

```
In [2]: #chect CT images classification
        #https://www.kaggle.com/code/zahraahesham/chect-ct-images-classification/notebook
```

This dataset already split, so we don't need to split it again

```
In [3]: import os
import numpy as np
import torch
from torchvision import datasets, transforms as T, models
from torch import nn, optim
from torch.utils.data import DataLoader
import warnings
#warnings.filterwarnings( "ignore")
from tqdm import tqdm
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
from sklearn.metrics import auc

# Image Count by Folder:
# -----
# Train      :    613 images
# Test       :    315 images
# Valid      :     72 images
# -----
# TOTAL      :   1000 images
```

```
In [4]: #Getting name of classes in dataset
data_dir = 'ChestCTDataset'
def get_classes(data_dir):
    all_data = datasets.ImageFolder(data_dir)
    return all_data.classes
```

```
In [5]: classes = get_classes(data_dir + "/test" )
print(classes, len(classes))
```

```
['adenocarcinoma', 'large.cell.carcinoma', 'normal', 'squamous.cell.carcinoma'] 4
```

```
In [6]: #Image augmentation and preprocessing

def get_data_loaders(data_dir, batch_size, train =False):
    """Prepare data loaders for training, validation, and testing."""
    if train:
        # Data augmentation for training
        transform = T.Compose([
            T.RandomHorizontalFlip(),
            T.RandomVerticalFlip(),
```

```

        T.RandomRotation( 10),
        T.Resize(( 299, 299)),
        T.ToTensor(),
        T.Normalize(mean =[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])
    train_data = datasets.ImageFolder(os.path.join(data_dir, "train/"), transform=transform)
    train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True, num_workers=4)
    return train_loader, len(train_data)
else:
    # Validation and test data transformations
    transform = T.Compose([
        T.Resize(( 299, 299)),
        T.ToTensor(),
        T.Normalize(mean =[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])
    val_data = datasets.ImageFolder(os.path.join(data_dir, "valid/"), transform=transform)
    test_data = datasets.ImageFolder(os.path.join(data_dir, "test/"), transform=transform)
    val_loader = DataLoader(val_data, batch_size=batch_size, shuffle=False, num_workers=4)
    test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False, num_workers=4)
    return val_loader, test_loader, len(val_data), len(test_data)

```

In [7]: *# Dataset loaders*

```

batch_size = 32

(train_loader, train_data_len) = get_data_loaders(data_dir, batch_size, train=True)
(val_loader, test_loader, valid_data_len, test_data_len) = get_data_loaders(data_dir, batch_size, train=False)

classes = get_classes(data_dir + "/train")
num_classes = len(classes)

dataloaders = {"train": train_loader, "val": val_loader}
dataset_sizes = {"train": train_data_len, "val": valid_data_len}

# Device setup
device = torch.device( 'cuda' if torch.cuda.is_available() else 'cpu')

```

In [8]: *# efficientnet_b0*

```

# Load the pre-trained Inception-v3 model
model = models.efficientnet_b0(pretrained=False)

# Update the fully connected layer for the number of classes
model.classifier[1] = nn.Linear(model.classifier[1].in_features, num_classes)
model = model.to(device)

# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)

# Learning rate scheduler
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)

```

```
C:\Users\hm\anaconda3\envs\image\Lib\site-packages\torchvision\models\_utils.py:208:
UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed
in the future, please use 'weights' instead.
  warnings.warn(
C:\Users\hm\anaconda3\envs\image\Lib\site-packages\torchvision\models\_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecate
d since 0.13 and may be removed in the future. The current behavior is equivalent t
o passing `weights=None`.
  warnings.warn(msg)
```

```
In [9]: def train_model(model, criterion, optimizer, scheduler, num_epochs=20):
        """Train and validate the model."""

        for epoch in range(num_epochs):

            print(f"Epoch {epoch+1}/{num_epochs}")
            print('-' * 10)

            # Each epoch has a training and validation phase
            for phase in ['train', 'val']:
                if phase == 'train':
                    model.train() # Set model to training mode
                else:
                    model.eval() # Set model to evaluation mode
                running_loss = 0.0
                running_corrects = 0

                # Iterate over data
                for inputs, labels in dataloaders[phase]:
                    inputs = inputs.to(device)
                    labels = labels.to(device)

                    # Zero the parameter gradients
                    optimizer.zero_grad()

                    # Forward pass
                    with torch.set_grad_enabled(phase == 'train'):
                        outputs = model(inputs)

                    # Handle auxiliary logits for Inception-v3
                    if isinstance(outputs, tuple) or hasattr(outputs, "logits"):
                        main_outputs = outputs.logits if hasattr(outputs, "logits")
                        aux_outputs = outputs.aux_logits if hasattr(outputs, "aux_lo
                        loss = criterion(main_outputs, labels)
                        if aux_outputs is not None: # Add auxiliary loss if presen
                            loss += 0.4 * criterion(aux_outputs, labels)
                    else:
                        main_outputs = outputs
                        loss = criterion(main_outputs, labels)
                    _, preds = torch.max(main_outputs, 1) # Extract predictions fr
                    # Backward pass and optimization in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # Statistics
                running_loss += loss.item() * inputs.size(0)
```

```

        running_corrects += torch.sum(preds == labels.data)
    # Step the scheduler in the training phase
    if phase == 'train':
        scheduler.step()
    epoch_loss = running_loss / dataset_sizes[phase]
    epoch_acc = running_corrects.double() / dataset_sizes[phase]
    print(f"{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}")

    return model

```

```

In [10]: # Train the model
         model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=2)

```

```

Epoch 1/2
-----
train Loss: 1.1482 Acc: 0.4829
val Loss: 1.4334 Acc: 0.3194
Epoch 2/2
-----
train Loss: 1.0198 Acc: 0.4845
val Loss: 1.5237 Acc: 0.3194

```

```

In [11]: def evaluate_model(model_name, y_true, y_pred, classes):

    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)

    print("Accuracy:", accuracy)
    print("Precision :", precision)
    print("Recall:", recall)
    print("F1-score:" , f1)

    print("Classification Report:" )
    print(classification_report(y_true, y_pred))

    # Plot confusion matrix
    plt.figure(figsize =(6, 5))
    sns.heatmap(cm, annot =True, fmt='d', cmap='Blues', xticklabels=classes, ytickl
    plt.xlabel( "Predicted Label" )
    plt.ylabel( "True Label")
    plt.title( "Confusion Matrix" )
    plt.show()

```

```

In [12]: y_true = []
         y_proba = []
         y_pred = []
         # Disable gradient calculation
         with torch.no_grad():
             for x, y in tqdm(test_loader):
                 # Forward pass
                 output = model(x.to(device))

                 # Store true labels

```

```

y_true.extend(y.numpy())

# Get predicted probabilities (softmax output)
probabilities = torch.softmax(output, dim =1) # Assuming it's a multi-class
y_proba.extend(probabilities.cpu().numpy()) # Store probabilities

# Get predicted classes (argmax for final class prediction)
pred = torch.argmax(probabilities, axis =1).cpu().numpy()
y_pred.extend(pred)
# Convert to numpy arrays for further processing
y_true = np.array(y_true)
y_proba = np.array(y_proba)
y_pred = np.array(y_pred)

evaluate_model( "efficientnet_b0" , y_true, y_pred, classes)

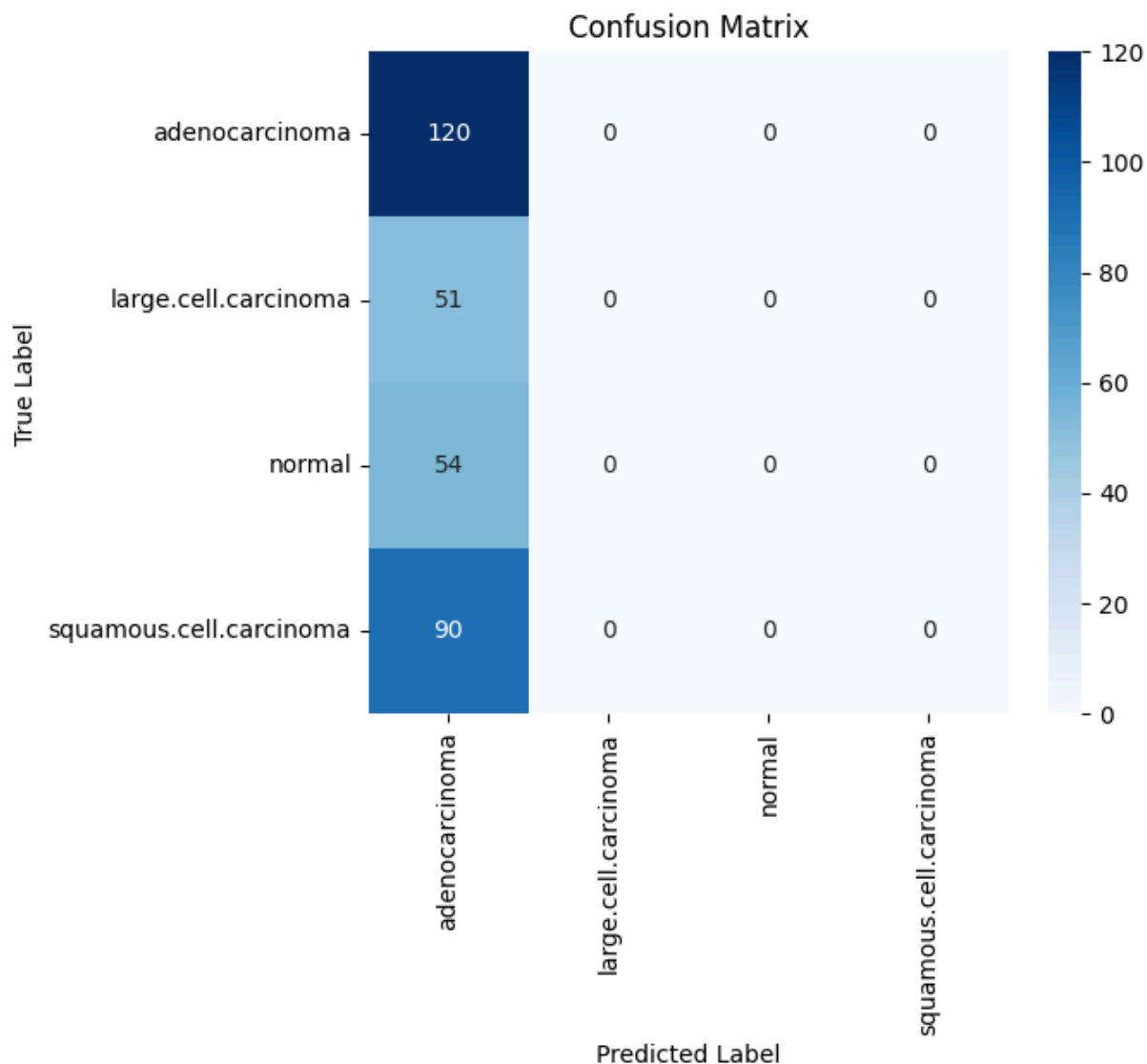
```

```

100%|██████████| 10/10 [00:34<00:00, 3.46s/it]
C:\Users\hm\anaconda3\envs\image\Lib\site-packages\sklearn\metrics\_classification.p
y:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this behavio
r.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\hm\anaconda3\envs\image\Lib\site-packages\sklearn\metrics\_classification.p
y:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this behavio
r.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\hm\anaconda3\envs\image\Lib\site-packages\sklearn\metrics\_classification.p
y:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this behavio
r.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\hm\anaconda3\envs\image\Lib\site-packages\sklearn\metrics\_classification.p
y:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this behavio
r.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
Accuracy: 0.38095238095238093
Precision : 0.14512471655328796
Recall: 0.38095238095238093
F1-score: 0.21018062397372744
Classification Report:

```

	precision	recall	f1-score	support
0	0.38	1.00	0.55	120
1	0.00	0.00	0.00	51
2	0.00	0.00	0.00	54
3	0.00	0.00	0.00	90
accuracy			0.38	315
macro avg	0.10	0.25	0.14	315
weighted avg	0.15	0.38	0.21	315



```
In [19]: # Resnet18
from torchvision import models
import torch.nn as nn
import torch.optim as optim
# Initialize ResNet-18 from scratch (no pretrained weights)
model = models.resnet18(pretrained=False)
# Replace the final fully connected layer to match the number of
# output classes
model.fc = nn.Linear(model.fc.in_features, num_classes)
model = model.to(device)
# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
# Learning rate scheduler
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)
model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=2)
```

```
C:\Users\hm\anaconda3\envs\image\Lib\site-packages\torchvision\models\_utils.py:208:
UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed
in the future, please use 'weights' instead.
  warnings.warn(
C:\Users\hm\anaconda3\envs\image\Lib\site-packages\torchvision\models\_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecate
d since 0.13 and may be removed in the future. The current behavior is equivalent t
o passing `weights=None`.
  warnings.warn(msg)
```

Epoch 1/2

train Loss: 1.0790 Acc: 0.5057

val Loss: 7.8742 Acc: 0.1944

Epoch 2/2

train Loss: 0.9563 Acc: 0.5644

val Loss: 1.7842 Acc: 0.3472

```
In [21]: y_true = []
y_proba = []
y_pred = []
# Disable gradient calculation
with torch.no_grad():
    for x, y in tqdm(test_loader):
        # Forward pass
        output = model(x.to(device))

        # Store true Labels
        y_true.extend(y.numpy())

        # Get predicted probabilities (softmax output)
        probabilities = torch.softmax(output, dim =1) # Assuming it's a multi-clas
y_proba.extend(probabilities.cpu().numpy()) # Store probabilities

        # Get predicted classes (argmax for final class prediction)
        pred = torch.argmax(probabilities, axis =1).cpu().numpy()
        y_pred.extend(pred)
# Convert to numpy arrays for further processing
y_true = np.array(y_true)
y_proba = np.array(y_proba)
y_pred = np.array(y_pred)
evaluate_model( "Resnet18", y_true, y_pred, classes)
```

100%|██████████| 10/10 [00:34<00:00, 3.49s/it]

Accuracy: 0.17142857142857143
Precision : 0.029481346678798908
Recall: 0.17142857142857143
F1-score: 0.05031055900621118
Classification Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	120
1	0.00	0.00	0.00	51
2	0.17	1.00	0.29	54
3	0.00	0.00	0.00	90
accuracy			0.17	315
macro avg	0.04	0.25	0.07	315
weighted avg	0.03	0.17	0.05	315

```
C:\Users\hm\anaconda3\envs\image\Lib\site-packages\sklearn\metrics\_classification.p
y:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this behavio
r.
```

```
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
```

```
C:\Users\hm\anaconda3\envs\image\Lib\site-packages\sklearn\metrics\_classification.p
y:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this behavio
r.
```

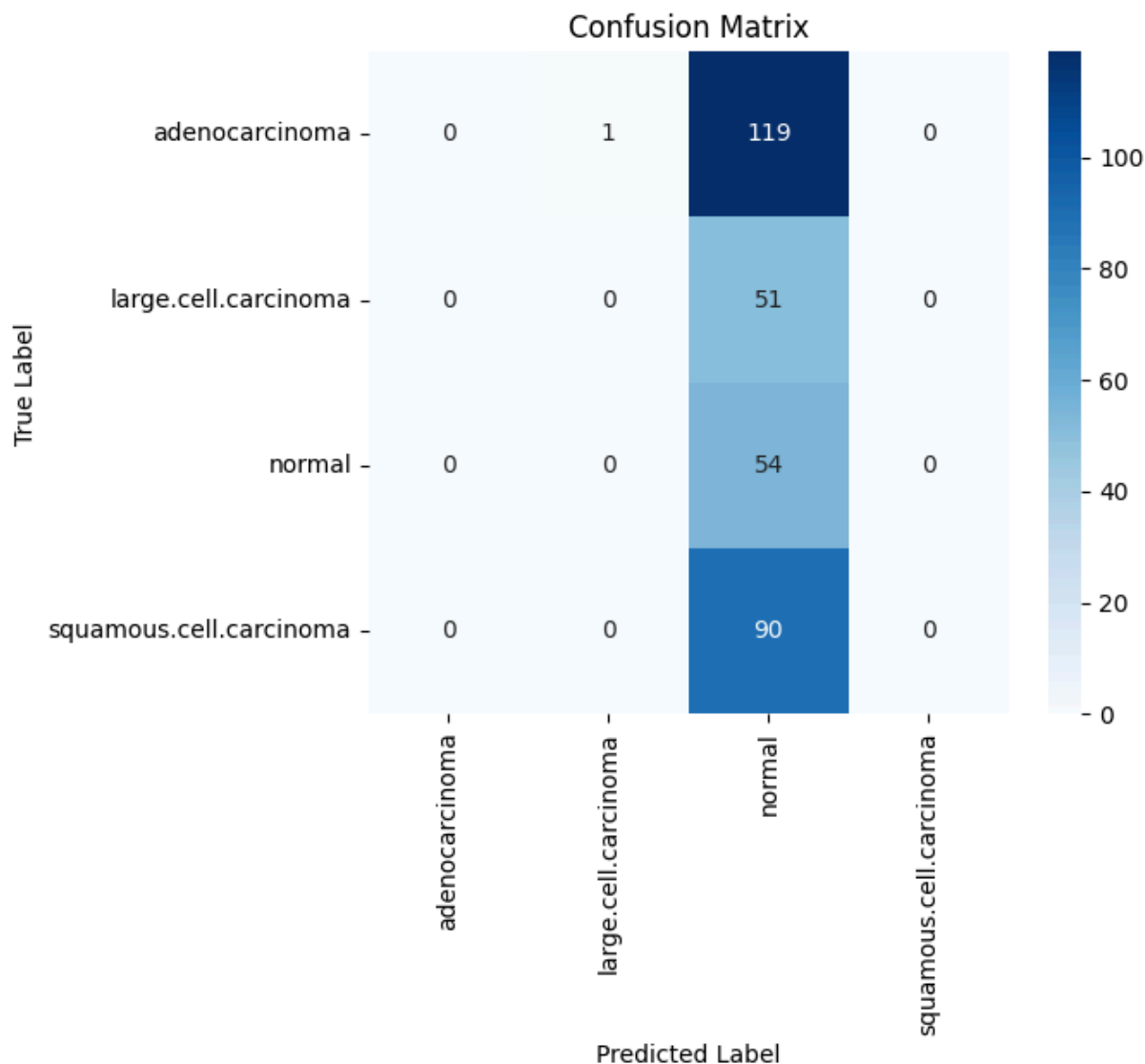
```
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
```

```
C:\Users\hm\anaconda3\envs\image\Lib\site-packages\sklearn\metrics\_classification.p
y:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this behavio
r.
```

```
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
```

```
C:\Users\hm\anaconda3\envs\image\Lib\site-packages\sklearn\metrics\_classification.p
y:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this behavio
r.
```

```
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
```



```
In [22]: # Densenet121
model = models.densenet121(pretrained =False)
# Replace the classifier to match the number of output classes
model.classifier = nn.Linear(model.classifier.in_features, num_classes)
model = model.to(device)
# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = optim.AdamW(model.parameters(), lr =0.001)
# Learning rate scheduler
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size =3, gamma=0.97)
model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=1)

y_true = []
y_proba = []
y_pred = []
# Disable gradient calculation
with torch.no_grad():
    for x, y in tqdm(test_loader):
        # Forward pass
        output = model(x.to(device))
```

```

# Store true labels
y_true.extend(y.numpy())

# Get predicted probabilities (softmax output)
probabilities = torch.softmax(output, dim =1) # Assuming it's a multi-class
y_proba.extend(probabilities.cpu().numpy()) # Store probabilities

# Get predicted classes (argmax for final class prediction)
pred = torch.argmax(probabilities, axis =1).cpu().numpy()
y_pred.extend(pred)
# Convert to numpy arrays for further processing
y_true = np.array(y_true)
y_pred = np.array(y_pred)
evaluate_model( "Densenet121", y_true, y_pred, classes)

```

C:\Users\hm\anaconda3\envs\image\Lib\site-packages\torchvision\models_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.

```

warnings.warn(
C:\Users\hm\anaconda3\envs\image\Lib\site-packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing `weights=None`.
warnings.warn(msg)

```

Epoch 1/1

train Loss: 1.0310 Acc: 0.4992

val Loss: 2.8004 Acc: 0.2639

100%|██████████| 10/10 [01:06<00:00, 6.60s/it]

C:\Users\hm\anaconda3\envs\image\Lib\site-packages\sklearn\metrics_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

```

_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])

```

C:\Users\hm\anaconda3\envs\image\Lib\site-packages\sklearn\metrics_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

```

_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])

```

C:\Users\hm\anaconda3\envs\image\Lib\site-packages\sklearn\metrics_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

```

_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])

```

C:\Users\hm\anaconda3\envs\image\Lib\site-packages\sklearn\metrics_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

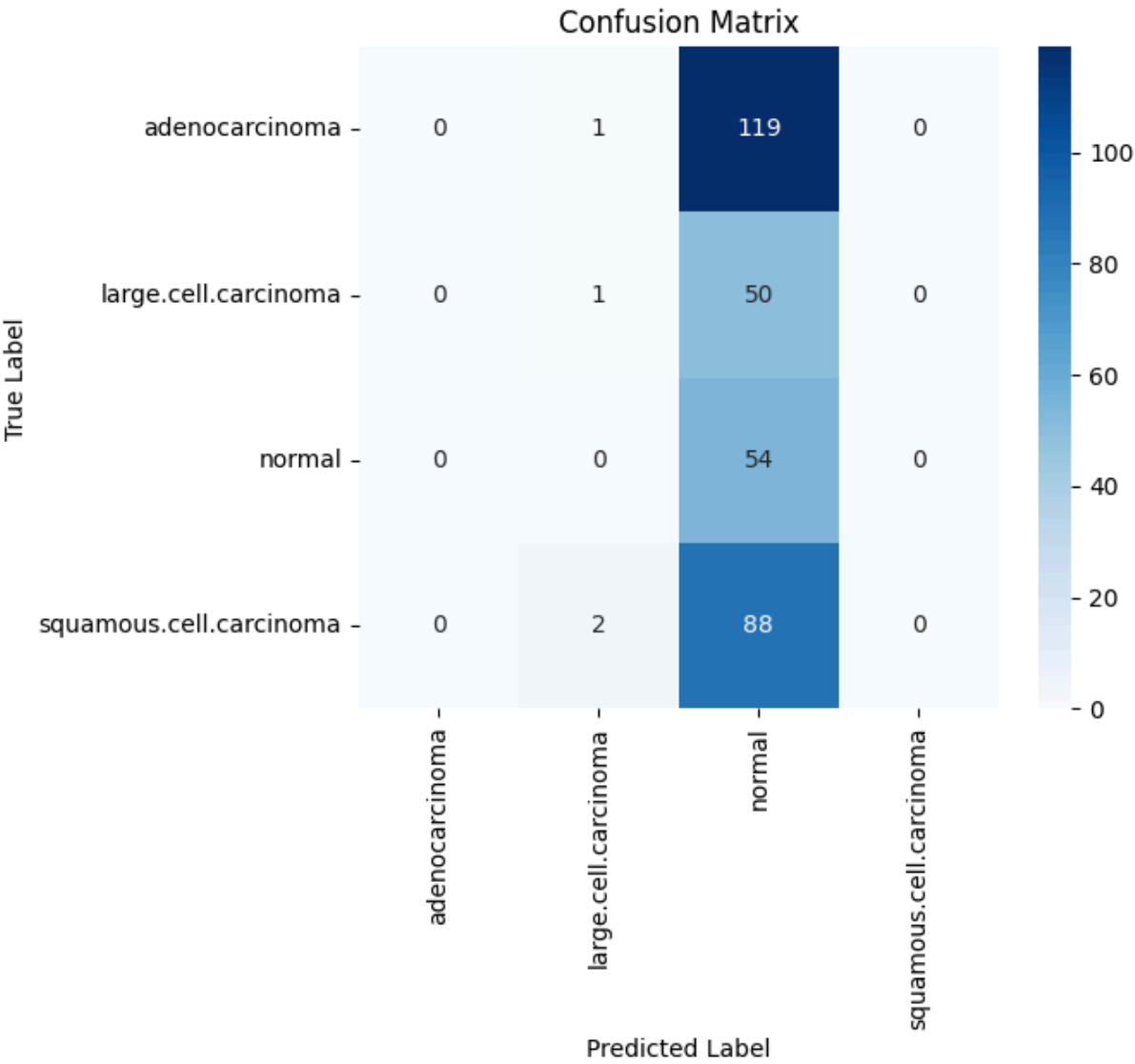
```

_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])

```

Accuracy: 0.1746031746031746
Precision : 0.07024192313581382
Recall: 0.1746031746031746
F1-score: 0.05661151633754374
Classification Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	120
1	0.25	0.02	0.04	51
2	0.17	1.00	0.30	54
3	0.00	0.00	0.00	90
accuracy			0.17	315
macro avg	0.11	0.25	0.08	315
weighted avg	0.07	0.17	0.06	315



In []:

In []:

In []:

In []:

In []: