به نام خدا

ابتدا قسمت ج را توضیح میدهم و بعد توضیحات پیاده سازی را میدهم.

بخش ج)

در این پیاده سازی ما از **crossentropy** استفاده کرده ایم. این تابع خطا برای تسک های طبقه بندی و کلاس بندی بسیار مناسب است. این کلاس یک توزیع پیشبینی احتمالاتی محاسبه میکند در خروجی خودش برای هر کلاس و از **softmax** هم کمک گرفته است. این توزیع از برچسب های واقعی استخراج میشوند و از روش **one-hot** استفاده کرده ایم.

همچنین به صورت ضمنی از **negative log-likelihood** استفاده کرده ایم.

چرا از تابع؟ زیرا داده های **Caltech** دارای چندین کلاس هستند و این تابع برای مسائل طبقه بندی بسیار مناسب است. همانطور که گفتیم برای هر کلاس یک توزیع احتمالاتی محاسبه میشود. همچنین این تابع مشتق پذیر است که برای محاسبه کاهش گرادیان بسیار مفید است که بهینه سازی هایی مثل **adam** که در این کد هم استفاده کردیم از این روش استفاده میکنند.

بخش د)

مدل ما در یک اجرا یک اشتباهی بین داده های کلاس dalmatian یا سگ هایی که خال خالی هستند با گورخر ها انجام داد که نشان میدهد مدل گیج شده است. همچنین از کیفیت پایین بعضی عکس ها مدل رنج میبرد همچنین از تار بودن آنها. مدل بدون regularization اطمینان بیشتری دارد اما پیشبینی آن غلط است در حالی که با L1 اطمینان مدل کم شده است ولی عملکرد بهتری دارد.

در بعضی کلاس ها مدل به داده های بیشتری نیاز دارد.

توضیح پیاده سازی:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Subset, ConcatDataset
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np
import random
```

✓ 10.4s

در این قسمت کتاب خانه ها را وارد کردیم.

```python
class SimpleCNN(nn.Module):
    def __init__(self, l1_lambda=0, l2_lambda=0, dropout_rate=0):
        super(SimpleCNN, self).__init__()
        self.l1_lambda = l1_lambda
        self.l2_lambda = l2_lambda
        self.conv1 = nn.Conv2d(3, 16, kernel_size=5, padding=2)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2, 2)
        self.dropout1 = nn.Dropout(dropout_rate)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=5, padding=2)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(2, 2)
        self.dropout2 = nn.Dropout(dropout_rate)
        self.fc1 = nn.Linear(32 * 56 * 56, 128)  # Assuming input size of 224x224
        self.relu3 = nn.ReLU()
        self.dropout3 = nn.Dropout(dropout_rate)
        self.fc2 = nn.Linear(128, 102)  # Caltech-101 has 101 categories + background

    def forward(self, x):
        x = self.pool1(self.relu1(self.conv1(x)))
        x = self.dropout1(x)
        x = self.pool2(self.relu2(self.conv2(x)))
        x = self.dropout2(x)
        x = x.view(-1, 32 * 56 * 56)
        x = self.relu3(self.fc1(x))
        x = self.dropout3(x)
        x = self.fc2(x)
        return x
```

در این قسمت معماری مدل کانولوشنی شبکه عصبی خودمان را که برای کلاس بندی داده های عکس است تعریف کردیم. همچنین l1,l2 و به همراه droput لحاظ کرده ایم.

```python
data_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.Grayscale(num_output_channels=3),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

try:
    full_dataset = datasets.Caltech101(root='./data', download=True, transform=data_transform)
except RuntimeError as e:
    if "HTTP Error 503: Service Unavailable" in str(e):
        print("Error: Caltech-101 server might be unavailable. Try again later or use a different dataset.")
        exit()
```

[4] ✓ 0.0s

```
Files already downloaded and verified
```

در مرحله بعد داده ها را لود کردیم و همانطور که با شما صحبت کردم اجازه دادید که داده های که خاکستری و دارای ۱ چنل هستند را به ۳ چنل تبدیل کنم. به تنسور ها تبدیل کردیم و در نهایت دیتاست را دانلود کردیم.

```
train_size = int(0.8 * len(subset_dataset))
val_size = int(0.1 * len(subset_dataset))
test_size = len(subset_dataset) - train_size - val_size

train_dataset, val_dataset, test_dataset = torch.utils.data.random_split(subset_dataset, [train_size, val_size, test_size])

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

✓ 0.0s

همانطور که در سوال از ما خواسته شد ۸۰ درصد داده ها را برای **train** و بقیه را برای ارزیابی و تست کنار گذاشتیم.

در مرحله بعد **dataloader** ها را ساختیم.

```python
def train_and_evaluate(model, train_loader, val_loader, test_loader, num_epochs=10):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    train_losses = []
    val_losses = []
    train_accuracies = []
    val_accuracies = []

    for epoch in range(num_epochs):
        # Training phase
        model.train()
        running_loss = 0.0
        correct_train = 0
        total_train = 0

        for i, (images, labels) in enumerate(train_loader):
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)

            # Add L1 and/or L2 regularization loss if applicable
            if model.l1_lambda > 0:
                loss += model.l1_regularization_loss()
            if model.l2_lambda > 0:
                loss += model.l2_regularization_loss()

            loss.backward()
            optimizer.step()
```

مدل CNN را بر روی داده های آموزشی آموزش می دهد و عملکرد آن را بر روی داده های اعتبارسنجی ارزیابی می کند.

حلقه آموزشی اصلی، از جمله پاس رو به جلو، محاسبه ضرر، انتشار پس‌انداز و بهینه‌سازی را پیاده‌سازی می‌کند.

از دست دادن و دقت آموزش و اعتبارسنجی را محاسبه و پیگیری می کند.

در صورتی که در مدل مشخص شده باشد، منطق اضافه کردن از دست دادن تنظیم L1 و/یا L2 را شامل می شود.

اطلاعات پیشرفت آموزش را برای هر دوره چاپ می کند.

معیارهای آموزشی و اعتبار سنجی را برای تجزیه و تحلیل بیشتر برمی گرداند.

در قسمت بعدی همانطور که خواسته شده بود برای بهینه سازی ها و بدون بهینه سازی خروجی گرفتیم و بررسی کردیم.

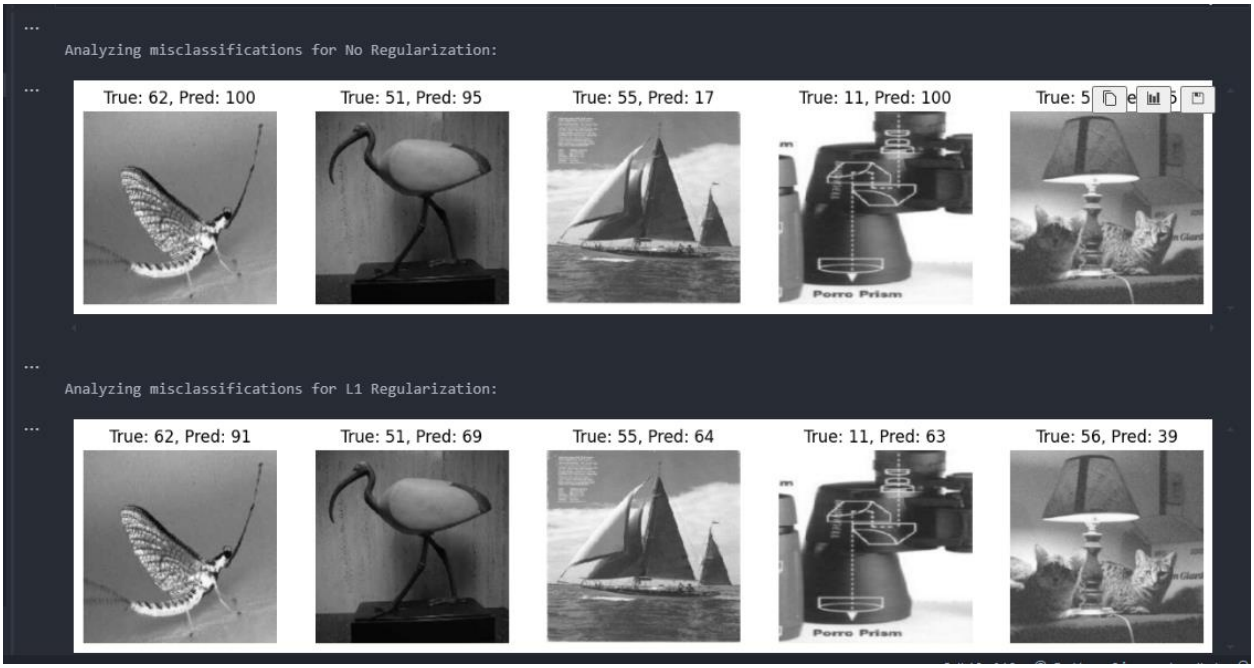
```python
# --- Analyze Misclassifications ---
def analyze_misclassifications(model, test_loader, num_images=5):
    model.eval()
    misclassified_images = []
    true_labels = []
    predicted_labels = []

    with torch.no_grad():
        for images, labels in test_loader:
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)

            for i in range(len(images)):
                if predicted[i] != labels[i]:
                    misclassified_images.append(images[i])
                    true_labels.append(labels[i])
                    predicted_labels.append(predicted[i])

                    if len(misclassified_images) >= num_images:
                        break
            if len(misclassified_images) >= num_images:
                break

    # Display misclassified images
    fig, axes = plt.subplots(1, num_images, figsize=(15, 3))
    for i in range(num_images):
        img = misclassified_images[i].permute(1, 2, 0).numpy()
        # Unnormalize the image for display
        img = img * np.array([0.229, 0.224, 0.225]) + np.array([0.485, 0.456, 0.406])
```



در این بخش همانطور که در بخش آخر سوال خواستید چند داده به عنوان تست انتخاب کردیم و عملکرد را بررسی کردیم.

یک مدل آموزش دیده را در مجموعه آزمایشی ارزیابی می کند و تصاویر طبقه بندی اشتباه را شناسایی می کند.

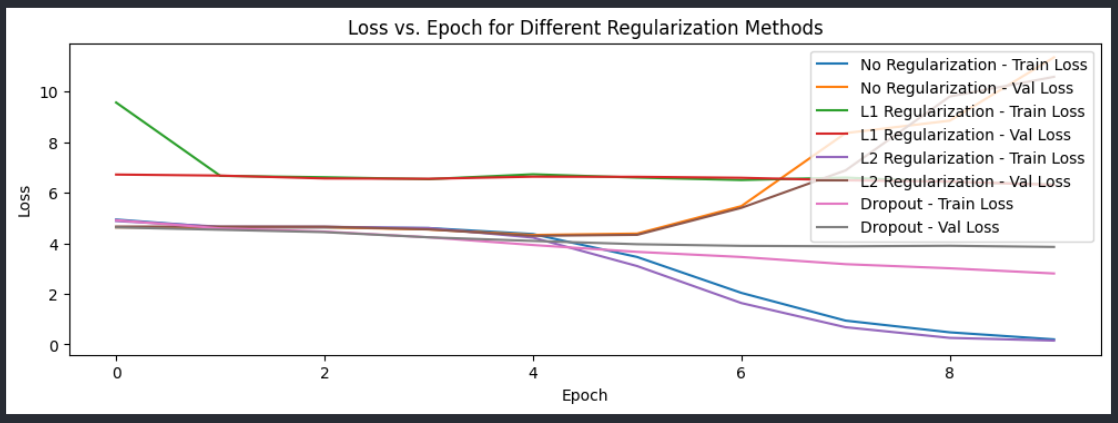
تعداد مشخصی از تصاویر طبقه بندی شده اشتباه را به همراه برچسب های واقعی و پیش بینی شده آنها نمایش می دهد.

به بازرسی بصری و درک خطاهای مدل کمک می کند.

در نهایت خروجی های خواسته شده را **plot** کردیم با استفاده از کتابخانه **matplotlib**.

[21] ✓ 0.4s Python

```
<matplotlib.legend.Legend at 0x220dc8c0550>
```



[22] ✓ 0.3s