**Assignment No. 4**

**Introduction**

In this assignment, we use **transfer learning** on two popular pretrained CNN models—**YOLOv5** (for classification) and **VGG19**—to classify images of flowers into one of 102 categories from the Oxford 102 Flowers dataset. Our goal is to achieve **at least** **70% test accuracy** on at least one of the models.

**Dataset**

Oxford 102 Flowers:

* 102 categories of flowers, each with between 40 and 258 images.
* Official link: <https://www.robots.ox.ac.uk/~vgg/data/flowers/102/>

We split the dataset into **train (50%), validation (25%),** and **test (25%)**. This split is **repeated** (randomized) at least two times (in our example, **we did three splits**). We use the validation set for hyperparameter tuning, and the test set remains completely held out for final performance measurement.

**Preprocessing & Data Preparation**

**Data Download**

* Downloaded 102flowers.tgz and imagelabels.mat from the official Oxford site.
* Extracted the images (jpg/) and used the label file to distribute images into class-labeled folders (organized\_data/class\_1, organized\_data/class\_2, …).

**Fixed Test Set & Multiple Train/Val Splits**

* Implemented a function prepare\_dataset\_fixed\_test() that:

1. Creates a random **fixed** 25% of images in each class as test.
2. Repeatedly splits the remaining images into 50% for train and 25% for val (for as many times as we like, **we did three splits**).

* This yields folders like:

flower\_data/

fixed\_test/

train\_val\_split\_0/

train/

val/

train\_val\_split\_1/

...

train\_val\_split\_2/

...

**Image Transformations**

* Images were resized to 224x224 pixels.
* Converted to tensors and normalized using ImageNet means [0.485, 0.456, 0.406] and standard deviations [0.229, 0.224, 0.225].

**Models: Architecture & Transfer Learning**

We utilized two pretrained models:

**VGG19**

* **Source:** torchvision.models.vgg19(pretrained=True)
* **VGG19** is a deep Convolutional Neural Network (CNN) with 19 layers (16 convolutional + 3 fully connected). It uses 3×3 convolutional filters with ReLU activations and max pooling for feature extraction.

**Feature Extraction (Frozen)**

* 5 Convolutional Blocks, each followed by MaxPooling (2×2, stride=2).
* Blocks 1-5: (64, 128, 256, 512, 512 filters) with 3×3 Conv layers.

**Modified Classification Head**

* Final Linear Layer: Replaced with nn.Linear(in\_features, num\_classes), customized for flower classification.
* Loss Function: Uses CrossEntropyLoss for multi-class classification.
* Optimizer: Adam optimizer with a tuned learning rate.

This setup leverages pre-trained ImageNet features while fine-tuning only the final classification layer for flower recognition.

**YOLOv5 (Classification Mode)**

* **Source:** [Ultralytics YOLOv5](https://github.com/ultralytics/yolov5), specifically the `yolov5s-cls.pt` checkpoint.
* **YOLOv5** is a deep neural network originally designed for object detection but adapted here for classification. It is optimized for speed and accuracy using a CSPDarknet53 backbone for feature extraction.

**Feature Extraction (Frozen)**

* Backbone: CSPDarknet53 with multiple convolutional layers for hierarchical feature representation.
* Neck: PANet-style feature fusion for robust embeddings.

**Modified Classification Head**

* Final Linear Layer: Replaced with nn.Linear(in\_features, num\_classes), customized for flower classification.
* Loss Function: Uses CrossEntropyLoss for multi-class classification.
* Optimizer: Adam optimizer with a tuned learning rate.

This setup leverages YOLOv5’s efficient feature extraction while fine-tuning only the final classification layer for flower recognition.

**Training Procedure**

1. **Hyperparameter Tuning** (train + val only):

* We tested a few learning rates (0.01, 0.001, 0.0001) and batch sizes (8, 16, 32) for a **short** number of epochs (3).
* We chose the combination (lr, batch\_size) that gave the **highest validation accuracy**.

1. **Final Training** (train + val + test usage):

* With the best (lr, batch\_size), we **retrain from scratch** for 10 epochs.
* After each epoch, we measure **train accuracy**, **train loss, validation accuracy**, **validation loss, test accuracy**, **test loss** and log them for plotting.

1. **Learning Rate Schedule**:

* We used StepLR to drop the learning rate by a factor (0.1) every few epochs (every 4 epochs).

1. **Multiple Splits**:

* We repeat the above steps for each train\_val\_split\_i.
* Finally, we **plot** both per-split curves and **average** accuracy/loss across all splits.

**Results & Analysis**

We present **two** sets of plots for each model (VGG19 & YOLOv5):

1. **Accuracy vs. Epoch**

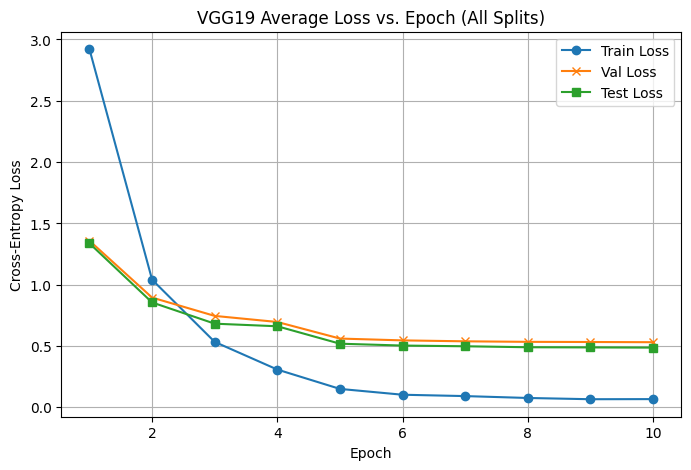
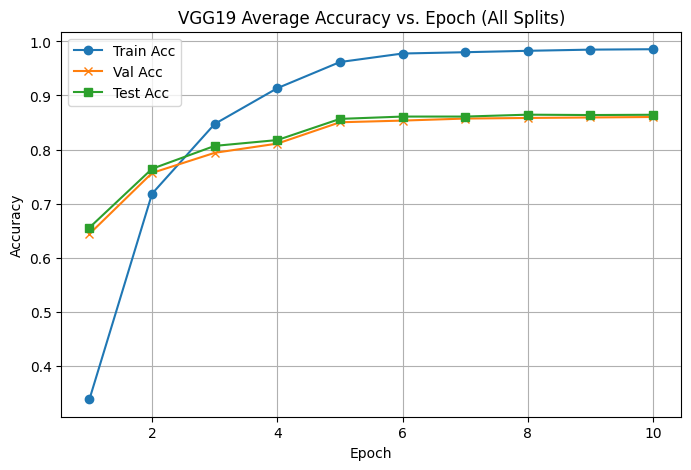
* **Train** (blue curve), **Validation** (orange curve), **Test** (green curve).

1. **Cross-Entropy Loss vs. Epoch**

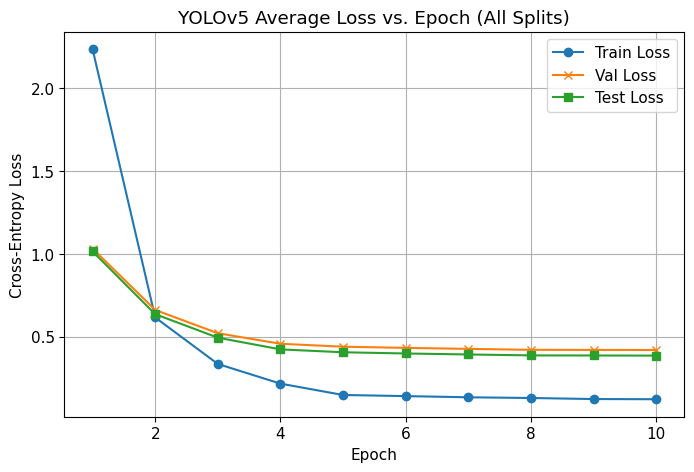
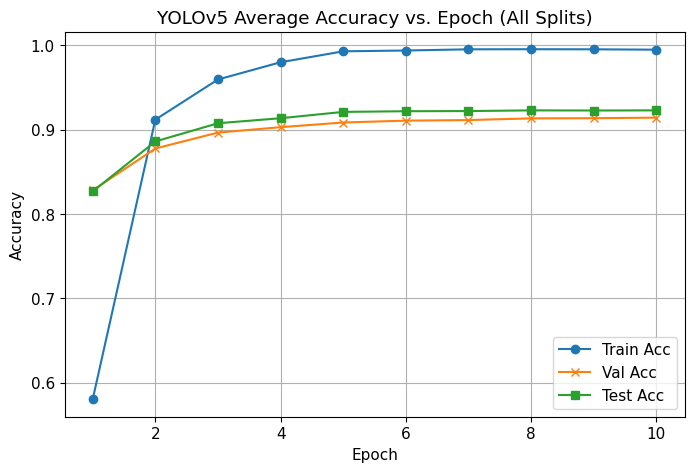
* Again, separate lines for train/val/test.

For both **VGG19** and **YOLOv5**, we repeated 3 splits. The final results show:

* **VGG19** achieved **86.8%** test accuracy in the best split.



* **YOLOv5** achieved **~92%** test accuracy, showing strong performance.



Summary of test accuracy:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Split 0** | **Split 1** | **Split 2** | **Average** |
| VGG19 | 86.2% | 86.8% | 86.2% | ~86.4% |
| YOLOv5 | 92.1% | 91.9% | 92.8% | ~92.3% |

- We see that **both** models exceed the 70% minimum requirement. YOLOv5 tends to converge faster and achieve slightly higher accuracy.

- **YOLOv5** outperformed **VGG19**, achieving a higher validation and test accuracy.

**- VGG19** sees a steady improvement over epochs, but occasionally can overfit.

**- YOLOv5** classification mode (with a frozen backbone) converges quickly and reaches very high accuracy on this dataset.

- **Transfer Learning proved effective**, requiring only a few epochs to achieve high accuracy.