**Mammography Image Classification Using ViT (Improvised Script)**

**Model Guide for ViTDuzgun.py**

1. **Starting with Importing the required libraries and GPU check**

“import os

import pandas as pd

…

print("Num GPUs Available: ", len(tf.config.experimental.list\_physical\_devices('GPU')))”

* After importing the required libraries for our calculations, we check if a GPU is avaiable to achieve faster computation times, drastically.

1. **Loading Metadata and Image Paths section**

“meta\_data\_path = 'F:/Dataset/archive/csv/meta.csv'

…

df\_dicom = pd.read\_csv(dicom\_info\_path, delimiter=';')”

* In this section, we are specifying files and directories for further calculations. CSV files are read into pandas for easier data manipulation.

1. **Extract and filter section**

“series\_descriptions = df\_dicom.SeriesDescription.unique()

cropped\_image\_paths = df\_dicom[df\_dicom.SeriesDescription == 'cropped images'].image\_path

…

cropped\_image\_paths = cropped\_image\_paths.str.replace('CBIS-DDSM/jpeg', jpeg\_dir)

…

full\_mammo\_dict = {path.split("/")[4]: path for path in full\_mammo\_paths}

…”

* As fort his section, we are extracting unique image types, filtering their paths based on descriptions, and updating paths to have a correct organization in dictionaries.

1. **Loading, renaming columns section**

“mass\_train = pd.read\_csv(mass\_train\_path)

mass\_test = pd.read\_csv(mass\_test\_path)

mass\_train = mass\_train.rename(columns={

'left or right breast': 'left\_or\_right\_breast',

…

})

mass\_test = mass\_test.rename(columns={

'left or right breast': 'left\_or\_right\_breast',

…})”

* To start, specifying the paths to training and test datasets are done, and they are read into pandas dataframes. This section is made to ensure data at our hand is consistently stored.
* In the next block, we are renaming the columns of datasets we declared to have consistent naming across all of them. For instance, whether the mass is in the left or right breast is related to column “'left or right breast' -> 'left\_or\_right\_breast'”.

1. **Image path section**

“def update\_image\_paths(df):

for idx, row in df.iterrows():

full\_image\_key = row['image\_file\_path'].split("/")[2]

cropped\_image\_key = row['cropped\_image\_file\_path'].split("/")[2] if pd.notna(row['cropped\_image\_file\_path']) else None

if full\_image\_key in full\_mammo\_dict:

df.at[idx, 'image\_file\_path'] = full\_mammo\_dict[full\_image\_key]

if cropped\_image\_key and cropped\_image\_key in cropped\_images\_dict:

df.at[idx, 'cropped\_image\_file\_path'] = cropped\_images\_dict[cropped\_image\_key]

update\_image\_paths(mass\_train)

update\_image\_paths(mass\_test)”

* This section of the script is to make sure that image file paths in our dataset are correct, consistent as for the directory structure we have prepared.

1. **Missing values section**

“mass\_train['mass\_shape'].bfill(inplace=True)

mass\_train['mass\_margins'].bfill(inplace=True)

mass\_test['mass\_margins'].bfill(inplace=True)”

* In this section, we are checking for missing values in important columns, such as 'mass\_shape' and 'mass\_margins' variables that can be found in the 4th section where we prepared the column using “backfilling” method.

1. **Image display, merge, and resizing section**

“def display\_images(column, number):

number\_to\_visualize = number

rows = 1

cols = number\_to\_visualize

…

display\_images('image\_file\_path', 5)

display\_images('cropped\_image\_file\_path', 5)

…

full\_mass\_data = pd.concat([mass\_train, mass\_test], axis=0)

…

def resize\_image(image, size=(224, 224)):

return cv2.resize(image, size)”

* In this section, we are setting a function to display the images from a column in our dataset. We have two parameters; “column” for file path, and “number” for the number of images to be displayed.
* Firstly, we get the image path by the line “image\_path = row[column]”, and check that if that image path exists or not by the following if statement. After setting the required title for that image in the following lines, we call that function to display the image.
* After these steps are completed, we concatenate training and test datasets to achieve easy processing among all the data we have (preprocessing) and resize the images to 224x224 pixel format, that is commonly preferred in the field.

1. **Preprocess and updating DataFrame section**

“def load\_and\_preprocess\_images(image\_paths, feature\_extractor, batch\_size=100):

images = []

valid\_paths = []

for i in range(0, len(image\_paths), batch\_size):

batch\_paths = image\_paths[i:i+batch\_size]

batch\_images = []

for path in batch\_paths:

if path and os.path.exists(path):

image = mpimg.imread(os.path.abspath(path))

…

feature\_extractor = ViTImageProcessor.from\_pretrained('google/vit-base-patch16-224-in21k')

X, valid\_image\_paths = load\_and\_preprocess\_images(full\_mass\_data['image\_file\_path'], feature\_extractor)”

* To start the process, we defined a function taking 3 parameters; list of image file paths to be loaded, extracting the features (transformers library used), and the number of images to be processed.
* Starting with initializations, an empty list to store processed images, and a list to store paths is declared. Then we start the iteration based on the batch we took as an input in the previous bulletpoint. A list to store the images in the current batch is also created in loop.
* After passing the tests such as path validation, feature extraction process starts. We are using the Google ViT model that is avaiable online, in the line “feature\_extractor = ViTImageProcessor.from\_pretrained('google/vit-base-patch16-224-in21k')”. With this process, we make sure that images are in correct format, and we can continue our calculations consistently. Finally, we update the dataframe for records.

1. **Preparing labels, converting, and re splitting data section**

“y = full\_mass\_data['pathology'].replace({'MALIGNANT': 1, 'BENIGN': 0, 'BENIGN\_WITHOUT\_CALLBACK': 0}).values

…

X = tf.convert\_to\_tensor(X)

…

# Split data into train, test, and validation sets (70, 20, 10)

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X.numpy(), y.numpy(), test\_size=0.3, random\_state=42)

X\_test, X\_val, y\_test, y\_val = train\_test\_split(X\_temp, y\_temp, test\_size=0.33, random\_state=42)

…”

* In this section, we start by alligning numerics for for training such as 1 for Malignant. Following that, image data and labels is converted into tensors for training. At the end, Data is splitted into training (70%), test (20%), and validation (10%) sets using “train\_test\_split” and they also get converted into tensors after these steps are done.

1. **Checks, and encoding section**

“train\_class\_counts = np.bincount(y\_train.numpy())

…

print("Number of occurrences of each class in y\_train:")

…

train\_datagen = ImageDataGenerator(rotation\_range=40,

…

train\_data\_augmented = train\_datagen.flow(X\_train.numpy(), y\_train, batch\_size=32)

def check\_batch\_shape(data\_gen):

for X\_batch, y\_batch in data\_gen:

print(f"Batch X shape: {X\_batch.shape}")

break

check\_batch\_shape(train\_data\_augmented)

…

tf.keras.mixed\_precision.set\_global\_policy('mixed\_float16')

…”

* We start by checking occurances in classes in training and test sets for any unwanted occurances and labels are translated into one-hot encoding labels for represent categories as numerics. In the augmentation section, we aim to increase diversity in training data alongside with other techniques following the augmentation section to improve results.

1. **Defining the model, learning rate, resamples section**

“with strategy.scope():

vit\_model = TFViTModel.from\_pretrained('google/vit-base-patch16-224-in21k')

inputs = tf.keras.Input(shape=(3, 224, 224), name='pixel\_values', dtype=tf.float32)

…

model = tf.keras.Model(inputs=inputs, outputs=outputs)

metrics=['accuracy']

)

…

def lr\_schedule(epoch, lr):

if epoch < 7:

return lr

else:

return lr \* tf.math.exp(-0.01)

early\_stopping = tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=15, restore\_best\_weights=True)

…

smote = SMOTE(random\_state=42)

X\_train\_resampled, y\_train\_resampled = smote.fit\_resample(X\_train.numpy().reshape(X\_train.shape[0], -1), y\_train.argmax(axis=1))

…

* In this section, we are defining the ViT model and integrate it. Detailed information for ViT model can be found in our ViT manual, extra sections will be discussed here.
* In addition to our base ViT scripts, this model uses early stopping by decreasing the learning rate after 7 epochs that is defined as declared as “learning\_rate=0.0001” initially, and adapts resampling strategies to prevent overfitting and saving resources and augment it.

1. **Training, evaluating, generating reports section**

“history = model.fit(

train\_data\_augmented,

epochs=100,

…

model.summary()

model.evaluate(X\_test, y\_test)

…

cm\_labels = ['MALIGNANT', 'BENIGN']

y\_pred\_test = model.predict(X\_test)

y\_pred\_train = model.predict(X\_train)

…

train\_cm = confusion\_matrix(y\_true\_classes\_train, y\_pred\_classes\_train))”

* In this part, training the model using our augmented data (see previous actions such as early stopping, validation data, etc.) starts.
* Following the training, model summary and evaluation is printed and the classifications are done referring to the previous labels declared in section 9, such as MALIGNANT.

1. **Plotting and saving section**

“def plot\_confusion\_matrix(cm, labels, title):

row\_sums = cm.sum(axis=1, keepdims=True)

normalized\_cm = cm / row\_sums

…

history\_dict = history.history

loss\_values = history\_dict['loss']

val\_loss\_values = history\_dict['val\_loss']

acc = history\_dict['accuracy']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss\_values, 'b', label='Training Loss')

…

model\_save\_path = "F:/Dataset/archive/vit\_batch32\_earlyStop\_CategoricalCrossEntropy.keras"

model.save(model\_save\_path)”

* In this final section, we are defining a function to plot the confusion matrix to have a good visual on our results, for both train and test sets. This first part includes the characteristics of that visual. Alongside this, training history is also plotted for both sets to see loss and accuracy over epochs.
* Finally, we save the trained model to a path of our choice.