Delft University of Technology

Recognizing Flowers - A Neural Network Approach

Assignment 2 - EE4685 Machine Learning, a Bayesian Perspective

04/04/2022

Authors:
David Gomes - 5624940
Pablo de Anta - 5616352

Supervised by:
Dr.Ir. Justin Dauwels
Dr. Hadi Jamali-Rad

EEMCS Faculty

TU Delft

1 Introduction

One of the fundamental problems of machine learning has always been classification. Classification is essentially an exercise of extraction of features from a data set, which for our assignment, it will be images. Based on that feature extraction, the algorithm then tries to assign such features to new data sets and ultimately, distinguish one from another. We humans do this every time with almost everything we interact with. But machines have to come up with this information solely from data. In this assignment, our algorithms will have to be able to classify flower pictures into the five categories of flowers that are present: daisies, dandelions, roses, sunflowers and tulips.

This problem has a certain degree of complexity. Pictures from the flowers can come in all shapes, angles, sizes and there might even be color variation. Thus, the features that are extracted and chosen to be indicative of a certain class, need to be robust enough such that they can be used in all possible scenarios. Coming up with what features should be chosen and how should they be ordered hierarchically to proceed with the classification is what will ultimately decided how accurate a certain algorithm is.

When faced to solve this problem, practitioners from all fields have found Neural Networks (especially Convolutional ones) to be one of the most reliable kind of algorithms [1]. Allowing the machine to extract features with little supervision has proven to be successful, but it is still not ideal. Such processes present one of the biggest challenges for AI today: explainability [2]. Leaving the machine to come up with its own criteria behind the classification can be troublesome when assets are at risk and it is imperative to know why the algorithm took a decision. Alternatives have appeared, with Bayesian Machine Learning leading the charge. Still, it is possible that trade-offs have to be made between performance, explainability and the complexity of the algorithm.

To further explore this topic, in this assignment we will experiment with different algorithms, both non-Bayesian and Bayesian, analyze their performance and discuss the reasons behind this possible difference in results.

2 Motivation

Feature recognition in images is probably one of the biggest application fields of AI. More and more applications are relying on algorithms to detect structures in images that may guide decision making. A clear example of this are autonomous machines. Any autonomous machine that needs to navigate its environment safely shall have some kind of input from the real world which it will use to avoid obstacles or detect targets. A cheap and reliable option so far has been common cameras, powered up with the help of Artificial Intelligence. This is the case of autonomous vehicles, such as the Tesla Model 3, which uses a powerful deep neural network to be able to identify obstacles and structures in the road [3], and the case for autonomous drones surveying high risk areas, such as petrochemical plants, where they need to identify possible structural damages while navigating a complex 3D space [4].

Regardless of current interest and craze, it is important to understand why AI image classification is still a relevant topic nowadays. First of all, although great strides have been made with some of the examples above, there are still more applications to be implemented and algorithms should be made robust to unknown situations - this is easy to see in the case of autonomous vehicles. Even if we were to accept that these algorithms are good enough and take into consideration the fact that many "pre-cooked" algorithms capable of solving our challenge already exist, it is still interesting to perform these exercises. This is because the interest does not really lie in developing optimal performance algorithms, but more so because of the discussion and comparison between solutions, and ultimately our improvement of understanding the functioning behind Bayesian models.

Most current Neural Network models act as black-boxes, proving difficult to obtain information behind the reasoning that lead the algorithm to make the decision it made [5]. Other alternatives have been developed, such as Bayesian Neural Networks, which allow programmers to understand better what is happening. But these are not as widely implemented in current day application despite the fact they are desirable. Here lies the core of this project. We aim to play around with different algorithms on this data set, and compare their outcomes. It is okay if accuracy and precision are not as high as "industry standards", as long as there can be reasoning as to why this might be. By pushing research forward the implementation of the better algorithms is ought to increase, benefiting all of society.

On this note, we would like to end this chapter with a personal reason for us. As prospective engineers that will most likely interact with AI in their professional careers, it is paramount to us to be aware of the current state of the art and what alternatives are there. We chose the topic of Flower Recognition because it implied to work with Neural Networks, a topic that is fascinating to both of us, as well as it was both approachable conceptually and challenging technically. We aim to finish this project being more aware of the impact different Neural Networks have on the final results of an algorithm and specially compare some Bayesian approaches of our own to see whether we can get as good of a performance, or even better, than regular Neural Networks.

3 Proposed Methodology

3.1 Preparing Data

In order to make sure our algorithms get the best training experience, the data was prepared to make sure the algorithm could work with it properly. All the steps allowed for algorithms to be robust while avoiding over fitting. In this section, we will go over the main steps followed in the algorithm and explain the reasoning behind some more elaborate or optional steps.

As with any project, we began by importing all the relevant resources and data sets. Most importantly, with the data set imported, we started by first analyzing how did the different classes compare in terms of data points out of the total amount of 4317 images. As it can be observed in Figure 1, although they are quite even, it can be seen that dandelion and tulip classes have approximately 1000 images each, while the other three classes, daisy, rose and sunflower are around 750 images. Out of them five, sunflower has the least amount of data and dandelion the most. This could affect the performance of our algorithm, so to try to reduce its bias and balance this difference, we will proceed to augment our data set.

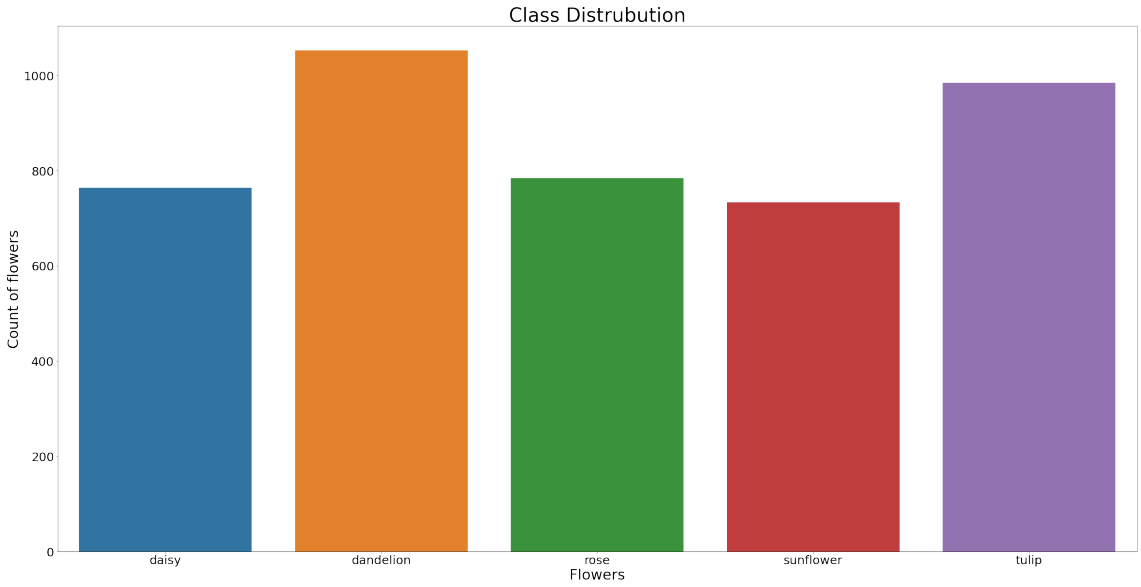

Figure 1: Class distribution found in the original image data set.

But before augmenting the set, we also want to make sure that the images are comparable between themselves. We first start by displaying random pictures and evaluating their pixel intensity profiles. In the example shown in Figure 2, it is clear that these profiles are different and that the size of the images are different as well. In order to allow the algorithm to operate more smoothly, some modifications to the data set need to be done aiming to have less difference of values, becoming more accurate. To achieve this, we proceed to resize all pictures to the average size of 220x220 pixels and to normalize pixel intensity. By normalizing the pixel intensity, we are getting our data to all display values in the same range, reducing how skewed it is. This then helps the algorithm learn fast and better. Another added benefit of normalizing data, is that it can also help with diminishing and exploding gradients.

Having all of our set displaying data points with common characteristics, it is now the moment to augment our data, as previously mentioned. If we were to have done this before the normalization of data, and this step after the augmentation, the computational cost would be much higher. Hence, this order is preferred. To augment a data set with images, it was decided to go with transformations of the current data set. In order to have different augmented data sets to compare performance between such, there were a few transformations done and several combinations of these. We applied simple transformations consisting of random image flips, rotations and affine movements, plus there were a total of three pre-built transformation methods from Pytorch chosen: CIFAR10, IMAGENET and SVHN. They each reference an actual image database from which said transformations are implemented. Because we lacked big amounts of data points to make the algorithm robust enough, we thought of implementing these methods, at the expense of future computational cost.

With our data set complete, the next step is to divide it between training data, testing data and validation data. We decided to include validation data because after the previous steps we made sure our data was as big and reliable as possible. By implementing a validation test and using it in the learning procedure, we could prevent possible over-fitting and loss of accuracy in our algorithm by implementing early stopping (in practice, we would run the models for 15

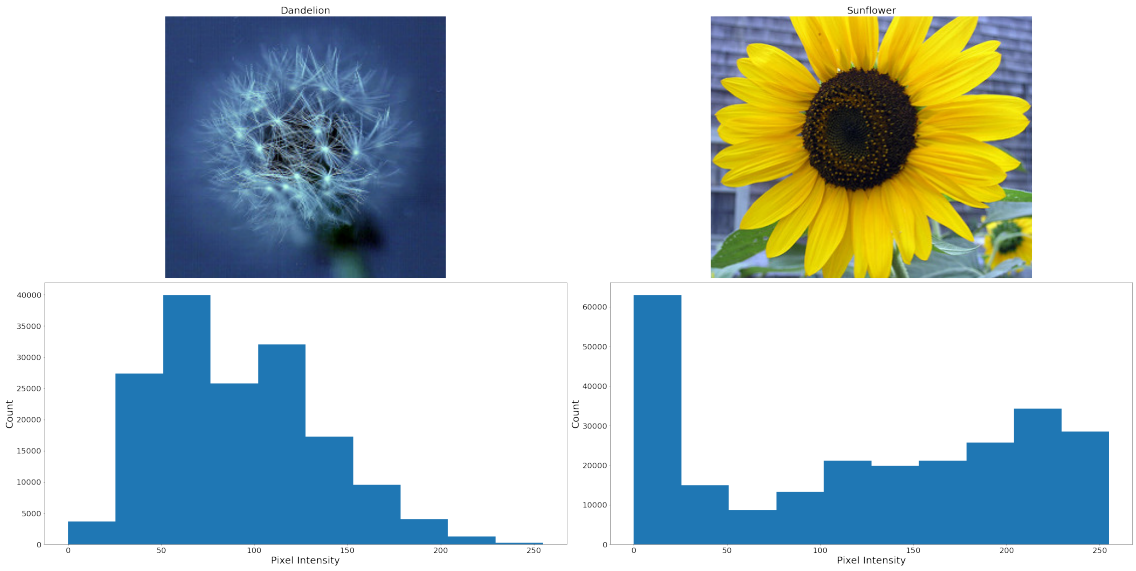

Figure 2: Two random images extracted from the original data set. Their class is displayed above. Below, a profile with their pixel intensities is displayed.

epochs and choose the one with the best accuracy in the validation set). The splitting was also done taking into account the fact that not all data was equal, as some was original and other was augmented, but for the validation set it was decided to not use augmented data, as it did not resemble really the real life scenarios that the algorithm could face. From all the data given, we split it 72% for training, 8% for validation and 20% for testing. Furthermore, all the data augmentations only influenced the training set.

3.2 Training the algorithms - Non Bayesian Approach

Having prepared our data, we started to move onto training our Convolutional Neural Networks (CNNs) of choice. We started by trying to tackle the problem with the non-Bayesian algorithms, and specifically, using Resnet18. We chose Resnet18 because of its simplicity, since this first step was only aiming to check whether we were on a good track or not. Having only 18 layers makes it cheap for us to compute one epoch and tune parameters until we were happy with its initial performance. Once this performance of Resnet18 was examined and we knew the code worked, we moved onto comparing the performance of other CNNs that were available to us.

The ResNets [6] are a widely known family in the deep learning realm, usually identified with a number corresponding to the ammount of layers it possesses (eg. ResNet18). Furthermore, ResNets are convolutional neural networks (CNNs), that is, most of the layers perform a convolution operation that consists in a moving kernel mapping a number of nearby pixels as the input of the next layer. By doing this, the network can learn positional information since with each convolutional layer applied the information of nearby pixels is propagated reaching more and more pixels the more depth the network has. ResNets also possess Max Pooling layers, that extract the highest pixel value out of all the ones considered (given the kernel size). By doing this, the network can extract features from images, such as edges or shapes. In the end, ResNets have a Average Pooling layer (same as max pooling but regarding the average value of the considered pixels) and a fully connected layer that performs the classification with a softmax activation function.

The true strenght of ResNets lies on the constant sums between the layer's input and the layer's output, having this residual information propagated, hence the name. This characteristic helps to protect the network from the vanishing gradient problem. Another important aspect of ResNets is batch normalization. Picking up the idea of normalizing the input data, the same can be done for every layer in the network. However, the normalization here is based on the samples in each batch. This allows for a faster and more robust convergence of the NN parameters as well as some regularization factor, contradicting over-fitting.

To aid the network in the first steps of its learning, while parameters and features were being optimized while developing the nets, we chose to couple Resnet18 with Transfer Learning. Transfer Learning is a technique that aims to take advantage of a pre-trained network in somewhat similar features to those of interest in a given project, and then re-train that network in the new set of features that are relevant in the new project. The pre-trained NN are trained in huge datasets like IMAGENET, offering a good generalization for feature extracting. This is particularly good in our case, since we do not have much data to work with, utilizing a pre-trained network can immensely boost the performance of a NN. In our case, PyTorch offers a range of pretained models, as previoulsy said, we chose ResNet18.

We kept all the layers except the last one regarding classification. The structure that was maintained from the previous algorithm referred to feature extraction of the images, while the last layer implemented by us allowed the algorithm to finally classify images in the five flower classes we had in this project. We then froze all the parameters corresponding to the feauture extracting and only optimized the last classification layer based on the training samples. Having this aid, we decided to run Resnet18 for a few epochs and check how well it performs on both the augmented set and the original set, as thanks to the transfer learning the algorithm can converge much faster. We also run our network without the Transfer Learning, although it increased this step's computational cost, to check if there was a significant difference in performance. These initial steps allow us to check if all the work done up until now and the reasoning behind it is correct. Results should be acceptable due to all the previous data preparation. Once it is looking like it is a promising approach, we will move on to increase the number of epochs and also the types of networks we are training. To do this, the performance of the network is analysed by plotting the training set and validation set average losses and accuracy of each epoch.

We have mentioned as well that this iterative process of running single epochs was to both check performance and fine tune parameters. Specifically, these parameters that were being tuned were mainly the learning rates. We started using standard values, and we stopped making them more complex as soon as we were able to see good performance. Once satisfied, we moved on to using higher number of epochs.

One common factor we implemented between all the CNNs was the optimality criterion. For simplicity and how used we were to using these tools, we decided to use Cross Entropy Loss, Eq,1, as the criterion because of its often use in classification tasks and ADAMw as the optimizer. ADAMw was chosen because the ADAM algorithm [7] was recommended by professor Justin Dauwels since it is a common of-the-shelf algorithm, and when researching the subject, Pytorch's website indicated that the revision of that algorithm, ADAMw, gave even better results. Based on the Stochastic Gradient Descent (SGD), ADAM takes into account the direction associated with the change of loss regarding the NN parameters in each batch, and performs two operations considering all the previous directions taken: diminishes the direction variance and gives "momentum" to the average direction taken.

$$Loss = -\frac{1}{output\,size} \sum_{i=1}^{output\,size} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i) \tag{1}$$

Finally, we generated a confusion matrix out of one of the nets to see how well did it classify images and if there were any apparent biases towards certain classes.

3.3 Training the algorithms - Bayesian Approach

Bayesian networks work based on the assumption that the parameters that regular networks try to estimate are not deterministic but rather have a stochastic behaviour. Such distributions are used in the new estimation of parameters and the network is then able to predict outcomes based on statistical matching [8]. Each network functions slightly differently, but they are all based on Bayes equation [Equation 2] and on Variational Inference methods [9]. This last method allows us to move from an inference type of problem - with sometimes impossible to compute scales - to an optimization problem - which can be more easily calculated. Optimizing the loss between data and predictions allows the algorithm to find values for the parameters that best match data and create an output.

$$P(\theta|\mathbf{D}) = \frac{P(\mathbf{D}|\theta) * P(\theta)}{\int P(\mathbf{D}|\theta) * P(\theta) d\theta} \tag{2}$$

Before implementing a fully Bayesian Neural Network, we wanted to try to experiment and implement a regular Neural Network and substitute the last layer by a Bayesian classifier. This was done to see if performance dropped when implementing Bayesian methodologies or if cost increased. The method chosen was the Naive Bayes algorithm. To implement it, we used the pre-trained CNN GoogleNet, and removed the last classification layer. By doing this, we let the CNN extract the features from each image (80 features), and fed them into Naive Bayes, leaving the classification to the Bayesian algorithm.

Secondly, we moved onto implementing a fully Bayesian Neural Network. Bayesian Neural Networks have gain popularity in the last years, because they will come up with different results each time an image is fed, they present an important characteristic that traditional Neural Networks do not have: degrees of uncertainty. When a Neural Network is trained, it keeps changing the values of the weights and biases, but they are always a fixed number that does not change throughout the process. When tackling problems following a Bayesian methodology, this does not apply. The point is not to come up with a concrete value for those weights, but rather the distribution behind these parameters. What this implies is that for each weight/bias there is in the network, there are two additional and associated parameters (if the chosen distribution is chosen to be a Gaussian), the mean and the standard deviation. This has direct consequences,

such as the number of parameters doubling, resulting in a higher computational cost. Keeping this in mind, we decided to only assume that the weights between neurons were distributions, leaving the biases as numbers. This allowed to cap how many computations we would have to perform.

Furthermore, since Bayesian Neural Network can get costly quickly, it would not be feasible to implement this approach to deep NNs (at least, in our case) [10]. The considered approach was to then construct a rather simpler network but in a Bayesian way. The proposed architecture was that of a ResNet34, but with a Bayesian classification layer at the end. We also built a comparable regular ResNet34 in order to be able to assess performance with a similar net. If we were to use the previous architectures, it would not be fair to compare their performances as they operate differently. Another particularity we introduced was the last layer, (the classifier) as instead of it being a normal linear layer, we made it have the weights and biases work as Gaussian distributions sampled in each time, using both the kernel divergence with the variational inference. Obviously, this comes at cost, mainly the computational time. Finally, in order to be more confident in the final prediction of the network, we made it predict 50 times each image and then average the results. The deviation from this process could also be used to give insight as to how certain can one be of the outcome.

4 Results

4.1 Preparing Data

After seeing how distinct values for size and pixel intensity can be between images, and to ensure the algorithms work correctly, we proceeded to examine how the images compared after normalization. Two random images and their corresponding values can be seen in Figure 3

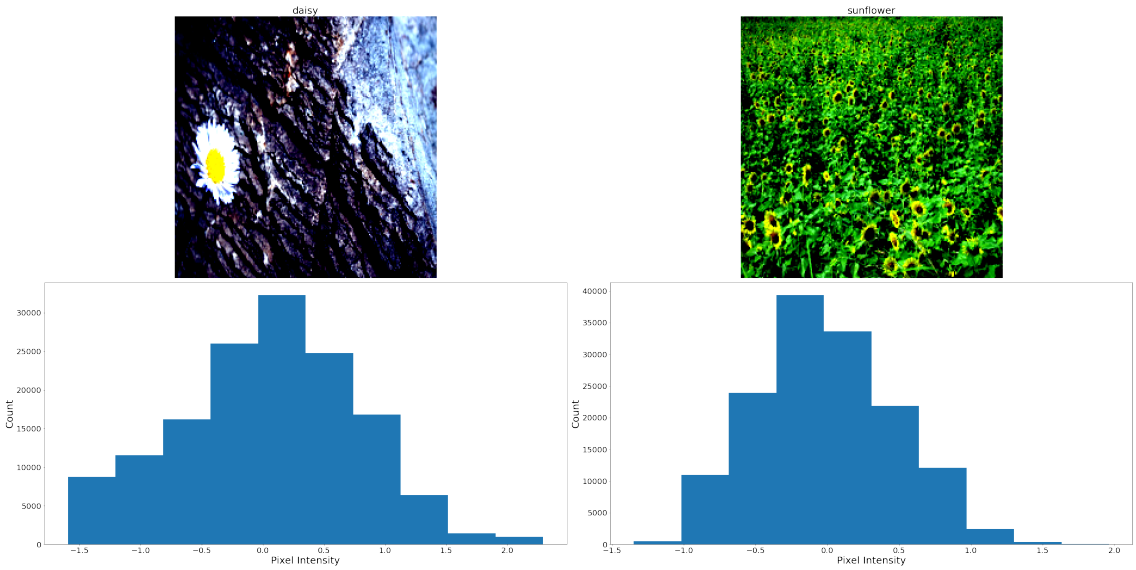

Figure 3: Two random images extracted from the original data set. Their class is displayed above. Below, a profile with their normalized pixel intensities is displayed.

The next key step is data augmentation. In Figure 4, augmented images and their corresponding classification can be observed. From left to right, the first image corresponds to the set augmented with CIFAR10, the second image corresponds to the set augmented with IMAGENET and the third image corresponds to the set augmented with SVHN.

4.2 Training the algorithms - Non Bayesian Approach

When initially training the ResNet18 under different circumstances, we obtained different values for its performance based on what version of ResNet18 was run. In Table 1, values for accuracy can be observed for each way we run the network, with 15 epochs and a batch size of 32. To understand the table, when referring to "Data Augmentation", the training data is augmented with the CIFAR10, IMAGENET, and SVHN augmentations, when referring to "Simple Data Augmentation", we applied rotations, image flips, collor jitter, random erasing and affine translations, and when referring to "One Data Augmentation", we only used rotations, flips and affine translations. In Table 2 we can see similar results for similarly trained ResNet101's.

Figure 4: Three images extracted each from an augmented set. From left to right, the images belong to the augmentation methods of CIFAR10, IMAGENET and SVHN. Their class is displayed above.

Algorithm and training conditions used	**Accuracy Obtained**	**Time Taken**
ResNet18 with raw images & Transfer Learning	90.55%	5m 17s
ResNet18 with Normalized Data & Transfer Learning	90.67%	4m 50s
ResNet18 with Data Augmentation & Transfer Learning	85.25%	12m 44s
ResNet18 with Simple Data Augmentation & Transfer Learning	71.77%	12m 25s
ResNet18 with One Data Augmentation & Transfer Learning	89.29%	8m 6s
ResNet18 with Normalized Data Augmentation & without Transfer Learning	80.41%	4m 29s

Table 1: Table with the accuracy of ResNet18 under different scenarios

We would like to remark that these results were obtained when running on a machine powered by a 32 GB DDR4 RAM, a 2667 MHz Intel(R) Core(TM) i7-8750H CPU @ 220GHz unit, and an NVIDA GeForce RTX 2080 with Max-Q Design unit for GPU.

The confusion matrix we acquired for the ResNet101 can be seen in Figure 5.

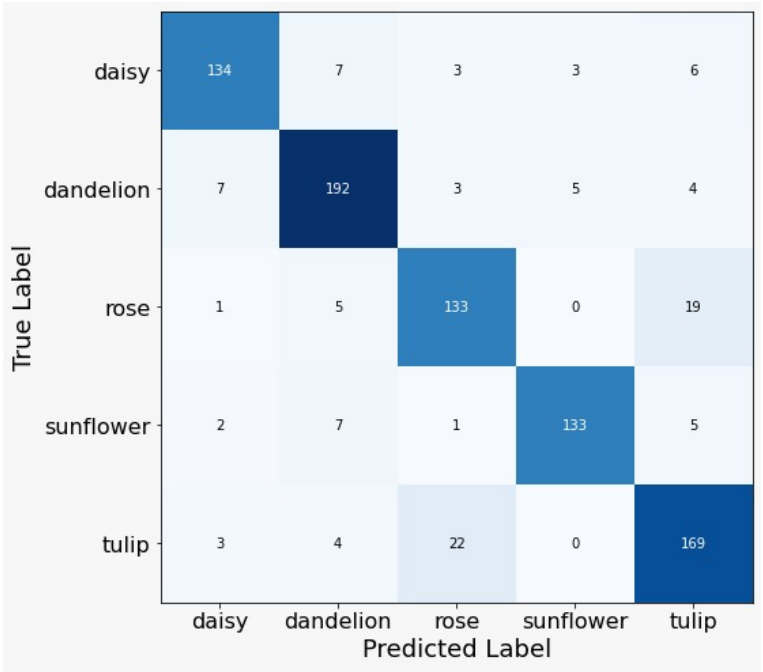

Figure 5: Confusion Matrix from the results we obtained after running ResNet101.

4.3 Training the algorithms - Bayesian Approach

For the first exercise, where a Naive Bayes layer was added at the end of a conventional GoogleNet neural network, we obtained the results that can be seen in Table 3. We also computed its confusion matrix, which can be seen in Figure 6.

Algorithm and training conditions used	Accuracy Obtained	Time Taken
ResNet101 with Normalized Data & Transfer Learning	86.52%	19m 39s
ResNet101 with Data Augmentation & Transfer Learning	72.47%	63m 53s
ResNet101 with One Data Augmentation & Transfer Learning	87.67%	31m 48s

Table 2: Table with the accuracy of ResNet101 under different scenarios

Algorithm Used - Batch Size: 32 | Epochs: 15	Accuracy Obtained	Time Taken
Naive Bayes Layer	31.11%	0m 1s

Table 3: Table with performance and computational time of the Naive Bayes layer we implemented

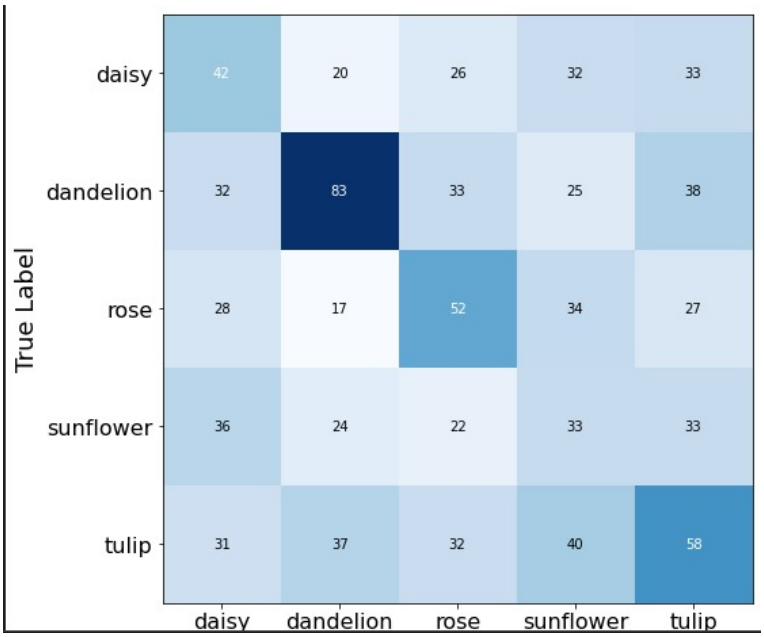

Figure 6: Confusion Matrix for the Naive Bayes architecture built using GoogleNet and a Bayesian Classification Layer.

When running the fully Bayesian Neural Network based on the ResNet34 architecture and comparing its performance to a regular ResNet34 net, we obtained the following results. Accuracy, loss and computational time can be seen in Table 5, and a comparison of how the nets perform at each epoch can be seen in Figure 7 and in Figure 8.

Algorithm Used	Test Accuracy	Test Loss	Time Taken
Bayesian ResNet34 Architecture	66.70%	2.0093	120m 32s
Non-Bayesian ResNet34 Architecture	57.95%	1.7841	20m 54s

Table 4: Table with performance and computational times for a ResNet34 with a Bayesian classification layer and a regular ResNet34

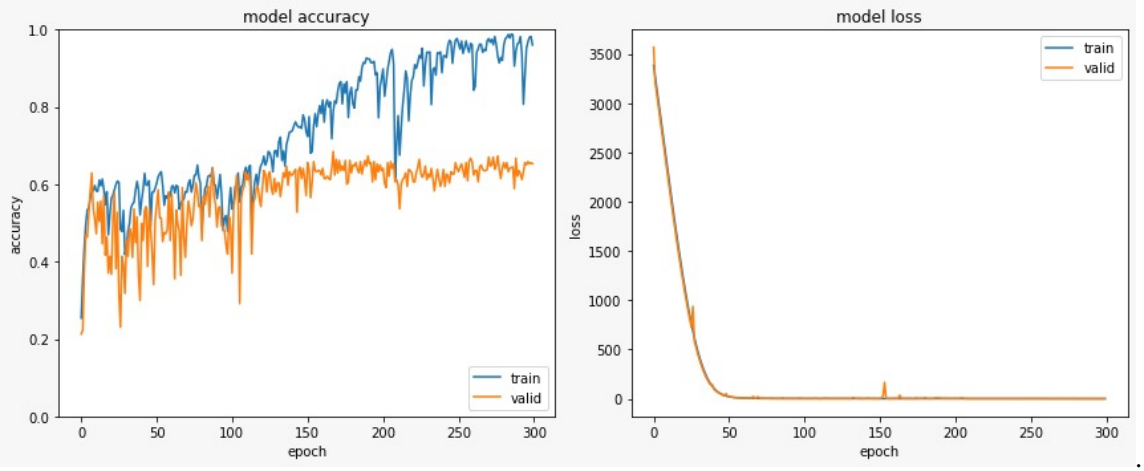

Figure 7: Accuracy and Loss of both the train and validity set in the ResNet34 Bayesian Neural Network across the epochs.

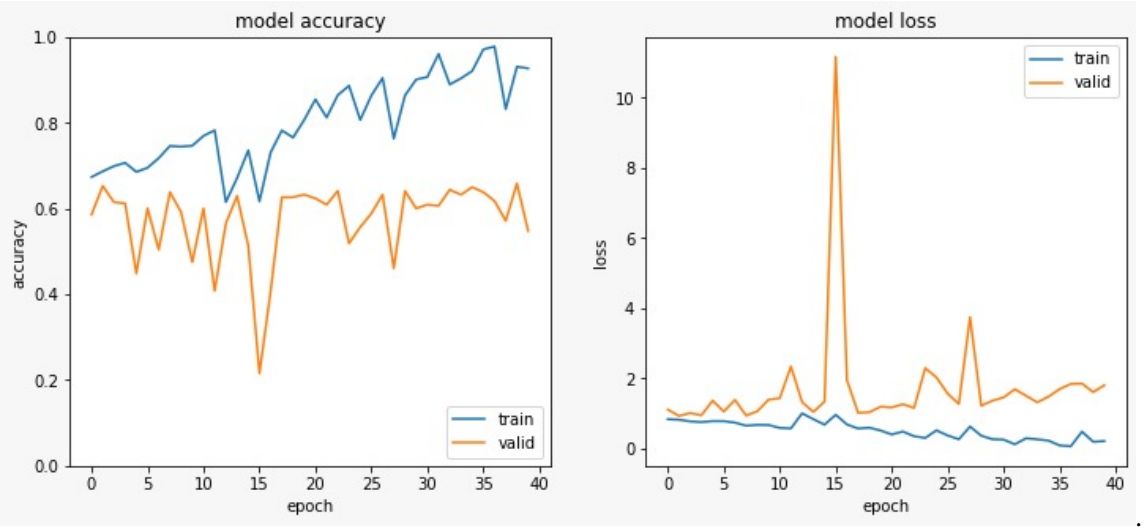

Figure 8: Accuracy and Loss of both the train and validity set in the ResNet34 Neural Network across the epochs.

5 Discussion

5.1 Preparing Data

Although the procedures followed for preparing the data to be fed to the algorithms can be considered quite standard, there are two main ideas that we would like to reflect on.

The first one is to acknowledge the efficacy of the normalization of pixel intensities procedure. We were quite surprised with the fact that when visualizing the values in Figure 3, their distributions actually resemble a normal distribution quite well. From Table 1 it can actually be seen that the benefit from normalizing the images is only substantial. However, by normalizing the images we are also preventing exploding gradients and fastening the training process, with a bigger set the results would be more evident. We believe that the fact that they adopted such a similar profile is one of the main reasons as to why the neural networks were able to work so well and produce such good results. The second idea we would like to point out was the decision of scaling the pictures to a common size, 220x220, this needs to be done since the in the case of a CNN the number of inputs and outputs needs to be hard coded in the NN. We chose this resolution, since most of the pictures had a similar resolution to this and, therefore, would not cause too much of a distortion. We also preferred choosing a relatively smaller resolution (when comparing to the images), since more pixels would result in more inputs leading to a more complex NN, and bigger computational times.

Yet, another issue we encountered was that not all the augmentations we performed were actually positive to train effectively the nets. That is why we thought it would be interesting to run the same network under several augmentation conditions. As we understand the issue, we believe that augmentation can end up training the net excessively on the augmented set, which based on the degree of how different are the images to what could be found in real life, can lead to "overfitting" and a bias towards the augmented set. Usually data augmentation works nicely, but you need enough data

points for the algorithm to not become skewed or begin to overfit, which we believe it is not our case. Were we to have had more data, we guess performances with the augmented set would have been better. And it is also the reason why simpler augmentations produce better results. It can actually be seen from Tables 1 and 2 that all augmentations actually injured the performance for the ResNet18. As for the ResNet101, only the smallest and simplest augmentation (Flips, Rotations, and Affine Translations) produced a slight improvement (about 1%). Data Augmentation can work as a regularization parameter, stopping networks from overfitting to their current training set. An example of this is a dataset that the majority of the flowers are placed in the middle of the picture, this would cause a bias. By shifting the flower's position within the image it would contradicte this behaviour. However, in most of our experiments, data augmentation never lead to a better result. We do think this is because we do not possess a big enough dataset and should have maybe implemented less augmentations and with less data. Furthermore, some of the augmentations applied can be seen as "too strong" distorting the images too much and actually deteriorating the performance.

5.2 Training the algorithms - Non Bayesian Approach

Regarding the Non Bayesian algorithms there are a few ideas we would like to comment on. Firstly, there are some quick conclusions we could draw from the processes we ran. It seemed that algorithms benefited partially from augmenting the data set giving them more points to train and test themselves. We already argued in the previous point what are the repercussions of augmenting data in conditions like ours. The other conclusion extracted was that the networks greatly benefited from applying Transfer Learning. Not only was accuracy reaching higher values (90.55% in a pre-trained ResNet18 vs 80.41% for the same network without pre-training), but the training time remained manageable for our machine for the ResNet18 dealing with the augmented data set. Furthermore, it can be seen that ResNet34 used to compare the Bayesian NN performs worse without transfer learning than ResNet18 (57.95% against 80.41%). This relates to the fact that ResNet18 is less complex and has less parameters to optimize. Therefore, with the given data ResNet18 actually manages to permorm better. The same might not have ocurred if we compared both pretrained versions.

The other main idea we wanted to discuss was the range in performance of the different algorithms. More specifically, why were there such big variations in the computing time and why was it that simple networks were able to compete with complex ones. This is the case when comparing the results between the ResNet18 and the ResNet101. Our main hypothesis here was that it was caused by the complexity of our data. Because we do not have a huge number of data points, even after augmentation, complex networks struggle to obtain sufficient information to run the network effectively, ending up overfitting on the training data. Thus, simple networks are better tailored for this problem and run better. This is a good example of how there might not be a single best algorithm for all scenarios, but rather specific cases where one might excel where the others do not.

Finally we would like to comment on the Confusion Matrix we obtained. We found interesting where the main confusions happened, particularly between roses and tulips. One reason the group though as to why this is happening, is because in some perspectives roses and tulips can indeed look similar, out of the five flowers they can be considered the more similar ones. Also, these cases follow the same structure, where one of the classes with the highest number of data points is being predicted as one of the classes with less points and vice versa. It seems that the efforts of normalization could not actually fully eliminate this potential bias, and we can find it represented here. The classes with less data points could probably not be learned as well by the algorithms, leading to these cases of miss-classification.

5.3 Training the algorithms - Bayesian Approach

First of all, we chose the googlenet as our feature extractor, since when experimenting we saw smaller CNNs performed better (since they would overfit less), therefore we tried googlenet and it actually was the NN with the best accuracy, 91,71%, making it the best candidate to extract features. While running the first Naive Bayes experiment, we were aware that it would probably not bring meaningful results, as coupling a Bayesian classifier with features extracted by a CNN would not give any of the benefits we usually look for in a Bayesian algorithm. Still, we wanted to see how well it performed and the results obtained can still be of use. Because the data was initially fed to a GoogleNet, it was able to extract a total of 80 features to classify on, meaning there were 80 dimensions to our problem. For a Naive Bayes to classify effectively with this many dimensions, the number of data points would have had to be way higher than our augmented data set. If we were able to get more data points, probably the accuracy would improve. Nevertheless, we did not follow that path as we knew there were not many more interesting conclusions to be extracted from this procedure. Besides this, the computing time was also worth thinking about. As it is only working in the final classification layer, and it is doing so following a statistical Bayesian method, it is expected that it does not take a substantial amount of time, being all the time related to the feature extraction itself.

Regarding the Confusion Matrix that was generated from the results of the Naive Bayes net, as seen in 6, we see results are not as optimal as desired either. We can use this matrix to see how the evident is the predominance of the dandelion class. It is the class with most correct outcomes and it is as well the class with a higher representation in the data set. It is clear that there is some kind of bias in the net that should be taken care off. Additionally, it is reasonable to affirm that out of the five classes, dandelions are the most distinct flowers. This means that they are easily distinguishable and recognizable.

Moving on to the fully Bayesian Neural Network we built, there are many things to be said. The first impression is clear, the model performs but could use an improvement in terms of accuracy. We believe these levels can be improved if we try to tackle the cause the issue is rooted in, the data. We already mentioned how Bayesian methods require very big amounts of data in order to provide high accuracy results. This can also be seen in our experiments when comparing the accuracy level at a certain epoch we obtain. To be able to obtain similar results to those of a CNN, we need to compute around 7 times as many epochs. If data points were to increase, we would be able to address better all the features that are being extracted, obtaining a more robust algorithm that should perform better. And performance of the algorithm would increase faster in the first epochs, allowing us to maybe not have to train the net for so long. Of course, it comes at the price of computational cost and waiting time. The Bayesian algorithm took enormous quantities of time when compared to the regular ResNet34, but it is understandable when the number of epochs needs to be so high in order to make the net competent.

Yet, there are good things to be learned from the Bayesian Network. The main take away here is that the loss of both the train set and the validation descend hand in hand through the epochs. This cannot be said as strongly about the regular ResNet34 we computed. It can be see that loss remains more or less steady, with a slight tendency to grow over epochs, while the Bayesian loss steadily decreases. What this means to us is that although accuracy is not amazing in the Bayesian Neural Network, it is reliable in terms of dealing with many kinds of data. The degree of reliability of the algorithm when comparing the test set to the validation set also means that in future runs we may even drop completely the validation set in order to put into the learning set to further increase performance.

However, taking everything into account, it has to be said that the major benefit that the Bayesian network provides us with is a measure of the uncertainty. No regular CNN can do so and although more costly, it is a totally new piece of information that should be taken into account when making actual life predictions. Knowing our algorithm is not actually sure about a prediction can be a more critical piece of information than the actual prediction. Final decisions should be taken once considered the level of uncertainty, specially when the repercussions of those are bigger than simply classifying a flower for a university report. Knowing this technology exists and can help us build a reasonable and nuanced future fills us up with excitement and motivation.

6 Conclusions

Regardless of network performance and how good were we at classifying flower pictures, we would like to reflect what were the most important takeaways from this project. First and foremost, we would like to re-state once again how critical it was to prepare data properly. Not only was it necessary as it allowed algorithms to have comparable data points based on which they could extract proper features and classify them, but it was very insightful for us. Seeing how not all augmentations are equal and how they can have both positive and negative effects on the results we obtain made us reflect on what step we take and why. Increasing the data points with not-so-different points gave stability to the networks, while also ensuring that they could work as fast as possible thanks to having normalized information. The second major takeaway was to see how relevant it is to always couple your algorithm to the proposed network. It is clear that a more complex network will not always yield better results, besides having higher computational times. Rather, it is imperative to think of the challenge at hand, understand what tools we have and use that which fits better. In our case, having a rather simple issue to tackle and without a huge amount of points, using the simplest network gave us the best combined result of accuracy and computational time. Lastly, Bayesian networks seem to be very useful, although costly. The fact that loss descends hand in hand between training set and validation set, and the additional piece of information that is uncertainty make them a useful tool despite cost. Although we ought to run these algorithms for very large number of epochs, with a higher data pool and optimized methods, we could reach optimal performance. Still, even when not performing as ideally as conventional networks, they still bring useful information, making them a preferred choice when taking critical decisions in the real world.

7 References

[1] S. Deepa and B. A. Devi, "A survey on artificial intelligence approaches for medical image classification," *Indian Journal of Science and Technology*, vol. 4, 2011.

[2] D. Shin, "The effects of explainability and causability on perception, trust, and acceptance: Implications for explainable ai," *International Journal of Human-Computer Studies*.

[3] Tesla, "Artificial intelligence and autopilot."

[4] Shell, "Digital technologies: Drones, virtual reality and artificial intelligence."

[5] D. Pedreschi, F. Giannotti, R. Guidotti, A. Monreale, S. Ruggieri, and F. Turini, "Meaningful explanations of black box ai decision systems,"

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[8] PureAI, "Researchers explore bayesian neural networks."

[9] J. Buckman, "Bayesian networks need not concentrate."

[10] StackExchange, "What are the downsides of bayesian neural networks."

Pablo de Anta
David Gomes

Recognizing Flowers - A Neural Network Approach

Useful Links

Code

- Github Link:
  - HTTPS: https://github.com/DavidMRGomes/Assignmet2_ML.git
  - SSH: git@github.com:DavidMRGomes/Assignmet2_ML.git

- If Github's link is broken, you may access the code as well in this Google Drive folder:
  - https://drive.google.com/drive/folders/13UOnFRO9rogYLOfcNPLKVIsZJ0oZSaml?usp=sharing

Video

- YouTube Link:
  - https://youtu.be/vowGM9QevA0