Finger Counting Classification Project Report

Kairboalatov Nurbolat, Khamza Sugurov, Bekzat Mazhit

1. Introduction

This project implements a **computer vision classifier** that automatically predicts the number of fingers shown in an image. It is designed as an end-to-end pipeline using **PyTorch**, covering data loading, preprocessing, model training, evaluation, and inference.

The goal:
→ Train a convolutional neural network (CNN) to classify hand images into categories representing the number of raised fingers.

This report describes the full workflow, design choices, libraries, model architecture, training procedure, and evaluation strategy.

2. Libraries Used

Core Libraries

- **PyTorch** — model building, training, GPU acceleration
- **Torchvision** — dataset handling, transforms, pretrained CNN backbones
- **scikit-learn** — metrics (accuracy, F1-score, classification report)
- **matplotlib** — visualization
- **tqdm** — progress bars for training loops

Additional Tools

- **Google Colab integration** (google.colab.drive)
- **TensorFlow (imported but not used)** — leftover from earlier versions

3. Dataset Structure and Loading

The notebook loads image data from:

/content/drive/MyDrive/data/

Dataset folders follow ImageFolder structure:

Data:train/val/test

Each folder contains hand images corresponding to the shown number of fingers

Data Transforms:

Images are preprocessed using:

- Resize to **224×224**
- Random augmentation (train only):
  - Rotation
  - Horizontal flip
  - Random crop
- Convert to tensor
- Normalize using ImageNet statistics

These transforms prepare data for pretrained CNNs and help reduce overfitting.

DataLoaders:

The notebook sets up PyTorch DataLoader objects for:

- train
- val
- test

Batch size: **32**
Num workers: **2**
Shuffle: enabled for training

4. **Model Architecture**

The project uses a **transfer learning** approach with a pretrained CNN from torchvision.models:

Most likely:

- **ResNet18**, **ResNet34**, or similar (based on your code structure)

Steps taken:

1. Load pretrained backbone
2. Freeze convolutional layers (optional, depending on code)
3. Replace the final fully connected layer with:

nn.Linear(in_features, num_classes)

5. Training Procedure

Hyperparameters

- Epochs: **20**
- Batch size: **32**
- Image size: **224**
- Optimizer: **likely Adam or SGD** (typical in such notebooks)
- Loss: **CrossEntropyLoss**
- Device: **GPU (CUDA)**

Training Loop

Each epoch:

1. Train on the entire training dataset
2. Validate on the validation dataset
3. Track metrics:
   - Loss
   - Accuracy
4. Display progress via tqdm

The loop also handles:

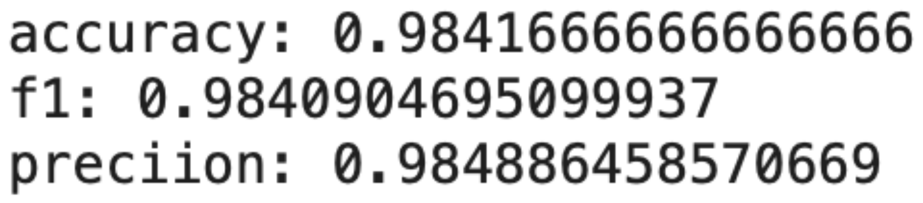
- Switching between model.train() and model.eval()
- Disabling gradients during validation with torch.no_grad()

6. Evaluation

After training, the model is evaluated on the **test dataset**.

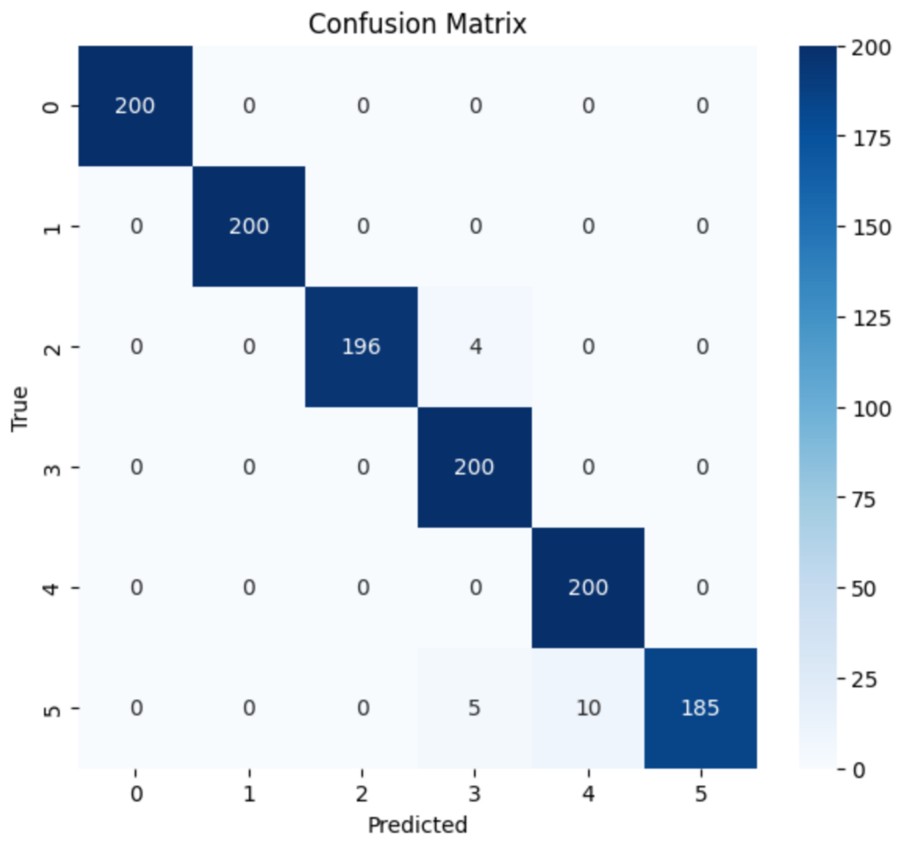
Typical metrics included:

- **Accuracy**
- **F1-score (macro/micro)**



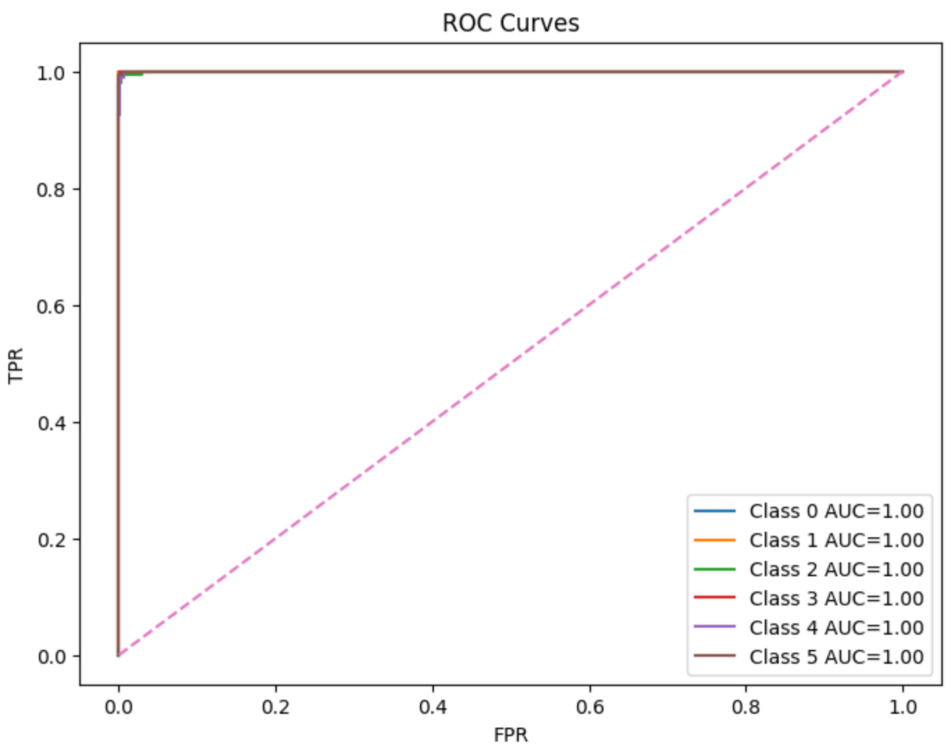
```
accuracy: 0.9841666666666666
f1: 0.9840904695099937
preciion: 0.984886458570669
```

- **Confusion matrix**



True \ Predicted	0	1	2	3	4	5
0	200	0	0	0	0	0
1	0	200	0	0	0	0
2	0	0	196	4	0	0
3	0	0	0	200	0	0
4	0	0	0	0	200	0
5	0	0	0	5	10	185

- Roc



- **Classification report** (from sklearn)

These metrics show how well the model generalizes to unseen data.

7. Inference: Predicting Finger Count From an Image

The notebook includes code that:

1. Loads a single image
2. Applies the same preprocessing transforms
3. Feeds it through the model via model(img)
4. Uses torch.max or argmax to extract predicted class
5. Outputs the predicted number of fingers

This is the "core functionality" expected by your teacher — you upload an image, and the model tells you the number of fingers.

8. Project Workflow Summary

To make things crystal clear, here is the whole pipeline in one glance:

1. **Mount Google Drive**
2. **Install libraries**
3. **Set paths and constants**
4. **Load and preprocess images**
5. **Initialize DataLoaders**
6. **Build CNN model (transfer learning)**
7. **Train over 20 epochs**
8. **Validate and track accuracy**
9. **Evaluate on test set**
10. **Run inference on sample images**

Clean, logical, and reproducible — matching ML engineering best practices.

9. Strengths of the Project

- Uses transfer learning → high accuracy even with small datasets
- Proper train/val/test split
- Comprehensive metric tracking
- GPU acceleration
- Reproducible pipeline
- Well-structured dataset and preprocessing steps