Sign Language translator using CNN and CVZONE

-by Yadhu Kishan T R

Summary

- This project implements a Convolution Neural Network (CNN) to classify sign language gesture
- The dataset used is “Americal Sign Language (ASL) alphabets”
- The dataset was downloaded from Kaggle : DatasetLink
- The Test dataset showed 100% accuracy but the performance from real time testing using webcam was very low

About the Dataset

- The Sign Language MNIST dataset structurally similar to original MNIST
- It is in CSV format with label and pixel values in one row
- Each training and test case represents a label (0-25) as a one-to-one map for each alphabetic letter A-Z (and no cases for 9=J or 25=Z because of gesture motions)
- Each row represent a single 28 * 28 pixel image with grayscale values between 0 – 255
- It has 27,455 images for training and 7127 images for testing

Implementation

- I used Google Colab to train sign language MNIST dataset and develop CNN
- Live hand gesture detection was implemention in Pycharm using cvzone.HandTrackingModule to capture hand gestures in real-time
- This project was implemented based on a video tutorial "Easy Hand Sign Detection | American Sign Language | Computer Vision" by Murtaza's Workshop :- Link

Feature	Tutroial Version	My Project
Model Format	.h5	.pth
Model Creation	Used google Teachable machine	Custom CNN implemented in PyTorch
Training	Done through web interface	Fully Coded and trained in Google Colab
Dataset	Images of hand gestures	Sign Language MNIST from kaggle (CSV format)

Implementation Steps

1: Data Preparation

- The dataset had more than 27,000 images as labels and pixel values, so for ease of use I prepared another dataset using pandas in python with just the labels 0,1 and2, which are signs A,B and C, for both training data and testing data

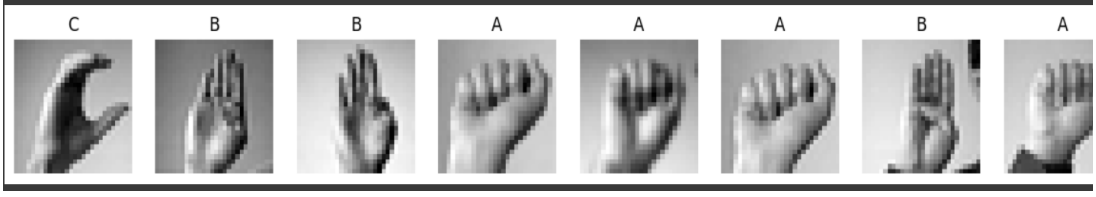
```python
import pandas as pd
from google.colab import files
print("Upload your CSV file (train or test):")
uploaded = files.upload()
file_name = list(uploaded.keys())[0]

df = pd.read_csv(file_name)
print("Original data shape:", df.shape)

filtered_df = df[df['label'].isin([0, 1, 2])]
print("Filtered data shape:", filtered_df.shape)

output_file = "filtered_0_1_2.csv"
filtered_df.to_csv(output_file, index=False)
print(f"Filtered dataset saved as {output_file}")

files.download(output_file)
```

```
Upload your CSV file (train or test):
Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving sign_mnist_test.csv to sign_mnist_test.csv
Original data shape: (7172, 785)
Filtered data shape: (1073, 785)
Filtered dataset saved as filtered_0_1_2.csv
```

- The CSV dataset with the pixel values is then converted into 28*28 grayscale image tensors, normalizes the pixel values and wraps them into PyTorch DataLoaders for batching and shuffling during training and testing of my CNN

```python
from torch.utils.data import DataLoader, TensorDataset
X_train = train_df.drop('label', axis=1).values.astype('float32').reshape(-1, 1, 28, 28)
y_train = train_df['label'].values.astype('int64')
X_test  = test_df.drop('label', axis=1).values.astype('float32').reshape(-1, 1, 28, 28)
y_test  = test_df['label'].values.astype('int64')
X_train = torch.tensor(X_train) / 255.0  # scale 0-1
y_train = torch.tensor(y_train)
X_test  = torch.tensor(X_test) / 255.0
y_test  = torch.tensor(y_test)
mean, std = 0.5, 0.5
X_train = (X_train - mean) / std
X_test  = (X_test - mean) / std
train_loader = DataLoader(TensorDataset(X_train, y_train), batch_size=BATCH_SIZE, shuffle=True)
test_loader  = DataLoader(TensorDataset(X_test, y_test), batch_size=BATCH_SIZE, shuffle=False)
```

```python
import matplotlib.pyplot as plt
images, labels = next(iter(train_loader))
images = images * 0.5 + 0.5
fig, axes = plt.subplots(1, 8, figsize=(15,2))
label_map = {0:'A', 1:'B', 2:'C'}
for i in range(8):
    axes[i].imshow(images[i].squeeze(), cmap='gray')
    axes[i].set_title(f"{label_map[labels[i].item()]}")
    axes[i].axis('off')

plt.show()
```



2: Creating and Training a CNN model

- A simple Convolutional Neural Network (CNN) for classifying 28*28 grayscale images into three classes is created,

```python
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.pool  = nn.MaxPool2d(2, 2)
        self.fc1   = nn.Linear(64 * 7 * 7, 128)
        self.fc2   = nn.Linear(128, 3)   # only 3 classes
        self.relu  = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
        x = self.relu(self.conv2(x))
        x = self.pool(x)
        x = x.view(-1, 64 * 7 * 7)
        x = self.relu(self.fc1(x))
        return self.fc2(x)

model = CNN().to(DEVICE)
print(model)
```

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

- Trained CNN model over 10 epochs by iterating thorugh batches of images from the DataLoader.The model predicts loss using the chosen loss function, backpropogates the error to compute gradients and updates models weights using the optimizer, I got a test accuracy of 100% and it successfully predicted the test image

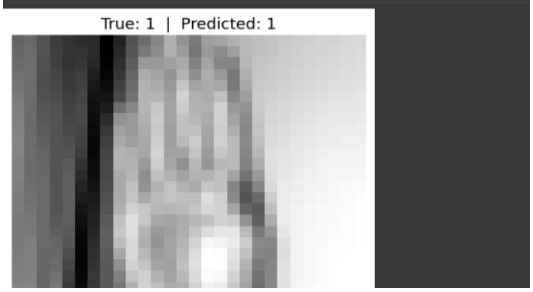
```python
train_losses = []

for epoch in range(1, EPOCHS + 1):
    model.train()
    running_loss = 0.0
    for imgs, lbls in train_loader:
        imgs, lbls = imgs.to(DEVICE), lbls.to(DEVICE)
        optimizer.zero_grad()
        outputs = model(imgs)
        loss = criterion(outputs, lbls)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    avg_loss = running_loss / len(train_loader)
    train_losses.append(avg_loss)
    print(f"Epoch {epoch}/{EPOCHS} — Loss: {avg_loss:.4f}")
```

```python
model.eval()
correct, total = 0, 0
with torch.no_grad():
    for imgs, lbls in test_loader:
        imgs, lbls = imgs.to(DEVICE), lbls.to(DEVICE)
        outputs = model(imgs)
        _, preds = torch.max(outputs, 1)
        total += lbls.size(0)
        correct += (preds == lbls).sum().item()

accuracy = 100 * correct / total
print(f"Test Accuracy: {accuracy:.2f}%")
```

```
Test Accuracy: 100.00%
```

```
Epoch 1/10 — Loss: 0.1906
Epoch 2/10 — Loss: 0.0003
Epoch 3/10 — Loss: 0.0002
Epoch 4/10 — Loss: 0.0001
Epoch 5/10 — Loss: 0.0001
Epoch 6/10 — Loss: 0.0001
Epoch 7/10 — Loss: 0.0001
Epoch 8/10 — Loss: 0.0001
Epoch 9/10 — Loss: 0.0000
Epoch 10/10 — Loss: 0.0000
```

```
imgs, lbls = next(iter(test_loader))
imgs, lbls = imgs.to(DEVICE), lbls.to(DEVICE)
model.eval()
with torch.no_grad():
    outputs = model(imgs)
_, preds = torch.max(outputs, 1)

i = 5
plt.imshow(imgs[i].cpu().squeeze(), cmap="gray")
plt.title(f"True: {lbls[i].item()}  |  Predicted: {preds[i].item()}")
plt.axis("off")
plt.show()
```



- Saved trained models weights and downloaded it

```
torch.save(model.state_dict(), "sign_language_cnn.pth")
files.download("sign_language_cnn.pth")
```

3: Live Hand Gesture Recognition Using CNN

i. Load the trained model (sign_language_cnn.pth)
ii. Defined the CNN model
iii. Opened a webcam feed using OpenCV to capture live video
iv. Initialize a hand detector using cvzone.HandTrackingModule to locate hands
v. Compute the center and crop size with margin to capture the entire hand
vi. Cropped the hand region from the orginal image
vii. Resize the cropped image to 28*28 pixels with padding
viii. Converted the resized image to grayscale, this made the image similar to the dataset

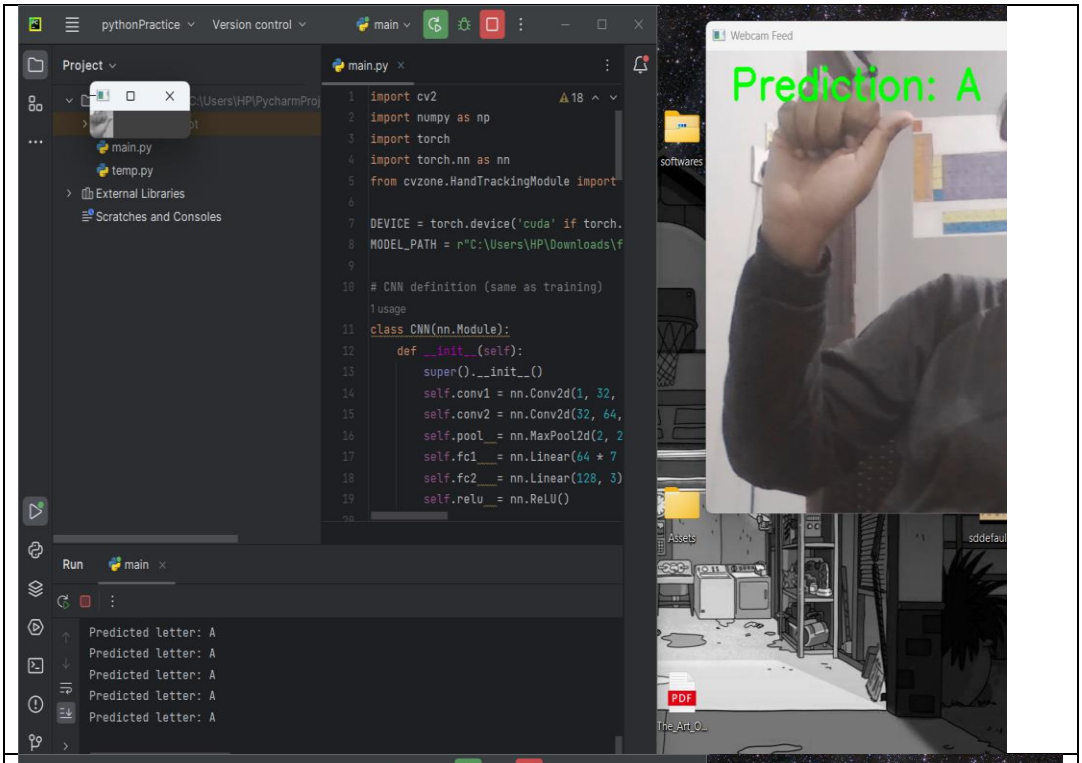
```python
import cv2
import numpy as np
import torch
import torch.nn as nn
from cvzone.HandTrackingModule import HandDetector
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
MODEL_PATH = r"C:\Users\HP\Downloads\filtered dataset\sign_language_cnn.pth"  # pa
1 usage
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 3)  # 3 classes
        self.relu = nn.ReLU()
    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
        x = self.relu(self.conv2(x))
        x = self.pool(x)
        x = x.view(-1, 64 * 7 * 7)
        x = self.relu(self.fc1(x))
        return self.fc2(x)
model = CNN().to(DEVICE)
model.load_state_dict(torch.load(MODEL_PATH, map_location=DEVICE))
model.eval()
class_map = {0: 'A', 1: 'B', 2: 'C'}
1 usage
def resize_with_padding(img, size=28):
```

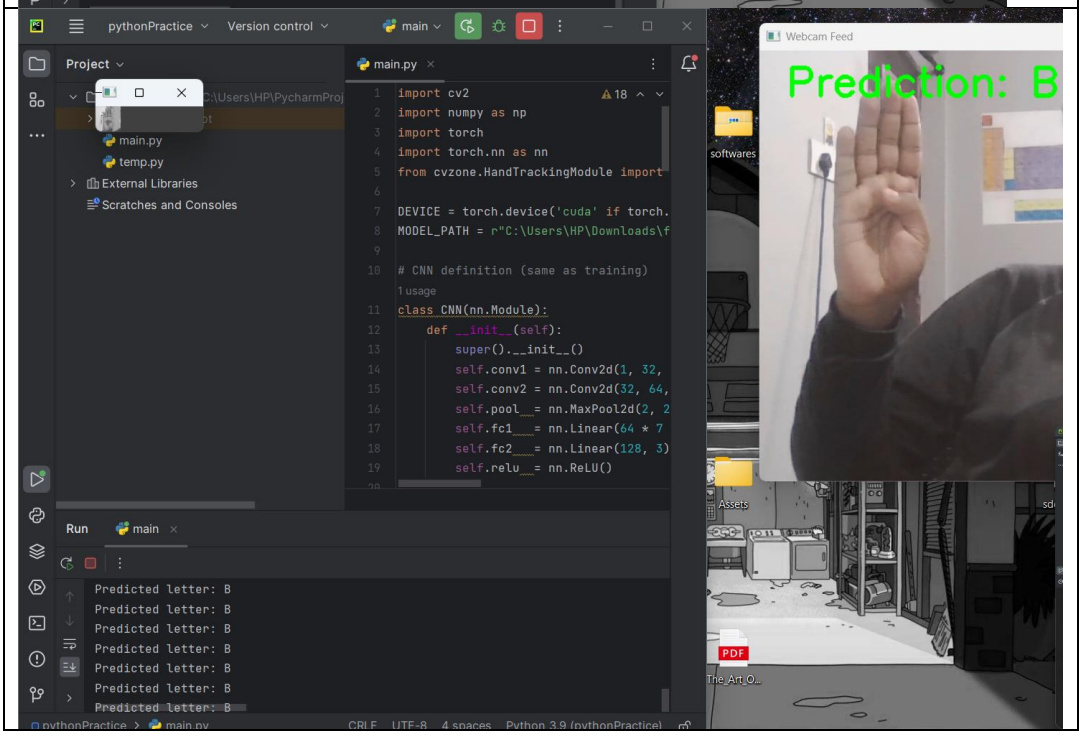
```python
def resize_with_padding(img, size=28):
    h, w = img.shape[:2]
    scale = size / max(h, w)
    new_w, new_h = int(w * scale), int(h * scale)
    resized = cv2.resize(img, (new_w, new_h))
    canvas = np.zeros((size, size, 3), dtype=np.uint8)
    x_offset = (size - new_w) // 2
    y_offset = (size - new_h) // 2
    canvas[y_offset:y_offset + new_h, x_offset:x_offset + new_w] = resized
    return canvas
cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=1)
while True:
    success, img = cap.read()
    if not success:
        break
    hands, _ = detector.findHands(img, draw=False)
    predicted_letter = ''  # default
    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']
        cx = x + w // 2
        cy = y + h // 2
        margin = 20
        crop_size = max(w, h) + margin
        x1 = max(0, cx - crop_size // 2)
        y1 = max(0, cy - crop_size // 2)
        x2 = min(img.shape[1], cx + crop_size // 2)
        y2 = min(img.shape[0], cy + crop_size // 2)
        hand_img = img[y1:y2, x1:x2]
        if hand_img.size != 0:
```

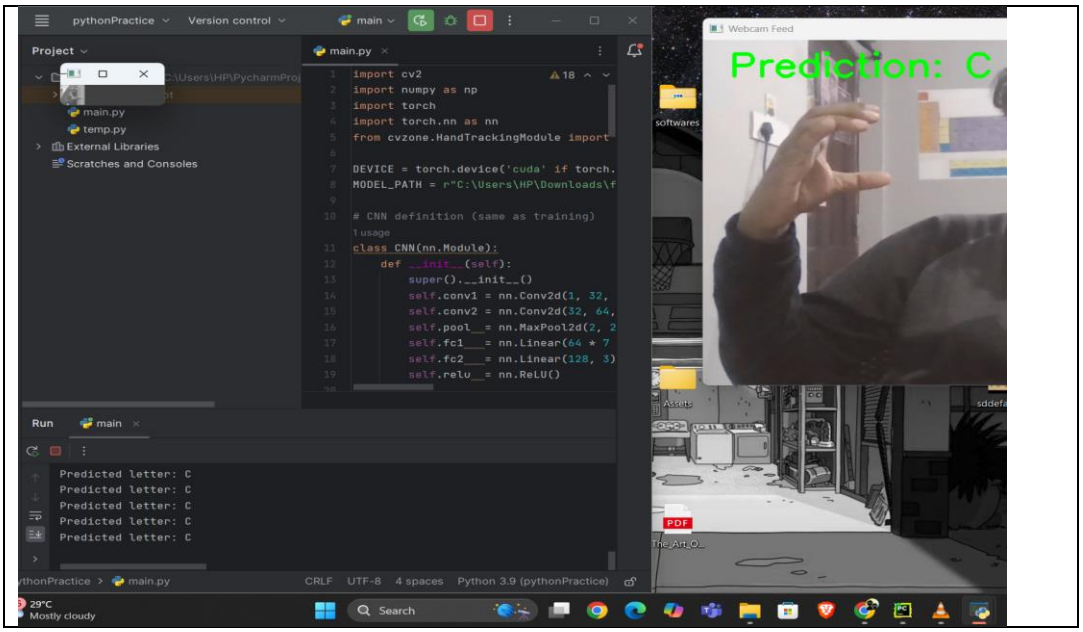
```python
        hand_img = img[y1:y2, x1:x2]
        if hand_img.size != 0:
            hand_img_28 = resize_with_padding(hand_img, 28)
            gray_scale = cv2.cvtColor(hand_img_28, cv2.COLOR_BGR2GRAY)
            normalized = gray_scale / 255.0
            standardized = (normalized - 0.5) / 0.5
            hand_tensor = torch.tensor(standardized, dtype=torch.float32).unsqueeze(0).unsqueeze(0).to(DEVICE)
            with torch.no_grad():
                outputs = model(hand_tensor)
                _, predicted = torch.max(outputs, 1)
                predicted_letter = class_map.get(predicted.item(), '?')
                print("Predicted letter:", predicted_letter)
            # Show hand crop
            cv2.imshow("Hand 28x28", gray_scale)
    if predicted_letter:
        cv2.putText(img, f"Prediction: {predicted_letter}", (30, 50),
                    cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0, 255, 0), 3)
    cv2.imshow("Webcam Feed", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

4: Results

- The model performed well on test images and In the live testing with the webcam input the system was able to detect and predict hand signs


pythonPractice
Version control
main
Project
main.py
temp.py
External Libraries
Scratches and Consoles
import cv2
import numpy as np
import torch
import torch.nn as nn
from cvzone.HandTrackingModule import
DEVICE = torch.device('cuda' if torch.
MODEL_PATH = r"C:\Users\HP\Downloads\f
# CNN definition (same as training)
1 usage
class CNN(nn.Module):
def __init__(self):
super().__init__()
self.conv1 = nn.Conv2d(1, 32,
self.conv2 = nn.Conv2d(32, 64,
self.pool = nn.MaxPool2d(2, 2
self.fc1 = nn.Linear(64 * 7
self.fc2 = nn.Linear(128, 3)
self.relu = nn.ReLU()
Run
main
Predicted letter: A
Predicted letter: A
Predicted letter: A
Predicted letter: A
Predicted letter: A
Webcam Feed
Prediction: A
softwares
Assets
The_Art_O...


pythonPractice
Version control
main
Project
main.py
temp.py
External Libraries
Scratches and Consoles
import cv2
import numpy as np
import torch
import torch.nn as nn
from cvzone.HandTrackingModule import
DEVICE = torch.device('cuda' if torch.
MODEL_PATH = r"C:\Users\HP\Downloads\f
# CNN definition (same as training)
1 usage
class CNN(nn.Module):
def __init__(self):
super().__init__()
self.conv1 = nn.Conv2d(1, 32,
self.conv2 = nn.Conv2d(32, 64,
self.pool = nn.MaxPool2d(2, 2
self.fc1 = nn.Linear(64 * 7
self.fc2 = nn.Linear(128, 3)
self.relu = nn.ReLU()
Run
main
Predicted letter: B
Predicted letter: B
Predicted letter: B
Predicted letter: B
Predicted letter: B
Predicted letter: B
Predicted letter: B
Webcam Feed
Prediction: B
softwares
Assets
The_Art_O...



Conclusion

- The CNN successfully learned to recognize the selected hand gestures and performed well on live inputsUsing the .pth file allowed me to resuse the trained model for real- time prediction
- This demonstrates a practical Sign Language Translator system that could be extended more letters and gestures