PROJECT 3: CIFAR-10 Dataset Classification

CSCI-6450: Machine Learning

Done by Kundan and Chirag

Contribution

Slides: Sai Kundan Suddapalli

Code: Sai Kundan Suddapalli

Documentation/Report: Chirag Simha

About CIFAR-10 Dataset

The CIFAR-10 dataset is a collection of 60,000 32x32 color images, categorized into 10 classes, with 6,000 images per class. It's widely used for benchmarking machine learning models in image classification tasks. This dataset's balanced classes and manageable size make it a popular choice for both learning and research.

Data Preprocessing & Argumentation

```python
# Data Preprocessing
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),  # Randomly flip the image horizontally
    transforms.RandomRotation(10),      #he image by 10 degrees
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),  # Randomly change brightness, contrast, saturation, and hue
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
dataset = datasets.CIFAR10(root=data_dir, train=True, download=True, transform=transform_train)
train_size = int(0.8 * len(dataset))  # 80
val_size = len(dataset) - train_size   # 20


# Split the dataset into training and validation sets
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])
test_dataset = datasets.CIFAR10(root=data_dir,train=False, download=True,transform =transform_test)
```

The CIFAR-10 dataset was split into 80% training and 20% validation sets. Images were normalized for preprocessing.

To improve model robustness and generalization, we used image augmentation. This included random horizontal flips (10% probability), rotations, and color jittering (adjusting saturation, brightness, and contrast).

Model Architecture 1: Simple CNN

```
class SimpleCNN(nn.Module):
    def __init__(self, num_classes):
        super(SimpleCNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),  # Batch normalization after the first conv layer
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),  # Batch normalization after the second conv layer
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),  # Batch normalization after the third conv layer
            nn.ReLU(),
            nn.AdaptiveAvgPool2d((1, 1)),
            nn.Dropout(0.1)
        )
        self.fc = nn.Linear(256, num_classes)

    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x
```

Model Architecture 2: STN + Simple CNN

This model improves image classification by using a Spatial Transformer Network (STN) to align images before a Simple CNN classifies them.

Components:

- **STN:** Extracts features; predicts alignment; applies transformation.
- **Simple CNN:** Extracts features from the aligned image and classifies.

```
class SpatialTransformer(nn.Module):
    def __init__(self, input_channels=3):
        super(SpatialTransformer, self).__init__()
        # Localization network
        self.localization = nn.Sequential(
            nn.Conv2d(input_channels, 32, kernel_size=7, stride=1, padding=3),
            nn.ReLU(True),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
            nn.ReLU(True),
            nn.MaxPool2d(2, 2)
        )
        self.fc_loc = nn.Sequential(
            nn.Linear(64 * 8 * 8, 128),  # Adjust based on input size
            nn.ReLU(True),
            nn.Linear(128, 6)
        )
        # Initialize the weights/bias with identity transformation
        self.fc_loc[2].weight.data.zero_()
        self.fc_loc[2].bias.data.copy_(torch.tensor([1, 0, 0, 0, 1, 0], dtype=torch.float))

    def forward(self, x):
        # Compute the transformation parameters
        xs = self.localization(x)
        xs = xs.view(xs.size(0), -1)
        theta = self.fc_loc(xs)
        theta = theta.view(-1, 2, 3)
        # Apply the transformation to the input feature map
        grid = F.affine_grid(theta, x.size(), align_corners=False)
        x = F.grid_sample(x, grid, align_corners=False)
        return x

class SimpleCNN(nn.Module):
    def __init__(self, num_classes):
        super(SimpleCNN, self).__init__()
        self.stn = SpatialTransformer()  # Add the STN module
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
49 fewer lines; before #1  2 seconds ago
```

Overview

1. Input image to STN.
2. STN aligns the image.
3. Aligned image to CNN.
4. CNN classifies the image.

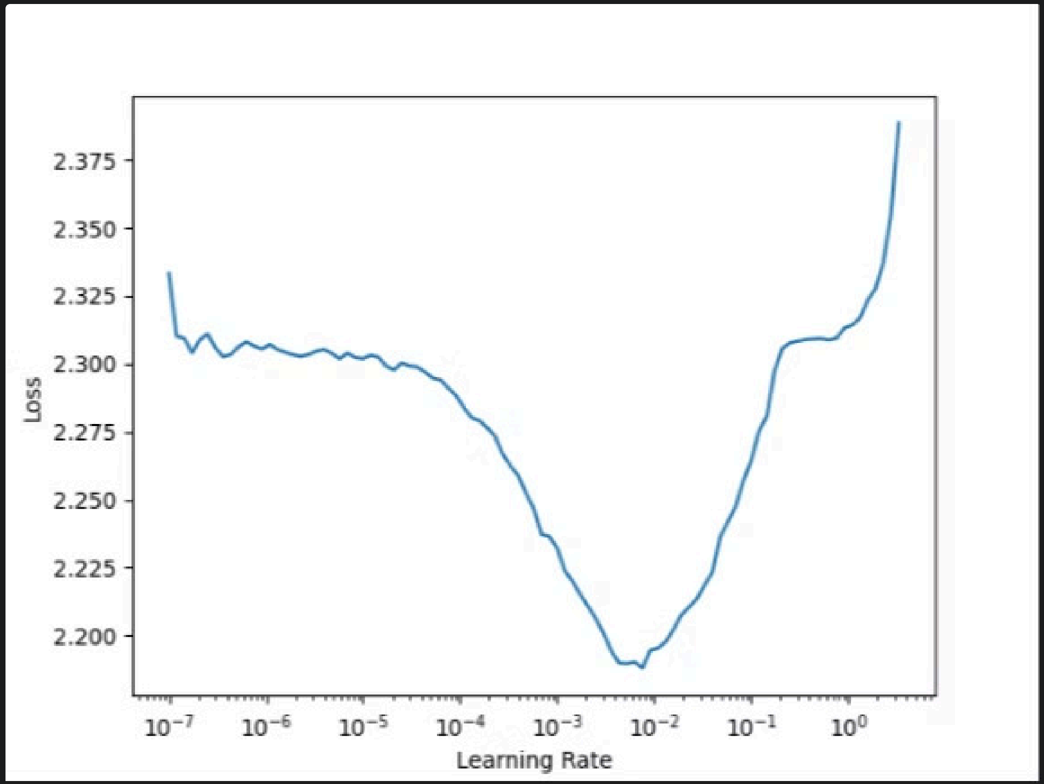
Optimizer and Cross-Entropy Loss

Both Adam and SGD optimizers were used.

Adam yielded better results

Efficient Learning Rate

We utilized the FastAI library, which automatically determines an optimal learning rate for the model and cross-entropy function. The resulting learning rate was 1e-3.



Training

we used Early stop patience 15

Batch_size = 128 , epochs = 2000 , learning rate =1e-3 /0.001 ,patience = 50   for combo

Batch_size = 64, epochs = 1000 , learning rate =1e-2 /0.01 , patience = 15   for Simple CNN

```python
import torch
from tvt.validate import validate_model  # Ensure this import exists

def train_model(model, optimizer, criterion, train_loader, val_loader, epochs, patience, device):
    best_val_loss = float('inf')
    patience_counter = 0
    history = {'train_loss': [], 'train_acc': [], 'val_loss': [], 'val_acc': []}

    for epoch in range(1, epochs + 1):
        # Training Phase
        model.train()
        total_loss, correct, total = 0.0, 0, 0
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            total_loss += loss.item()
            _, preds = torch.max(outputs, 1)
            correct += (preds == labels).sum().item()
            total += labels.size(0)

        train_loss = total_loss / len(train_loader)
        train_acc = correct / total

        # Validation Phase
        val_loss, val_acc = validate_model(model, val_loader, criterion, device)  # Validate here

        # Save metrics
        history['train_loss'].append(train_loss)
        history['train_acc'].append(train_acc)
        history['val_loss'].append(val_loss)
        history['val_acc'].append(val_acc)

        # Early Stopping
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            patience_counter = 0
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print(f"Early stopping at epoch {epoch}")
```

Pretext Task

To enhance model accuracy, we incorporated an image rotation pretext task. The model was trained to classify images rotated by 0, 90, 180, and 270 degrees, forcing it to learn rotation-invariant features.

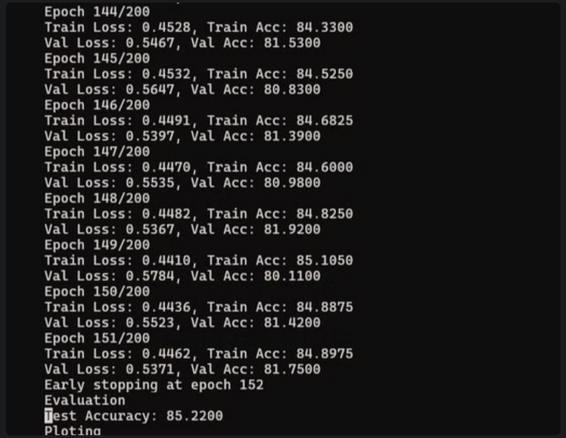
epocs of 200

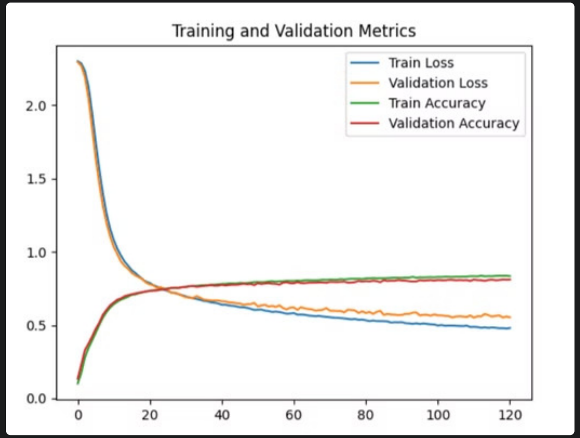
```
Val Loss: 0.2379, Val Acc: 91.4019
Epoch 187/200
Train Loss: 0.2977, Train Acc: 88.8537
Val Loss: 0.2398, Val Acc: 91.3350
Epoch 188/200
Train Loss: 0.2958, Train Acc: 88.9450
Val Loss: 0.2405, Val Acc: 91.3637
Epoch 189/200
Train Loss: 0.2974, Train Acc: 88.8406
Val Loss: 0.2380, Val Acc: 91.4169
Epoch 190/200
Train Loss: 0.2943, Train Acc: 88.9956
Val Loss: 0.2411, Val Acc: 91.3713
Epoch 191/200
Train Loss: 0.2964, Train Acc: 88.8706
Val Loss: 0.2478, Val Acc: 90.9462
Epoch 192/200
Train Loss: 0.2940, Train Acc: 88.9819
Val Loss: 0.2449, Val Acc: 91.1150
Epoch 193/200
Train Loss: 0.2958, Train Acc: 88.8994
Val Loss: 0.2462, Val Acc: 91.0787
Epoch 194/200
Train Loss: 0.2932, Train Acc: 88.9162
Val Loss: 0.2398, Val Acc: 91.2987
Epoch 195/200
Train Loss: 0.2955, Train Acc: 88.9237
Val Loss: 0.2442, Val Acc: 91.1475
Epoch 196/200
Train Loss: 0.2943, Train Acc: 88.9213
Val Loss: 0.2381, Val Acc: 91.4037
Epoch 197/200
Train Loss: 0.2932, Train Acc: 88.9781
Val Loss: 0.2483, Val Acc: 90.8800
Epoch 198/200
Train Loss: 0.2943, Train Acc: 89.0037
Val Loss: 0.2318, Val Acc: 91.6119
Epoch 199/200
Train Loss: 0.2930, Train Acc: 88.9531
Val Loss: 0.2496, Val Acc: 90.8519
Epoch 200/200
Train Loss: 0.2918, Train Acc: 88.9894
Val Loss: 0.2347, Val Acc: 91.4437
Output layer is changing 4 to 10
Training
```

Results with Simple CNN

85%

Test Accuracy

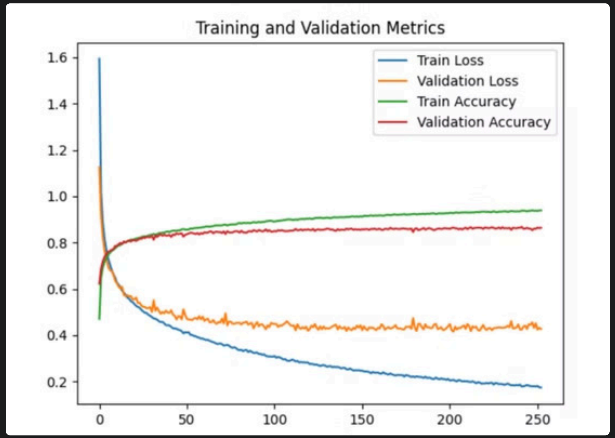


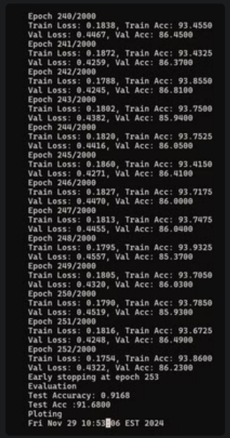


Results with Combo architecture

91%

Test Accuracy





```
Epoch 240/2000
Train Loss: 0.1838, Train Acc: 93.4550
Val Loss: 0.4467, Val Acc: 86.4500
Epoch 241/2000
Train Loss: 0.1872, Train Acc: 93.4325
Val Loss: 0.4259, Val Acc: 86.3700
Epoch 242/2000
Train Loss: 0.1788, Train Acc: 93.8550
Val Loss: 0.4245, Val Acc: 86.8100
Epoch 243/2000
Train Loss: 0.1802, Train Acc: 93.7500
Val Loss: 0.4382, Val Acc: 85.9400
Epoch 244/2000
Train Loss: 0.1820, Train Acc: 93.7525
Val Loss: 0.4416, Val Acc: 86.0500
Epoch 245/2000
Train Loss: 0.1860, Train Acc: 93.4150
Val Loss: 0.4271, Val Acc: 86.4100
Epoch 246/2000
Train Loss: 0.1827, Train Acc: 93.7175
Val Loss: 0.4470, Val Acc: 86.0000
Epoch 247/2000
Train Loss: 0.1813, Train Acc: 93.7475
Val Loss: 0.4455, Val Acc: 86.0400
Epoch 248/2000
Train Loss: 0.1795, Train Acc: 93.9325
Val Loss: 0.4557, Val Acc: 85.3700
Epoch 249/2000
Train Loss: 0.1805, Train Acc: 93.7050
Val Loss: 0.4320, Val Acc: 86.0300
Epoch 250/2000
Train Loss: 0.1790, Train Acc: 93.7850
Val Loss: 0.4519, Val Acc: 85.9300
Epoch 251/2000
Train Loss: 0.1816, Train Acc: 93.6725
Val Loss: 0.4248, Val Acc: 86.4900
Epoch 252/2000
Train Loss: 0.1754, Train Acc: 93.8600
Val Loss: 0.4322, Val Acc: 86.2300
Early stopping at epoch 253
Evaluation
Test Accuracy: 0.9168
Test Acc :91.6800
Ploting
Fri Nov 29 10:53:06 EST 2024
```

Q&A

We welcome your questions and look forward to discussing this further.