Double-click (or enter) to edit

```

```

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

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

```
!rm -r ~/.kaggle
```

```
!mkdir ~/.kaggle
```

```
!mv ./kaggle.json ~/.kaggle/
```

```
mv: cannot stat './kaggle.json': No such file or directory
```

```
!chmod 600 ~/.kaggle/kaggle.json
```

```
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
```

```
!kaggle datasets download -d kabilan03/sign-language-dataset
```

```
Traceback (most recent call last):
  File "/usr/local/bin/kaggle", line 5, in <module>
    from kaggle.cli import main
    ...ne 23, in <module>
    ...xtended.py", line 166, in authenticate
    self.config_file, self.config_dir))
OSError: Could not find kaggle.json. Make sure it's located in /root/.kaggle. Or use the environment method.
```



```
! unzip sign-language-dataset.zip
```

```
Archive:  sign-language-dataset.zip
replace Sign Language Dataset/0/IMG_1118.JPG? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

```
import torch
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torchvision.models as models
import torch.nn as nn
import torch.optim as optim
import numpy as np
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
```

```
# Specify transforms using torchvision.transforms as transforms
# library
transformations = transforms.Compose([
    transforms.Resize(255),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

```
from torchvision.datasets import ImageFolder
from torch.utils.data import Subset
from sklearn.model_selection import train_test_split
from torchvision.transforms import Compose, ToTensor, Resize
from torch.utils.data import DataLoader
```

```
def train_val_dataset(dataset, val_split=0.25):
    train_idx, val_idx = train_test_split(list(range(len(dataset))), test_size=val_split)
    datasets = {}
    datasets['train'] = Subset(dataset, train_idx)
    datasets['val'] = Subset(dataset, val_idx)
    return datasets

dataset = ImageFolder('/content/drive/MyDrive/Lichen Planus Dataset/Lichen Planus binary Dataset', transform=transformations)
print(len(dataset))
```

```
datasets = train_val_dataset(dataset)
print(len(datasets['train']))
print(len(datasets['val']))
```

```
174
130
44
```

```
# # Load in each dataset and apply transformations using
# # the torchvision.datasets as datasets library
# train_set = datasets.ImageFolder("/content/drive/MyDrive/Neural Networks/knife/training_set", transform = transformations)
# val_set = datasets.ImageFolder("/content/drive/MyDrive/Neural Networks/knife/test_set", transform = transformations)
```

```
# Put into a Dataloader using torch library
train_loader = torch.utils.data.DataLoader(datasets['train'], batch_size=32, shuffle=True)
val_loader = torch.utils.data.DataLoader(datasets['val'], batch_size =32, shuffle=True)
```

```
# Get pretrained model using torchvision.models as models library
model = models.densenet161(pretrained=True)
# Turn off training for their parameters
for param in model.parameters():
    param.requires_grad = False
```

```
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated sinc
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None`
  warnings.warn(msg)
```



```
# Create new classifier for model using torch.nn as nn library
classifier = nn.Sequential(nn.Linear(classifier_input, 1024),
                           nn.ReLU(),
                           nn.Linear(1024, 512),
                           nn.ReLU(),
                           nn.Linear(512, num_labels),
                           nn.LogSoftmax(dim=1))
# Replace default classifier with new classifier
model.classifier = classifier
```

```
# Set the error function using torch.nn as nn library
criterion = nn.NLLLoss()
# Set the optimizer function using torch.optim as optim library
optimizer = optim.Adam(model.classifier.parameters())
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

```
            (relu1): ReLU(inplace=True)
            (conv1): Conv2d(2112, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu2): ReLU(inplace=True)
            (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
          (denselayer24): _DenseLayer(
            (norm1): BatchNorm2d(2160, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu1): ReLU(inplace=True)
            (conv1): Conv2d(2160, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu2): ReLU(inplace=True)
            (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
        )
        (norm5): BatchNorm2d(2208, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (classifier): Sequential(
        (0): Linear(in_features=2208, out_features=1024, bias=True)
        (1): ReLU()
        (2): Linear(in_features=1024, out_features=512, bias=True)
        (3): ReLU()
        (4): Linear(in_features=512, out_features=2, bias=True)
        (5): LogSoftmax(dim=1)
      )
    )
```

```
epochs = 5
for epoch in range(epochs):
    train_loss = 0
    val_loss = 0
    accuracy = 0

    # Training the model
    model.train()
```



```
        # Move to device
        inputs, labels = inputs.to(device), labels.to(device)
        # Clear optimizers
        optimizer.zero_grad()
        # Forward pass
        output = model.forward(inputs)
        # Loss
        loss = criterion(output, labels)
        # Calculate gradients (backpropogation)
        loss.backward()
        # Adjust parameters based on gradients
        optimizer.step()
        # Add the loss to the training set's rnning loss
        train_loss += loss.item()*inputs.size(0)

        # Print the progress of our training
        counter += 1
        print(counter, "/", len(train_loader))

    # Evaluating the model
    model.eval()
    counter = 0
    # Tell torch not to calculate gradients
    with torch.no_grad():
        for inputs, labels in val_loader:
            # Move to device
            inputs, labels = inputs.to(device), labels.to(device)
            # Forward pass
            output = model.forward(inputs)
            # Calculate Loss
            valloss = criterion(output, labels)
            # Add loss to the validation set's running loss
            val_loss += valloss.item()*inputs.size(0)

            # Since our model outputs a LogSoftmax, find the real
            # percentages by reversing the log function
            output = torch.exp(output)
            # Get the top class of the output
            top_p, top_class = output.topk(1, dim=1)
            # See how many of the classes were correct?
            equals = top_class == labels.view(*top_class.shape)
            # Calculate the mean (get the accuracy for this batch)
            # and add it to the running accuracy for this epoch
            accuracy += torch.mean(equals.type(torch.FloatTensor)).item()

            # Print the progress of our evaluation
            counter += 1
            print(counter, "/", len(val_loader))
```

```
    # Get the average loss for the entire epoch
    train_loss = train_loss/len(train_loader.dataset)
    valid_loss = val_loss/len(val_loader.dataset)
    # Print out the information
    print('Accuracy: ', accuracy/len(val_loader))
    print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(epoch, train_loss, valid_loss))
```

```
 1 / 5
 2 / 5
 3 / 5
 4 / 5
 5 / 5
 1 / 2
 2 / 2
 Accuracy:  0.34375
 Epoch: 0        Training Loss: 0.965318         Validation Loss: 1.504247
 1 / 5
 2 / 5
 3 / 5
 4 / 5
 5 / 5
 1 / 2
 2 / 2
 Accuracy:  0.7864583432674408
 Epoch: 1        Training Loss: 0.557471         Validation Loss: 0.335267
 1 / 5
 2 / 5
 3 / 5
 4 / 5
 5 / 5
 1 / 2
 2 / 2
 Accuracy:  0.96875
```



```
 3 / 5
 4 / 5
 5 / 5
 1 / 2
 2 / 2
 Accuracy:  0.984375
 Epoch: 3        Training Loss: 0.213716         Validation Loss: 0.119651
 1 / 5
 2 / 5
 3 / 5
 4 / 5
 5 / 5
 1 / 2
 2 / 2
 Accuracy:  0.96875
 Epoch: 4        Training Loss: 0.126533         Validation Loss: 0.083813
```

```
    # Get the average loss for the entire epoch
    train_loss = train_loss/len(train_loader.dataset)
    valid_loss = val_loss/len(val_loader.dataset)
    # Print out the information
    print('Accuracy: ', accuracy/len(val_loader))
    print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(epoch, train_loss, valid_loss))
```

```
 Accuracy:  0.96875
 Epoch: 4        Training Loss: 0.000973         Validation Loss: 0.083813
```

```
model.eval()
```

```
        )
        (denselayer22): _DenseLayer(
          (norm1): BatchNorm2d(2064, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu1): ReLU(inplace=True)
          (conv1): Conv2d(2064, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu2): ReLU(inplace=True)
          (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        )
        (denselayer23): _DenseLayer(
          (norm1): BatchNorm2d(2112, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu1): ReLU(inplace=True)
          (conv1): Conv2d(2112, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu2): ReLU(inplace=True)
          (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        )
        (denselayer24): _DenseLayer(
          (norm1): BatchNorm2d(2160, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu1): ReLU(inplace=True)
          (conv1): Conv2d(2160, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu2): ReLU(inplace=True)
          (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        )
      )
      (norm5): BatchNorm2d(2208, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (classifier): Sequential(
      (0): Linear(in_features=2208, out_features=1024, bias=True)
      (1): ReLU()
      (2): Linear(in_features=1024, out_features=512, bias=True)
      (3): ReLU()
      (4): Linear(in_features=512, out_features=2, bias=True)
      (5): LogSoftmax(dim=1)
```



```
# Process our image
def process_image(image_path):
    # Load Image
    img = Image.open(image_path)

    # Get the dimensions of the image
    width, height = img.size

    # Resize by keeping the aspect ratio, but changing the dimension
    # so the shortest size is 255px
    img = img.resize((255, int(255*(height/width))) if width < height else (int(255*(width/height)), 255))

    # Get the dimensions of the new image size
    width, height = img.size

    # Set the coordinates to do a center crop of 224 x 224
    left = (width - 224)/2
    top = (height - 224)/2
    right = (width + 224)/2
    bottom = (height + 224)/2
    img = img.crop((left, top, right, bottom))

    # Turn image into numpy array
    img = np.array(img)

    # Make the color channel dimension first instead of last
    img = img.transpose((2, 0, 1))

    # Make all values between 0 and 1
    img = img/255

    # Normalize based on the preset mean and standard deviation
    img[0] = (img[0] - 0.485)/0.229
    img[1] = (img[1] - 0.456)/0.224
    img[2] = (img[2] - 0.406)/0.225

    # Add a fourth dimension to the beginning to indicate batch size
    img = img[np.newaxis,:]

    # Turn into a torch tensor
    image = torch.from_numpy(img)
    image = image.float()
    return image
```

```
# Using our model to predict the label
def predict(image, model):
    # Pass the image through our model
```

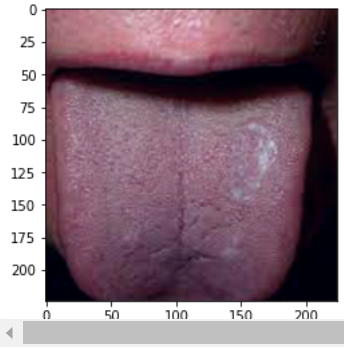
```
    output = model.forward(image)

    # Reverse the log function in our output
    output = torch.exp(output)

    # Get the top predicted class, and the output percentage for
    # that class
    probs, classes = output.topk(1, dim=1)
    return probs.item(), classes.item()
```

```
# Show Image
def show_image(image):
    # Convert image to numpy
    image = image.numpy()

    # Un-normalize the image
    image[0] = image[0] * 0.226 + 0.445

    # Print the image
    fig = plt.figure(figsize=(25, 4))
    plt.imshow(np.transpose(image[0], (1, 2, 0)))
```

```
# Process Image
image = process_image("/content/drive/MyDrive/Lichen Planus Dataset/Lichen Planus binary Dataset/Oral lichen planus/1.jpg")
# Give image to model to predict output
imc = image.cpu()
top_prob, top_class = predict(imc, model)
# Show the image
show_image(image)
# Print the results
print("The model is ", top_prob*100, "% certain that the image has a predicted class of ", top_class  )
```

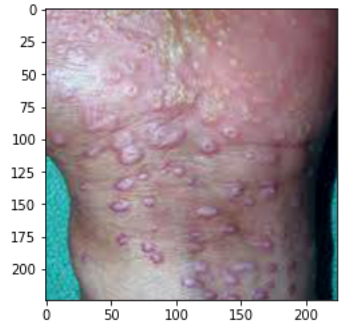


```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB
The model is  69.11953687667847 % certain that the image has a predicted class of  1
```




```
# Process Image
image = process_image("/content/drive/MyDrive/Lichen Planus Dataset/Lichen Planus binary Dataset/Cutaneous lichen planus/11.jpg")
# Give image to model to predict output
imc = image.cpu()
top_prob, top_class = predict(imc, model)
# Show the image
show_image(image)
# Print the results
print("The model is ", top_prob*100, "% certain that the image has a predicted class of ", top_class  )
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for fl
The model is  99.65928196907043 % certain that the image has a predicted class of  0
```



✓ 0s completed at 16:54

Automatic saving failed. This file was updated remotely or in another tab. Show diff