Appendix D: Training a CNN on the CIFAR-10 dataset

Contents

```
import numpy as np
import torch
from PIL import Image
from torch import nn, optim
from torchvision import datasets, transforms, utils
from torchsummary import summary
import matplotlib.pyplot as plt
plt.style.use('ggplot')
plt.rcParams.update({'font.size': 16, 'axes.labelweight': 'bold', 'axes.grid':
import sys, os
sys.path.append(os.path.join(os.path.abspath(".."), "code"))
from plotting import *
from set_seed import *
DATA_DIR = DATA_DIR = os.path.join(os.path.abspath(".."), "data/")
```

```
set_seed(1)
```

1. Introduction

In this notebook, we will develop a Convolutional Neural Network (CNN) using the CIFAR-10 dataset. This notebook covers the following key concepts:

- Developing a CNN on the CIFAR-10 training dataset, evaluating its performance on the validation set, and saving the trained model.
- Evaluating the saved model's performance on vertically flipped images from the validation set.
- Training a new model on randomly flipped training images to improve its robustness, followed by evaluating it on randomly flipped validation images.

In this notebook we'll work with the popular CIFAR-10 dataset.

2. CNN on the CIFAR-10 dataset

```
from torchvision import datasets, transforms

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

train_dataset = datasets.CIFAR10(root='./data', train=True, transform=transfor
valid_dataset = datasets.CIFAR10(root='./data', train=False, transform=transfo
```

```
Files already downloaded and verified
```

```
Files already downloaded and verified
```

```
IMAGE_SIZE = 32
BATCH_SIZE = 32
trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=BATCH_SIZE
validloader = torch.utils.data.DataLoader(valid_dataset, batch_size=BATCH_SIZE

# Plot samples
sample_batch = next(iter(trainloader))
plt.figure(figsize=(10, 8)); plt.axis("off"); plt.title("Sample Training Image
plt.imshow(np.transpose(utils.make_grid(sample_batch[0], padding=1, normalize=
```

```
device = torch.device('mps' if torch.backends.mps.is_available() else 'cpu')
```

```
class CIFAR10_CNN(nn.Module):
    def __init__(self):
        super(CIFAR10_CNN, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Flatten(),
            nn.Linear(128 * 4 * 4, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )

    def forward(self, x):
        return self.model(x)
```

```
# Instantiate the model
cifar10_model = CIFAR10_CNN().to(device)
print(cifar10_model)
```

```
CIFAR10_CNN(
  (model): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fa
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fa
    (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fa
    (9): Flatten(start_dim=1, end_dim=-1)
    (10): Linear(in_features=2048, out_features=512, bias=True)
    (11): ReLU()
    (12): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

```
summary(cifar10_model, (3, 32, 32));
```

```
===============================================================================
Layer (type:depth-idx)                  Output Shape              Param #
===============================================================================
├─Sequential: 1-1                       [-1, 10]                  --
|    └─Conv2d: 2-1                      [-1, 32, 32, 32]          896
|    └─ReLU: 2-2                        [-1, 32, 32, 32]          --
|    └─MaxPool2d: 2-3                   [-1, 32, 16, 16]          --
|    └─Conv2d: 2-4                      [-1, 64, 16, 16]          18,496
|    └─ReLU: 2-5                        [-1, 64, 16, 16]          --
|    └─MaxPool2d: 2-6                   [-1, 64, 8, 8]            --
|    └─Conv2d: 2-7                      [-1, 128, 8, 8]           73,856
|    └─ReLU: 2-8                        [-1, 128, 8, 8]           --
|    └─MaxPool2d: 2-9                   [-1, 128, 4, 4]           --
|    └─Flatten: 2-10                    [-1, 2048]                --
|    └─Linear: 2-11                     [-1, 512]                 1,049,088
|    └─ReLU: 2-12                       [-1, 512]                 --
|    └─Linear: 2-13                     [-1, 10]                  5,130
===============================================================================
Total params: 1,147,466
Trainable params: 1,147,466
Non-trainable params: 0
Total mult-adds (M): 12.52
===============================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 0.44
Params size (MB): 4.38
Estimated Total Size (MB): 4.83
===============================================================================
def trainer(model, criterion, optimizer, trainloader, validloader, epochs=5, v
    """Simple training wrapper for PyTorch network."""

    train_loss, valid_loss, valid_accuracy = [], [], []
    for epoch in range(epochs):  # for each epoch
        train_batch_loss = 0
        valid_batch_loss = 0
        valid_batch_acc = 0

        # Training
        model.train()
        for X, y in trainloader:
            X, y = X.to(device), y.to(device)
            optimizer.zero_grad()
            y_hat = model(X)
            loss = criterion(y_hat, y)
            loss.backward()
            optimizer.step()
            train_batch_loss += loss.item()
        train_loss.append(train_batch_loss / len(trainloader))

        # Validation
        model.eval()

        with torch.no_grad():  # this stops pytorch doing computational graph
            for X, y in validloader:
                X, y = X.to(device), y.to(device)
                y_hat = model(X)
                _, y_hat_labels = torch.softmax(y_hat, dim=1).topk(1, dim=1)
                loss = criterion(y_hat, y)
                valid_batch_loss += loss.item()
                valid_batch_acc += (y_hat_labels.squeeze() == y).type(torch.fl
        valid_loss.append(valid_batch_loss / len(validloader))
        valid_accuracy.append(valid_batch_acc / len(validloader))  # accuracy



        # Print progress
        if verbose:
            print(f"Epoch {epoch + 1}:",
                  f"Train Loss: {train_loss[-1]:.3f}.",
                  f"Valid Loss: {valid_loss[-1]:.3f}.",
                  f"Valid Accuracy: {valid_accuracy[-1]:.2f}.")

    results = {"train_loss": train_loss,
               "valid_loss": valid_loss,
               "valid_accuracy": valid_accuracy}
    return results
```

```
cifar10_model = CIFAR10_CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(cifar10_model.parameters(), lr=2e-3)
# results = trainer(cifar10_model, criterion, optimizer, trainloader, validloa
```

```
Epoch 1: Train Loss: 1.309. Valid Loss: 1.028. Valid Accuracy: 0.64.
Epoch 2: Train Loss: 0.911. Valid Loss: 0.855. Valid Accuracy: 0.70.
Epoch 3: Train Loss: 0.751. Valid Loss: 0.832. Valid Accuracy: 0.71.
Epoch 4: Train Loss: 0.642. Valid Loss: 0.966. Valid Accuracy: 0.68.
Epoch 5: Train Loss: 0.546. Valid Loss: 0.878. Valid Accuracy: 0.73.
Epoch 6: Train Loss: 0.466. Valid Loss: 0.913. Valid Accuracy: 0.72.
Epoch 7: Train Loss: 0.401. Valid Loss: 0.995. Valid Accuracy: 0.72.
Epoch 8: Train Loss: 0.347. Valid Loss: 1.045. Valid Accuracy: 0.71.
Epoch 9: Train Loss: 0.310. Valid Loss: 1.191. Valid Accuracy: 0.72.
Epoch 10: Train Loss: 0.263. Valid Loss: 1.300. Valid Accuracy: 0.71.
Epoch 11: Train Loss: 0.246. Valid Loss: 1.368. Valid Accuracy: 0.72.
Epoch 12: Train Loss: 0.235. Valid Loss: 1.501. Valid Accuracy: 0.70.
Epoch 13: Train Loss: 0.227. Valid Loss: 1.467. Valid Accuracy: 0.71.
Epoch 14: Train Loss: 0.200. Valid Loss: 1.710. Valid Accuracy: 0.70.
Epoch 15: Train Loss: 0.213. Valid Loss: 1.818. Valid Accuracy: 0.71.
Epoch 16: Train Loss: 0.193. Valid Loss: 1.770. Valid Accuracy: 0.71.
Epoch 17: Train Loss: 0.190. Valid Loss: 1.863. Valid Accuracy: 0.71.
Epoch 18: Train Loss: 0.194. Valid Loss: 1.898. Valid Accuracy: 0.70.
Epoch 19: Train Loss: 0.173. Valid Loss: 2.081. Valid Accuracy: 0.71.
Epoch 20: Train Loss: 0.186. Valid Loss: 2.144. Valid Accuracy: 0.71.
```

```python
# Assuming 'cifar10_model' is your trained model and 'validloader' is your Dat
cifar10_model.eval()
cifar10_model.to('cpu')

# CIFAR-10 class labels
class_labels = [
    "airplane", "automobile", "bird", "cat", "deer",
    "dog", "frog", "horse", "ship", "truck"
]

# Function to display images with predictions and true labels
def show_predictions(model, dataloader, class_labels, num_images=5):
    model.eval()  # Set model to evaluation mode
    with torch.no_grad():
        # Get a batch of validation data
        data_iter = iter(dataloader)
        images, true_labels = next(data_iter)

        # Get predictions
        outputs = model(images)
        _, predicted_labels = torch.max(outputs, 1)

        # Show images with predictions and true labels
        fig, axes = plt.subplots(1, num_images, figsize=(15, 3))
        for i in range(num_images):
            ax = axes[i]
            img = images[i].permute(1, 2, 0)  # Convert (C, H, W) to (H, W, C)
            ax.imshow((img * 0.5 + 0.5).numpy())  # Denormalize image
            ax.axis("off")
            ax.set_title(
                f"True: {class_labels[true_labels[i]]}\nPred: {class_labels[pr
                fontsize=10
            )
        plt.tight_layout()
        plt.show()
```

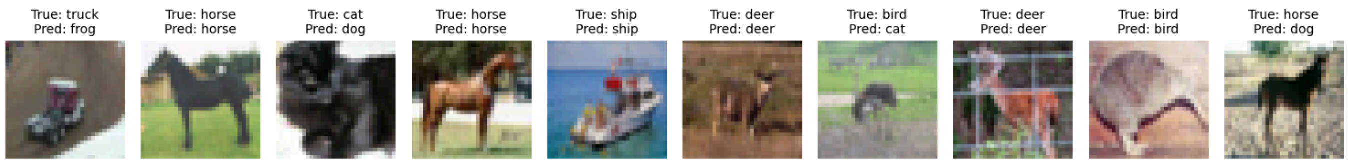
```python
# Save model
PATH = "../models/CIFAR10-cnn.pt"
# torch.save(cifar10_model.state_dict(), PATH)

# Load model
cifar10_model.load_state_dict(torch.load(PATH))
```

```
/var/folders/b3/g26r0dcx4b35vf3nk31216hc0000gr/T/ipykernel_74927/744367782.py:
  cifar10_model.load_state_dict(torch.load(PATH))
```

```
<All keys matched successfully>
```

Let's examine some predictions made by the model.

```
# Call the function to display predictions
show_predictions(cifar10_model, validloader, class_labels, num_images=10)
```



What's the validation accuracy?

```
# Evaluate model performance
def evaluate_model(model, dataloader, class_labels):
    model.eval()
    model.to('cpu')  # Ensure the model is on CPU
    correct = 0
    total = 0
    all_predictions = []
    all_labels = []

    with torch.no_grad():
        for images, labels in dataloader:
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            all_predictions.append(predicted)
            all_labels.append(labels)

    accuracy = 100 * correct / total
    print(f"Validation Accuracy on Flipped Images: {accuracy:.2f}%")
    return all_predictions, all_labels
```

```
# Evaluate and visualize
all_predictions, all_labels = evaluate_model(cifar10_model, validloader, class
```

```
Validation Accuracy on Flipped Images: 70.82%
```

The accuracy and the predictions look reasonable.

2. How robust is the model?

Let's vertically flip the validation images and evaluate the model's performance on this modified validation set.

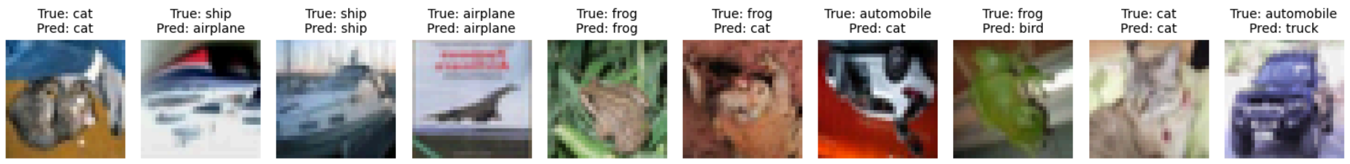
```
# Validation transforms (vertical flip)
valid_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),  # Random horizontal flip
    transforms.RandomVerticalFlip(p=0.5),    # Random vertical flip
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

valid_dataset_flipped = datasets.CIFAR10(root='./data', train=False, transform

validloader_flipped = torch.utils.data.DataLoader(valid_dataset_flipped, batch
```

```
Files already downloaded and verified
```

```
# Evaluate and visualize
all_predictions, all_labels = evaluate_model(cifar10_model, validloader_flippe
```

```
Validation Accuracy on Flipped Images: 48.71%
```

```
show_predictions(cifar10_model, validloader_flipped, class_labels, num_images=
```



Flipping the validation images caused a significant drop in validation accuracy. The model seems to highly sensitive to image orientation.

3. Making the model more robust

Let’s randomly flip the training images to introduce variation in the training data. While this doesn’t increase the number of datapoints in the dataset, it effectively increases the number of unique examples the model encounters during training, as random transformations generate new variations in each epoch.

```
# Augment training data with horizontal and vertical flips
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),  # Random horizontal flip
    transforms.RandomVerticalFlip(p=0.5),    # Random vertical flip
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Standard CIFAR-1
])

# Load CIFAR-10 dataset
train_dataset_flipped = datasets.CIFAR10(root='./data', train=True, transform=
valid_dataset_flipped = datasets.CIFAR10(root='./data', train=False, transform

trainloader_flipped = torch.utils.data.DataLoader(train_dataset, batch_size=BA
validloader_flipped = torch.utils.data.DataLoader(valid_dataset, batch_size=BA
```

```
Files already downloaded and verified
```

```
Files already downloaded and verified
```

```
cifar10_model_flipped = CIFAR10_CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(cifar10_model_flipped.parameters(), lr=2e-3)
# results = trainer(cifar10_model_flipped, criterion, optimizer, trainloader_f
Epoch 1: Train Loss: 1.293. Valid Loss: 1.031. Valid Accuracy: 0.63.
Epoch 2: Train Loss: 0.888. Valid Loss: 0.927. Valid Accuracy: 0.68.
Epoch 3: Train Loss: 0.735. Valid Loss: 0.892. Valid Accuracy: 0.69.
Epoch 4: Train Loss: 0.622. Valid Loss: 0.854. Valid Accuracy: 0.72.
Epoch 5: Train Loss: 0.529. Valid Loss: 0.919. Valid Accuracy: 0.72.
Epoch 6: Train Loss: 0.447. Valid Loss: 0.950. Valid Accuracy: 0.72.
Epoch 7: Train Loss: 0.384. Valid Loss: 1.025. Valid Accuracy: 0.72.
Epoch 8: Train Loss: 0.333. Valid Loss: 1.105. Valid Accuracy: 0.72.
Epoch 9: Train Loss: 0.297. Valid Loss: 1.188. Valid Accuracy: 0.72.
Epoch 10: Train Loss: 0.259. Valid Loss: 1.293. Valid Accuracy: 0.72.
Epoch 11: Train Loss: 0.245. Valid Loss: 1.448. Valid Accuracy: 0.71.
Epoch 12: Train Loss: 0.225. Valid Loss: 1.519. Valid Accuracy: 0.72.
Epoch 13: Train Loss: 0.221. Valid Loss: 1.584. Valid Accuracy: 0.71.
Epoch 14: Train Loss: 0.216. Valid Loss: 1.596. Valid Accuracy: 0.72.
Epoch 15: Train Loss: 0.193. Valid Loss: 1.857. Valid Accuracy: 0.71.
Epoch 16: Train Loss: 0.190. Valid Loss: 1.922. Valid Accuracy: 0.70.
Epoch 17: Train Loss: 0.195. Valid Loss: 1.913. Valid Accuracy: 0.72.
Epoch 18: Train Loss: 0.184. Valid Loss: 1.808. Valid Accuracy: 0.72.
Epoch 19: Train Loss: 0.177. Valid Loss: 2.115. Valid Accuracy: 0.71.
Epoch 20: Train Loss: 0.186. Valid Loss: 2.039. Valid Accuracy: 0.70.
```

```python
# Save model
PATH = "../models/CIFAR10-flipped-images-cnn.pt"
# torch.save(cifar10_model_flipped.state_dict(), PATH)

# Load model
cifar10_model_flipped.load_state_dict(torch.load(PATH))
```

```
/var/folders/b3/g26r0dcx4b35vf3nk31216hc0000gr/T/ipykernel_74927/2235365162.py
  cifar10_model_flipped.load_state_dict(torch.load(PATH))
```

```
<All keys matched successfully>
```

```
# Assuming 'cifar10_model' is your trained model
cifar10_model.eval()
cifar10_model.to('cpu')

# CIFAR-10 class labels
class_labels = [
    "airplane", "automobile", "bird", "cat", "deer",
    "dog", "frog", "horse", "ship", "truck"
]

# Function to display images with predictions and true labels
def show_predictions(model, dataloader, class_labels, num_images=5):
    model.eval()  # Set model to evaluation mode
    with torch.no_grad():
        # Get a batch of validation data
        data_iter = iter(dataloader)
        images, true_labels = next(data_iter)

        # Get predictions
        outputs = model(images)
        _, predicted_labels = torch.max(outputs, 1)

        # Show images with predictions and true labels
        fig, axes = plt.subplots(1, num_images, figsize=(15, 3))
        for i in range(num_images):
            ax = axes[i]
            img = images[i].permute(1, 2, 0)  # Convert (C, H, W) to (H, W, C)
            ax.imshow((img * 0.5 + 0.5).numpy())  # Denormalize image
            ax.axis("off")
            ax.set_title(
                f"True: {class_labels[true_labels[i]]}\nPred: {class_labels[pr
                fontsize=10
            )
        plt.tight_layout()
        plt.show()
```

How's the performance of `cifar10_model_flipped` model on the flipped validation images?

```
all_predictions, all_labels = evaluate_model(cifar10_model_flipped, validloade
```

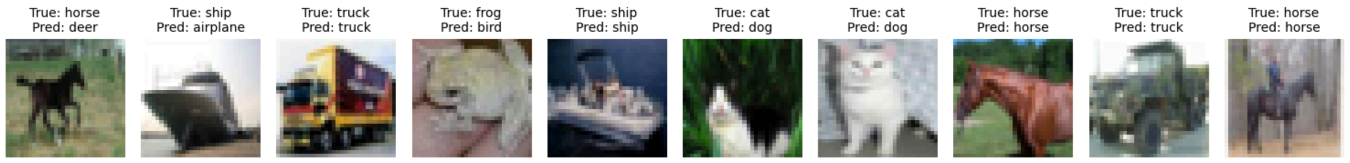
```
Validation Accuracy on Flipped Images: 70.25%
```

The model trained on randomly flipped images performas much better on flipped validation images. How's the performance of this model on the original validation images?

```
all_predictions, all_labels = evaluate_model(cifar10_model_flipped, validloade
Validation Accuracy on Flipped Images: 70.25%
```

The model is also performing well on the original validation images.

```
show_predictions(cifar10_model_flipped, validloader, class_labels, num_images=
```



Our model is more robust now!!