Configuration Manual

MSc Research Project
Data Analytics

Jayaram Vennapu
Student ID: x22219978

School of Computing
National College of Ireland

Supervisor: Shubham Subhnil

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Jayaram Vennapu		
Student ID:	X22219978		
Programme:	MSc Data Analytics	**Year:**	2024 - 25
Module:	MSc Research Project		
Lecturer:	Shubham Subhnil		
Submission Due Date:	12/12/2024		
Project Title:	Multimodal Fake News Detection: Integrating OCR and Deep Learning Models for Text and Image Analysis		
Word Count:	976	**Page Count:**	13

Signature: Jayaram Vennapu

Date: 12/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☐NA
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐NA
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐NA

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Jayaram vennapu

Student ID: x22219978

1. Introduction

This configuration guide details the experiment setup and results of this study on fake news identification by a dual Optical Character Recognition (OCR) and multimodal deep learning framework. The system incorporates text and image analysis in the detection of fake news with high precision through state-of-the art OCR, BERT based transformer and CNN-based ResNet models. This document has all details about the packages and software used, coupled with all configurations required that ensures this system provides experimental environment thereby similar results.

2. Deployment Environment

The data for this project is collected from Roboflow website.

Hardware Specification:

- Processor: Intel i5 3.60 GHz or equivalent
- RAM: 8.0 GB or higher
- GPU – NIVIDA RTX 3050 or equivalent (recommended for faster training of deep learning models).

This above-mentioned Hardware Specs based Local System was used to create the environment, to re-run the setup it is not necessary to have the same specification to re-create the environment, but it’s preferred to have these specs so that the model training can be done more precise and less time complexity.

2.1 Software Specification

2.1.1 Operating System: Windows 10/11 or Ubuntu 20.04+

2.1.2 Programming Language: Python version 3.1

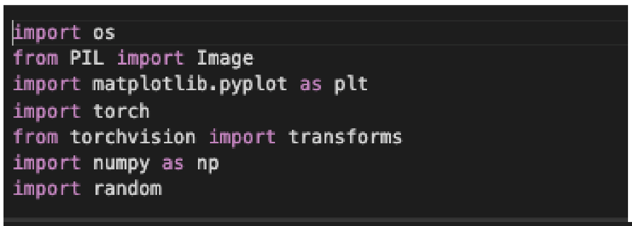
2.1.3 Integrated Development Environment (IDE): Google Colab for coding and running the models.

2.1.4 If you are running this from the local machine, please ensure to change the path, You can find this folder in the downloaded Dataset

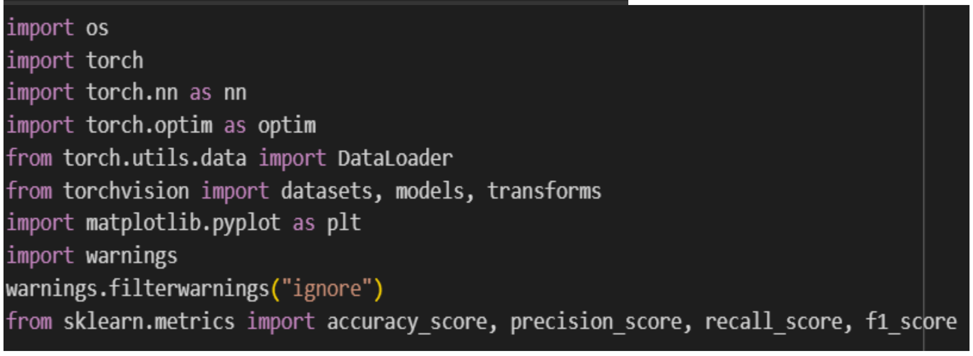
2.2 Python Libraries Required

Figure 2 shows the list of the necessary Python Libraries required for the execution of thecode. This mentioned python libraries can be installed using the pip command.

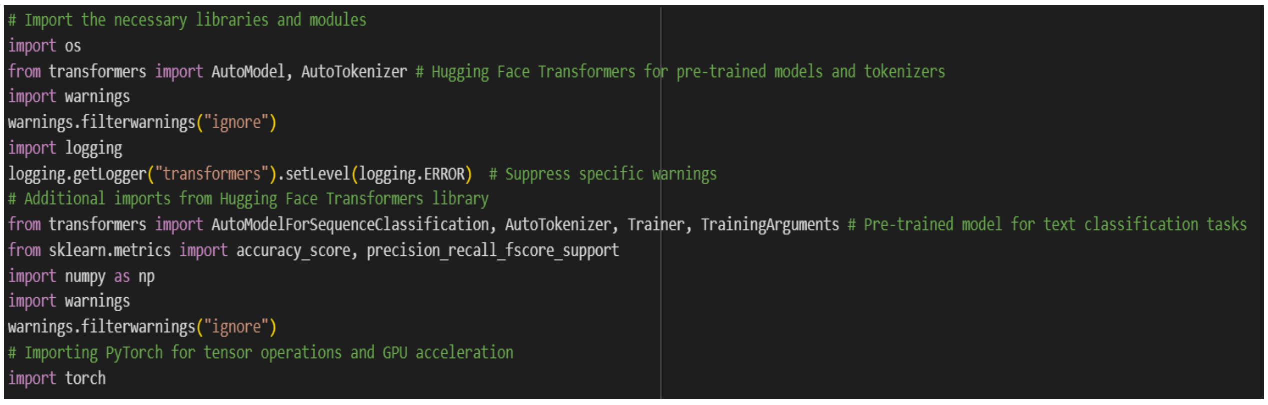
- Torch (torchvision)
- Pandas
- Scikit-learn
- Matplotlib
- Seaborn
- Numpy
- Keras
- Transformers



```
import os
from PIL import Image
import matplotlib.pyplot as plt
import torch
from torchvision import transforms
import numpy as np
import random
```



```
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```



```
# Import the necessary libraries and modules
import os
from transformers import AutoModel, AutoTokenizer # Hugging Face Transformers for pre-trained models and tokenizers
import warnings
warnings.filterwarnings("ignore")
import logging
logging.getLogger("transformers").setLevel(logging.ERROR)  # Suppress specific warnings
# Additional imports from Hugging Face Transformers library
from transformers import AutoModelForSequenceClassification, AutoTokenizer, Trainer, TrainingArguments # Pre-trained model for text classification tasks
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
import numpy as np
import warnings
warnings.filterwarnings("ignore")
# Importing PyTorch for tensor operations and GPU acceleration
import torch
```

Figure 1: Libraries Imported

3. Data Source

The dataset for this project is sourced from Roboflow's Fake News Image Classifier Dataset. This dataset contains images labelled as "real" or "fake," providing the basis for both text and image analysis.

- **Dataset Structure:**
  - **Images:** Labeled image files for real and fake news.
  - **Labels:** Metadata or annotations indicating the real/fake classification.

- **Data Splitting:**
  Divide the dataset as follows:
  - **Training Set:** 88% of the total dataset for training models.
  - **Validation Set:** 8% for fine-tuning hyperparameters.
  - **Test Set:** 4% for evaluating model performance.

- **Data Augmentation Techniques:**
  Augment images to improve model robustness:
  - Rotate within the plane.
  - Adjust saturation and exposure.
  - Apply random crops.
  - Resize images to 224 × 224 pixels.
  - **Pre-processing Text Data from Images:**
    Use Optical Character Recognition (OCR) to extract textual content from images. Process the extracted text to prepare it for tokenization and model training.

  Apply data augmentation techniques such as:
    - Image rotation
    - Saturation and exposure adjustment
    - Random cropping
    - Resizing images to 224 × 224 pixels

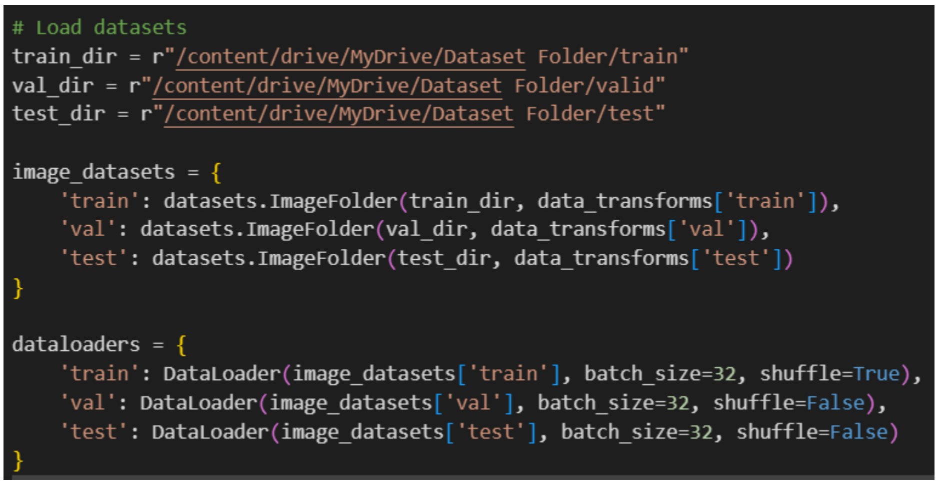
4. Project Code Files

- **Data Pre-processing:** Handles data augmentation, resizing, and OCR-based text extraction.
- **Text Analysis:** Prepares text data using BERT-base-uncased tokenizer and fine-tunes a BERT transformer model.
- **Image Analysis:** Implements ResNet-18 for image classification.
- **Multimodal Integration:** Combines text and image predictions for final fake news classification.

5. Data Preparation

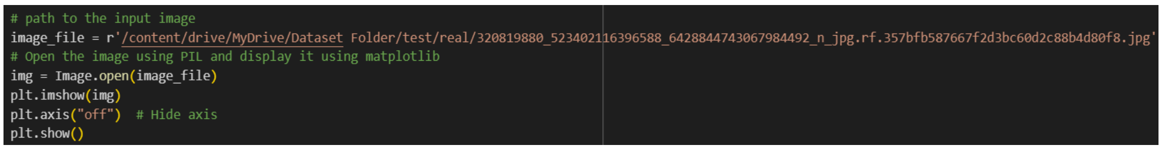
5.1 Extracting Data

Loading the datasets from file uploaded:



```
# Load datasets
train_dir = r"/content/drive/MyDrive/Dataset Folder/train"
val_dir = r"/content/drive/MyDrive/Dataset Folder/valid"
test_dir = r"/content/drive/MyDrive/Dataset Folder/test"

image_datasets = {
    'train': datasets.ImageFolder(train_dir, data_transforms['train']),
    'val': datasets.ImageFolder(val_dir, data_transforms['val']),
    'test': datasets.ImageFolder(test_dir, data_transforms['test'])
}

dataloaders = {
    'train': DataLoader(image_datasets['train'], batch_size=32, shuffle=True),
    'val': DataLoader(image_datasets['val'], batch_size=32, shuffle=False),
    'test': DataLoader(image_datasets['test'], batch_size=32, shuffle=False)
}
```

Figure 2: Loading the datasets

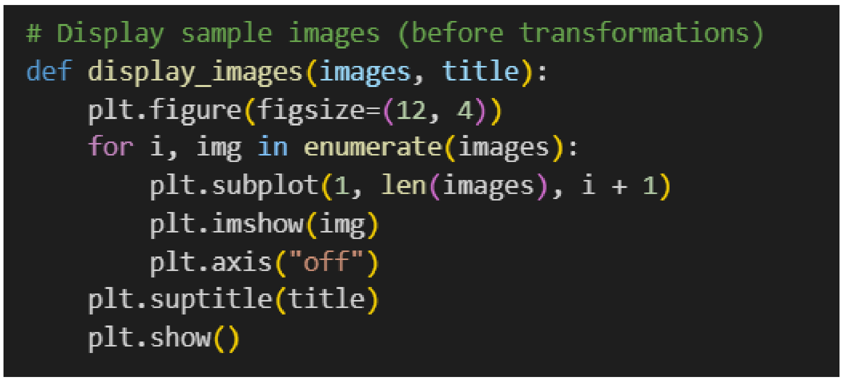


```
# path to the input image
image_file = r'/content/drive/MyDrive/Dataset Folder/test/real/320819880_523402116396588_6428844743067984492_n_jpg.rf.357bfb587667f2d3bc60d2c88b4d80f8.jpg'
# Open the image using PIL and display it using matplotlib
img = Image.open(image_file)
plt.imshow(img)
plt.axis("off")  # Hide axis
plt.show()
```

Figure 3: Importing test image for text extraction in OCR

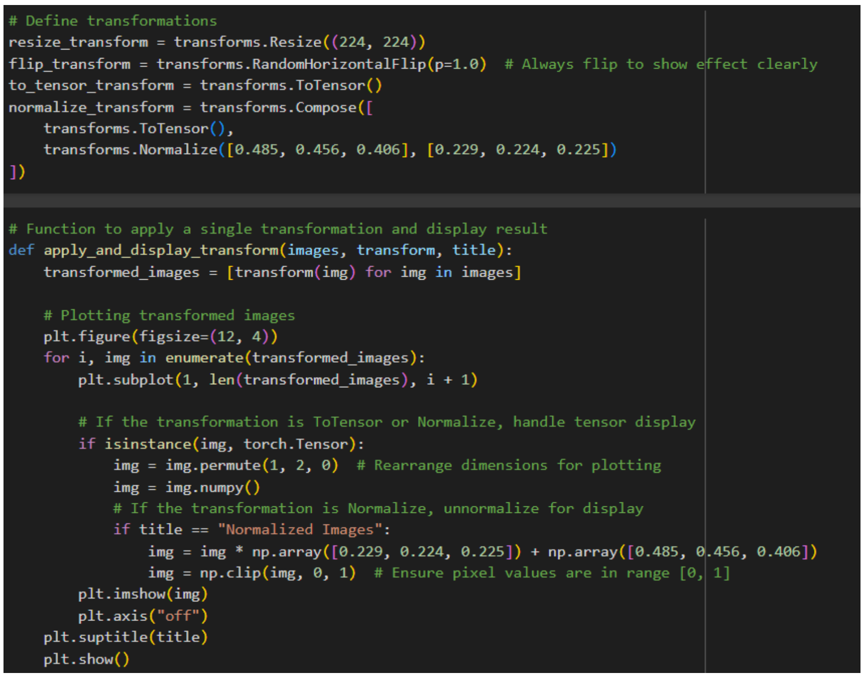
5.2 Data Pre-processing

- Handling Missing Values: Impute missing data using mean/median or interpolate.
- Data Separation: separating the data variables.
- This format is repeated for every model building code as well as EDA.



```
# Display sample images (before transformations)
def display_images(images, title):
    plt.figure(figsize=(12, 4))
    for i, img in enumerate(images):
        plt.subplot(1, len(images), i + 1)
        plt.imshow(img)
        plt.axis("off")
    plt.suptitle(title)
    plt.show()
```

Figure 4: To apply transformation



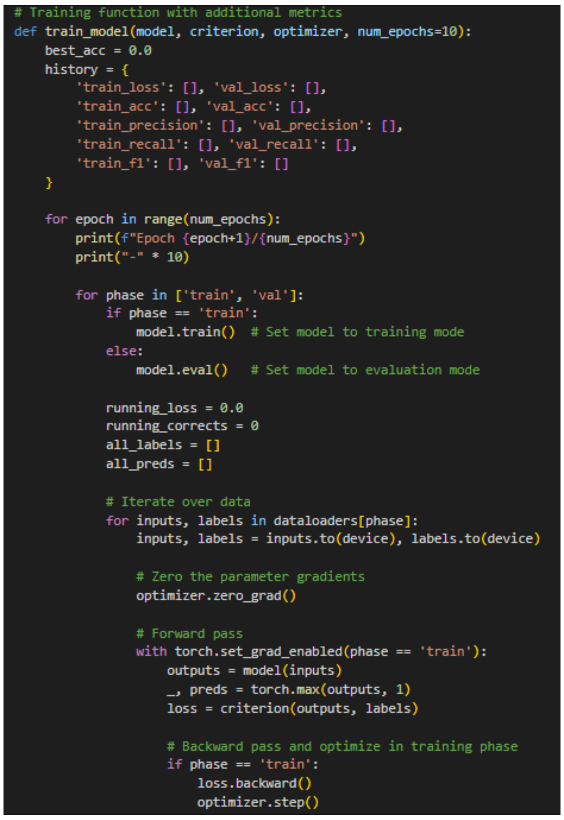
```python
# Define transformations
resize_transform = transforms.Resize((224, 224))
flip_transform = transforms.RandomHorizontalFlip(p=1.0)  # Always flip to show effect clearly
to_tensor_transform = transforms.ToTensor()
normalize_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

```python
# Function to apply a single transformation and display result
def apply_and_display_transform(images, transform, title):
    transformed_images = [transform(img) for img in images]

    # Plotting transformed images
    plt.figure(figsize=(12, 4))
    for i, img in enumerate(transformed_images):
        plt.subplot(1, len(transformed_images), i + 1)

        # If the transformation is ToTensor or Normalize, handle tensor display
        if isinstance(img, torch.Tensor):
            img = img.permute(1, 2, 0)  # Rearrange dimensions for plotting
            img = img.numpy()
            # If the transformation is Normalize, unnormalize for display
            if title == "Normalized Images":
                img = img * np.array([0.229, 0.224, 0.225]) + np.array([0.485, 0.456, 0.406])
                img = np.clip(img, 0, 1)  # Ensure pixel values are in range [0, 1]
        plt.imshow(img)
        plt.axis("off")
    plt.suptitle(title)
    plt.show()
```

Figure 5: Applying transformation on image

6. Model Building

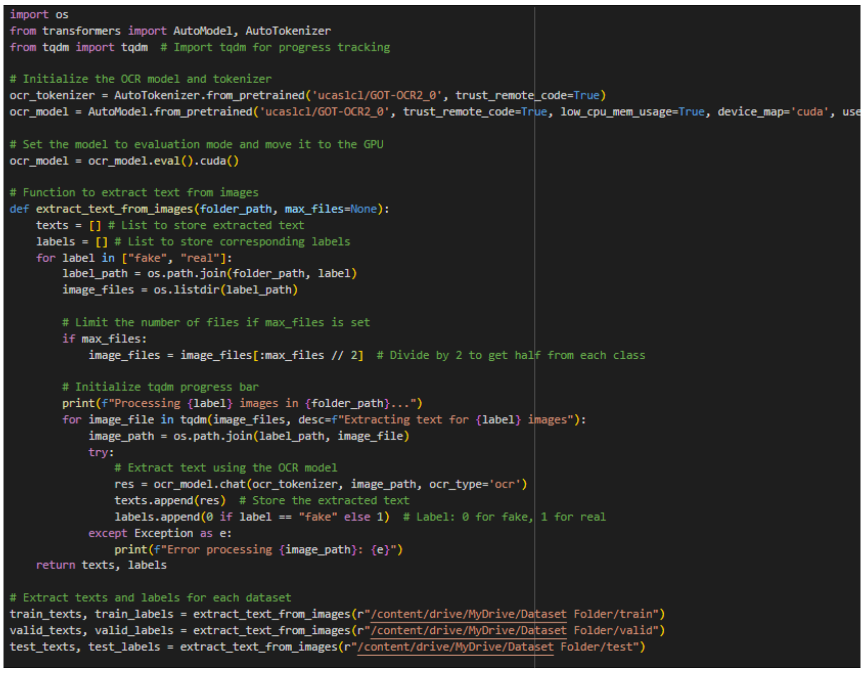
Image Analysis- Resnet Model:



```python
# Training function with additional metrics
def train_model(model, criterion, optimizer, num_epochs=10):
    best_acc = 0.0
    history = {
        'train_loss': [], 'val_loss': [],
        'train_acc': [], 'val_acc': [],
        'train_precision': [], 'val_precision': [],
        'train_recall': [], 'val_recall': [],
        'train_f1': [], 'val_f1': []
    }

    for epoch in range(num_epochs):
        print(f"Epoch {epoch+1}/{num_epochs}")
        print("-" * 10)

        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluation mode

            running_loss = 0.0
            running_corrects = 0
            all_labels = []
            all_preds = []

            # Iterate over data
            for inputs, labels in dataloaders[phase]:
                inputs, labels = inputs.to(device), labels.to(device)

                # Zero the parameter gradients
                optimizer.zero_grad()

                # Forward pass
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # Backward pass and optimize in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()
```

Figure 6: Model training code snippet for resnet

OCR & Tokenizer:

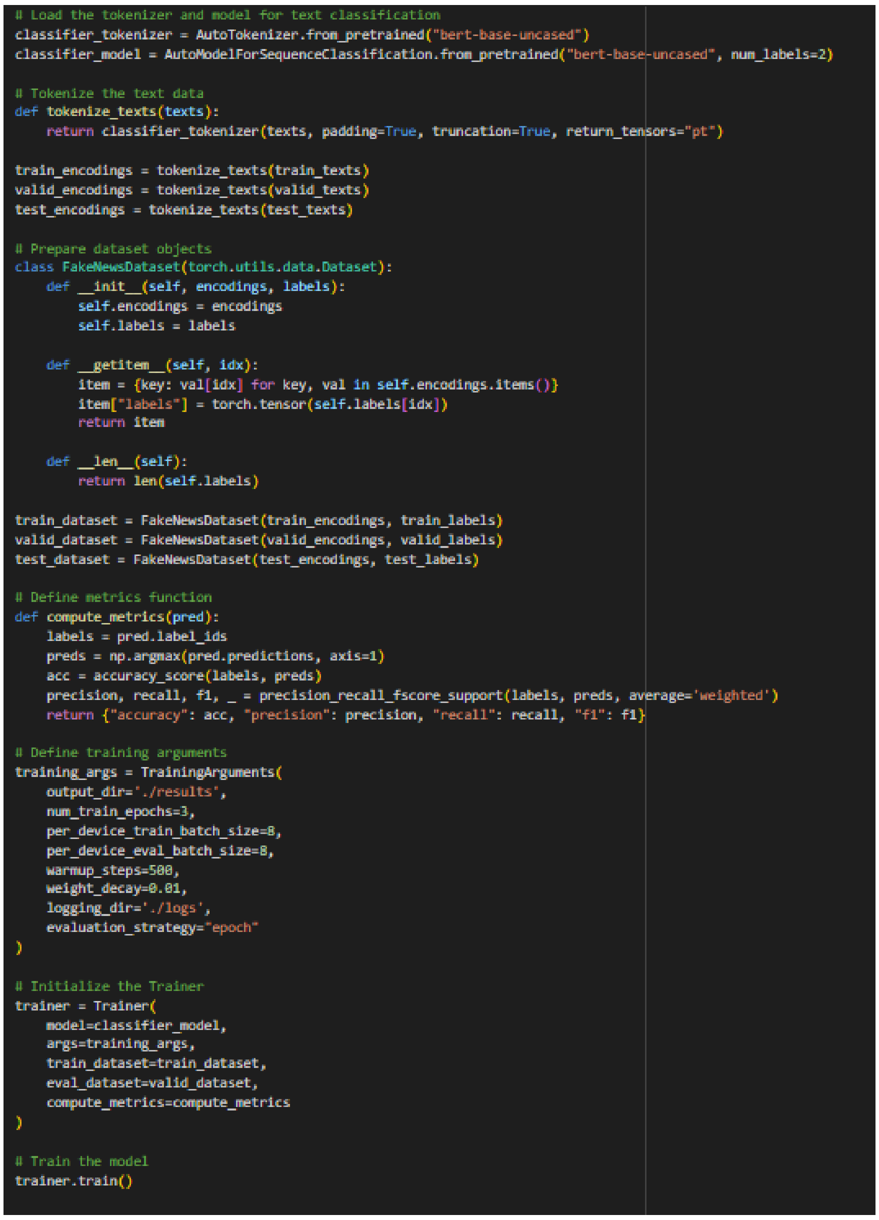


```python
import os
from transformers import AutoModel, AutoTokenizer
from tqdm import tqdm  # Import tqdm for progress tracking

# Initialize the OCR model and tokenizer
ocr_tokenizer = AutoTokenizer.from_pretrained('ucaslcl/GOT-OCR2_0', trust_remote_code=True)
ocr_model = AutoModel.from_pretrained('ucaslcl/GOT-OCR2_0', trust_remote_code=True, low_cpu_mem_usage=True, device_map='cuda', use

# Set the model to evaluation mode and move it to the GPU
ocr_model = ocr_model.eval().cuda()

# Function to extract text from images
def extract_text_from_images(folder_path, max_files=None):
    texts = [] # List to store extracted text
    labels = [] # List to store corresponding labels
    for label in ["fake", "real"]:
        label_path = os.path.join(folder_path, label)
        image_files = os.listdir(label_path)

        # Limit the number of files if max_files is set
        if max_files:
            image_files = image_files[:max_files // 2]  # Divide by 2 to get half from each class

        # Initialize tqdm progress bar
        print(f"Processing {label} images in {folder_path}...")
        for image_file in tqdm(image_files, desc=f"Extracting text for {label} images"):
            image_path = os.path.join(label_path, image_file)
            try:
                # Extract text using the OCR model
                res = ocr_model.chat(ocr_tokenizer, image_path, ocr_type='ocr')
                texts.append(res)  # Store the extracted text
                labels.append(0 if label == "fake" else 1)  # Label: 0 for fake, 1 for real
            except Exception as e:
                print(f"Error processing {image_path}: {e}")
    return texts, labels

# Extract texts and labels for each dataset
train_texts, train_labels = extract_text_from_images(r"/content/drive/MyDrive/Dataset Folder/train")
valid_texts, valid_labels = extract_text_from_images(r"/content/drive/MyDrive/Dataset Folder/valid")
test_texts, test_labels = extract_text_from_images(r"/content/drive/MyDrive/Dataset Folder/test")
```

Figure 7: OCR model with tokenizer and text with labels extraction

Models trained and used:

- Text analysis transformer- BERT transformer
- Image analysis- Resnet-18



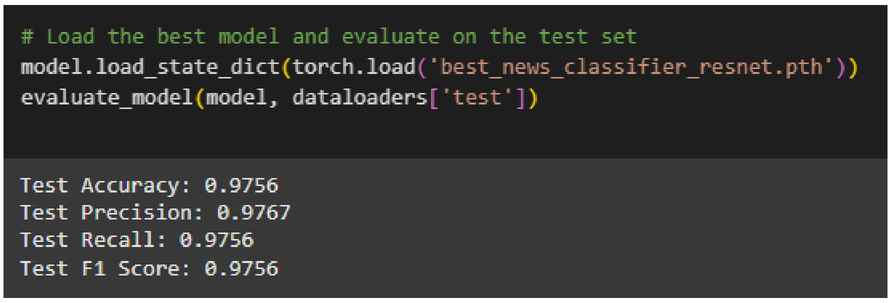
```
# Load the tokenizer and model for text classification
classifier_tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
classifier_model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2)

# Tokenize the text data
def tokenize_texts(texts):
    return classifier_tokenizer(texts, padding=True, truncation=True, return_tensors="pt")

train_encodings = tokenize_texts(train_texts)
valid_encodings = tokenize_texts(valid_texts)
test_encodings = tokenize_texts(test_texts)

# Prepare dataset objects
class FakeNewsDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: val[idx] for key, val in self.encodings.items()}
        item["labels"] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = FakeNewsDataset(train_encodings, train_labels)
valid_dataset = FakeNewsDataset(valid_encodings, valid_labels)
test_dataset = FakeNewsDataset(test_encodings, test_labels)

# Define metrics function
def compute_metrics(pred):
    labels = pred.label_ids
    preds = np.argmax(pred.predictions, axis=1)
    acc = accuracy_score(labels, preds)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average='weighted')
    return {"accuracy": acc, "precision": precision, "recall": recall, "f1": f1}

# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    evaluation_strategy="epoch"
)

# Initialize the Trainer
trainer = Trainer(
    model=classifier_model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    compute_metrics=compute_metrics
)

# Train the model
trainer.train()
```

Figure 8: Tokenization and Text Classification

7.3 Evaluation

Metrics Calculated:

1. Accuracy
2. F1 Score
3. Precision
4. Recall



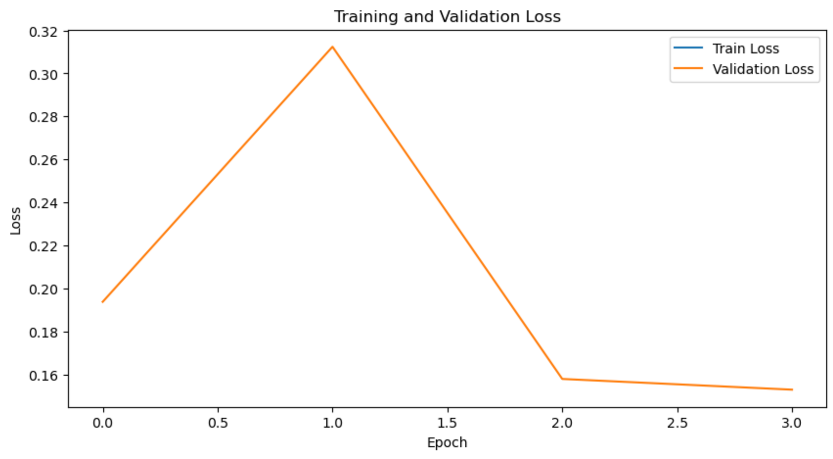
```
# Load the best model and evaluate on the test set
model.load_state_dict(torch.load('best_news_classifier_resnet.pth'))
evaluate_model(model, dataloaders['test'])
```

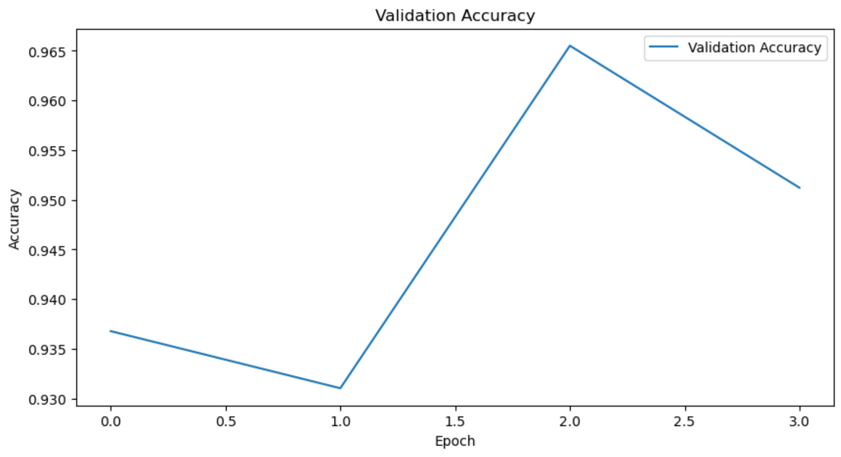
```
Test Accuracy: 0.9756
Test Precision: 0.9767
Test Recall: 0.9756
Test F1 Score: 0.9756
```

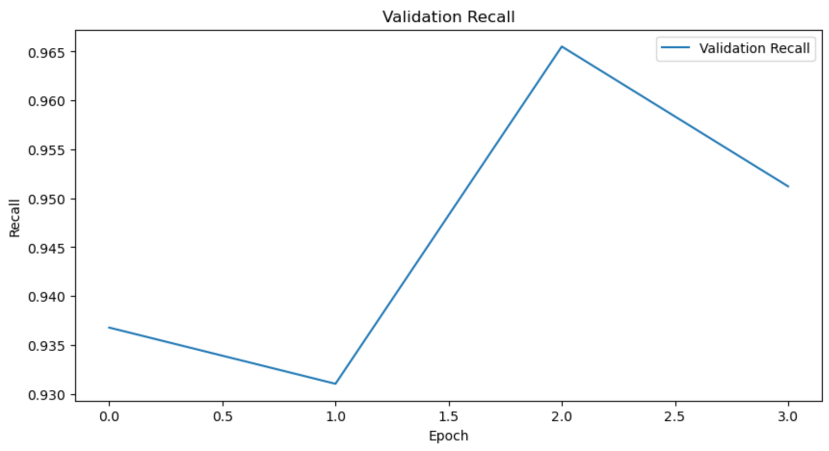
Figure 9: Results for Resnet

7. Results and Visualizations

- **Text Analysis:**
  BERT achieved a test accuracy of 95.12% and an F1-score of 95.12%.
- **Image Analysis:**
  ResNet-18 achieved a test accuracy of 97.56% and an F1-score of 97.56%.
- **Overall:**
  The multimodal system effectively integrates text and image predictions, achieving robust results in fake news detection.







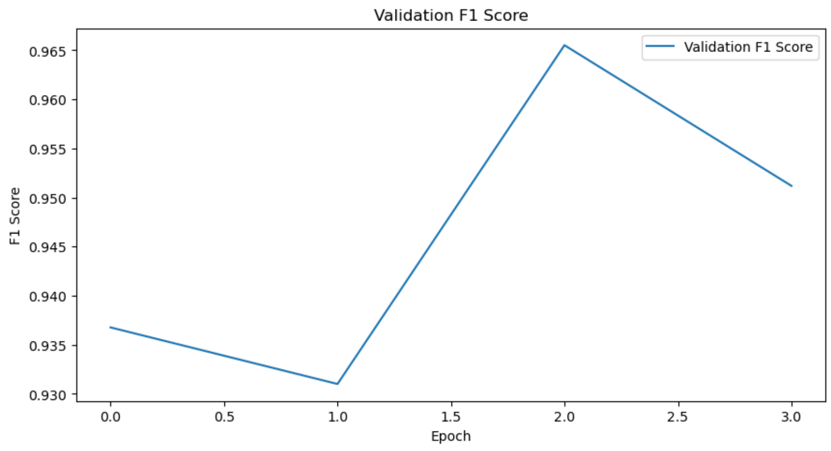
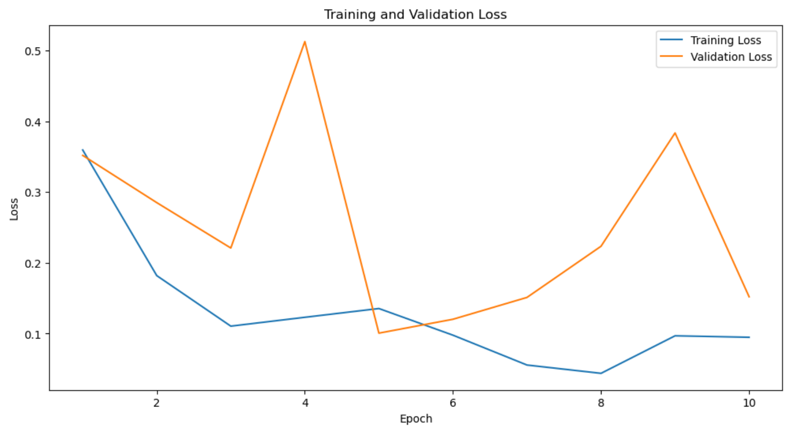
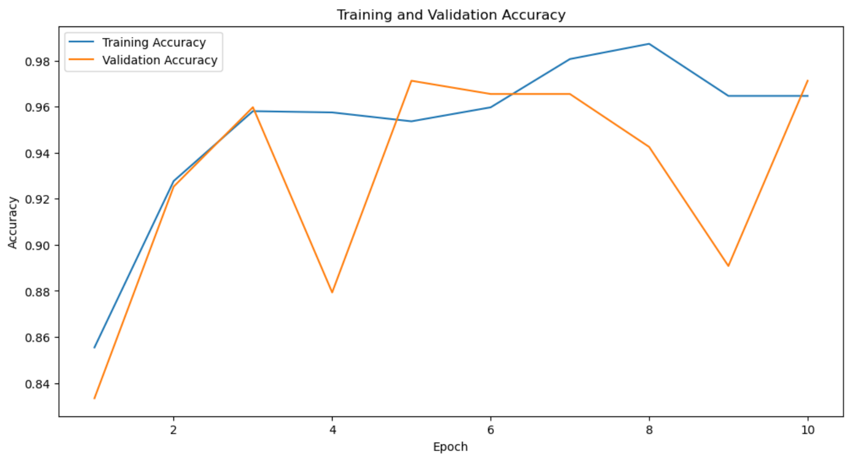
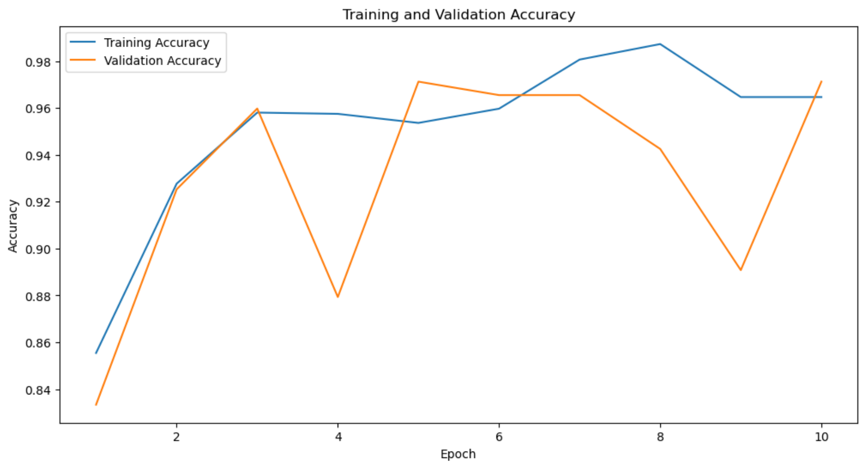
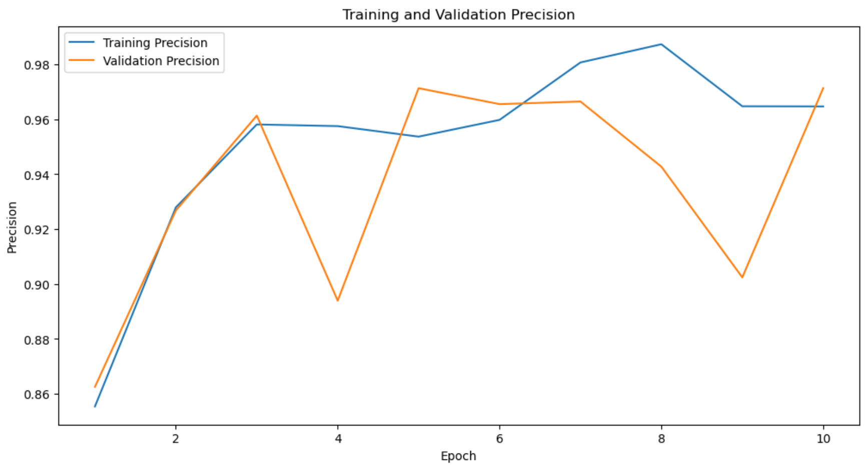


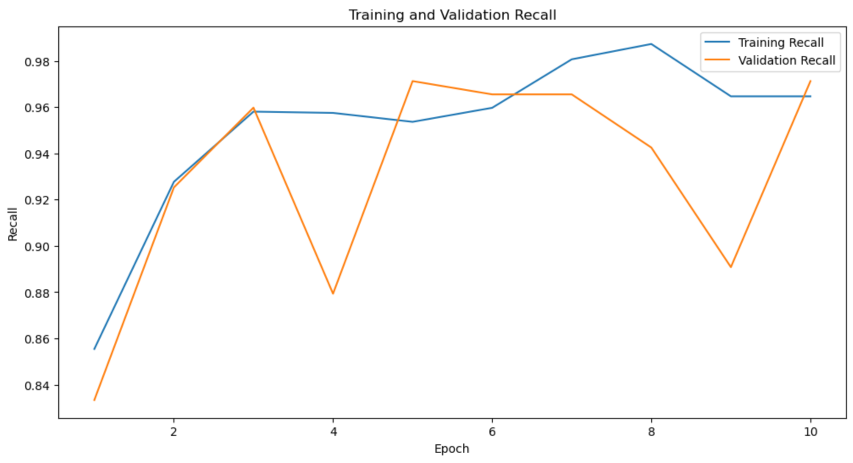
Figure 10: Training validation accuracy, loss, precision, recall for OCR




Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
0.84
0.86
0.88
0.90
0.92
0.94
0.96
0.98
2
4
6
8
10


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
0.84
0.86
0.88
0.90
0.92
0.94
0.96
0.98
2
4
6
8
10


Training and Validation Precision
Training Precision
Validation Precision
Precision
Epoch
0.86
0.88
0.90
0.92
0.94
0.96
0.98
2
4
6
8
10



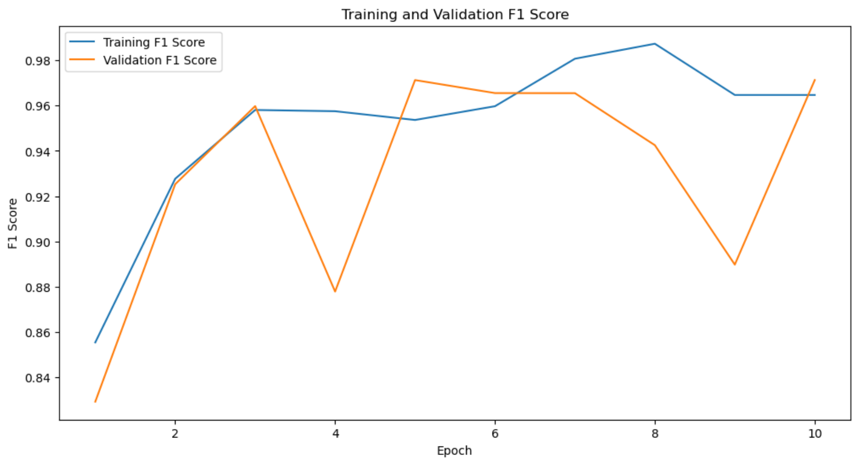


Figure 11: Resulting visualization for Training and validation loss, accuracy, precision, recall and f1 score (Resnet model)