INF8245E - MACHINE LEARNING (FALL 2021) KAGGLE COMPETITION

TEAM NAME : COFFEE ADDICTS

Guillaume Thibault
1948612
g.thibault@polymtl.ca

Jacob Brisson
1954091
jacob.brisson@polymtl.ca

Mathieu Guay
1957228
mathieu-2.guay@polymtl.ca

1 FEATURE DESIGN

The feature designing is dependent on the chosen model. For the Linear SVM and Random Forest classifiers, we transformed the images into a flatten data vector, so that each pixel becomes a feature of these models. For the CNN approach, we kept the images in a (96,96,1) format, because CNN uses the spacial aspect of the data and not only the pixel values. However, in the 3 experiments, the images were already in gray scale and we then didn't need to transform colored images. We only divided the pixel values by 255, so that they would be scaled in between 0 and 1. For the CNN models, we also transformed the class values to one-hot encoded vectors, so that we could use the categorical-entropy loss function with the keras library.

The use of image pre-processing has also been tried and used to reduce image noise, thus improve model performance. There are several types of pre-processing for images and some have been tested. The first filter tested is the Mean filter which consists in using a sliding-window that moves and replaces the central value by the mean [4]. The effect of this filter can be seen in the figure 1. The filter reduced the level of noise in the image but it also made the image blurrier. The contrast between the shapes is less present and it is more difficult to identify the animal for a person, but this filter can still be useful. The second filter used is the Median Filter. Using the same concept as the first filter, this one replaces the central value of the window by the median value. The effect of this filter can be seen in the following figure and once again, the filter reduces the noise of the image, but it kept more contrast between the different elements of the image.

The third filter tested is the Gaussian filter. This filter is a modified version of the mean filter where the weights of the impulse function are normally distributed around the origin [4]. This filter gives a very blurred image. We can however use this filter and subtract it from the original image values to obtain the image corresponding to the high-frequency component. Moreover, if we add the high-frequency component image to the basic image, we can then find a very well defined image with a noise reduction.

The last filter tested is the Sobel Edge Detection filter. This one is an extension of the horizontal and vertical gradient. The horizontal Sobel filter detects edges in the horizontal direction, while the vertical Sobel filter detects edges in the vertical direction by taking the gradient at the point [4]. Subsequently, the two filters can be combined by taking the Euclidean distance of the intensity of the two filters. The result obtained by this filter can be seen in the following image. The image obtained is very clear and we can quickly distinguish the main lines of the figure.

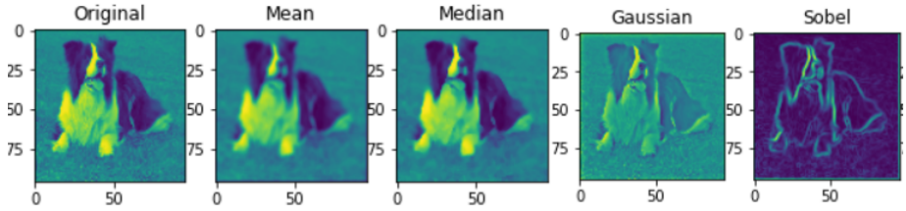

Figure 1: Filters

2 ALGORITHMS

2.1 SVM

The Support Vector Machine is a linear model for solving classification and regression problems. It can be applied to both linear and non-linear problems and works well for many practical problems. The goal of the SVM model is to create a lines or hyperplanes in order to separate the data into classes [3]. The main hyper-parameter of this model is a constant C to smooth the decision surfaces and thus reduce the overfit of the data.

2.2 RANDOM FOREST

The random forest model is an extension of the decision trees model seen in class. It is comprised many different decision trees, all combined into one model - an ensemble. First off all, the random forest model calculates the predictions made by all the defined decision trees. Afterwards, the model compares all the predictions, and picks the prediction that appears most often. What makes the model powerful and useful in many different problems is that the underlying decision trees have low correlation. Therefore, on average, the errors made by individual trees are compensated by the other ones. In this fashion, the variance of the result goes down, so the predictions are more reliable [9].

2.3 CNN

For the CNN model, the goal is to automatically find the best features to describe an image by doing convolutions of filters on the image. These convolutions are doing dot products between the image and the filters by moving the filter on the image and the resulting values are then passed onto the next layer of convolutions. To reduce the number of features of the model, we can use maxpooling, which takes the result of a convolution and replace nxn (pooling size) values by the maximum of those values. Then, when the feature selection process is done, the model uses a flatten layer, which transforms the representation of the features to a vector that can be passed to a regular dense neural network. The last part of the CNN will then combine the features and use non-linear activation functions to transform them and pass them as input for the last layer, which is in our case a 11 neurons layer with a softmax activation function to predict the class of the animal in the image in the 11 classes possible.

3 METHODOLOGY

To divide the training and validation data, we had access to a training dataset of 11 887 images. We decided to use a single validation set chosen with 20% of the randomly sampled training data. The tuning of the hyper parameters and the evaluation of the models are done on this validation data. To evaluate the performance of the models, we used the micro-f1 score on the validation set. We can then explain, for each model, which are the hyper parameters chosen and the methodology to train the model.

3.1 SVM

For the SVM model, a simplified dataset was first used. The cat and spider images were used to determine which type of SVM used between a linear or non-linear model. The baseline performance of the models after trying them with a few kernels and some normalization constants is 63% for the linear model and 78% for the nonlinear model with the rbf kernel.

The next step was to add some classes. The purpose of this addition is to verify that the model continues to perform even in the face of a problem with several classes. For this step, spider, cat, goat and horse data were used. The first model generated with the SVM method gave an accuracy of 60%. With this result we can imagine that the optimization of the parameters will increase the score, but unfortunately, the complexity of the code makes that an exhaustive search of many parameters is extremely long. Moreover, when we want to take the complete dataset with more than 11000 rows of data, the model training time becomes in turn extremely long and the search for the best parameters

becomes unattainable. For this reason, the SVM model was put aside and our attention was focused on the CNN model with has better potential.

3.2 RANDOM FOREST

There are many hyperparameters that are possible to test using grid-search method. In this hyperparameter list is included pruning alpha ccp_alpha, $n\_estimators$, max_depths, $'gini'$ or $'entropy'$ criterion. Before starting a grid search, the effect of each of these hyperparameters is tested to see if they have a big influence on the f1 score. Out of these hyperparameters, the ccp_alpha had no influence on the scores. It therefore is left out of the hyperparamter list. The other hyperparameters do have influence on the results, but the f1 score yielded while testing different values were very low compared to a baseline CNN model. For this reason, the grid-search on the different hyperparameter was left behind - all our focus was put on making the CNN model as accurate as possible. Nevertheless, we shall present the results we have found for this hyperparameter list.

3.3 CNN

Unlike, the other models, the CNN approach has a very large number of hyper parameters to test to find the best performance. In fact, we had to test the number of convolution layers, the number of pooling layers, their layout in the network and the number of dense layers. For each convolution or pooling layer, we had to test the size of the filters and the stride value, which is the number of pixels for each movement of the filters on the images. We also tested the padding value, which could be "valid", for no padding, or "same", to have an even amount of zero padding around the images. We could also play with the number of neurons for each layer and their activation function (for convolution or dense layers). Because of this huge amount of hyper parameters, we had to manually try different architectures to find the best model, because the performance was more impacted by the layout of the architecture than by the number of neurons or other specific hyper parameters.

To train the CNN model, we first decided to base our model on already known architectures. We tested the Alexnet architecture, the SqueezeNet and the VGG 11 and 16 architectures, all implemented with the keras library [5],[7],[8]. However, none of these pre-made architectures were able to learn on our dataset. We then chose to test a simpler model, which is presented in the Appendix as model *A*. This model and other variant were overfitting to the training data. We then added Dropout layers in between the Dense layers, to put some inputs of the neurons at 0 with a probability "p", which is another hyper parameter. This made it harder for the model to learn too much precise weights according to the training data and it is then more generalizable. Then, the model *A* and other variant were stuck at an average performance of 0.58. To solve this issue, we tried to use generators to train the models [2]. The goal of the generators is to take images as input and produce modified images based on the desired transformations. These can be a rotation of the image, a zoom in or zoom out, an augmentation of the brightness, etc. We decided to use a zoom modification of 0.2, meaning that we are producing new images zoomed by a factor in [0.8,1.2], where a factor < 1 means a zoom in and a factor > 1 means a zoom out. We also added a modification of the shear intensity, which is used to change the perception angle of the images. We used a shear range value of also 0.2. Finally, we added random horizontal flips to the produced images. With this generator, we can use the "flow" method to generate new group of modified images from the original training data. These groups are of size batch_size, which was set in our case to 16. Thus, we can use the generated data and not the original data to train our model. This is useful because the small modifications to the images makes it harder for the model to overfit to the training data and it will learn to recognize variations of the initial images, making the model more generalizable. With this new data, the models were again not able to learn anything. To solve this problem, we added BatchNormalization layers to our CNN. These layers are useful to standardize the input of the activation layers, so that the inputs are centered in the function. This modification resulted in a huge improvement in the learning speed of the model, making it possible to learn from the generated data. Finally, we realised that the model was learning slower and slower with the number of epochs. This was because the learning rate was too high in the end and thus the variation of the steps towards the minimum of the loss function was too big. To solve this issue, we used a function to reduce the learning rate over time. This function returns the initial learning rate (0.001) multiplied by an exponential of (-k * epoch), where k is set to 0.02. This makes it easier for the model to learn in the end of the training.

With all these improvements added to our training method, we are able to test different architectures to find the best one with respect to the validation set performance. We nevertheless set some hyperparameters to constant values : batch_size = 16, epochs = 150, optimizer = *adam*, loss = *categorical_crossentropy*, activation = *relu*, conv_size = (3,3), maxPooling_size = (2,2) and padding = *same*. We then tested different layouts, different number of neurons per layer, different probability of dropouts and different strides values to find the best ones according to validation performance. Before predicting the test set, we added the validation data to the model. We tested the addition of these data by two methods. The first is to simply train the model for a few epochs (e.g. 5). The second method tested is to use the same image generator used for training, but this time with the images from the validation set. In order to check that the training is done in a favourable way, the training set is used for the validation and the training is stopped after about ten epoches to keep the good validation of the training set.

3.4 Model Ensemble

Several models were generated with the CNN method including some with the use of filter and the use of generator to train the validation set. All these models were used to predict the test set and only 58.8% of the datapoints were predicted by the same value by all models. It is then interesting to look at the other datapoints. By comparing the predictions of the models for the datapoints that did not reach a consensus, we see that most of the time, the majority of the predictions are similar. In order to improve the quality of the predictions, the concept of model ensemble was used. The type used was the aggregation of prediction by majority vote [1]. This principle consists in taking the most commonly predicted prediction by the set of models for an image and if no prediction are the same, to take a random prediction.

4 Results

4.1 Random Forest

The hyperparameters tested are: $Depths = [None, 25, 30, 35, 40, 45, 50]$ and $n\_estimator = [10, 20, 50, 100, 200, 500, 1000, 2000]$. The graphs they generate are presented below

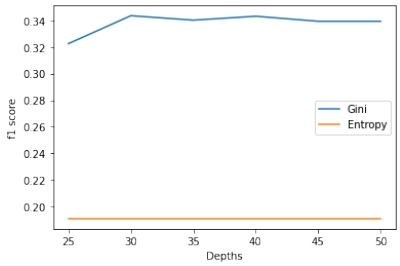

Figure 2: Micro f1 score for varying max_depths hyperparameter

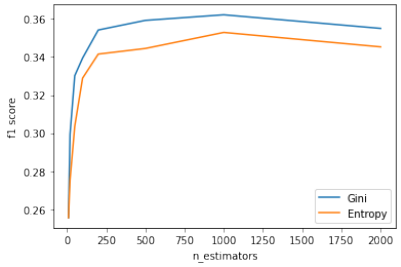

Figure 3: Micro f1 score for varying n_estimators hyperparameter

4.2 CNN

Several architectures were tested and trained, the F1 score of each model was then tested.

Model	Micro f1 score
B	0.7512
C	0.7471
D	0.7573
E	0.7769
F	0.6675

Figure 4: Results for CNN models

The architecture with the highest score (architecture E) was then selected and several models were generated with it. The first two models were generated with the same data, but with a different generator. For model 1 the parameters used for the generator are: $shear_range = 0.2$, $zoom_range = 0.2$ and $horizontal_flip = True$ while for model 2: $shear_range = 0.18$, $zoom_range = 0.18$ and $horizontal_flip = True$.

A model using the images with the Gaussian filter was generate to compared the filtered version to the original one. Finally a last model using Sobel filters was trained for the same purposes and also to compare it to the Gaussian filtered model.

The next table show the F1 mirco accuracy of the four models after training data, but also after training validation data by the two methods mentioned above. In order to verify the accuracy of the two methods of training validation data, a second validation set consisting of 286 images was initially set aside for this purpose (Validation Set 2).

Model				Accuracy		
#	Image Filter	Seed	Model train on	Training Set	Validation Set 1	Validation Set 2
1	None	100	Pre valiadion set	0.992428226	0.758604207	0.723776224
			Normal Validation Set	0.998527711	0.999521989	0.772727273
			Generated Validation Set	0.968766432	0.923040153	0.734265734
2	None	200	Pre valiadion set	0.998843201	0.764340344	0.734265734
			Normal Validation Set	0.999894836	0.994741874	0.772727273
			Generated Validation Set	0.987695867	0.867112811	0.713286713
3	Gaussian Filter	300	Pre valiadion set	0.983173835	0.75334608	0.758741259
			Normal Validation Set	0.99642444	1	0.772727273
			Generated Validation Set	0.945945946	0.914435946	0.716783217
4	Sobel Edge Filter	400	Pre valiadion set	0.997686402	0.703632887	0.692307692
			Normal Validation Set	0.692307692	1	0.681818182
			Generated Validation Set	0.980544747	0.942638623	0.702797203

Figure 5: Accuracy of the models generated

4.3 Model Essemble

Finally, the appropriateness of using the model sets was tested using the model using the Gaussian filters. With the three models generated, namely the one after the training set and the ones after the validation set training by the two methods seen, the validation set two was predicted. Subsequently, the three prediction sets were combined by majority voting and the following table represents the results before and after this vote.

Model	3 with Gaussian Filter		
Set	pre valid	w/ normal valid	w/ gen valid
Accuracy	0.758741259	0.772727273	0.716783217
Model Essemble	0.77972028		

Figure 6: Results Model essemble using model 3

5 Discussion

5.1 Random Forest

As was explained in the previous section, we can observe the results for f1 score are low. This was the reason why the model was not more explored than this. However, we can still observe patterns

on the tested hyperparameters.
First off all, we can see that in all our tests, the $'gini'$ criterion is better than the $'entropy'$ hyperparameter. If we were to test other hyperparameters, we could assume with high degree of certainty that $'gini'$ would give us better results than $'entropy'$. Secondly, we can see that the $n\_estimators$ hyperparameter seems to find a maximum for f1 score. If more attention had been given to the analysis of the Random Forest method, we would have made smaller and smaller grid searches to find the best value for $n\_estimators$.

5.2 CNN

As we can see, we obtain a better performance on the validation set with the model E with a 0.7769 micro f1 score. We didn't try a lot of different architectures because for each model it takes around 100 epochs to converge and it is very time consuming without a permanent access to a GPU. Since we didn't have unlimited GPU access, we weren't able to try as much models as we wanted. Also, some of the models tested weren't promising, meaning that they weren't learning or that they had very poor validation performance, and we didn't include them in the results. However, even though the model E is the best single CNN, we found that we could combine multiple models to have a better generalization on our predictions. Thus, we obtained a performance of 77.972% on the validation set two with the combinations of models, which is better than every single models.

Looking at the results of the different models generated, we see that the models using Sobel's filters perform significantly worse than the others. This is why the model was ignored for the final prediction of the contest. Moreover, we see that majority voting significantly increases the prediction accuracy and its use for the final prediction is then paramount. To make the contest predictions, the four models having an accuracy with more than 75% were combined by the model ensemble method by majority voting. The models used are: *Model 1 with validation add normally, Model 2 with validation add normally, Model 3 with validation add normally and Model 3 without the validation set added*. The result of the accuracy on the public kaggle dataset is then of 80.832%.

5.3 Conclusion

During this competition, we learned the importance of using data augmentation to increase the number of "different" images and the importance of using filters to clean the noise. Also, we found that, for this task of image classification, SVM and Random Forest classifiers were not performing well, while CNN models could achieve performances close to 80% and even more when combining multiple models. To obtain better results, we could let the model train longer and with more data generated with the generators. We could also have used a random grid search for tuning the hyperparameters to try more hyperparameters in very different combinations. This would have helped, because neural networks can be very sensitive to certain parameters and testing all combinations of parameters that have very little impact is not useful. Finally we would have tried also deeper neural networks [6].

6 Statement of Contributions

Each member of the team has contributed to most of the parts of the assignments. We have each coded a part of the final notebook file and written a part of the report. As for individual contributions, Guillaume spent his time on three aspects - the initial feature design, linear SVM model and the CNN model. In particular, he worked on the filtering of the initial dataset and made sure it worked. The SVM data presented in the above report has solely been generated by him. He also contributed on the CNN model for the model ensemble bit. On the other hand, Mathieu spent his time trying out the random forest model, and then also hopped on working to optimize the CNN model. He mostly spent his time trying to find the optimal architecture for the CNN model. Finally, Jacob spent most of his time working on the CNN model, trying out a bunch of different models, and also helped for the pre-processing of the data. He was the one to implement the data augmentation for the CNN model, which greatly improved the accuracy.

We hereby state that all the work presented in this report is that of the authors.

7 REFERENCES

[1] Alhamid, M. (2021). Ensemble Models. Towardsdatascience. https://towardsdatascience.com/ensemble-models-5a62d4f4cb0c

[2] Brownlee, J. (2019). How to Configure Image Data Augmentation in Keras. Machinelearningmastery. https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

[3] Gandhi, R. (2018). Support Vector Machine — Introduction to Machine Learning Algorithms. Towardsdatascience. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[4] Ghosh, A. (2020). Image Processing. Medium. https://medium.com/journey-to-machine-learning-deep-learning/chapter-five-image-processing-e5b92ee87c0b

[5] Hassan, M.U. (2018). VGG16 – Convolutional Network for Classification and Detection. Neurohive. https://neurohive.io/en/popular-networks/vgg16/

[6] Karpathy A. (2019). A Recipe for Training Neural Networks. Andrej Karpathy blog. http://karpathy.github.io/2019/04/25/recipe/2-set-up-the-end-to-end-trainingevaluation-skeleton–get-dumb-baselines

[7] Katsios, D. (2021). CNN ARCHITECTURES: SQUEEZENET. Machinelearningtokyo. https://machinelearningtokyo.com/2020/04/11/cnn-architectures-squeezenet/

[8] Saxena, S. (2021). Introduction to The Architecture of Alexnet. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/03/introduction-to-the-architecture-of-alexnet/

[9] Yiu, T. (2019). Understanding Random Forest. Towardsdatascience. https://towardsdatascience.com/understanding-random-forest-58381e0602d2

A APPENDIX

Type	Neurons	Filter Size	Stride
Conv2D	64	3x3	2
Activation => relu			
MaxPooling2D		2x2	2
Conv2D	128	3x3	1
Activation => relu			
MaxPooling2D		2x2	2
Conv2D	128	3x3	1
Activation => relu			
MaxPooling2D		2x2	2
Flatten			
Dense => relu	512		
Dense => relu	256		
Dense => softmax	11		

Figure 7: Model A

Type	Neurons	Filter Size	Stride
Conv2D	64	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Conv2D	128	3x3	1
BatchNormalization			
Activation => relu			
Conv2D	128	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Conv2D	256	3x3	1
BatchNormalization			
Activation => relu			
Conv2D	256	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Flatten			
Dense => relu	1024		
BatchNormalization			
Dropout => 0.5			
Dense => relu	512		
BatchNormalization			
Dropout => 0.2			
Dense => softmax	11		

Figure 8: Model B

Type	Neurons	Filter Size	Stride
Conv2D	64	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Conv2D	96	3x3	1
BatchNormalization			
Activation => relu			
Conv2D	96	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Conv2D	192	3x3	1
BatchNormalization			
Activation => relu			
Conv2D	192	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Flatten			
Dense => relu	512		
BatchNormalization			
Dropout => 0.5			
Dense => relu	256		
BatchNormalization			
Dropout => 0.5			
Dense => softmax	11		

Figure 9: Model C

Type	Neurons	Filter Size	Stride
Conv2D	64	3x3	2
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Conv2D	128	3x3	1
BatchNormalization			
Activation => relu			
Conv2D	128	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Conv2D	256	3x3	1
BatchNormalization			
Activation => relu			
Conv2D	256	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Conv2D	512	3x3	1
BatchNormalization			
Activation => relu			
Conv2D	512	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Flatten			
Dense => relu	1024		
BatchNormalization			
Dropout => 0.5			
Dense => relu	512		
BatchNormalization			
Dropout => 0.5			
Dense => softmax	11		

Figure 10: Model D

Type	Neurons	Filter Size	Stride
Conv2D	64	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Conv2D	128	3x3	1
BatchNormalization			
Activation => relu			
Conv2D	128	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Conv2D	256	3x3	1
BatchNormalization			
Activation => relu			
Conv2D	256	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Conv2D	256	3x3	1
BatchNormalization			
Activation => relu			
Conv2D	256	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Conv2D	256	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Flatten			
Dense => relu	1024		
BatchNormalization			
Dropout => 0.5			
Dense => relu	512		
BatchNormalization			
Dropout => 0.65			
Dense => softmax	11		

Figure 11: Model E

Type	Neurons	Filter Size	Stride
Conv2D	64	3x3	2
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Conv2D	128	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Conv2D	128	3x3	1
BatchNormalization			
Activation => relu			
MaxPooling2D		2x2	2
Flatten			
Dense => relu	512		
BatchNormalization			
Dropout => 0.4			
Dense => relu	256		
BatchNormalization			
Dropout => 0.2			
Dense => softmax	11		

Figure 12: Model F