MIT CSAIL

6.819/6.869 Advances in Computer Vision

Spring 2022

Problem Set 6

Jiahui Tang
jiahuita@mit.edu

Problem 1 *:Train an image classifier model*

Accuracy:

1. Train Top-1 Accuracy: 0.9765
2. Val Top-1 Accuracy: 0.9571
3. Test Top-1 Accuracy: 0.9281

Design Choices:

1. Model: ResNet-50
2. Epoch: 7
3. Early Stopping: True
4. Learning Rate: 0.001
5. Pretrained Model in in Imagenet-1k
6. SGD Optimizer
7. Data Augmentation by random horizontal flip of 0.3

Performance Impact:

1. Model: Most models perform well, Alexnet perform slightly worse than the rest
2. Epoch: Determines the epoches and time it trains, the larger may be better but it could also stablize and stuck in local optima in last few epoch, wasting training time
3. Early Stopping: Early stop and save time if there's no significant improvements
4. Learning Rate: If too small, it would be too slow to learn in each step of gradient descent, making it hard to converge; if too large, it could jump around stationary point and result in oscillation and unstable perform, and also never converge to optima. 0.001 is just right.

5. Pretrained Model in in Imagenet-1k: With pretrained weight, we finetune on current dataset, it is more likely to reach higher accuracy within short amount of time and few number of epochs

6. SGD Optimizer: This SGD optimizer is efficient to converge and works well in this model to find optimal solution

7. Data Augmentation by random horizontal flip of 0.3: augmenting training data could make the training data more robust, covering some out of distribution observations and data points.

Codes for data augmentation:

```
def get_dataloaders(dataset_dir, input_size, batch_size, shuffle = True,
                    transform=get_image_transforms()):
    transform_train = transforms.Compose([transform,
                                          transforms.RandomHorizontalFlip(0.2)
                                          ])

    data_transforms = {
        'train': transform_train,
        'val': transform,
        'test': transform
    }
    # Create training, validation and test datasets
    image_datasets = {x: datasets.ImageFolder(os.path.join(dataset_dir, x),
                            data_transforms[x]) for x in data_transforms.keys()}
    # Create training, validation and test dataloaders
    # Never shuffle the test set
    dataloaders_dict = {x: torch.utils.data.DataLoader(image_datasets[x],
                        batch_size=batch_size, shuffle=False if x != 'train' else shuffle,
                        num_workers=4) for x in data_transforms.keys()}
    return dataloaders_dict
```

Codes for design choices:

```
# Models to choose from [resnet, alexnet, vgg, squeezenet, densenet]
# You can add your own, or modify these however you wish!
model_name = 'resnet50'

# Number of classes in the dataset, normal, benign, malignant
num_classes = 3

# Batch size for training (change depending on how much memory you have)
batch_size = 32

# Shuffle the input data?
shuffle_datasets = True
```

```
# Number of epochs to train for
num_epochs = 7

# Learning rate
learning_rate = 0.001
### IO
# Path to a model file to use to start weights at
resume_from = None

# Whether to use a pretrained model, trained for classification in Imagenet-1k
pretrained = True

# Save all epochs so that you can select the model from a particular epoch
save_all_epochs = False

# Whether to use early stopping (load the model with best accuracy), or not
early_stopping = True

# Directory to save weights to
save_dir = models_dir + '/trained_model_1'
os.makedirs(save_dir, exist_ok=True)
```

Problem 2 *:Metrics*

(a) **Confusion Metrics**

The plot for confusion matrix looks like below:

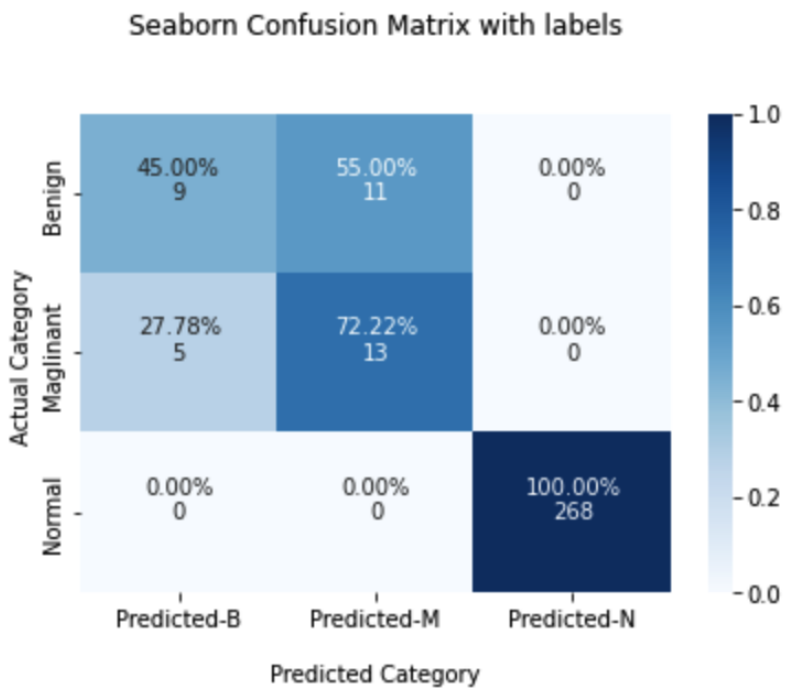

Figure 1: Confusion Matrix

Codes as below:

```
def confusion_matrix_manual(y_label, y_pred, num_classes = 3):
    y_label_flatten = np.array(y_label).flatten()
    y_pred_flatten = np.array(y_pred).flatten()
    result = np.zeros((num_classes, num_classes)).astype(int)
    for y, p in zip(y_label_flatten, y_pred_flatten):
        result[y][p] += 1
    return np.array(result)

print(confusion_matrix(y_label, y_pred))

### Complete your code here to plot your confusion matrix
### please plot it without using metrics packages here for better understanding

def plot_confusion_matrix(y_label, y_pred, title='Confusion matrix'):

    #Get the confusion matrix
    cf_matrix = confusion_matrix_manual(y_label, y_pred)
    cf_matrix_percentage = cf_matrix/np.sum(cf_matrix, axis=1, keepdims=True)

    group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
```

```
    group_percentages = ["{0:.2%}".format(value) for value in
                         (cf_matrix/np.sum(cf_matrix, axis=1,keepdims=True)).flatten()]

    labels = [f"{v1}\n{v2}\n" for v1, v2 in
                    zip(group_percentages, group_counts)]

    labels = np.asarray(labels).reshape(3,3)

    ax = sns.heatmap(cf_matrix_percentage, annot=labels, fmt='', cmap='Blues')

    ax.set_title('Seaborn Confusion Matrix with labels\n\n');
    ax.set_xlabel('\nPredicted Category')
    ax.set_ylabel('Actual Category ');

    ## Ticket labels - List must be in alphabetical order
    ax.xaxis.set_ticklabels(['Predicted-B','Predicted-M', 'Predicted-N'])
    ax.yaxis.set_ticklabels(['Benign','Maglinant', 'Normal'])

    ## Display the visualization of the Confusion Matrix.
    plt.show()

plot_confusion_matrix(y_label, y_pred)
```

(b)**AUPRC Metrics**

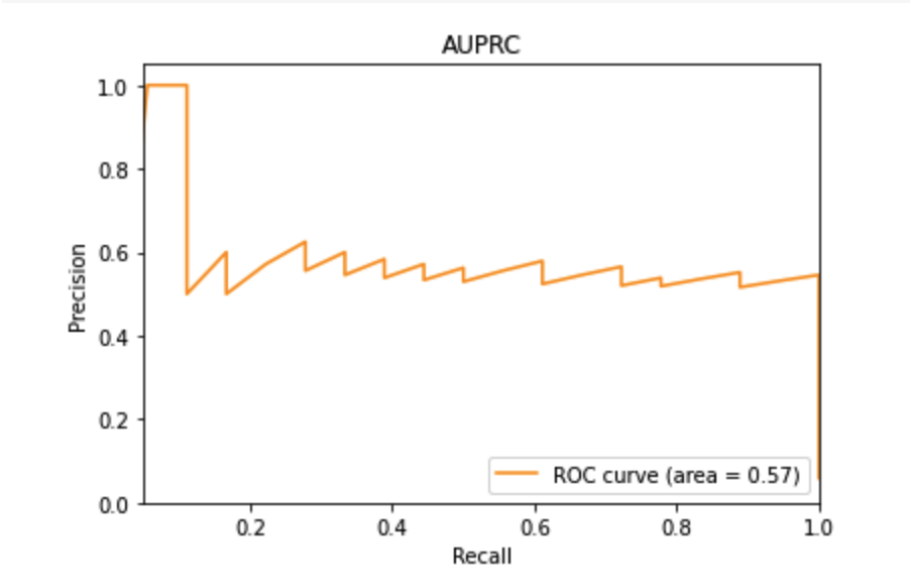

Figure 2: AUPRC Plot

Code as below:

```
def binary_label(prev_label):
```

```
    return np.array([i if i==1 else np.array([0]) for i in prev_label])

y_label_binary = binary_label(y_label).astype(int).flatten()

def plot_auprc():
    recall, precision = [], []

    for threshold in np.arange(0, 1, 1e-6):
        y_pred_binary = (outputs[:,1] >= threshold).astype(int).flatten()
        tp = ((y_label_binary==1)&(y_pred_binary==1)).sum()
        fn = ((y_label_binary==1)&(y_pred_binary==0)).sum()
        fp = ((y_label_binary==0)&(y_pred_binary==1)).sum()
        recall.append(tp/(tp+fn))
        p = tp/(tp+fp) if tp+fp>0 else 0
        precision.append(p)

    auc = 0
    for i in range(len(recall)-1):
        width = np.abs(recall[i+1] - recall[i])
        height = (precision[i] + precision[i+1])/2
        auc += width * height

    plt.figure()
    plt.plot(
        recall,
        precision,
        color="darkorange",
        label="ROC curve (area = %0.2f)" % auc,
    )
    plt.xlim([0.05, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.title("AUPRC")
    plt.legend(loc="lower right")
    plt.show()


plot_auprc()
```

The area under curve is around 0.57, for verification, the result given by calling

```
    metrics.precision_recall_curve()
```

is 0.58, which is quite close.

(c)**Fairness Metrics**

i)

When evaluated using fairness testset, the vgg model 2 has higher accuracy, which is 68.29%, comparing to vgg model 1 with accuracy of 64.88%.

Code and result shown as below:

```
### TODO 3
def accuracy(y_label, y_pred):
    return (y_label==y_pred).mean()

print(f"vgg model 1 prediction accuracy: {accuracy(y_label, y_pred_f1):.2%}")
print(f"vgg model 2 prediction accuracy: {accuracy(y_label, y_pred_f2):.2%}")
```

```
vgg model 1 prediction accuracy: 64.88%
vgg model 2 prediction accuracy: 68.29%
```

Figure 3: Accuracy

ii)

Code and result shown as below:

```
def equalized_odds(mat):
    #print(mat)
    tn, fp, fn, tp = mat.flatten()
    return tp/ (tp + fn)

def get_senior_group(y_label, y_pred, age_list):
    young_y_label, young_y_pred = [], []
    old_y_label, old_y_pred = [], []
    for i in range(len(y_label)):
        age = age_list[i]
        if age >= 60 and age < 70:
            young_y_label.append(y_label[i])
            young_y_pred.append(y_pred[i])
        elif age >= 70:
            old_y_label.append(y_label[i])
            old_y_pred.append(y_pred[i])

    young_y_label, young_y_pred, old_y_label, old_y_pred =
        np.array(young_y_label).flatten(),
        np.array(young_y_pred).flatten(),
        np.array(old_y_label).flatten(),
        np.array(old_y_pred).flatten()

    mat_young, mat_old =
```

```
        confusion_matrix_manual(young_y_label, young_y_pred, num_classes = 2),
        confusion_matrix_manual(old_y_label, old_y_pred, num_classes = 2)
    eo_young, eo_old = equalized_odds(mat_young), equalized_odds(mat_old)
    return eo_young, eo_old

y_pred_f1_binary, y_pred_f2_binary =
        binary_label(y_pred_f1).astype(int).flatten(),
        binary_label(y_pred_f2).astype(int).flatten()
y_label_binary = binary_label(y_label)

eo_young1, eo_old1  = get_senior_group(y_label_binary, y_pred_f1_binary, age_list)
eo_young2, eo_old2  = get_senior_group(y_label_binary, y_pred_f2_binary, age_list)

print("vgg1", eo_young1, eo_old1)
print("vgg2", eo_young2, eo_old2)

eo_lst = [eo_young1, eo_old1, eo_young2, eo_old2]

col = [(i, j) for i in ['fairness_vgg_1', 'fairness_vgg_2']
              for j in ['Age[60, 70)', 'Age[70, max)']]
col = [[i[0] for i in col], [i[1] for i in col]]

result = pd.DataFrame(np.array(eo_lst).reshape(1, -1),
                    columns = col, index=['equalized odds'])
result
```

	fairness_vgg_1		fairness_vgg_2	
	Age[60, 70)	Age[70, max)	Age[60, 70)	Age[70, max)
equalized odds	0.625	0.684783	0.613636	0.608696

Figure 4: Fairness Metrics Table

And the plot for age distribution for the malignant class used in training both models is below.

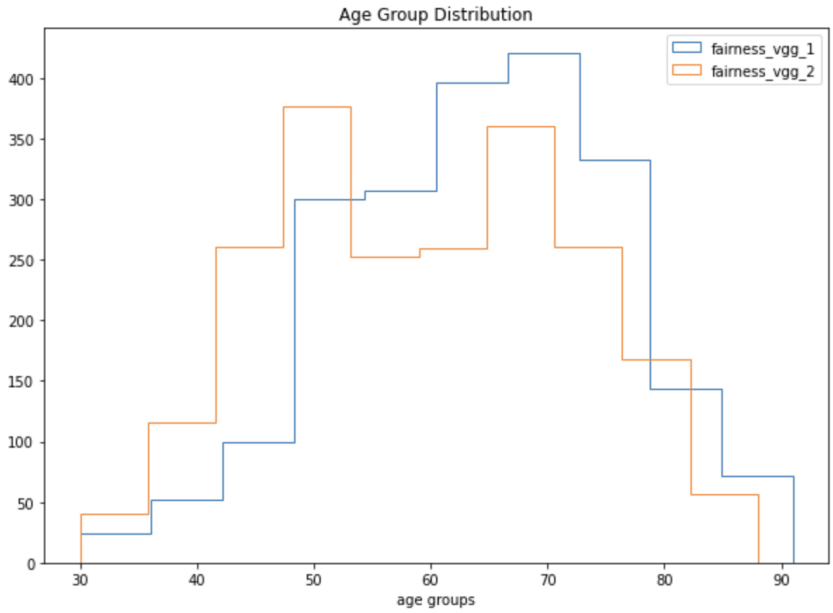

Figure 5: Age Distribution

In order for fairness to be achieved, equalized odds, or sensitivity in the sub groups should be close to each other. Thus, comparing two groups, model 2 is more fair, the equalized odds are close to each other. Though model 1 has higher recall in both groups.

Consider individual fairness, higher recall should be preferred, as we want less FN in each group. FN could be severe for an individual who actually has disease but were not detected. Thus, if I would like to achieve individual fairness, I would choose model 1.

Consider group fairness, to achieve fair equalized odds in each group, without bias towards one group or other, then choose model 2 would be better given the similar sensitivity.

Also, based on age distribution, there's also a large group of patients around age 40-50, which we are not sure about their recall and equalized odds. We should also make sure we are not discriminating against these age groups too, as they are equally vulnerable. We should be fair to individuals and groups in those age groups too, preventing FN.

The choice of model depends on what you cares more, as a doctor, or as a system.

Problem 3 *:Where is the model looking at?*

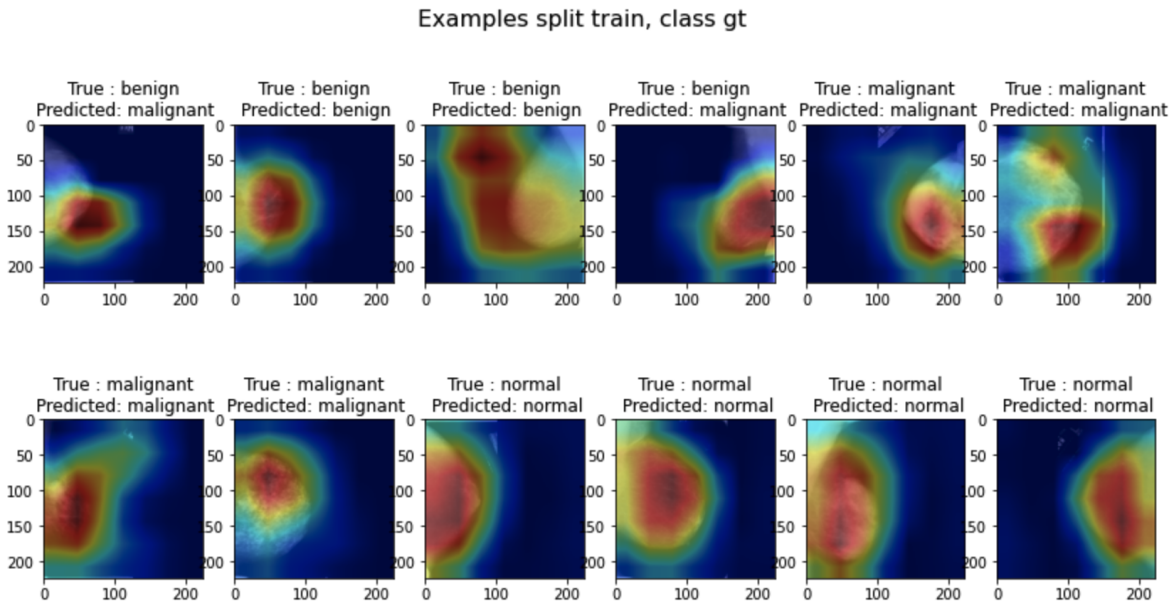

Figure 6: GradCAM in Training dataset

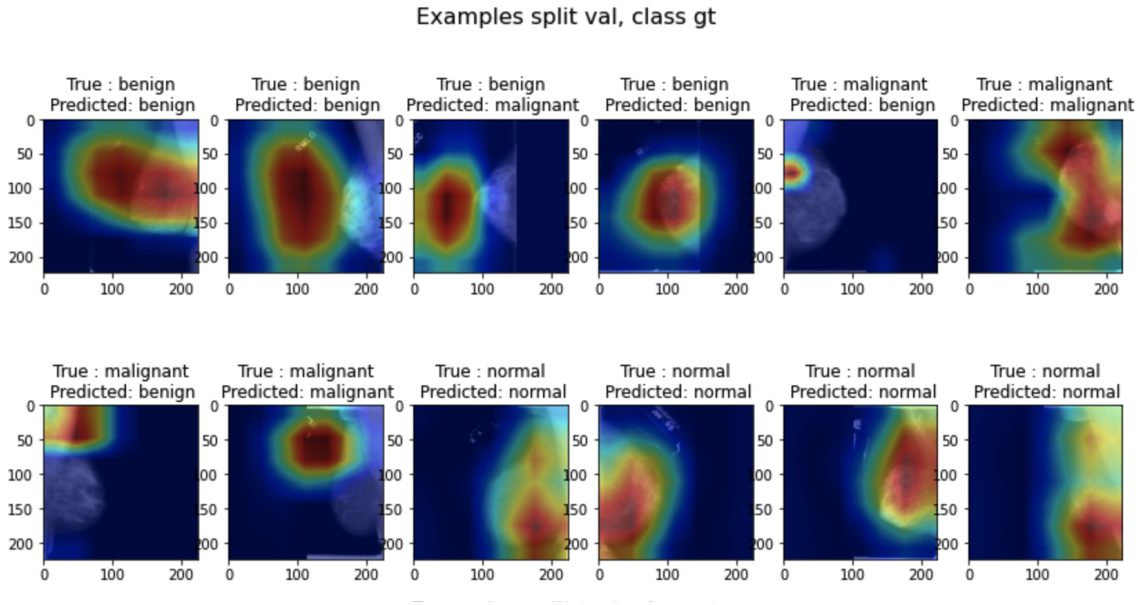

Figure 7: GradCAM in Validation dataset

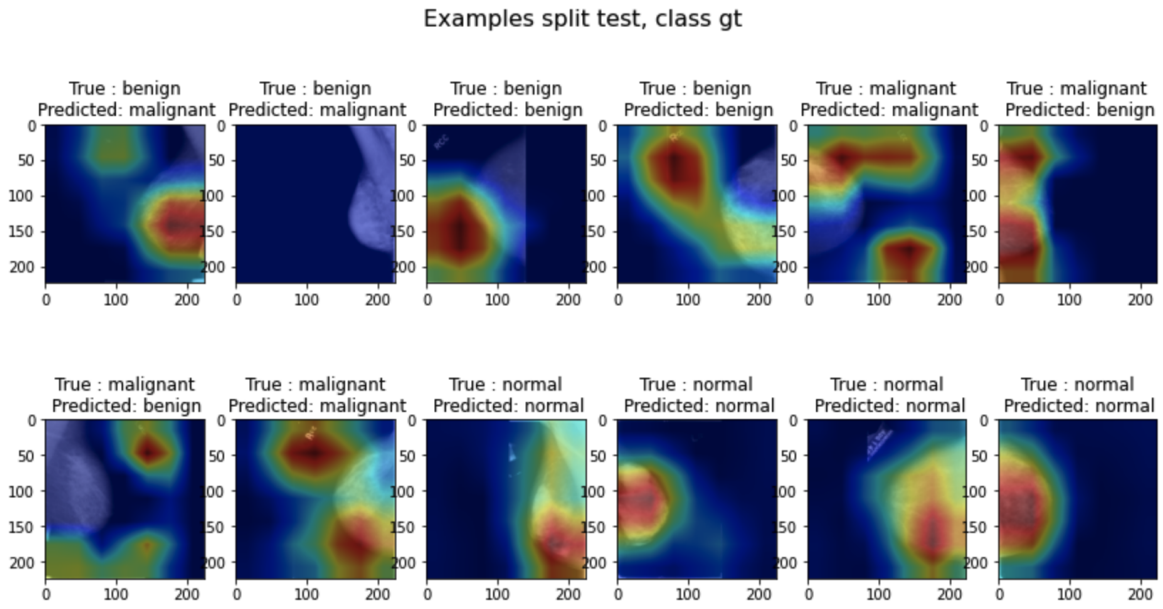

Figure 8: GradCAM in Testing dataset

Looking at highlighted region, in the training dataset, in most cases, the model is looking at the same place of where the breast is, especially for those correctly classified samples. For incorrectly classified samples, there's a portion of overlapping and also some missing coverage.

However, in validation and testing dataset, highlighted region are not exactly hitting expected target of breast in most case. For correctly classified cases, there's a considerable overlapping of highlighting area and target breast area. For incorrectly classified cases, the highlights may be completely missing the target breast area, or only covering the border of the breast object, there's less overlapping.

Problem 4 *:Evaluate on external dataset*

(a) **Performance on External Dataset**

Performance:

1. Top-1 Accuracy: 0.9477
2. External Top-1 Accuracy: 0.8791

From both the accuracy and confusion matrix evaluation, the performance on external dataset is worse than original dataset. Accuracy drops to 87.91% comparing to 94.77%. The recall also significantly drops, the model tends to predict everything as normal, causing many False Negative, and leaving small value of True Positive for both benign and malignant category. This could be due to out of distribution data samples, when model see something they haven't see before, it performs poorly.

```
Top-1 Acc: 0.9477124183006537
External Top-1 Acc: 0.8790849673202614
```

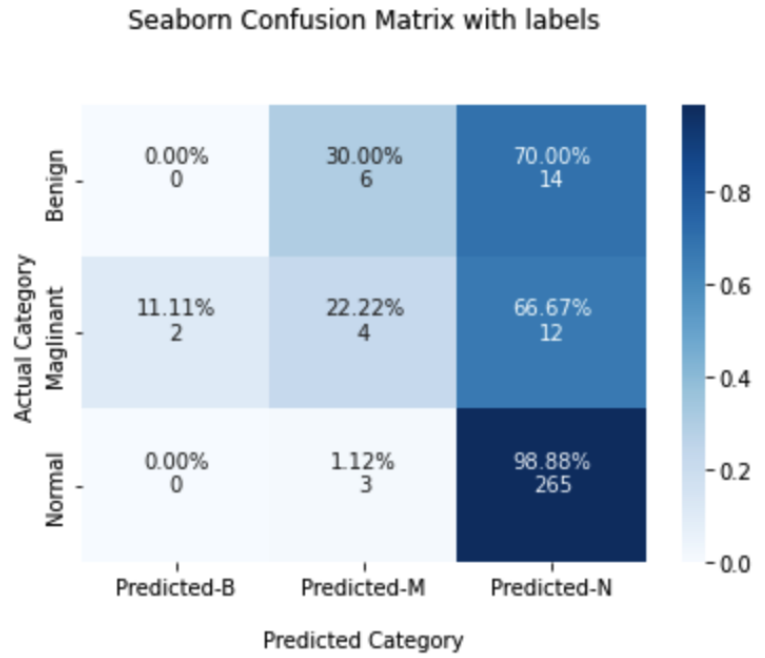

Figure 9: 4a output

(b)**Visualization of Two Data Samples**

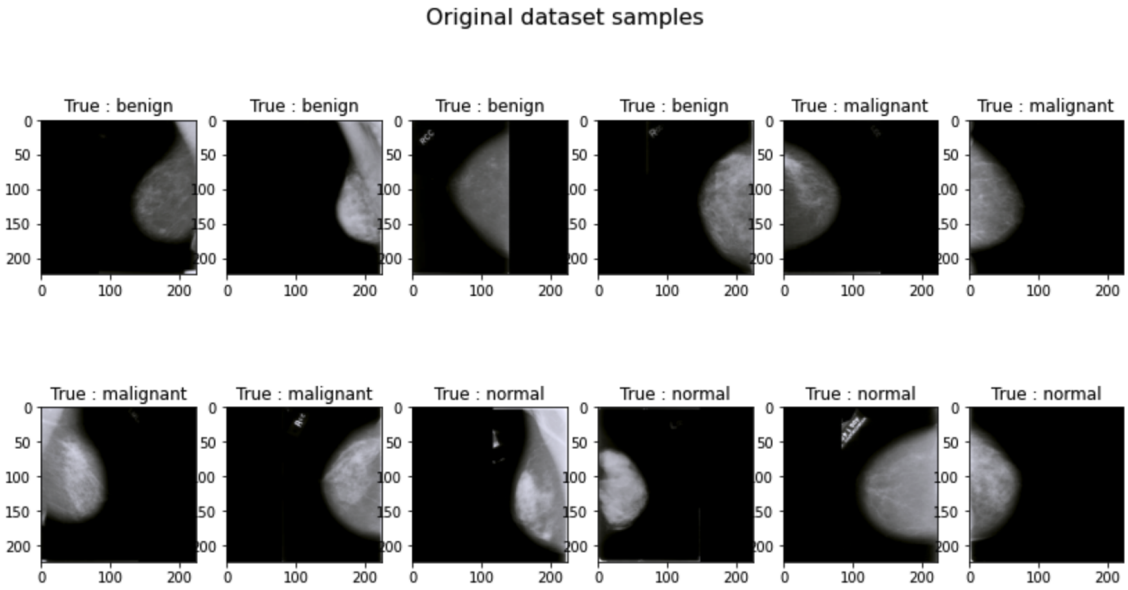

Figure 10: Original Data Samples

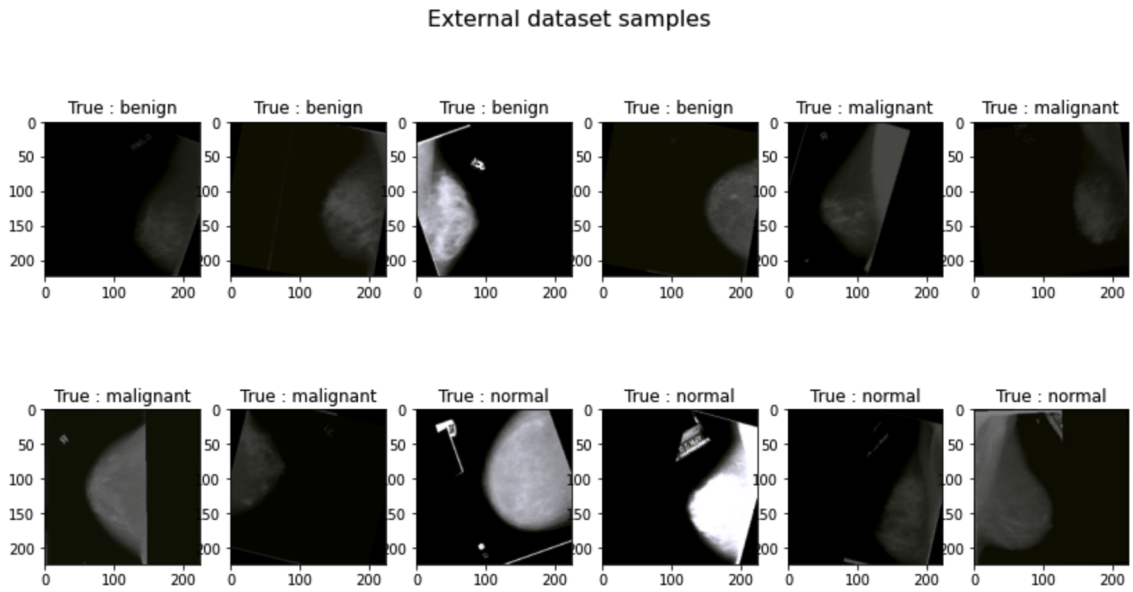

Figure 11: External Data Samples

Comparing two data samples, the original data samples are more consistent in terms of its brightness, direction, and position of objects in scanned film mammography images (either in left side or right side).

However, the external dataset are transformed, for example, rotated, flipped, and also have inconsistent brightness. It seems it is not normalized, or standardized. This may leads to inconsistent performance.

Problem 5 *:Class Imbalance*

(a) **Loss Reweighting**

Weight is defined to be total samples/3/samples per class. So that the sum of weights over all samples should sum up to total samples. Below is the code:

```
# TODO: add the weights per class so that the loss is balanced.
# The sum of weights over all samples should sum up to total_samples (i.e. the same as if th
weights_per_class = torch.tensor([total_samples/3/i for i in samples_per_class]).to(device)
# ENDTODO
```

(b) **Change the Sampling Procedure**

Codes are below:

```
class_i = np.random.choice(len(self.class_probabilities), p = self.class_probabilities)
i = np.random.choice(self.items_per_class[class_i])
```

Results are as expected, three classes' sample are more or less the same and rebalanced.

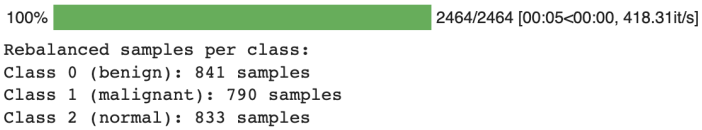

Figure 12: 5b output