VGG-13 Image Classification

1. Describe the CNN model you have defined.

The Convolutional Neural Netwok is based on the VGG-13 architecture, which consists of 10 convolutional layers and 3 fully connected layers. ReLU activation function is used after each convolutional layer, except the output layer. The first two convolutional layers have 64 filters with kernel size 3x3, stride of 1, and padding of 1. The next two convolutional layers have 128 filters, the next two convolutional layers have 256 filters, the last two convolutional layers have 512 filters. Max pooling is applied after every two convolutional layers, with 2x2 kernel size and stride of 2. There are three fully connected layers. Each fully connected layer has 4096, 4096, 1000 and 3 (num_classes: number of class labels) neurons respectively. The output layer has the log SoftMax activation function. The early stopping, built on top of the base VGG-13 model, has dropout layers between the fully connected layers.

2. Describe how the techniques (regularization, dropout, early stopping) have impacted the performance of the model.

The base model achieved accuracy of 88.93% on the test dataset.

Regularization: While defining the optimizer for the model, I added 1 parameter, weight_decay = 1e-5, which is L2 regularization parameter. Regularization helps to prevent overfitting by adding a penalty term to the loss function. It helps in the model generalization capability.

After applying the regularization techniques on the base model, the accuracy of the model increased and reached 92.00%.

Dropout: On top of the above model (base + regularization), I added the dropout layers between the fully connected layers with a probability of 0.5, which helps to drop some neurons randomly during the training phase. Thus, helping to reduce overfitting and improves the model robustness.

After applying dropout layers, the accuracy of the model decreased slightly and attained 90.84% on the test dataset.

Early Stopping: Then, I implemented early stopping by monitoring the validation loss during the training phase. If the validation loss doesn't improve for a certain number of epochs, the counter value is increased. Once it crosses the patience value of 4, the training is stopped, and the model weights are saved from the epoch where the validation loss was minimum when the counter increment started. The model stopped the training phase at epoch 19. At the point where the model weights were saved, the training accuracy and loss were 92.71% and 0.0018 respectively. The validation accuracy and loss were 94.167% and 0.0019 respectively.

After implementation of early stopping, I observed the accuracy increased significantly to 93.02% and the test loss was 0.1995.

Image Augmentation: After the implementation of the early stopping method, we used the image augmentation technique to alter the dataset and test on the model built. We resized the images to 64x64 size, added random horizontal flip with p=0.5 during the transformation and loaded these images in the dataloader to test the model. We received the test accuracy of 87.46% with 0.3422 loss.

3. Discuss the results and provide relevant graphs:

a. Report training accuracy, training loss, validation accuracy, validation loss, testing accuracy, and testing loss.

Training Accuracy, Training Loss, Validation Accuracy, Validation Loss, Testing Accuracy, and Testing Loss of the base model:

```
[18] print(f"Training Accuracy: {train_accuracy*100:.4f}, Training Loss: {LOSS:.4f}")
     print(f"Validation Accuracy: {validation_accuracy*100:.4f}, Validation Loss: {validation_loss:.4f}")
     print(f"Testing Accuracy: {test_accuracy*100:.4f}, Testing Loss: {test_loss:.4f}")

     Training Accuracy: 90.0714, Training Loss: 0.0045
     Validation Accuracy: 88.9111, Validation Loss: 0.0111
     Testing Accuracy: 88.9333, Testing Loss: 0.3125
```

Training Accuracy, Training Loss, Validation Accuracy, Validation Loss, Testing Accuracy, and Testing Loss of the **(base + regularization) model**:

```
[26] print(f"Training Accuracy: {train_accuracy*100:.4f}, Training Loss: {LOSS:.4f}")
     print(f"Validation Accuracy: {validation_accuracy*100:.4f}, Validation Loss: {validation_loss:.4f}")
     print(f"Testing Accuracy: {test_accuracy*100:.4f}, Testing Loss: {test_loss:.4f}")

     Training Accuracy: 91.7333, Training Loss: 0.0036
     Validation Accuracy: 91.9556, Validation Loss: 0.0075
     Testing Accuracy: 92.0000, Testing Loss: 0.2216
```

Training Accuracy, Training Loss, Validation Accuracy, Validation Loss, Testing Accuracy, and Testing Loss of the **(base + regularization + dropout) model**:

```
[34] print(f"Training Accuracy: {train_accuracy*100:.4f}, Training Loss: {LOSS:.4f}")
     print(f"Validation Accuracy: {validation_accuracy*100:.4f}, Validation Loss: {validation_loss:.4f}")
     print(f"Testing Accuracy: {test_accuracy*100:.4f}, Testing Loss: {test_loss:.4f}")

     Training Accuracy: 89.8476, Training Loss: 0.0045
     Validation Accuracy: 90.6889, Validation Loss: 0.0063
     Testing Accuracy: 90.8444, Testing Loss: 0.2608
```

Training Accuracy, Training Loss, Validation Accuracy, Validation Loss, Testing Accuracy, and Testing Loss of the model **(base + regularization + dropout + early stopping):**

```
[41] print(f"Training Accuracy: {train_accuracy*100:.4f}, Training Loss: {LOSS:.4f}")
     print(f"Validation Accuracy: {validation_accuracy*100:.4f}, Validation Loss: {validation_loss:.4f}")
     print(f"Testing Accuracy: {test_accuracy*100:.4f}, Testing Loss: {test_loss:.4f}")

     Training Accuracy: 94.3333, Training Loss: 0.0024
     Validation Accuracy: 92.8444, Validation Loss: 0.0019
     Testing Accuracy: 93.0222, Testing Loss: 0.1995
```

Training Accuracy, Training Loss, Validation Accuracy, Validation Loss, Testing Accuracy, and Testing Loss of the model **(base + regularization + dropout + early stopping + image augmentation):**

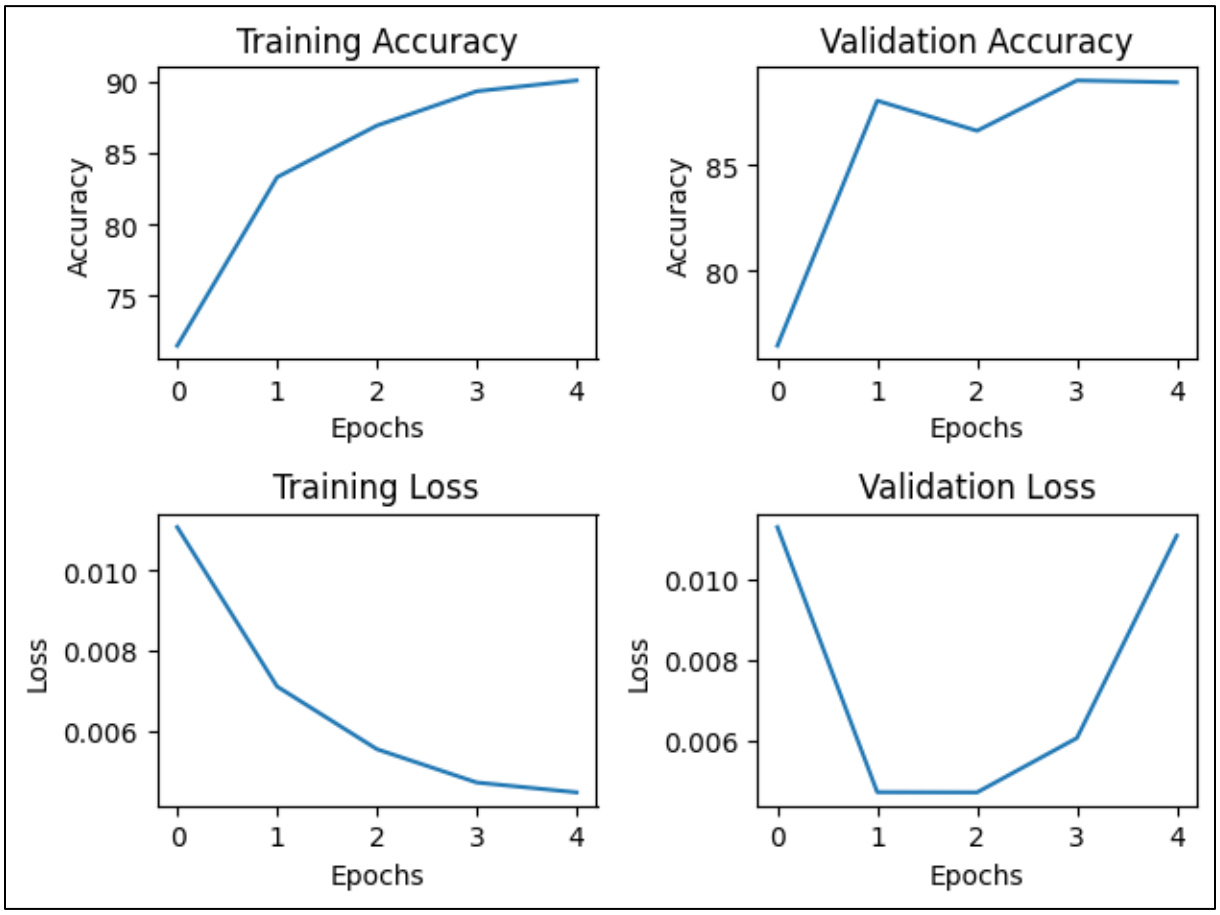
```
[51] print(f"Training Accuracy: {train_accuracy*100:.4f}, Training Loss: {LOSS:.4f}")
     print(f"Validation Accuracy: {validation_accuracy*100:.4f}, Validation Loss: {validation_loss:.4f}")
     print(f"Testing Accuracy: {test_accuracy*100:.4f}, Testing Loss: {test_loss:.4f}")

     Training Accuracy: 87.7952, Training Loss: 0.0050
     Validation Accuracy: 87.1333, Validation Loss: 0.0069
     Testing Accuracy: 87.4667, Testing Loss: 0.3422
```

b. Plot the training and validation accuracy over time (epochs).

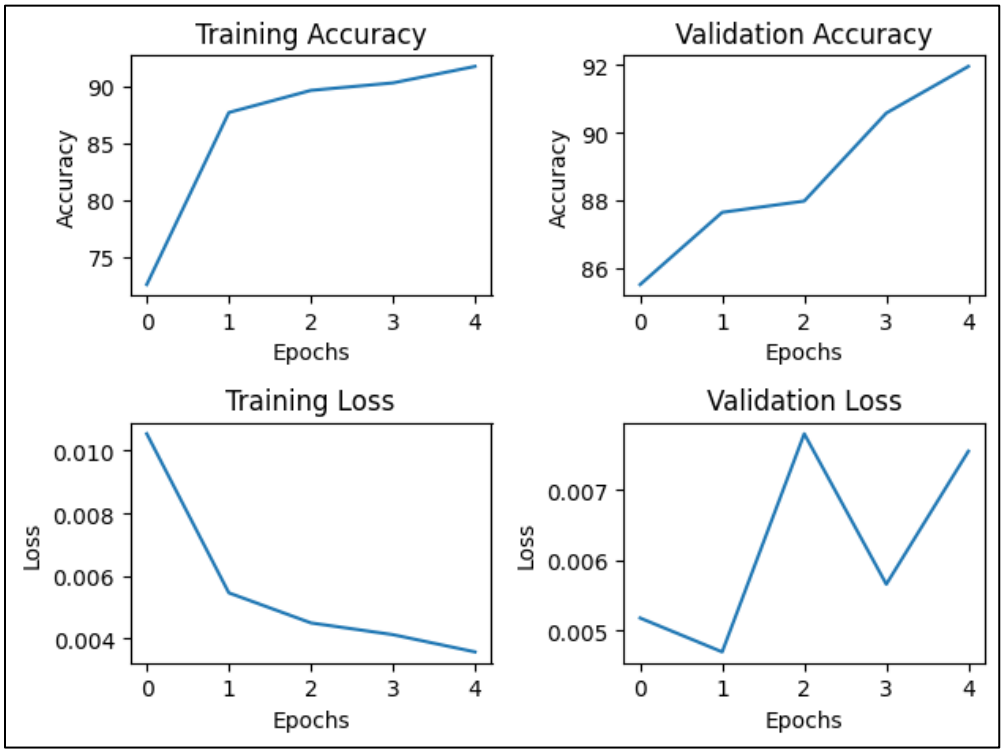
c. Plot the training and validation loss over time (epochs).

Plots for part b) and c):

Training Accuracy, Validation Accuracy and Training Loss, Validation Loss over time (epochs) for the **base model**:

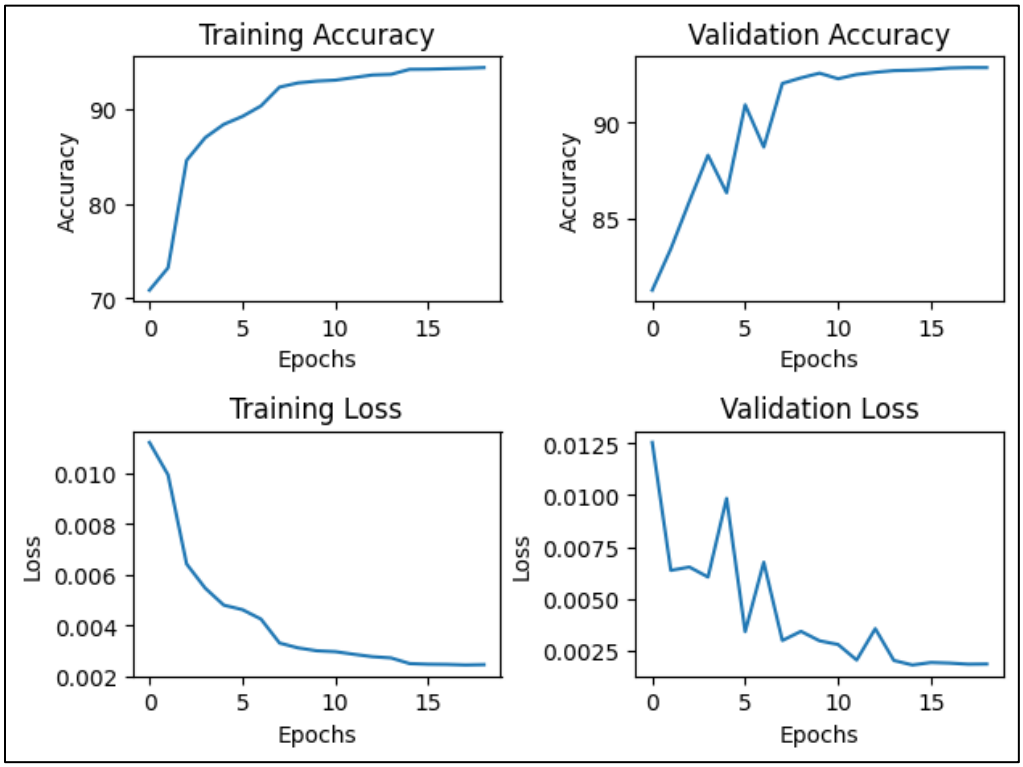


Training Accuracy, Validation Accuracy and Training Loss, Validation Loss over time (epochs) for the **(base + regularization) model**:

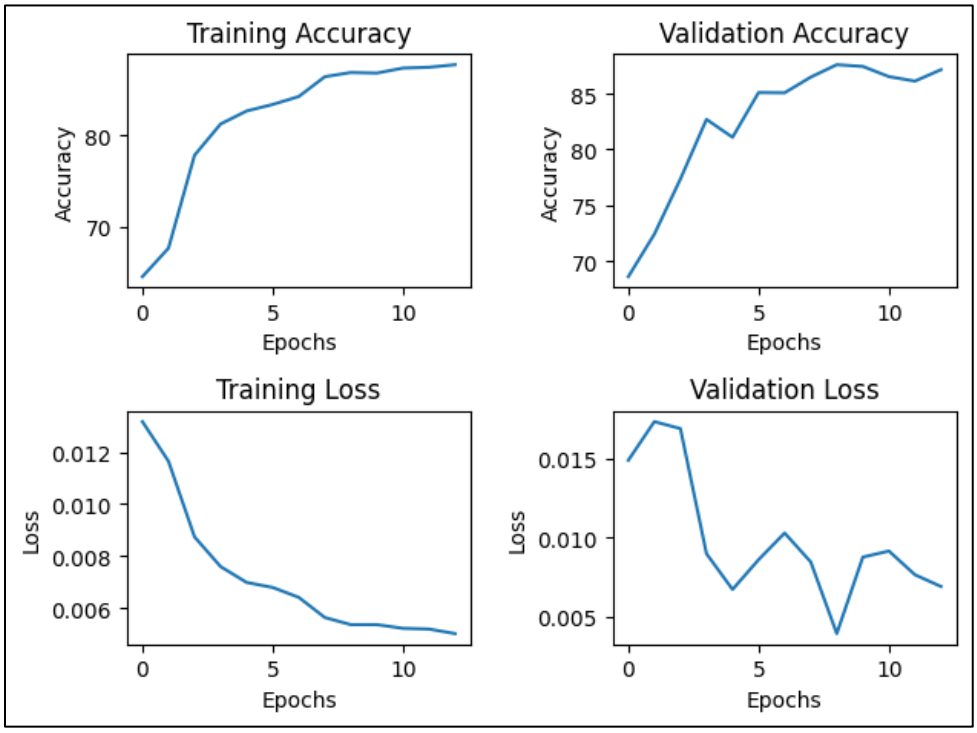


Training Accuracy, Validation Accuracy and Training Loss, Validation Loss over time (epochs) for the **(base + regularization + dropout)** model:

Training Accuracy, Validation Accuracy, and Training Loss, Validation Loss over time (epochs) for the **(base + regularization + dropout + early stopping) model:**



Training Accuracy, Validation Accuracy, and Training Loss, Validation Loss over time (epochs) for the **(base + regularization + dropout + early stopping + image augmentation) model:**



d. Generate a confusion matrix using the model's predictions on the test set.

Confusion Matrix using the **base model**'s predictions on the test set:

```
Confusion Matrix:
 [[1264   69   58]
 [ 195 1387  118]
 [  43   15 1351]]
```

Confusion Matrix using the **(base + regularization) model**'s predictions on the test set:

```
Confusion Matrix:
 [[1393  120   54]
 [  71 1327   53]
 [  38   24 1420]]
```

Confusion Matrix using the **(base + regularization + dropout) model**'s predictions on the test set:

```
Confusion Matrix:
 [[1314   80   57]
 [ 154 1373   69]
 [  34   18 1401]]
```

Confusion Matrix using the **(base + regularization + dropout + early stopping) model**'s predictions on the test set:

```
Confusion Matrix:
 [[1398  102   53]
 [  59 1342   28]
 [  45   27 1446]]
```

Confusion Matrix using the **(base + regularization + dropout + early stopping + image augmentation) model**'s predictions on the test set:

```
Confusion Matrix:
 [[1331  168   81]
 [ 111 1225   48]
 [  95   61 1380]]
```

e. Report any other evaluation metrics used to analyze the model's performance on the test set.

Evaluation metrics of the **(base)** model:

```
[ ]  # Evaluation
     accuracy_test = accuracy_score(test_predicted_labels, test_true_labels) * 100
     precision_test = precision_score(test_predicted_labels, test_true_labels, average='micro')
     recall_test = recall_score(test_predicted_labels, test_true_labels, average='macro')
     f1_test = f1_score(test_predicted_labels, test_true_labels, average='macro')
     confusion_matrix_test = confusion_matrix(test_predicted_labels, test_true_labels)

     print(f'Accuracy: {accuracy_test:.3f}')
     print(f'Precision: {precision_test:.3f}')
     print(f'Recall: {recall_test:.3f}')
     print(f'F1 score: {f1_test:.3f}')
     print('\nConfusion Matrix: \n', confusion_matrix_test)
```

```
Accuracy: 88.933
Precision: 0.889
Recall: 0.894
F1 score: 0.890

Confusion Matrix:
 [[1264   69   58]
 [ 195 1387  118]
 [  43   15 1351]]
```

Evaluation metrics of the **(base + regularization)** model:

```
[ ]  # Evaluation
     accuracy_test = accuracy_score(test_predicted_labels, test_true_labels) * 100
     precision_test = precision_score(test_predicted_labels, test_true_labels, average='micro')
     recall_test = recall_score(test_predicted_labels, test_true_labels, average='macro')
     f1_test = f1_score(test_predicted_labels, test_true_labels, average='macro')
     confusion_matrix_test = confusion_matrix(test_predicted_labels, test_true_labels)

     print(f'Accuracy: {accuracy_test:.3f}')
     print(f'Precision: {precision_test:.3f}')
     print(f'Recall: {recall_test:.3f}')
     print(f'F1 score: {f1_test:.3f}')
     print('\nConfusion Matrix: \n', confusion_matrix_test)
```

```
Accuracy: 92.000
Precision: 0.920
Recall: 0.921
F1 score: 0.920

Confusion Matrix:
 [[1393  120   54]
 [  71 1327   53]
 [  38   24 1420]]
```

Evaluation metrics of the **(base + regularization + dropout)** model:

```
[ ]  # Evaluation
     accuracy_test = accuracy_score(test_predicted_labels, test_true_labels) * 100
     precision_test = precision_score(test_predicted_labels, test_true_labels, average='micro')
     recall_test = recall_score(test_predicted_labels, test_true_labels, average='macro')
     f1_test = f1_score(test_predicted_labels, test_true_labels, average='macro')
     confusion_matrix_test = confusion_matrix(test_predicted_labels, test_true_labels)

     print(f'Accuracy: {accuracy_test:.3f}')
     print(f'Precision: {precision_test:.3f}')
     print(f'Recall: {recall_test:.3f}')
     print(f'F1 score: {f1_test:.3f}')
     print('\nConfusion Matrix: \n', confusion_matrix_test)
```

```
Accuracy: 90.844
Precision: 0.908
Recall: 0.910
F1 score: 0.909

Confusion Matrix:
 [[1314   80   57]
 [ 154 1373   69]
 [  34   18 1401]]
```

Evaluation metrics of the **(base + regularization + dropout + early stopping)** model:

```
[ ]  # Evaluation
     accuracy_test = accuracy_score(test_predicted_labels, test_true_labels) * 100
     precision_test = precision_score(test_predicted_labels, test_true_labels, average='micro')
     recall_test = recall_score(test_predicted_labels, test_true_labels, average='macro')
     f1_test = f1_score(test_predicted_labels, test_true_labels, average='macro')
     confusion_matrix_test = confusion_matrix(test_predicted_labels, test_true_labels)

     print(f'Accuracy: {accuracy_test:.3f}')
     print(f'Precision: {precision_test:.3f}')
     print(f'Recall: {recall_test:.3f}')
     print(f'F1 score: {f1_test:.3f}')
     print('\nConfusion Matrix: \n', confusion_matrix_test)
```

```
Accuracy: 93.022
Precision: 0.930
Recall: 0.931
F1 score: 0.930

Confusion Matrix:
 [[1398  102   53]
 [  59 1342   28]
 [  45   27 1446]]
```

Evaluation metrics of the **(base + regularization + dropout + early stopping + image augmentation)** model:

```python
# Evaluation
accuracy_test = accuracy_score(test_predicted_labels, test_true_labels) * 100
precision_test = precision_score(test_predicted_labels, test_true_labels, average='micro')
recall_test = recall_score(test_predicted_labels, test_true_labels, average='macro')
f1_test = f1_score(test_predicted_labels, test_true_labels, average='macro')
confusion_matrix_test = confusion_matrix(test_predicted_labels, test_true_labels)

print(f'Accuracy: {accuracy_test:.3f}')
print(f'Precision: {precision_test:.3f}')
print(f'Recall: {recall_test:.3f}')
print(f'F1 score: {f1_test:.3f}')
print('\nConfusion Matrix: \n', confusion_matrix_test)
```

```
Accuracy: 87.467
Precision: 0.875
Recall: 0.875
F1 score: 0.875

Confusion Matrix:
 [[1331  168   81]
 [ 111 1225   48]
 [  95   61 1380]]
```