**CS 5588 Data Science Capstone**

**Hands-On (2/6)**

**Multi-Modal Transformer with Flask or Streamlit Application**

This hands-on session will help students build and deploy a multi-modal model using text, images, and numerical data relevant to their project. The goal is to adapt this framework to their own dataset and integrate different pre-trained models suitable for their project tasks.

**Overview**

* **Objective**: Implement a Transformer-based multi-modal model tailored to students' datasets.
* **Tools**: PyTorch, Hugging Face Transformers, Flask, Streamlit.
* **Dataset**: Students will use their own dataset.
* **Outcome**: A Flask API or a Streamlit-based application for real-time predictions.

**Agenda**

|  |
| --- |
| 1. Set up dataset relevant to the project |
| 1. Prepare multi-modal data (text, images, numerical data) |
| 1. Select and fine-tune a suitable model for each modality |
| 1. Build a Flask API or Streamlit UI |
| 1. Test and deploy the model |

**Step 1: Load Project Dataset**

Students should replace the following code with their own dataset.

import torch

import torchvision.transforms as transforms

from transformers import AutoTokenizer

from PIL import Image

import pandas as pd

# Load project dataset

df = pd.read\_csv("your\_project\_dataset.csv")

# Define relevant columns based on the dataset

TEXT\_COLUMN = "your\_text\_column"

IMAGE\_COLUMN = "your\_image\_column"

NUMERICAL\_COLUMNS = ["your\_numerical\_feature1", "your\_numerical\_feature2"]

LABEL\_COLUMN = "your\_label\_column"

# Tokenize text data

tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

def tokenize\_text(text):

return tokenizer(text, return\_tensors="pt", padding=True, truncation=True)

# Preprocess images

def preprocess\_image(image\_path):

transform = transforms.Compose([

transforms.Resize((224, 224)),

transforms.ToTensor(),

])

image = Image.open(image\_path).convert("RGB")

return transform(image).unsqueeze(0)

# Example usage

text\_tokens = tokenize\_text(df[TEXT\_COLUMN].iloc[0])

image\_tensor = preprocess\_image(df[IMAGE\_COLUMN].iloc[0])

print("Image Shape:", image\_tensor.shape, "Text Tokens:", text\_tokens)

**Step 2: Select and Fine-Tune Relevant Models**

Instead of using a fixed model, students will select models relevant to their project.

from transformers import AutoModel, VisionEncoderDecoderModel

import torch.nn as nn

class MultiModalModel(nn.Module):

def \_\_init\_\_(self, text\_model\_name, image\_model\_name, numerical\_input\_size, output\_classes):

super().\_\_init\_\_()

# Select appropriate text model

self.text\_encoder = AutoModel.from\_pretrained(text\_model\_name)

# Select appropriate image model

self.image\_encoder = torch.hub.load("pytorch/vision", image\_model\_name, pretrained=True)

self.image\_encoder.fc = nn.Identity()

# Numerical feature processing

self.fc\_numeric = nn.Linear(numerical\_input\_size, 128)

# Final classifier

self.fc\_combined = nn.Linear(512 + 768 + 128, output\_classes)

def forward(self, text\_tokens, image\_tensor, numerical\_data):

text\_features = self.text\_encoder(\*\*text\_tokens).last\_hidden\_state[:, 0, :]

image\_features = self.image\_encoder(image\_tensor)

numeric\_features = self.fc\_numeric(numerical\_data)

combined = torch.cat((text\_features, image\_features, numeric\_features), dim=1)

return self.fc\_combined(combined)

# Create the model instance with project-relevant choices

model = MultiModalModel(

text\_model\_name="bert-base-uncased",

image\_model\_name="resnet18",

numerical\_input\_size=len(NUMERICAL\_COLUMNS),

output\_classes=len(df[LABEL\_COLUMN].unique())

)

print("Model Ready:", model)

**Step 3: Save and Verify the Model**

Before deploying, students must ensure the trained model is saved properly.

import os

# Save the trained model

torch.save(model.state\_dict(), "multi\_modal\_model.pth")

print("Model saved successfully")

# Verify model file

if os.path.exists("multi\_modal\_model.pth"):

print("Model file found")

else:

print("Model file is missing Train and save it again")

**Step 4: Choose Flask API or Streamlit UI**

Students can choose one of the two deployment options.

**Option 1: Flask API Deployment**

Create a file named app.py.

from flask import Flask, request, jsonify

from flask\_cors import CORS

import torch

from PIL import Image

import torchvision.transforms as transforms

from transformers import AutoTokenizer

app = Flask(\_\_name\_\_)

CORS(app)

# Load trained model

model = MultiModalModel(

text\_model\_name="bert-base-uncased",

image\_model\_name="resnet18",

numerical\_input\_size=len(NUMERICAL\_COLUMNS),

output\_classes=len(df[LABEL\_COLUMN].unique())

)

if os.path.exists("multi\_modal\_model.pth"):

model.load\_state\_dict(torch.load("multi\_modal\_model.pth", map\_location=torch.device("cpu")))

model.eval()

print("Model loaded successfully")

else:

print("Model file not found Train and save the model first")

tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

@app.route("/predict", methods=["POST"])

def predict():

data = request.json

text\_tokens = tokenizer(data["symptoms"], return\_tensors="pt", padding=True, truncation=True)

image = Image.open(data["image\_path"]).convert("RGB")

transform = transforms.Compose([transforms.Resize((224, 224)), transforms.ToTensor()])

image\_tensor = transform(image).unsqueeze(0)

numerical\_data = torch.tensor([[data["heart\_rate"], data["temperature"]]], dtype=torch.float32)

with torch.no\_grad():

prediction = model(text\_tokens, image\_tensor, numerical\_data)

response = {"prediction": torch.argmax(prediction, dim=1).item()}

return jsonify(response)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(port=5000, debug=True)

Run Flask:

python app.py

**Option 2: Streamlit UI Deployment**

Create a file named streamlit\_app.py.

import streamlit as st

import torch

from PIL import Image

import torchvision.transforms as transforms

from transformers import AutoTokenizer

st.title("Multi-Modal Prediction")

# Load trained model

model = MultiModalModel(

text\_model\_name="bert-base-uncased",

image\_model\_name="resnet18",

numerical\_input\_size=len(NUMERICAL\_COLUMNS),

output\_classes=len(df[LABEL\_COLUMN].unique())

)

model.load\_state\_dict(torch.load("multi\_modal\_model.pth", map\_location=torch.device("cpu")))

model.eval()

tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

# Input fields

symptoms = st.text\_area("Enter Symptoms")

heart\_rate = st.number\_input("Heart Rate", min\_value=50, max\_value=200, value=80)

temperature = st.number\_input("Body Temperature", min\_value=30.0, max\_value=45.0, value=37.0)

image\_file = st.file\_uploader("Upload Image", type=["jpg", "png"])

if st.button("Predict"):

if symptoms and image\_file:

image\_path = "uploaded\_image.jpg"

with open(image\_path, "wb") as f:

f.write(image\_file.read())

# Process input

text\_tokens = tokenizer(symptoms, return\_tensors="pt", padding=True, truncation=True)

image = Image.open(image\_path).convert("RGB")

transform = transforms.Compose([transforms.Resize((224, 224)), transforms.ToTensor()])

image\_tensor = transform(image).unsqueeze(0)

numerical\_data = torch.tensor([[heart\_rate, temperature]], dtype=torch.float32)

with torch.no\_grad():

prediction = model(text\_tokens, image\_tensor, numerical\_data)

diagnosis = "Positive" if torch.argmax(prediction, dim=1).item() == 1 else "Negative"

st.success(f"Prediction: {diagnosis}")

else:

st.warning("Please enter symptoms and upload an image")

Run Streamlit:

streamlit run streamlit\_app.py

**Final Task**

* Test the model with project data
* Deploy Flask API or Streamlit UI
* Evaluate model performance

This hands-on session provides a customizable framework for students to integrate domain-specific models in their projects.