Deep Learning - Training a Simple Convolution Neural Network Model

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
import matplotlib.pyplot as plt

# Step 1: Define image dimensions, batch size, and directory paths
image_height = 100
image_width = 100
batch_size = 32

train_dir = '/content/drive/MyDrive/TrainingImages/'

# Step 2: Data generators for training
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255, validation_split=0.1)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'  # Use only training data
)

validation_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'  # Use only validation data
)

# Step 3: Define the model architecture
model = Sequential([
    Conv2D(8, (3, 3), activation='relu', input_shape=(image_height, image_width, 3)),
    Conv2D(8, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(16, activation='relu'),
    Dense(16, activation='relu'),
    Dense(10, activation='softmax')
])

# Step 4: Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Step 5: Train the model
history = model.fit(train_generator,
                    epochs=20,
                    validation_data=validation_generator)

# Step 6(a): Plot learning curves
plt.plot(history.history['accuracy'], marker='o', label='Training Accuracy', color='blue')
plt.plot(history.history['val_accuracy'], marker='o', label='Validation Accuracy', color='red')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Learning Curves')
plt.legend()
plt.show()
```

```
Found 4201 images belonging to 10 classes.
Found 464 images belonging to 10 classes.
Epoch 1/20
132/132 [==============================] - 1161s 9s/step - loss: 1.0615 - accuracy: 0.6798 - val_loss: 0.3367 - val_accuracy: 0.8276
Epoch 2/20
132/132 [==============================] - 25s 188ms/step - loss: 0.1064 - accuracy: 0.9738 - val_loss: 0.0148 - val_accuracy: 1.0000
Epoch 3/20
132/132 [==============================] - 25s 185ms/step - loss: 0.0250 - accuracy: 0.9955 - val_loss: 0.0199 - val_accuracy: 1.0000
Epoch 4/20
128/132 [============================>.] - ETA: 0s - loss: 0.0101 - accuracy: 0.9983
```

Transfer Learning via Feature Extraction

```
import torch
```

```
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

# Step 1: Define configuration variables
image_height = 224  # Modified image height
image_width = 224  # Modified image width
batch_size = 32
num_classes = 10
num_epochs = 10
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Step 2: Load the dataset using ImageDataLoader and organize it on disk
train_dir = "/content/drive/MyDrive/TrainingImages/"
test_dir = "/content/drive/MyDrive/TestingImages/"

train_dataset = torchvision.datasets.ImageFolder(
    root=train_dir,
    transform=torchvision.transforms.Compose([
        torchvision.transforms.Resize((image_height, image_width)),
        torchvision.transforms.ToTensor(),
    ])
)

test_dataset = torchvision.datasets.ImageFolder(
    root=test_dir,
    transform=torchvision.transforms.Compose([
        torchvision.transforms.Resize((image_height, image_width)),
        torchvision.transforms.ToTensor(),
    ])
)

# Define data loaders for training and testing
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=batch_size,
    shuffle=True
)

test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=batch_size,
    shuffle=False
)

# Define a function to calculate accuracy
def calculate_accuracy(loader, model):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    return correct / total

# Define the model, criterion, and optimizer
resnet18 = torchvision.models.resnet18(pretrained=True)
for param in resnet18.parameters():
    param.requires_grad = False

num_ftrs = resnet18.fc.in_features
resnet18.fc = nn.Linear(num_ftrs, num_classes)
resnet18 = resnet18.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(resnet18.parameters(), lr=0.001)

# Lists to store training and validation accuracies
train_accuracies, val_accuracies = [], []

# Train the model
for epoch in range(num_epochs):
    running_loss = 0.0
```

```
        correct_train, total_train = 0, 0
        resnet18.train()

        # Use train_loader for both training and validation
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = resnet18(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

            # Calculate training accuracy
            _, predicted = torch.max(outputs.data, 1)
            total_train += labels.size(0)
            correct_train += (predicted == labels).sum().item()

        # Calculate training accuracy
        train_accuracy = correct_train / total_train
        train_accuracies.append(train_accuracy)

        # Calculate validation accuracy
        val_accuracy = calculate_accuracy(train_loader, resnet18)  # Use train_loader for validation
        val_accuracies.append(val_accuracy)

        print("Epoch: {}/{}.. ".format(epoch+1, num_epochs),
              "Training Loss: {:.3f}.. ".format(running_loss/len(train_loader)),
              "Training Accuracy: {:.3f}.. ".format(train_accuracy),
              "Validation Accuracy: {:.3f}".format(val_accuracy))

    # Plot learning curves
    epochs = range(1, num_epochs + 1)
    plt.plot(epochs, train_accuracies, marker='o', label='Training Accuracy', color='blue')
    plt.plot(epochs, val_accuracies, marker='o', label='Validation Accuracy', color='red')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Learning Curves')
    plt.legend()
    plt.show()


    # Step 6: Extract features from ResNet18 for training and test datasets
    class FeatureExtractor:
        def __init__(self, model):
            self.model = model
            self.features = None
            self.hook = self.model.layer4.register_forward_hook(self.hook_fn)

        def hook_fn(self, module, input, output):
            self.features = output

        def extract(self, x):
            self.model(x)
            return self.features

    train_feature_extractor = FeatureExtractor(resnet18)
    test_feature_extractor = FeatureExtractor(resnet18)

    train_features = []
    train_labels = []

    test_features = []
    test_labels = []

    # Extract features and labels from the training dataset
    for images, labels in train_loader:
        features_batch = train_feature_extractor.extract(images.to(device))
        train_features.append(features_batch.detach().cpu().numpy())
        train_labels.append(labels.numpy())

    # Extract features and labels from the test dataset
    for images, labels in test_loader:
        features_batch = test_feature_extractor.extract(images.to(device))
        test_features.append(features_batch.detach().cpu().numpy())
        test_labels.append(labels.numpy())

    train_features = np.concatenate(train_features)
    train_labels = np.concatenate(train_labels)
    test_features = np.concatenate(test_features)
    test_labels = np.concatenate(test_labels)


    # Flatten the features
```

```
train_features_flattened = train_features.reshape(train_features.shape[0], -1)
test_features_flattened = test_features.reshape(test_features.shape[0], -1)

# Step 7: Train SVM prediction model using extracted features
svm_model = SVC(kernel='rbf', C=10)
svm_model.fit(train_features_flattened, train_labels)

# Step 8: Evaluate SVM model
train_predictions = svm_model.predict(train_features_flattened)
test_predictions = svm_model.predict(test_features_flattened)

train_accuracy = accuracy_score(train_labels, train_predictions)
test_accuracy = accuracy_score(test_labels, test_predictions)

print("SVM Training Accuracy", train_accuracy)
print("SVM Test Accuracy", test_accuracy)
```

Transfer Learning via Fine-Tuning

```
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split

# Step 1: Define configuration variables
image_height = 224  # Modified image height
image_width = 224  # Modified image width
batch_size = 32
num_classes = 10
num_epochs = 10
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Step 2: Load the dataset using ImageDataLoader and organize it on disk
train_dir = "/content/drive/MyDrive/TrainingImages/"
test_dir = "/content/drive/MyDrive/TestingImages/"

# Load the dataset without splitting
full_dataset = torchvision.datasets.ImageFolder(
    root=train_dir,
    transform=torchvision.transforms.Compose([
        torchvision.transforms.Resize((image_height, image_width)),
        torchvision.transforms.ToTensor(),
    ])
)

# Split the training dataset into training and validation sets manually
val_split = 0.1
train_size = int(len(full_dataset) * (1 - val_split))
val_size = len(full_dataset) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(full_dataset, [train_size, val_size])

train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=batch_size,
    shuffle=True
)

val_loader = torch.utils.data.DataLoader(
    val_dataset,
    batch_size=batch_size,
    shuffle=False
)

test_dataset = torchvision.datasets.ImageFolder(
    root=test_dir,
    transform=torchvision.transforms.Compose([
        torchvision.transforms.Resize((image_height, image_width)),
        torchvision.transforms.ToTensor(),
    ])
)

test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=batch_size,
    shuffle=False
)

# Step 3: Build the model (fine-tuning)
resnet18 = torchvision.models.resnet18(weights=None)
num_ftrs = resnet18.fc.in_features
resnet18.fc = nn.Linear(num_ftrs, num_classes).to(device)
resnet18 = resnet18.to(device)

# Step 4: Train the model
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(resnet18.parameters(), lr=0.001)

train_accuracies, val_accuracies, test_accuracies = [], [], []

# Train the model
for epoch in range(num_epochs):
    running_loss = 0.0
    correct_train, total_train = 0, 0
    resnet18.train()
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = resnet18(images)
        loss = criterion(outputs, labels)
```

```
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
```