Classification of flowers

The goal of the project is to figure out the species of flowers using a dataset of input images containing 5 classes of flowers: Lilly, Lotus, Orchid, Sunflower and Tulip.

Link:

https://github.com/lenadub/FINAL-PROJECT-AI.git

Team Members:

Student Name	Student ID	Contribution in the project
CAUVIN Marion	73109	This is a joint effort. We all contributed to each of the steps.
COURSIER Fanny	73206	This is a joint effort. We all contributed to each of the steps.
DUBOIS Léna	73150	This a a joint effort. We all contributed to each of the steps.
MULLER Julie	73191	This a a joint effort. We all contributed to each of the steps.

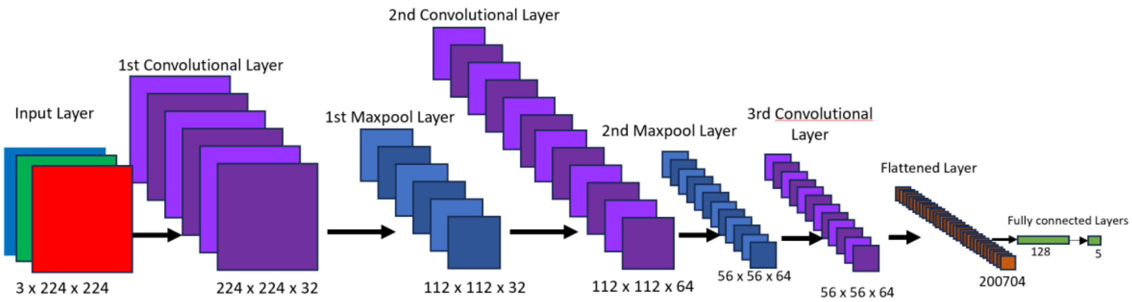
We would like to thank the Machine Learning blog authors on the Internet who helped us get an even greater insight into CNN theory and design (normalization, number of layers, dropout, pytorch code, etc)

Model Architecture:

A **Convolution Neural network architecture** was selected because CNNs have been popular and proved effective, for quite a while, to solve image classification problems. The **final** selected model is made of **9 layers including the input layer and the output layer**. It has **7 hidden layers**. The **total number of neurons excluding the input layer** is **3412101** (see table below for number of neurons per layer)**.**

The layered architecture of the final model from bottom to top is as follows:

Id	Layer Description	Activation	Specific processing	# of neurons
1	**The Image Input Layer.**	N/A	Input Normalization applied	224x224 *3 = 150 528
2	The first Convolutional Layer *Number of kernels : 32* *Stride : 1, padding 1, size : 3x3*	ReLU	Adam optimisation backprop	224*224*32 = 1 605 632
3	A Maxpool Layer , Window : 2x2 Stride : 2	None	Adam optimisation for backprop	112*112*32 = 401 408
4	A second Convolutional Layer *Number of kernels : 64* *Stride : 1, padding 1, size : 3x3*	ReLU	Adam optimisation for backprop	112 * 112 * 64 = 802 816
5	A second Maxpool Layer Window : 2x2 Stride : 2	None	Adam optimisation for backprop	56 * 56* 64 = 200 704
6	A third Convolutional Layer *Number of kernels : 64* *Stride : 1, padding 1, size : 3x3*	ReLU	Adam optimisation for backprop	56 * 56*64 = 200 704
7	A flattened layer.	None	Adam optimisation for backprop	200 704
8	A fully connected layer	ReLU	Adam optimisation for backprop Dropout for regularization	128
9	**The output layer made a fully connected layer used for classification**	None	Adam optimisation	5



The dropout at the 8th layer (fully-connected) was added to narrow down the overfitting (training loss decreasing while validation loss increasing).

We tested 4 other models, which had no better accuracy and some had overfitting issues:

- The model above without the third convolutional layer
- The model above without the third convolutional layer and no dropout
- The model above without the third convolutional layer and with 32 kernels in the second convolutional layer instead of 64.
- The model above without the third convolutional layer and with a larger size kernel (6x6) in the first two convolutional layers.

Dataset Description:

The dataset included 5000 images of flowers divided into 5 species, with 1000 samples for each species.
We divided the dataset into 3 parts:

- Training dataset
    - This dataset was used to train the CNN
    - Its size was set to 80% of the initial dataset
- Validation dataset
    - This dataset including 499 images (10% of the initial dataset) was used to check the progress and performance of the training. After each training epoch, the trained CNN model was evaluated against the validation dataset. In other words, we recorded the loss and accuracy trends measured against the validation dataset while the model was getting trained. It was a way to check whether the model was learning properly (training loss and validation loss decreasing together and accuracy improving) and detect sign of overfitting (training loss decreasing while validation loss increasing)
- Test dataset
    - This dataset of 501 images (10% of the initial dataset) was used to test the model on unseen data once the training phase was completed.

The data was augmented with random rotation (0 to 20 degrees) as well as horizontal flips (with probability 0.1) to make the data closer to what we get in real life.

Methodology:

Machine Learning Coding Environment
We relied upon Google Colab using a T4 GPU and used the Pytorch environment.

Mini-batch
The model was submitted batches of 32 images in each epoch. Mini-batch is computationally efficient and is also memory efficient especially in the google colab environment where GPU memory is limited. It is also mentioned that mini-batch helps against overfitting.

Learning rate:
The learning rate was set to 0.001. It is the recommended learning rate on the Internet when using the Adam optimisation. We were concerned such a small value would make the training phase very long on a Google Colab T4 GPU. However, we found training was quite quick, so we kept this value for all the models tested for this project

Number of epochs:
The number of epochs was set to 50 for the selected model. After that, it started to show signs of overfitting.

The second-best model (selected model without third convolutional layer) got an accuracy close to the best model with epoch number set to 100. There were then signs of overfitting.

Loss function
The loss function used for training was cross entropy.

Changes in parameters
The alternate models and selected model were trained using Input Normalization with the following means and standard deviations for the three RGB colour channels.

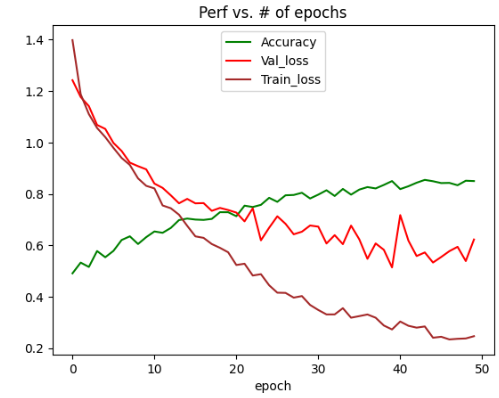
	First models		**Final Model**	
RGB	Mean	STD	Mean	STD
R	0.5	0.5	0.485	0.229
G	0.5	0.5	0.456	0.224
B	0.5	0.5	0.406	0.225

Normalization is used to mitigate gradient descent or explosion and help convergence.
The normalization increased the accuracy.
The green values which are considered as best on many Internet sites improved accuracy compared with the orange values.

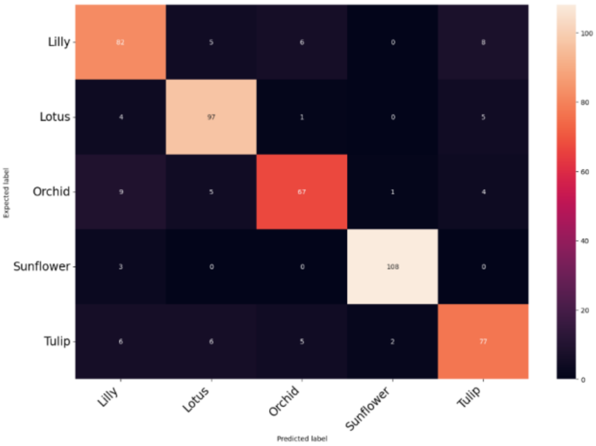
As mentioned in the model architecture section, we adjusted the size and the number of kernels without any significant increase in accuracy.

Results :

Training and validation performance



Confusion matrix on testing dataset



We can notice that after 50 epochs the validation accuracy is still increasing asymptoticly just over 84%. The validation loss is stabilizing and has not really started increasing yet while the training loss keeps decreasing. Overfitting is unlikely at this stage.

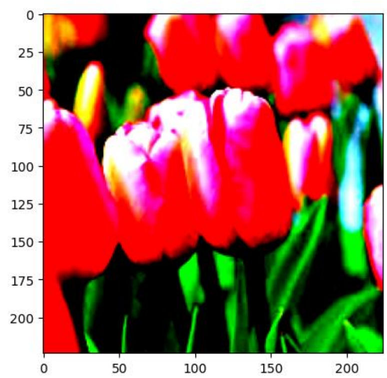
For the testing phase, we got:

- we got a testing accuracy of 0.860 which is quite fair.
- A testing loss of 0.629
- the confusion matrix above

Examples of predictions on testing dataset (note the images shown below are a normalized version of the original ones)

Classification ok

Expected label 4 - Predicted label 4



Classification Nok

Expected label 0 - Predicted label 3