**Mammography Image Classification Using Random Forest, Naive Bayes, and CNN w/CUDA (Majority Voting)**

**Model Guide for MajorityVoting.py**

1. **Starting with Importing the required libraries**

“import time

…

import ssl”

1. **Setting up GPU**

“device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')”

* To start with, please check the reqirements that is avaiable on the GitHub readme document of our project. Following that, please refer to the latest avaiable version that is avaiable for your device from PyTorch official site “<https://pytorch.org/get-started/locally/>” to set up PyTorch locally to utilize your GPU.
* In this section, we are checking if our GPU is avaiable and setting it accordingly.

1. **Transforming section**

“transform = transforms.Compose([

transforms.Resize((224, 224)),

transforms.Grayscale(num\_output\_channels=3),

transforms.ToTensor(),

transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),

])”

* In this section, we are resizing the images. Having a consistent image size is efficient in terms of computing power, since we are handling large data. Size of 224x224 pixels is often used in this area as we researched.
* As for “num\_output\_channels=3” parameter, eventhough we are using grayscale images, it is stated that various pretrained models can expect 3 channel input images, since they are used in several different areas, not only medical images. By doing so, we try to avoid some compatibility problems.
* “transforms.Normalize” method is used to ensure the consistency.

1. **Dataset class section**

“class MammographyDataset(Dataset):

def \_init\_(self, data\_dir, data\_transform=None):

…he

def \_len\_(self):

…

def \_getitem\_(self, idx):

…”

* In this section, we are listing all the files we have in our dataset. Following that, we return the number of images using the “\_len\_” section. With “\_getitem\_”, we open an image, apply transformations, and return the image and its label.

1. **Loading dataset, train and test loads**

“dataset = MammographyDataset(data\_dir='D:/jpeg', data\_transform=transform)

X = [dataset[i][0].numpy() for i in range(len(dataset))]

y = [dataset[i][1] for i in range(len(dataset))]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

X\_train = np.array(X\_train)

X\_test = np.array(X\_test)

y\_train = np.array(y\_train)

y\_test = np.array(y\_test)”

* Firstly, we are assigning our dataset variable to the actual dataset we have typing its directory. By capital “X” variable, we create a list of image data from our dataset. With lowercase “y” variable, we create a list of labels from our dataset.
* In the following line, we are using %20 of the data for testing. As fort he “random\_state” section, we decided to use a fixed number to ensure that we get healthy results comparing the outcomes from different tests we run, not because we had a different split of test and train.
* Following line is used to split our data into training and test sets:
  + “X\_train” contains 80% of the images for training.
  + “X\_test” contains 20% of the images for testing.
  + “y\_train” contains the labels corresponding to X\_train.
  + “y\_test” contains the labels corresponding to X\_test.
* Finally, each of them are converted into arrays to work efficiently.

1. **Random Forest section**

“print("Training Random Forest Classifier...")

vit\_model = models.vit\_b\_16(weights=models.ViT\_B\_16\_Weights.DEFAULT)

vit\_model = vit\_model.to(device)

vit\_features = [vit\_model(torch.tensor(img).unsqueeze(0).to(device)).cpu().detach().numpy().flatten() for img in X\_train]

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(vit\_features, y\_train)

vit\_features\_test = [vit\_model(torch.tensor(img).unsqueeze(0).to(device)).cpu().detach().numpy().flatten() for img in X\_test]

y\_pred\_rf = rf\_model.predict(vit\_features\_test)

print("Random Forest Classifier trained.")”

* We start with initializing a ViT model that is pretrained in the line after printing that the process has started and then move that to the GPU that we have declared as “device” at the second part of our manual.
* By “vit\_features”, we extract features from training images using the model and store them in a list
* Following that initialization, by using parameters “n\_estimators=100, random\_state=42” we declare the tree number in the forest, that is 100 in this case. As for the “random\_state=42” variable, we use this to ensure that the results are consistent across different runs.
* In the remaining 2 lines just before printing that our model has done with training, we train the classifier on the extracted features and corresponding labels and extracting features from each test image using the model and stores them in a list.

1. **Naive Bayes section**

“print("Training Naive Bayes Classifier...")

nb\_model = GaussianNB()

X\_train\_nb = [img.flatten() for img in X\_train]

X\_test\_nb = [img.flatten() for img in X\_test]

nb\_model.fit(X\_train\_nb, y\_train)

y\_pred\_nb = nb\_model.predict(X\_test\_nb)

print("Naive Bayes Classifier trained.")”

* After printing that the training has started, initialization of the classifier is done. This process is followed by the flattening of each image in test set, and them getting stored in a list.
* We train the classifier on the flattened training images, including their labels.
* In the end, trained classifier is used to make predictions on flattened test images.

1. **CNN model section**

“class CNNModel(nn.Module):

def \_\_init\_\_(self):

…

def forward(self, x):

…

cnn\_model = CNNModel()

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(cnn\_model.parameters(), lr=0.001)

…

num\_epochs = 30

for epoch in range(num\_epochs):

…

y\_pred\_cnn = []

with torch.no\_grad():

…”

* As for the CNN section, model used in the majority voting script is nearly identical to the individual CNN manual we prepared. You can refer to the manual prepared for the CNN model, as it follows the more or less the same implementation. Extra information in additional to that manual can be found under this statement.
* For “optimizer = torch.optim.Adam(cnn\_model.parameters(), lr=0.001)”, by stating that “lr=0.001” we set our learning rate to “0.001”. Learning rate determines the size of the steps the optimizer takes to adjust the model's weights during training.
* Note: In short, our goal was to achieve minimum loss in a fast way. For instance, in the initial dataset, using a learning rate of “0.001” with an epoch number around “30” was majorly getting better results in terms of precision. Additionally, for getting faster results, including additional layers or higher channel values for CNN model in “\_init\_” section can be done, though it is VRAM heavy.

1. **Majority voting section**

“print("Performing Majority Voting...")

y\_pred\_ensemble = []

for i in range(len(y\_test)):

votes = [y\_pred\_rf[i], y\_pred\_nb[i], y\_pred\_cnn[i]]

y\_pred\_ensemble.append(np.bincount(votes).argmax())

accuracy\_ensemble = accuracy\_score(y\_test, y\_pred\_ensemble) \* 100

print(f'Accuracy on test set (Majority Voting): {accuracy\_ensemble:.2f}%')”

* In this final section, we combine the predictions from Random Forest, Naive Bayes, and CNN models using majority voting. The final prediction for each test sample is determined by the most common prediction among the three models used in the script and the result is printed.