AI vs. Human-Generated Images Detection

Course: INFO-6147
Student: Harrison Kim, 1340629
Date: August 15, 2025

0. Environment Setup

```
!pip install -q torch torchvision kagglehub pandas tqdm matplotlib scikit-learn numpy
```

```
import torch

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'Using device: {device}')
```

```
Using device: cuda
```

1. Dataset Selection:

- Choose a suitable image dataset for your project. You can consider any of the well-known datasets here Datasets — [Torchvision 0.17 documentation] for simplicity.
- Ensure that the dataset contains a reasonable number of classes and a sufficient number of images per class.
- If the dataset has very large number of images, you can use a subset (e.g., 1000 images per class if number of classes are 10 or less)
- If the dataset has more than 20 classes, you can use a subset of the classes (e.g., only use 10 classes)

[ ] ↳ 8 cells hidden

2. Data Preprocessing:

- Perform data preprocessing steps such as resizing images, normalizing pixel values, and splitting the dataset into training, validation, and test sets.
- Apply data augmentation techniques to increase the diversity of the training data.

2.1 Perform data preprocessing steps such as resizing images, normalizing pixel values, and splitting the dataset into training, validation, and test sets.

```
import torch
from torch.utils.data import Dataset
from PIL import Image

# Define ImageDataset class
class ImageDataset(Dataset):
    def __init__(self, dataframe, transform=None):
        self.dataframe = dataframe
        self.transform = transform
        self.image_paths = dataframe['image_path'].values
        self.labels = dataframe['label'].values
        self.label_to_idx = {label: idx for idx, label in enumerate(sorted(set(self.labels)))}

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

```

```
    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        label = self.labels[idx]
        label_idx = self.label_to_idx[label]
        try:
            img = Image.open(img_path).convert('RGB')
            if self.transform:
                img = self.transform(img)
        except Exception as e:
            print(f'Error loading image {img_path}:', e)
            img = torch.zeros(3, 32, 32)
        return img, label_idx

# 1. Split df into train, validation and test datasets
from sklearn.model_selection import train_test_split

temp = df[df['split'] == 'train'].reset_index(drop=True)
train_df, val_df = train_test_split(
    temp,
    test_size=0.2,
    stratify=temp['label'],
    random_state=42
)
test_df = df[df['split'] == 'test'].reset_index(drop=True)

# 2. Define transform (resize, normalize)
from torchvision import transforms
transform = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
])

# 3. Create ImageDataset for each split
# train_dataset = ImageDataset(train_df, transform=transform)
val_dataset = ImageDataset(val_df, transform=transform)
test_dataset = ImageDataset(test_df, transform=transform)

print('Train set size:', len(train_df))
print('Validation set size:', len(val_dataset))
print('Test set size:', len(test_dataset))
```

```
Train set size: 40000
Validation set size: 10000
Test set size: 10000
```

2.2 Apply data augmentation techniques to increase the diversity of the training data.

```
from torchvision import transforms

# Define data augmentation for training
train_transform = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
])

# Re-create train_dataset with augmentation
train_dataset = ImageDataset(train_df, transform=train_transform)
print('Data augmentation applied to training dataset.')
```

```
Data augmentation applied to training dataset.
```

3. Model Selection and Architecture:

- Select an appropriate deep learning architecture for image classification. You can start with a convolutional neural network (CNN).
- Define the architecture of your model, including the number of layers, activation functions, and any regularization techniques.

3.1 Select an appropriate deep learning architecture for image classification. You can start with a convolutional neural network (CNN).

- The CNN model uses Conv2d, ReLU, MaxPool2d, and Linear layers.

3.2 Define the architecture of your model, including the number of layers, activation functions, and any regularization techniques.

3.2.1 Define a custom CNN model with:

- **Conv2d**: This layer extracts key features from the input image, such as edges, textures, and shapes, which are essential for classification.
- **ReLU**: The activation function introduces non-linearity, enabling the model to learn more complex and abstract patterns beyond linear relationships.
- **MaxPool**: By reducing the spatial dimensions of the feature maps, this layer retains the most important information and helps prevent overfitting, while also improving computational efficiency.
- **Linear**: The fully connected layer combines all extracted features and produces the final output, determining the predicted class for each image.

```
import torch.nn as nn

class Net(nn.Module):
    def __init__(self, num_classes):
        super(Net, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(16, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(64 * 4 * 4, 128),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x

model = Net(num_classes=num_classes)

print("CNN model with dropout(0.5) initialized.")
```

```
CNN model with dropout(0.5) initialized.
```

4. Model Training:

- Train your deep learning model using the training dataset.
- Monitor training progress, including loss and accuracy, and consider using early stopping to prevent overfitting.

```
# Initial hyperparameter values
BATCH_SIZE = 64
NUM_EPOCHS = 25
LR = 0.0001
```

4.1 Train your deep learning model using the training dataset.

4.1.1 Load dataset into dataloaders

```
from torch.utils.data import DataLoader

train_dataset = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_dataset = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)
test_dataset = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)
```

4.1.2 Train custom CNN model

```
# Training loop for custom CNN model
import torch.optim as optim
from tqdm import tqdm
import copy

model = model.to(device)
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = optim.Adam(model.parameters(), lr=LR)

# Lists to store metrics
cnn_train_loss_list = []
cnn_train_acc_list = []
cnn_val_loss_list = []
cnn_val_acc_list = []

best_val_acc = 0.0
counter = 0
for epoch in range(NUM_EPOCHS):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for images, labels in tqdm(train_dataset, desc=f"Epoch {epoch+1}/{NUM_EPOCHS}"):
        images = images.to(device)
        if not isinstance(labels, torch.Tensor):
            labels = torch.tensor(labels)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * images.size(0)
        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)
    train_loss = running_loss / total
    train_acc = correct / total

```

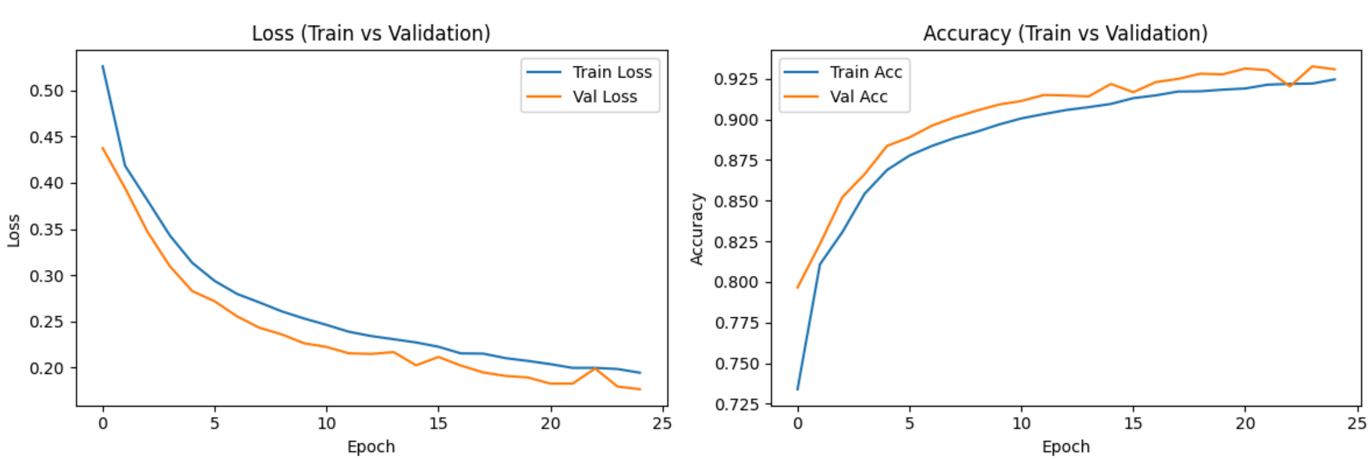
```
    # Store metrics
    cnn_train_loss_list.append(train_loss)
    cnn_train_acc_list.append(train_acc)

    # Calculate and record validation loss
    model.eval()
    val_loss = 0.0
    val_correct = 0
    val_total = 0
    with torch.no_grad():
        for images, labels in val_dataset:
            images = images.to(device)
            if not isinstance(labels, torch.Tensor):
                labels = torch.tensor(labels)
            labels = labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            val_loss += loss.item() * images.size(0)
            _, predicted = torch.max(outputs, 1)
            val_correct += (predicted == labels).sum().item()
            val_total += labels.size(0)
    val_loss = val_loss / val_total
    val_acc = val_correct / val_total
    cnn_val_loss_list.append(val_loss)
    cnn_val_acc_list.append(val_acc)

    if val_acc > best_val_acc:
        best_val_acc = val_acc
        counter = 0
        best_model_state = copy.deepcopy(model.state_dict())
    else:
        counter += 1
        if counter >= 5:
            print("Early stopping triggered.")
            break

    print(f"Epoch {epoch+1}/{NUM_EPOCHS} | Train Loss: {train_loss:.4f} | Train Acc: {train_acc:.4f}")

print("\nTraining complete.")
print("cnn_train_loss_list:", cnn_train_loss_list)
print("cnn_train_acc_list:", cnn_train_acc_list)
```

```
Epoch 1/25: 100%|██████████| 625/625 [00:51<00:00, 12.17it/s]
Epoch 1/25 | Train Loss: 0.5261 | Train Acc: 0.7340
Epoch 2/25: 100%|██████████| 625/625 [00:45<00:00, 13.77it/s]
Epoch 2/25 | Train Loss: 0.4184 | Train Acc: 0.8108
Epoch 3/25: 100%|██████████| 625/625 [00:45<00:00, 13.66it/s]
Epoch 3/25 | Train Loss: 0.3809 | Train Acc: 0.8306
Epoch 4/25: 100%|██████████| 625/625 [00:45<00:00, 13.82it/s]
Epoch 4/25 | Train Loss: 0.3429 | Train Acc: 0.8543
Epoch 5/25: 100%|██████████| 625/625 [00:45<00:00, 13.83it/s]
Epoch 5/25 | Train Loss: 0.3133 | Train Acc: 0.8689
Epoch 6/25: 100%|██████████| 625/625 [00:45<00:00, 13.68it/s]
Epoch 6/25 | Train Loss: 0.2937 | Train Acc: 0.8777
Epoch 7/25: 100%|██████████| 625/625 [00:48<00:00, 12.79it/s]
Epoch 7/25 | Train Loss: 0.2796 | Train Acc: 0.8836
Epoch 8/25: 100%|██████████| 625/625 [00:47<00:00, 13.16it/s]
Epoch 8/25 | Train Loss: 0.2704 | Train Acc: 0.8885
Epoch 9/25: 100%|██████████| 625/625 [00:45<00:00, 13.88it/s]
Epoch 9/25 | Train Loss: 0.2607 | Train Acc: 0.8924
Epoch 10/25: 100%|██████████| 625/625 [00:45<00:00, 13.84it/s]
Epoch 10/25 | Train Loss: 0.2529 | Train Acc: 0.8968
Epoch 11/25: 100%|██████████| 625/625 [00:44<00:00, 14.14it/s]
Epoch 11/25 | Train Loss: 0.2461 | Train Acc: 0.9006
Epoch 12/25: 100%|██████████| 625/625 [00:45<00:00, 13.88it/s]
Epoch 12/25 | Train Loss: 0.2386 | Train Acc: 0.9032
Epoch 13/25: 100%|██████████| 625/625 [00:45<00:00, 13.68it/s]
Epoch 13/25 | Train Loss: 0.2340 | Train Acc: 0.9058
Epoch 14/25: 100%|██████████| 625/625 [00:46<00:00, 13.52it/s]
Epoch 14/25 | Train Loss: 0.2305 | Train Acc: 0.9075
Epoch 15/25: 100%|██████████| 625/625 [00:44<00:00, 13.93it/s]
Epoch 15/25 | Train Loss: 0.2271 | Train Acc: 0.9095
Epoch 16/25: 100%|██████████| 625/625 [00:45<00:00, 13.88it/s]
```

```
Epoch 16/25 | Train Loss: 0.2224 | Train Acc: 0.9131
Epoch 17/25: 100%|██████████| 625/625 [00:44<00:00, 14.01it/s]
Epoch 17/25 | Train Loss: 0.2152 | Train Acc: 0.9147
Epoch 18/25: 100%|██████████| 625/625 [00:44<00:00, 13.93it/s]
Epoch 18/25 | Train Loss: 0.2149 | Train Acc: 0.9171
Epoch 19/25: 100%|██████████| 625/625 [00:45<00:00, 13.80it/s]
Epoch 19/25 | Train Loss: 0.2100 | Train Acc: 0.9173
Epoch 20/25: 100%|██████████| 625/625 [00:47<00:00, 13.12it/s]
Epoch 20/25 | Train Loss: 0.2070 | Train Acc: 0.9183
Epoch 21/25: 100%|██████████| 625/625 [00:44<00:00, 14.00it/s]
Epoch 21/25 | Train Loss: 0.2036 | Train Acc: 0.9190
Epoch 22/25: 100%|██████████| 625/625 [00:44<00:00, 14.06it/s]
Epoch 22/25 | Train Loss: 0.1995 | Train Acc: 0.9213
Epoch 23/25: 100%|██████████| 625/625 [00:44<00:00, 14.05it/s]
Epoch 23/25 | Train Loss: 0.1995 | Train Acc: 0.9218
Epoch 24/25: 100%|██████████| 625/625 [00:43<00:00, 14.24it/s]
Epoch 24/25 | Train Loss: 0.1983 | Train Acc: 0.9220
Epoch 25/25: 100%|██████████| 625/625 [00:45<00:00, 13.87it/s]
Epoch 25/25 | Train Loss: 0.1943 | Train Acc: 0.9246

Training complete.
cnn_train_loss_list: [0.5260564805984497, 0.4183649462223053, 0.3809396066904068, 0.34290345828533175, 0.31329
cnn_train_acc_list: [0.734, 0.8108, 0.8306, 0.8543, 0.868875, 0.877725, 0.883625, 0.888475, 0.8924, 0.896825, 0
```

4.2 Monitor training progress, including loss and accuracy, and consider using early stopping to prevent overfitting.

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(cnn_train_loss_list, label='Train Loss')
plt.plot(cnn_val_loss_list, label='Val Loss')
plt.title('Loss (Train vs Validation)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1,2,2)
plt.plot(cnn_train_acc_list, label='Train Acc')
plt.plot(cnn_val_acc_list, label='Val Acc')
plt.title('Accuracy (Train vs Validation)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.tight_layout()
plt.show()
```



5. Hyperparameter Tuning:

- Experiment with different hyperparameters (e.g., learning rate, batch size) to optimize the model's performance.
- Keep a record of the hyperparameters used and their impact on the model.

5.1 Experiment with different hyperparameters (e.g., learning rate, batch size) to optimize the model's performance.

```
# Hyperparameter search for learning rate and batch size
import copy
from tqdm import tqdm

search_learning_rates = [0.0005, 0.001, 0.005]
search_batch_sizes = [128, 192, 256]
num_epochs_hp = 5

results = []

for lr in search_learning_rates:
    for batch_size in search_batch_sizes:
        print(f"\nTraining with learning rate={lr}, batch size={batch_size}")
        # Re-create dataloaders with new batch size
        train_loader = DataLoader(train_dataset.dataset, batch_size=batch_size, shuffle=True)
        val_loader = DataLoader(val_dataset.dataset, batch_size=batch_size, shuffle=False)

        # Re-initialize model and optimizer
        model = Net(num_classes=num_classes).to(device)
        optimizer = optim.Adam(model.parameters(), lr=lr)

        best_val_acc = 0.0
        counter = 0
        for epoch in range(num_epochs_hp):
            model.train()
            running_loss = 0.0
            correct = 0
            total = 0
            for images, labels in tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs_hp} (train)"):
                images = images.to(device)
                if not isinstance(labels, torch.Tensor):
                    labels = torch.tensor(labels)
                labels = labels.to(device)
                optimizer.zero_grad()
                outputs = model(images)
                loss = criterion(outputs, labels)
                loss.backward()
                optimizer.step()
                running_loss += loss.item() * images.size(0)
                _, predicted = torch.max(outputs, 1)
                correct += (predicted == labels).sum().item()
                total += labels.size(0)
            train_loss = running_loss / total
            train_acc = correct / total

            # Validation
            model.eval()
            val_loss = 0.0
            val_correct = 0
            val_total = 0
            with torch.no_grad():
                for images, labels in tqdm(val_dataset, desc="Validating"):
                    images = images.to(device)
                    if not isinstance(labels, torch.Tensor):
                        labels = torch.tensor(labels)
                    labels = labels.to(device)
                    outputs = model(images)
                    loss = criterion(outputs, labels)
```

```
                    val_loss += loss.item() * images.size(0)
                    _, predicted = torch.max(outputs, 1)
                    val_correct += (predicted == labels).sum().item()
                    val_total += labels.size(0)
            val_loss = val_loss / val_total
            val_acc = val_correct / val_total

            if val_acc > best_val_acc:
                best_val_acc = val_acc
                counter = 0
                best_model_state = copy.deepcopy(model.state_dict())
            else:
                counter += 1
                if counter >= 3:
                    print("Early stopping triggered.")
                    break

        print(f"\nBest validation accuracy: {best_val_acc:.4f}")
        results.append({
            'learning_rate': lr,
            'batch_size': batch_size,
            'best_val_acc': best_val_acc
        })
```

```
Training with learning rate=0.0005, batch size=128
Epoch 1/5 (train): 100%|██████████| 313/313 [00:43<00:00,  7.23it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 16.84it/s]
Epoch 2/5 (train): 100%|██████████| 313/313 [00:43<00:00,  7.14it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 16.99it/s]
Epoch 3/5 (train): 100%|██████████| 313/313 [00:49<00:00,  6.26it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 16.84it/s]
Epoch 4/5 (train): 100%|██████████| 313/313 [00:44<00:00,  7.03it/s]
Validating: 100%|██████████| 157/157 [00:08<00:00, 17.96it/s]
Epoch 5/5 (train): 100%|██████████| 313/313 [00:43<00:00,  7.17it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 16.99it/s]
Best validation accuracy: 0.9113

Training with learning rate=0.0005, batch size=192
Epoch 1/5 (train): 100%|██████████| 209/209 [00:43<00:00,  4.84it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 16.80it/s]
Epoch 2/5 (train): 100%|██████████| 209/209 [00:43<00:00,  4.79it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 17.05it/s]
Epoch 3/5 (train): 100%|██████████| 209/209 [00:43<00:00,  4.77it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 17.24it/s]
Epoch 4/5 (train): 100%|██████████| 209/209 [00:43<00:00,  4.85it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 16.80it/s]
Epoch 5/5 (train): 100%|██████████| 209/209 [00:42<00:00,  4.94it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 17.41it/s]
Best validation accuracy: 0.8983

Training with learning rate=0.0005, batch size=256
Epoch 1/5 (train): 100%|██████████| 157/157 [00:42<00:00,  3.66it/s]
Validating: 100%|██████████| 157/157 [00:08<00:00, 17.88it/s]
Epoch 2/5 (train): 100%|██████████| 157/157 [00:43<00:00,  3.60it/s]
Validating: 100%|██████████| 157/157 [00:08<00:00, 17.85it/s]
Epoch 3/5 (train): 100%|██████████| 157/157 [00:42<00:00,  3.69it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 17.06it/s]
Epoch 4/5 (train): 100%|██████████| 157/157 [00:42<00:00,  3.70it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 17.03it/s]
Epoch 5/5 (train): 100%|██████████| 157/157 [00:43<00:00,  3.63it/s]
Validating: 100%|██████████| 157/157 [00:08<00:00, 18.21it/s]
Best validation accuracy: 0.9041

Training with learning rate=0.001, batch size=128
Epoch 1/5 (train): 100%|██████████| 313/313 [00:44<00:00,  7.11it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 16.92it/s]
Epoch 2/5 (train): 100%|██████████| 313/313 [00:43<00:00,  7.22it/s]
Validating: 100%|██████████| 157/157 [00:08<00:00, 17.58it/s]
Epoch 3/5 (train): 100%|██████████| 313/313 [00:43<00:00,  7.13it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 17.16it/s]
Epoch 4/5 (train): 100%|██████████| 313/313 [00:43<00:00,  7.18it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 17.20it/s]
```

```
Epoch 5/5 (train): 100%|██████████| 313/313 [00:44<00:00,  7.07it/s]
Validating: 100%|██████████| 157/157 [00:08<00:00, 17.83it/s]
Best validation accuracy: 0.9261

Training with learning rate=0.001, batch size=192
Epoch 1/5 (train): 100%|██████████| 209/209 [00:43<00:00,  4.84it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 16.27it/s]
Epoch 2/5 (train): 100%|██████████| 209/209 [00:43<00:00,  4.83it/s]
Validating: 100%|██████████| 157/157 [00:09<00:00, 17.06it/s]
```

5.2 Keep a record of the hyperparameters used and their impact on the model.

```
import pandas as pd

# Create a DataFrame to summarize hyperparameter search results
results_df = pd.DataFrame(results)
print("Hyperparameter summary table:")
print(results_df)

# Optionally, display the best configuration
best_row = results_df.loc[results_df['best_val_acc'].idxmax()]
print(f"\nBest configuration:\nLearning rate: {best_row['learning_rate']}, Batch size: {best_row['batch_size']
```

```
Hyperparameter summary table:
   learning_rate  batch_size  best_val_acc
0         0.0005         128        0.9113
1         0.0005         192        0.8983
2         0.0005         256        0.9041
3         0.0010         128        0.9261
4         0.0010         192        0.9208
5         0.0010         256        0.9219
6         0.0050         128        0.9121
7         0.0050         192        0.9212
8         0.0050         256        0.9209

Best configuration:
Learning rate: 0.001, Batch size: 128.0, Best Val Acc: 0.9261
```

6. Evaluation:

- Evaluate your trained model using the validation dataset to assess its performance.
- Calculate relevant metrics such as accuracy, precision, recall, and F1-score.
- Visualize the model's predictions and misclassifications.

6.1 Evaluate your trained model using the validation dataset to assess its performance.

```
# Evaluate CNN model on validation dataset
model.eval()
val_loss = 0.0
val_correct = 0
val_total = 0
cnn_true = []
cnn_pred = []
with torch.no_grad():
    for images, labels in tqdm(val_dataset, desc="Validating"):
        images = images.to(device)
        if not isinstance(labels, torch.Tensor):
            labels = torch.tensor(labels)
        labels = labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)
        val_loss += loss.item() * images.size(0)
        _, predicted = torch.max(outputs, 1)
        val_correct += (predicted == labels).sum().item()
        val_total += labels.size(0)
```

```
        cnn_true.extend(labels.cpu().numpy())
        cnn_pred.extend(predicted.cpu().numpy())
val_loss = val_loss / val_total
val_acc = val_correct / val_total
print(f"\nCNN Validation Loss: {val_loss:.4f} | CNN Validation Accuracy: {val_acc:.4f}")
```

```
Validating: 100%|██████████| 157/157 [00:09<00:00, 17.06it/s]CNN Validation Loss: 0.2834 | CNN Validation Accur
```

6.2 Calculate relevant metrics such as accuracy, precision, recall, and F1-score.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# CNN metrics
cnn_acc = accuracy_score(cnn_true, cnn_pred)
cnn_prec = precision_score(cnn_true, cnn_pred, average='macro')
cnn_rec = recall_score(cnn_true, cnn_pred, average='macro')
cnn_f1 = f1_score(cnn_true, cnn_pred, average='macro')
print('CNN metrics:')
print(f'Accuracy: {cnn_acc:.4f}, Precision: {cnn_prec:.4f}, Recall: {cnn_rec:.4f}, F1-score: {cnn_f1:.4f}')
```

```
CNN metrics:
Accuracy: 0.8758, Precision: 0.8920, Recall: 0.8758, F1-score: 0.8745
```

```
Start coding or generate with AI.
```

8. Final Model Testing:

- Test your final model on the held-out test dataset to assess its generalization to unseen data.

8.1 Test your final model on the held-out test dataset to assess its generalization to unseen data.

```
model.eval()
test_true = []
test_pred = []
with torch.no_grad():
    for images, labels in test_dataset:
        images = images.to(device)
        if not isinstance(labels, torch.Tensor):
            labels = torch.tensor(labels)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        test_true.extend(labels.cpu().numpy())
        test_pred.extend(predicted.cpu().numpy())

test_acc = accuracy_score(test_true, test_pred)
test_prec = precision_score(test_true, test_pred, average='macro')
test_rec = recall_score(test_true, test_pred, average='macro')
test_f1 = f1_score(test_true, test_pred, average='macro')
print(f'Test Accuracy: {test_acc:.4f}, Precision: {test_prec:.4f}, Recall: {test_rec:.4f}, F1-score: {test_f1:
```

```
Test Accuracy: 0.8759, Precision: 0.8919, Recall: 0.8759, F1-score: 0.8746
```

9. Documentation and Reporting:

- Create a project report summarizing your dataset, model architecture, training process, evaluation results, and insights gained.
- Include visualizations and explanations to make your findings clear.

9.1 Create a project report summarizing your dataset, model architecture, training process, evaluation results, and insights gained.

```
Start coding or generate with AI.
```

9.2 Include visualizations and explanations to make your findings clear.

```
Start coding or generate with AI.
```

10. Presentation:

- Prepare a brief presentation to showcase your project's key findings and outcomes.
- Share your experiences, challenges faced, and lessons learned during the project.

10.1 Prepare a brief presentation to showcase your project's key findings and outcomes.

10.2 Share your experiences, challenges faced, and lessons learned during the project.

11. Conclusion:

- Conclude your capstone project by summarizing your achievements and any future work or improvements that could be made to the model.
- Remember to maintain good coding practices and seek guidance or feedback from your instructor throughout the project.
- This capstone project will demonstrate your ability to apply deep learning techniques to real-world problems and showcase your skills to potential employers or collaborators.

11.1 Conclude your capstone project by summarizing your achievements and any future work or improvements that could be made to the model.

11.2 Remember to maintain good coding practices and seek guidance or feedback from your instructor throughout the project.

11.3 This capstone project will demonstrate your ability to apply deep learning techniques to real-world problems and showcase your skills to potential employers or collaborators.