 224, 224])
torch.Size([128, 7])
```

In [61]:

```
from sklearn.metrics import classification_report

# test the accuracy
def get_predictions(model, data_loader, device):
    model.eval() # Set the model to evaluation mode

    with torch.no_grad(): # Disable gradient computation
        for images, labels in data_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images.float())
            _, predicted = torch.max(outputs, 1)
            #decode the labels
            labels = torch.argmax(labels, dim=1)

    return predicted, labels

y_pred, y_truth = get_predictions(model, test_loader, device)

#convert y_pred and y_truth to numpy
y_pred = y_pred.cpu().numpy()
y_truth = y_truth.cpu().numpy()

print(classification_report(y_pred, y_truth))
```

precision	recall	f1-score	support
~	~	~	~

0	0.62	0.61	0.64	12
1	0.91	0.84	0.87	57
2	0.00	0.00	0.00	0
3	0.14	0.25	0.18	4
4	1.00	1.00	1.00	2
5	0.50	0.50	0.50	4
6	0.67	0.50	0.57	4
accuracy			0.76	83
macro avg	0.55	0.54	0.54	83
weighted avg	0.80	0.76	0.78	83

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1
471: UndefinedMetricWarning: Recall and F-score are ill-defined and being se
t to 0.0 in labels with no true samples. Use `zero_division` parameter to co
ntrol this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1
471: UndefinedMetricWarning: Recall and F-score are ill-defined and being se
t to 0.0 in labels with no true samples. Use `zero_division` parameter to co
ntrol this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1
471: UndefinedMetricWarning: Recall and F-score are ill-defined and being se
t to 0.0 in labels with no true samples. Use `zero_division` parameter to co
ntrol this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
```

In [62]: `#For testing the model  
test_accuracy = get_accuracy(model, test_loader, device)  
print(f'Test Accuracy: {test_accuracy:.2%}')`

Test Accuracy: 78.08%

## Tuning Transfer Learning

In [ ]: `#Transfer Model (*Note: Skip the training Step)  
from torchvision.models import resnet152, ResNet152_Weights  
weights = ResNet152_Weights.DEFAULT  
transfer_model = resnet152(weights=weights)  
transfer_model = transfer_model.to(device)`

```
#import pretrained resnet50
from torchvision.models import resnet50, ResNet50_Weights
weights = ResNet50_Weights.DEFAULT
rn50 = resnet50(weights=weights)

ctr=0
for param in rn50.parameters():

    #unfreeze the first 40 layers
    if ctr < 40:
        param.requires_grad = True
    else:
        param.requires_grad = False

    ctr +=1
#change the final layer to have enough output classes
rn50.fc = nn.Linear(rn50.fc.in_features, 7)

for param in rn50.fc.parameters():
    param.requires_grad = True

rn50 = rn50.to(device)
```

Downloading: "https://download.pytorch.org/models/resnet152-f82ba261.pth" to /root/.cache/torch/hub/checkpoints/resnet152-f82ba261.pth  
100%|██████████| 230M/230M [00:04<00:00, 57.0MB/s]  
Downloading: "https://download.pytorch.org/models/resnet50-11ad3fa6.pth" to /root/.cache/torch/hub/checkpoints/resnet50-11ad3fa6.pth  
100%|██████████| 97.8M/97.8M [00:01<00:00, 94.1MB/s]

In [ ]: model2=rn50

In [ ]: # Train the model  
lr = 1e-4  
num\_epochs = 20  
optimizer = optim.Adam(model2.parameters(), lr=lr)  
criterion = nn.CrossEntropyLoss() # Multi-classification loss function appl  
scheduler = lr\_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1  
train(model2, train\_loader, val\_loader, criterion, optimizer, num\_epochs=num

---

KeyboardInterrupt Traceback (most recent call last)  
<ipython-input-43-54120b2f4d8f> in <cell line: 7>()  
5 criterion = nn.CrossEntropyLoss() # Multi-classification loss funct  
ion applies softmax  
6 scheduler = lr\_scheduler.ReduceLROnPlateau(optimizer, mode='min',  
factor=0.1, patience=5, eps=1e-4)  
----> 7 train(model2, train\_loader, val\_loader, criterion, optimizer,  
num\_epochs=num\_epochs)

<ipython-input-32-90cac2f76738> in train(model, train\_loader, val\_loader, cr  
iterion, optimizer, num\_epochs, scheduler)
35 model.train() # Set the model to train mode
36
----> 37 for images, labels in train\_loader:
38 # To Enable GPU Usage
39 if use\_cuda and torch.cuda.is\_available():

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py in __
next__(self)
    629                     # TODO(https://github.com/pytorch/pytorch/issues/767
50)
    630                     self._reset() # type: ignore[call-arg]
--> 631             data = self._next_data()
    632             self._num_yielded += 1
    633             if self._dataset_kind == _DatasetKind.Iterable and \

/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py in __n
ext_data(self)
    673     def _next_data(self):
    674         index = self._next_index() # may raise StopIteration
--> 675         data = self._dataset_fetcher.fetch(index) # may raise StopI
teration
    676         if self._pin_memory:
    677             data = _utils.pin_memory.pin_memory(data, self._pin_memo
ry_device)

/usr/local/lib/python3.10/dist-packages/torch/utils/data/_utils/fetch.py in f
etch(self, possibly_batched_index)
    47             if self.auto_collation:
    48                 if hasattr(self.dataset, "__getitem__") and self.datase
t.__getitems__:
--> 49                     data = self.dataset.__getitems__(possibly_batched_i
ndex)
    50             else:
    51                 data = [self.dataset[idx] for idx in
possibly_batched_index]

/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataset.py in __get
items__(self, indices)
    417                 return self.dataset.__getitems__([self.indices[idx] for
idx in indices]) # type: ignore[attr-defined]
    418             else:
--> 419                 return [self.dataset[self.indices[idx]] for idx in
indices]
    420
    421     def __len__(self):


/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataset.py in <list
comp>(.0)
    417                 return self.dataset.__getitems__([self.indices[idx] for
idx in indices]) # type: ignore[attr-defined]
    418             else:
--> 419                 return [self.dataset[self.indices[idx]] for idx in
indices]
    420
    421     def __len__(self):


/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataset.py in __get
item__(self, idx)
    346             else:
    347                 sample_idx = idx - self.cumulative_sizes[dataset_idx - 1
]
--> 348                 return self.datasets[dataset_idx][sample_idx]
    349
    350
```

```
350     @property
<ipython-input-7-1888b5ca2896> in __getitem__(self, index)
    29
    30     if self.transform: #Applies transform onto image
--> 31         img = self.transform(img)
    32
    33     label_one_hot = torch.nn.functional.one_hot(torch.tensor(label),
num_classes=self.num_classes).float() #converts to one hot encoding

/usr/local/lib/python3.10/dist-packages/torchvision/transforms/transforms.py
in __call__(self, img)
    93         def __call__(self, img):
    94             for t in self.transforms:
--> 95                 img = t(img)
    96             return img
    97

/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py in _wrapped_call_(self, *args, **kwargs)
    1530             return self._compiled_call_impl(*args, **kwargs) # type
: ignore[misc]
    1531         else:
-> 1532             return self._call_impl(*args, **kwargs)
    1533
    1534     def _call_impl(self, *args, **kwargs):

/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py in _call_
impl(self, *args, **kwargs)
    1539             or _global_backward_pre_hooks or
_global_backward_hooks
    1540             or _global_forward_hooks or _global_forward_pre_hook
s):
-> 1541             return forward_call(*args, **kwargs)
    1542

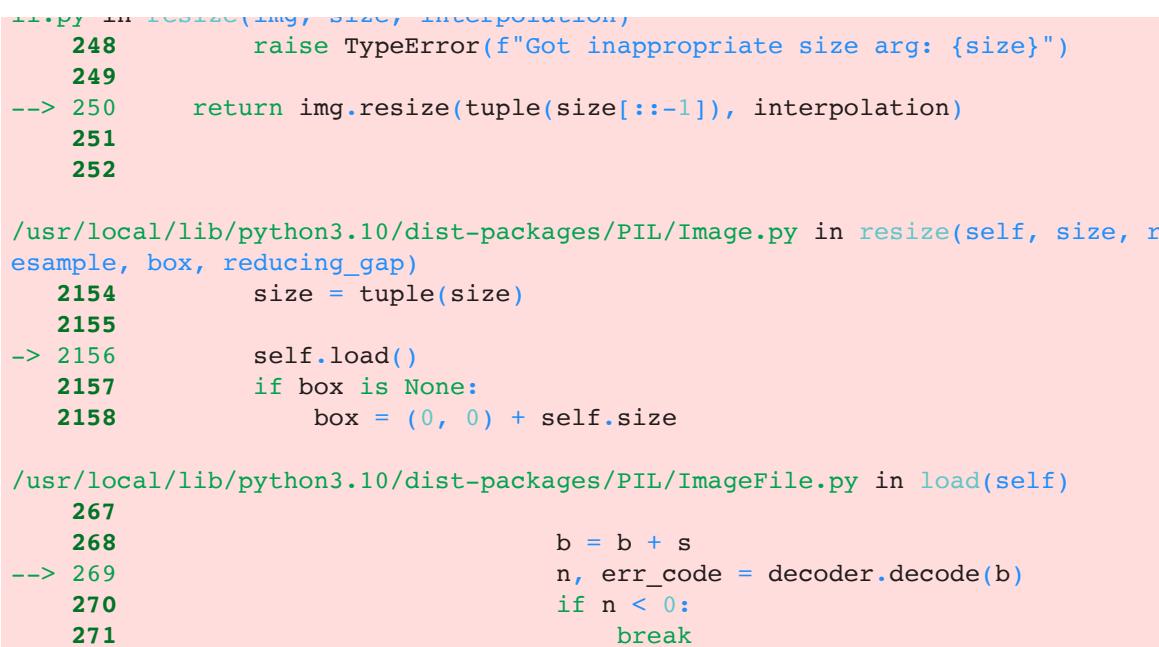
    1543     try:

/usr/local/lib/python3.10/dist-packages/torchvision/transforms/transforms.py
in forward(self, img)
    352             PIL Image or Tensor: Rescaled image.
    353             """
--> 354             return F.resize(img, self.size, self.interpolation, self.max
_size, self.antialias)
    355
    356     def __repr__(self) -> str:

/usr/local/lib/python3.10/dist-packages/torchvision/transforms/functional.py
in resize(img, size, interpolation, max_size, antialias)
    466             warnings.warn("Anti-alias option is always applied for P
IL Image input. Argument antialias is ignored.")
    467             pil_interpolation = pil_modes_mapping[interpolation]
--> 468             return F_pil.resize(img, size=output_size, interpolation=pil
_interpolation)
    469
    470             return F_t.resize(img, size=output_size, interpolation=interpol
ation.value, antialias=antialias)

/usr/local/lib/python3.10/dist-packages/torchvision/transforms/_functional_p
il.py in resize(img, size, interpolation)
```

```


  248         raise TypeError(f"Got inappropriate size arg: {size}")
  249
--> 250     return img.resize(tuple(size[::-1]), interpolation)
  251
  252

/usr/local/lib/python3.10/dist-packages/PIL/Image.py in resize(self, size, resample, box, reducing_gap)
  2154         size = tuple(size)
  2155
-> 2156         self.load()
  2157         if box is None:
  2158             box = (0, 0) + self.size

/usr/local/lib/python3.10/dist-packages/PIL/ImageFile.py in load(self)
  267
  268             b = b + s
--> 269             n, err_code = decoder.decode(b)
  270             if n < 0:
  271                 break

```

`KeyboardInterrupt`:

```

In [ ]: #save the model
torch.save(model2.state_dict(), 'model2.pth')

#For testing the model
test_accuracy = get_accuracy(model2, test_loader, device)
print(f'Test Accuracy: {test_accuracy:.2%}')

```

## Loading/Splitting Full Dataset: (Old way of dataloading)

```

In [ ]: #data loading
X=np.asarray(metadata_balanced['image'].tolist())
X=X/255 #normalize
y=metadata_balanced['label']
#convert y into np array
y=y.to_numpy()

#now into tensor
y=torch.from_numpy(y)
y_catog=F.one_hot(y, num_classes=7)

```

```

In [ ]: x_train, x_test, y_train, y_test = train_test_split(X, y_catog, test_size=0.
#convert xtrain and y train into data sets
x_train=torch.from_numpy(x_train)
x_test=torch.from_numpy(x_test)

```

```

In [ ]: print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

#check type of x and y train

```

```
##check type of x and y train
print(type(x_train))
print(type(y_train))
print(type(x_test))
print(type(y_test))

torch.Size([2867, 128, 128, 3])
torch.Size([717, 128, 128, 3])
torch.Size([2867, 7])
torch.Size([717, 7])
<class 'torch.Tensor'>
<class 'torch.Tensor'>
<class 'torch.Tensor'>
<class 'torch.Tensor'>
```

In [ ]:

```
#check the values in x_train
print(x_train.shape)
#permute
x_train=x_train.permute(0,3,1,2)
x_test=x_test.permute(0,3,1,2)

print(x_train.shape)

torch.Size([2867, 128, 128, 3])
torch.Size([2867, 3, 128, 128])
```

In [ ]:

```
#convert this into dataset

x_train_tensor = x_train.clone().detach()
y_train_tensor = y_train.clone().detach()

x_test_tensor = x_test.clone().detach()
y_test_tensor = y_test.clone().detach()

train_dataset = torch.utils.data.TensorDataset(x_train_tensor.to(device), y_train_tensor)
test_dataset = torch.utils.data.TensorDataset(x_test_tensor.to(device), y_test_tensor)

#split train into train and val (75% of 80 gives total split to 60/20/20)
train_size=int(0.75*len(train_dataset))
val_size=len(train_dataset)-train_size
train_dataset, val_dataset=torch.utils.data.random_split(train_dataset, [train_size, val_size])

#convert into dataloader
bs=128
train_loader=torch.utils.data.DataLoader(train_dataset, batch_size=bs, shuffle=True)
test_loader=torch.utils.data.DataLoader(test_dataset, batch_size=bs, shuffle=False)
val_loader=torch.utils.data.DataLoader(val_dataset, batch_size=bs, shuffle=False)
```

In [ ]:

```
#check the shape of test_dataset
print(len(train_dataset))
print(len(test_dataset))
print(len(val_dataset))
```

```
2150
717
717
```

Check the shape of all the dataloaders

```
In [ ]: # Function to show a tensor image with its label
def show_image(tensor, label=None):
    # Convert tensor to numpy array
    np_img = tensor.numpy()

    # Normalize the image for display
    np_img = (np_img - np_img.min()) / (np_img.max() - np_img.min())
    # Display the image
    plt.imshow(np_img)
    if label is not None:
        plt.title(f"Label: {label}")
    plt.axis('off')  # Hide the axes
    plt.show()
```