Image Classification Report

QUESTION 1: What effect does increasing the image size have on the training time and accuracy? Why might this happen?

Results: The results are illustrated in Table 1.

Image Size (pixels)	Training Time (5 epochs, seconds)	Accuracy (%)
64 × 64	156.99	73.49
128 × 128	710.30	74.89

Table 1

Answer: When we increase the image size from 64 × 64 to 128 × 128, the area of the image increases by 4 times from the original size. Consequently, the time required for processing, especially in convolutional layers, also expands to approximately 4 times that of the original size. The accuracy does not show a significant change but may slightly increase, as a larger image size allows the model to capture more details, potentially improving accuracy due to the additional processed information.

QUESTION 2: How does changing the number of filters affect the model's accuracy? What might be the reason for this change?

Code: The codes are illustrated in Figure 1.

```
model = nn.Sequential(
    nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, stride=1, padding=1),  # First conv layer
    nn.ReLU(),  # Activation function
    nn.MaxPool2d(kernel_size=2, stride=2),  # Pooling layer (reduce size)

    nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1),  # Second conv layer
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),  # Pooling

    nn.Flatten(),  # Flatten the output from the conv layers for the fully connected layer
    nn.Linear(64 * 16 * 16, 64),  # Fully connected layer
    nn.ReLU(),
    nn.Linear(64, 2),  # Output layer (assuming 2 classes in the dataset)
    nn.Softmax(dim=1)  # Softmax for classification
)
```

Figure 1

Result: The results are illustrated in Table 2.

Number of Filters (first layer and second layer)	Accuracy (%)
16 and 32	73.49
32 and 64	47.77

Table 2

Answer: When we increase the number of filters in the first and second layers from 16 and 32 to 32 and 64, respectively, the accuracy dropped significantly from 73.49% to 47.77%. During training, the loss remains stable at 0.6933, indicating that the model might not be learning effectively and could be stuck in a local minimum or plateau. This behavior suggests that the model may be overfitting due to the excessive number of filters on a relatively simple dataset. The higher model complexity leads to weaker generalization, capturing noise rather than meaningful patterns, which explains the significant drop in accuracy.

QUESTION 3: How does adding a dropout layer affect the model's training and test accuracy? Explain why dropout can help prevent overfitting.

Code: The codes are illustrated in Figure 2.

```
    nn.Flatten(),  # Flatten the output from the conv layers for the fully connected layer
    nn.Linear(64 * 16 * 16, 64),  # Fully connected layer
    nn.ReLU(),
    nn.Dropout(p=0.5),
```

Figure 2

Result: The results are illustrated in Table 3 and Table 4.

Epoch	1	2	3	4	5
Training Loss (No Dropout)	0.6933	0.6933	0.6933	0.6933	0.6933
Training Loss (With Dropout)	0.6443	0.5908	0.5690	0.5522	0.5291

Table 3

Accuracy (No Dropout, %)	47.77
Accuracy (With Dropout, %)	74.69

Table 4

Answer: When we added a dropout layer between the fully connected layers, it is clear that the loss converges more effectively, and the accuracy returns to the original level, even slightly increasing to 74.69%. The reason a dropout layer can help prevent overfitting is that it randomly deactivates some neurons during training, reducing the model's over-reliance on specific patterns in the training data and simultaneously decreasing the model's complexity. Additionally, the dropout layer encourages the model to learn features through different paths, making it more robust to new data and enhancing the model's generalization ability.

QUESTION 4: Add a third convolutional layer to the model with 128 filters, observe how this affects the model's performance and training time, and describe what you found.

Code: The codes are illustrated in Figure 3.

```
nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1, padding=1),  # Third conv layer
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),  # Pooling

    nn.Flatten(),  # Flatten the output from the conv layers for the fully connected layer
    nn.Linear(128 * 8 * 8, 64),  # Fully connected layer
    nn.ReLU(),
    nn.Dropout(p=0.5),
```

Figure 3

Result: The results are illustrated in Table 5.

Number of Convolutional Layers	Training Time (5 epochs, seconds)	Accuracy (%)
2	207.61	74.69
3	242.72	75.54

Table 5

Answer: When we added a third convolutional layer to the original model, both the training time and accuracy increased. The higher network complexity requires more computation time for training, while also allowing the neural network to capture more details. As the network depth increases, we can achieve better performance. With this improvement, we achieved the highest accuracy across all tasks. However, a deeper network can also lead to issues like overfitting. Therefore, techniques like ResNet have been developed to address this problem, enabling us to train much deeper neural networks while maintaining strong performance on tasks like image classification.