Deep Learning: Assignment #2

Alejandro Gómez de Miguel - CIT Student ID: [redacted]

12/05/2020

Introduction

This report is aimed at evaluating the performance of Convolutional Neural Networks and Deep Learning techniques and, additionally, provide a theoretical introduction to a research field that has gained increasing attention during the last decade: Adversarial Machine Learning. In particular, the first two sections of this document discuss the results obtained from applying a range of algorithms and modeling methodologies to a multi-class image recognition problem, while the third part elaborates on the theoretical aspects of the topic selected for research. The database used for experimentation is a pre-processed version of the Flowers-17 dataset. It contains 1020 and 340 RGB images for training and validation, respectively, of 128 pixels wide and 128 pixels high. Each of the images corresponds to a single class among 17 different flower types in the dataset. These have been cropped, normalized, and reshaped in advance, which greatly reduces the preparation time of the data. Please note, along with this report we've provided Python notebooks containing the code of the algorithms. These are written in Python 3 using TensorFlow and Keras high-level APIs. Our experiments were run in Google Collaboratory using GPU hardware accelerator to speed up the executions. All figures and tables are our own.

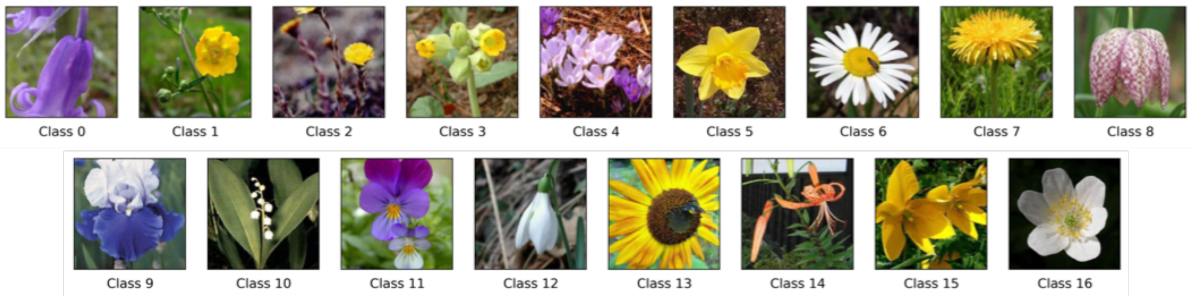

Figure 1: Randomly selected images representing each of the classes in the Flower-17 dataset.

1. Part A: Convolutional Neural Networks

1.1. Model Building and Data Augmentation

There are mainly two reasons that make the Flowers-17 database a challenging multi-class classification problem. Firstly, there's substantial variability in pose and lightning of flower images within the same class. Secondly, although the number of images per class is balanced across the dataset, there's a reduced overall number of observations for what an image recognition problem usually requires. In particular for Deep Learning models, their level of accuracy increases as more data is fed into an algorithm.

We start tackling this problem by building a set of four different Convolutional Neural Network architectures to evaluate their performance on the given dataset. Table 1 summarizes high-level information on the configurations. It's worth noting that each of the networks is deeper than its predecessor, which allows us to study their behavior with regard to the models' complexity. These networks have a Softmax output layer with as many neurons as flower types. Additional details around the number of filters and their respective sizes, as well as the size of the pooling operations and strides, are available in the associated Jupyter notebooks. All the algorithms in this part of the report were trained using Stochastic Gradient Descent as optimizer and a learning rate of 0.01.

Name	Convolutions	Pooling	Fully Connected	Padding	Activation	Parameters
Baseline	x1	x1	x1	Yes	ReLu	8,390,673
Model Mk-1	x4	x4	x1	Yes	ReLu	18,337,425
Model Mk-2	x6	x3	x1	Yes	ReLu	33,998,257
Model Mk-3	x9	x4	x1	Yes	ReLu	38,386,513

Table 1: Summary of CNN architectures.

Figure 2 below shows the CNNs' learning curves after 50 epochs of training. None of the algorithms resulted in appropriate models as they were closely fitted to the training data, which ultimately represents an overfitting issue. It is also noticeable the positive relationship between complexity and overfitting - deeper neural networks weren't able to perform better on this problem although they are more powerful algorithms compared to the baseline model. In fact, we noticed that as soon as the models' accuracy on the training data reaches the maximum value possible, the validation loss of deeper architectures grows exponentially.

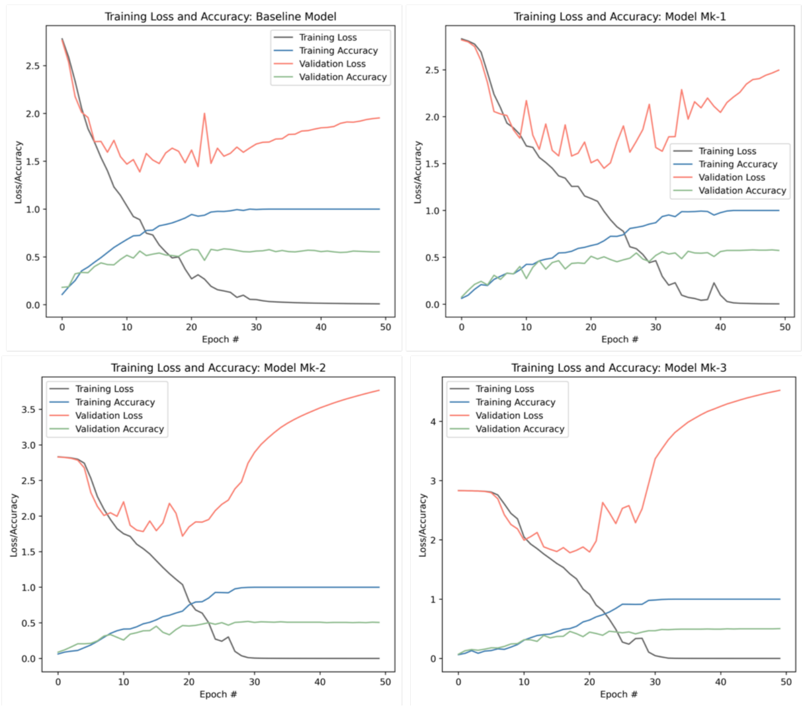

Figure 2: CNNs loss and accuracy on training and validation data.

As a result, the lack of generalization of these algorithms led to a lower level of accuracy on unseen data. Table 2 contains the results obtained for the best models found. At the specific epoch number where the algorithms reached the minimum validation loss, the baseline model outperforms the other configurations being 0.54 accurate on the validation set. This metric progressively decreases as more layers are added to the models, indicating that they are not learning specific patterns but just memorizing the images to minimize their loss function.

A high degree of overfitting and poor performance clearly suggest the need for more training instances. Data augmentation is a computer vision technique that asses this issue in classification problems. It consists in generating new artificial observations based on samples extracted from the original dataset. Essentially, it produces variants of an image by performing cropping, zooming, flipping, or shearing, among other image processing methodologies.

Name	Train Loss	Train Accuracy	Validation Loss	Validation Accuracy
Baseline	0.686	0.809	1.459	0.541
Model Mk-1	0.777	0.741	1.477	0.526
Model Mk-2	1.03	0.664	1.718	0.461
Model Mk-3	1.427	0.503	1.784	0.455

Table 2: Loss and accuracy on training and validation sets at the epoch where validation loss is minimized.

In this analysis, three different augmentation configurations are implemented to evaluate their impact on the two deepest architectures shown in Table 1, Model Mk-2, and Model Mk-3. The experiments consist in training these models using batches of artificially generated images while performing validation on the same set of original images used before.

- **Configuration #1:** Randomly zoom an image up to 50%.
- **Configuration #2:** Randomly flip an image horizontaly and/or vertically.
- **Configuration #3:** Apply a wide range of augmentation techniques at random.

The third configuration may include variants #1 and/or #2, along with changes in images' width and/or orientation by rotating and/or shearing them (please see more detail on this in the Jupyter Notebooks). This introduces substantial variability in the observations - analogous to increasing diversity in the dataset. Ultimately, Data Augmentation essentially works as a regularization technique, thus requiring more time to train the algorithms. Consequently, we have increased the number of epochs to 150. Figure 3 illustrates the learning curves of Model Mk-2 on the training settings outlined above. In terms of validation loss, we observed that flipping techniques performed worse than zooming, regardless, both resulted in overfitting at around epoch 50 when the loss gradually increases over time.

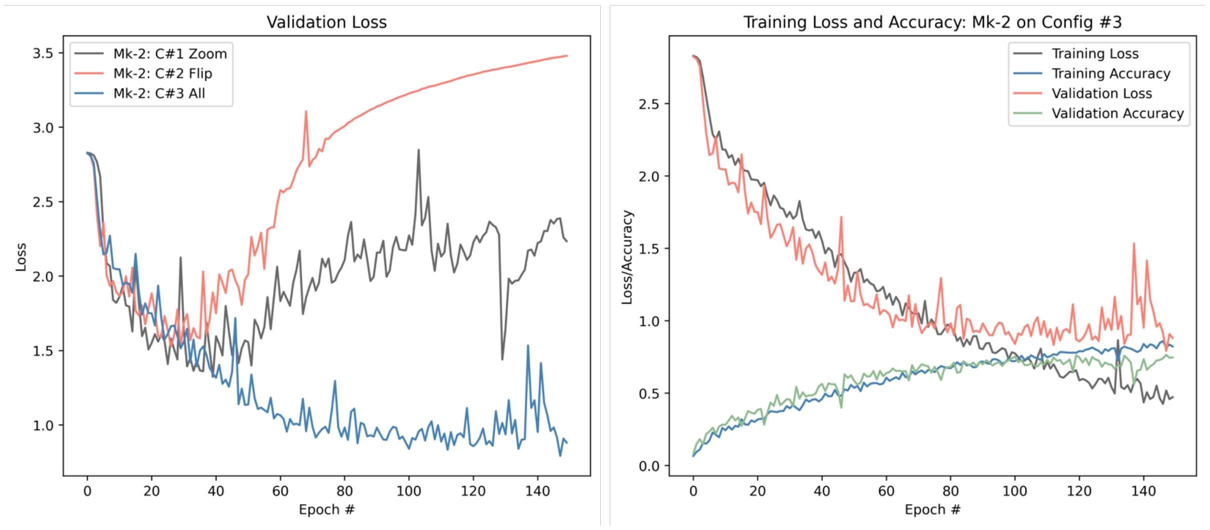

Figure 3: Model Mk-2 performance on augmented data.

Conversely, applying a variety of augmentation techniques at random has surely helped improve the performance of the neural network. In this sense, overfitting is significantly less notorious during training and the optimizer reaches new minimum validation loss values that weren't obtained before. Logically, an image generator with a wider set of augmentation capabilities is able to produce more unique variants of a single observation, which translates into a richer dataset and a barrier to the model for memorizing the images.

These training patterns have also been found in Model Mk-3's learning curves. In this case, although this is a deeper architecture we obtained similar results. It demonstrates that augmentation techniques in isolation aren't as effective as randomized combinations of them.

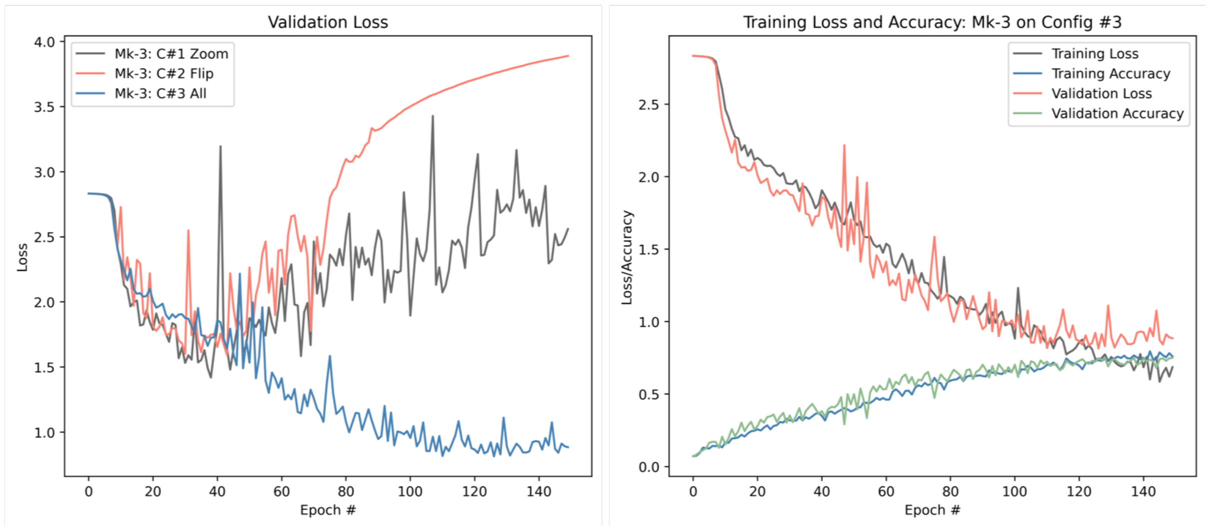

Figure 4: Model Mk-3 performance on augmented data.

In terms of level of accuracy, these last two models trained on artificially generated images constitute a significant improvement from previous evaluations of the algorithms. Table 3 shows the best models found using configuration #3 during the training process with respect to the validation loss. We observe an approximately ~0.3 accuracy uplift on validation data. The fact that their performance on the training and validation sets isn't far from each other indicates that the degree of overfitting has been largely reduced thanks to Data Augmentation. Finally, the results showed that Mk-2 yielded slightly higher accuracy than Mk-3, which could potentially mean that the latter, as a more complex model, may need more iterations to train.

Name	Train Accuracy	Validation Accuracy
Model Mk-2 (Plain Data)	0.664	0.461
Model Mk-2 (Augmented Data, C#3)	0.815	0.764
Model Mk-3 (Plain Data)	0.503	0.455
Model Mk-3 (Augmented Data, C#3)	0.732	0.747

Table 3: Validation accuracy of Models Mk-1 and Mk-2 at the epoch where the loss is minimized.

1.2. CNN Ensemble

Neural Networks are powerful Deep Learning algorithms sensitive to changes in their architectural design. For instance, models with a different number of layers, neurons, or activation functions potentially learn unique patterns of the data that make them reach a variety of valid solutions. Furthermore, they contain an important stochastic component usually referred to as weight initialization. The initial state of the weights determines the origin of an optimizer, from where it will find a path that leads to a local minimum of the objective loss function. Additionally, algorithms trained on different subsets of the data develop learning capabilities that differ from each other. Ultimately, this represents a large space of training configurations and variability that is not covered when developing and evaluating a single model.

In this section, we discuss the implementation and performance of an ensemble of Convolutional Neural Networks. This technique is aimed at combining the predictive capabilities of multiple learners in order to achieve an overall higher level of accuracy on a Machine Learning problem. In particular, we consider multiple different network architectures and model their output probabilities using a meta learner to maximize the level of accuracy of the ensemble. This technique is known as Stacking.

Firstly, we design three additional base algorithms apart from Mk-2 and Mk-3 to introduce variability in the ensemble's source of knowledge. It is worth noting the lower number of trainable parameters in Mk-4 and Mk-5 due to be using the resulting data from the convolutions directly without padding it (Table 5). The new architectures also use Leaky ReLu and Tanh as activation functions instead of ReLu.

Name	Convolutions	Pooling	Fully Connected	Padding	Activation	Trainable Parameters
Model Mk-2	x6	x3	x1	Yes	ReLu	33,998,257
Model Mk-3	x9	x4	x1	Yes	ReLu	38,386,513
Model Mk-4	x6	x3	x1	No	Leaky ReLu	19,318,193
Model Mk-5	x7	x4	x1	No	Tanh	11,285,777
Model Mk-6	x6	x3	x1	Yes	Leaky ReLu	34,598,417

Table 4: Base learners in the CNNs ensemble.

These networks are trained sequentially on augmented data and checkpointed every time they reach a new minimum validation loss during training. Storing their weights on-the-go allows us to load the models afterward to predict on a given set of images. Each of the models has a Softmax output layer with 17 neurons representing the classes in the dataset, which means that a single prediction constitutes a vector with predicted probabilities for each flower type - the higher the probability, the more confident the model is to classify an image as that specific class.

The meta-learner is built on top of the networks' predictions. It aims to find the optimal distribution of weights for each base-learner to maximize accuracy. Practically, it assigns a weight to every output neuron in the ensemble instead of assigning a single weight to each neural network. In other words, this approach lets the meta-learner know when a base-learner does particularly well on predicting a specific flower type so that it can skew the predictions towards this feature value. In that respect, we have selected as meta-learner a Random Forest Classifier for mainly three reasons. Firstly, it is a non-linear Machine Learning algorithm that could find complex patterns in the networks' output data. Secondly, it can handle datasets with a large number of features due to its selection process of variables based on information gain. Lastly, it allows us to study and analyze the CNNs' contribution to the final prediction using Random Forest's variable importance feature.

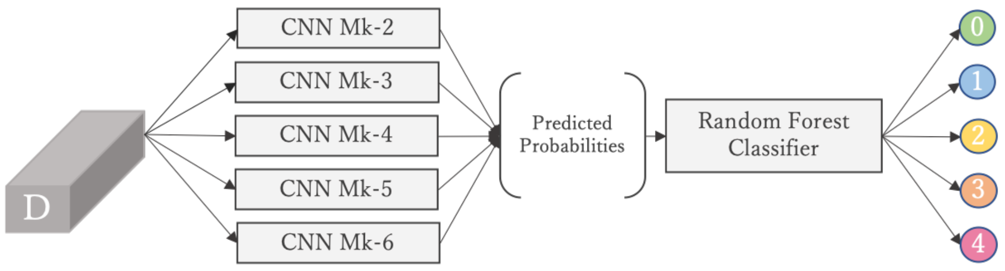

Figure 5: Forward pass illustration of the ensemble proposed using Stacking.

The meta-learner is trained on a dataset with 17 class labels and 85 features, where each of the variables represents the output probability of a neuron in a model's softmax layer. Due to the limited amount of images, once the CNNs have been trained on augmented data, we generate this dataset using the networks' predictions on the entire non-augmented original training set. The Random Forest algorithm is then trained to perform a multi-class classification problem on a subset of the resulting probability data and tested on a hold-out set. Once the base-learners and meta-learner are fully trained, we forward pass the validation data through the ensemble and collect the predicted class for each image.

We must highlight that the Random Forest algorithm was trained using the predicted probabilities that the networks made on the non-augmented original training set. As mentioned earlier, we took this approach due to work with a limited amount of data, however, it could potentially lead to a biased meta-learner model. Although the networks were not trained specifically on this data (as it was augmented), their level of accuracy is higher on the training set than the validation set. A proper strategy to perform Stacking would be to split the original training set, then use a subset of this data to train the CNNs with augmentation techniques, and finally, build the training dataset for the Random Forest model by passing the rest of the holdout training data through the trained networks and storing their predicted probabilities. Regardless, the results obtained from our *biased* implementation showed a positive impact on the validation set.

For efficiency, each of the base-learners was trained for 150 epochs storing the models' weights that reached the lowest validation loss. In table 5, we refer to Bagging as the ensemble strategy where the output probabilities of the base-learners are aggregated by taking the average. On the flip side, Stacking performs the pipeline of operations illustrated in Figure 5 using a Random Forest model as meta-learner.

Name	Validation Accuracy
Base-learner Mk-2	0.773
Base-learner Mk-3	0.788
Base-learner Mk-4	0.705
Base-learner Mk-5	0.723
Base-learner Mk-6	0.744
Bagging - Average Model	0.808
Stacking - Random Forest	0.820

Table 5: Validation accuracy of individual base-learners and ensemble techniques.

The results demonstrate the great importance of variability in the ensemble. Even a simple technique such as averaging the output probabilities of the base-learners yielded a significant improvement in the level of accuracy on the validation set. In fact, the ensemble using Bagging outperforms some of the models up to 0.10 absolute accuracy points, which is an outstanding contribution to the problem we assess in this analysis. Nevertheless, the most advanced version of the ensemble using Stacking resulted in 0.82 validation accuracy - 0.02 points higher than Bagging. Ultimately, it's a *smarter* way to evaluate the significance of each model's prediction with respect to the true class labels in the dataset. In this respect, Figure 6 shows the importance of each of the 85 output neurons in the Softmax layers of the models. Table 6 also contains the overall total score each CNN gets in the Random Forest meta-learner.

Name	Feature Importance Score
Base-learner Mk-2	0.268
Base-learner Mk-3	0.294
Base-learner Mk-4	0.076
Base-learner Mk-5	0.178
Base-learner Mk-6	0.182

Table 6: Aggregated feature importance scores - note sum of scores is 1.

Mk-2 and Mk-3 predictions are extensively being considered to determine the final class label an image belongs to, however, it's worth noting the distinct contribution of each output neuron within the same model.

For instance, the confidence level of Learner Mk-2 when the image corresponds to the 13th class is heavily weighted for the final prediction. However, the meta-learner has learned that it doesn't perform that well on images corresponding to the 15th class, but Mk-5 does it. Consequently, despite Mk-5's lower performance in the rest of flower types, its output probability for this class will be more representative in the final prediction.

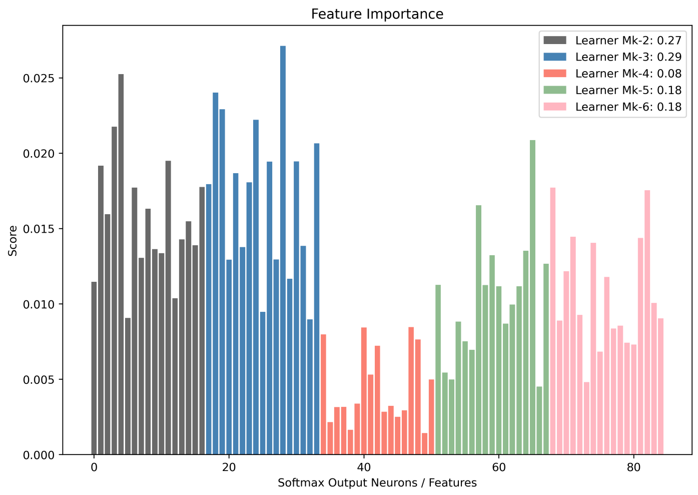

Figure 6: Random Forest's feature importance analysis.

The main difference between Bagging and Stacking is that the latter doesn't assume all models perform equally well at predicting every class. As a matter of fact, the example we mentioned earlier should contribute positively to the validation accuracy. Regardless, these techniques aren't commonly used in real-world implementations due to computational issues - the cost of training, implementing, and performing real-time predictions is relatively high and significantly slower than running a single model. However, their effectiveness is usually mentioned in the literature for reaching state-of-the-art performance.

2. Part B: Transfer Learning

2.1. Feature Extraction

Over the last decade, Convolution Neural Networks have gained increasing attention in both industry and academia, resulting in major contributions to the field of Deep Learning. Specifically for image recognition, a large number of successful neural network architectures have been proposed to deal with complex classification problems involving up to a thousand different labels - such as the case of ImageNet. These kind of models are powerful feature extractors due to the vast number of layers composing their architecture, and the richness of their convolutional filters thanks to the millions of observations on which they were trained. These last two factors require a considerable amount of computational resources that may not be always available or even feasible.

Feature Extraction is a Transfer Learning technique consisting in leveraging the recognition capabilities of these powerful models and applying them to problems they were not originally trained for. Essentially, given some set of data, it firstly collects the multi-dimensional feature map that a pre-trained CNN outputs before passing it through the fully connected layers at the top level. Secondly, the feature map is reshaped to regular tabular data and fed into a Machine Learning model that is trained on a classification problem using these features and the original class labels. Lastly, the outputs from this model are the predicted classes.

For this part of the project, we have selected four well-known feature extractors with different architectures and feature maps. In table 7, depth refers to the models' total number of layers excluding the fully connected ones, and tabular features is the number of variables after reshaping the feature map.

Feature Extractor	Total Depth	Trainable Parameters	Feature Map Shape	Tabular Features
VGG16	19	14,714,688	(4, 4, 512)	8,192
InceptionV3	311	21,768,352	(2, 2, 2048)	8,192
MobileNet	87	3,206,976	(4, 4, 1024)	16,384
Xception	132	20,806,952	(4, 4, 2048)	32,768

Table 7: Pre-trained Convolutional Neural Networks selected for Feature Extraction.

Although these models were all trained on the ImageNet dataset, their weights are still extremely useful to find patterns in other image recognition problems. Please note, despite the number of features is remarkably large with respect to the number of observations in the Flowers-17 dataset, we don't apply any feature selection technique to the resulted variables. However, we do normalize the data before being passed through Machine Learning algorithms.

For the experiments, we selected five classifiers to model the extracted feature data and map it into classes: Naive Bayes, K-Nearest Neighbors, Support Vector Machine, Logistic Regression, and Random Forest. These are of different nature and complexity, but they share the ability to handle multi-class classification problems. Hyper-parameter optimization and cross-validation are out of the scope of this analysis, therefore, after some experimentation we ran the algorithms on a set of training data using sub-optimal hyper-parameter values that *generally* worked better than their default configuration.

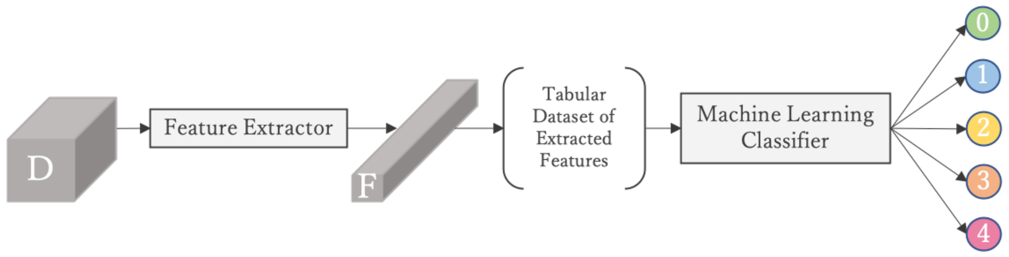
D
Feature Extractor
F
Tabular Dataset of Extracted Features
Machine Learning Classifier
0
1
2
3
4

Figure 7: Forward pass illustration of a Machine Learning algorithm using Feature Extraction.

The pipeline of operations to evaluate these models is described as follows. Firstly, in order to obtain meaningful features from images, the feature extractors are used to predict on the original Flowers-17 training and validation sets. The resulting feature maps are reshaped, normalized, and stored as tabular datasets in arrays. Secondly, the learning algorithms outlined above are trained on the new training databases - there are four different variants of a single Machine Learning model, one for each feature extractor. Finally, these models are tested on the tabular validation set to evaluate their level of accuracy.

Table 8 shows the results obtained from putting this into practice. There's a significant number of Model/Extractor pairs that outperformed the previous maximum level of accuracy using Stacking. As a matter of fact, with MobileNet as a feature extractor only the weaker models are below 0.9 accuracy on the validation set, which constitutes a major improvement in the classification problem. We were not expecting Naive Bayes and K-Nearest Neighbors to be well-suited to this image recognition task. They were selected specifically to demonstrate the fact that they may struggle with complex high-dimensional datasets. These models are very sensitive to noisy data as they both assume that all the features are equally important to predict the correct class label. Conversely, Support Vector Machine, Logistic Regression, and Random Forest are able to evaluate variables concerning their predictive potential, therefore, this allows them to deal with thousands of features relatively better such as in this case.

Model / Feature Extractor	VGG16	InceptionV3	MobileNet	Xception
Naive Bayes	0.450	0.655	0.705	0.364
K-Nearest Neighbors	0.664	0.711	0.852	0.685
Support Vector Machine	0.838	0.805	0.929	0.770
Logisitic Regression	0.891	0.864	0.944	0.820
Random Forest	0.864	0.820	0.926	0.794

Table 8: Validation accuracy of Machine Learning models using Transfer Learning's Feature Extraction.

Regardless, we observed that even the more complex algorithms performed worse on extracted features in an extremely high dimensional space. Our main hypothesis refers to the *curse of dimensionality.* Xception's feature map is 2048 filters deep of 4x4 each, resulting in 32,768 features and only 1,020 training instances. Although advanced algorithms combat this issue more effectively than weak classifiers such as the ones mentioned earlier, it makes the data sparse difficulting the search for patterns in a limited amount of observations. For that reason, we believe pre-processing and feature selection techniques could be great additions to the pipeline when using pre-trained models with large feature maps.

As presented in Table 7, VGG16 and InceptionV3 have the smallest feature maps with 8,192 features each. They worked similarly well, although VGG16 yielded a slightly higher level of accuracy. Nevertheless, the validation results showed that Machine Learning models trained on these *reduced* datasets did not perform better than on MobileNet's 16,384 features. In the end, despite being more compacted versions of the images' patterns, VGG16 and InceptionV3's features may not be as informative as MobileNet's ones in their raw format.

In our experiments, Logistic Regression always performed better than any other model. The highest level of validation accuracy is 0.944 using Mobilenet, followed by Random Forest and Support Vector Machine obtaining 0.926 on the same extracted features. As mentioned earlier, we used sub-optimal hyper-parameters to train these algorithms, which could potentially lead to misleading conclusions about the performance of these learners. In any case, the results highlight the powerfulness of a relatively simple Softmax classifier using regularization, an architecture that is analogous to building a multi-class Logistic Regression model.

2.2. Fine Tuning

In the previous section, we used Machine Learning algorithms to learn the patterns behind feature maps of pre-trained models to classify images contained in the Flowers-17 dataset. However, this analysis didn't consider fully connected neural networks as top-level models. Fine Tuning is a Transfer Learning technique aimed at building neural networks on top of existing feature extractors to further optimize the weights of the new architecture precisely assessing the maximum level of accuracy possible on a given problem. As in Feature Extraction, due to be *recycling* existing pre-trained convolutional filters, this technique greatly helps with small datasets such as the one we're running experiments on.

The design and training process of a model using Fine Tuning consists of the following steps. Firstly, in order to let data flow through the system, a brand new fully connected neural network with input and output layers is appended to the resulting feature map of an existing pre-trained Convolutional Neural Network. Secondly, the training process is divided into two phases. During Phase A, we freeze the feature extractor's weights and the top-model is normally trained on the incoming feature maps. For Phase B, previously selected weights of the feature extractor get unfrozen and the entire model is re-trained at a lower learning rate. During this phase, both filters and neuron connections are gradually *fine tuned* to minimize the overall loss of the algorithm, Lastly, the resulting model is evaluated on the validation set.

For this part, we selected two of the best performing feature extractors from the previous section: VGG16 and MobileNet. After prototyping a variety of models, we observed that adding more than one fully connected layer to the top model could become counterproductive resulting in longer training times but achieving similar results. Likewise, 512 neural units seemed enough to reach a reasonably high level of accuracy on this specific classification problem.

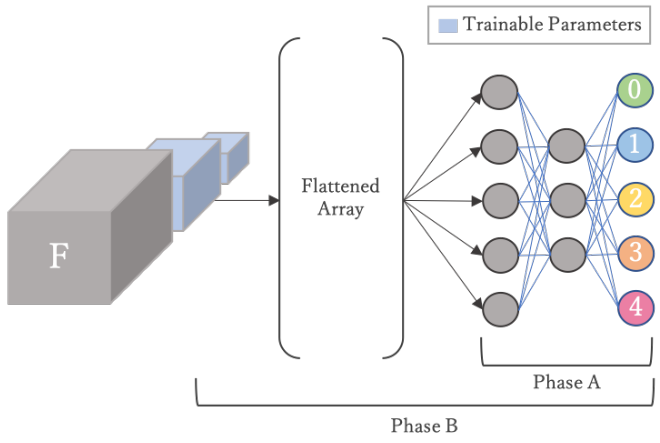

Figure 8: Forward pass illustration of a neural network using Fine Tuning

Our experiments with Fine Tuning considered the application of different optimizers, normalization, and regularization techniques to the fully connected network. We designed 8 configurations for each feature extractor from where we draw conclusions about their performance. The layers that get frozen/unfrozen during the training phases are determined from the beginning. Additionally, both Phase A and B of training take 25 and 50 epochs to complete on each run, using learning rates of 0.001 and 0.0001, respectively. Finally, in configurations where regularization is applied, the dropout probability is set at 0.5.

Model / Feature Extractor	VGG16	MobileNet
Adam		
Phase A	0.891	0.938
Phase B	0.920	0.941
SGD		
Phase A	0.779	0.891
Phase B	0.867	0.902
Adam + Batch Normalization		
Phase A	0.888	0.949
Phase B	0.932	0.947
SGD + Batch Normalization		
Phase A	0.808	0.882
Phase B	0.844	0.882
Adam + Dropout		
Phase A	0.888	0.935
Phase B	0.923	0.947
SGD + Dropout		
Phase A	0.785	0.917
Phase B	0.876	0.911
Adam + Batch Normalization + Dropout		
Phase A	0.885	0.947
Phase B	0.941	0.967
SGD + Batch Normalization + Dropout		
Phase A	0.805	0.905
Phase B	0.855	0.908

Table 9: Validation accuracy of Fine Tuning models under different configurations.

Table 9 shows the models' level of accuracy on the validation set at Phase A and B of Fine Tuning. As shown in previous sections, these metrics relate to the best model found during training i.e. the one that obtained the lowest validation loss.

At a high level, we observed a clear pattern indicating that whenever Adam is used as optimizer, the results are always better than using Stochastic Gradient Descent. Moreover, this performance gap further increases when Adam is combined with any of the more advanced configurations presented in this section. Indeed, the results showed that most of the time, Stochastic Gradient Descent doesn't get benefited from using additional normalization and regularization techniques, regardless of the feature extractor. However, we did notice that models using Adam along with dropout and batch normalization outperformed the variants that used these techniques in isolation.

In terms of phasing, running Phase B doesn't seem to add much to the final validation accuracy when the top classifier is run on a Stochastic Gradient Descent configuration using MobileNet as a feature extractor. Generally, we measured larger jumps inaccuracy from Phase A to B when VGG16 is applied. Although both models were trained on the same ImageNet data, this clearly indicates that the original VGG16 filters might not be as useful as MobileNet's ones to perform a flower recognition task. However, the main advantage of Transfer Learning's Fine Tuning is that we're able to adjust some of their weights slightly to make them fit a specific classification problem much better.

Finally, it's worth looking back at previous results presented in this report to appreciate the enormous progress made on the Flowers-17 dataset. Table 10 summarizes the best models found in each section of this document.

Model	Validation Accuracy
Baseline Model	0.541
Learner Mk-2 + Data Augmentation	0.764
x5 Base-Learners + Stacking RF	0.820
MobileNet + Logistic Regression	0.944
MobileNet + NN (Adam + Batch Normalization + Dropout)	0.967

Table 9: Validation accuracy on the Flowers-17 dataset.

3. Part C: Research

3.1. Adversarial Machine Learning

Convolutional Neural Networks have proven to be excellent problem solvers in image recognition scenarios. Their outstanding capabilities have led researchers and practitioners to develop state-of-the-art models that in a variety of cases were able to achieve human-level performance. For that reason and moving towards real-world applications, during the last decade there has been a notorious work on understanding the potential drawbacks of these models. More specifically, Adversarial Machine Learning is an emerging area of research aimed at fooling intelligent systems and discover their vulnerabilities and weaknesses. It assesses ethical and security issues with regard to the lack of robustness of learning algorithms by introducing adversarial observations that make them incorrectly guess the actual class label of these images. Adversarial examples might be indistinguishable from the original observations to a human's pair of eyes, however, the literature around this area has effectively shown that machines can be very sensitive to minor changes in an image's pixels.

Adversarial attacks can be broadly grouped into two main categories: White-Box Attacks and Black-Box Attacks. In the first one, a suspect or adversary has access to the model's architecture, weights, parameters, and data distribution. The attacker is fully informed about the algorithm's behavior and design attacks accordingly to exploit its vulnerabilities. Such attacks are usually based on the gradient direction of the algorithms and their specific loss function. On the flip side, Back-Box Attacks are designed without having detailed knowledge about the model itself or its main components and architecture. Hence, the adversary only has information from previous experiments' results and crafts new attacks based on them. This is usually based on greedy local search approaches that try to approximate the gradient direction of the models by observing the relationship between input and output values.

Adversarial strategies can be further divided into Evasion, Exploratory and Poisoning attacks. The first two methods occur during the testing phase of an existing pre-trained model. Essentially, Evasion attacks provide malicious samples to a machine with the goal of fooling the system on a Black-Box or White-Box environment; while Exploratory attacks assume that the adversary only has Black-Box access and sends attacks in order to extract information of the model's behavior to create new curated adversarial examples. Alternatively, Poisoning attacks take place during the training phase of an algorithm, where the adversary feeds a learning system with *poisoned* samples to compromise its capabilities.

All these adversarial strategies have specific goals based on the expected result of a concrete attack. It's possible to distinguish between (1) Confidence Reduction, (2) Misclassification, (3) Targeted Misclassification and (4) Source/Target Misclassification. For instance, (1) an adversary could aim to lower the confidence level of a model for a specific task. In the domain of face identification, it would be possible to make a system doubt about a certain subject's biometric data using corrupted observations. Moreover, (2) the attacker might be interested in increasing the overall error rate of the model by making it misclassify an input instance to any other existing class within the output space. For example, a subject's face image being predicted as someone completely different. Additionally, (3) the adversary could attempt to bias the overall misclassification rate towards a specific target class. For instance, each member of a group of persons could potentially be identified as the same subject, compromising a security recognition system. Finally, (4) this classification error could be targeted in both ways making the model struggle at differentiating between a given pair of classes. This is analogous to force the classification of subject #1 as subject #2 every time subject #1 is examined, which in a real-world scenario translates into important privacy issues.

In recent years, the literature has proposed a large number of successful adversarial techniques representing innovative methods to fool intelligent systems. Due to the extensive list of methodologies, in this report we cover three major adversarial attack contributions that were of our own interest:

Fast Gradient Sign Method (FGSM)

This adversarial technique assumes the attacker has White-Box access and the goal is forcing the model to classify an input image incorrectly increasing the overall error rate. It's implemented as a generator that creates adversarial examples in a sequential fashion. The theoretical background is based on the backpropagation algorithm implemented on gradient-based models. When training neural networks, the weights are updated following the opposite direction of the gradients in order to find the minimum value possible of a loss function. On the contrary, the main idea behind this adversarial technique lies in modifying an image's pixels in the same direction of the gradients so that the loss gets increased in such a way that the image is classified incorrectly. For this process, the original imaged is passed once through the network, storing the loss with respect to the true class and the resulting gradients. The perturbation that is applied to the image is then given by a matrix of the sign of the gradients multiplied by an arbitrary weight factor ε. Notice this method doesn't guarantee a successful attack - it just crafts a malicious variant of the original image by increasing the expected loss using a perturbation-level parameter ε. However, the authors claimed a success rate of 97.5% on adversarial examples generated from the MNIST test set.

One-Pixel

This adversarial technique is classified under Black-Box Attacks and it aims to make a model classify an input image to any class but the correct true label. The novelty of this attack is its ability to fool a system by changing only one pixel of a given image. It uses a population-based optimization algorithm called Differential Evolution, where a single perturbation is represented as a rank-1 array of length 5 containing *x-y* coordinates and RGB values for a single pixel. Perturbations are randomly altered creating children that compete with their parents in subsequent iterations for maximizing fitness. This approach accesses the target model as an oracle to calculate the fitness score of each candidate solution as the output probability. The results showed that despite being a relatively simple technique, it was able to fool state-of-the-art deep neural networks trained on the CIFAR-10 dataset with a 67.9% success rate.

Jacobian-based Saliency Map Attack

This adversarial technique requires White-Box access to the model and the main distinction with the attacks mentioned earlier is that it allows two-way targeting i.e. a pair of classes. This approach focuses on applying minimum changes to the original images enough to fool a gradient-based classifier in a less extreme way than One-Pixel does. In this respect, instead of perturbing every single pixel (Fast Gradient Sign Method), or drastically alter a single one (One-Pixel), it finds pixel coordinates in an iterative fashion that resulted in a higher likelihood of evading the system. The algorithm creates a saliency map of the gradients belonging to the output layers, and increase/decrease accordingly the pixel values that degrade the most the performance on the target label. The authors reported a 97% rate of adversarial samples successfully misclassified generated from the MNIST test set.

To conclude, it's also worth mentioning that part of the existing work on Adversarial Machine Learning also focuses on combatting such malicious attacks. For instance, some of the current methodologies to strengthen the models' predictions consider including adversarial examples during training in a similar way that data augmentation works. Others proposed building non-differentiable models on top of the neural networks, such as standard Machine Learning algorithms, to hide information about the networks' gradients and avoid gradient-based attacks. Additionally, the usage of smoothing filters over images during training was also proven to improve the robustness of the classifiers. More advanced methodologies suggest the usage of Generative Adversarial Networks to help a learning system differentiate between real and fake images. In the end, all the existing and future contributions around this area will be key for the maturity of Deep Learning systems operating in real-world environments.

References

[**1**] H. J. Veen, N. Dat, A. Segnini. "Kaggle Ensembling Guide". [accessed April 28, 2020]. *https://mlwave.com/kaggle-ensembling-guide/.* 2015.

[**2**] D.H. Wolpert. "Stacked generalization". *Neural Networks.* vol. 5. pp. 241-259. 1992.

[**3**] I. J. Goodfellow and J. Shlens and C. Szegedy. "Explaining and Harnessing Adversarial Examples". *arXiv:1412.6572.* 2014.

[**4**] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik and A. Swami. "The Limitations of Deep Learning in Adversarial Settings". *2016 IEEE European Symposium on Security and Privacy.* pp. 372-387. 2016.

[**5**] J. Su, D. V. Vargas and K. Sakurai. "One Pixel Attack for Fooling Deep Neural Networks". *IEEE Transactions on Evolutionary Computation.* vol. 23. no. 5. pp. 828-841. 2019.

[**6**] N. Akhtar and A. Mian. "Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey". *IEEE Access.* vol. 6. pp. 14410-14430, 2018.

[**7**] A. Chakraborty and M. Alam and V. Dey and A. Chattopadhyay and D. Mukhopadhyay. "Adversarial Attacks and Defences: A Survey". *arXiv:1810.00069.* 2018.