Code

January 21, 2025

1 MECH 3465 Robotics & Machine Intelligence - Coursework 1

Dan Nehushtan - 201594650 Louie Burns - 201588498

1.0.1 Introduction

This coursework applies Convolutional Neural Networks (CNNs) to classify aircraft in remote sensing images, optimising performance through hyperparameter tuning and pre-processing. The evaluation uses the weighted F1-score to assess model improvements.

1.0.2 Tasks

1. Design a convolutional neural network (CNN) for aircraft classification, justifying its suitability for the task.
2. Document the hyperparameter tuning process and analyse its impact on model performance.
3. Optimise the CNN through pre-processing techniques or alternative network structures, using weighted F1-score for evaluation.
4. Provide a detailed evaluation, comparing the initial and optimised models, including metrics and a critical reflection on results.

1.1 1. Dataset Preparation

1.1.1 1.1 Importing Libraries

[1]:
```
# INSTALL MISSING LIBRARIES (This depends on device and GPU used - some stuff␣
 ↪may already be satisfied, some may not)
%pip install datasets --quiet >NUL 2>&1
%pip install pandas --quiet >NUL 2>&1
%pip install scikit-learn --quiet >NUL 2>&1
%pip install matplotlib --quiet >NUL 2>&1
%pip install nbconvert

# Import libraries used
import os
import time
import torch
from torch import nn, optim
from torch.utils.data import TensorDataset, DataLoader, Dataset
from torchvision import transforms
```

```
from torchvision.transforms.functional import crop, resize, to_tensor
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score
from collections import Counter
```

```
Note: you may need to restart the kernel to use updated packages.
Note: you may need to restart the kernel to use updated packages.
Note: you may need to restart the kernel to use updated packages.
Note: you may need to restart the kernel to use updated packages.
```

1.1.2 1.2 Prepare the device and load the model

[2]:
```
## check devices

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# If CUDA is available, print the GPU name and the device being used
if torch.cuda.is_available():
    # Get the name of the current GPU
    gpu_name = torch.cuda.get_device_name(device)
    print(f"CUDA is available. Using GPU: {gpu_name}")
else:
    print("CUDA is not available. Using CPU.")
```

```
CUDA is available. Using GPU: NVIDIA GeForce RTX 2060 SUPER
```

1.1.3 1.3 Loading and Preprocessing the Dataset

- Load the AircraftRecognition Dataset
- Split into training and testing segments
- The training segment is used for training the model, while the testing portion of the data is used to evaluate the accuracy of the model.

The dataset is extracted to the directory /AircraftRecognitionDataset. You need to create a local dirctory data folder. It contains 1 folder, images, containing all train and test set (10,000 images). Also it contains 19 text files which categorise the images. Verify this using os.listdir.

[3]:
```
import zipfile

# Define the path to the zip file
zip_file_path = 'AircraftRecognitionDataset.zip'
extracted_folder_path = 'AircraftRecognitionDataset'  # Folder where you want to␣
↪extract

# Check if the folder exists, if not, create it
if not os.path.exists(extracted_folder_path):
    os.makedirs(extracted_folder_path)
```

```
# Extract the zip file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extracted_folder_path)

print(f'Zip file extracted to: {extracted_folder_path}')

# Set the path to the extracted 'dataoriginal' folder
data_dir = os.path.join(extracted_folder_path, 'dataoriginal')  # Path to the␣
 ↪dataset folder inside extracted

# List the text files in 'dataoriginal'
txt_files = [f for f in os.listdir(data_dir) if f.endswith(".txt")]
print("Text files found:", len(txt_files))
```

```
Zip file extracted to: AircraftRecognitionDataset
Text files found: 19
```

This depends on where the dataset is located locally.

[4]:
```
# Define the folder containing the images
image_folder = './AircraftRecognitionDataset/dataoriginal/images'

# Paths to the text files
train_label_file = './AircraftRecognitionDataset/dataoriginal/
 ↪images_manufacturer_train.txt'
test_label_file = './AircraftRecognitionDataset/dataoriginal/
 ↪images_manufacturer_test.txt'
val_label_file = './AircraftRecognitionDataset/dataoriginal/
 ↪images_manufacturer_trainval.txt'
```

[5]:
```
# Define label mapping
label_mapping = {
    "Boeing": 0,
    "Airbus": 1,
    "ATR": 2,
    "Cessna": 3,
    "Embraer": 4
}

# Define transformations for image processing
transform = transforms.Compose([
    transforms.Resize((64, 64)),  # Resize to 64x64
    transforms.ToTensor(),          # Convert to tensor
])

def filter_data(label_file, selected_labels):
    image_data = []
    labels = []
```

```
    with open(label_file, "r") as f:
        for line in f:
            parts = line.strip().split(maxsplit=1)
            if len(parts) != 2:
                continue  # Skip malformed lines
            filename, label = parts
            if label in selected_labels:
                image_path = os.path.join(image_folder, filename + ".jpg")
                try:
                    if os.path.exists(image_path):
                        image = Image.open(image_path).convert("RGB")
                        image_tensor = transform(image)
                        image_data.append(image_tensor)
                        labels.append(selected_labels[label])
                except Exception as e:
                    print(f"Error processing {image_path}: {e}")

    # Convert lists to PyTorch tensors
    image_tensor = torch.stack(image_data)
    label_tensor = torch.tensor(labels, dtype=torch.long)

    return image_tensor, label_tensor, labels
```

1.1.4 1.3 Exploratory Data Analysis (EDA)

Visualise class distributions, dataset sizes, and other statistics.

[6]:
```
# Process training, validation, and test data for selected labels
train_image_tensor, train_label_tensor, train_labels =␣
 ↪filter_data(train_label_file, label_mapping)
val_image_tensor, val_label_tensor, val_labels = filter_data(val_label_file,␣
 ↪label_mapping)
test_image_tensor, test_label_tensor, test_labels = filter_data(test_label_file,␣
 ↪label_mapping)

# Print dataset sizes
print(f"Number of training images: {len(train_image_tensor)}")
print(f"Number of validation images: {len(val_image_tensor)}")
print(f"Number of test images: {len(test_image_tensor)}")

# Print counts for each class in training, validation, and test datasets
train_counts = Counter(train_labels)
val_counts = Counter(val_labels)
test_counts = Counter(test_labels)

print("Training dataset class distribution:")
```

```
for label, count in train_counts.items():
    print(f"{list(label_mapping.keys())[list(label_mapping.values()).
 ↪index(label)]}: {count}")

print("Validation dataset class distribution:")
for label, count in val_counts.items():
    print(f"{list(label_mapping.keys())[list(label_mapping.values()).
 ↪index(label)]}: {count}")

print("Test dataset class distribution:")
for label, count in test_counts.items():
    print(f"{list(label_mapping.keys())[list(label_mapping.values()).
 ↪index(label)]}: {count}")

# Create TensorDatasets
train_dataset = TensorDataset(train_image_tensor, train_label_tensor)
val_dataset = TensorDataset(val_image_tensor, val_label_tensor)
test_dataset = TensorDataset(test_image_tensor, test_label_tensor)

# Example usage of DataLoader
batch_size = 64
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True,␣
 ↪drop_last=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False,␣
 ↪drop_last=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False,␣
 ↪drop_last=True)

print("DataLoaders for selected labels created successfully.")
```

```
Number of training images: 1599
Number of validation images: 3199
Number of test images: 1601
Training dataset class distribution:
Boeing: 733
Airbus: 434
ATR: 66
Cessna: 133
Embraer: 233
Validation dataset class distribution:
Boeing: 1466
Airbus: 867
ATR: 133
Cessna: 266
Embraer: 467
Test dataset class distribution:
Boeing: 734
```

```
Airbus: 433
ATR: 67
Cessna: 134
Embraer: 233
DataLoaders for selected labels created successfully.
```

1.2 2. Model Development

1.2.1 2.1 Initial Model Design

Define the initial CNN architecture and explain the design choices.

[7]:
```python
# Define a simplified CNN model
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(16 * 32 * 32, len(label_mapping))
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = x.view(x.size(0), -1)  # Flatten
        x = self.fc1(x)
        return x

# Instantiate the simplified model
model = SimpleCNN()
```

1.2.2 2.2 Training Process

Train the initial model and validate its performance.

[8]:
```python
# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Training loop
num_epochs = 15
train_losses, test_losses = [], []
train_accuracies, test_accuracies = [], []
f1_scores = []

for epoch in range(num_epochs):
    # Training phase
    model.train()
    running_loss = 0.0
    correct_train = 0
```

```
    total_train = 0
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()

    train_accuracy = correct_train / total_train

    # Validation phase
    model.eval()
    test_loss = 0.0
    correct_test = 0
    total_test = 0
    true_labels = []
    predicted_labels = []
    with torch.no_grad():
        for images, labels in test_loader:
            outputs = model(images)
            loss = criterion(outputs, labels)
            test_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total_test += labels.size(0)
            correct_test += (predicted == labels).sum().item()
            true_labels.extend(labels.cpu().numpy())
            predicted_labels.extend(predicted.cpu().numpy())

    test_accuracy = correct_test / total_test
    f1 = f1_score(true_labels, predicted_labels, average='weighted')

    # Store metrics
    train_losses.append(running_loss / len(train_loader))
    test_losses.append(test_loss / len(test_loader))
    train_accuracies.append(train_accuracy)
    test_accuracies.append(test_accuracy)
    f1_scores.append(f1)

    print(f"Epoch [{epoch+1}/{num_epochs}], Train Loss: {running_loss/
↪len(train_loader):.4f}, Test Loss: {test_loss/len(test_loader):.4f}, Train Acc:
↪ {train_accuracy:.4f}, Test Acc: {test_accuracy:.4f}, F1 Score: {f1:.4f}")

print("Training complete.")
```

```
# Plotting results
epochs = range(1, num_epochs + 1)
```

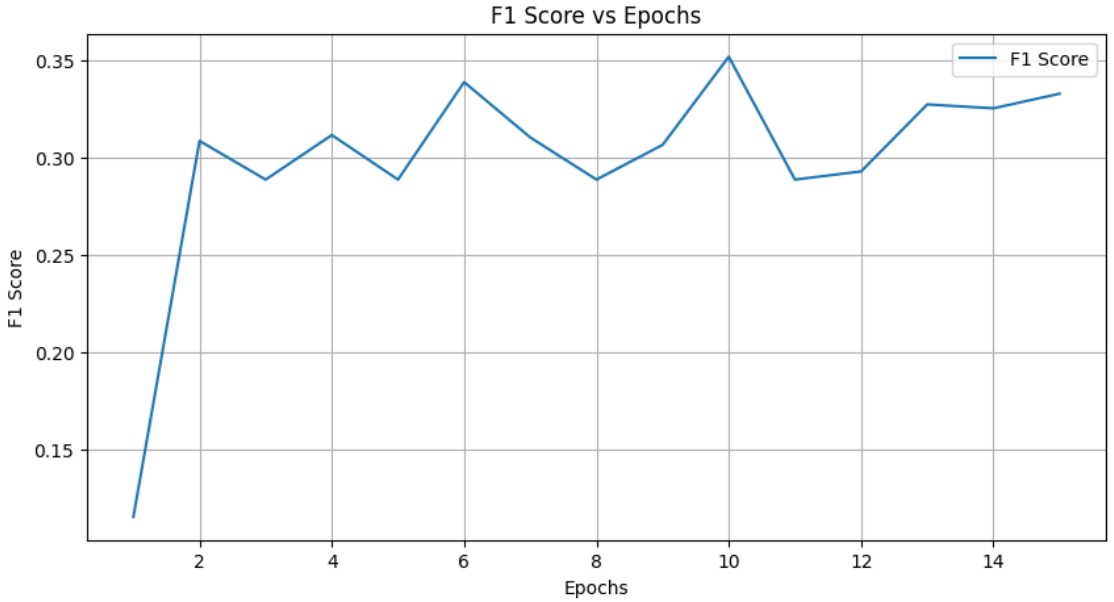
```
Epoch [1/15], Train Loss: 1.4191, Test Loss: 1.3897, Train Acc: 0.4043, Test
Acc: 0.2706, F1 Score: 0.1153
Epoch [2/15], Train Loss: 1.3495, Test Loss: 1.3638, Train Acc: 0.4329, Test
Acc: 0.3569, F1 Score: 0.3084
Epoch [3/15], Train Loss: 1.3286, Test Loss: 1.3268, Train Acc: 0.4512, Test
Acc: 0.4587, F1 Score: 0.2885
Epoch [4/15], Train Loss: 1.3249, Test Loss: 1.3296, Train Acc: 0.4603, Test
Acc: 0.4581, F1 Score: 0.3114
Epoch [5/15], Train Loss: 1.3173, Test Loss: 1.3229, Train Acc: 0.4577, Test
Acc: 0.4587, F1 Score: 0.2885
Epoch [6/15], Train Loss: 1.3157, Test Loss: 1.3440, Train Acc: 0.4531, Test
Acc: 0.3956, F1 Score: 0.3386
Epoch [7/15], Train Loss: 1.3105, Test Loss: 1.3284, Train Acc: 0.4531, Test
Acc: 0.4581, F1 Score: 0.3102
Epoch [8/15], Train Loss: 1.3052, Test Loss: 1.3142, Train Acc: 0.4668, Test
Acc: 0.4587, F1 Score: 0.2885
Epoch [9/15], Train Loss: 1.2976, Test Loss: 1.3127, Train Acc: 0.4616, Test
Acc: 0.4612, F1 Score: 0.3064
Epoch [10/15], Train Loss: 1.2908, Test Loss: 1.3179, Train Acc: 0.4577, Test
Acc: 0.4406, F1 Score: 0.3516
Epoch [11/15], Train Loss: 1.2863, Test Loss: 1.3245, Train Acc: 0.4681, Test
Acc: 0.4587, F1 Score: 0.2885
Epoch [12/15], Train Loss: 1.2771, Test Loss: 1.3036, Train Acc: 0.4694, Test
Acc: 0.4581, F1 Score: 0.2927
Epoch [13/15], Train Loss: 1.2653, Test Loss: 1.3051, Train Acc: 0.4753, Test
Acc: 0.4650, F1 Score: 0.3271
Epoch [14/15], Train Loss: 1.2671, Test Loss: 1.2994, Train Acc: 0.4727, Test
Acc: 0.4650, F1 Score: 0.3252
Epoch [15/15], Train Loss: 1.2665, Test Loss: 1.2960, Train Acc: 0.4772, Test
Acc: 0.4644, F1 Score: 0.3326
Training complete.
```

1.2.3 2.3 Metrics for Initial Model

Visualise the performance of the initial model using metrics like accuracy and F1-score.

[9]:
```
# Plot Loss
plt.figure(figsize=(10, 5))
plt.plot(epochs, train_losses, label='Training Loss')
plt.plot(epochs, test_losses, label='Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Test Loss vs Epochs')
plt.legend()
```

```
plt.grid(True)
plt.show()

# Plot Accuracy
plt.figure(figsize=(10, 5))
plt.plot(epochs, train_accuracies, label='Training Accuracy')
plt.plot(epochs, test_accuracies, label='Test Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Test Accuracy vs Epochs')
plt.legend()
plt.grid(True)
plt.show()

# Plot F1 Score
plt.figure(figsize=(10, 5))
plt.plot(epochs, f1_scores, label='F1 Score')
plt.xlabel('Epochs')
plt.ylabel('F1 Score')
plt.title('F1 Score vs Epochs')
plt.legend()
plt.grid(True)
plt.show()
```



1.3 3. Model Optimisation

1.3.1 3.1 Hyperparameter Tuning

Adjust hyperparameters such as learning rate, batch size, and architecture details.

First we will increase the number of manufacturers mapped for training.

[10]:
```
# Define label mapping
label_mapping = {
    "Boeing": 0,
    "Airbus": 1,
    "ATR": 2,
    "Antonov": 3,
    "BritishAerospace": 4,
    "Beechcraft": 5,
    "LockheedCorporation": 6,
    "DouglasAircraftCompany": 7,
    "Canadair": 8,
    "Cessna": 9,
    "McDonnellDouglas": 10,
    "deHavilland": 11,
    "Robin": 12,
    "Dornier": 13,
    "Embraer": 14,
    "Eurofighter": 15,
    "LockheedMartin": 16,
    "DassaultAviation": 17,
    "Fokker": 18,
    "BombardierAerospace": 19,
    "GulfstreamAerospace": 20,
    "Ilyushin": 21,
    "Fairchild": 22,
    "Piper": 23,
    "CirrusAircraft": 24,
    "Saab": 25,
    "Supermarine": 26,
    "Panavia": 27,
    "Tupolev": 28,
    "Yakovlev": 29
}

def select_balanced_classes(train_file, val_file, test_file, label_mapping,␣
 ↪num_classes=5):
    def count_labels(file_path):
        counts = Counter()
        with open(file_path, "r") as f:
            for line in f:
                _, label = line.strip().split(maxsplit=1)
                if label in label_mapping:
                    counts[label] += 1
        return counts

    train_counts = count_labels(train_file)
    val_counts = count_labels(val_file)
```

```
    test_counts = count_labels(test_file)

    # Combine counts for all datasets
    combined_counts = {label: train_counts[label] + val_counts[label] +␣
↪test_counts[label] for label in label_mapping}

    # Sort by total count and select top labels
    selected_labels = dict(sorted(combined_counts.items(), key=lambda item:␣
↪item[1], reverse=True)[:num_classes])

    # Map selected labels to indices
    selected_labels = {label: idx for idx, (label, _) in␣
↪enumerate(selected_labels.items())}

    return selected_labels

# Intelligently select classes based on data distribution
selected_labels = select_balanced_classes(train_label_file, val_label_file,␣
↪test_label_file, label_mapping)
```

[11]:
```
# Define transformations for image processing
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),  # Random horizontal flip
    transforms.RandomRotation(5),     # Random rotation up to 15 degrees
    transforms.Resize((128, 128)),  # Resize to 128x128
    transforms.ToTensor(),           # Convert to tensor
    transforms.Normalize([0.5], [0.5])  # Normalize
])

def filter_data(label_file, selected_labels):
    image_data = []
    labels = []

    with open(label_file, "r") as f:
        for line in f:
            parts = line.strip().split(maxsplit=1)
            if len(parts) != 2:
                continue  # Skip malformed lines
            filename, label = parts
            if label in selected_labels:
                image_path = os.path.join(image_folder, filename + ".jpg")
                try:
                    if os.path.exists(image_path):
                        image = Image.open(image_path).convert("RGB")
                        image_tensor = transform(image)
                        image_data.append(image_tensor)
                        labels.append(selected_labels[label])
```

```
                except Exception as e:
                    print(f"Error processing {image_path}: {e}")

    # Convert lists to PyTorch tensors
    image_tensor = torch.stack(image_data)
    label_tensor = torch.tensor(labels, dtype=torch.long)

    return image_tensor, label_tensor, labels


# Process training, validation, and test data for selected labels
train_image_tensor, train_label_tensor, train_labels =␣
↪filter_data(train_label_file, selected_labels)
val_image_tensor, val_label_tensor, val_labels = filter_data(val_label_file,␣
↪selected_labels)
test_image_tensor, test_label_tensor, test_labels = filter_data(test_label_file,␣
↪selected_labels)

# Print dataset sizes
print(f"Number of training images: {len(train_image_tensor)}")
print(f"Number of validation images: {len(val_image_tensor)}")
print(f"Number of test images: {len(test_image_tensor)}")

# Print counts for each class in training, validation, and test datasets
train_counts = Counter(train_labels)
val_counts = Counter(val_labels)
test_counts = Counter(test_labels)

print("Training dataset class distribution:")
for label, count in train_counts.items():
    print(f"{list(selected_labels.keys())[list(selected_labels.values()).
↪index(label)]}: {count}")

print("Validation dataset class distribution:")
for label, count in val_counts.items():
    print(f"{list(selected_labels.keys())[list(selected_labels.values()).
↪index(label)]}: {count}")

print("Test dataset class distribution:")
for label, count in test_counts.items():
    print(f"{list(selected_labels.keys())[list(selected_labels.values()).
↪index(label)]}: {count}")

# Create TensorDatasets
train_dataset = TensorDataset(train_image_tensor, train_label_tensor)
val_dataset = TensorDataset(val_image_tensor, val_label_tensor)
```

```
test_dataset = TensorDataset(test_image_tensor, test_label_tensor)

# Example usage of DataLoader
batch_size = 32  # Increased batch size
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
print("DataLoaders for selected labels created successfully.")
```

```
Number of training images: 1667
Number of validation images: 3333
Number of test images: 1667
Training dataset class distribution:
Boeing: 733
Airbus: 434
Canadair: 134
Cessna: 133
Embraer: 233
Validation dataset class distribution:
Boeing: 1466
Airbus: 867
Canadair: 267
Cessna: 266
Embraer: 467
Test dataset class distribution:
Boeing: 734
Airbus: 433
Canadair: 133
Cessna: 134
Embraer: 233
DataLoaders for selected labels created successfully.
```

1.3.2 3.2 Improved Model Design

Define the optimised CNN architecture with improved performance.

[12]:
```
# Define a simplified CNN model
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(16)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(32)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(64)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 16 * 16, 128)
```

```
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(128, len(selected_labels))
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.relu(self.bn1(self.conv1(x))))
        x = self.pool(self.relu(self.bn2(self.conv2(x))))
        x = self.pool(self.relu(self.bn3(self.conv3(x))))
        x = x.view(-1, 64 * 16 * 16)
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(16)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(16 * 64 * 64, len(selected_labels))
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.relu(self.bn1(self.conv1(x))))
        x = x.view(-1, 16 * 64 * 64)
        x = self.fc1(x)
        return x

# Instantiate the simplified model
model = SimpleCNN()
```

[13]:
```
# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)  # Fixed learning rate

# Define a learning rate scheduler
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=9, gamma=0.5)  #␣
 ↪Decays learning rate by 0.5 every 9 epochs

# Lists to store metrics
train_losses, test_losses = [], []
train_accuracies, test_accuracies = [], []
f1_scores = []
learning_rates = []

# Training loop
num_epochs = 40  # Increased number of epochs
for epoch in range(num_epochs):
```

```
    # Training phase
    model.train()
    running_loss = 0.0
    correct_train = 0
    total_train = 0
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()

    # Step the scheduler
    scheduler.step()
    learning_rates.append(scheduler.get_last_lr()[0])

    train_accuracy = correct_train / total_train

    # Validation phase
    model.eval()
    test_loss = 0.0
    correct_test = 0
    total_test = 0
    true_labels = []
    predicted_labels = []
    with torch.no_grad():
        for images, labels in test_loader:
            outputs = model(images)
            loss = criterion(outputs, labels)
            test_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total_test += labels.size(0)
            correct_test += (predicted == labels).sum().item()
            true_labels.extend(labels.cpu().numpy())
            predicted_labels.extend(predicted.cpu().numpy())

    test_accuracy = correct_test / total_test
    f1 = f1_score(true_labels, predicted_labels, average='weighted')

    # Store metrics
    train_losses.append(running_loss / len(train_loader))
    test_losses.append(test_loss / len(test_loader))
    train_accuracies.append(train_accuracy)
```

```
    test_accuracies.append(test_accuracy)
    f1_scores.append(f1)

    print(f"Epoch [{epoch+1}/{num_epochs}], Train Loss: {running_loss/
 ↪len(train_loader):.4f}, Test Loss: {test_loss/len(test_loader):.4f}, Train Acc:
 ↪ {train_accuracy:.4f}, Test Acc: {test_accuracy:.4f}, F1 Score: {f1:.4f},␣
 ↪Learning Rate: {scheduler.get_last_lr()[0]}")

print("Training complete.")

# Create Epochs for plotting results
epochs = range(1, num_epochs + 1)
```

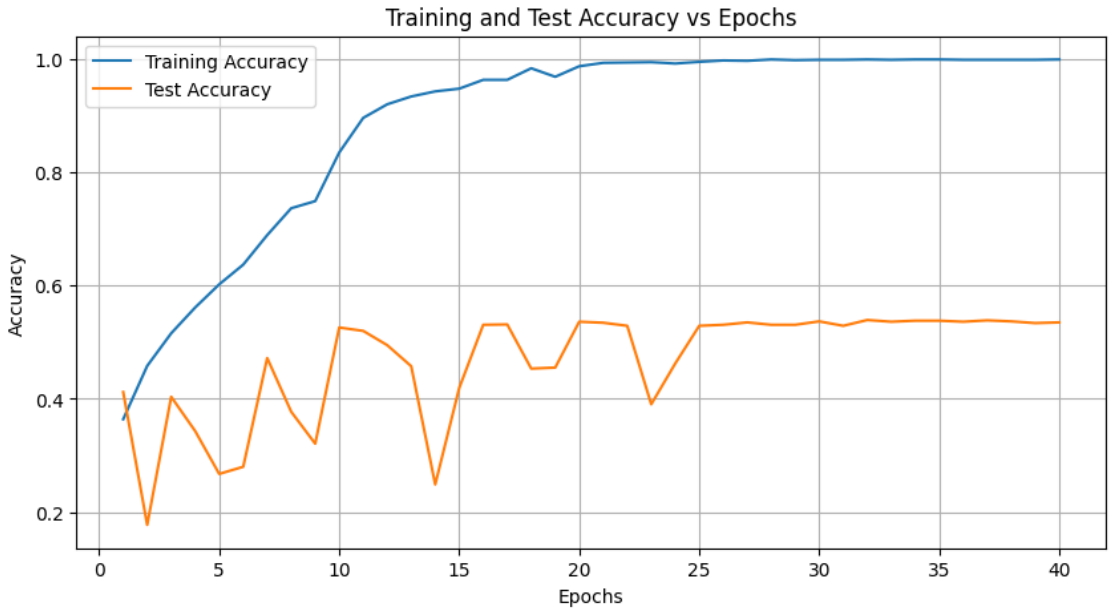
```
Epoch [1/40], Train Loss: 9.1959, Test Loss: 2.0513, Train Acc: 0.3641, Test
Acc: 0.4121, F1 Score: 0.3307, Learning Rate: 0.01
Epoch [2/40], Train Loss: 1.3687, Test Loss: 2.4558, Train Acc: 0.4583, Test
Acc: 0.1776, F1 Score: 0.1606, Learning Rate: 0.01
Epoch [3/40], Train Loss: 1.1897, Test Loss: 1.4528, Train Acc: 0.5159, Test
Acc: 0.4037, F1 Score: 0.3963, Learning Rate: 0.01
Epoch [4/40], Train Loss: 1.0382, Test Loss: 2.3682, Train Acc: 0.5615, Test
Acc: 0.3431, F1 Score: 0.2831, Learning Rate: 0.01
Epoch [5/40], Train Loss: 1.0304, Test Loss: 3.0175, Train Acc: 0.6023, Test
Acc: 0.2675, F1 Score: 0.1386, Learning Rate: 0.01
Epoch [6/40], Train Loss: 0.9436, Test Loss: 2.4723, Train Acc: 0.6371, Test
Acc: 0.2801, F1 Score: 0.1495, Learning Rate: 0.01
Epoch [7/40], Train Loss: 0.8308, Test Loss: 1.5123, Train Acc: 0.6893, Test
Acc: 0.4721, F1 Score: 0.4305, Learning Rate: 0.01
Epoch [8/40], Train Loss: 0.6985, Test Loss: 1.9315, Train Acc: 0.7367, Test
Acc: 0.3773, F1 Score: 0.3419, Learning Rate: 0.01
Epoch [9/40], Train Loss: 0.7009, Test Loss: 2.6508, Train Acc: 0.7493, Test
Acc: 0.3209, F1 Score: 0.2518, Learning Rate: 0.005
Epoch [10/40], Train Loss: 0.5131, Test Loss: 1.2718, Train Acc: 0.8350, Test
Acc: 0.5261, F1 Score: 0.4950, Learning Rate: 0.005
Epoch [11/40], Train Loss: 0.3976, Test Loss: 1.4083, Train Acc: 0.8962, Test
Acc: 0.5201, F1 Score: 0.4830, Learning Rate: 0.005
Epoch [12/40], Train Loss: 0.3389, Test Loss: 1.2983, Train Acc: 0.9202, Test
Acc: 0.4949, F1 Score: 0.4939, Learning Rate: 0.005
Epoch [13/40], Train Loss: 0.2970, Test Loss: 1.4776, Train Acc: 0.9340, Test
Acc: 0.4577, F1 Score: 0.4604, Learning Rate: 0.005
Epoch [14/40], Train Loss: 0.2716, Test Loss: 2.8273, Train Acc: 0.9430, Test
Acc: 0.2490, F1 Score: 0.2206, Learning Rate: 0.005
Epoch [15/40], Train Loss: 0.2471, Test Loss: 1.7417, Train Acc: 0.9478, Test
Acc: 0.4193, F1 Score: 0.4040, Learning Rate: 0.005
Epoch [16/40], Train Loss: 0.2048, Test Loss: 1.4096, Train Acc: 0.9634, Test
Acc: 0.5309, F1 Score: 0.5010, Learning Rate: 0.005
Epoch [17/40], Train Loss: 0.1898, Test Loss: 1.4626, Train Acc: 0.9634, Test
Acc: 0.5315, F1 Score: 0.5055, Learning Rate: 0.005
Epoch [18/40], Train Loss: 0.1639, Test Loss: 1.7807, Train Acc: 0.9838, Test
```

```
Acc: 0.4535, F1 Score: 0.4325, Learning Rate: 0.0025
Epoch [19/40], Train Loss: 0.1641, Test Loss: 1.6966, Train Acc: 0.9688, Test
Acc: 0.4553, F1 Score: 0.4503, Learning Rate: 0.0025
Epoch [20/40], Train Loss: 0.1268, Test Loss: 1.4149, Train Acc: 0.9874, Test
Acc: 0.5363, F1 Score: 0.5145, Learning Rate: 0.0025
Epoch [21/40], Train Loss: 0.1127, Test Loss: 1.3864, Train Acc: 0.9934, Test
Acc: 0.5345, F1 Score: 0.5219, Learning Rate: 0.0025
Epoch [22/40], Train Loss: 0.1078, Test Loss: 1.4369, Train Acc: 0.9940, Test
Acc: 0.5291, F1 Score: 0.5112, Learning Rate: 0.0025
Epoch [23/40], Train Loss: 0.1104, Test Loss: 2.0805, Train Acc: 0.9946, Test
Acc: 0.3905, F1 Score: 0.3601, Learning Rate: 0.0025
Epoch [24/40], Train Loss: 0.1080, Test Loss: 1.6883, Train Acc: 0.9922, Test
Acc: 0.4625, F1 Score: 0.4589, Learning Rate: 0.0025
Epoch [25/40], Train Loss: 0.0953, Test Loss: 1.4062, Train Acc: 0.9952, Test
Acc: 0.5291, F1 Score: 0.5190, Learning Rate: 0.0025
Epoch [26/40], Train Loss: 0.0866, Test Loss: 1.4943, Train Acc: 0.9976, Test
Acc: 0.5309, F1 Score: 0.5117, Learning Rate: 0.0025
Epoch [27/40], Train Loss: 0.0838, Test Loss: 1.5700, Train Acc: 0.9970, Test
Acc: 0.5351, F1 Score: 0.5034, Learning Rate: 0.00125
Epoch [28/40], Train Loss: 0.0766, Test Loss: 1.4738, Train Acc: 0.9994, Test
Acc: 0.5309, F1 Score: 0.5229, Learning Rate: 0.00125
Epoch [29/40], Train Loss: 0.0743, Test Loss: 1.4553, Train Acc: 0.9982, Test
Acc: 0.5309, F1 Score: 0.5222, Learning Rate: 0.00125
Epoch [30/40], Train Loss: 0.0713, Test Loss: 1.4617, Train Acc: 0.9988, Test
Acc: 0.5369, F1 Score: 0.5265, Learning Rate: 0.00125
Epoch [31/40], Train Loss: 0.0709, Test Loss: 1.4722, Train Acc: 0.9988, Test
Acc: 0.5291, F1 Score: 0.5219, Learning Rate: 0.00125
Epoch [32/40], Train Loss: 0.0688, Test Loss: 1.4783, Train Acc: 0.9994, Test
Acc: 0.5393, F1 Score: 0.5280, Learning Rate: 0.00125
Epoch [33/40], Train Loss: 0.0683, Test Loss: 1.4998, Train Acc: 0.9988, Test
Acc: 0.5363, F1 Score: 0.5211, Learning Rate: 0.00125
Epoch [34/40], Train Loss: 0.0662, Test Loss: 1.4726, Train Acc: 0.9994, Test
Acc: 0.5381, F1 Score: 0.5273, Learning Rate: 0.00125
Epoch [35/40], Train Loss: 0.0636, Test Loss: 1.4906, Train Acc: 0.9994, Test
Acc: 0.5381, F1 Score: 0.5253, Learning Rate: 0.00125
Epoch [36/40], Train Loss: 0.0624, Test Loss: 1.4922, Train Acc: 0.9988, Test
Acc: 0.5363, F1 Score: 0.5252, Learning Rate: 0.000625
Epoch [37/40], Train Loss: 0.0645, Test Loss: 1.4988, Train Acc: 0.9988, Test
Acc: 0.5387, F1 Score: 0.5246, Learning Rate: 0.000625
Epoch [38/40], Train Loss: 0.0596, Test Loss: 1.4951, Train Acc: 0.9988, Test
Acc: 0.5369, F1 Score: 0.5272, Learning Rate: 0.000625
Epoch [39/40], Train Loss: 0.0606, Test Loss: 1.4727, Train Acc: 0.9988, Test
Acc: 0.5339, F1 Score: 0.5248, Learning Rate: 0.000625
Epoch [40/40], Train Loss: 0.0615, Test Loss: 1.4848, Train Acc: 0.9994, Test
Acc: 0.5351, F1 Score: 0.5242, Learning Rate: 0.000625
Training complete.
```

1.3.3 3.3 Metrics for Improved Model

Visualise the performance of this improved model using metrics like accuracy and F1-score.

[14]:
```
# Plot Loss
plt.figure(figsize=(10, 5))
plt.plot(epochs, train_losses, label='Training Loss')
plt.plot(epochs, test_losses, label='Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Test Loss vs Epochs')
plt.legend()
plt.grid(True)
plt.show()

# Plot Accuracy
plt.figure(figsize=(10, 5))
plt.plot(epochs, train_accuracies, label='Training Accuracy')
plt.plot(epochs, test_accuracies, label='Test Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Test Accuracy vs Epochs')
plt.legend()
plt.grid(True)
plt.show()
```



1.4 4. Results and Evaluation

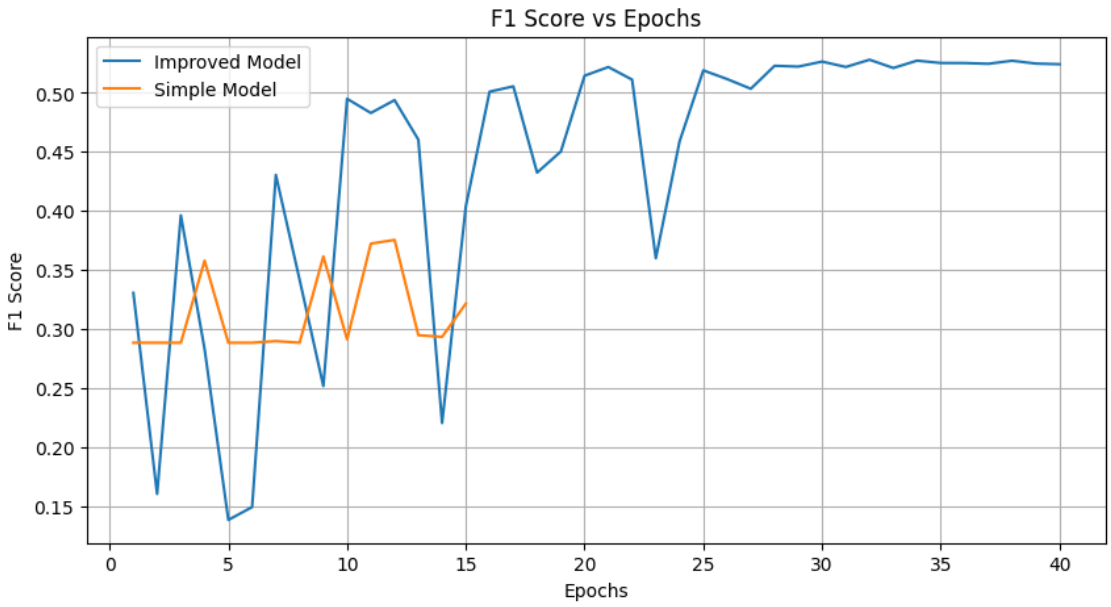
1.4.1 4.1 Metrics Comparison

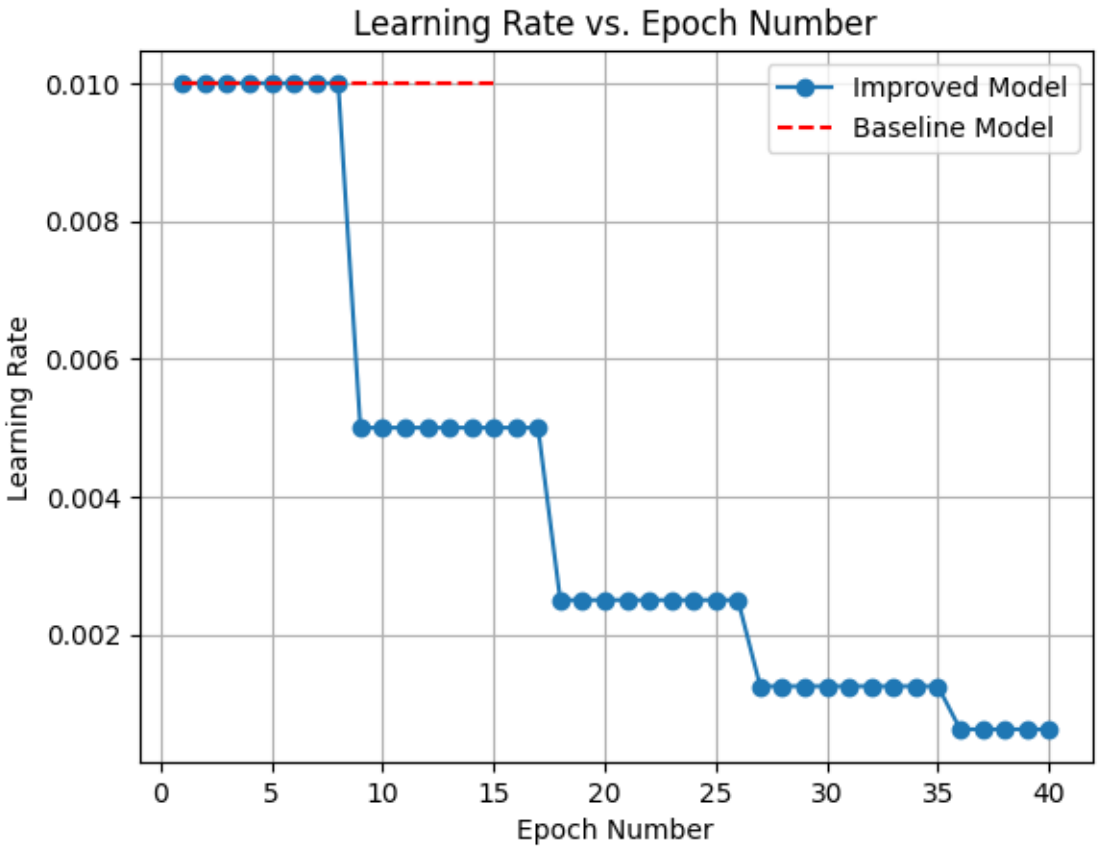
Compare the performance of the initial and optimised models primarily using metrics like F1-score.

Generate plots for metrics and training curves.

[15]:
```
# Plot F1 Score
plt.figure(figsize=(10, 5))

# Improved Model F1 Scores
plt.plot(epochs, f1_scores, linestyle='-', label="Improved Model")

# Plot the F1 values from the Baseline Model, this is hard-coded in
plt.plot(
    range(1, 16),
    [0.2885368466152528, 0.2885368466152528, 0.2885368466152528, 0.
 ↪35800560282056604,
     0.2885368466152528, 0.2885368466152528, 0.28990515812498047, 0.
 ↪2885368466152528,
     0.36161119202439296, 0.2913618247972723, 0.3724076408386039, 0.
 ↪3755616336834515,
     0.29490095228198265, 0.29349818381047593, 0.32165556746495966],␣
 ↪linestyle='-', label="Simple Model"
)
```

```
# Formatting the plot
plt.xlabel('Epochs')
plt.ylabel('F1 Score')
plt.title('F1 Score vs Epochs')
plt.legend()
plt.grid(True)
plt.show()

baseline = [0.01] * 15

# Plot improved model (all 50 epochs)
plt.plot(range(1, num_epochs + 1), learning_rates, marker='o', linestyle='-',␣
↪label="Improved Model")
# Plot baseline model (up to epoch 15)
plt.plot(range(1, 16), baseline, linestyle='--', color='red', label="Baseline␣
↪Model")
plt.title("Learning Rate vs. Epoch Number")
plt.xlabel("Epoch Number")
plt.ylabel("Learning Rate")
plt.grid(True)
plt.legend()
plt.show()
```





1.4.2 4.2 Discussion

All in the Word Document - Report