Machine Learning Course

Exercise 4

Name:

Eden Koresh

ID:

206845315

Date:

02/02/2025

Contents

Description of Models

In this problem we have implemented transfer learning for 2 pretrained CNNs: VGG19 and YOLOv5 on the 102-category flower dataset. VGG19 is a CNN primarily used for image classification and feature extraction, whereas YOLOv5 is an object detection model that can identify multiple objects in an image along with their locations.

VGG19

VGG19 is an extension of the Visual Geometry Group family of networks. It consists of **19 layers**, including **16 convolutional layers** and **3 fully connected layers**.

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace=True)
    (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False
)
    (19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (24): ReLU(inplace=True)
    (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (26): ReLU(inplace=True)
    (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False
)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (31): ReLU(inplace=True)
    (32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (33): ReLU(inplace=True)
    (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (35): ReLU(inplace=True)
    (36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False
)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

The architecture of VGG19 is characterized by its use of small 3×3 convolutional filters, stacked in increasing depth across different stages. The first two convolutional layers operate on an input image with three channels (RGB) and output 64 feature maps. Each convolutional layer is followed by a ReLU activation function to introduce non-linearity, which helps in training deeper networks by mitigating the vanishing gradient problem. After two convolutional layers, a max-pooling layer with a 2×2 kernel and a stride of 2 reduces the spatial dimensions of the feature maps by half.

The convolutional layers are grouped into five blocks, each followed by a max-pooling layer with a 2×2 kernel and stride of 2, effectively reducing the spatial dimensions while preserving essential features.

As the network progresses, the number of convolutional filters increases to 128, 256, and finally 512 in the deeper stages. The model maintains its structure of two to four convolutional layers per stage, followed by a max-pooling operation. The repeated use of small convolutional filters ensures that the network can capture intricate features while keeping the number of parameters manageable. The deeper layers capture higher-level patterns, such as textures and object parts, while the earlier layers focus on edges and simple shapes.

After the convolutional and pooling layers, the feature maps pass through an adaptive average pooling layer, which standardizes the output size regardless of input dimensions. The output is then flattened into a vector of 25088 elements and fed into a fully connected classifier. The classifier consists of three dense layers: two with 4096 neurons each, followed by the final layer, which outputs 1000 class probabilities (for ImageNet classification). ReLU activations are applied to the fully connected layers, and dropout regularization with a probability of 0.5 helps prevent overfitting.

For our dataset we modified the classifier block to match our 102 possible labels. The last layer was modified to output 102 class probabilities instead of 1000 like the original block.

```
Sequential(
  (0): Linear(in_features=25088, out_features=4096, bias=True)
  (1): ReLU(inplace=True)
  (2): Dropout(p=0.5, inplace=False)
  (3): Linear(in_features=4096, out_features=4096, bias=True)
  (4): ReLU(inplace=True)
  (5): Dropout(p=0.5, inplace=False)
  (6): Linear(in_features=4096, out_features=102, bias=True)
)
```

In addition, we froze all convolutional layers. The pretrained CNN already learns general features (edges, textures, shapes) from ImageNet. These features are useful for flower classification, so we don't need to retrain them. Freezing the convolutional layers also saves time and reduces overfitting because we only train the fully connected layers.

YOLOv5

YOLOv5 (You Only Look Once version 5) is a deep learning model designed for real-time object detection. Unlike VGG19, which is primarily a classification network, YOLOv5 is an end-to-end object detection framework capable of simultaneously localizing and classifying objects within an image. It predicts bounding boxes and class labels in a single forward pass through the network.

The architecture of YOLOv5 consists of three main components: the backbone, neck, and head. The backbone is a feature extractor based on a modified Cross Stage Partial Network (CSPNet), which enhances gradient flow and reduces computational cost while maintaining accuracy. The backbone processes the input image through multiple convolutional layers, using batch normalization and the Swish activation function to improve training stability and performance.

The neck of YOLOv5 is responsible for multi-scale feature fusion. It includes a Feature Pyramid Network (FPN) and a Path Aggregation Network (PAN), which help in detecting objects at different scales. These structures enable the network to learn both fine-grained details for small objects and high-level features for larger ones. The PAN further refines the feature representations before passing them to the detection head.

The detection head is the final stage of YOLOv5, where bounding boxes, confidence scores, and class predictions are generated. It utilizes anchor-based predictions to refine object localization, applying non-maximum suppression (NMS) to eliminate redundant detections. The head consists of convolutional layers that output feature maps corresponding to different object sizes, ensuring accurate detection across varying scales.

YOLOv5-CLS is a classification-only version of YOLOv5, meaning it does not perform object localization or generate bounding boxes. Instead, it is designed to categorize an entire input image into a single class label. This algorithm does not output coordinates for detected objects. Instead, it produces a class label and confidence score for the entire input image. The model consists of a feature extractor (CSPNet-based backbone) followed by a fully connected classification head.

Preprocessing

The preprocessing steps convert raw images into a format suitable for training a deep learning model. Those are the steps:

Image resizing

Since deep learning models require fixed-size inputs, I resized all images to 224x224 using torchvision.transforms which uses interpolation methods such as bilinear interpolation. This specific size is the default size of the pretrained VGG19 model. For the YOLOv5 model I used 640x640 size to match its default. This ensures a consistent input shape across all images.

A smaller image size increased the training and testing speed, but it is important to match the pretrained default size of the model when applying transfer learning.

Normalization

The images were normalized by subtracting the mean and dividing by the standard deviation. It ensures the pixels value are centered around 0 improving the model's ability to learn efficiently.

Data augmentation

We added slight augmentations to the data to ensure it is able to detect an image even when the image is not perfectly centered and aligned. We added random horizontal flips which randomly flip the image left-to-right with a probability of 50%. In addition, we added a random rotation up to ±15 degrees. This augmentation helps the model to generalize more its classification. **All the data augmentation only applied to the training set.**

Results

Let's review the results of the 2 models we used in the transfer learning:

Table 1: Accuracy results for both models

Model		Train Accuracy	Validation Accuracy	Test Accuracy
VGG19	Run 1 - 159	85.17%	71.28%	73.14%
	Run 2 - 42	80.58%	71.23%	70.80%
YOLOv5	Run 1 - 159	-	98%	98%
	Run 2 - 42	-	97.9%	97.9%

In this table we can see the accuracy results for the 2 pretrained models. We applied the model to 2 runs with different random seeds for splitting the data (159 and 42). The application of YOLOv5 we used has not generated train accuracy that we can extract, therefore it is blank.

It's easy to notice that the YOLOv5 showed more optimized results than the VGG19 model. Let's review the loss and accuracy graphs of the models for the best run (159):

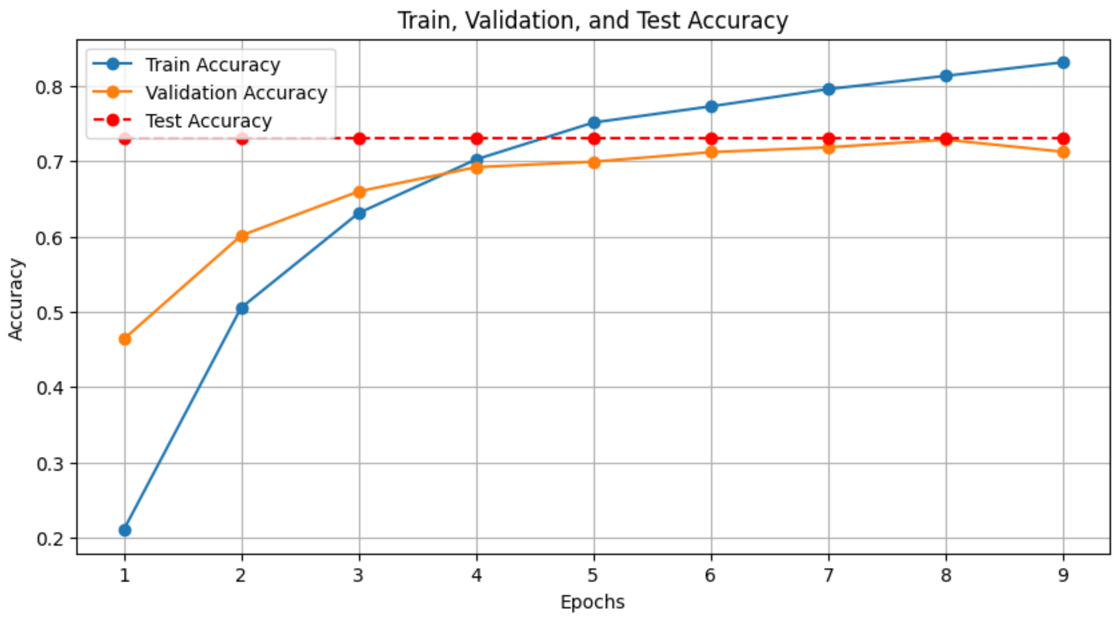

Figure 1: Train, Validation and Test accuracies of VGG19

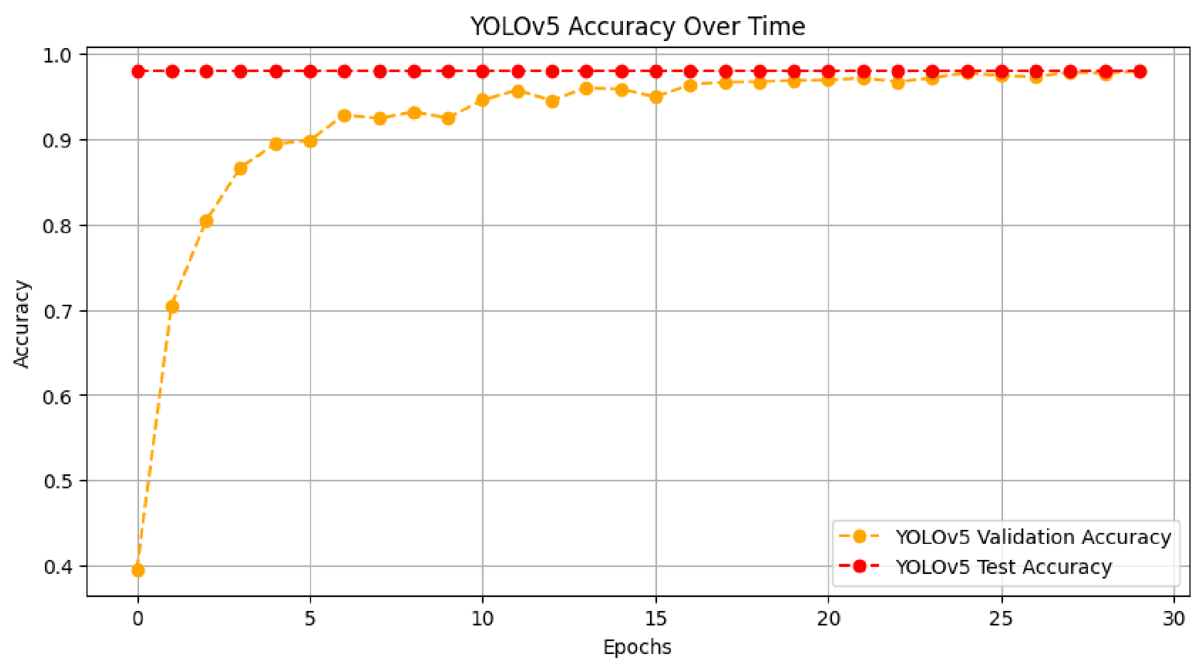

Figure 2: Validation and Test accuracies of YOLOv5

From the figures above we can see the trends with the accuracies of both models. In the VGG19 model we can see that the training accuracy of the model did not reach yet the plateau hence we can infer that further epochs would increase the training accuracy significantly. However, the validation reached the plateau therefore, the further epoch would not improve much the accuracy values. In contrast, in the YOLOv5 model we can see about the validation accuracy also reaching the plateau however, the accuracy levels are phenomenal, around 98%.

For both models, the test accuracy surpassed or matched the validation accuracy.

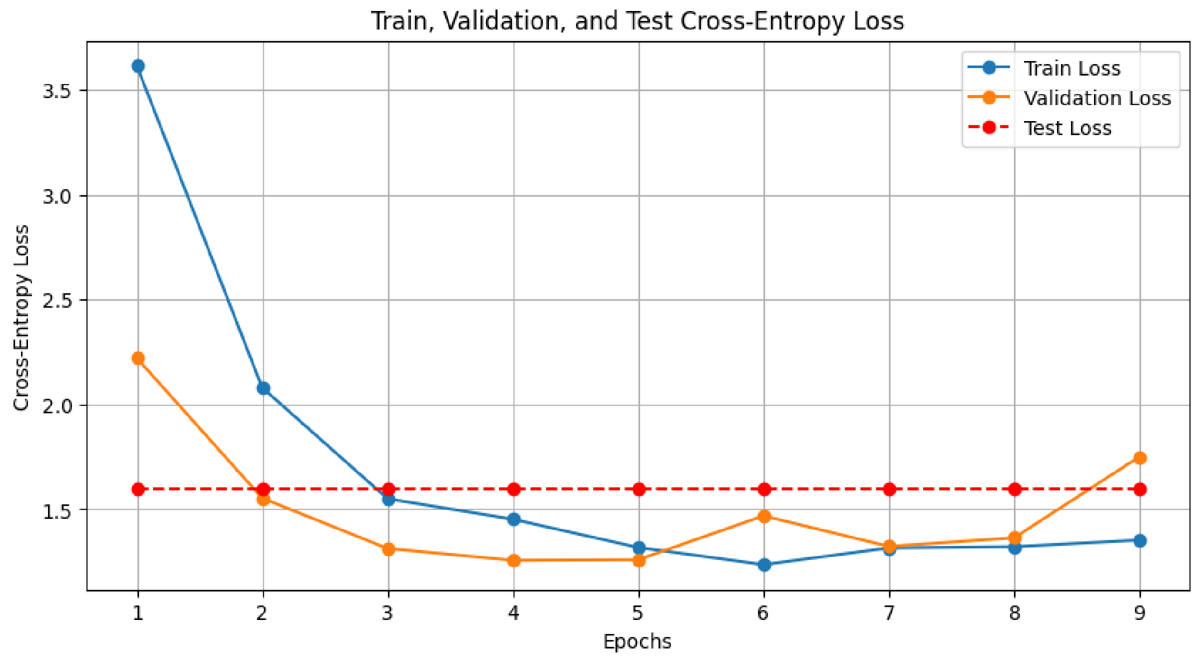

Figure 3: Loss values of VGG19

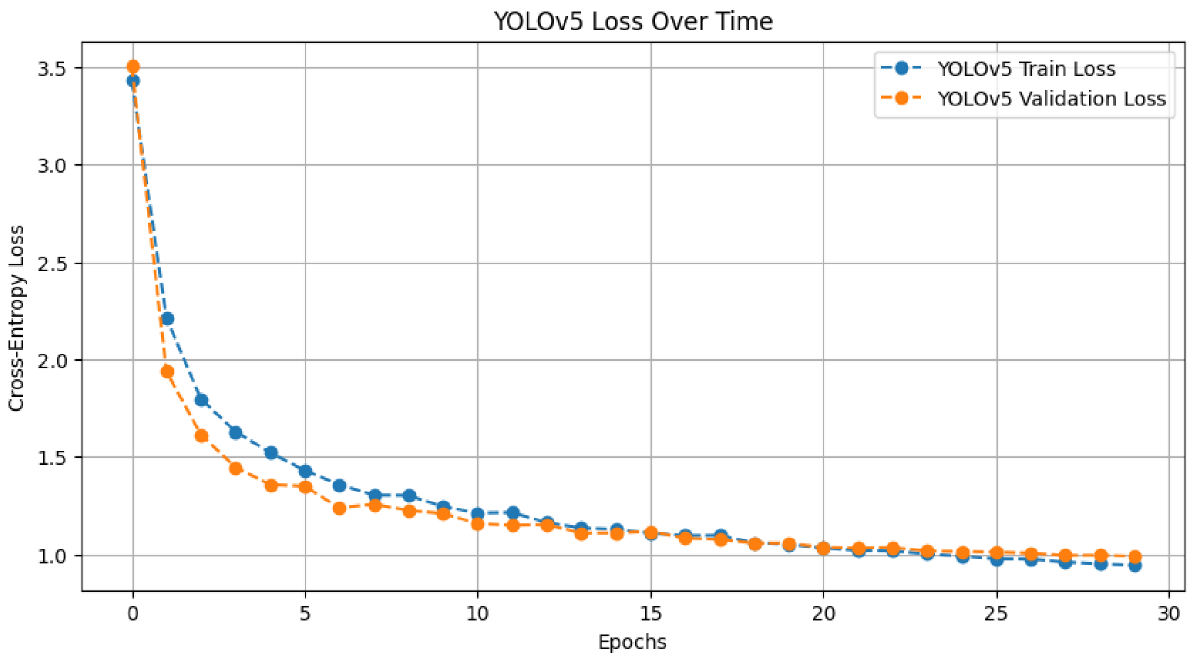

Figure 4: Loss values of YOLOv5

The VGG19 model did not improve after the 5th epoch, the validation loss started to increase where the YOLOv5 continued to decrease as the epochs increased.

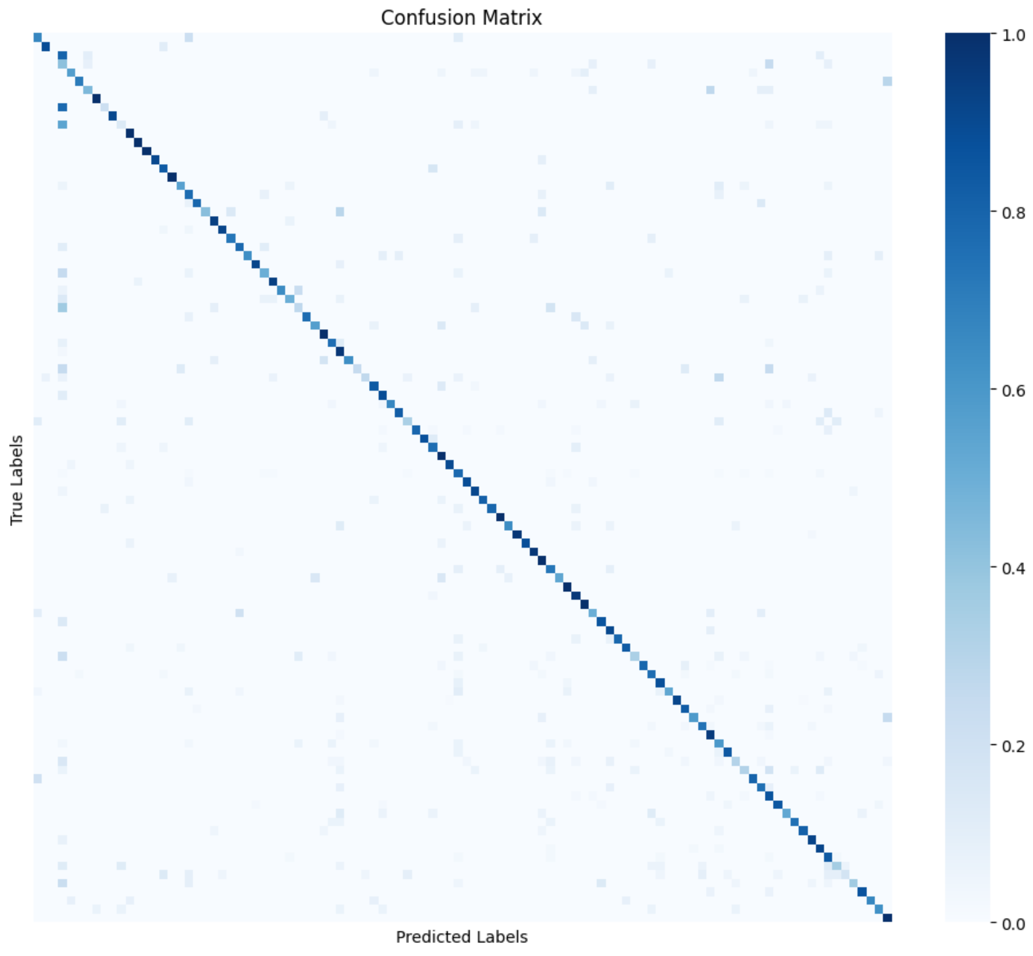

Figure 5: Confusion matrix of VGG19

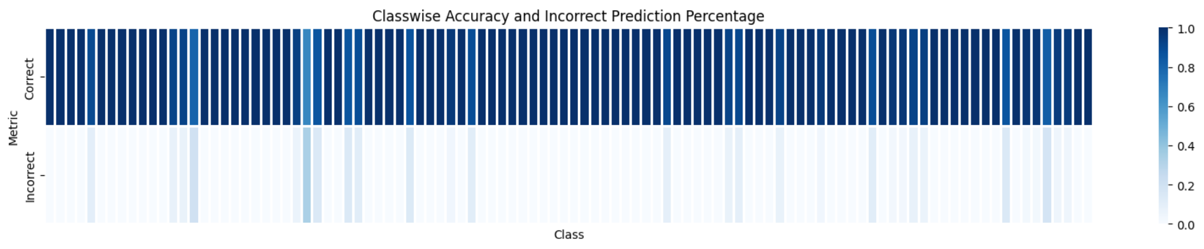

Figure 6: Heatmap of YOLOv5

YOLOv5 model did not provide the labels classification to each image, only the final accuracy for each class. Therefore, we can only show a heatmap with the correct and incorrect probabilities where at VGG19 we can see the true and false classes classified by the model.

In YLOv5 we can see that false classification is rare and miniscule. There are some classes that are below the majuraty classification true prediction percentage, however it is almost not noticeable. The lowest is 0.67 where the next one is 0.8. Mainly the prediction went very well and when looking at the top 5 prediction (where the correct label is in the top 5 classes predicted) we got a higher prediction for all classes, mainly 100% in the majority of classes.

In VGG19 there are some false classifications. Mainly the diagonal is dark blue which means quite a high probability for true positive, however there are many classes that are mostly classified incorrectly. The classification in the incorrect classes is diverged between all the classes which means that the prediction and training of the model needs to be more tuned to match our dataset.