Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Fall, Year: 2023), B.Sc. in CSE (Day)

Real-time Bangla handwritten character classification using Deep Learning

---

Course Title: Data Mining Lab
Course Code: CSE 436
Section: 201 D7

Students Details

Name	**ID**
Himel Saha	201002111
Md. Abdul Kayum	201002334

Submission Date: 12-01-2024
Course Teachers Name: Mr. Abdullah Al Farhad

[For teachers use only: Dont write anything inside this box]

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

Navigating the rich tapestry of Bangla script, our project unfolds the realm of real-time handwritten character classification using the prowess of deep learning techniques such as **EfficientNetV2**. Utilizing EfficientNetV2, the model will recognize and classify Bangla characters in real-time scenarios. The focus is on adapting to diverse handwriting styles and variations, ensuring swift and accurate character identification. The project's objective is to showcase the capabilities of deep learning in addressing the unique challenges of Bangla script, contributing to improved real-time character classification for handwritten Bangla text.

1.2 Motivation

This project is fueled by the idea of making technology understand Bangla handwriting in real time, simplifying how we interact with our language. Imagine a tool that effortlessly recognizes handwritten characters, enhancing accessibility and communication.

- The project is driven by the vision of enabling technology to understand Bangla handwriting in real-time.
- Aiming to simplify language interaction, the project envisions a tool that effortlessly recognizes handwritten Bangla characters.
- The primary motivation is to offer a user-friendly solution for the Bangla script, making digital engagement easy using natural handwriting.
- Lastly, we aspire to harness the power of EfficientNetV2 to elevate our understanding and knowledge through the implementation of this project.

So, it's about making technology work for us, effortlessly bridging the gap between traditional writing and the digital age.

1.3 Problem Definition

1.3.1 Problem Statement

In this project, we are implementing EfficientNetV2 for real-time Bangla handwritten character classification to enhance user-friendly applications and adapt to diverse handwriting styles.

- How can EfficientNetV2 be utilized to enhance real-time Bangla handwritten character classification, considering diverse handwriting styles, and offering practitioners a streamlined approach for effective character recognition on resource-constrained platforms?
- How can we simplify the comparison of deep learning methodologies for real-time Bangla handwritten character classification, enabling practitioners to make informed decisions in diverse scenarios, and advancing language-specific machine learning practices?

By formulating these problem statements, the project endeavors to address the unique challenges in efficiently implementing EfficientNetV2 for real-time Bangla handwritten character classification.

1.3.2 Complex Engineering Problem

Table 1.1: Summary of the attributes touched by the mentioned projects

Attributess	**Relevance to Project**
P1: CNN Models	Attain expertise in various CNN methodologies through continuous learning. (Highly relevant to the projects success.)
P2: Performance Optimization (Training Speed vs Accuracy)	In this project, we are navigating a delicate balance between training latency and accuracy, strategically optimizing the trade-off to achieve efficient and effective real-time Bangla handwritten character classification.
P3: EfficientNetV2	Unleashing the power of EfficientNetV2, our project pioneers advanced deep learning techniques for real-time Bangla handwritten character classification, ensuring a harmonious blend of efficiency and accuracy.
P4: Digital Platforms	This project is crucial for smoothly integrating Bangla handwriting into digital platforms, making communication more accessible in the digital age.

1.4 Design Goals/Objectives

The main goal of this project is to efficiently implement EfficientNetV2 for real-time Bangla handwritten character classification, harmonizing accuracy, and training efficiency. The following goals are associated with this project:

- To optimize EfficientNetV2 to enable swift and accurate real-time recognition of Bangla handwritten characters.
- To enhance the model's adaptability to diverse Bangla handwriting styles, ensuring robust classification across varied script variations.
- To streamline the integration of the classification system into digital platforms, providing a user-friendly solution for seamless interaction with Bangla script in various applications.

1.5 Application

The following real-life applications showcase the versatility of this project:

1. Educational Technology: Integration of our project into educational platforms for real-time evaluation and feedback on handwritten assignments, quizzes, or assessments in the Bangla language.

2. Document Digitization Services: Enhancing OCR (Optical Character Recognition) systems for efficient digitization of handwritten Bangla documents, facilitating data retrieval and analysis.

3. Human-Computer Interaction (HCI): Incorporation into HCI interfaces, allowing users to interact with digital devices through natural Bangla handwriting for tasks like inputting text or commands.

4. Online Form Processing: Streamlining the processing of online forms in Bangla by recognizing and validating handwritten responses in real time, improving user experience.

5. Bangla Language Support in Digital Assistants: Enabling digital assistants to understand and respond to handwritten queries or commands in the Bangla language, enhancing accessibility and user engagement.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

In the era of real-time data processing and machine learning, this project focuses on the implementation of EfficientNetV2 for the classification of Bangla handwritten characters. By leveraging the efficiency and adaptability of EfficientNetV2, the goal is to achieve swift and accurate recognition, revolutionizing the way we interact with and understand Bangla handwriting in various applications.

2.2 Project Details

2.2.1 Key Components

In this project, we have used the following key components:

- **EfficientNetV2 Model:** The EfficientNetV2 model serves as the project's neural powerhouse, meticulously designed to achieve both efficiency and precision in the real-time recognition of Bangla handwritten characters.
- **Bangla Handwritten Dataset:** A comprehensive dataset of Bangla handwritten characters used for training, validating, and testing the EfficientNetV2 model, ensuring its ability to generalize across various styles and variations.
- **Data Preprocessing Module:** Techniques to preprocess and augment the dataset, preparing it for training by applying normalization, resizing, and other transformations to enhance the model's ability to learn and generalize.
- **Hyperparameter Tuning:** Through meticulous hyperparameter tuning, we've fine-tuned the model to attain optimal performance, ensuring the most effective and accurate outcomes in our real-time Bangla handwritten character classification project.

2.2.2 Tools and libraries

Here are some tools and libraries used in the project:

- **Software Tools:**
  - Desktop/Laptop
  - Google Colab Notebook
  - Google Chrome
- **Language, Libraries & Techniques :**
  - Python: Python programming language for coding.
  - Pytorch: Used as the primary deep learning framework.
  - Matplotlib: For data visualization
  - Pygame: For dynamic Bangla character rendering

2.2.3 EfficientNetV2 Architecture

Stage	Operator	Stride	#Channels	#Layers
0	Conv3x3	2	24	1
1	Fused-MBConv1, k3x3	1	24	2
2	Fused-MBConv4, k3x3	2	48	4
3	Fused-MBConv4, k3x3	2	64	4
4	MBConv4, k3x3, SE0.25	2	128	6
5	MBConv6, k3x3, SE0.25	1	160	9
6	MBConv6, k3x3, SE0.25	2	256	15
7	Conv1x1 & Pooling & FC	-	1280	1

Figure 2.1: EfficientNetV2 Architecture

	EfficientNet (2019)	ResNet-RS (2021)	DeiT/ViT (2021)	EfficientNetV2 (ours)
Top-1 Acc.	84.3%	84.0%	83.1%	83.9%
Parameters	43M	164M	86M	24M

(b) Parameter efficiency.

Figure 2.2: Parameter Comparison with other models

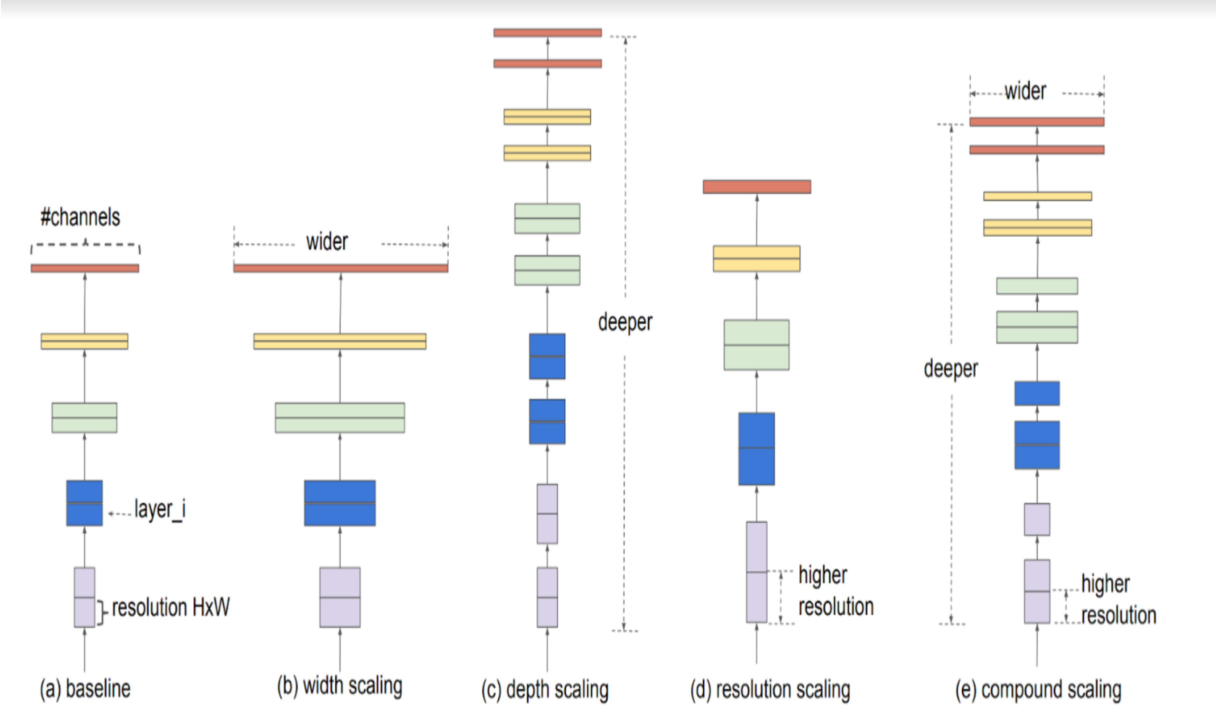

Figure 2.3: Image Scaling method of EfficientNetV2

2.3 Implementation

2.3.1 Importing Libraries:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import transforms, datasets
from PIL import Image
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import pygame
from tqdm import tqdm
from PIL import Image
import os
import warnings
warnings.filterwarnings("ignore")
```

2.3.2 EfficientNetV2 Model

```
class EfficientNetV2(nn.Module):
    def __init__(self):
        super(EfficientNetV2, self).__init__()


        self.MBConv1 = nn.Sequential(
            nn.Conv2d(3, 24, kernel_size=3, stride=2,
               padding=2),
            nn.BatchNorm2d(24),
            nn.ReLU(inplace=True),
            nn.Conv2d(24, 24, kernel_size=3, stride=1,
               padding=1),
            nn.BatchNorm2d(24),
            nn.ReLU(inplace=True),

        )


        self.MBConv2 = nn.Sequential(
            nn.Conv2d(24, 48, kernel_size=3, stride=2,
               padding=2),
            nn.BatchNorm2d(48),
            nn.ReLU(inplace=True),
            nn.Conv2d(48, 48, kernel_size=3, stride=1,
               padding=1),
            nn.BatchNorm2d(48),
            nn.ReLU(inplace=True),

        )


        self.MBConv3 = nn.Sequential(
            nn.Conv2d(48, 64, kernel_size=3, stride=2,
               padding=2),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, stride=1,
               padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
        )


        self.MBConv4 = nn.Sequential(
            nn.Conv2d(64, 96, kernel_size=3, stride=2,
               padding=2),
```

```
            nn.BatchNorm2d(96),
            nn.ReLU(inplace=True),
            nn.Conv2d(96, 96, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(96),
            nn.ReLU(inplace=True),
        )


        self.MBConv5 = nn.Sequential(
            nn.Conv2d(96, 160, kernel_size=3, stride=2, padding=2),
            nn.BatchNorm2d(160),
            nn.ReLU(inplace=True),
            nn.Conv2d(160, 160, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(160),
            nn.ReLU(inplace=True),
        )

        self.MBConv6 = nn.Sequential(
            nn.Conv2d(160, 256, kernel_size=3, stride=2, padding=2),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
        )


        self.MBConv7 = nn.Sequential(
            nn.Conv2d(256, 1280, kernel_size=3, stride=2,padding=2),
            nn.BatchNorm2d(1280),
            nn.ReLU(inplace=True),
            nn.Conv2d(1280, 1280, kernel_size=3, stride=1,padding=1),
            nn.BatchNorm2d(1280),
            nn.ReLU(inplace=True),
        )

        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.classifier = nn.Linear(1280,50)

    def forward(self, x):
        x = self.MBConv1(x)
```

```
        x = self.MBConv2(x)
        x = self.MBConv3(x)
        x = self.MBConv4(x)
        x = self.MBConv5(x)
        x = self.MBConv6(x)
        x = self.MBConv7(x)


        x = self.avg_pool(x)
        x = x.view(-1,1280)
        x = self.classifier(x)

        return x
```

2.3.3 Data Preprocessing and Loading

```
# Data preprocessing and loading
data_transform = transforms.Compose([
    transforms.Resize((40, 40)),
    transforms.ToTensor(),
])

absolute_path = '/content/drive/My Drive/Colab Notebooks/Datasets/Bangla_Dataset/Train'
absolute_path1 = '/content/drive/My Drive/Colab Notebooks/Datasets/Bangla_Dataset/Test'

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
train_dataset= datasets.ImageFolder(root=absolute_path, transform=data_transform)
test_dataset=datasets.ImageFolder(root=absolute_path1, transform=data_transform)

train_loader= torch.utils.data.DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader= torch.utils.data.DataLoader(test_dataset, batch_size=32, shuffle=False)
```

2.3.4 Parameters

```
num_classes = 50
model = EfficientNetV2()
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

2.3.5 Training Loop

```
train_losses = []
test_losses = []
num_epochs=1
for epoch in range(num_epochs):
    running_loss = 0.0
    model.train()
    # Wrap train_loader with tqdm for a progress bar
    for i, data in tqdm(enumerate(train_loader, 0), desc=f'Epoch {epoch + 1}/{num_epochs}', total=len(train_loader)):
        inputs, labels = data
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss = running_loss + loss.item()

    # Calculate and store training loss
        train_loss = running_loss / len(train_loader)
        train_losses.append(train_loss)
    print(f"Epoch {epoch + 1}, Train Loss: {train_loss:.5f}")

    model.eval()
    test_loss = 0.0
    with torch.no_grad():
        for data in test_loader:
            inputs, labels = data
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            test_loss = test_loss + loss.item()

    # Calculate and store Test loss
            test_loss = test_loss / len(test_loader)
            test_losses.append(test_loss)
    print(f"Epoch {epoch + 1}, Test Loss: {test_loss:.5f}\n")
print("Training finished")
```

2.3.6 Model Evaluation

```
model.eval()
all_preds = []
all_labels = []
with torch.no_grad():
    for inputs, labels in tqdm(test_loader, desc='Testing'):
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())
        loss = criterion(outputs, labels)

# Calculate accuracy
accuracy = accuracy_score(all_labels, all_preds)
print(f'Overall Accuracy: {accuracy * 100:.2f}%')
```

2.3.7 Initializing Pygame

```
# Initialize pygame
pygame.init()

# Set the dimensions of the canvas
canvas_size = (200, 200)

# Set up the canvas
canvas = pygame.display.set_mode(canvas_size)
pygame.display.set_caption("Draw a Character")

# Colors
white = (255, 255, 255)
black = (0, 0, 0)

# Initialize the canvas as white
canvas.fill(white)

drawing = False
last_pos = None

# Main loop
running = True
while running:
    for event in pygame.event.get():
```

```
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.MOUSEBUTTONDOWN:
            drawing = True
            last_pos = event.pos
        elif event.type == pygame.MOUSEBUTTONUP:
            drawing = False
        elif event.type == pygame.MOUSEMOTION:
            if drawing:
                pygame.draw.line(canvas, black, last_pos, event.pos, 7)
                last_pos = event.pos

    pygame.display.flip()

# Save the image
image_folder = "Bangla_Dataset/SingleImage"
os.makedirs(image_folder, exist_ok=True)
image_path = os.path.join(image_folder, "image.png")
pygame.image.save(canvas, image_path)


image_path = 'Bangla_Dataset/SingleImage/image.png'
img = Image.open(image_path)
img_resized = img.resize((40, 40))
img_resized.save('Bangla_Dataset/SingleImage/image.png')
print("Thank you for your inputed character...!")

pygame.quit()
```

2.3.8 Testing our Model

```
# Test single image
model.eval()
image_path = 'Bangla_Dataset/SingleImage/image.png'  # Path to your single test image
image = Image.open(image_path).convert('RGB')
image_tensor = data_transform(image).unsqueeze(0)  # Add a batch dimension

with torch.no_grad():
    output = model(image_tensor)
    _, predicted_class = torch.max(output, 1)

# Load class labels
```

```
class_labels = train_dataset.classes

# Print predicted class
print(f'Predicted Class: {class_labels[predicted_class.item()]}')

#Print Character
if class_labels[predicted_class.item()] == "Character_1":
    print('prediction : অ')
elif class_labels[predicted_class.item()] == "Character_2":
    print('prediction : আ')
elif class_labels[predicted_class.item()] == "Character_3":
    print('prediction : ই')
elif class_labels[predicted_class.item()] == "Character_4":
    print('prediction : ঈ')
elif class_labels[predicted_class.item()] == "Character_5":
    print('prediction : উ')
elif class_labels[predicted_class.item()] == "Character_6":
    print('prediction : ঊ')
elif class_labels[predicted_class.item()] == "Character_7":
    print('prediction : ঋ')
elif class_labels[predicted_class.item()] == "Character_8":
    print('prediction : এ')
elif class_labels[predicted_class.item()] == "Character_9":
    print('prediction : ঐ')
elif class_labels[predicted_class.item()] == "Character_10":
    print('prediction : ও')
elif class_labels[predicted_class.item()] == "Character_11":
    print('prediction : ঔ')
elif class_labels[predicted_class.item()] == "Character_12":
    print('prediction : ক')
elif class_labels[predicted_class.item()] == "Character_13":
    print('prediction :খ')
elif class_labels[predicted_class.item()] == "Character_14":
```

```
    print('prediction : গ')
elif class_labels[predicted_class.item()] == "Character_15":
    print('prediction : ঘ')
elif class_labels[predicted_class.item()] == "Character_16":
    print('prediction : ঙ')
elif class_labels[predicted_class.item()] == "Character_17":
    print('prediction : চ')
elif class_labels[predicted_class.item()] == "Character_18":
    print('prediction : ছ')
elif class_labels[predicted_class.item()] == "Character_19":
    print('prediction : জ')
elif class_labels[predicted_class.item()] == "Character_20":
    print('prediction : ঝ')
elif class_labels[predicted_class.item()] == "Character_21":
    print('prediction : ঞ')
elif class_labels[predicted_class.item()] == "Character_22":
    print('prediction : ট')
elif class_labels[predicted_class.item()] == "Character_23":
    print('prediction : ঠ')
elif class_labels[predicted_class.item()] == "Character_24":
    print('prediction : ড')
elif class_labels[predicted_class.item()] == "Character_25":
    print('prediction : ঢ')
elif class_labels[predicted_class.item()] == "Character_26":
    print('prediction : ণ')
elif class_labels[predicted_class.item()] == "Character_27":
    print('prediction : ত')
elif class_labels[predicted_class.item()] == "Character_28":
    print('prediction : থ')
elif class_labels[predicted_class.item()] == "Character_29":
    print('prediction : দ')
elif class_labels[predicted_class.item()] == "Character_30":
```

```
    print('prediction : ধ')
elif class_labels[predicted_class.item()] == "Character_31":
    print('prediction : ন')
elif class_labels[predicted_class.item()] == "Character_32":
    print('prediction : প')
elif class_labels[predicted_class.item()] == "Character_33":
    print('prediction : ফ')
elif class_labels[predicted_class.item()] == "Character_34":
    print('prediction : ব')
elif class_labels[predicted_class.item()] == "Character_35":
    print('prediction : ভ')
elif class_labels[predicted_class.item()] == "Character_36":
    print('prediction : ম')
elif class_labels[predicted_class.item()] == "Character_37":
    print('prediction : য')
elif class_labels[predicted_class.item()] == "Character_38":
    print('prediction : র')
elif class_labels[predicted_class.item()] == "Character_39":
    print('prediction : ল')
elif class_labels[predicted_class.item()] == "Character_40":
    print('prediction : শ')
elif class_labels[predicted_class.item()] == "Character_41":
    print('prediction : ষ')
elif class_labels[predicted_class.item()] == "Character_42":
    print('prediction : স')
elif class_labels[predicted_class.item()] == "Character_43":
    print('prediction : হ')
elif class_labels[predicted_class.item()] == "Character_44":
    print('prediction : ড়')
elif class_labels[predicted_class.item()] == "Character_45":
    print('prediction : ঢ়')
elif class_labels[predicted_class.item()] == "Character_46":
```

```
    print('prediction : য়')
elif class_labels[predicted_class.item()] == "Character_47":
    print('prediction : ৎ')
elif class_labels[predicted_class.item()] == "Character_48":
    print('prediction : ং')
elif class_labels[predicted_class.item()] == "Character_49":
    print('prediction : ঃ')
elif class_labels[predicted_class.item()] == "Character_50":
    print('prediction :  ঁ')
else:
    print('prediction : No Match')
```

2.3.9 Ploting Training vs Testing loss

```
plt.figure(figsize=(7, 4))
plt.plot(train_losses, label="Training Loss")
plt.plot(test_losses, label="Test Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

2.3.10 Hyperparameter Tuning (Different Batch Sizes)

```
batch_sizes = [8,16,32]
batch_size_train_losses = {}
batch_size_test_losses = {}
batch_size_overall_accuracies = {}

for batch_size in batch_sizes:
    # Set up DataLoader for train dataset
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

    model = EfficientNetV2()
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    train_losses = []
```

```
    test_losses = []
    overall_accuracies = []

    num_epochs = 5

    for epoch in range(num_epochs):
        running_loss = 0.0
        correct_train = 0
        total_train = 0

        model.train()

        for i, data in tqdm(enumerate(train_loader, 0), desc=f'Epoch {epoch + 1}/{num_epochs}', total=len(train_loader)):
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

            _, predicted = torch.max(outputs.data, 1)
            total_train += labels.size(0)
            correct_train += (predicted == labels).sum().item()

        train_accuracy = correct_train / total_train
        train_loss = running_loss / len(train_loader)
        train_losses.append(train_loss)
        print(f"Epoch {epoch + 1}, Train Loss: {train_loss:.3f}")

        model.eval()
        test_loss = 0.0
        correct_test = 0
        total_test = 0

        with torch.no_grad():
            for data in test_loader:
                inputs, labels = data
                inputs, labels = inputs.to(device), labels.to(device)
```

```python

                outputs = model(inputs)
                loss = criterion(outputs, labels)
                test_loss += loss.item()

                _, predicted = torch.max(outputs.data, 1)
                total_test += labels.size(0)
                correct_test += (predicted == labels).sum().item()

        test_loss= test_loss / len(test_loader)
        test_losses.append(test_loss)
        print(f"Epoch {epoch + 1}, Test Loss: {test_loss: .5f}\n")

        # Calculate overall accuracy
        overall_accuracy = ((correct_train + correct_test) / (total_train + total_test))*100
        overall_accuracies.append(overall_accuracy)

        print(f"Epoch {epoch + 1}, Overall Accuracy: {overall_accuracy:.2f}%")

    batch_size_train_losses[batch_size] = train_losses
    batch_size_test_losses[batch_size] = test_losses
    batch_size_overall_accuracies[batch_size] = overall_accuracies
```

2.3.11 Ploting Batch Sizes vs Train/Test Loss

```python
# Plot batch size vs Train loss graph
plt.figure(figsize=(7, 4))
for batch_size in batch_sizes:
    plt.plot(batch_size_train_losses[batch_size], label=f"Training Loss for Batch Size {batch_size}", marker='o')

plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.title("Batch Size vs Training Loss")
plt.show()
```

```
# Plot batch size vs Test loss graph
plt.figure(figsize=(7, 4))
for batch_size in batch_sizes:
    plt.plot(batch_size_test_losses[batch_size], label=f"Testing Loss for Batch Size {batch_size}",marker='o')

plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.title("Batch Size vs Test Loss")
plt.show()
```

2.3.12 Ploting Accuracy for each batch size

```
# Plotting overall accuracy for each batch size
for batch_size in batch_sizes:
    overall_accuracies = batch_size_overall_accuracies[batch_size]
    plt.plot(range(1, num_epochs + 1), overall_accuracies, label=f'Overall Acc (Batch Size {batch_size})', marker='o')

plt.xlabel('Epoch')
plt.ylabel('Overall Accuracy')
plt.title('Overall Model Accuracy Over Epochs for Each Batch Size')
plt.legend()
plt.grid(True)
plt.yticks(range(0, 101, 10))
plt.show()
```

2.3.13 Hyperparameter Tuning (Different Learning Rates)

```
learning_rates = [0.001, 0.01, 0.1]
lr_train_losses = {}
lr_test_losses = {}
lr_overall_accuracies = {}

for learning_rate in learning_rates:
```

```
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    train_losses = []
    test_losses =[]
    overall_accuracies = []
    num_epochs= 5
    for epoch in range(num_epochs):
        running_loss = 0.0
        correct_train = 0
        total_train = 0
        model.train()
        for i, data in tqdm(enumerate(train_loader, 0), desc=f'Epoch {epoch + 1}/{num_epochs}', total=len(train_loader)):
            inputs, labels = data
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss = running_loss + loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total_train += labels.size(0)
            correct_train += (predicted == labels).sum().item()

        train_accuracy = correct_train / total_train
        train_loss = running_loss / len(train_loader)
        train_losses.append(train_loss)
        print(f"Epoch {epoch + 1}, Train Loss: {running_loss / len(train_loader): .5f}")

        model.eval()
        test_loss = 0.0
        correct_test = 0
        total_test = 0

        with torch.no_grad():
            for data in test_loader:
                inputs, labels = data
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                test_loss=test_loss+loss.item()
                _, predicted = torch.max(outputs.data, 1)
                total_test += labels.size(0)
```

```
                correct_test += (predicted == labels).sum().item()

        # Calculate and store Test loss
        test_loss= test_loss/len(test_loader)
        test_losses.append(test_loss)
        print(f"Epoch {epoch + 1}, Test Loss: {test_loss: .5f}\n")

        overall_accuracy = ((correct_train + correct_test) / (total_train + total_test))*100
        overall_accuracies.append(overall_accuracy)

        print(f"Epoch {epoch + 1}, Overall Accuracy: {overall_accuracy:.2f}%")

    lr_train_losses[learning_rate] = train_losses
    lr_test_losses[learning_rate] = test_losses
    lr_overall_accuracies[learning_rate] = overall_accuracies
```

2.3.14 Ploting Learning rate vs Loss graph

```
# Plot learning rate vs loss graph
plt.figure(figsize=(7, 4))
for learning_rate in learning_rates:
    plt.plot(lr_train_losses[learning_rate], label=f"LR={learning_rate}",marker='o')

plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.title("Learning Rate vs Train Loss")
plt.show()

# Plot learning rate vs loss graph
plt.figure(figsize=(7, 4))
for learning_rate in learning_rates:
    plt.plot(lr_test_losses[learning_rate], label=f"LR={learning_rate}",marker='o')

plt.xlabel("Epoch")
plt.ylabel("Loss")
```

```
plt.legend()
plt.grid(True)
plt.title("Learning Rate vs Test Loss")
plt.show()
```

2.3.15 Ploting Accuracy for each learning rate

```
label_x, label_y = max(learning_rates) + 1, 0

for learning_rate in learning_rates:
    overall_accuracies = lr_overall_accuracies[
       learning_rate ]
    plt.plot(range(1, num_epochs + 1),
       overall_accuracies, label=f'Overall Acc (Learning
        Rate {learning_rate})', marker='o')
    plt.text(label_x, label_y, f'Learning Rate-{
       learning_rate}: {overall_accuracies[-1]:.2f}%',
       ha='left', va='center', color='black')
    label_y += 10
plt.xlabel('Epoch')
plt.ylabel('Overall Accuracy')
plt.title('Overall Model Accuracy Over Epochs for Each
   Learning Rate')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)
plt.yticks(range(0, 101, 10))
plt.show()
```

Chapter 3

Performance Evaluation

3.1 Results Analysis/Testing

3.1.1 Dataset Pictures

- These are some of the samples that we have used to train our model.

Figure 3.1: Dataset

3.1.2 Train/Test loss

```
Epoch 1/10: 100%|██████████| 782/782 [13:51<00:00,  1.06s/it]

Epoch 1, Train Loss:  3.58138
Epoch 1, Test Loss:  0.01135

Epoch 2/10: 100%|██████████| 782/782 [12:53<00:00,  1.01it/s]

Epoch 2, Train Loss:  1.91092
Epoch 2, Test Loss:  0.00128

Epoch 3/10: 100%|██████████| 782/782 [12:53<00:00,  1.01it/s]

Epoch 3, Train Loss:  1.26261
Epoch 3, Test Loss:  0.00165

Epoch 4/10: 100%|██████████| 782/782 [12:29<00:00,  1.04it/s]

Epoch 4, Train Loss:  0.80611
Epoch 4, Test Loss:  0.00041

Epoch 5/10: 100%|██████████| 782/782 [12:32<00:00,  1.04it/s]

Epoch 5, Train Loss:  0.54342
Epoch 5, Test Loss:  0.00556
```

Figure 3.2: Train/Test loss

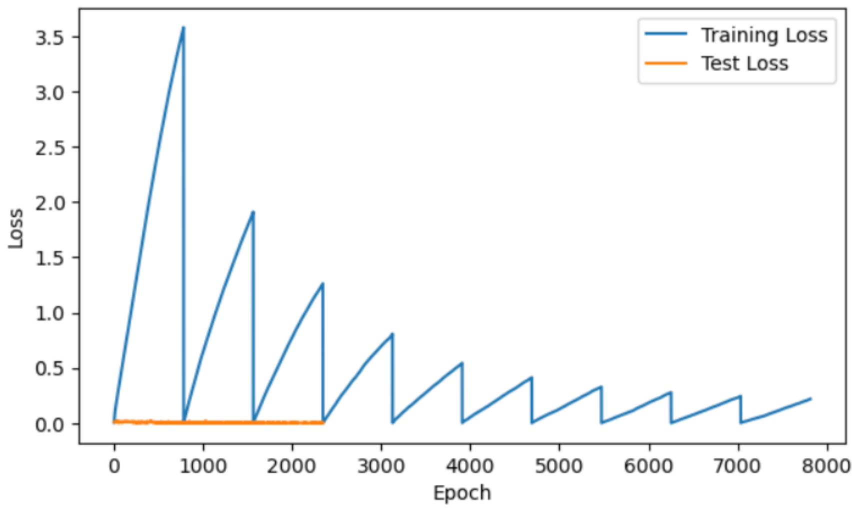

Figure 3.3: Loss Graph

3.1.3 Overall Accuracy

- For,

  Epoch=10

  Optimizer= Adam

  Learning Rate= 0.001

  Batch Size= 32

  Dataset: Bangla Handwritten Character Dataset

  Accuracy: 95.52

```
In [7]: # Calculate accuracy
        accuracy = accuracy_score(all_labels, all_preds)
        print(f'Overall Accuracy: {accuracy * 100:.2f}%')

        Overall Accuracy: 95.52%
```

Figure 3.4: Overall Accuracy

3.1.4 Classification

- The outcomes underscore the model's accuracy in predicting the correct output for the provided input.

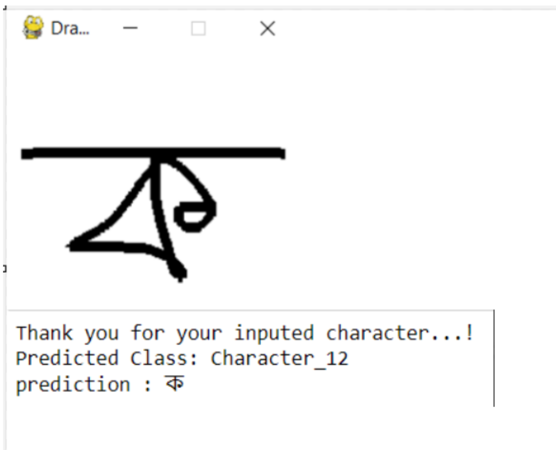

Figure 3.5: Model Predicts (ক)

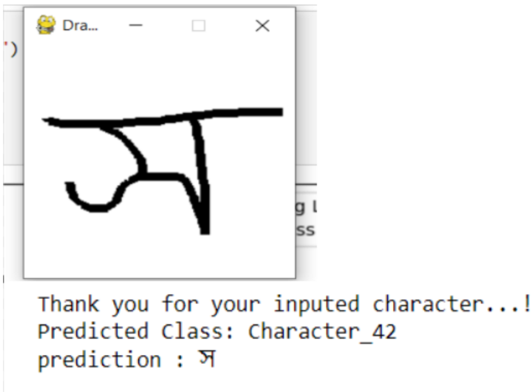

Figure 3.6: Model Predicts (স)

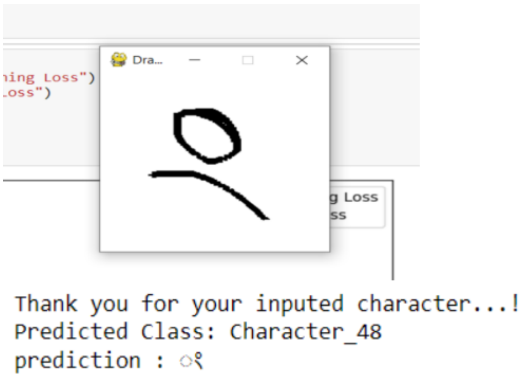

Figure 3.7: Model Predicts (ং)

3.1.5 Batch Size Tuning

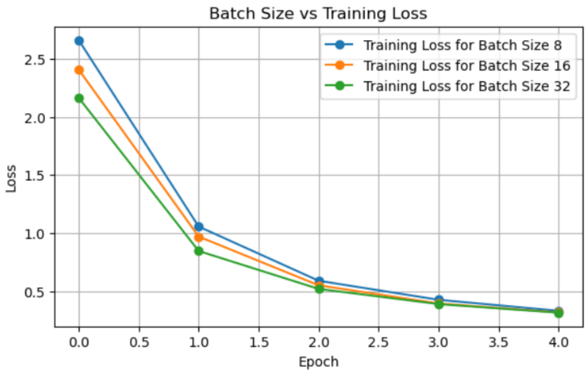

Figure 3.8: Batch Size vs Training Loss

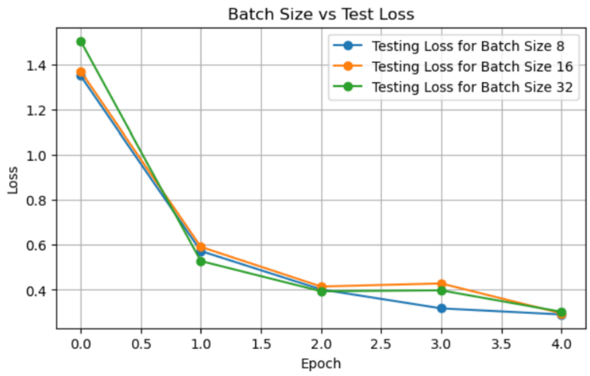

Figure 3.9: Batch Size vs Testing Loss

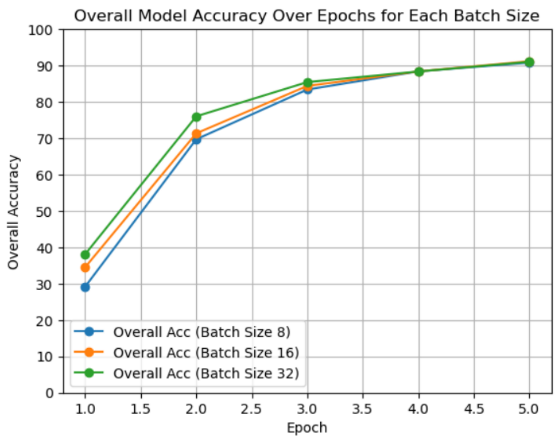

Figure 3.10: Batch Size vs Accuracy

3.1.6 Learning Rate Tuning

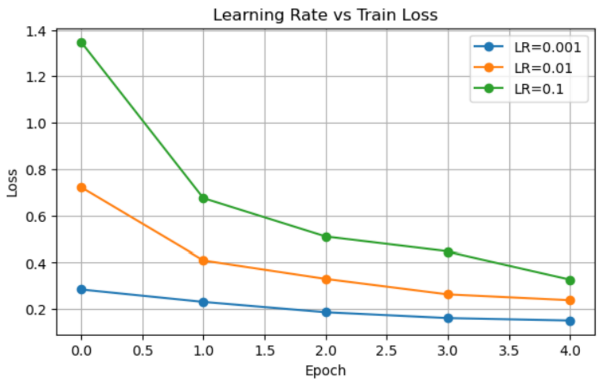

Figure 3.11: Learning Rate vs Training Loss

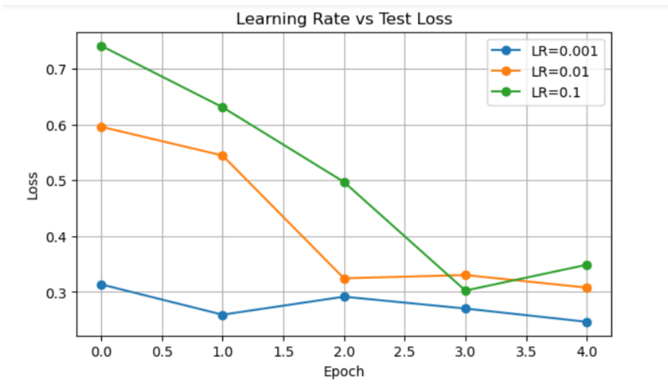

Figure 3.12: Learning Rate vs Testing Loss

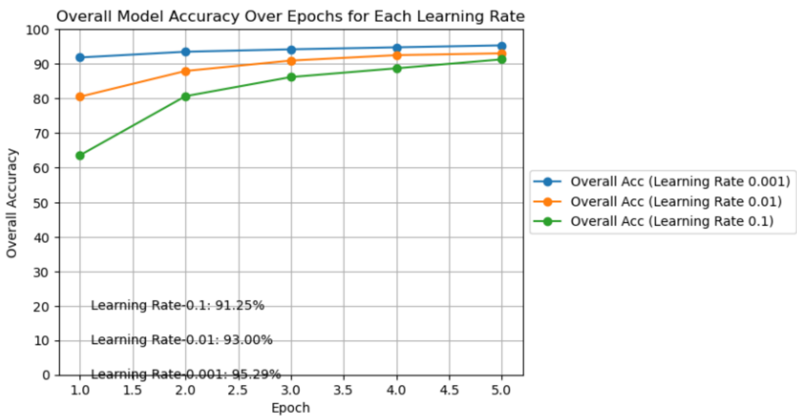

Figure 3.13: Learning Rate vs Accuracy

3.2 Results Overall Discussion

3.3 Overview of Results

The results demonstrate the effectiveness of the EfficientNetV2 model, showcasing accurate real-time prediction of Bangla handwritten characters. This success underscores the project's contribution to advancing accessibility and usability in digital platforms for the Bengali language.

3.3.1 Key Findings and Observations

1. **EfficientNetV2:**
    - EfficientNet elevates accuracy by efficiently scaling model dimensions, enabling superior feature extraction in Bangla handwritten character recognition.
    - Simultaneously, it reduces overall training delay through optimized architecture, ensuring faster convergence and effective utilization of computational resources.
    - EfficientNet outperforms other models by achieving superior accuracy with fewer parameters, providing a compelling combination of efficiency and performance in Bangla handwritten character recognition.
2. **Batch Size:**
    - A batch size of 32 facilitates improved training loss as it allows the model to efficiently generalize patterns across a larger set of training examples.
    - This larger batch size also contributes to reduced testing loss by enabling more stable and accurate validation during evaluation.
    - Ultimately, the enhanced overall accuracy with a batch size of 32 underscores the model's ability to effectively leverage larger data subsets, refining its learning and prediction capabilities.
3. **Learning Rate:**
    - A learning rate of 0.001 is conducive to lower training loss as it allows the model to make more refined weight adjustments during optimization, converging gradually towards an optimal solution.
    - This smaller learning rate contributes to decreased testing loss by preventing overshooting and promoting better generalization.
    - Ultimately, the superior overall accuracy with a learning rate of 0.001 reflects its effectiveness in facilitating a more nuanced and precise convergence of the model during training.

Overall, this project significantly improves real-time Bangla handwritten character prediction with EfficientNetV2, showcasing progress in character recognition and setting the stage for future advancements in deep learning.

Chapter 4

Conclusion

4.1 Discussion

This project focuses on advancing real-time Bangla handwritten character prediction through EfficientNetV2. Leveraging state-of-the-art deep learning techniques, EfficientNetV2 achieves accurate and efficient recognition of Bangla characters, making strides in enhancing accessibility and usability across digital platforms for the Bengali language.

4.2 Limitations

There are several limitations to consider when developing this project. Some of the main limitations include:

- **Data Diversity:** Limited availability of diverse Bangla handwritten character datasets may restrict the model's ability to generalize effectively across various handwriting styles and variations.
- **Computational Resources:** The resource-intensive nature of EfficientNetV2 may pose challenges for deployment on resource-constrained devices, limiting its accessibility in certain environments.
- **Translation Variability:** The model's performance could be influenced by variations in how individuals write Bangla characters, potentially leading to challenges in accurately predicting less conventional or stylized handwriting.
- **Localization Challenges:** The model may face difficulties in localizing characters within a larger context, potentially impacting its accuracy when characters are closely positioned or connected.

4.3 Scope of Future Work

There are several potential avenues for future development within this project. Some possible directions include:

- **Word-Level Prediction:** Extend the model to predict entire words, considering the contextual relationship between characters to enhance accuracy in word recognition.
- **Sentence Structure Understanding:** Explore techniques to predict sentence structures, taking into account the grammatical rules and linguistic patterns inherent in the Bengali language.
- **Sequential Learning:** Investigate sequential learning approaches to improve the model's ability to predict words and sentences coherently and sequentially, considering the order and flow of language.

These envisioned future steps are poised to overcome current limitations, fostering advancements in performance, scalability, and security within the project.

References

[1] MatriVasha: Bangla Handwritten Compound Character Dataset and Recognition || Retrieved from https://data.mendeley.com/datasets/v39pc2g2wp/1?fbclid=IwAR103up9nbfSpEcqgR3R4y6TwvmOU2xq6WSUTsxPghlRFSixA5iyJOkSBpY

[2] EfficientNetV2:EfficientNetV2: Faster, Smaller, and Higher Accuracy than Vision Transformers || Retrieved from https://towardsdatascience.com/efficientnetv2-faster-smaller-and-higher-accuracy-than-vision-transformers-98e2

[3] Keras.io || Retrieved from https://keras.io/api/applications/efficientnet_v2/

[4] EfficientNetV2-based dynamic gesture recognition using transformed scalogram from triaxial acceleration signal https://academic.oup.com/jcde/article/10/4/1694/7218563

[5] EfficientNetv2 Model for Breast Cancer Histopathological Image Classification https://ieeexplore.ieee.org/abstract/document/9750693