Exercise 4

Omri Ben Hemo 313255242

Ben Ganon 318731007

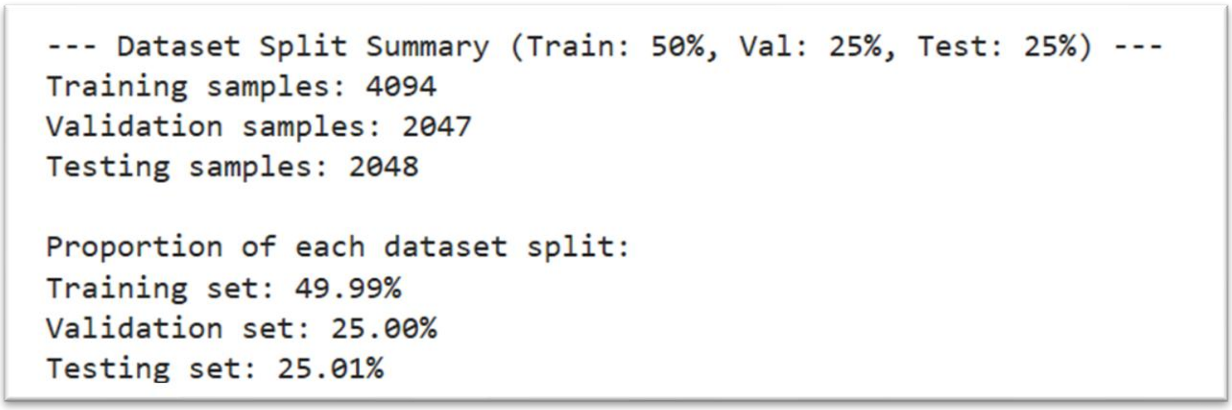
Two pre-trained models, **YOLOv5** and **VGG19**, were adapted to classify flower images into their respective categories.

The goal was to achieve a minimum test accuracy of 70%.

Dataset The primary dataset used for training and evaluation was the 102 Category Flower Dataset from Oxford University

The dataset was randomly divided as follows:

- Training Set: 50%
- Validation Set: 25%
- Test Set: 25%

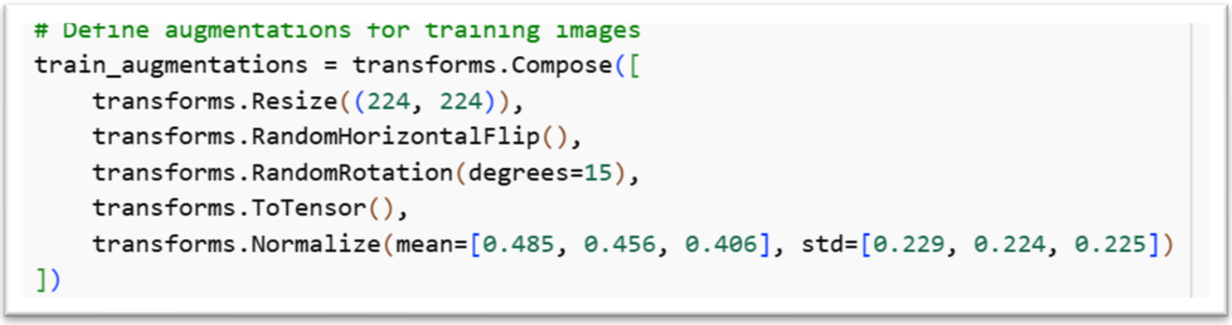


```
--- Dataset Split Summary (Train: 50%, Val: 25%, Test: 25%) ---
Training samples: 4094
Validation samples: 2047
Testing samples: 2048

Proportion of each dataset split:
Training set: 49.99%
Validation set: 25.00%
Testing set: 25.01%
```

This random split was repeated twice to ensure generalizability.

Preprocessing Steps To prepare the dataset for training, the following preprocessing steps were performed:

- Image resizing to match the input dimensions required by YOLOv5 and VGG19.
- Normalization of pixel values to the range [0,1].
- Data augmentation techniques such as rotation, flipping, and normalization were applied.
- Label encoding for categorical classification.



```
# Define augmentations for training images
train_augmentations = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(degrees=15),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Model Selection and Adaptation Both YOLOv5 and VGG19 were fine-tuned to classify flower images:

VGG19

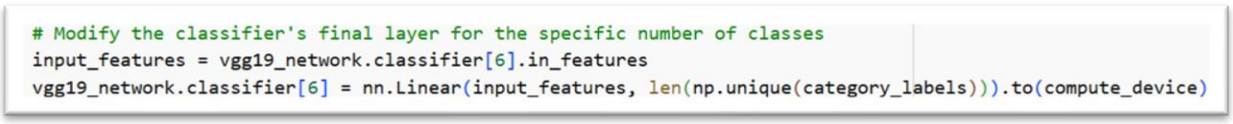
In the VGG19 model, we modified the final fully connected layer in the classifier.

Originally, this layer was designed for 1000 classes (ImageNet classification).

We replaced it with a new nn.Linear layer that has:

- The same input features as the original classifier layer.
- An output size equal to the number of unique category labels in our dataset.

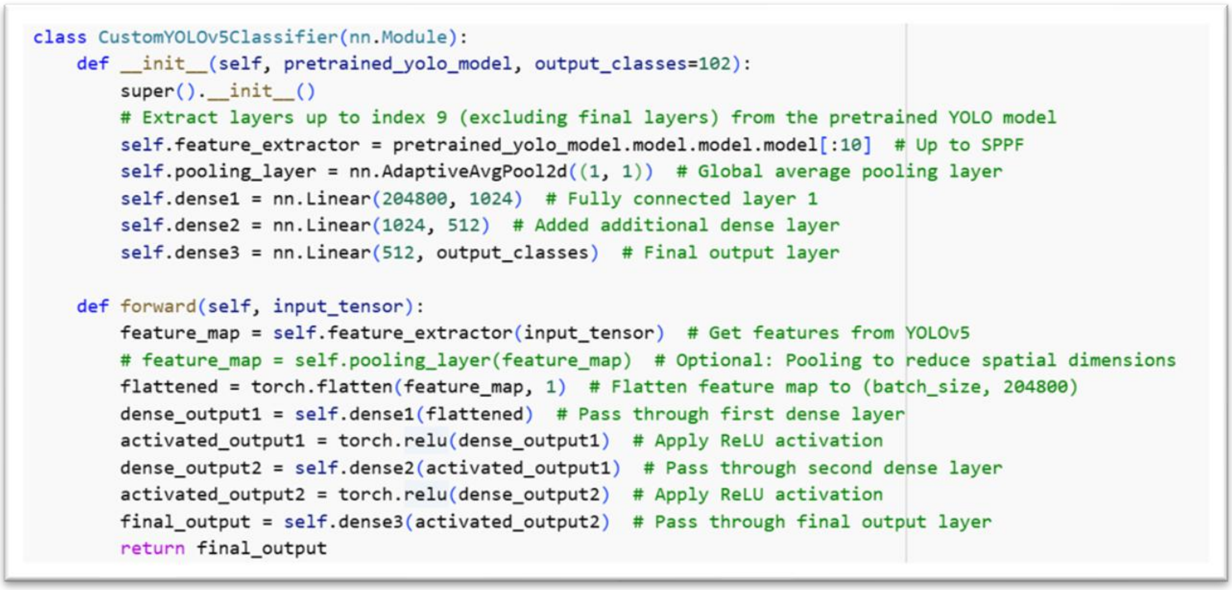
This modification allows VGG19 to be fine-tuned for a custom classification task with a different number of output classes.



```
# Modify the classifier's final layer for the specific number of classes
input_features = vgg19_network.classifier[6].in_features
vgg19_network.classifier[6] = nn.Linear(input_features, len(np.unique(category_labels))).to(compute_device)
```

YOLOv5

- We used the YOLOv5 model as a feature extractor by retaining layers up to index 9 (including the SPPF layer) and removing the detection-specific layers.
- Added an optional AdaptiveAvgPool2d layer to reduce the spatial dimensions of the feature map, making it compatible with fully connected layers.
- Dense Layers:
  - Added a fully connected layer (Linear) with 204800 inputs and 1024 outputs.
  - Applied ReLU activation after the first dense layer for non-linearity.
  - Added a second dense layer with 1024 inputs and 512 outputs, followed by ReLU activation.
  - Added a final dense layer to map the features to the desired number of output classes.
- This structure transforms YOLOv5 into a custom classifier tailored for a multi-class classification task.



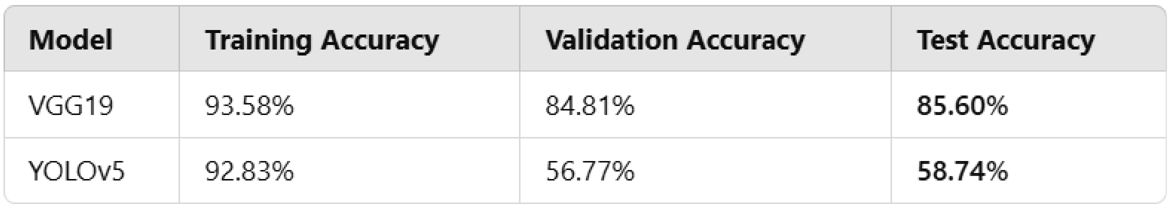
```python
class CustomYOLOv5Classifier(nn.Module):
    def __init__(self, pretrained_yolo_model, output_classes=102):
        super().__init__()
        # Extract layers up to index 9 (excluding final layers) from the pretrained YOLO model
        self.feature_extractor = pretrained_yolo_model.model.model.model[:10]  # Up to SPPF
        self.pooling_layer = nn.AdaptiveAvgPool2d((1, 1))  # Global average pooling layer
        self.dense1 = nn.Linear(204800, 1024)  # Fully connected layer 1
        self.dense2 = nn.Linear(1024, 512)  # Added additional dense layer
        self.dense3 = nn.Linear(512, output_classes)  # Final output layer

    def forward(self, input_tensor):
        feature_map = self.feature_extractor(input_tensor)  # Get features from YOLOv5
        # feature_map = self.pooling_layer(feature_map)  # Optional: Pooling to reduce spatial dimensions
        flattened = torch.flatten(feature_map, 1)  # Flatten feature map to (batch_size, 204800)
        dense_output1 = self.dense1(flattened)  # Pass through first dense layer
        activated_output1 = torch.relu(dense_output1)  # Apply ReLU activation
        dense_output2 = self.dense2(activated_output1)  # Pass through second dense layer
        activated_output2 = torch.relu(dense_output2)  # Apply ReLU activation
        final_output = self.dense3(activated_output2)  # Pass through final output layer
        return final_output
```

Training Process

- Optimizer: Adam optimizer was used with a learning rate of 0.001.
- Loss Function: Categorical Cross-Entropy.
- Number of Epochs: 5.
- Performance was monitored on validation loss.

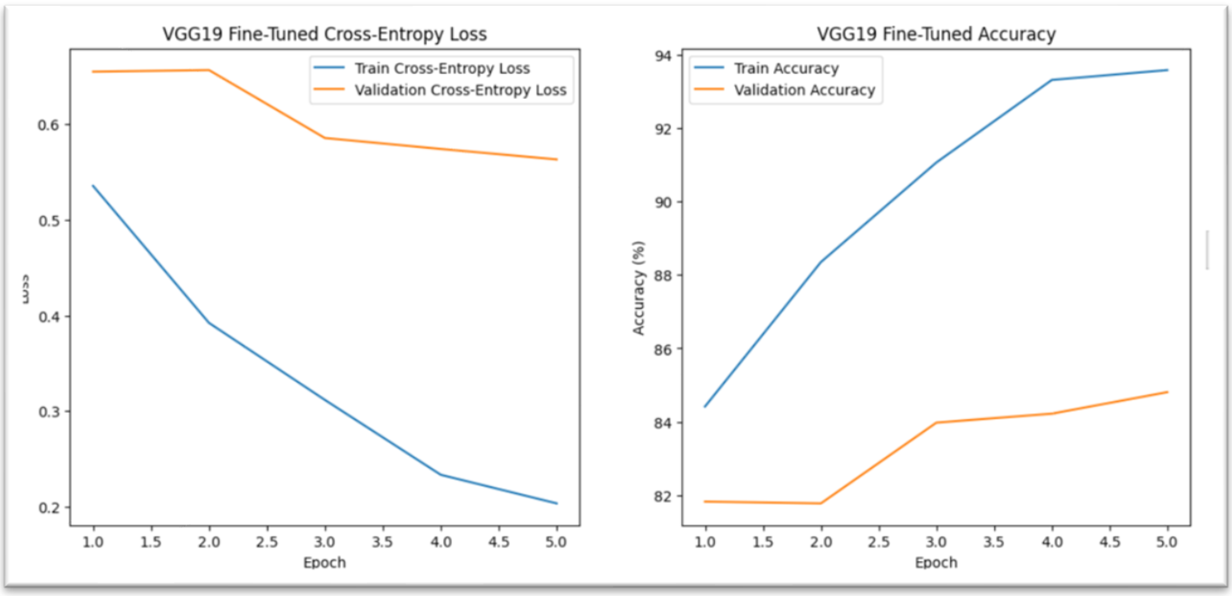
Performance Metrics

- Accuracy
- Cross-Entropy Loss
- Training and Validation Loss vs. Epochs



Model	Training Accuracy	Validation Accuracy	Test Accuracy
VGG19	93.58%	84.81%	**85.60%**
YOLOv5	92.83%	56.77%	**58.74%**

VGG19

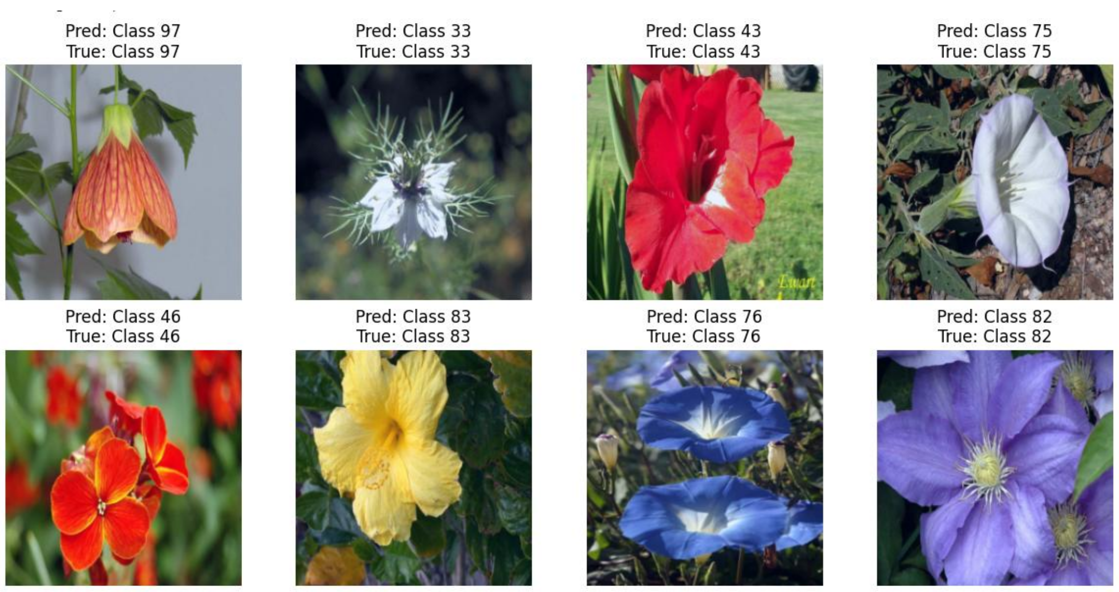


Left Graph: Cross-Entropy Loss

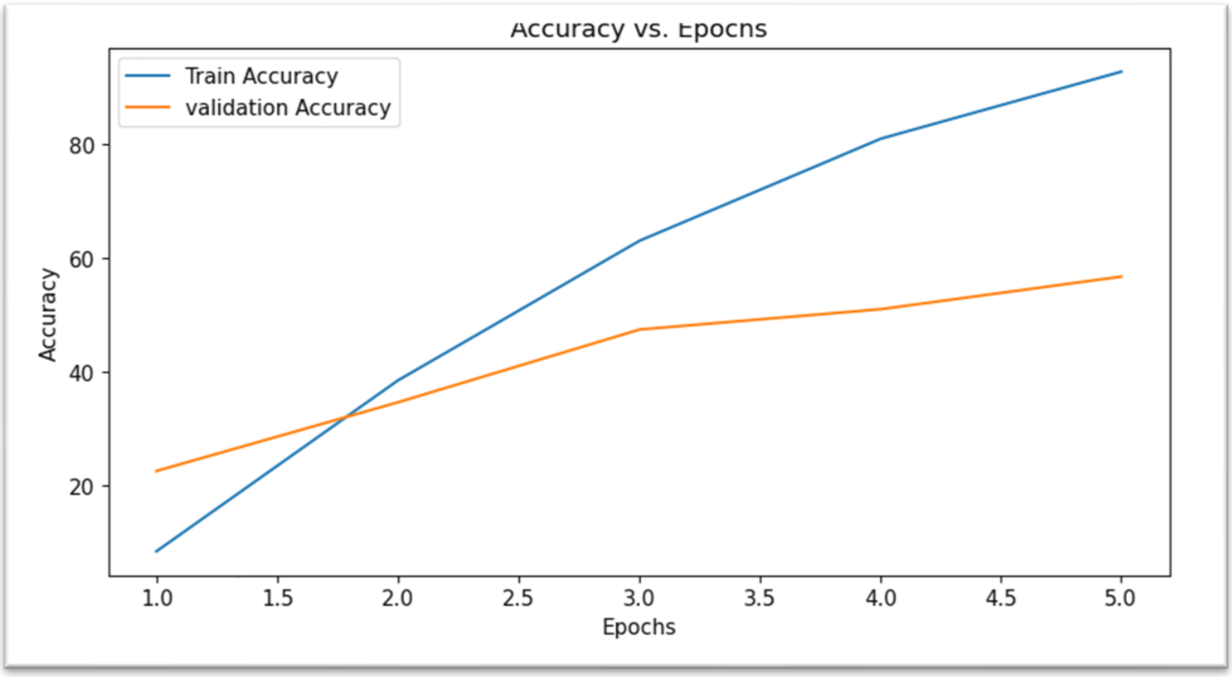
- **Observation**:
  - The training loss decreases consistently over the epochs, indicating that the model is learning from the training data effectively.
  - The validation loss decreases as well but at a slower rate, showing some generalization to the validation set.
- **Insight**:
  - The gap between the training and validation loss is widening slightly, which could be an early sign of overfitting. The training performance improves faster than validation performance.

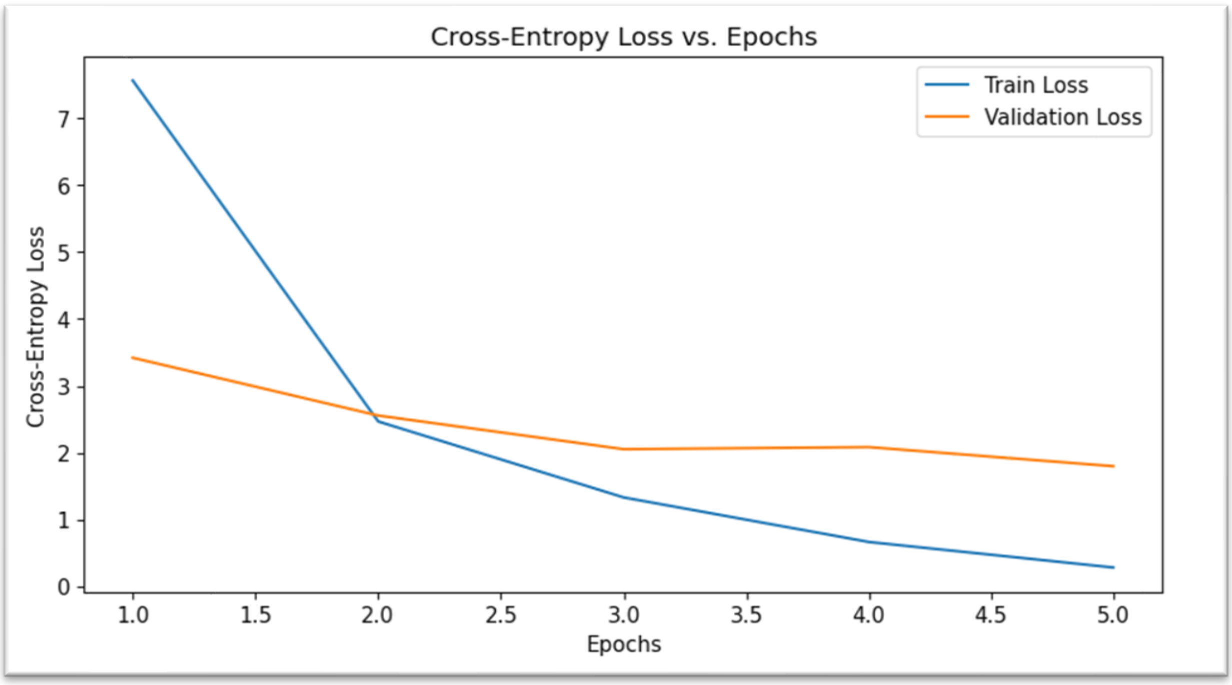
Right Graph: Accuracy

- **Observation**:
  - The training accuracy increases steadily, reaching a very high level (~94%) by the fifth epoch.
  - Validation accuracy also improves over time, albeit more slowly, leveling off around 85%.
- **Insight**:
  - The training accuracy being significantly higher than validation accuracy suggests a possible overfitting issue, where the model is performing better on training data than on unseen validation data.


Pred: Class 97
True: Class 97
Pred: Class 33
True: Class 33
Pred: Class 43
True: Class 43
Pred: Class 75
True: Class 75
Pred: Class 46
True: Class 46
Pred: Class 83
True: Class 83
Pred: Class 76
True: Class 76
Pred: Class 82
True: Class 82

YOLOv5


Accuracy vs. Epochs
Train Accuracy
validation Accuracy
Accuracy
Epochs
20
40
60
80
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

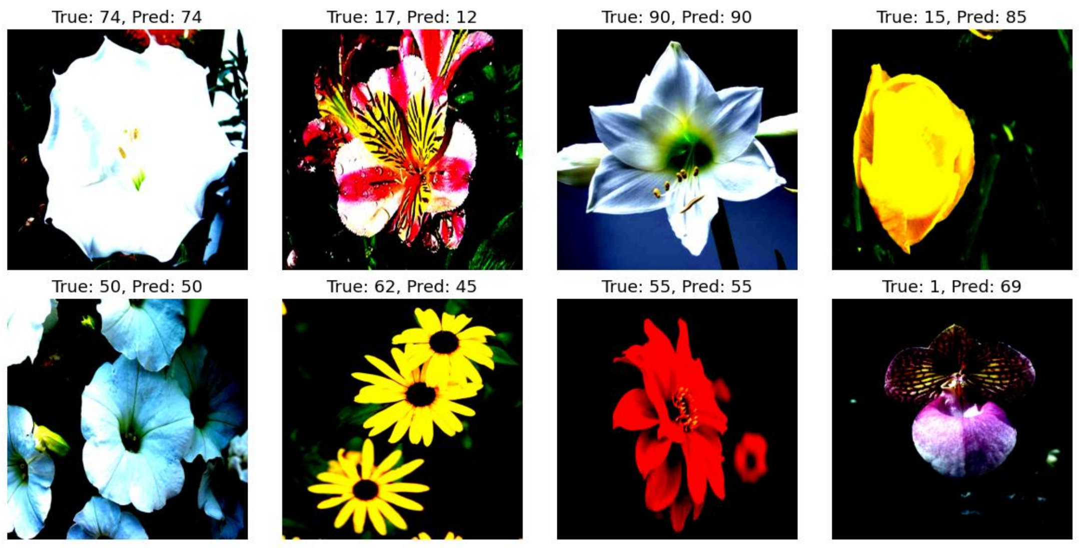


Top Graph: Accuracy vs. Epochs

- **Observation**:
  - **Train Accuracy** increases steadily with epochs, indicating that the model is learning from the training data effectively.
  - **Validation Accuracy** also improves but at a slower rate, with a noticeable gap between training and validation accuracy by the fifth epoch.
- **Insight**:
  - The gap between training and validation accuracy suggests that the model might be overfitting to the training data as it performs significantly better on training data than on unseen validation data.

Bottom Graph: Cross-Entropy Loss vs. Epochs

- **Observation**:
  - **Train Loss** decreases sharply in the initial epochs and continues to decline steadily, showing that the model is optimizing well for the training data.
  - **Validation Loss** decreases initially but plateaus after the second epoch, which could indicate that the model's generalization to the validation data is not improving much further.
- **Insight**:
  - The plateau in validation loss, coupled with the growing gap between training and validation metrics, indicates potential overfitting. The model is likely learning specific patterns in the training set that do not generalize to unseen data.



Results and Discussion

- The VGG19 model achieved a test accuracy of 85%, surpassing the required 70%.
- The YOLOv5 model performed slightly lower, around 58%, due to its primary focus on object detection rather than classification.
- The accuracy and loss graphs show a steady convergence, indicating stable training.
- Additional dataset augmentation could further improve accuracy.

Conclusion

This exercise demonstrated the power of Transfer Learning in image classification tasks. VGG19, with its deep feature extraction capabilities, outperformed YOLOv5 for this classification problem. Future improvements could include using additional pre-trained models or fine-tuning hyperparameters for better results.