Final report

Edward Dang
CS 4342

Tech Stack: Python
Dataset: German Traffic Sign Recognition Benchmark
Tools: Google Colab

Task Description: This project focus is solving German Traffic Signs using multi-class classification with a Kaggle Dataset. This project classifies the traffic signs into 43 different categories. It involves data pre-processing, training the model, implementing a custom Convolution Neural Network, pre-trained ResNet50 and VGG16, testing the models, and evaluating the loss and accuracy.

Dataset: The GTSRB dataset (German Traffic Sign Recognition Benchmark) contains over 50,000 images of German traffic signs and can be classified into 43 classes. Each image batch has a variety in resolution, lighting, and distortions.

Preprocessing Steps:

1. Resizing: All of the images were resized to 32x 32 so we can standardize the inputs for the models
2. Data Manipulation:
    a. Horizontal Flips: This flipped the images horizontally to have more variations
    b. Rotations: This rotated the images slightly to have different orientations and perspectives as not all the pictures were taken perfectly
    c. Color Variation: Changing the brightness, saturation, contrast, and hue because not all pictures are 100% perfect.
    d. Scaling & Translation: Scaling up and down the pictures along with translating it to different locations (small amount) as not all signs will be in a center nor correct sizing
    e. Normalization: The pixel values were normalized to [-1,1]

Implementation of models

- **Custom CNN Model:** Explanation of Code:
- Data Preprocessing:
    - Transformation of inputs so that we can have variation of distorted images, for better training accuracy
    - Load Training dataset with the project structure ./archive/Train/0…/00000.png
        - Helper function to access data file TrainDataset()
    - Load Testing dataset with the project structure ./archive/Test/00000.png
        - Helper function to access data file TestDataset()
    - Load Testing.csv so that we can assure the png file is classified in the right class
- Data Visualization
    - Testing a visualization of a batch of images and their classes

- **CNN Architecture:**
    1. Convolutional Layers
        a. Layer 1: 32 Filters | 3x3 Kernel | ReLU activations | 2x2 MaxPooling
        b. Layer 2: 64 Filters | 3x3 Kernel | ReLU activations | 2x2 MaxPooling
    2. Fully Connected Layers
        a. FC1: 64x8x8 (Flattening) → Maps out 512 neurons → ReLu → 0.3 Dropout
        b. FC2: 512 neurons → 43 classes
    3. Output:
        a. Log softmax for classification probabilities
- Train Model:
    - The outer loop iterates over the number of specified epochs. The inner loop iterates over batches of data provided by the train_loader. Then we zero the gradients to prevent them from accumulating over multiple passes. The forward pass computes the model's predictions, then backwards propagation computes the gradient of the loss. At the end of each epoch, the average loss and accuracy are calculated, stored, and returned
- Evaluate Model
    - We first disable gradient calculations to save memory and computation time. The loop iterates over batches of test data provided by the test_loader. The model processes the input images to produce the predictions. The loss is computed using the predictions and actual labels. The batch loss is added to the total test loss. Total is incremented by the number of samples in the batch and counts how many predictions match the actual labels. Then we calculate accuracy by, correct * 100 / total, and loss by, test_loss /= len(test_loader)
- Visualize Predictions
    - We gather the training losses and accuracies (the training method keeps track of each component throughout each epoch and returns them) through the training method and using plt.plot to visualize the graph.

Pre-trained ResNet50 and VGG16 and Fine-Tuning: ResNet50:

- Resnet50 is a CNN that is 50 layers deep and uses residual connections. It has pretrained data of millions of images using Image net. Its size is 224x224 (Matlab reference).
- The changes I made to fine tune the model was replacing the final fully connected layer with a new layer to classify my specific dataset's (43 classes). Implemented Adam optimizer to improve training speed (Built-in Reference). Implemented cross-entropy loss function because its a multi class classification.

VGG16:

- VGG-16 is a simple CNN architecture that is 16 layers deep, with convolutional layers. It has a pretrained network that can classify images into 1000 different categories (MatLab Reference).
- The changes I made to fine tune the model was replacing the final fully connected layer with a new layer to classify my specific dataset's (43 classes). Implemented Adam

optimizer to improve training speed (Built-in Reference). Additionally for the Adam optimize I used different learning rates 0.0001 for pretrained feature layers and 0.001 for classifying layer. Implemented cross-entropy loss function because its a multi class classification.

Transferred Learning

- For both models I used their pretrained weights. I then trained the model with the GTSRB dataset with their preset layers, and then made the last layer specific to my classifications.

Hyperparameters Selection: Batch Size: 32

- The batch size was chose to be 32 because its the most optimal for GPU Processing and training speed

Learning Rate: 0.001

- Standard learning rate with the most efficient time to accuracy

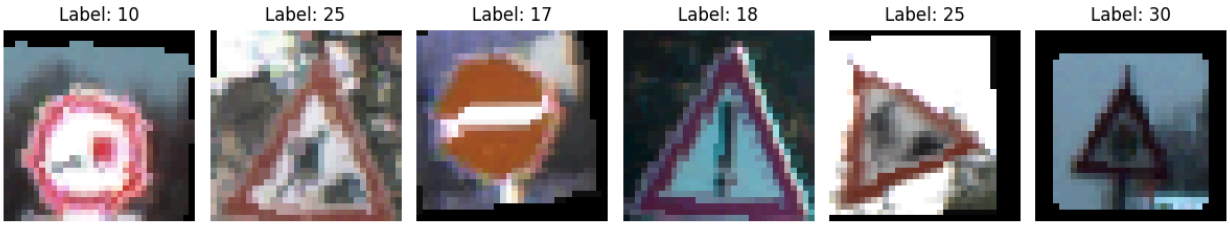
Weight Decay: 1e^-5

- Standardize the model and prevents overfitting

Dropout rate: 0.3

- Prevent overfitting and handles outliers

Results and Analysis::



Training and Testing outputs:

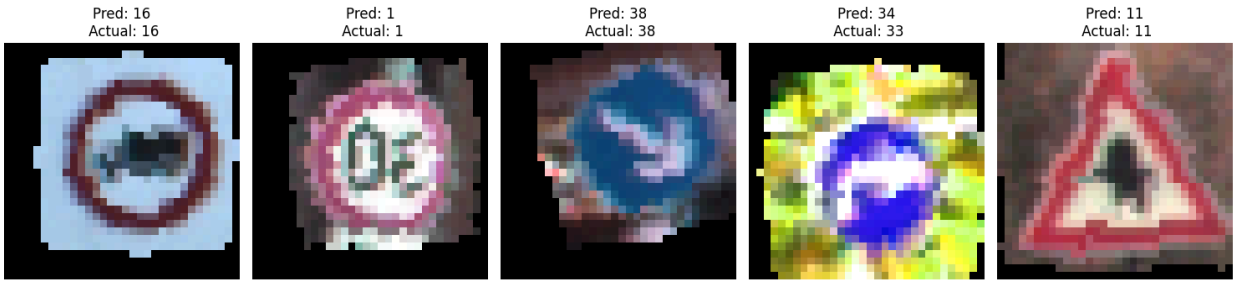
- Custom CNN:
  - Training accuracy peaks around 70% - 80% after 10 Epochs. Testing accuracy 77.88%. Average loss is around 0.6 - 0.8
- ResNet50:
  - Training accuracy peak around 88% - 92% after 10 Epochs. Testing accuracy is 88.18%. Average loss is around 0.2 - 0.3.
- VGG16
  - Training accuracy peak around 90% - 94% after 10 Epochs. Testing accuracy is 91.88%. Average loss is around 0.2 - 0.3.

```
Training Custom CNN...
Epoch 1/10, Loss: 2.0867, Accuracy: 36.01%
Epoch 2/10, Loss: 1.4004, Accuracy: 53.59%
Epoch 3/10, Loss: 1.1388, Accuracy: 61.83%
Epoch 4/10, Loss: 0.9916, Accuracy: 66.24%
Epoch 5/10, Loss: 0.8842, Accuracy: 69.76%
Epoch 6/10, Loss: 0.8045, Accuracy: 72.49%
Epoch 7/10, Loss: 0.7475, Accuracy: 74.43%
Epoch 8/10, Loss: 0.6863, Accuracy: 76.45%
Epoch 9/10, Loss: 0.6447, Accuracy: 78.12%
Epoch 10/10, Loss: 0.6036, Accuracy: 79.43%
Testing Custom CNN...
Test Loss: 0.6738, Test Accuracy: 77.88%
```

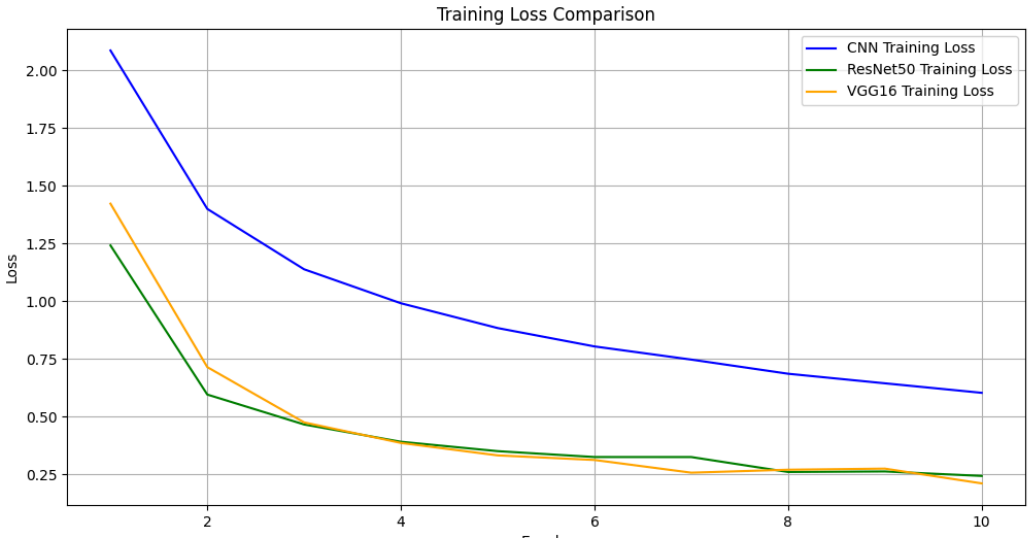
```
Training ResNet50...
Epoch 1/10, Loss: 1.2421, Accuracy: 62.01%
Epoch 2/10, Loss: 0.5961, Accuracy: 80.94%
Epoch 3/10, Loss: 0.4665, Accuracy: 85.15%
Epoch 4/10, Loss: 0.3916, Accuracy: 87.40%
Epoch 5/10, Loss: 0.3508, Accuracy: 88.85%
Epoch 6/10, Loss: 0.3254, Accuracy: 89.60%
Epoch 7/10, Loss: 0.3253, Accuracy: 89.63%
Epoch 8/10, Loss: 0.2606, Accuracy: 91.54%
Epoch 9/10, Loss: 0.2629, Accuracy: 91.57%
Epoch 10/10, Loss: 0.2435, Accuracy: 92.22%
Testing ResNet50...
Test Loss: 0.4075, Test Accuracy: 88.18%
```

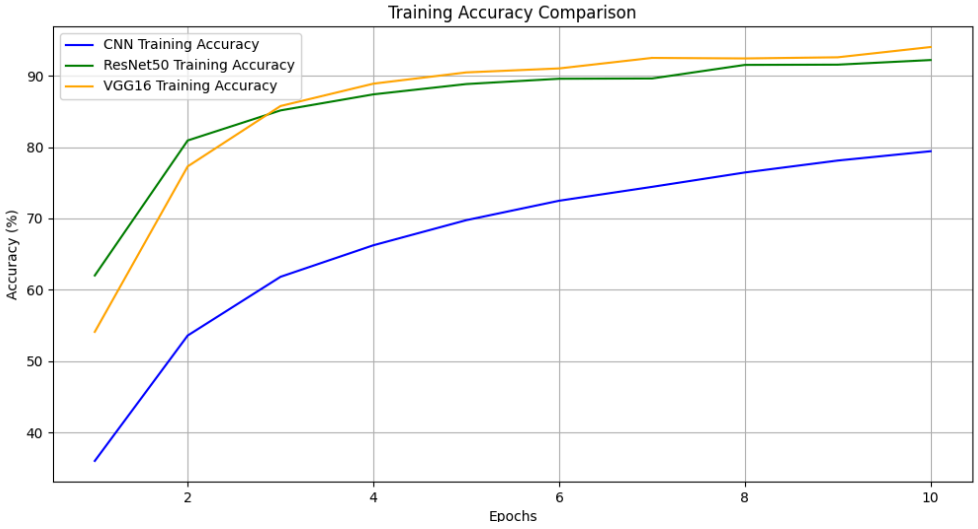
```
Training VGG16...
Epoch 1/10, Loss: 1.4231, Accuracy: 54.10%
Epoch 2/10, Loss: 0.7149, Accuracy: 77.30%
Epoch 3/10, Loss: 0.4754, Accuracy: 85.77%
Epoch 4/10, Loss: 0.3864, Accuracy: 88.90%
Epoch 5/10, Loss: 0.3325, Accuracy: 90.48%
Epoch 6/10, Loss: 0.3126, Accuracy: 91.04%
Epoch 7/10, Loss: 0.2576, Accuracy: 92.52%
Epoch 8/10, Loss: 0.2704, Accuracy: 92.44%
Epoch 9/10, Loss: 0.2751, Accuracy: 92.60%
Epoch 10/10, Loss: 0.2112, Accuracy: 94.04%
Testing VGG16...
Test Loss: 0.3672, Test Accuracy: 91.88%
```

CNN Prediction vs Actual:



Graphs





Comparison:

- Custom CNN: Very efficient with processing, but lacks a lot of depth (layers) and accuracy, compared to ResNet50 and VGG16
- ResNet50: Best overall, has a good ratio for training time to accuracy compared to Custom CNN and VGG16. Efficient on timing and has strong performance due to its depth in layers and residual connections
- VGG16: Best accuracy, but very slow training. Only a little higher accuracy score than ResNet50.