Ben-Gurion University of the Negev

Machine Learning Course

Assignment 4 – CNN Flower Classifiers

Lecturer: Dr Lior Rokah

Students: Jacob Yaacubov 312028970

Barak Milshtein 209309954

1. Introduction

This report presents the 4th assignment of our machine learning course. This is the Implementation and evaluation of two Convolutional Neural Network (CNN) architectures for flower classification using transfer learning. The objective is to classify images from the Oxford 102 Flowers dataset into 102 different flower categories.

Now short intro to Transfer learning, which is a machine learning technique where a model trained on a large dataset (such as ImageNet) is adapted to a new, related task. This approach is particularly effective when the target dataset is relatively small, as it leverages the learned features from the source domain. Usually, we try to use transfer learning on similar tasks; a model good at detecting features might be a good candidate for transfer learning for detection, segmentation, etc.

As requested, two architectures were evaluated:

- VGG19: A deep convolutional network known for its simplicity and effectiveness, which utilizes depth as its strength
- YOLOv5: A modern, efficient architecture used for object detection but has a classification version.

2. Dataset

The Oxford 102 Flowers dataset contains 8,189 images of flowers belonging to 102 different categories. 2.1 Data Split

The dataset was split as follows:

- Training set: 50% (4,094 images)
- Validation set: 25% (2,047 images)
- Test set: 25% (2,048 images)

Two random splits were created using different random seeds (42 and 123) to analyze the effect of data distribution on model performance.

3. Image Preprocessing

Image preprocessing is crucial for CNN training. The following preprocessing steps were created:

All classification models, VGG & YoloV,5 use 224x224 images. For learning the features and deal with large variance in picture orientation and other possible cases we augmented the data with simple rotation and flip.

3.1 Preprocessing for VGG19

Step	Description
Resize	Resize all images to 224x224 pixels
ToTensor	Convert PIL images to PyTorch tensors and scale pixel values to [0, 1]
Normalize	Normalize using ImageNet mean [0.485, 0.456, 0.406] and std [0.229, 0.224, 0.225]

Data Augmentation (Training Only)

- RandomHorizontalFlip: 50% probability of horizontal flip
- RandomRotation: Random rotation up to 10 degrees

3.2 Preprocessing for YOLOv5

Step	Description
Resize	Resize all images to 224x224 pixels
ToTensor	Convert PIL images to PyTorch tensors and scale pixel values to [0, 1]
Normalize	Normalize using ImageNet mean [0.485, 0.456, 0.406] and std [0.229, 0.224, 0.225]

Same data augmentation techniques (RandomHorizontalFlip and RandomRotation) were applied during training.

4. Network Architectures

4.1 VGG19 Architecture

VGG19 is a deep convolutional neural network developed by the Visual Geometry Group at Oxford. It consists of 19 layers with learnable weights (16 convolutional layers and 3 fully connected layers).

Original VGG19 Structure

- Features (Convolutional Layers): 16 convolutional layers with 3x3 kernels
- 5 Max Pooling layers (2x2, stride 2)
- Classifier: 3 fully connected layers (4096 -> 4096 -> 1000)

Transfer Learning Modifications

Component	Status	Description
Features (Conv layers)	FROZEN	Pretrained on ImageNet, weights not updated
Classifier [0-5]	TRAINABLE	Fully connected layers fine-tuned
Classifier [6]	REPLACED	New head: Linear(512->102)

Parameter Count

- Total parameters: ~143 million
- Frozen parameters (features): ~20 million
- Trainable parameters: ~123 million

4.2 YOLOv5 Classification Architecture

YOLOv5 is considered a very efficient architecture originally designed for object detection. The classification variant (YOLOv5s-cls) was used for this task, pretrained on ImageNet.

YOLOv5s-cls Structure

- Backbone (Layers 0-8): CSPDarknet with C3 modules for feature extraction
- Layer 9: SPPF (Spatial Pyramid Pooling Fast) + Classification head
- Final layer: Linear(1280 -> 1000) for ImageNet classes

Transfer Learning Modifications

Component	Status	Description
Backbone (Layers 0-9)	FROZEN	Pretrained feature extractors, weights not updated
model.9.linear	REPLACED	Changed from Linear(1280->1000) to Linear(1280->102)

Parameter Count

- Total parameters: ~5.4 million
- Frozen parameters: ~5.3 million
- Trainable parameters: ~130,000 (final linear layer)

5. Training Configuration

5.1 Hyperparameters

Hyperparameter	VGG19	YOLOv5
Learning Rate	0.01 (initial)	0.001
Batch Size	32	64
Epochs	35	20
Optimizer	SGD (momentum=0.9)	Adam
Loss Function	CrossEntropyLoss	CrossEntropyLoss
Image Size	224 x 224	224 x 224
LR Scheduler	ReduceLROnPlateau	None
Weight Decay	1e-4	None

5.2 Optimizer Selection

We used different optimizers for each model based on transfer learning best practices:

VGG19 - SGD with Momentum, SGD with momentum (0.9) was chosen for VGG19 because Adam gave worse results, as well as the article [3] mentions that it often produces better generalization for transfer learning tasks. Combined with a higher initial learning rate (0.01), it allows aggressive exploration before fine-tuning. Weight decay (1e-4) was added for regularization given the large number of trainable parameters

YOLOv5-cls - Adam: Adam optimizer was used for YOLOv5-cls as the default, which work well with the model's smaller number of trainable parameters (~130K). The lower learning rate (0.001) was sufficient since Adam automatically adjusts per-parameter learning rates. So we didn’t change it because it had good enough results.

5.3 Learning Rate Scheduler

The scheduler proved essential for VGG19's performance. During epochs 1-13 with lr=0.01, validation accuracy plateaued around 70%. The scheduler automatically reduced the learning rate to 0.001 at epoch 14, after which validation accuracy jumped from 70% to 85%. This demonstrates the importance of learning rate scheduling — a high initial learning rate allows rapid exploration, while a reduced learning rate enables fine-tuning for optimal accuracy.

5.4 Training Environment

- Framework: PyTorch 2.8.0
- GPU: NVIDIA Tesla T4z (15GB VRAM)
- Data Loading: 4 workers with pin_memory enabled

6. Results

6.1 Test Accuracy Summary

Model / Split	Split 42	Split 123
VGG19	84.23%	85.30%
YOLOv5	91.75%	91.89%
Improvement	7.52% (YOLOV5)	6.59%(YOLOV5)

6.2 Loss Summary

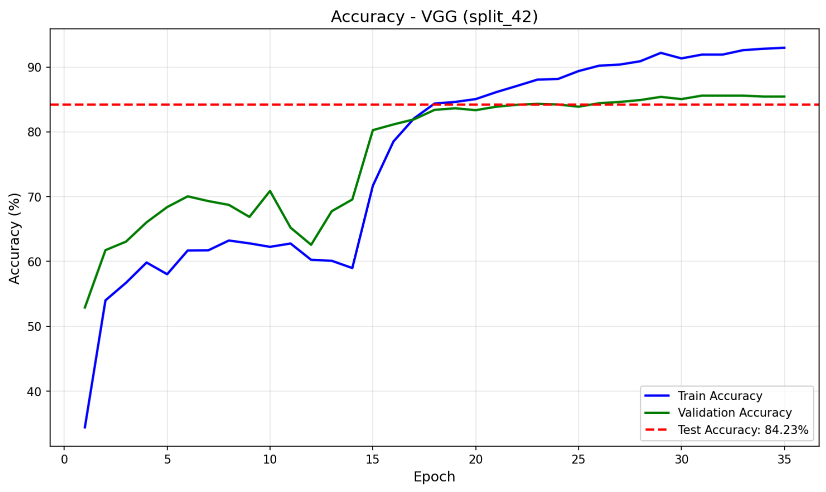
Model / Split	Split 42	Split 123
VGG19	0.6410	0.5751
YOLOv5	0.3169	0.2807
Improvement	-0.3241 (49% lower)	0.2944 (51% lower)

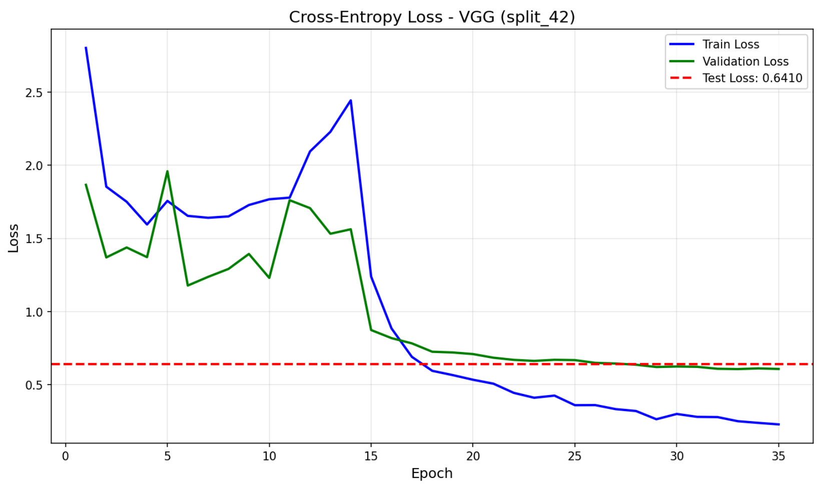
YOLOv5-cls achieved approximately 50% lower cross-entropy loss, A loss of 0.28 means the model assigns high probability to the correct flower class, while VGG19's 0.57-0.64 loss indicates more uncertainty in its predictions.The lower loss reflects better-calibrated predictions and more discriminative feature extraction

6.3 Training Curves

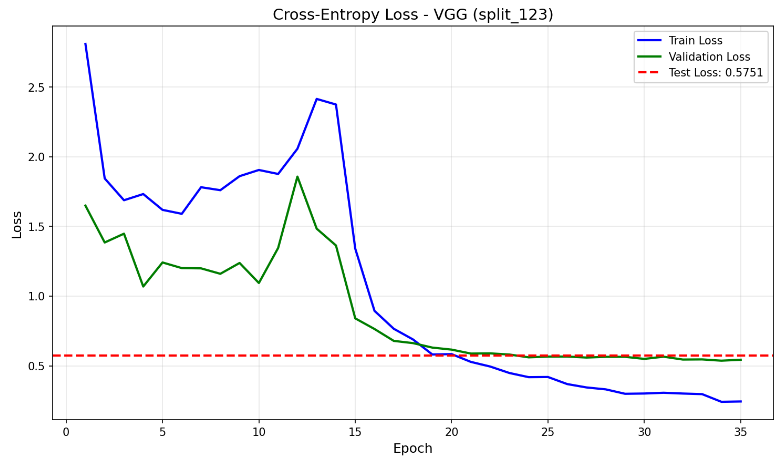
The following graphs show the training and validation accuracy/loss over epochs:

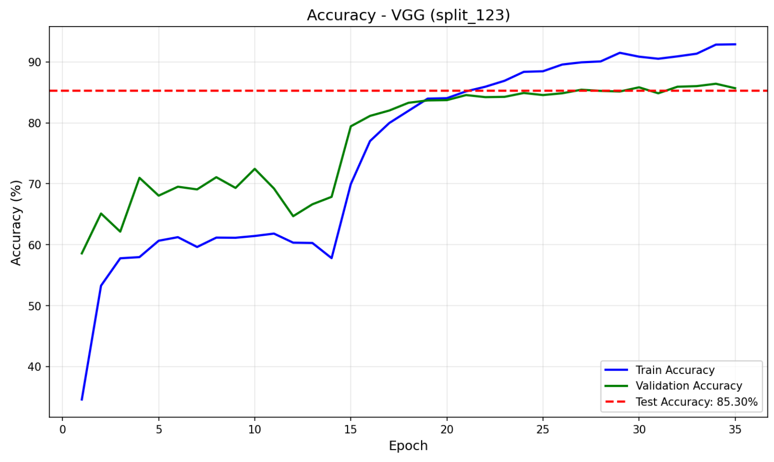
VGG19 - Split 42



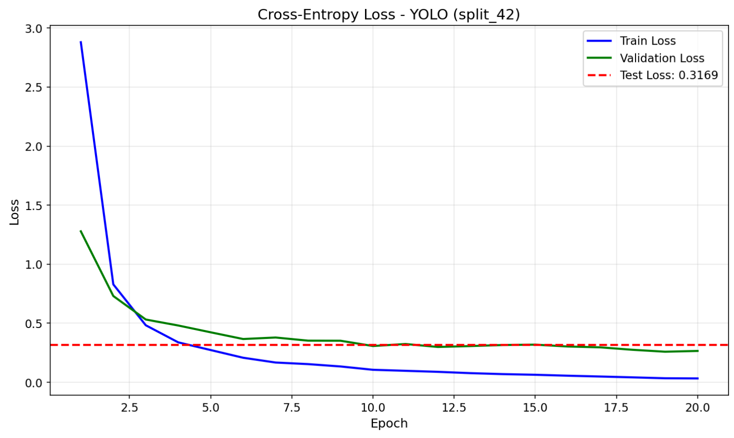


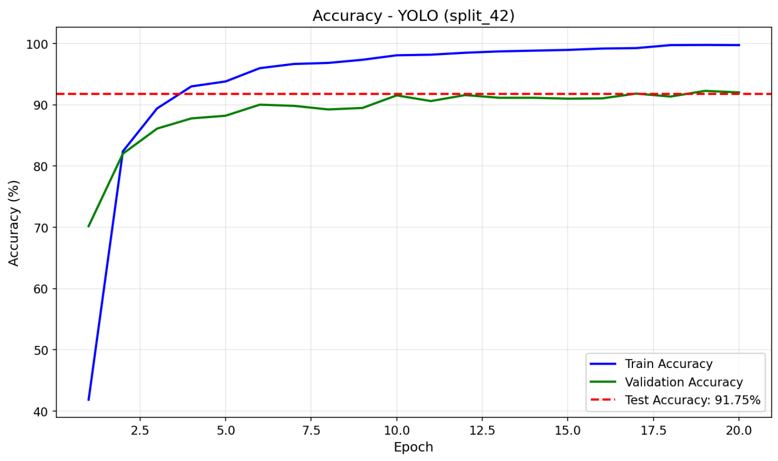
VGG19 - Split 123



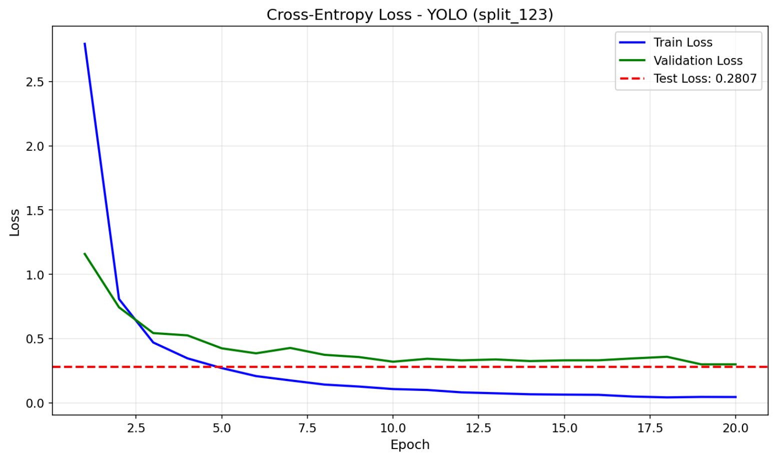


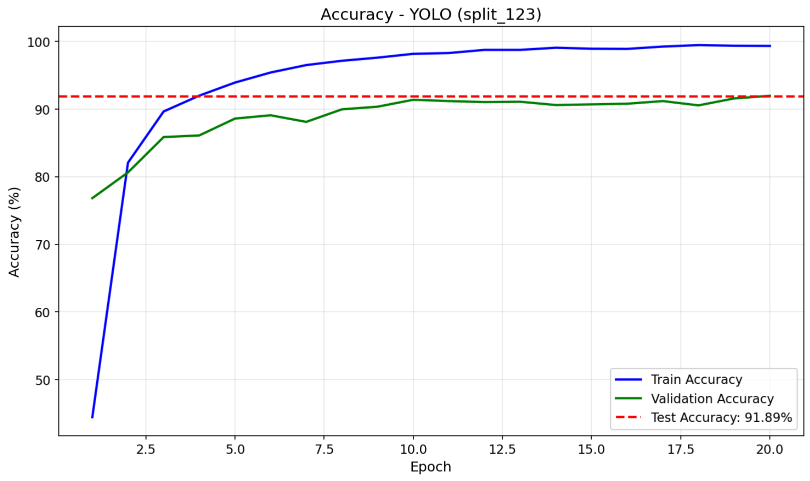
YOLOv5 - Split 42





YOLOv5 - Split 123





7. Analysis and Discussion

7.1 Model Comparison

The two architectures we chose represent different eras of deep learning research. VGG19, developed in 2014 by Oxford's Visual Geometry Group, follows a straightforward "deeper is better" approach — it stacks 19 layers of simple 3x3 convolutions, resulting in a massive 143 million parameters. They used this since a stack of 2 units of 3x3 filters can cover the same receptive field as a 5x5 filter.

Pros:

- Simple, Easy to experiment with

Cons:

- Computationally expensive 140million parameters, takes longer to train

YOLOv5-cls, released in 2020, takes a smarter approach. Instead of just going deeper, it uses modern techniques like CSP (Cross Stage Partial) connections that achieve better feature extraction with only 10 layers and roughly 5 million parameters — nearly 30 times smaller than VGG19.

Pros:

- Fast, lightweight, Small number of parameters, memory efficient.

Cons:

- Complex CSP connections, harder to understand the architecture, a black-box feel.

Additionally, YOLOv5-cls benefits from superior pretrained features, it is trained using modern techniques like advanced augmentation and optimized training recipes that didn't exist when VGG19 was developed. This explains why YOLOv5-cls achieved better accuracy with 1000x fewer trainable parameters.

Our transfer learning strategies reflected these architectural differences. With VGG19, we froze the convolutional feature extractor and fine-tuned the entire 3-layer classifier, which still meant training around 124 million parameters. The final layer was replaced to output 102 flower classes instead of ImageNet's 1000 categories. For YOLOv5-cls, we froze layers 0-8 (the backbone) and only trained the classification head (layer 9), replacing its final linear layer from 1280→102 outputs. This meant training roughly 130 thousand parameters — about 1000 times fewer than VGG19.

The results are quite dramatic: YOLOv5-cls achieved 91.89% test accuracy while VGG19 reached around 85%, demonstrating that modern efficient architectures can outperform older heavyweight models, especially when training data is limited. Sometimes less really is more.

7.2 Effect of Random Seeds

To evaluate model robustness, we trained both architectures on two different data splits using random seeds 42 and 123, each maintaining the same 50/25/25 train/validation/test ratio.

Different seeds might hint at how robust the model, large varying seeds might tell us about instability, high variance if seed difference is high, we want to have a seed difference of 3% and less for good, reliable results. Low sees a difference means that the model learned real patterns (petal shapes, colors, textures) that work regardless of which specific images it trained on .Large variance would suggest overfitting or insufficient training data.

For YOLOv5-cls, seed 42 achieved 91.75% test accuracy while seed 123 achieved 91.89%, a difference of only 0.14%. Similarly, VGG19 showed 84.23% for seed 42 and 85.30%for seed 123, with a 1.07% difference.

Both models show excellent robustness across different data splits. Both are well within the acceptable range of <3%. This indicates that both models learned genuine flower features rather than being sensitive to which specific images ended up in training versus testing.

Model / Split	Seed 42	Seed 123	Differance
VGG19	84.23%	85.30%	1.07%
YOLOv5	91.75%	91.89%	0.14%

7.3 Transfer Learning Benefits

Training a deep neural network from scratch requires millions of images and days of computation, many resources. Transfer learning solves this by starting with models already trained onmillions of images. These pretrained models have already learned universal visual features: edges, textures, shapes, and patterns that apply to almost any image task.

In our experiment, transfer learning provided three clear benefits.

First, faster convergence — instead of starting from random weights and learning everything from zero, our models began with meaningful features and only needed to learn "what makes a flower different from other flowers." YOLOv5-cls reached 90%+ accuracy within just 20 epochs, something that would take many hours and effort to train from zero.

Second, better generalization with limited data. We only had ~4,000 training images for 102 flower classes (40 images per class). Training from scratch with so little data would cause severe overfitting. But because our pretrained backbones already understood basic visual concepts, they could generalize well .

Third, reduced computational requirements. When we freeze pretrained layers, we only trained a fraction of the total parameters — 130K for YOLOv5-cls instead of 5M, and 124M classifier parameters for VGG19 instead of all 143M. This meant faster training, less GPU memory, and lower energy costs.

8. Conclusion

YOLOv5-cls was the clear winner for this flower classification task. It achieved approximately 7% higher test accuracy (91.82% vs 84.77%) while training nearly 1000 times fewer parameters. This efficiency advantage translated to faster training YOLOv5-cls converged in just 20 epochs with a fixed learning rate, while VGG19 required 35 epochs and a learning rate scheduler to reach its peak performance which are still lower than what YOLO achieved. YOLO feels more like a plug & play model.

The performance gap comes down to architectural design. VGG19, developed in 2014, follows a "deeper is better" approach with simple stacked 3×3 convolutions. It may have been good in past times, but this design is parameter-inefficient — most of the 143 million parameters sit in the massive fully-connected classifier layers that are prone to overfitting and hard to dune correctly in a fast way. YOLOv5-cls, newer, uses modern techniques like CSP connections that extract richer features with far fewer parameters.

The optimizer and batch size choices were selected to meet the needs of each architecture. YOLOv5-cls worked well with Adam optimizer (lr=0.001) and larger batch sizes (64), benefiting from Adam's adaptive learning rates that handle the small parameter count efficiently. VGG19, with huge number of trainable parameters, required SGD with momentum (0.9) and a smaller batch size (32) to navigate the larger parameter space with more caution. The *ReduceLROnPlateau* scheduler helped us to get over the 80%% accuracy— without the automatic learning rate reduction from 0.01 to 0.001 at epoch 14, validation accuracy would have remained stuck around 70% to 80% instead of reaching 85%.

Model robustness was excellent for both architectures. The seed variance was minimal — only 0.14% for YOLOv5-cls and approximately 1% for VGG19 — indicating both models learned genuine flower features rather than memorizing specific training examples. This consistency across different data splits demonstrates reliable generalization. VGG19 might perform better in more complex patterns or more fine-grained feature learning.

Both models comfortably exceeded the 70% accuracy requirement, with VGG19 achieving 84-85% and YOLOv5-cls reaching 91-92%. The success of both approaches validates transfer learning as a powerful technique for image classification tasks where training data is limited

9. References

1. Ultralytics YOLOv5. https://github.com/ultralytics/yolov5
2. Oxford 102 Flowers Dataset. https://www.robots.ox.ac.uk/~vgg/data/flowers/102/
3. https://github.com/NajdBinrabah/Deep-Learning-with-PyTorch-and-Captum

10. Appendix

10.1 GitHub Repository

https://github.com/Jacob170/flower_classification

10.2 Dataset Link

https://www.robots.ox.ac.uk/~vgg/data/flowers/102/