Josiah Lopez ViT Deliverables

Python Code:

#Cell 1

#Install necessary libraries

!pip install pytorch-lightning #Framework from Pytroch

!pip install transformers #Pre-trained model HUGGING FACE

!pip install torchmetrics #F1 and Loss and Accuracy

!pip install transformers torchmetrics

import torch #Core Library

import torch.nn as nn #Neural Network

from torchvision import transforms #Image Preprocessing

from transformers import ViTModel #import for hugging face

from torch.utils.data import DataLoader, Dataset #Handling datasets and batches

from PIL import Image

import os

from torchmetrics import Accuracy, F1Score #The Metrics

#Cell 2

#Define custom dataset for wildfire images

class SatelliteWildfireDataset(Dataset):

def \_\_init\_\_(self, image\_dir):

self.image\_dir = image\_dir #Root directory for image data

self.images = [] #Store image file paths

self.labels = [] #Store labels

categories = ['Smoke', 'Seaside', 'Land', 'Haze', 'Dust', 'Cloud']

#Loop through categories to import the images and assign the labels

for i in range(len(categories)):

category\_name = categories[i]

folder = os.path.join(image\_dir, category\_name) #Path to category folder

files = os.listdir(folder)

for file in files:

if file.endswith('.tif'): #Only process .tiff images

self.images.append(os.path.join(folder, file))

self.labels.append(i) #Assign numerical label based on category index

def \_\_len\_\_(self):

return len(self.images)

def \_\_getitem\_\_(self, index):

image\_path = self.images[index] #Get path of the image at the given indez

image = Image.open(image\_path).convert('RGB') #Open and convert to RGB

label = self.labels[index] #Get corresponding label

transform = transforms.Compose([

transforms.Resize((224, 224)), #Resize the image to fit the ViT's expected input size

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

])

image = transform(image)

return image, label

#Cell 3

#Define custom model using pre-trained ViT

class WildfireViTModel(nn.Module):

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

super(WildfireViTModel, self).\_\_init\_\_()

#Load pre-trained ViT

self.vit = ViTModel.from\_pretrained('google/vit-base-patch16-224')

#Freeze hugging face to avoid it from learning during the training, need code to speed up

for parameters in self.vit.parameters():

parameters.requires\_grad = False

#Custom Layers

self.extra\_layer = nn.Linear(768, 256) #Reduce ViT output from 768 to 256

self.relu = nn.ReLU() #Activation function for non-linearity

self.final\_layer = nn.Linear(256, 6) #Categorize from 256 to one of the 6 classes

def forward(self, input\_images):

outputs = self.vit(pixel\_values=input\_images) #Pass image through ViT

cls\_output = outputs.last\_hidden\_state[:, 0, :]

hidden = self.extra\_layer(cls\_output)

activated = self.relu(hidden) #Activate ReLU

logits = self.final\_layer(activated)

return logits

#Cell 4

from torchmetrics import Accuracy, F1Score

def train\_and\_evaluate(model, train\_loader, test\_loader, criterion, optimizer, epochs):

#Use GPU if available

if torch.cuda.is\_available():

model = model.cuda()

class\_count = 6 #Number of classes in the dataset

#Metrics for training and validation

train\_accuracy\_metric = Accuracy(task="multiclass", num\_classes=class\_count).to('cuda' if torch.cuda.is\_available() else 'cpu')

val\_accuracy\_metric = Accuracy(task="multiclass", num\_classes=class\_count).to('cuda' if torch.cuda.is\_available() else 'cpu')

f1\_metric = F1Score(task="multiclass", num\_classes=class\_count, average='macro').to('cuda' if torch.cuda.is\_available() else 'cpu')

#Training loop over number of epochs

for epoch in range(epochs):

model.train() #Set model to training mode

train\_loss = 0 #Track training loss

train\_accuracy\_metric.reset() #Rest accuracy metric

f1\_metric.reset() #Rest F1 Metric

#Iterate over training batches

for images, labels in train\_loader:

#Move to cuda if available

if torch.cuda.is\_available():

images = images.cuda()

labels = labels.cuda()

optimizer.zero\_grad()

outputs = model(images) #Forward pass

loss = criterion(outputs, labels) #Compute Loss

loss.backward() #Backpropagation

optimizer.step() #Update the weights

train\_loss += loss.item() #Gather loss

preds = torch.argmax(outputs, dim=1) #Get predictied class

train\_accuracy\_metric.update(preds, labels) #Update accuracy

f1\_metric.update(preds, labels) #Update F1 Score

#Calculate average metrics for training

avg\_train\_loss = train\_loss / len(train\_loader)

train\_accuracy = train\_accuracy\_metric.compute().item()

train\_f1 = f1\_metric.compute().item()

model.eval() #Put model in Evaluation mode

val\_loss = 0 #Track validation loss

val\_accuracy\_metric.reset() #Reset validation accuracy

f1\_metric.reset() #Rest F1 metric

#Validation loop

with torch.no\_grad():

for images, labels in test\_loader:

#Move to cuda if available

if torch.cuda.is\_available():

images = images.cuda()

labels = labels.cuda()

outputs = model(images) #Forward pass

loss = criterion(outputs, labels) #Compute loss

val\_loss += loss.item() #Accumulate loss

preds = torch.argmax(outputs, dim=1) #Get predicted class

val\_accuracy\_metric.update(preds, labels) #Update accuracy

f1\_metric.update(preds, labels) #New F1 score

#Compute average metrics for validation

avg\_val\_loss = val\_loss / len(test\_loader)

val\_accuracy = val\_accuracy\_metric.compute().item()

val\_f1 = f1\_metric.compute().item()

#Results printed to user for the epochs

print(f"Epoch {epoch + 1}:")

print(f" Train Loss: {avg\_train\_loss:.4f}, Train Accuracy: {train\_accuracy:.4f}, Train F1: {train\_f1:.4f}")

print(f" Val Loss: {avg\_val\_loss:.4f}, Val Accuracy: {val\_accuracy:.4f}, Val F1: {val\_f1:.4f}")

return model  
  
  
  
#Cell 5

from google.colab import drive

drive.mount('/content/drive') #Mount Google Drive to access data

image\_dir = "/content/drive/MyDrive/archive" #Directory of the image data

dataset = SatelliteWildfireDataset(image\_dir)

#Split data set into training and testing (80/20 split)

train\_size = int(0.8 \* len(dataset))

test\_size = len(dataset) - train\_size

train\_data, test\_data = torch.utils.data.random\_split(dataset, [train\_size, test\_size])

#Create data loaders for batching

train\_loader = DataLoader(train\_data, batch\_size=16, shuffle=True) #Shuffle for training

test\_loader = DataLoader(test\_data, batch\_size=16, shuffle=False) #No shuffle for testing

#ITS TIMEEEEEEE FOR REAAALLLLLLLLLLL

model = WildfireViTModel() #Initate the model

criterion = nn.CrossEntropyLoss() #Define the loss function

optimizer = torch.optim.Adam(model.parameters(), lr=0.001) #Define the optimizer

#TRAIN THIS CODE PLEAAASEEEEEEEEE

model = train\_and\_evaluate(model, train\_loader, test\_loader, criterion, optimizer, epochs=1) #You can change the epoch here. right now its set to 1 so it doesnt run forever

torch.save(model.state\_dict(), '/content/drive/MyDrive/wildfire\_model.pth')

print("Saved the model to my Drive!") #PLEASE SAVE  
  
  
  
#Cell 6

from google.colab import files

drive.mount('/content/drive') #Mount drive AGAIN to gain access to the saved model

model = WildfireViTModel() #Create new model

#Load saved weights into the model

model.load\_state\_dict(torch.load('/content/drive/MyDrive/wildfire\_model.pth', weights\_only=True)) #Location of where the model is

#Move to cuda if available

if torch.cuda.is\_available():

model = model.cuda()

model.eval()

print("Please upload an image to test!")#Message printed to user

uploaded = files.upload() #Prompt user to upload image

file\_name = list(uploaded.keys())[0] #Get then name of the uploaded file

print(f"Got your file: {file\_name}")

image = Image.open(file\_name).convert('RGB') #Open and conver the image to RGB

#Define transformation AGAIN (Same method that was used for training)

transform = transforms.Compose([

transforms.Resize((224, 224)), #Resize the ViT input size

transforms.ToTensor(), #Conver to tensor

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) #Normalize

])

image = transform(image) #Transform

image = image.unsqueeze(0) #Add batch dimension

if torch.cuda.is\_available():

image = image.cuda()

#Perform inference (inferenbce is when the saved model is loaded, the uploaded image is classified, and the result is printed)

with torch.no\_grad():

output = model(image) #Call the model

prediction = torch.argmax(output, dim=1).item() #Get class index

#Map predicition to category name

categories = ['Smoke', 'Seaside', 'Land', 'Haze', 'Dust', 'Cloud']

result = categories[prediction] #Get predicited category

print(f"This looks like: {result}!") #Print result/catergory to the user

os.remove(file\_name) #Delete the uploaded file from google collab

print(f"Deleted {file\_name} so it’s gone now.") #Let the user know that it was deleted

Test Script:

Step by step instruction to implement and use the code

1. Load the code “JosiahViTAttemptWildFire” into your preferred development platform.
2. Save the Archive folder somewhere on your computer or google collab
3. Change the image directory in the Execution cell to where you saved the Archive folder. This is marked with the note “Change Archive Location”
4. Change the location to save the model to somewhere on your computer or google drive. This can be found in the execution cell next to the note “Choose location to save model.
5. Afterwards you can start running the code. Make sure that you run the code starting from cell 1.
6. In cell 5 it is possible that Google Collab will ask you to access the Google Drive. If you have your archive and/or the model being saved to your google drive then you should allow that process to happen
7. When cell 5 completes cell 6 will run. It will prompt the user to import an image. Import an image from the “SELECTED DATA SET” folder. The images from this folder have not been given to the model for training.
8. After importing the image you should get a message that will say “This looks like:” and then it will tell you what it is your imported.

Additional notes:

* If you are training the model in google collab and you are not changing the number of epochs, expect it to take 5-6 hours to run. Do not let the tab disconnect from the internet nor let your computer turn off or go to sleep. If you do it is possible that you will need to start the training again.

Training Results:

A graph on a white sheet

AI-generated content may be incorrect.

In the graph above we track the loss over 5 epochs. The blue line represents training loss and we can see it decrease steadily from 0.5793 to 0.0944. This shows the model is learning well on the new training data. The validation loss is represented by the orange line and we can see that it starts at 0.3609, drops a little to 0.2611 in the fourth epoch, but then increases to 0.2955 by epoch 5. This suggest that the model improves its fit to the training data. It could be that is started overfitting after epoch 4 as the validation loss beings to rise.

A graph on a white sheet

AI-generated content may be incorrect.

In the graph above we it displays the accuracy over 5 epoch. The blue line represents the training accuracy. It increased from 0.7963 to 0.9662 by epoch 4 and then it dupped to 0.9663 in epoch 5, showing robust learning on the training data. Validation accuracy is represented by the orange line. It improves from 0.8643 to 0.9146 by epoch 4 and then its decreases to 0.9119 in epoch 5. The peak at epoch 4 and then a small decline in epoch 5 could suggest that the model generalizes well at first but then could have some overfitting.

A graph on a white sheet

AI-generated content may be incorrect.

In the graph above we have the F-1 Score over the same 5 epoch. We see that the training, represented by the blue line, for the F-1 score consistently improves through all 5 epoch. This tells us that there is a strong performance on the training set. The orange line represents the validation for the F-1 score and it also rises from 0.8624 to 0.9128 at epoch 4 however its drop off slightly at 0.9089 at epoch 5. What this means is that we have good generalization up to epoch 4 but then there could be some overfitting when it gets to epoch 5.

In summary from the three graphs we can see that the model learns effectively on the training data. In all three graphs we notice that there is a peak at epoch 4 and then it goes down at epoch 5. This shows some form of overfitting. Epoch 4 appears to be the optimal point for balancing the training and also the validation performance.