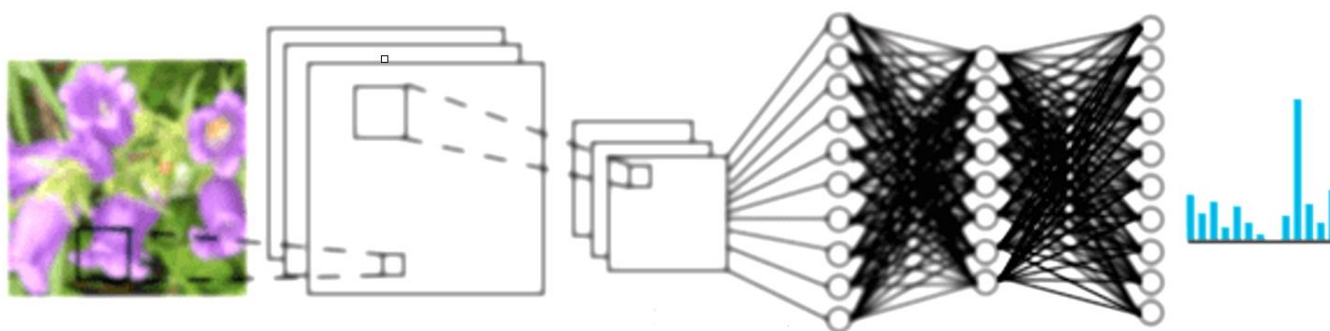


---

-Task 2-

*Flower Classifier based on Task Transfer using ResNet50V2*

---



**Lecturer:** Dr. Ahron Bar-Hillel

**Teaching Assistant:** Mrs. Liel Cohen-Lavi

# 1. Introduction

In this report, we describe the development process of a flower classifier by using transfer learning based on a pre-trained *Convolutional Neural Network* (CNN) called *ResNet50V2*.

The proposed network was originally trained on images belongs to the popular *ImageNet* dataset of 1000 classes. The pre-trained network was used as a baseline pipe for transfer knowledge in order to train a binary flower classifier on a dataset of images collected by the *Volkani* institute. Further, the *Volkani* dataset contains 472 cropped images of flowers and non-flowers, with corresponding labels. Note that due to the large original images size from which both flower and non-flower rectangles were cropped from are not given. In addition, the abovementioned Baseline pipe was optimized in order to find the best hyperparameters using a K-Fold Cross Validation procedure on 301 first images (ascending order) by shuffling and dividing the data by a 90%-10% ratio, which referred to as *Train-set* and tested on the rest of the images per class referred to as *Validation-set* respectively. Further, after finding the best Baseline pipe configuration using the minimal Averaged Validation Error, and in order to test pipe's generalization capability, by testing it on the next 171 images per referred to as *Test-set*. Finally, we used several optimization techniques such as Data Augmentation, Architecture Modification, and Representation Learning which will be elaborated below.

The report is organized as follows: Sec. 2 discusses the modifications and optimization performed on the ResNet50V2 model, referred to as *Baseline Pipe*. Sec. 3 discusses several techniques that were used in order to achieve an improved pipe with higher performance compared to the baseline pipe. Finally, in Sec. 4 we present the results found using the best configuration and optimization technique.

## 2. Baseline ResNet50V2 Optimization

This section describes the optimization process of ResNet50V2 CNN, known as our Baseline model. First, in subsection 2.1 we describe the concept of Residual Network based on He, Zhang et al. paper [1]. Second, in subsection 2.2 we describe how we used transfer learning from a 1000 class classifier which was trained on dataset known as the *ImageNet*, to a binary flower classifier. Third, in subsection 2.3 we describe the hyperparameter tuning process for obtaining the best baseline model configuration found to fit our task, without changing the ResNet50V2 architecture. Finally, in order to test the model's generalization capability, in subsection 2.4 we present a performance evaluation based on the test data using our best-selected Baseline model which described in subsection 2.3.

### 2.1 Residual Network

Training deep convolutional neural networks is much more difficult than training shallow networks. To address this issue, He, Zhang et al. [1] presented a Residual Learning framework to apply the training of substantially deeper neural networks. Fig 1. Below describes the residual network architecture which was created by reformulating the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. The authors also provided a comprehensive empirical evidence and showed that these residual networks known as *ResNets* can gain accuracy from the considerably increased depth and moreover, easier to optimize.

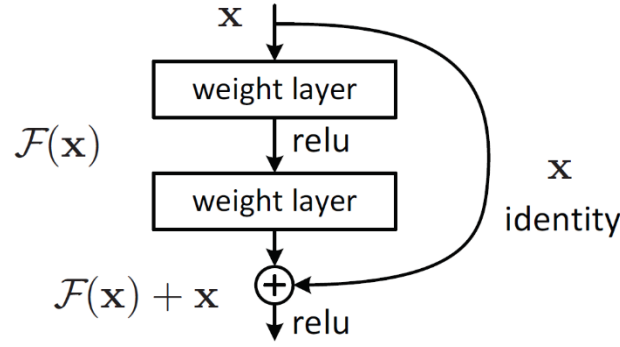


Figure 1 - Residual learning: a building block [1].

## 2.2 Transfer Learning using ResNet50V2

In order to address the task of binary classification, we needed to modify the original ResNet50V2 architecture [1], since it was built to classify images within the ImageNet dataset, which has 1000 classes. First, after initiating a ResNet50V2-based model with its pre-trained ImageNet weights, we removed the top fully connected (FC) layers and the last output layer which had 1000 neurons, one per each original class. Second, we transformed the layer before last  $H^{L-1}$ , which was a  $7 \times 7 \times 2048$  using global average pooling over the  $7 \times 7$  spatial dimensions to get a vector  $V$  of 2048 elements, with  $V[k] = \frac{\sum_{i,j} H^{L-1}(i,j,k)}{7 \cdot 7}$ . Third, in order to create the output neuron  $p$  for our modified network we used a dense layer and a sigmoid function based on their following equations:

$$p = \sigma(w_L \cdot V - b_L) \text{ with } \sigma(x) = \frac{1}{1 + \exp(-x)}$$

As a result, we were able to estimate the probability that an image is a flower, denoted as  $\hat{p}(y = \text{flower}|x)$ . Moreover, in order to train the modified network, for every single image we minimized the loss using the following binary cross-entropy loss function:

$$\text{loss}(y, p(x)) = \begin{cases} \log p & y = 1 \\ \log(1 - p) & y = 0 \end{cases}$$

Then, the total loss was also calculated by the following equation:  $L = \sum_{i=1}^N \text{loss}(y_i, p(x_i))$ .

After the abovementioned steps, we were able to adjust the classification task from a multiclass to a binary. Then, we optimized our initial baseline model, which will be elaborated in the next subsection.

## 2.3 Baseline Model - Hyperparameter Tuning

In this subsection, we describe the hyperparameter tuning process for obtaining the best baseline model configuration without changing the abovementioned ResNet50V2 architecture. In order to find the best hyperparameters configuration for our Baseline pipe, we conducted several experiments to find the best values for the following hyperparameters: Number of trainable layers, Batch Size, Optimizer & Epochs. In addition, we used **KfoldCrossValidation(..)** function to conduct a 3-Fold Cross-Validation (CV) with a ratio of 90\10 between both *Train* and *Validation* sets respectively. Although computationally expensive, by using CV the training error is being

averaged and therefore, provides more reliable results in order to determine which hyperparameter configuration outperformed. Further, instead of tuning each hyperparameter separately, which can result in much poor performance, we tuned two hyperparameters simultaneously which takes the interaction between into account. Experiments configuration is described below.

### 2.3.1 Hyperparameters Optimization - Experiment 1 Configuration

In this experiment, using the **hyperparameterOptimizationExperiment()** procedure, we used different hyperparameters values in order to find the best candidates for optimizing our Baseline model in general. In particular, the hyperparameters that were optimized in this experiment are the amount of Trainable Layers and Batch Size. Further, it's important to specify that the reason for choosing to freeze the training of some convolution layers was saving time and to preserve some of the 1<sup>st</sup> layers weights since they usually relate to low-level features such as edges, simple shapes, etc. which can be relevant for our flower classification task. Furthermore, we used a fixed split ratio of 90\10 between both *Train* and *Validation* sets respectively. In order to see the interaction between both hyperparameters, we used a grid search of 30 different combinations using a 3-Fold CV ( $3 \times 10 = 30$ ) procedure which describes below. In addition, because of time and performance constraints, we used only 3 epochs for training and picked a small subset of batch size and trainable layers to be tested described below.

- 3 Batch Size Candidates - *Batch – Size*  $b \in \{32, 64, 128\}$
- 10 Trainable Layers Candidates -  $N_{Trainable\ Layers} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

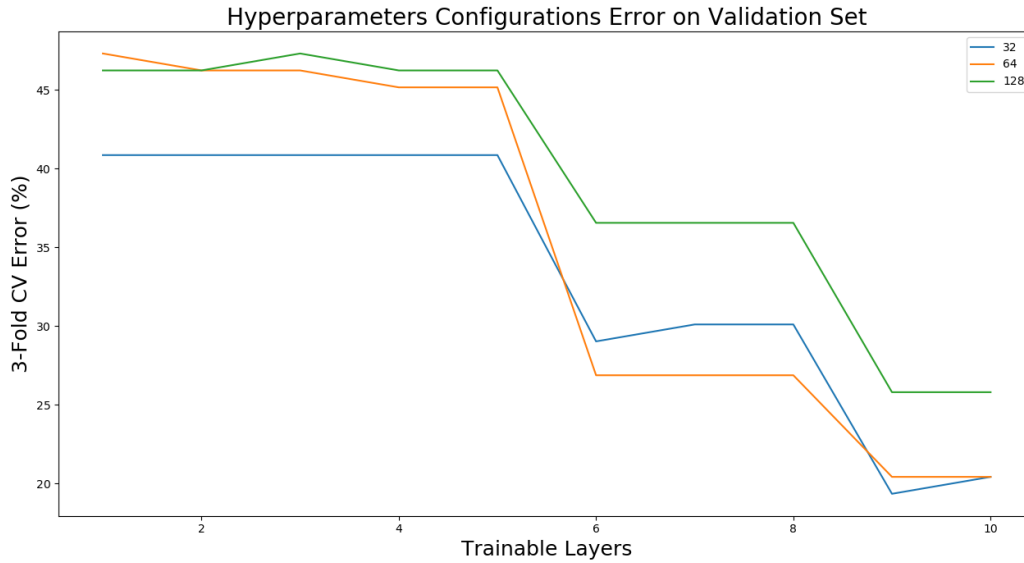


Figure 2 - 3-Fold Cross Validation results using a Grid Search while Tuning Trainable Layers (X Axis) and Batch Size (Color Axis) based on Validation Error (Axis Y).

By viewing Fig. 2, generated using our **plot3DErrorPerformanceChart()** function, we can infer that 9 Trainable Layers using Batch Size of 32 are the hyperparameters candidates that outperformed the rest, and achieved an averaged error of 20%. Then, we used the achieved values and continued to the 2<sup>nd</sup> experiment described below.

### 2.3.2 Hyperparameters Optimization - Experiment 2 Configuration

In the 2<sup>nd</sup> experiment, in order to determine which Optimizer outperforms the rest, first, we have fixed the number of trainable layers to 9, and we set the batch size to 32 based on the results derived from the 1<sup>st</sup> experiment. Second, note that each optimizer had 3 epochs for convergence and it was tested its robustness using a 5-fold Cross-Validation with a fixed split ratio of 90\10 between both *Train* and *Validation* sets. Moreover, we compared the performance of the following Optimizers: *Stochastic Gradient Descent (SGD)*, *RMSProp*, *Adam*, *Nadam* & *Adadelata*.

By viewing Fig. 3 below, we can infer that both *Nadam* and *Adadelata* optimizers outperformed the rest, and achieved an averaged error of 20%. Then, we used the achieved values and continued to the 3<sup>rd</sup> experiment described below.

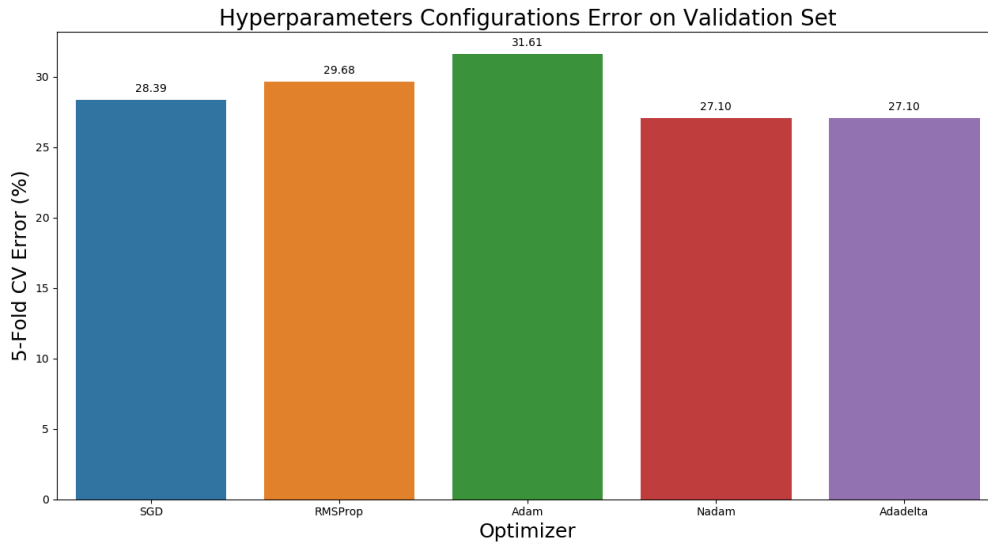


Figure 3 - 5-Fold Cross Validation results using a Grid Search while tuning Optimizers (X Axis) based on Validation Error (Axis Y).

### 2.3.3 Hyperparameters Optimization - Experiment 3 Configuration

In order to select the best optimizer derived from the 2<sup>nd</sup> experiment's results, in the 3<sup>rd</sup> experiment, we compared the validation error between *Nadam* and *Adadelata* optimizers, while testing them on several epochs with a batch size of 32, and 9 trainable layers. Further, we used a 3-fold Cross-Validation with a fixed split ratio of 90\10 between both *Train* and *Validation* sets. Moreover, we compared the performance of each optimizer to a different amount of epochs from the following subset:  $N_{epochs} \in \{1, 3, 5, 7, 9, 11, 13, 15\}$ .

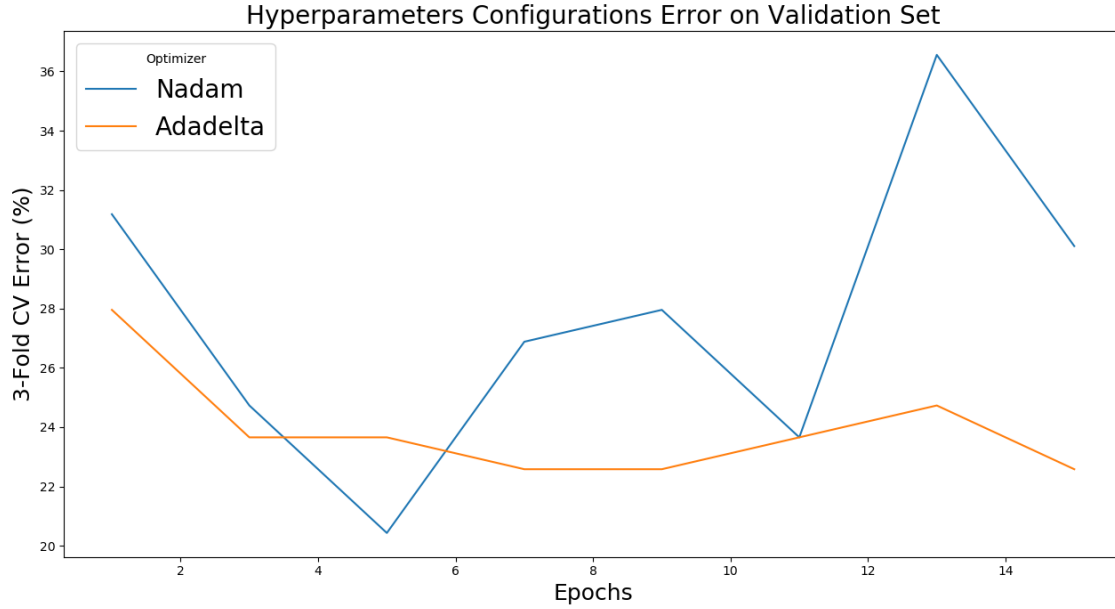


Figure 4 - 3-Fold Cross Validation results using a Grid Search while tuning Nadam & Adadelata optimizers (Color Axis) based on the amount of Epochs (X Axis) and Validation Error (Axis Y)

By viewing Fig. 4 above, we can infer that *Nadam* outperformed *Adadelata* and achieved an average error rate of 20% using 5 epochs. However, it's important to note the variance differences between both optimizers, such that *Nadam* variance is much noisier comparing to *Adadelata* variance which is much more smoother. As a result, we preferred to work with a more stable optimizer, and chose *Adadelata* with 7 epochs, which is the smallest amount of epochs among all epochs that achieved the same validation error of 23%.

## 2.4 Baseline Pipe Generalization Evaluation

In order to test our Baseline pipe generalization capability, in this subsection, we present a performance evaluation of the best-selected hyperparameters configuration found in the previous subsection, using the *Test-set*. Note that the best-derived hyperparameter configuration was composed of the following values:

- $N_{Trainable\ Layers} = 9$
- $N_{Batch\ Size} = 32$
- $O_{Optimizer} = Adadelata$
- $N_{epochs} = 7$

Results showed that an accuracy score of 77.2% was achieved on *Test-set* using the abovementioned configuration. In addition, since our CNN classification is estimated using the output neuron  $p(y = 1|x)$ , a default threshold of 0.5 is being set also in order to distinguish between both classes based on their predicted probabilities. Since threshold value has an effect on classification results, we introduce two important classification concepts for deciding which

threshold to use. First, the fraction of the total amount of relevant instances that were actually retrieved referred to as *Recall*. Second, the fraction of relevant instances among the retrieved instance referred to as *Precision*. Moreover, both KPIs equations are described below:

- Recall:  $\frac{\#detected\ real\ objects}{\#real\ objects}$
- Precision  $\frac{\#detected\ real\ objects}{\#detections}$

By varying threshold  $t$  between 0 and 1, we can obtain a recall-precision curve composed of achievable (recall, precision) points. Therefore, in order to analyze our Baseline pipe performance, in Fig. 5, a Recall-Precision curve is being displayed. Furthermore, in Fig. 6 we present the derived confusion matrix.

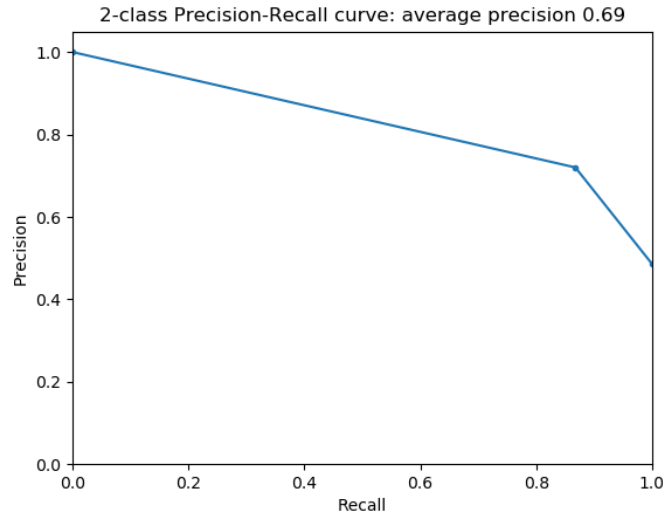


Figure 1 – Baseline Pipe Precision-Recall Curve

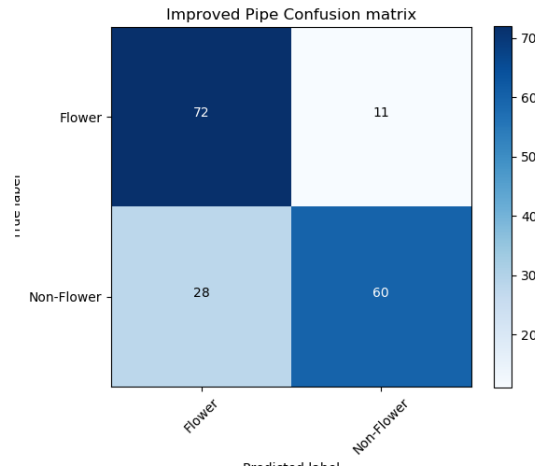


Figure 6 - Baseline Pipe Confusion Matrix using Test set

### 3. Improved Optimization Techniques

In order to improve the baseline model, we have decided to add a few advanced modifications to it. Simerly to the baseline model, we have removed the ResNet top layer. Though this time, as described in Fig. 7, we have replaced it with the following explained structure; An average pooling layer which converts a 7x7x2048 to an averaged 2048 vector than two global layers with a width of 1024 neurons based on a RELU activation function and a neural decay addition (Kernel-Regulazers) with an L2 norm penalty, finally, we have added a 1 numeral layer with a softmax activation function. Before each any inner product of the wights and input batch, preceding to any RELU function, we have decided to perform batch normalization in order to reduce to wights sifting magnitude during backpropagation. The batch normalization and neural decay intuitions were to keep the wights close to their original values based on the assumption that the original ResNet features are quite informative, therefore the wights' deviation towards the flower classification problem should be milled.

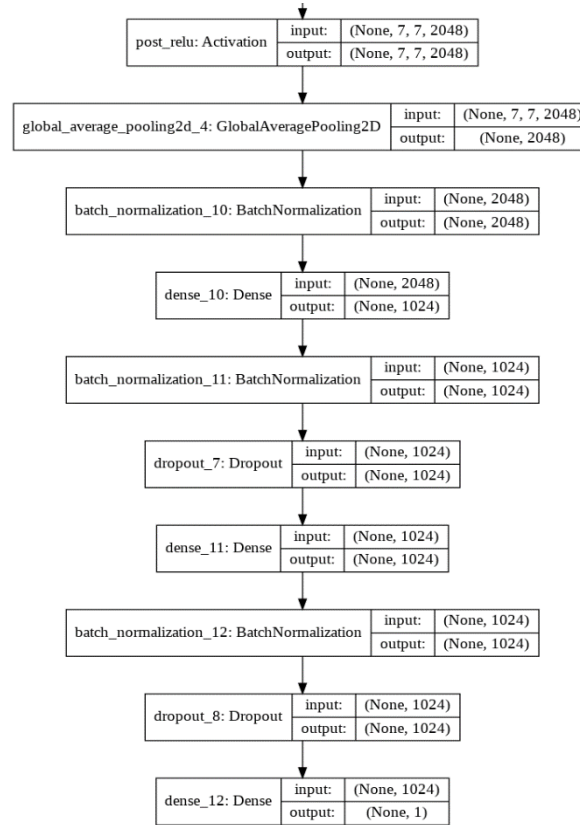


Figure 7: Global Layers additions

#### 3.1 Data Augmentation

A transfer learning is commonly used when we encounter a problem where data is scarce, Yet in a similar problem, data can be found on a scale that enables us to train a decent classifier. In our problem, we face a similar situation where the flower dataset is relatively small, especially if our goal is to train a CNN. One resolution to our problem is using data augmentation, where images originally taken from the dataset are altered and transformed in a



way that may resemble an image that would have fitted in our dataset and thus artificially enhances the volume of samples in the dataset. We have used an image data generator from Keras preprocessing class in order to generate transformed images using the following transformations: rotation, width shifting, height shifting, zoom range, shear, and horizontal flip. The data augmentation was performed in each batch, where a transformation was chosen randomly on input insertion. Note that original images were inserted as well.

### 3.2 Representation Learning using ResNet50V2

Representation learning traverses around the idea of creating a new representation of given features that would better represent the input in a way that would contribute to a better understanding of the learned subject compared to an equivalent understanding with the use of the original features. In our case, we have decided to take advantage of the smart and complicated features created by our new improved CNN and use those features as an input to the Support Vector Machine (SVM) classifier that we have used in the first assignment.

The process was as follows, we took the improved architect described in the previous section, based on the baseline hyperparameter tuning, trained it for 7 epochs on the training dataset using the data augmentation generator, later we have removed the last two layers and used the trained network to produce a training dataset for the SVM (Dimensions: N Samples X 1024) using the original training dataset, we did the same for the testing dataset. Finally, we have trained and tested the SVM classifier with the features produced by our improved network.

## 4. Results

In this section, after deriving our best representation learning-based pipe, we have tested its generalization capability. we present a performance evaluation of the best-selected hyperparameters configuration found in the previous subsection, using the *Test-set*. Note that in this pipe we used an improved ResNet architecture for the task of feature engineering. In addition, we use a similar hyperparameter configuration to our best-derived Baseline pipe, which is described above. Furthermore, the extracted features were fed to an SVM classifier configured by the best hyperparameter configuration found on our 1<sup>st</sup> image recognition task, which is presented below:

- C - 0.1
- Gamma - 0.1
- Degree – 2

Results showed, that our improved pipe was able to achieve an accuracy score of 83.04% on *Test-set* using the abovementioned configuration. Further in Fig. 8, a Recall-Precision curve is being displayed, while in Fig. 9 we present the derived confusion matrix. Furthermore, in Table 1. we present images of the 5 worst errors of type 1 and type 2. Note that the images are sorted in descending order according to their score, i.e., the first image for each error type is the one with the highest score and the 5th image is the one with the 5th highest score.

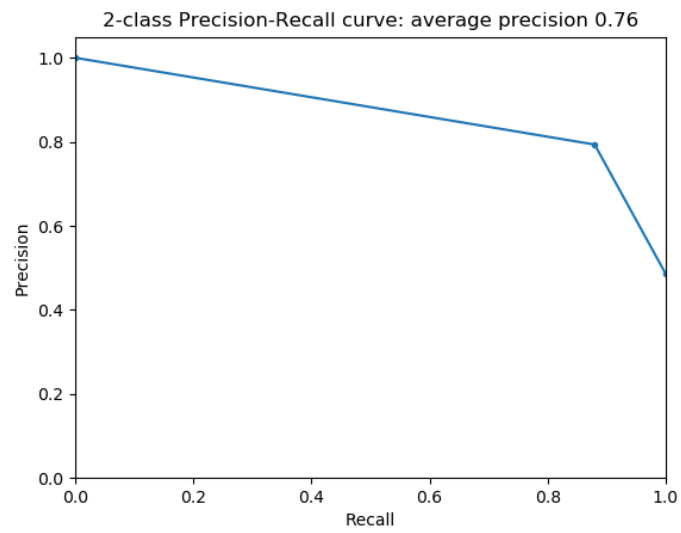


Figure 8 – Improved Pipe Precision-Recall Curve

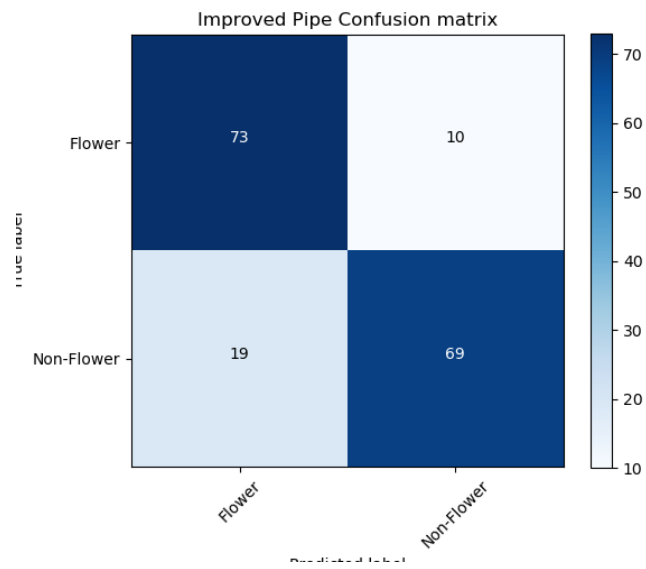






Figure 9 - Improved Pipe Confusion Matrix using Test set

| Type One Error  | Type Two Error  |
|---|---|
| <p>Score: 0.49138262298932384</p>    | <p>Score: 0.9711811223780893</p>    |
| <p>Score: 0.42191668857950165</p>   | <p>Score: 0.9529241047723321</p>   |
| <p>Score: 0.41147218975004574</p>  | <p>Score: 0.9497370802869625</p>  |

|  |  |
|--|--|
| Score: 0.40053443925701115<br>  | Score: 0.9397868023524525<br>  |
| Score: 0.31604935274891766<br> | Score: 0.9393446172248948<br> |

*Table 1 – Images of The 5 worst errors of type 1 and type 2 (on test data)*

## 5. Reference

[1] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).