Task 1: Breast Ultrasound (3/10 marks, 1+2 marks respectively)

Task 1.1. Split data for training and testing. You need to use the first (sorted in an ascending order by ID) 80% of images per class for training and the remaining 20% for testing. Create a CSV file for this. Report in the PDF file a table with the number of images for training and testing for each class.

Code:

```
FOLDER_PATH = "/share/data_drive1/Dataset_BUSI_with_GT4"
FILES = glob.glob(os.path.join(FOLDER_PATH, "*", "*.png"))

def flatten(t):
    return [item for sublist in t for item in sublist]

dt = {}
patient_ids = []
unique_patient_ids = []
labels = []
files = []
for i in FILES:
    if "mask" in i:
        # ignore any mask files
        pass
    else:
        patient_id = i.split("(")[1].split(")")[0]
        categ = i.split("/")[-2]

        if categ == "normal":
            label = 0
            unique_patient_id = "n" + patient_id
        elif categ == "benign":
            label = 1
            unique_patient_id = "b" + patient_id

        elif categ == "malignant":
            label = 2
            unique_patient_id = "m" + patient_id

        patient_id = int(patient_id)

        patient_ids.append(patient_id)
        unique_patient_ids.append(unique_patient_id)
        labels.append(label)
        files.append(i)

# convert lists to dataframe
df = pd.DataFrame([files, patient_ids, unique_patient_ids, labels]).T
df = df.rename(columns={0: "filepath", 1: "patient_id", 2: "unique_patient_id", 3:
"label"})

# sort values in ascending order by label, then by patient_id
df = df.sort_values(["label", "patient_id"])
df.reset_index(drop=True, inplace=True)

train_pid, test_pid = [], []
y_train, y_test = [], []
for i in set(df["label"]):
    X = df[df["label"] == i]["unique_patient_id"].values
    y = df[df["label"] == i]["label"].values
    X_train_by_class, X_test_by_class, y_train_by_class, y_test_by_class =
        train_test_split(X, y, test_size=0.2, shuffle=False)

    print("class {}: {} train samples, {} test samples".format(i, len(X_train_by_class),
        len(X_test_by_class)))
    train_pid.append(X_train_by_class)
    test_pid.append(X_test_by_class)
```

```
    y_train.append(y_train_by_class)
    y_test.append(y_test_by_class)

# flatten list
train_pid = flatten(train_pid)
test_pid = flatten(test_pid)
y_train = flatten(y_train)
y_test = flatten(y_test)

print("\nTotal:\t {} train samples, {} test samples".format(len(train_pid), len(test_pid)))
```

Result:

```
class 0: 106 train samples, 27 test samples
class 1: 349 train samples, 88 test samples
class 2: 168 train samples, 42 test samples

Total:    623 train samples, 157 test samples
```

Class	Number of Train Samples	Number of Test Samples
0 Normal	106	27
1 Benign	349	88
2 Malignant	168	42
Total	623	157

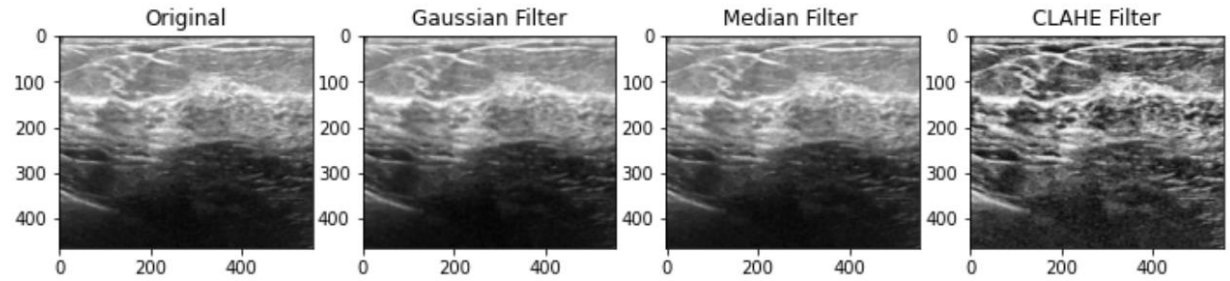
Task 1.2. Use 3 different filtering techniques (choose what you see appropriate) to reduce noise. This will create 3 versions of each image which will be added to the training set. Give a brief description why you used these filters and the main parameters used for each filter.

Code:

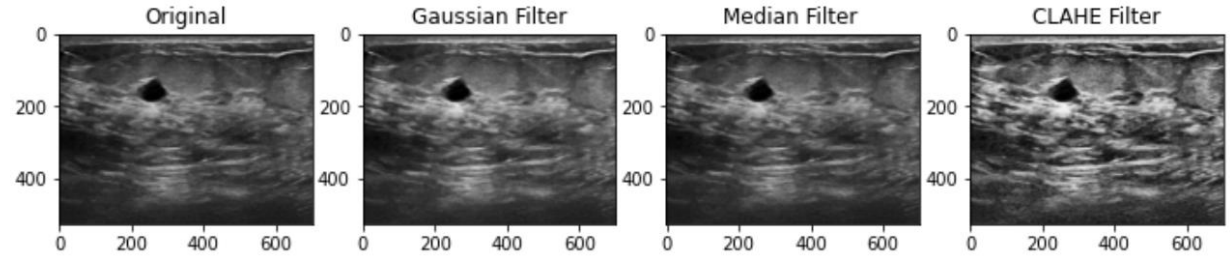
```
X_train_filepaths = []
for unique_pid in train_pid:
    filepath = df[df["unique_patient_id"] == unique_pid]["filepath"].values[0]
    filepath_out = filepath.split(".png")[0]

    label = df[df["unique_patient_id"] == unique_pid]["label"].values[0]

    # Using 0 to read image in grayscale mode
    img = cv2.imread(filepath, 0)

    # apply filters
    img_gauss = filters.gaussian(img, sigma=1)
    img_med = filters.median(img)

    # CLAHE Adaptive Histogram Equalization
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    img_clahe = clahe.apply(img)

    cv2.imwrite(filepath_out + "_gauss.png", img_gauss)
    cv2.imwrite(filepath_out + "_med.png", img_med)
    cv2.imwrite(filepath_out + "_clahe.png", img_clahe)

    X_train_filepaths.append([filepath, filepath_out + "_gauss.png", filepath_out +
         "_med.png", filepath_out + "_clahe.png"])

    # plot sample images
    if "(15)" in filepath_out:
        fig, ((ax1, ax2, ax3, ax4)) = plt.subplots(1, 4, figsize=(12,8))

        [print("benign") if label == 1 else print("malignant") if label == 2 else
         print("normal")]
        ax1.set_title("Original")
        ax1.imshow(img, cmap="gray")

        ax2.set_title("Gaussian Filter")
        ax2.imshow(img_gauss, cmap="gray")

        ax3.set_title("Median Filter ")
        ax3.imshow(img_med, cmap="gray")

        ax4.set_title("CLAHE Filter")
        ax4.imshow(img_clahe, cmap="gray")

        plt.show()

# flatten train_filepaths
X_train_filepaths = [item for sublist in X_train_filepaths for item in sublist]

# multiply each label by 4 (since there are 4 versions of each image: 1 ori + 3 augs)
y_train = [[i]*4 for i in y_train]

# flatten y_train
y_train = [item for sublist in y_train for item in sublist]
```

Result:

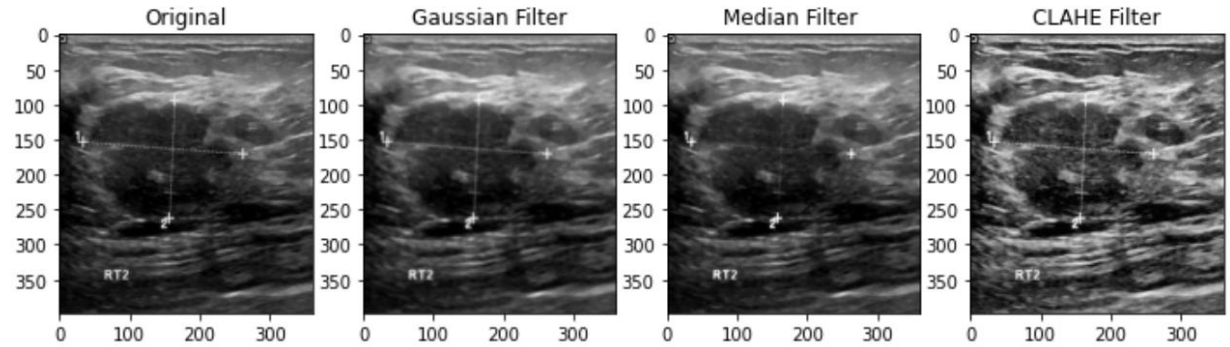
normal



benign



malignant



Brief Description:

I chose the Gaussian, median, and contrast limited adaptive histogram equalization (CLAHE) filters as are commonly used in mammograms [1, 2, 3]. The Gaussian and median filters are effective at reducing speckle and salt and pepper noises. The Gaussian filter takes a sigma input which defines the standard deviation of the Gaussian kernel, while the median filter returns the local median of an image for a given kernel size. The CLAHE filter helps to enhance the contrast of an image via a histogram adjustment strategy. This filter is used with the intention of highlighting important features of the image for the model to distinguish between the normal and abnormal regions.

Task 2: Classification of breast ultrasound images (5/10 marks)

Develop a machine learning algorithm trained and tested on Task 1.1 data. You should also aim to use the data from Task 1.2 for training only.

- Describe the data augmentation strategy you use and the choice of network architecture.
- You must try **exactly** 5 different values for some hyperparameters. You have to make decision on what you try. You must be smart on what to try. Discuss why you made these selections.
- Report the accuracy, confusion matrices, figures for training accuracy and loss for the top 2 experiments you did.

Code:

```
if torch.cuda.is_available():
    device = torch.device("cuda:0")
else:
    device = torch.device("cpu")

def flatten(t):
    return [item for sublist in t for item in sublist]

def train_val(exp_name):
    train_loss = []
    train_acc = []
    val_loss = []
    val_acc = []
    best_accuracy = 0
    for epoch_num in range(0, epochs):
        model.train()
        epoch_total_loss = 0
        correct = 0
        for batch_num, (inp, target) in enumerate(train_loader):
            optimizer.zero_grad()
            output = model(inp.to(device))
            batch_loss = criterion(output,target.to(device))
            _, batch_prediction = torch.max(output, dim=1)
            epoch_total_loss += batch_loss.item()
            batch_loss.backward()
            optimizer.step()
            correct += (batch_prediction == target.to(device)).float().sum()
        accuracy = 100 * correct / len(train_dataset)
        avrg_loss = epoch_total_loss / train_dataset.__len__()
        train_loss.append(avrg_loss)
        train_acc.append(accuracy.detach().cpu())

        model.eval()

        with torch.no_grad():
            labels = []
            predictions = []
            epoch_total_loss = 0

            for (inp, target) in val_loader:
                labels+=target
                batch_prediction = model(inp.to(device))
                batch_loss = criterion(batch_prediction,target.to(device))
                epoch_total_loss += batch_loss.item()

                _, batch_prediction = torch.max(batch_prediction, dim=1)
                predictions += batch_prediction.detach().tolist()
            accuracy_val = metrics.accuracy_score(labels, predictions)*100
            avrg_loss_val = epoch_total_loss / val_dataset.__len__()
            val_loss.append(avrg_loss_val)
            val_acc.append(accuracy_val)

        if accuracy_val > best_accuracy:
            best_accuracy = accuracy_val
```

```
            torch.save(model.state_dict(), FOLDER_PATH +
"/best_acc_{}.pt".format(exp_name))

        print("Epoch %d - loss=%0.4f - acc=%0.4f - val-loss=%0.4f - val-acc=%0.4f" %
(epoch_num, avrg_loss, accuracy, avrg_loss_val, accuracy_val))

    return train_loss, train_acc, val_loss, val_acc

def test(exp_name):
    model.eval()

    labels = []
    predictions = []

    test_loss = []
    test_acc = []
    for (inp, target) in test_loader:
        labels+=target
        batch_prediction = model(inp.to(device))
        _, batch_prediction = torch.max(batch_prediction, dim=1)
        predictions += batch_prediction.detach().tolist()
    accuracy = metrics.accuracy_score(labels, predictions)
    print("Test Accuracy = %0.2f" % (accuracy))

    confusion = metrics.confusion_matrix(labels, predictions)

    print(exp_name)
    sns.heatmap(confusion, annot=True, cmap='Blues')
    plt.title("Confusion Matrix (sample count)")
    plt.show()
    sns.heatmap(confusion/np.sum(confusion), annot=True,
            fmt='.1%', cmap='Blues')
    plt.title("Confusion Matrix (%)")
    plt.show()

X_train_filepaths = []
X_val_filepaths = []
for label in ["normal", "benign", "malignant"]:
    files = [i for i in X_train_filepaths if label in i]
    files = sorted(files)

    # indices of unique patients
    pid = [i*4 for i in range(int(len(files)/4))]

    # (incorrect) labels of unique patients - used as a placeholder..
    y_tr = [sorted(y_train)[i] for i in pid]

    # get indices of train-val splits
    X_tr_u, X_val_u, _, _ = train_test_split(pid, y_tr, test_size=0.2)
    X_tr_u = flatten([[i, i+1, i+2, i+3] for i in X_tr_u])
    X_val_u =  flatten([[i, i+1, i+2, i+3] for i in X_val_u])

    # get elements of train-val splits
    X_tr2 = [X_train_filepaths[i] for i in X_tr_u]
    X_val2 = [X_train_filepaths[i] for i in X_val_u]

    X_train_filepaths.append(X_tr2)
    X_val_filepaths.append(X_val2)

X_train_filepaths = flatten(X_train_filepaths)
X_val_filepaths = flatten(X_val_filepaths)

# create train & test directories, then move (copy) files
train_dest = FOLDER_PATH + "/train"
val_dest = FOLDER_PATH + "/val"
test_dest = FOLDER_PATH + "/test"

if not os.path.exists(train_dest):
```

```
    os.mkdir(train_dest)
    os.mkdir(train_dest + "/normal")
    os.mkdir(train_dest + "/benign")
    os.mkdir(train_dest + "/malignant")

if not os.path.exists(val_dest):
    os.mkdir(val_dest)
    os.mkdir(val_dest + "/normal")
    os.mkdir(val_dest + "/benign")
    os.mkdir(val_dest + "/malignant")

if not os.path.exists(test_dest):
    os.mkdir(test_dest)
    os.mkdir(test_dest + "/normal")
    os.mkdir(test_dest + "/benign")
    os.mkdir(test_dest + "/malignant")

for file in X_train_filepaths2:
    filename = file.split("/")[-1]
    folder_labelname = file.split("/")[-2]
    shutil.copy(file, os.path.join(train_dest, folder_labelname, filename))

for file in X_val_filepaths:
    filename = file.split("/")[-1]
    folder_labelname = file.split("/")[-2]
    shutil.copy(file, os.path.join(val_dest, folder_labelname, filename))

for file in X_test_filepaths:
    filename = file.split("/")[-1]
    folder_labelname = file.split("/")[-2]
    shutil.copy(file, os.path.join(test_dest, folder_labelname, filename))

train_dataset = DatasetFolder(root=train_dest + "/",
                              loader=cv2.imread,
                              extensions=(".png",),
                              transform=transforms.Compose([
                                  transforms.ToTensor(),
                                  transforms.RandomHorizontalFlip(p=0.5),
                                  transforms.RandomAffine(degrees=0, scale=(1, 1.15)),
                                  transforms.RandomApply(torch.nn.ModuleList([
                                      transforms.RandomRotation((90,90)),
                                      transforms.RandomRotation((180,180)),
                                      transforms.RandomRotation((270,270)),
                                  ]), p=0.5),
                                  transforms.Resize((256,256)),
                              ]
                          ))

val_dataset = DatasetFolder(root=val_dest + "/",
                              loader=cv2.imread,
                              extensions=(".png",),
                              transform=transforms.Compose([
                                  transforms.ToTensor(),
                                  transforms.Resize((256,256)),
                              ]
                          ))

test_dataset = DatasetFolder(root=test_dest + "/",
                              loader=cv2.imread,
                              extensions=(".png",),
                              transform=transforms.Compose([
                                  transforms.ToTensor(),
                                  transforms.Resize((256,256)),
                              ]
                          ))

train_loader = DataLoader(train_dataset,
                          batch_size=4,
```

```
                          shuffle=True,
                          num_workers=0)

val_loader = DataLoader(val_dataset,
            batch_size=4,
            shuffle=False,
            num_workers=0)

test_loader = DataLoader(test_dataset,
            batch_size=4,
            shuffle=False,
            num_workers=0)

# parameters
epochs = 15
lr = 1e-3
weights = [4.0, 1.0, 2.0]
weight_decay = 0.2
dropout = 0.5

model = resnet18(num_classes=3).to(device)
if dropout is not None:
    model.fc = nn.Sequential(nn.Dropout(dropout),
                             nn.Linear(model.fc.in_features,3)).to(device)

class_weights = torch.FloatTensor(weights).to(device)
criterion = CrossEntropyLoss(weight=class_weights)
optimizer = optim.Adam(model.parameters(),lr=lr,weight_decay=weight_decay)

train_loss1, train_acc1, test_loss1, test_acc1 = train_val("expX")

test("expX")

print("Train (blue) vs. Val (red)")
plt.plot([i for i in range(epochs)], train_loss1, color="b")
plt.plot([i for i in range(epochs)], test_loss1, color="r")
plt.xlim(0,20)
plt.title("Loss")
plt.show()

plt.plot([i for i in range(epochs)], train_acc1, color="b")
plt.plot([i for i in range(epochs)], test_acc1, color="r")
plt.xlim(0,20)
plt.title("Accuracy")
plt.show()
```

Description:
Based on a work published on abnormality diagnosis in mammograms [4], I chose ResNet18 as my network architecture as the authors have found it to give the best performance compared to other larger ResNet architectures.

I augmented the images primarily using geometric transformations to create combinations that are likely to be representative of a blind test image. Following [4], [5], and [6], I used a random horizontal flip, random upscaling up to 115%, and random rotation between 90, 180, and 270 degrees. I resized the images to 256.

Given the dissimilarity between ImageNet and mammograms, I decided to train the network from scratch instead of using pretrained ImageNet weights. My experiments were as follows:

Experiment 1. ResNet18 baseline, 15 epochs, learning rate of 1e-3. This gives an accuracy of 0.43.

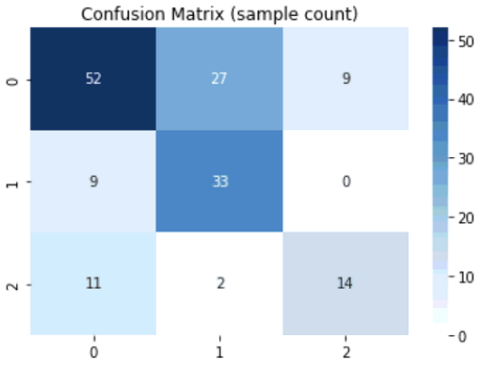
Experiment 2. Given that the classes were imbalanced with the test set having 27, 88, and 42 samples for normal, benign, and malignant images respectively, I added class weights in inverse proportions [4, 1, 2] to penalize the model more for misclassifying samples with the least number of samples. This increased the accuracy to 0.52.

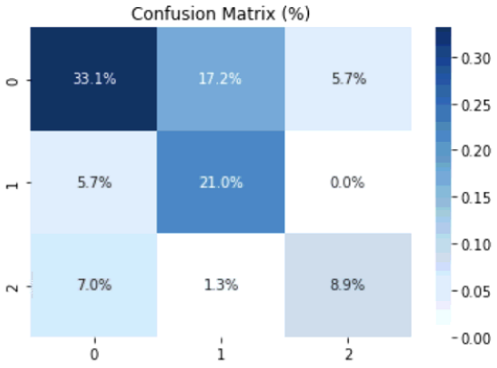
Experiment 3. In Experiments 1 and 2, the model had overfitted to the training set. To reduce this overfitting, I added dropout to the fully connected layer with a probability of 0.5. This increased the accuracy to 0.54.

Experiment 4. Given the sporadic accuracy and losses, I reduced the learning rate to 1e-5. This resulted in a more stable loss, and an improved accuracy to 0.63.

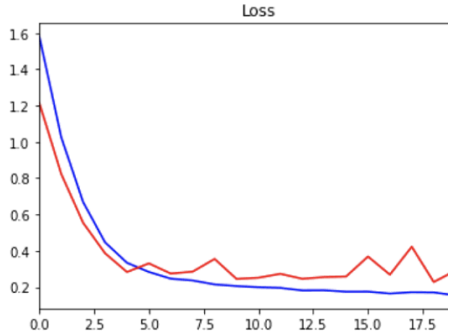
Experiment 5. Finally, I combined the previous experiments by adding class weight [4, 1, 2], dropout of 0.4, and learning rate 1e-5. This gave the best accuracy of 0.67..

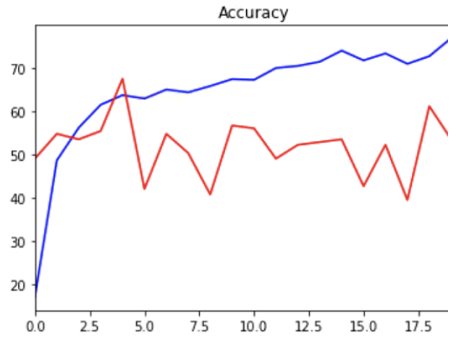
Results:
Experiment 4 (Accuracy 0.63)



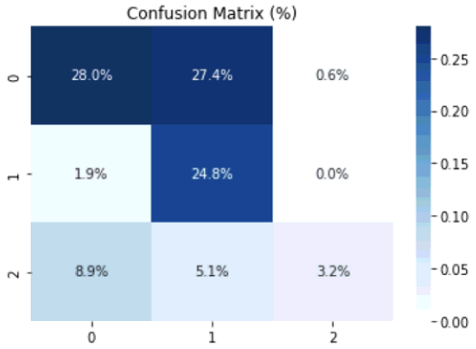


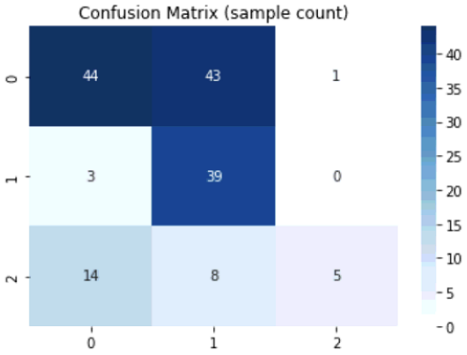
Training accuracy and loss



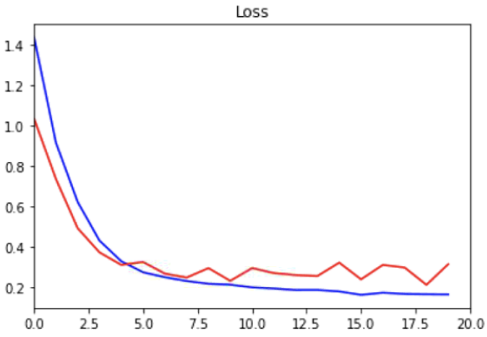


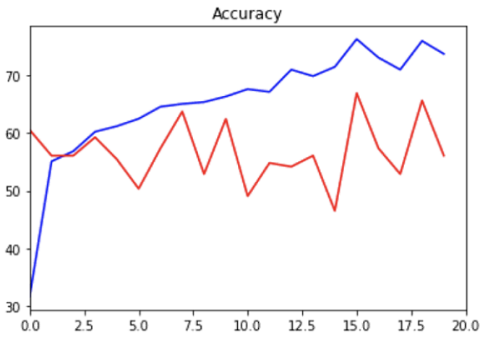
Experiment 5 (Accuracy 0.67)





Training accuracy and loss





Task 3: Visualization of heat maps (2/10 Marks, one mark each)

- Use the best classification model you achieved to create heat maps that highlight the regions on images which contributed to the classification results. Discuss briefly the way you achieved this.
- Report results by showing 6 images from testing set with heat maps super-imposed. These 6 images are 2 images (one correct and one incorrect classification) from each class.

Brief discussion:

I used GradCAM inspired by the code from https://github.com/jacobgil/pytorch-grad-cam. In order to find the correctly and incorrectly classified images, I used GradCAM's inbuilt function to compare the heatmap from the model's prediction versus the heatmap from the class labels. If the heatmaps match, the image is correctly classified; otherwise, the image is incorrectly classified.

Code:

```
!pip install grad-cam
import glob, os, cv2
from pytorch_grad_cam import GradCAM, ScoreCAM, GradCAMPlusPlus, AblationCAM, XGradCAM,
EigenCAM
from pytorch_grad_cam.utils.image import show_cam_on_image
from torchvision.models import resnet50

def my_gradcam(model_path, input_path, true_label, correct_prediction=True):
    '''
    true_label: "malignant", "benign", or "normal"
    '''
    model = resnet18(num_classes=3)
    model.load_state_dict(torch.load(model_path))
    target_layers = [model.layer4[-1]]

    if true_label == "malignant":
        target_category = 2
    elif true_label == "benign":
        target_category = 1
    elif true_label == "normal":
        target_category = 0

    if true_label in input_path:
        img = cv2.imread(input_path,1)
        img = cv2.resize(img, (256,256))
        filename = input_path.split("/")[-1].split(".png")[0]

        input_tensor = transforms.ToTensor()(img).unsqueeze_(0)
        input_tensor = input_tensor.expand(-1, 3, -1, -1)

        cam = GradCAM(model=model, target_layers=target_layers, use_cuda=True)

        grayscale_cam_predicted_label = cam(input_tensor=input_tensor,
                                            target_category=None)
        grayscale_cam_true_label = cam(input_tensor=input_tensor,
                                       target_category=target_category)

        if correct_prediction:
            if np.array_equal(grayscale_cam_predicted_label, grayscale_cam_true_label):
                print("{} Prediction".format(correct_prediction))
                print(filename)

                grayscale_cam = grayscale_cam_predicted_label[0, :]
                visualization = show_cam_on_image(img/255, grayscale_cam, use_rgb=True)

                plt.imshow(visualization)
                plt.show()
                return

        else:
```

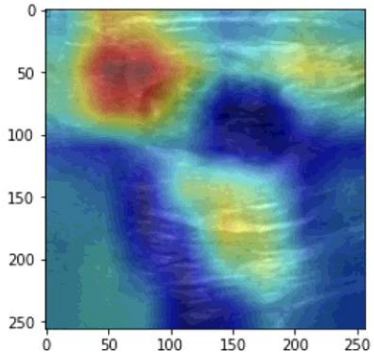
```
            if not np.array_equal(grayscale_cam_predicted_label, grayscale_cam_true_label):
                print("{} Prediction".format(correct_prediction))
                print(filename)

                grayscale_cam = grayscale_cam_predicted_label[0, :]
                visualization = show_cam_on_image(img/255, grayscale_cam, use_rgb=True)

                plt.imshow(visualization)
                plt.show()
                return


FOLDER_PATH = "/share/data_drive1/Dataset_BUSI_with_GT"
current_testpaths = glob.glob(os.path.join(FOLDER_PATH, "test", "*", "*"))

for testpath in current_testpaths:
    my_gradcam("/share/data_drive1/Dataset_BUSI_with_GT/best_acc_exp2.pt", testpath,
               "malignant", True)
    my_gradcam("/share/data_drive1/Dataset_BUSI_with_GT/best_acc_exp2.pt", testpath,
               "malignant", False)

for testpath in current_testpaths:
    my_gradcam("/share/data_drive1/Dataset_BUSI_with_GT/best_acc_exp2.pt", testpath,
               "benign", True)
    my_gradcam("/share/data_drive1/Dataset_BUSI_with_GT/best_acc_exp2.pt", testpath,
               "benign", False)

for testpath in current_testpaths:
    my_gradcam("/share/data_drive1/Dataset_BUSI_with_GT/best_acc_exp2.pt", testpath,
               "normal", True)
    my_gradcam("/share/data_drive1/Dataset_BUSI_with_GT/best_acc_exp2.pt", testpath,
               "normal", False)
```

Results:

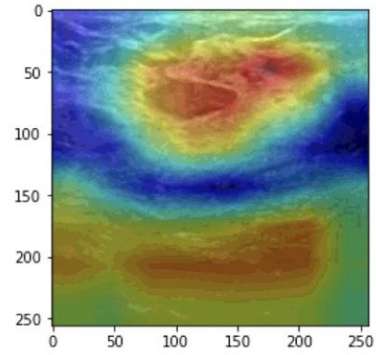
From the GradCAM outputs, the model seems to be able to focus on specific regions for the correctly predicted images. Conversely, on the incorrectly classified images, the heatmaps seem more spread out, and the model has a harder time focusing on specific regions of the image. This suggests that the model has not yet fully learned the proper features that distinguish the abnormalities on the mammograms

.

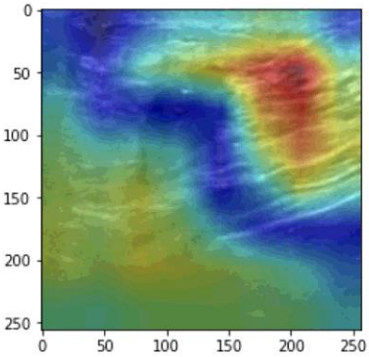
True Prediction
benign (399)



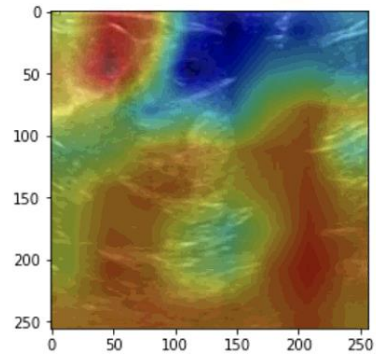
False Prediction
benign (417)



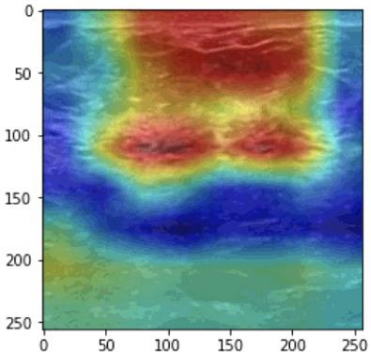
True Prediction
malignant (190)



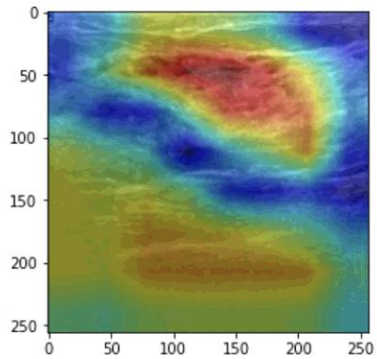
False Prediction
malignant (173)



True Prediction
normal (112)



False Prediction
normal (109)



References

[1] Mari, Kamarasan & Santhakumari, V. (2020). Early Detection of Breast Cancer using Image Processing Techniques: A Study on Mammogram Image Analysis. XII. 2266-2296.

[2] Kshema, & George, Jayesh & Dhas, Anto. (2017). Preprocessing filters for mammogram images: A review. 1-7. 10.1109/ICEDSS.2017.8073694.

[3] Pisano, Etta & Cole, Elodia & Hemminger, Bradley & Yaffe, Martin & Aylward, Stephen & Maidment, Andrew & Johnston, R. & Williams, Mark & Niklason, Loren & Conant, Emily & Fajardo, Laurie & Kopans, Daniel & Brown, Marylee & Pizer, Stephen. (2000). Image Processing Algorithms for Digital Mammography: A Pictorial Essay1. Radiographics : a review publication of the Radiological Society of North America, Inc. 20. 1479-91. 10.1148/radiographics.20.5.g00se311479.

[4] Yu, Xiang & Wang, Shuihua. (2019). Abnormality Diagnosis in Mammograms by Transfer Learning Based on ResNet18. Fundamenta Informaticae. 168. 219-230. 10.3233/FI-2019-1829.

[5] Abdelhafiz, D., Yang, C., Ammar, R. *et al.* Deep convolutional neural networks for mammography: advances, challenges and applications. *BMC Bioinformatics* **20,** 281 (2019). https://doi.org/10.1186/s12859-019-2823-4

[6] Costa, Arthur & Oliveira, Helder & Catani, Juliana & Barros, Nestor & Melo, Carlos & Vieira, Marcelo. (2019). Detection of Architectural Distortion with Deep Convolutional Neural Network and Data Augmentation of Limited Dataset. 10.1007/978-981-13-2517-5_24.