TER REPORT

Fedy Ben Naceur
Romain Magnet

Avril 2022

1 Introduction

The number of plant species exceeds 391000, therefore the task of classifying them is not feasible for a botanist, in addition some plants have high similarities and thus are extremely difficult to differentiate with a naked eye. The task of classifying these plants is vitally important to help scientist and botanists identify these species without the need of additional human effort. Hence there is a need to develop a system to identify and correctly classify these plants. The use of machine learning in classifying living species has been employed for classification and recognition tasks, models can generalize to help identify plants that suffer from diseases or are harmful to humans alongside many other problems The Pachamama challenge consists of creating a robust model capable of classifying living species, it's an AutoML challenge aimed to help experts deal with the enormous number of variety in nature. For the first part of the challenge we have been provided with a data set composed of flower images, the model will developed using this data set and evaluation will be done online on private data. The submission are evaluated using f1-score to account for class imbalances. There are 3 phases:

- Phase 0: public phase. They provide us with **Dataset0** containing labeled data. We will not be evaluated on this dataset, but it will be used to developp the model that we will then submit for other 2 phases.
- Phase 1: development phase. This time we do not have access to any part of the **Dataset1** which will be used to train and test on the model we submitted. The performance of our last submission on the test set will be displayed on the leaderboard.
- Phase 2: final phase. This will follow the same procedure as phase 1 but with the **Dataset2**. And we will only have a single submissions to assess our final performance on the challenge. Our performance on the test set will appear on the leaderboard when the organizers finish checking the submissions ensuring there is no cheating involved.

In this proposal we will detail our road-map and the ideas that we implemented to face this challenge alongside the results that we managed to obtain.

2 Credits

- The creators of the challenge are Lauzzana Gabriel, Khater Yara, Guaranda Maria Belen, Lopez Elsa and Pobelle Marion.
- The competition protocol was designed by Isabelle Guyon.
- The starting kit was adapted from an Jupyper notebook designed by Balazs Kegl.
- Ihsaan Ullah provided the sample bundle .
- Contact email of orginizers : pachamama@challenge.org
- Url of the competition : `https://codalab.lisn.upsaclay.fr/competitions/1447`

3 Data

As we discussed before the data set available for us consists of flower images, we will use this data locally to build, train and validate models before submitting them to the platform.
The data-set we're using for training our model and validating it before sending it to the platform for test, is composed of 8189 images with a 128*128 resolution. The following images are taken from the Dataset0:

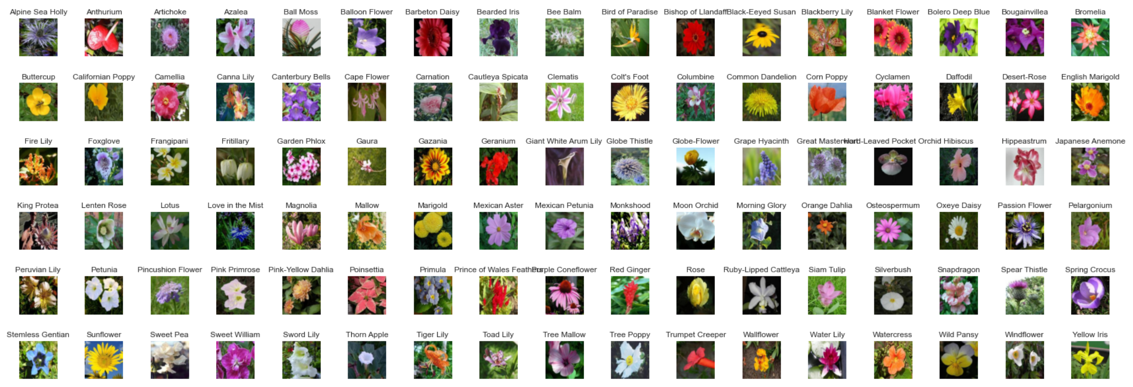


Figure 1: public dataset : Dataset0

From the samples we've shown above we can see that objects are well centered inside images, we can also notice that not images are taken with the same background, some were taken with a sky behind them others with grass, forests or even dark backgrounds. However like shown by Kai Xiao, Logan Engstrom, Andrew Ilyas Aleksander Madry in MITback, image backgrounds are a natural source of correlation between images and their labels in object recognition and model predictions do not exactly follow humans expectations due to correlations such as texture and color that humans don't necessarily rely on when identifying objects.

Here is the distribution of the images with their respective labels, we can see that this distribution is not uniform meaning that some classes are not well represented in the data set.

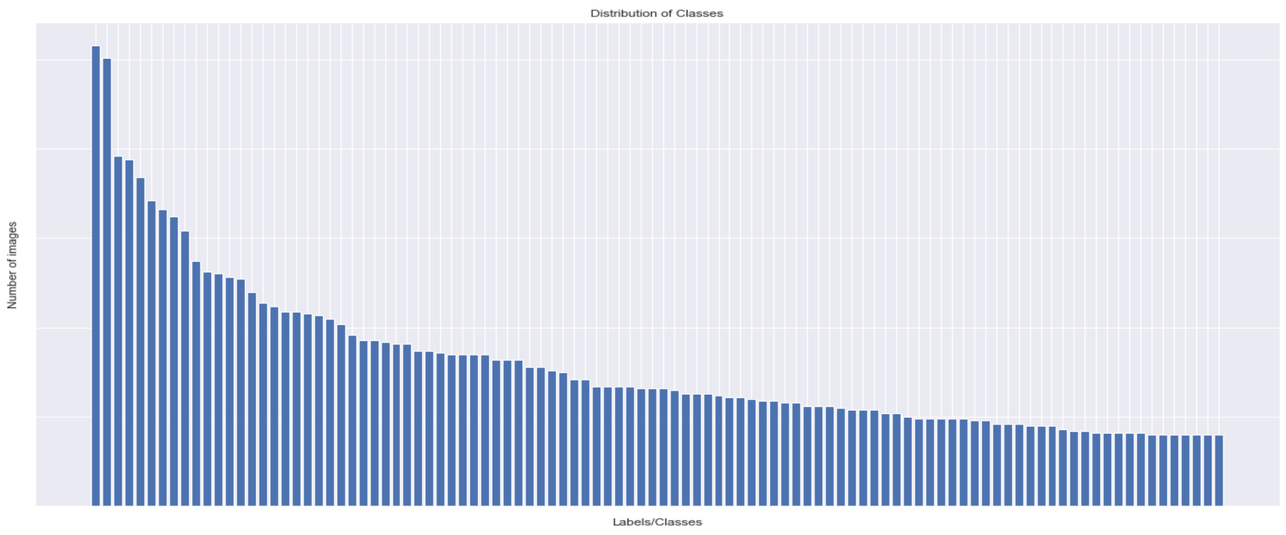
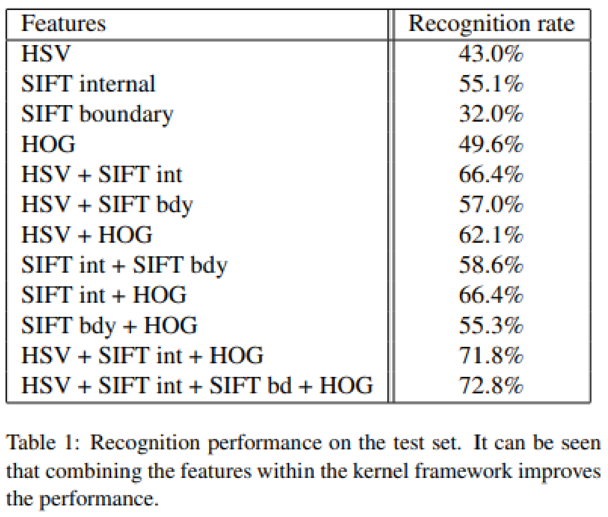


Figure 2: Distribution of classes in the Dataset

We will try to amend this problem in the preprocessing part through data augmentation techniques.

4 Preprocessing

A paper written in 2008 by the Visual Geometry Group in Oxford university showed how a similar problem in flower classification was handled [3], to approach the problem they designed features and corresponding Kernels which capture texture, colour and shape of petals. They also used segmentation to separate flowers in foreground and the rest in background and used a one-vs-all svm approach here are the results obtained:



Features	Recognition rate
HSV	43.0%
SIFT internal	55.1%
SIFT boundary	32.0%
HOG	49.6%
HSV + SIFT int	66.4%
HSV + SIFT bdy	57.0%
HSV + HOG	62.1%
SIFT int + SIFT bdy	58.6%
SIFT int + HOG	66.4%
SIFT bdy + HOG	55.3%
HSV + SIFT int + HOG	71.8%
HSV + SIFT int + SIFT bd + HOG	72.8%

Table 1: Recognition performance on the test set. It can be seen that combining the features within the kernel framework improves the performance.

Figure 3: Results from feature extraction methods described in [3]

We can see that this approach may not transfer well to an AutoML problem because the features being extracted are specific to flowers they might not generalize well to other data.
Instead of manually building these features, we will focus on building convolution neural networks that are capable of learning those representations without the need of manually engineering them ourselves, so that's one less thing to handle during preprocessing.
There are different approaches to preprocessing images before feeding them into a neural network, in the paper detailing the architecture of AlexNet [2] a part from resizing the images to the shape 256*256 pixels, a mean channel pixel value was subtracted from each pixel. The process was minimal but yielded good results on ImageNet for a CNN that was developed in 2012.
Of course we have to take into account the fact that our problem differs from normal classification and that our model must be capable of generalizing well to other similar samples, since we've already been provided data that is uniformly shaped we might want to start by looking into data augmentation techniques:

- Affine transformations include rotations, random cropping and scaling
- Elastic transformations consists of operations like applying Gaussian blur, brightness and color intensity adjustments ...
- Finally there more advanced data augmentation like mixup [5] where you essentially sample points that are in between decision boundaries.

Data augmentation was previously done before training the model, meaning you would create new samples by using a techniques like the ones stated above and save them on disk, but thanks to current deep learning API's we're able to produce the imaged on the fly during the process of training in memory. Augmenting can also be applied during the testing phase to make sure that the model makes robust predictions this was done in AlexNet [2] but also famous architectures like GoogleNet [4]. Data augmentation can be seen as a source of data collection but also a regularizer to avoid over-fitting and this is a critical point when addressing AutoMl problems.

4.1 Visualizing data augmentation

To augment data we chose to go with simple transformations here is a list of what we used to train the models :

- Rescaling the images by dividing by 255, this will allow us to have a pixel values between 0 and 1 .
- 90 angle rotations with horizontal and vertical flips

- Zooming on the image

Here is a plot showing the transformations mentioned on an image taken from the training set :

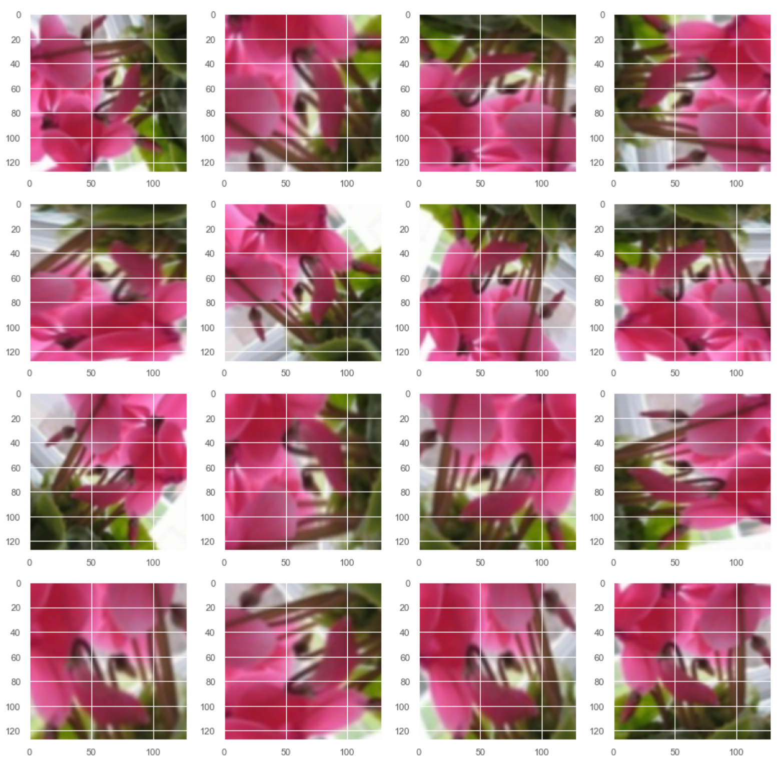


Figure 4: An augmented image taken from the training set

5 Models

Convolutional neural networks have been the go to for computer vision since the past decade so we expect to have the results working with these models. Machine learning algorithms like SVM, MLP do not generally generalize well especially working with images since you have to manually extract features whereas in an AutoML problem we want to be able to learn these features automatically.

Our roadmap will first consist of exploring machine learning algorithms implemented in scikit-learn paired with some feature extraction techniques to see how things were done before the surge of CNNs. Later we will build a CNN from scratch and fine tune it and finally we'll use transfer learning, essentially taking well known models trained on ImageNet and either training them ourselves if we have the resources or freezing the layers and working with the pretrained weights.

5.1 Baseline model

To get a good starting point for our problem we decided to go with a pre-trained resnet50 on the ImageNet data set. We first freeze the layers of the model and then add a last dense layer to perform the classification, this approach minimizes the parameters to update thus results in a fast training loop. Resnet was developed by Microsoft researchers [1] in 2015, the idea behind residual neural networks is to use skip connections over some layers that contain nonlinear activation functions with batch normalization. In figure 4 you can see the residual counterpart of the network in the middle compared with a VGG-19 network.
In the case of resnet50, the model has 50 hidden layers with a total of 24,689,126 parameters.

To train the base line model we used Adam optimizer with no weight decay alongside a categorical cross entropy loss function to train the model. When exploring models later in the project we will also compare Adam to other optimizers using different hyper parameters.For reference the original resnet was trained with SGD having 0.1 learning, 0.9 momentum and 0.0001 weight decay. The figure 5 below show the loss and accuracy of both training and validation sets during training. Training a CNN in general requires a lot of parameter tuning, as we know neural networks are prone to over-fitting and in this case we end up with a model that has more parameters than input sample dimension. It is then capable of learning the training dataset by hear which happened when we fit the base model.

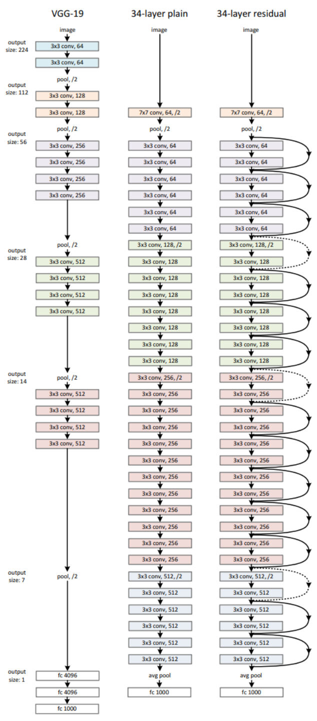


Figure 5: Residual neural network architecture. Figure form [1]

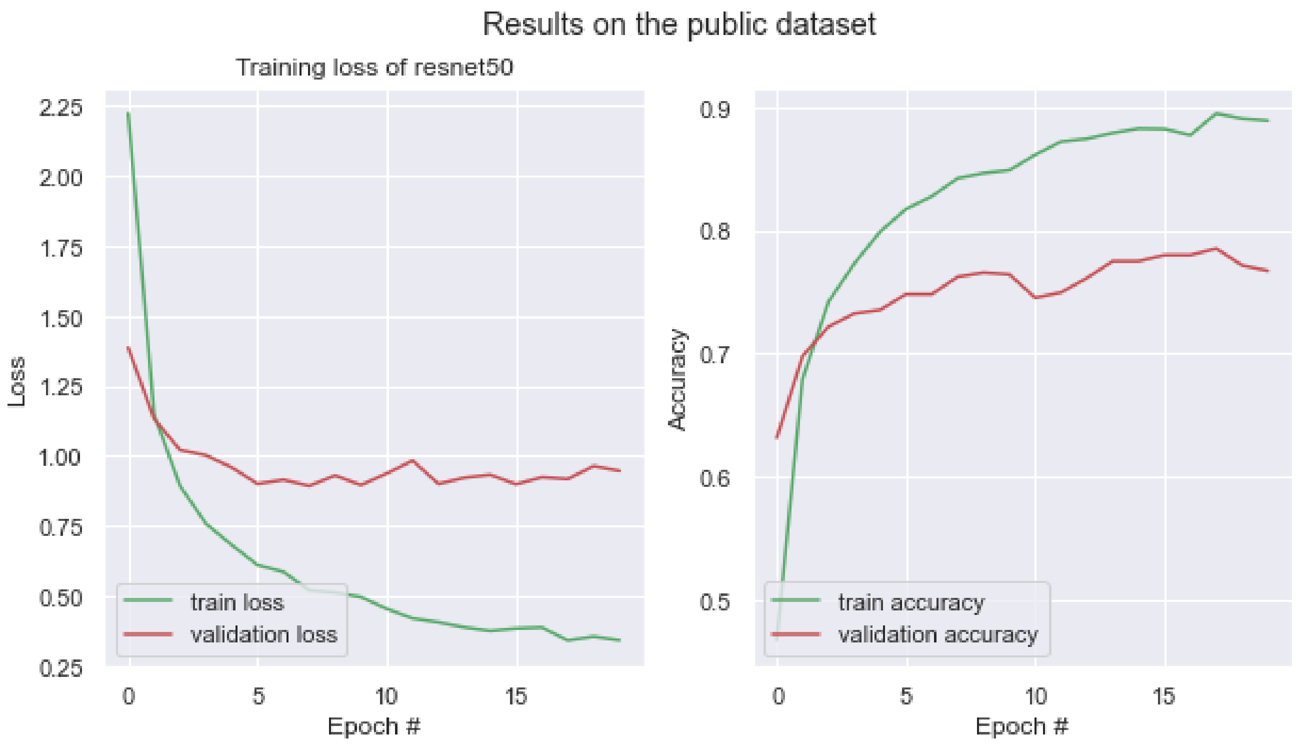


Figure 6: Loss and accuracy plot during training on the public data set

The above plots show the training loss and accuracy on both the training set and validation set for the baseline model. We can see in both plots the training loss and accuracy decreases whereas the validation loss and accuracy decreases until a certain point where it plateau this is a sign that the model is over-fitting. The use of techniques like data augmentation, weight decay and dropout layers may help reduce the model's ability of learning the data set by heart.

5.2 Transfer learning

An interesting idea will be to explore different state of the art models trained on data sets like ImageNet, for this part we will take a look at the best performing models that you can find here `https://paperswithcode.com/sota/image-classification-on-imagenet`. For this project here is the list of the models that we tried:

- ResNet50
- Xception
- DenseNet
- NASNet
- Vgg16
- Vgg19
- ResNeXt

One short coming of these models is that they were typically designed taking into account the fact that millions of data samples were available which makes them overfit fast for example the leading network in the ImageNet dashboard, ModelSoups (VIT-G/14) has 1843M parameters. To train the models we proceeded with two methods :

- Freezing the layers of the pretrained network and training only the classifier.
- Training the whole model from scratch, this was done for networks that showed potential but failed to converge or overfitted the training set.

6 Results

6.1 Bootstrap f1_score

To have a confidence interval on the score of the models we chose to go with the bootstrap method. We will sample pairs from both the predictions made by the model and the gold labels while respecting the indices and then compute the f1_score on the samples. We will repeat this process N times, in our case the N will be 100, and deduce a confidence interval (a 95% confidence interval) from the list of scores. Ideally we wanted to perform cross validation on the models but the process is computationally expensive especially for large neural networks, bootstrap will allow us to have a good estimation of the model performance. Below is the implementation of the bootstrap method discussed above :

```python
import scipy.stats as st
from sklearn.metrics import f1_score

def bootstrap_score(predictions, gold, it = 100, alpha = 0.95):
  scores = []
  for _ in range(it):
  bootstrap_sample_index = np.random.choice(predictions, size= predictions.shape)
  sample_score = f1_score(gold[bootstrap_sample_index],
        predictions[bootstrap_sample_index],
          average='weighted')
  scores.append(sample_score)
  scores = np.array(scores)
  conf_interval = st.t.interval(alpha, len(scores)-1, loc=np.mean(scores), scale=st.sem(scores))
  print(conf_interval)
  print('bootstrap score for model ', scores.mean(),'+/- ', scores.mean() - conf_interval[0])
  return conf_interval
```

6.2 Results on the public data set

Here below we summarize the results we obtained on the public data set:

model	da	F1_score	Bt F1_score	validation accuracy	training accuracy
Resnet 50	No	0.7987	0.7923 +/- 0.00166	0.7959	0.9803
Resnet 50	Yes	0.7903	0.8460 +/- 0.00171	0.7954	0.9201
Xception	No	0.7583	0.7094 +/- 0.00187	0.7583	0.9531
Xception	Yes	0.9311	0.9063 +/- 0.00112	0.9638	0.9360
Vgg16	Yes	0.8482	0.7355 +/- 0.0020	0.8491	0.93258
Vgg19	Yes	0.6674	0.6186 +/- 0.00205	0.6674	0.7573
DenseNet121	Yes	0.53697	0.4404 +/- 0.00216	0.5828	0.4478

Table 1: Results obtained on the training dataset

*bt : bootstrapped *da : data augmentation

Most of the models have no problem learning training set with the exception of Densenet and VGG, this is due to how computationally expensive is to train these models. Since training was done locally these models took longer than others to converge and surpassed the time limits we set for training so we didn't train until convergence. For the baseline model we obtained an initial bootstrapped f1-score of 0.7923 however if we take a look at the train set accuracy 0.98 we can clearly see that the model was over-fitting. When we introduced data augmentation although the model obtained less training and validation accuracy compared to the previous version the bootstrapped score was 5% better. This also translated to the Xception model where we managed to get 20% better bt f1_score with data augmentation, and a 21% higher validation accuracy. We can see that when using data augmentation we can help the model fight over-fitting and prevent the model from learning the train set, so data augmentation can also be seen as a form of regularization. It's also interesting to mention that some models particularly Resnet and Xception were considerably smaller in terms of parameters and fit time which allowed us to train them until convergence, compared to models like Vgg19 (143.7M parameters). We can deduce that number of parameters isn't the only factor when it comes to the score of the model, it is rather the architecture of the model that makes the difference. In a sense, an architecture defines the type of hidden representations that the model learns the better the representations the better the separability of samples will be when fed to the linear layers.

7 Best performing model

During our testing we found that Xception with data augmentation applied to the training set was the best performer out of the list of tested models.

7.1 Xception architecture

Xception stand for extreme inception, the model was developed by Google engineers and inspired by the Inception architecture [4], it relies on two main points :

- Depthwise Separable convolutions
- Shortcuts between Convolutions like shown in Resnet

The model is divided into three parts an entry flow, a middle flow and an exit flow, all convolution and separable convolution layers are followed by batch normalization as shown in the following figure :

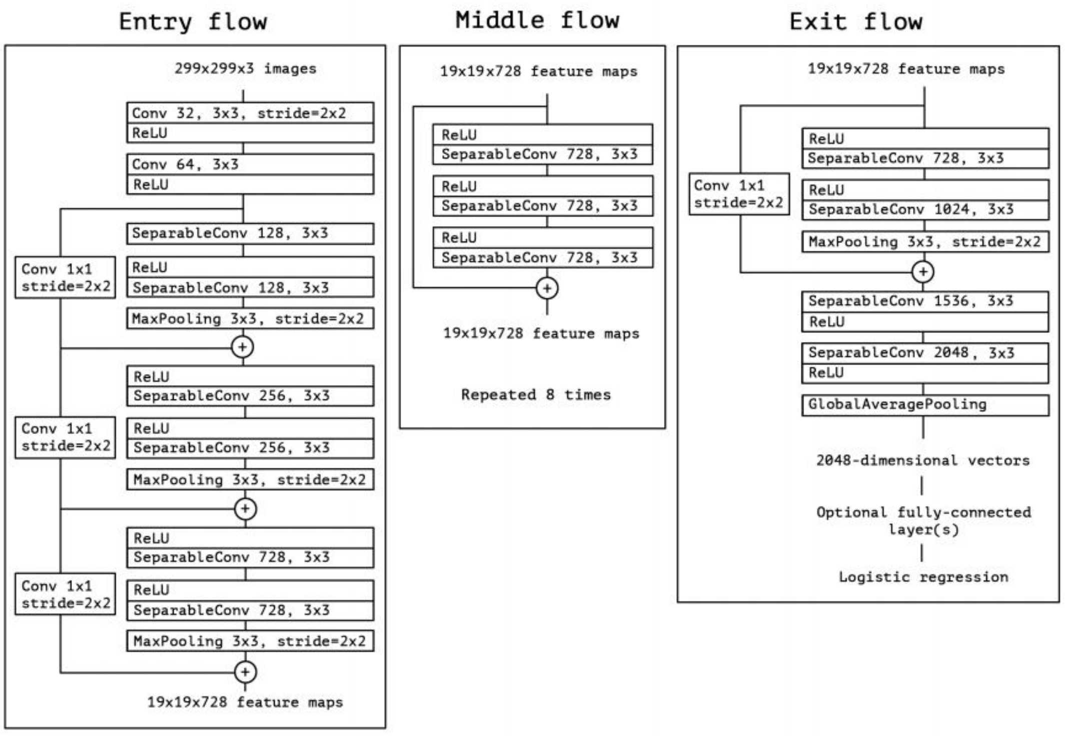


Figure 7: Architecture of Xception

As shown in the figure below we use the Xception model convolutions and then flatten the output of the model and pass it to fully connected layers to perform the classification

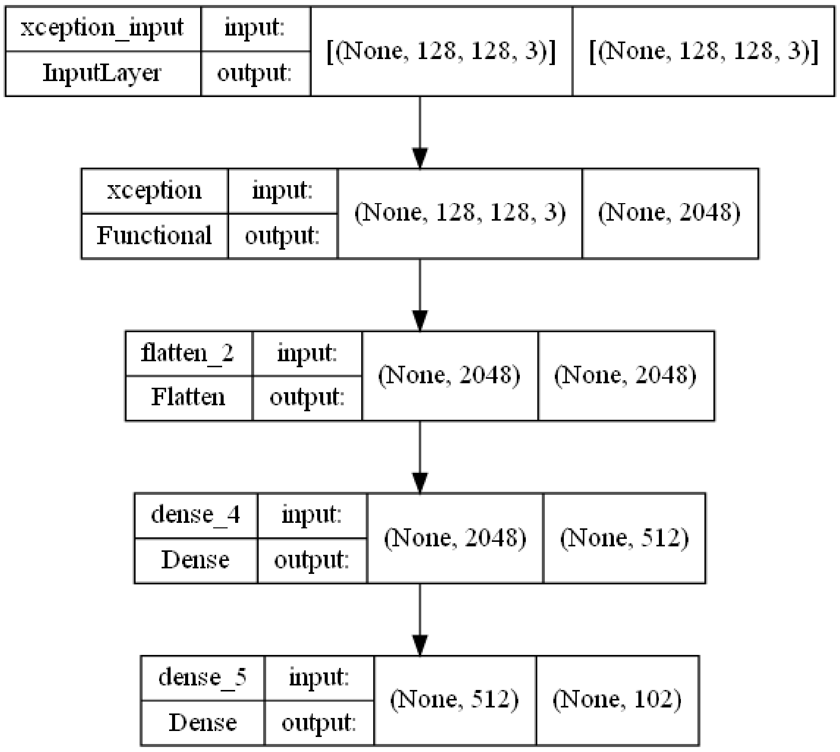


Figure 8: Architecture of the best performing model

7.2 Tuning the best performing model

7.2.1 Adding regularization

To better help the over-fitting of the model, some layers in Xception support regularization so we can manually loop over the layers of the model and when it is possible we add a regularize. We used and L2 regularizer with 0.0001 regularization. We then proceeded to train the model with the same hyper parameters and optimizer in this Adam with a learning rate of 0.0001 we obtained a bootstrapped f1-score of 0.9056+
-0.00126. This is slightly below the initial settings performance, reason for this score is probably that the Xception architecture already implements some sort of regularization techniques like dropout that are probably sufficient to help with the over-fitting of the model.

7.2.2 Layer tuning

Since we're using a pretrained model with image-net weights we won't have control over the convolution blocks of the model, however we do have control over the size and the number of linear layers that will do the classification. To fine tune them we use KerasTuner, it allows to set a search space for the size of a layer e.g (512,2048) and a step size e.g (128). You then specify a method for search, we used hyperband which launches multiple trials with time constraints and takes the most promising settings to do a whole training on them.

7.3 Grid Search

Grid search is a tuning technique that attempts to compute the optimum values of hyper parameters, it uses cross validation on the different parameters defined in the search space. We used this technique to choose an optimizer for the model, our search space was composed of SGD, Adam, RmSprop and Adagrad. For the learning rate the search space was 0.01, 0.001 and 0.0001, our best performing optimizer was Adam with a learning rate of 0.0001 we obtained an average of 0.78 F1_score on the 3 folds we performed. In this experiment we did not use data augmentation since the API of Keras and Sklearn where not compatible.

8 Visualizing feature maps

Deep convolutional neural networks are often referred to as black box algorithms, since we can't exactly see what's happening in the deep layers of the model. However it is still possible to see what the model is outputting in the first layers since the abstraction is not as high as the deep layers. The plots down below show the ouputs of respectively the first and second convolution blocks of the model :

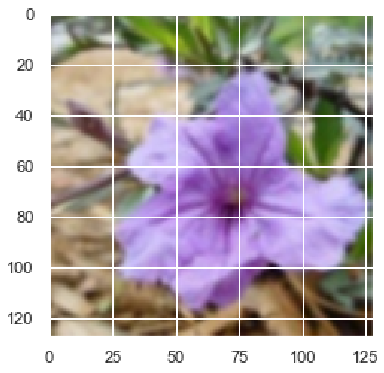


Figure 9: Original image

The original image represents a flower with it's background properly exposed, this will help us identify what are the correlations between the object and the background in the learned feature maps.

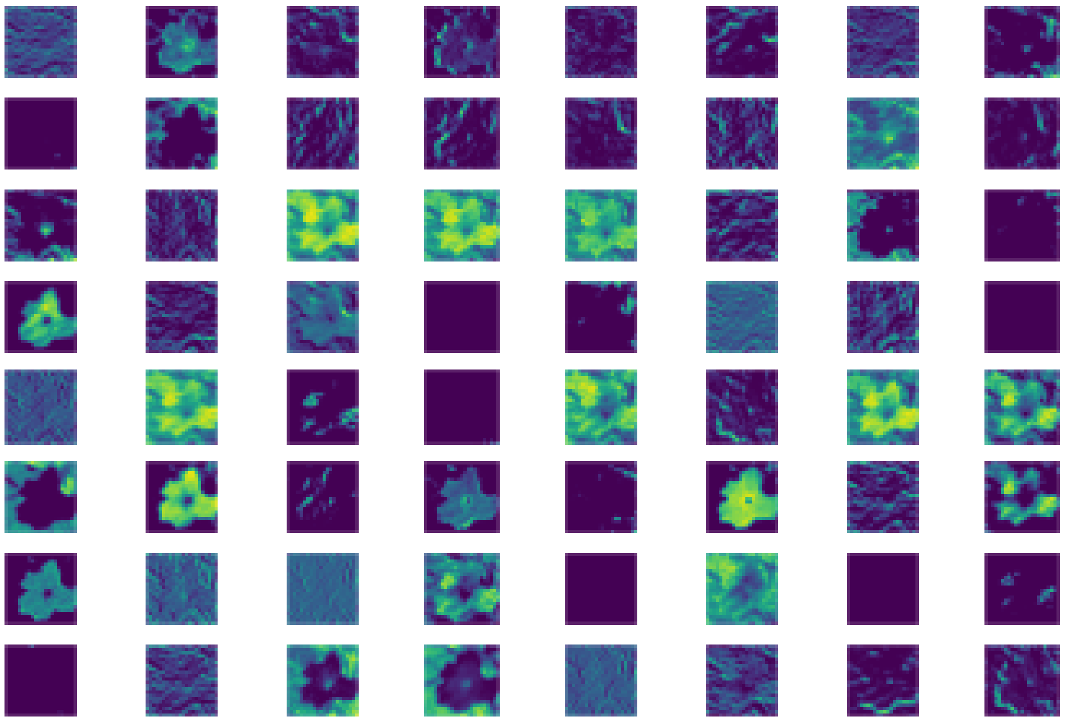


Figure 10: Output of the first convolution block

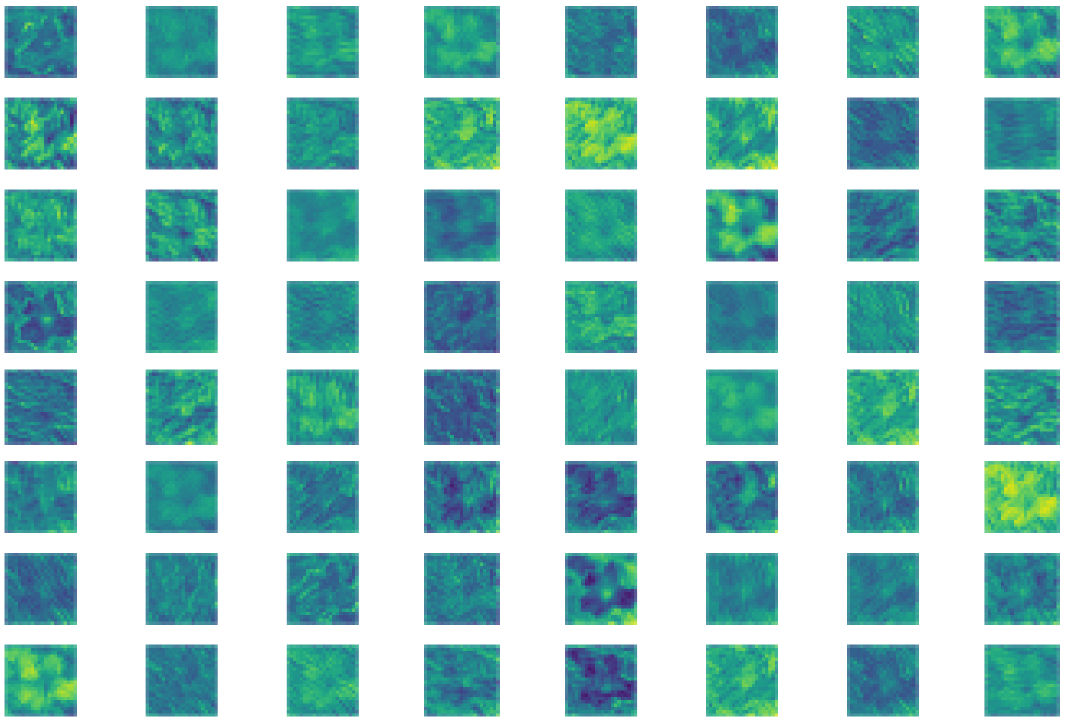


Figure 11: Output of the second convolution block

We can see that the feature maps learned by the first convolution block are easier to interpret than the outputs of the second layer, where in some cases we can distinguish the shape of a flower but not always. Some of the features the model is learning are edge detection, highlighting certain areas in the background ...

9 Conclusion

AutoML has become popular especially in problems similar to classifying living species, because they eliminate the need of field experts, this said automating machine learning in so easy task and a lot of care has to go into designing models that are capable of generalizing well on unseen data. In this project we explored different models and tested their performance using the bootstrap f1_score, although our model has respectable performance some areas that we didn't explore should be addressed like the effect of colour, as we saw in the output of the convolution blocks the model uses various learned feature maps for the classification so enhancing the color information of images by applying different filters to data during the training process could be a an interesting idea.. We can also consider other metrics of evaluating models those that we didn't have enough time to try like cross validation.

References

[1] Kaiming He et al. *Deep Residual Learning for Image Recognition.* 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems.* Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[3] Maria-Elena Nilsback and Andrew Zisserman. "Automated Flower Classification over a Large Number of Classes". In: *2008 Sixth Indian Conference on Computer Vision, Graphics Image Processing.* 2008, pp. 722–729. DOI: 10.1109/ICVGIP.2008.47.

[4] Christian Szegedy et al. *Going Deeper with Convolutions.* DOI: 10.48550/ARXIV.1409.4842. URL: https://arxiv.org/abs/1409.4842.

[5] Hongyi Zhang et al. "mixup: Beyond Empirical Risk Minimization". In: *International Conference on Learning Representations.* 2018. URL: https://openreview.net/forum?id=r1Ddp1-Rb.