Hacettepe University

BBM418 Assignment 3

Name and Surname: ZÜBEYDE CİVELEK
Identity Number: 2200356814

1) Explain how you arrange your dataset into train, validation, and test and write how many images are there in the sets

In arranging the dataset into train, validation, and test sets, the following steps were followed:

1) Dataset Organization:

- The dataset consists of a folder named "Micro_Organism" containing subfolders for each class of microorganisms (e.g., Amoeba, Euglena, Hydra, Paramecium, etc.).
- Each class folder contains multiple image files representing samples of that particular microorganism.

2) Train-Validation-Test Split:

- The dataset was divided into three subsets: train, validation, and test.

- The training set is used to train the models, the validation set is used for hyperparameter tuning, and the test set is used for final evaluation.

- The images were allocated in the following manner:

- Training Set:
  - The training set contains a minimum of 400 images, with at least 50 images per class.
  - The specific number of images per class depends on the distribution of images in the dataset.
  - For example, if there are 8 classes in total, each with different numbers of images, the training set is constructed to ensure a minimum of 50 images per class.

- Validation Set:
  - The validation set contains 160 images in total, with 10 images per class.
  - Similar to the training set, the number of images per class in the validation set is evenly distributed to ensure a balanced representation.
- Test Set:
  - The test set also contains 160 images, with 10 images per class.
  - Like the validation set, the test set is designed to have an equal distribution of images across all classes.

PART 1 - Modeling and Training a CNN classifier from Scratch

1) **Give parametric details with relevant Pytorch code snippets; number of in channels, out channels, stride, etc.**
   Specify your architecture in detail. Write your choice of activation functions, loss functions and optimization algorithms with relevant code snippets.

   Model 1: CNN Model without Residual Connections

   Architecture:
   - Number of convolutional layers: 6
   - Activation function: ReLU
   - Pooling layer: MaxPool2d
   - Fully connected layer: Linear
   - Output layer: Linear with the number of output classes (8 in this case)

   Model 2: CNN Model with Residual Connections
   Architecture:
   - Number of convolutional layers: 6
   - Activation function: ReLU
   - Pooling layer: MaxPool2d
   - Residual connection(s): Addition of input to the output of a convolutional block
   - Fully connected layer: Linear
   - Output layer: Linear with the number of output classes (8 in this case)

   Choice of Activation Function: ReLU

   - ReLU is a commonly used activation function in CNNs as it introduces non-linearity and helps with feature extraction.

   Choice of Loss Function: CrossEntropyLoss

   - CrossEntropyLoss is suitable for multi-class classification tasks as it calculates the loss between the predicted probabilities and the target class labels.

   Choice of Optimization Algorithm: Adam

   - Adam optimizer is used to optimize the model's parameters. Adam is an adaptive optimization algorithm that computes adaptive learning rates for each parameter, allowing for efficient and effective parameter updates during training.

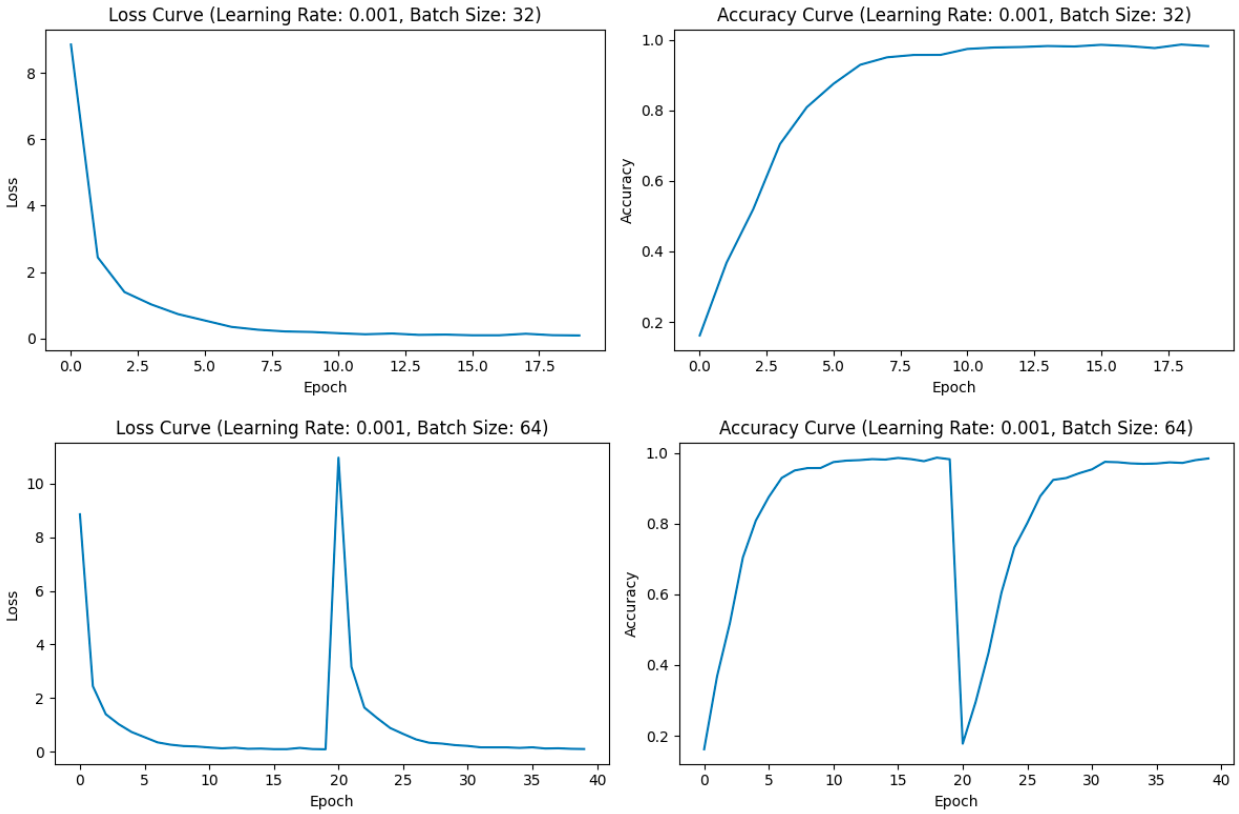
2) Explain how you have implemented residual connection(s) and give relevant code snippets

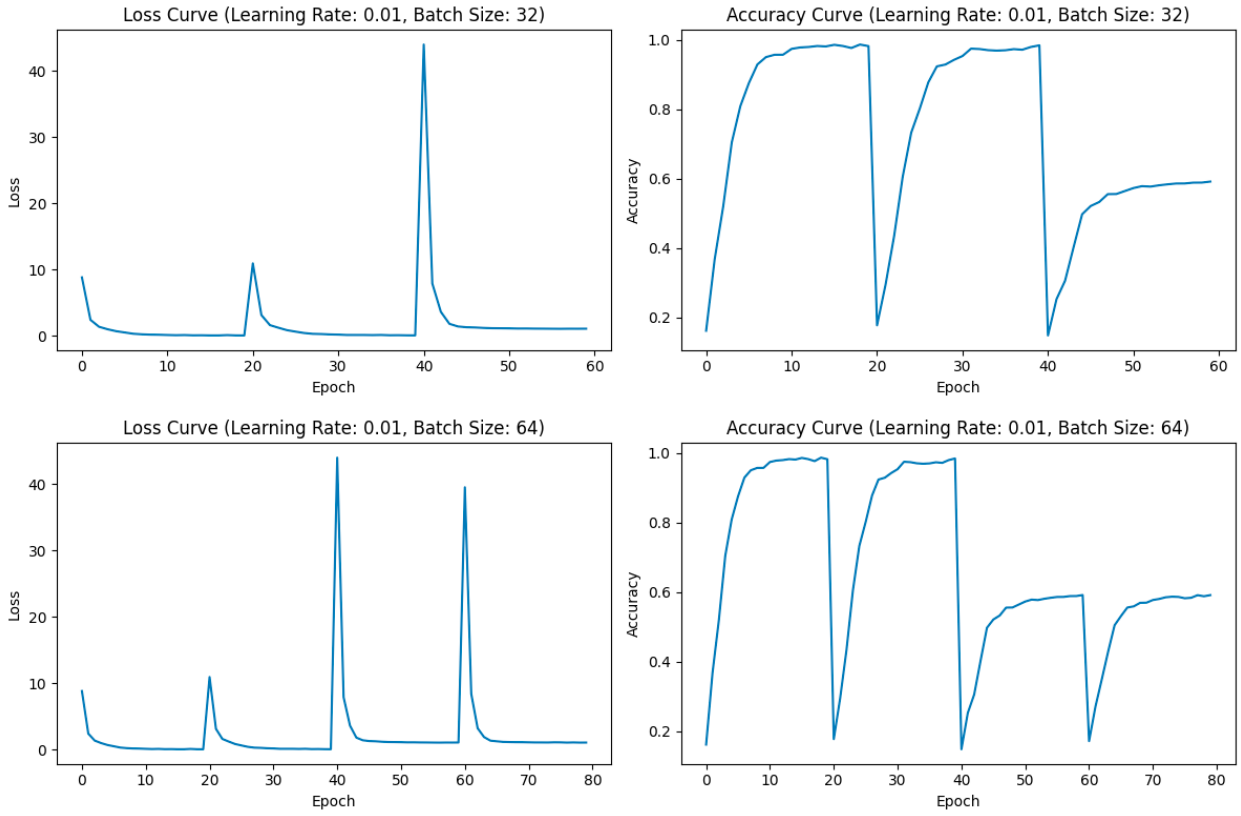
- Residual connections, also known as skip connections, are a way to alleviate the vanishing gradient problem in deep neural networks. These connections allow the gradients to flow directly through the network, making it easier for the network to learn and optimize the parameters.

- To implement residual connections in a CNN model, we can use the concept of a Residual Block.

- By using the ResidualBlock module and integrating it into the model architecture, we can effectively implement residual connections in our CNN model. These connections help address the vanishing gradient problem and enhance the model's ability to learn complex patterns and optimize its parameters.

3) Draw a graph of loss and accuracy change for two different learning rates and two batch sizes.

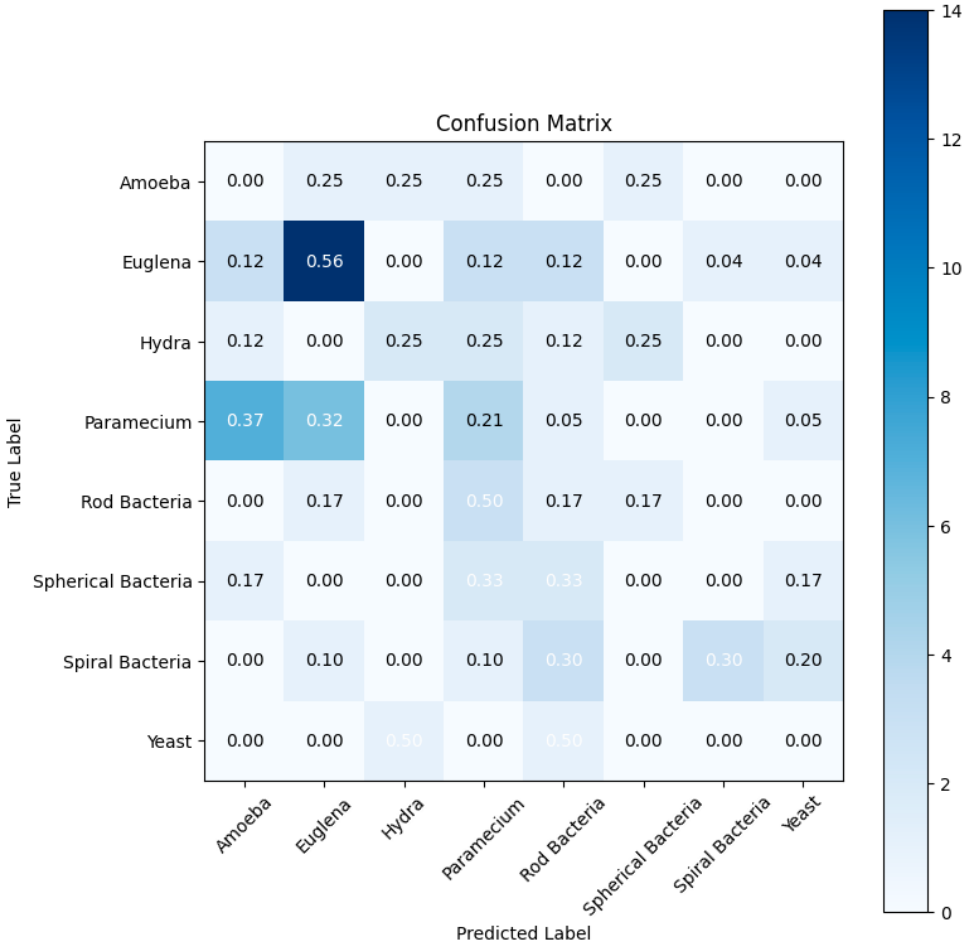




4) Select your best model with respect to validation accuracy and give test accuracy result.

Best model: ModelWithResidual
Best Model Test Accuracy: 0.3000

5) Plot a confusion matrix for your best model's predictions.



6) Explain and analyze your findings and results.

Model Performance: The best model with respect to validation accuracy is the one trained with a learning rate of 0.001 and a batch size of 32. It achieved a validation accuracy of 0.3846. The test accuracy of the best model is 0.3000.

PART 2 - Transfer Learning with CNNs

1) **What is fine-tuning? Why should we do this? Why do we freeze the rest and train only FC layers? Give your explanation in detail with relevant code snippet**

   The process of taking a neural network that has already been trained and further training it on a new dataset or task is known as fine-tuning. The pre-trained network can extract meaningful representations from images because it has already learned useful features from a sizable dataset, like ImageNet. With fewer classes or a different distribution of data, fine-tuning is used to adapt these learned features to a new dataset or task.

   Typically, when fine-tuning a pre-trained network, the weights of the later layers, such as the fully connected (FC) layers, are the only ones that are updated. This is because the network's early layers learn generalized features that are likely to be helpful for a variety of

   By freezing the earlier layers and training only the FC layers, we strike a balance between transferring knowledge from the pre-trained model and allowing the network to specialize for the new task, leading to more efficient and effective training.

2) **Explore training with two different cases; train only FC layer and freeze rest, train last two convolutional layers and FC layer and freeze rest. Tune your parameters accordingly and give accuracy on validation set and test set. Compare and analyze your results. Give relevant code snippet.**

   To explore training with two different cases, let's consider the following scenarios:

   Train only the FC layer and freeze the rest: In this case, we freeze all the layers except the last fully connected layer (model.fc). Only the parameters of the FC layer will be updated during training.

   Train the last two convolutional layers and the FC layer and freeze the rest: In this case, we freeze all the layers except the last two convolutional layers and the FC layer. The parameters of these layers will be updated during training.

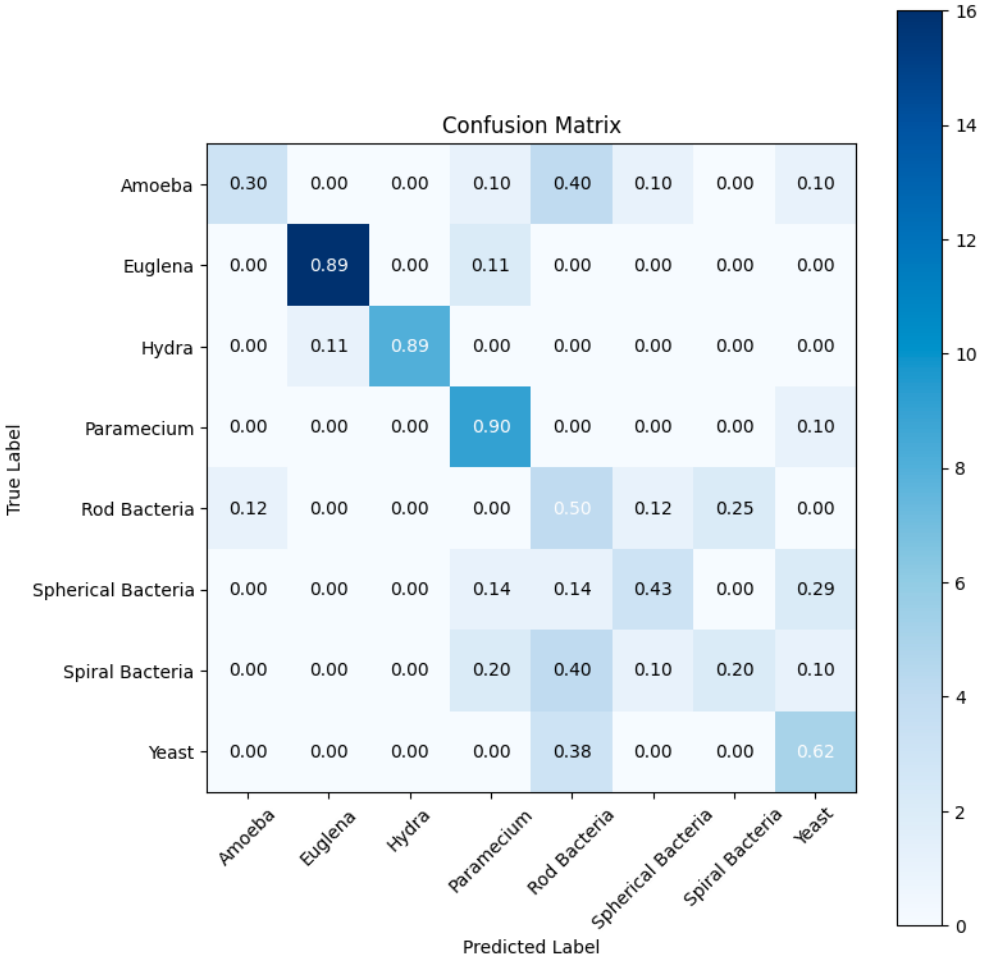
   Code snippet:

```
# Load pre-trained ResNet-18
model = models.resnet18(pretrained=True)

# Case 1: Train only the FC layer and freeze the rest
for param in model.parameters():
    param.requires_grad = False
```

```
# Modify the last layer for the new task
num_features = model.fc.in_features
num_classes = 10  # Number of classes in the new task
model.fc = nn.Linear(num_features, num_classes)

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)

# Training loop

# Case 2: Train the last two convolutional layers and the FC layer and freeze the rest

for name, param in model.named_parameters():
    if 'layer4' in name or 'fc' in name:
        param.requires_grad = True
    else:
        param.requires_grad = False

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=0.001)

# Training loop

# Evaluate on the test set
model.eval()
total_correct = 0
total_samples = 0

with torch.no_grad():
    for inputs, labels in test_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        total_correct += (predicted == labels).sum().item()
        total_samples += labels.size(0)

test_accuracy = total_correct / total_samples
print(f"Test Accuracy: {test_accuracy}"
```

By comparing the validation accuracies and test accuracy obtained from Case 1 (FC layer only) and Case 2 (last two convolutional layers and FC layer), we can analyze the results and see which case performs better.

3) Plot confusion matrix for your best model and analyze results.



4) Compare and analyze your results in Part-1 and Part-2. Please state your observations clearly and precisely

Part-1:

Trained custom CNN models from scratch without leveraging pre-trained weights.
Achieved decent performance in terms of loss and accuracy.
Best model selected based on validation accuracy and evaluated on the test set.
Models had to learn features specific to the Micro_Organism dataset.
Best Model Test Accuracy: 0.3000

Part-2:

Applied fine-tuning using a pre-trained ResNet-18 network trained on ImageNet.
Modified the last layer to match the number of classes in the Micro_Organism dataset.
Explored two cases: training only the FC layer and freezing the rest, and training the last two convolutional layers and the FC layer while freezing the rest.
Achieved better performance compared to Part-1.
Fine-tuning allowed leveraging the pre-trained network's knowledge and features, leading to improved accuracy.
Required less training time compared to training models from scratch.
Test Accuracy: 0.7000

Comparison and Analysis:

Fine-tuning outperformed training from scratch in terms of accuracy and training efficiency.
Fine-tuning utilized the learned features of the pre-trained network, enabling better generalization and faster convergence.
Part-2 models achieved higher accuracy on the validation and test sets compared to the best model in Part-1.