KAGGLE COMPETITION REPORT
TEAM META

Massine Juba Kadi 1852209
Department of Physic Engineering
Polytechnique Montreal
2500 Chem. de Polytechnique, Montréal, QC H3T 1J4
massine.kadi@polymtl.ca

Mostafa DadkhahKalateh 2153428
Department of Mathematical and Industrial Engineering
Polytechnique Montreal
2500 Chem. de Polytechnique, Montréal, QC H3T 1J4
mostafa.dadkhah@polymtl.ca

Morteza Asgari 2046947
Department of Software Engineering
Polytechnique Montreal
2500 Chem. de Polytechnique, Montréal, QC H3T 1J4
morteza.asgari@polymtl.ca

1 FEATURE DESIGN

We did not do a lot of feature design beside dividing pixel by 255 to have pixel values going from 0 to 1. And even this part was removed when we switched from TensorFlow layers of Data Augmentation (Data Augmentation v1) to using ImageDataGenerator (Data Augmentation v2). With ImageDataGenerator we used normalization and it was not useful to divide by 255 anymore. For the kernel size we use (3, 3), (5,5), (5,5) and (7,7) to capture more complex features as we progress through deeper in the network.

Our data augmentation consists of sample-wise standardization (mean-centring and rescaling with a standard deviation of 1) for validation and training. We also augmented the validation set with horizontal flipping, rotating, shifting in both directions, shearing, and zooming. This process helped avoid overfitting on the training set.

2 ALGORITHMS

We settled on using Convolution Neural Network (CNN) after doing a small literature review on the best models for image classification. Our reviews were almost unanimous on the choice of CNN. We still tested Logistic Regression, SVM, Random Forest and LSTM without anything past 0.50 accuracy.

We tried a lot of different architectures of CNN but settled with the best one we could find which had 22 layers with Batch Norm activated. It was composed of 4 Conv2D-BatchNorm-ReLU-MaxPool sequences, a flatten layer, one dense layer and an output layer with 11 neurons for the 11 classes. For the Conv layers, we went from 32 to 256 filters and from (3, 3) filter size to (7,7). We used "same" padding to keep the same size while going through the network to avoid negative dimensions and we used a kernel of (2,2) and stride 2 for the MaxPool layers following advice found on the internet (Ramesh (2021)). Finally, we used dense layers of 4 096 neurons between the flatten which had a dimension of 9 216 and the output layer of 11 neurons. In total this architecture had 39 662 219 total parameters and 39 661 259 trainable parameters.

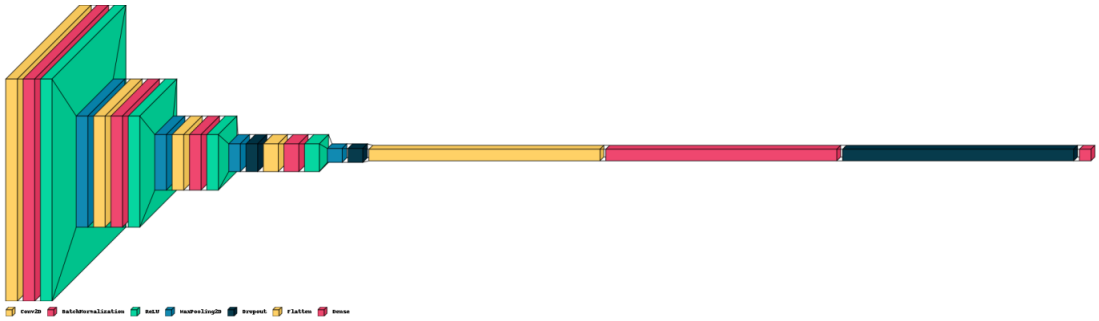

Figure 1: Final architecture

Using batch normalization before activation in each CNN layer helped. Without it, the training procedure was slower, and we did not reach F1-scores of more than 0.66. In fact, this takes care of the problem of vanishing gradient and it normalizes the data to be interpreted as a standard brightness and contrast of the resulting image.

In the prediction step, we used a variant of Ensembling Method in order to predict the input data. Basically, we took all validation examples, augmented them multiple times, ask the same network to give a prediction and used the most frequent label as output. For predicting the test set, we used six models which were trained on different folds of the training set. Two of them without class weights and four with weighting on losses based on the inverse ratio of their occurrence on the training set. Class weights were added in order to prevent the model to be biased in classes with more samples.

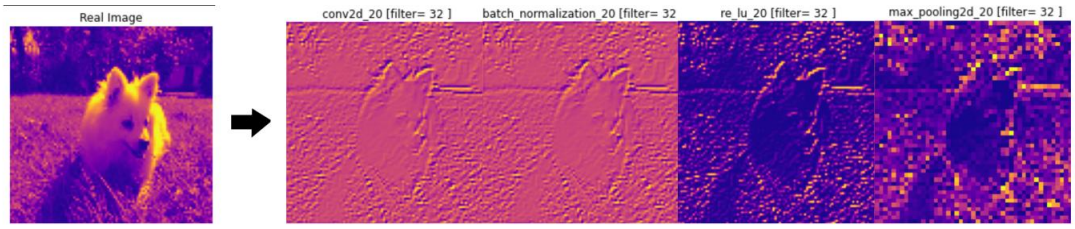

Figure 2: Feature maps vizualization

Here we can get a glimpse at how ConvNet is perceiving the raw images of different layers.

3 METHODOLOGY

The hardest part was to settle on an architecture that could both be powerful enough to reach high F1-score on training set and not be too complex so it still generalizes well on validation set. We tried a lot of different architecture from two million parameters all the way to 50 million. We used for this mainly two methods to go about searching.

First method was brute force Grid Search for the number of layers, the number of filters, the kernel size, the number of dense layers, etc. This search was not really successful and wasted us a lot of time. We realized the search space was too big to cover for any reasonable amount of time. We could sometimes get close to 10 000 different combinations. We switched to Random Search without finding great solutions. Moving to Colab Pro also helped a lot.

The second method was taking inspiration from the literature. We read some articles online on key tips to build efficient and powerful CNN(Ramesh (2021)), we found some custom design for similar use cases (input picture of 96x96x1) and looked at research articles and their corresponding architecture (VGG16, AlexNet, LeNet-5, etc).

This second method was the most effective and gave us our best model.

After settling on the architecture, we moved to a regularization search to find a way to overcome overfitting. We found online that L2+Dropout or Data Augmentation was a good way to counter overfitting in CNNs. We then needed to find the correct ratio of these elements. We did Hyperparameters Search for just the regularization terms with our fixed architecture. We started with a Grid Search then quickly moved to a Random Search to cover different regions of the parameters space. For L2 parameters we explored in logspace from 0.001 to 100 and then in linspace from 0.001 to 0.01 around the best value. For Dropout we explored from 0.0 all the way to 0.9 and found good results close to 0.5.

In the end the most effective method for us was using only Data Augmentation. We used ImageGenerator with shift=0.3, rot=30, zoom=0.3.

We also used K-folds with 4 stratified folds.

We tried several loss functions for the training purpose: Binary Cross-Entropy, Sparse Cross-Entropy, Categorical Cross-Entropy and Softmax F1 Loss(Maiza (2021)). The custom loss function calculated the two F1-scores for the true positives and true negatives and returns a weighted average of them. In the end, we found that binary cross-entropy had the best performance in terms of speed and F1-score. This is logical since it tries to reach the Softmax output to binary classes and choose the next step with more determination since the output is binary instead of probabilities.

4 RESULTS

We tried different models beside CNNs, as stated before: Logistic regression, SVM, Random Forest, LSTM, ResNet and DenseNet. They were all performing poorly as an example SVM was only able to achieve 0.26 F1-score on validation set.

We did a lot of hyperparameter tuning for architecture and regularization but in the end, as stated before, we found our architecture by taking inspiration from the literature and with the little time that was left to us we tested Data Augmentation and found some decent values.

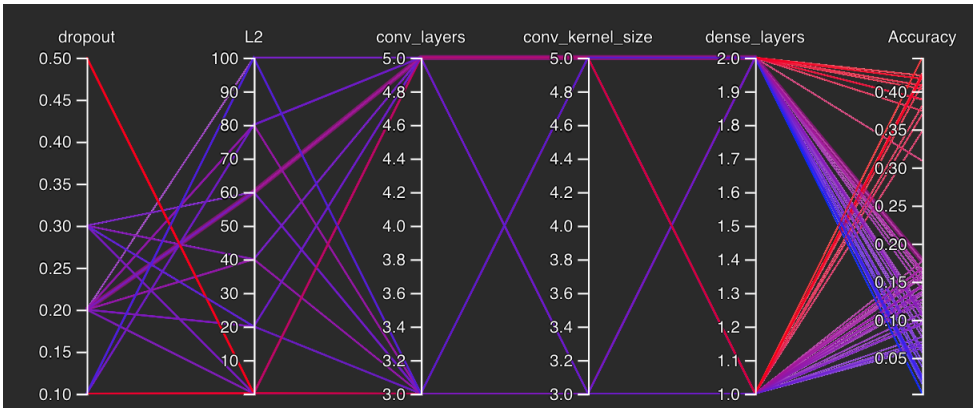

Figure 3: Accuracy vs epochs for different combinations of hyperparameters

Here we have a representation of the different hyperparameters and their respected accuracy.

dropout	L2	conv_layers	conv_kernel_size	dense_layers	Accuracy
0.10000	0.10000	3.0000	3.0000	1.0000	0.44701
0.50000	0.0010000	5.0000	5.0000	1.0000	0.44407
0.10000	80.000	3.0000	3.0000	2.0000	0.44029
0.10000	80.000	3.0000	5.0000	1.0000	0.43482
0.10000	0.0010000	5.0000	3.0000	2.0000	0.43440
0.10000	0.10000	3.0000	5.0000	2.0000	0.43398
0.50000	0.10000	5.0000	5.0000	1.0000	0.43103

Figure 4: Best combinations of hyperparameters

Here are the best combinations of the hyperparameters from one search for both architecture and regularization.

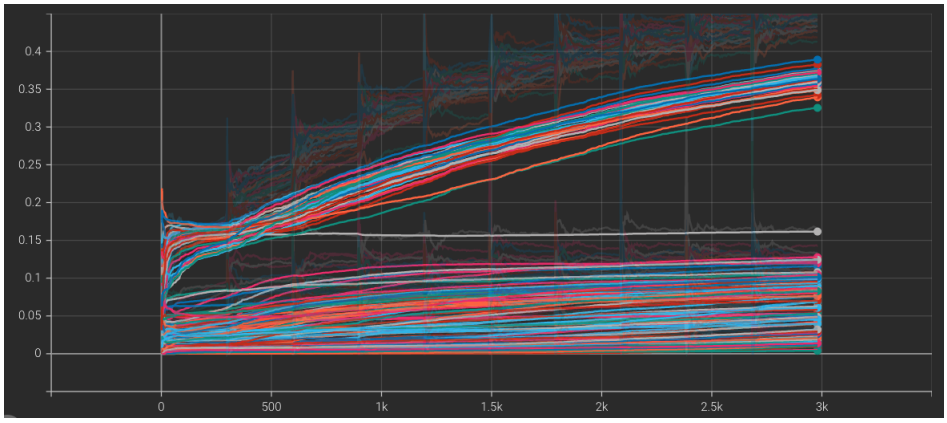

Figure 5: Batch accuracy vs learning step

Here is the evolution of batch accuracy for different learning steps and for all combinations of one of the search which had 123 runs.

Before Data Augmentation we reached 1.00 of F1-score on training set and 0.66 on validation set. After implementing Data Augmentation, it did not exceed 0.86 on training set and 0.75 for validation set which represent an increase of almost 0.10 for validation set.

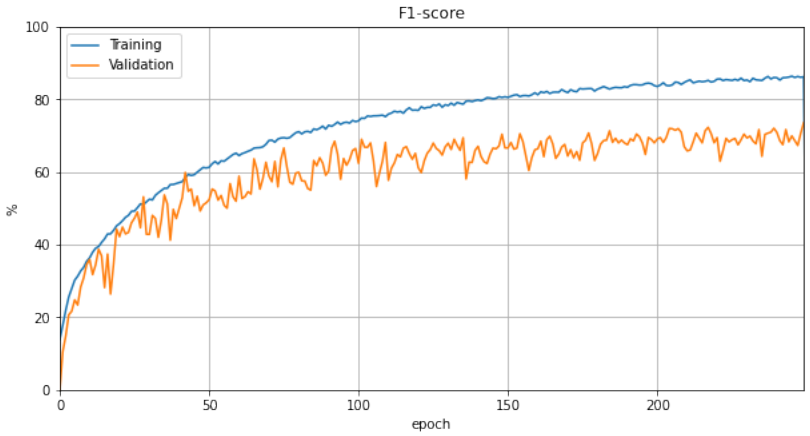

Figure 6: Training and validation F1-score

Here we see the learning curves for both training and validation which are typical with a smooth increase for the training F1 and a more erratic curve for validation.

We got 0.77 F1-score on validation set with the base model and 0.79 when we augmented the validation set 30 times with different combinations of Data Augmentation parameters.

By combining our six best models, we got our final score of 0.82512 in Kaggle by augmenting 50 times the test set. Which makes us the 9th team out of 33.

5 DISCUSSION

On the technical part of this project the difficulty was to settle on the architecture to use. We spent a lot of time and computation searching for a good architecture without founding a model that could compare with the first teams which had already 0.8 F1-score at the time. And it was the most discouraging part since we did not feel as if we had enough knowledge of CNNs to search effectively the parameter space which is almost infinite. Then we had the realization that we could just inspire ourselves on the literature and so then we found our best architecture. For future work the take away is to spend more time on reading the literature before starting any implementation or continuously reading during the implementation. Another aspect that was hard was information storing. We ran a lot of searches and trained a lot of models and we did not, until very far in the project, settled on a way to store models and performances. In future work it would be good to start with this part well implemented in the pipeline in order to never waste time at rerunning old results.

On the organization side and team work, we encounter many issues in splitting evenly the workload. Some of us worked almost daily on the project while others not as much. Another issue was that it wasn't until late in the project that we really shared our knowledge of the problem and so we ended up redoing many steps multiple time within the team. Advice for future work would be to hold everyone responsible for their part of the work and address any workload unbalance early on in the project.

6 STATEMENT OF CONTRIBUTIONS

Research:

- Methods to solve image classification [Massine, Mostafa, Morteza]
- Convolution Neural Network arhitecture [Massine]

Implementation:

- Data importation and exploration [Massine, Mostafa]
- Data augmentation v1 [Massine]
- Data augmentation v2 [Mostafa]
- Models Creation [Massine]
- Hyperparameters Search [Massine, Morteza]
- Training [Massine]
- Evaluation [Mostafa, Massine]
- Ensemble Method [Mostafa]

Computation :

- Hyperparameters search of both architecture and regularization [Massine]
- L2 Training [Morteza]
- Training differents models with ColabPro [Mostafa, Massine, Morteza]
- Testing simple CNNs [Massine]
- Testing Random Forest and SVM [Mostafa]

- Testing ResNet and DenseNet with random initialization [Massine]
- Testing LSTM [Morteza]
- Testing VGGNET19 [Morteza]

Contribution in the report:

- Feature design [Massine, Mostafa]
- Algorithms [Massine, Mostafa]
- Methodology [Massine, Mostafa]
- Results [Massine, Mostafa]
- Discussion [Massine]

Brainstorming [Massine, Mostafa, Morteza]

REFERENCES

A Maiza. The unknown benefits of using a soft-f1 loss in classification systems, 2021. URL https://towardsdatascience.com/the-unknown-benefits-of-using-a-soft-f1-loss-in-classification-systems\-753902c0105d.

Shashank Ramesh. A guide to an efficient way to build neural network architectures-part ii: Hyper-parameter, 2021. URL https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii\-hyper-parameter-42efca01e5d7.