Kaggle Data Competition 2 : Image Classification task

Daniel Galarreta-Piquette (# p0975627) & Lissethy Cevallos (# 20170443) → Team Name : Sur America

1 Introduction

For the *IFT6390* class, we did a second Kaggle data competition where we had to solve an image classification task. The goal was to predict one of the 6 classes (ant, spider, flower, bulldozer, lobster and dolphin) of hand-drawn scribbles that come from the quick-draw dataset. It was required to solve this problem using three different families of classifier and were able to use any python library.

In this document we detail how we first pre-processed the features, select the hyperparameters and for each tested model we will present the accuracy obtained and finally we show the selected algorithm as best and its characteristics compared with the others learning algorithms.

So we did a total of three experiments : A Convolutionnal Neural Network, a Support Vector Machine and lastly a K-nearest neighbour. In the end our best model was the CNN with a 83% accuracy, while the SVM did 49% and the K-NN was a third place with 45 % .

2 Dataset and metrics

We have a data set that contains images from QuickDraw classified into 6 categories. We are going to use 250 images from each category as our training set, and 60 000 images as the test set.

For this task, the performance of the classification models was evaluated using accuracy (i.e. : the number of correctly classified instances divided by the total number of instances).The Kaggle platform provided us with the final test accuracy.

3 Feature Design

Before implementing any algorithm, we need to process the image and we detail the steps as follows: cleaning the data and augmenting the data.

3.1 Cleaning data

Our data set was represented by a 1500x2 matrix, a 1500x784 feature matrix and the target vector of dimensions (1500,1).

Regarding the feature matrix, we did:

- Normalize each feature by subtracting the mean and dividing by standard deviation of the training data
- And for some models we reshape the feature matrix to (1500,28,28,1)
- Regarding to the target vector, we converted them as one-hot vectors

3.2 Augmenting the data

A common practice when the data is not sufficient is to synthetically generate new data from our original dataset [1]. By increasing our training data set we most likely get a more robust model.

To this aim, we reverse the columns pixels (Flip-horizontal), rotate the image clockwise by a given number of degrees from 0 to 360 (Rotation), shift the image to the left or to the right (Horizontal shifts) and shift the image to up or down (vertical shifts).

4 Algorithms

4.1 Convolutionnal Neural Network

Convolutionnal Neural network are part of the neural network family of functions. The basic idea is to take as input an image(in this case a scribble), and then apply layers of filters and function to the image to learn the features of the image. With each added layer, we are able to have more abstract representations of high-level (like what shapes are important in a spider). In the end, the network will have an understanding of the image similar to how a humain brain works.[2]

Those filters will be applied on every surface of the (28,28) until we decided it's enough with the convolution layers. Afterwards, we have a fully connected part where we will condense the representations in neurons and finally output the softmax for each class, hoping that the label with the highest probability is the true label of the image that we are predicting.

This is an example of what a convolutionnal neural networks can be represented for the hand-written digits of the MNIST dataset [1]

We will talk more about the architecture of our CNN network in the methodology section, but we can already see that CNN architecture can be more complex than just layers of convolutions and fully connected layers.

[1]https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

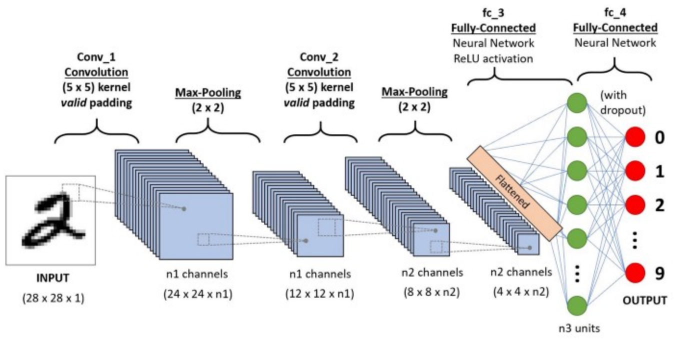

Figure 4.1: Example of a Convnet with MNIST dataset

4.2 SVM Classifier

We also decided to implement a SVM classifier as another hypothesis for a model, because it can do well with non-linear decisions when we apply the kernel trick.

This classifier aims to learn K hyperplanes defined by an affine function that maximizes the margin from the decision boundary between the kth class and all the other possible classes.

We also used the regularization term called C that controls the tolerance to test points that were mis-classified. The bigger the value of C is, the bigger the tolerance to mis-labelling. And considering our dataset, we had an intuition that we might need a considerable size for C.

Using Scikit-learn's SVM implementation, we decided to use the non-linear SVM by applying the *kernel trick*. This trick is computationnally great because it can implicitly project the data points in a higher dimension space; making the decision boundary not linear anymore. And image classification is not a linear type of decision boundary.[3] .$K\left(\mathbf{x}_i, \mathbf{x}_j\right) = \langle\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}$ where $\phi(\mathbf{x}$)

The idea is to replace all the dot products with a kernel function K. What happens is this :

$$g(\tilde{\mathbf{x}}) = \langle\tilde{w}, \tilde{\mathbf{x}}\rangle + b$$
$$= \langle\tilde{w}, \phi\left(\mathbf{x}\right)\rangle + b$$
$$= \langle\sum_{i=1}^{n} \alpha_i \phi\left(\mathbf{x}_i\right), \phi\left(\mathbf{x}\right)\rangle + b$$
$$= b + \sum_{i=1}^{n} \alpha_i \langle\phi\left(\mathbf{x}_i\right), \phi\left(\mathbf{x}_j\right)$$
$$= b + \sum_{i=1}^{n} \alpha_i K\left(\mathbf{x}_i, \mathbf{x}_j\right)$$

4.3 K-NN

The last algorithm that we tried is the K-Nearest Neighbour, a non-parametric model for machine learning. We were curious as to see how such a simple model might do with image classification tasks. [4]

The idea is that, for a test point, we will check all the nearest neighbours (decided by the value of K and a metric distance) and make them vote to decide which class the unseen image will belong to.

The algorithm can be resumed like this :

$$f(x) = argmax(\frac{1}{k}\Sigma_{i\in 1\ldots n|X_i \in V(x)} onehot_m(Y_i))$$

5 Methodology

All models were trained with the same ratio of 2/3 training and 1/3 validation, and all parameters and hyper-parameters were chosen based on a k-fold cross-validations for each different change. All of the accuracies are averages of k-fold cross-validations.

First, we have to indicate that Scikit-learn library was used and before testing some models we implement training and validation split, where we kept 33% of the data as validation and the rest for training.

5.1 CNN

The ConvNet is most commonly applied to analyzing image, due to its translation invariance characteristics and their shared-weights architectures. The model was tried with different numbers of hidden layers with the same activation function (*Relu* activation) and pool function (*Max-pool*). We say that *Relu* activation ($f(x) = max(0, x)$), is one of the simplest non-linear activation that is applied element-wise to the pre-activation neurons and helps us speeding up the training, while *Max-pool* helps us identify the maximum value within a specified neighbourhood without overlapping.[5]

Finally, we use fully connected layer with *SoftMax* activation whose results are easy to interpret , such as the probability that the class is equal to i (from 1 to 6 in our case) given the data.

There are two important subjects we have to mention, one of them is that we implemented data augmentation that was explained in *Features Design Section* and another subject is talking about the optimization of our loss function. Regarding to this last subject, Stochastic gradient descent method based on adaptive estimation of the first and second order moments, called Adam, was used in order to minimize the Cross-entropy loss function, setting the number of epochs. This is how we decided to calculate our weights.

Finally, we also added two important parameters to our architecture, namely the *Dropout* and *Batch Normalization*. The first one is a mechanism to prevent overfitting. It acts as regularizer to our model architecture. Now the second one normalizes the input layer by adjusting and scaling the activation, it reduces the covariance shift. One benefit of batch normalization is to speed up the process of the learning. We saw an improvement when used both of those processes on our layers and we decided to keep this architecture for our final model.

Lastly, to find the best model, we decided to monitor the validation accuracy, which will be at its higher point when our loss is at its minimum. So after a number of epochs (250), we thought it was enough to see if we had found our best accuracy. We then saved the weights of the model that gave us the best accuracy. We then recreated the architecture of our model, but then using only the best weights for the prediction phase.

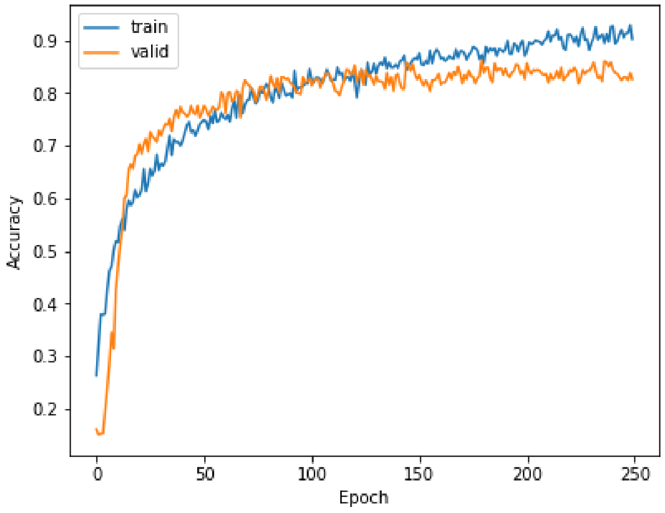

Figure 5.1: Validation accuracy for different epochs

We observe that our training and validation accuracies are both going up as the epochs are getting up, but at some point the validation accuracy kind of stays flat while the training keeps getting higher. Most likely, the training accuracy will get to almost 100% if we keep training for a long long time, but at that point we will be for sure in the overfit part of the process. So, by monitoring the validation accuracy, we know that as soon as it stops going up for a long time, and then starts to go down, we know that the highest point in validation accuracy is the *sweetspot* between underfitting and overfitting.

We thought that it was also interesting to see how our CNN models learns the features of an image. The following images 5.2, 5.3 are the feature maps of our model for a *bulldozer*. We see that in the first convolution layer, the tractor still appears as a tractor and is very recognizable. But as soon as we get *deeper* in the network, we notice that the model is learning the representation at a higher-level and way more abstract features (borders, corners and angles). So this is what happens in our model, we apply filters and focus on regions, abstract knowledge from it until we are ready to condense that knowledge in the fully connected layers.

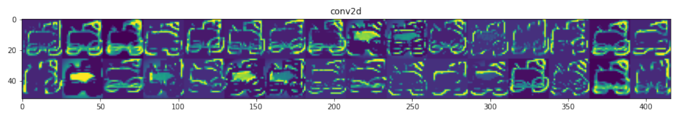

Figure 5.2: Feature maps with the 1st convolution

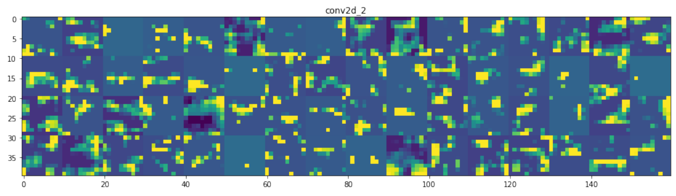

Figure 5.3: Feature maps with the 2nd convolution

5.2 K-NN

The K-NN principle assumes that data points that are close to each other should belong to the same cluster, but we need to define the number of points considered to classify the new point. We validate some K from 5 to 30 and we obtain the best accuracy with $K = 15$, but its accuracy was 45%. Which is the worst of all our tested models.

The figure in 5.4 demonstrates how we tried to implement the best K-NN. We did some pre-processing (normalization, data augmentation, reshaping (28,28) , one-hot encoding). We then tried to find for which value of K do we get the best accuracy on our validation set.

5.3 SVM

Like it was stated in the *Algorithms section*, these non-linear classifier models essentially depend on some support vectors and a hyperplane that help us divide the data into classes. It

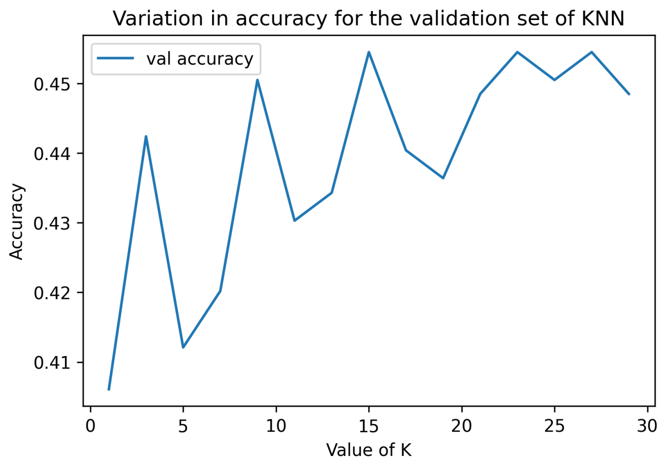

Figure 5.4: Validation accuracy for different values of K

selects the hyperplanes that maximizes the distance of some points which are the closest to the hyperplanes.

Through cross-validation and grid-search, we selected some hyperparameters. In our case we set $KFold = 5$ and it showed us that the best values of C and *gamma* were $C = 5$ and $Gamma = 0.001$ and that the kernel was of *Radial Basis function*, but with this technique we could only get an accuracy of 49%.

6 Results

These results correspond to the mean of a k-fold cross-validation on validation set. The ConvNet model performs way better than the KNN and the SVM. If we look at the results here:

Model	Accuracy
CNN	83.6%
SVM	49%
K-NN	45%

We will investigate the differences in accuracies in the next section. Now, let's take a look at the actual performance on each class of our models. This table will help us distinguish the confusion matrices

Class number	Class name
0	spider
1	ant
2	flower
3	dolphin
4	lobster
5	bulldozer

6.1 A visual idea of each models' performances on all 6 classes

The confusion matrices 6.1, 6.2, 6.3 , that follow, show us where we did good and where the predictions got mislabelled. If we take a look at the three of them we can notice some similarities. Class 2 and 3 have been the most easily distinguishable between all three of our models. If we look at our table with the labels and numbers, we see that 2 and 3 are flower and dolphin. Indeed, we are not surprised that dolphin has been predicted so well accross the board considering that its shape is really different from all the other classes. We are surprised though that flowers had features different enough for the model to recognize them and not mislabel them because it could also look like an ant or a spider in terms of features.

Now, looking at our best model, we notice that the 4^{th} class is where the major part of the loss is. The *culprit* is the class 1. So that means that lobsters and ants have similar features. If we think about it and look at some of the images for those specific classes, it is not that surprising. They share a certain roundness and straight lines.

If, the classes were different enough, may be the SVM and the K-NN could have done better performances. We see that they are able enough to predict two out of 6 classes. May be the features of the other classes were not different enough to distinguish them. Or may be it is that those two classes have more of a certain universality when people draw them. It is possible that the variety in scribbles for the other classes was higher than for class 2 and 3. This is where the convolutional network is really the best for such tasks. It was able to recognize features even when the drawings have differences.

7 Discussion

In the end, the CNN was our best model. Those models are really good because they can handler high dimensionnality quite well. Instead, K-NN don't handle high dimensionality that well. They are more suited for a task with less dimensions, but one of their advantages is the intpretability. If we compare to CNN, neural network in general are more complex and less interpretable. Biggest advantage of CNN is, as mentionned before, it's ability to recognize features or patterns within classes to such an abstract level, that slight changes in the images will probably not affect its performance so much.

One of the reasons KNN did not so well, is may be because it relies on the distance between the vectors to make a decision. Images are represented as vectors, and if a spider is drawn a certain area in the 28x28 matrix with a small changes in

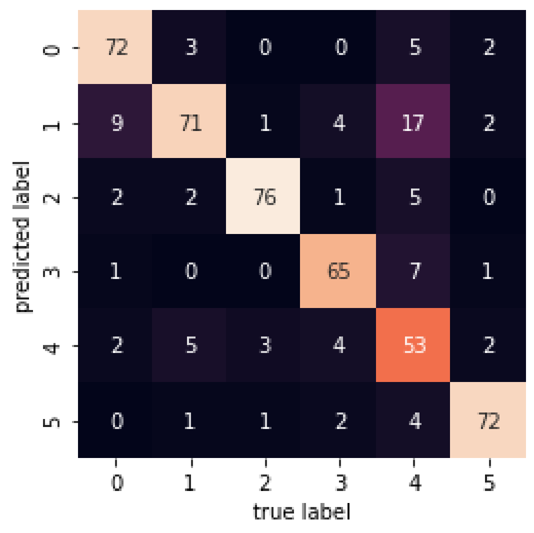

Figure 6.1: Confusion Matrix for 6 classes with CNN

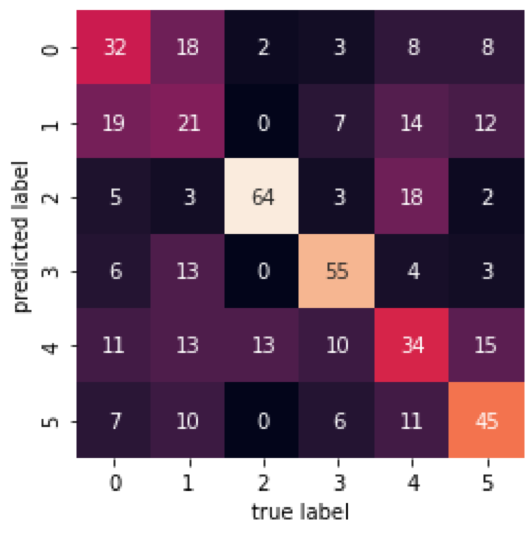

Figure 6.3: Confusion Matrix for 6 classes with SVM

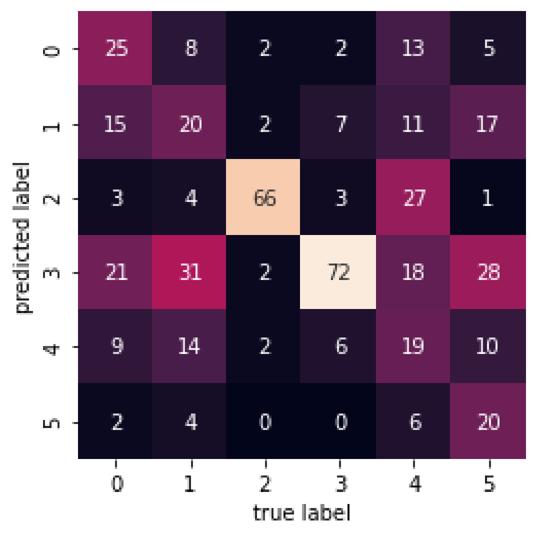

Figure 6.2: Confusion Matrix for 6 classes with K-NN

rotation, translation, viewpoint, scale, that can influence the image , then both those vectors might be different enough for the K-NN to mislabel it. This explains why it did not perform so well. There is not a lof of learning involved in this model, it's just going with the majority vote within a certain area..

Now for the SVM, it is able to put the features onto a higher dimensions to make non-linear decision boundaries, which is a good advantage of SVM. Because, in some areas, the data is not linearly separable. In our case, we thought that SVM might do okay, but it was not even close to the CNN. May be this is due to the fact that we did not find the best hyperplanes to distinguish one class from the others when we implemented our SVM. It is possible that even more pre-processing was necessary in order for SVM to work with this dataset.

Finally, an improvement to our task might have been to use transfer learning from pre-trained neural networks. The concept is quite simple, if we use a pre-trained CNN on thousands of images and classes, this model can re-use some of its knowledge on new unseen data and not learn from scratch all the features in our dataset. It already has an idea of what kind of features exist, and how they can be used to recognize new classes.

8 Statement of contributions

- Daniel G.-P. did : CNN implementation in Python, data augmentation solution, and worked on the second half of the report. (+ ReadMe file)
- Lissethy C. did : SVM implementation in Python, run tests of K-NN, worked on the first half of the report.

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

References

[1] Jason Brownlee. Machine learning mastery, making developers awesome at machine learning.

[2] Christopher M. Bishop. Neural networks. in pattern recognition and machine learning.

[3] Pascal Vincent. Kernel trick - lecture notes in IFT6390. Universite de Montreal, October 2020.

[4] Trevor Hastie Robert Tibshirani. Gareth James, Daniela Witten. *An introduction to statistical learning: with applications in R*, April 2013. http://www.ctan.org/pkg/booktabs.

[5] F. Chollet and others. Keras.