Homework 4: Report on the Kaggle Competition

TEAM NAME: 404NotFound
Adel Nabli (20121744), Myriam Laiymani (20140876),
Louis-François Préville-Ratelle (708048), Pierre Rosin (20025653),

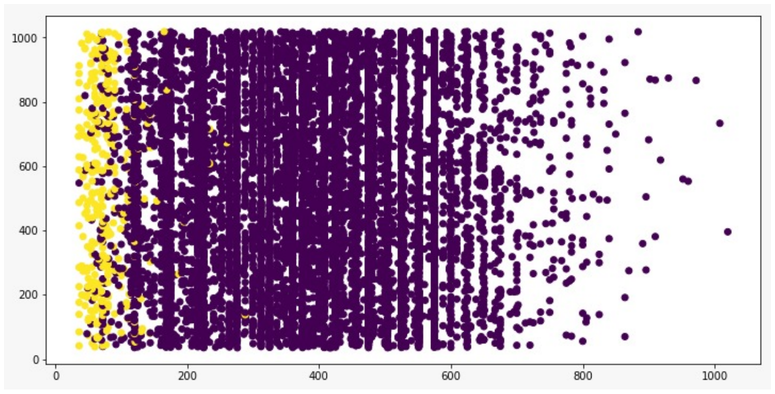
October 13, 2019

1 Introduction

The subject of image classification is a very important topic in Machine Learning. In this Kaggle competition, the goal is to classify human drawings among randomly generate noise. There are 31 categories in total. After the prepossessing steps, we tested a number of classification models. We will present the results of the best performing model, which is a convolutional neural network, and of the SVM model as a baseline.

2 Feature Design

The noise is a nuisance in the images. This is not that important for the CNNs, but it is more for other algorithms like the SVM. To remove most of it, we need to find the connected components inside each image. The pixels are the vertices. Two pixels share an edge if they are adjacent and they both have intensities above a certain threshold (that we determine). It is a standard algorithm to find all the connected components in a graph. Before removing the noise, we need to be able to distinguish it from the drawings. We noticed that the drawings are usually bigger than the noise, meaning that the area of the box containing the connected component of the drawing is usually bigger than all the connected components made of noise. So by keeping only the biggest connected component, we can get rid of most of the noise.



Areas of the boxes containing the biggest connected component of a given image. In yellow the empty images, in blue the images with a drawing.

The images left consist usually of a small drawing, without all the noise. We then reduce the images from 100 x 100 down to 44 x 44 (the biggest connected component encountered in the set is of size 42 x 42) to concentrate on the drawings. Finally, we center each drawing and re-scale it to make sure each drawing occupies all the space available in the image frame and augment the contrast to get rid of some blurry artifacts.

For the SVM, we added to the "image vector" 4 simple features that helped augment by 2% the final classification accuracy with that model:

- Area of the box containing the drawing in the original image
- Length of that box according to the x axis

- Length of that box according to the y axis
- Number of "holes" contained in a given drawing *(The intuition behind being that we can expect a pineapple drawing having more holes than an apple one. We used the `scikit-tda` toolkit to perform this task).*

Finally, we used data augmentation to improve the results of the CNN slightly. On each image in the training set, we performed a reflexion along the y-axis to double the training size. We also experimented some other techniques of data augmentation.

3 Algorithms

3.1 SVM

In its simplest instance, the SVM is a linear model used for classification and regression. It can be used to classify data that is not linearly separable. The kernel trick allows the SVM to become a non-linear model by operating in a higher dimensional space without paying the computational cost. This model will be as a reference for the classification.
We used the Radial Basis Function (RBF) kernel SVM with hyperparameters γ and C. The parameter γ defines how far the influence of a single training example reaches, hence controls the capacity of the model, whereas C acts as a regularization parameter. Our optimal values were $\gamma = 2$ and C=1.

```
SVC accuracy:  0.528
CPU times: user 3min 55s, sys: 433 ms, total: 3min 55s
Wall time: 3min 54s
```

3.2 CNN

For the CNNs, we design a standard convolution model using three combinations of double convolutions followed by max pooling. Batch normalization was used heavily after each convolution. We add at the end of the model two fully connected layers and a cross entropy loss. The relu activation function was performed after each convolution and after the first fully connected layer. That model was obtained from trials and errors until the results were satisfactory. We could have made our model smaller, but we kept it big because training was relatively fast. Here is a description of the CNN model that we used:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 200, 44, 44]           2,000
       BatchNorm2d-2          [-1, 200, 44, 44]             400
            Conv2d-3          [-1, 200, 44, 44]         360,200
       BatchNorm2d-4          [-1, 200, 44, 44]             400
            Conv2d-5          [-1, 100, 22, 22]         180,100
       BatchNorm2d-6          [-1, 100, 22, 22]             200
            Conv2d-7          [-1, 100, 22, 22]          90,100
       BatchNorm2d-8          [-1, 100, 22, 22]             200
            Conv2d-9          [-1, 200, 11, 11]         180,200
      BatchNorm2d-10          [-1, 200, 11, 11]             400
           Conv2d-11          [-1, 200, 11, 11]         360,200
      BatchNorm2d-12          [-1, 200, 11, 11]             400
           Linear-13                  [-1, 500]       2,500,500
      BatchNorm1d-14                  [-1, 500]           1,000
          Dropout-15                  [-1, 500]               0
           Linear-16                   [-1, 31]          15,531
================================================================
Total params: 3,691,831
Trainable params: 3,691,831
```

```
Non-trainable params: 0
________________________________________________________________
```

4 Methodology

For the SVM, we added the 4 features computed to each image vector before splitting the dataset into 3 different sets (train, validation, test) to tune the hyperparameters of the SVM. Once we found a good combination of hyper-parameters, we retrained the model using 70% of the data for training and 30% to test.
For the CNN, We used 90 percent of our data to train the model and the remaining 10 percent for the validation set. We did try several combinations of hyperparameters and model architectures to find a good combination. For the regularization, we used two techniques. We kept a dropout rate of 0.7 at the end of the model, and as mentioned previously, we used batch normalization extensively. Also, since our model is quite big, we had to use early stopping (around 12-17 epochs) since the model was overfitting a lot on the training data if given too many epochs. For gradient descent, we used Adam optimizer with its standard hyperparameters.
Once we found our best model, we retrained on the full dataset for around 15 epochs before submitting our predictions on Kaggle's website.

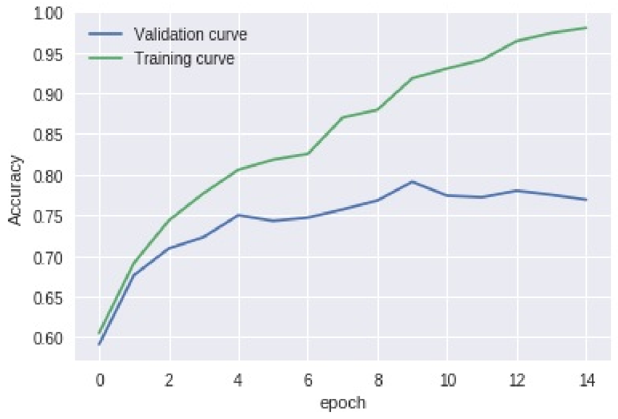
5 Results

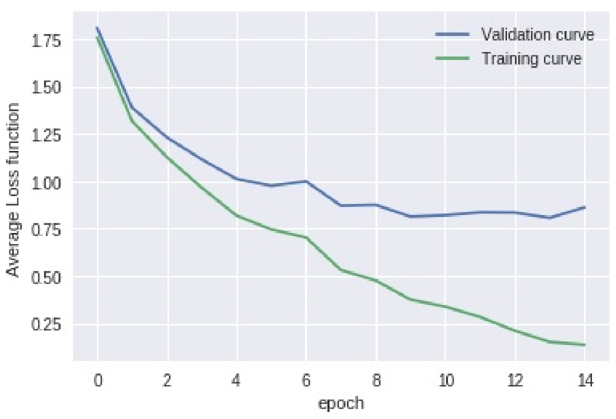
For the SVM, we had a classification accuracy of 52.8% on our test set. We also tried to see where it performed best and where it performed poorly, and the impact of adding the 4 features on the classification capability on each class. Adding the features resulted in a global increase of the accuracy, with a difference worth noting on the "empty" class:

	peanut	**mug**	**penguin**	**empty**	...	**screwdriver**	**shovel**	**parrot**	**nose**	**pear**
Without the 4 features	0.15	0.53	0.58	0.39	...	0.09	0.49	0.04	0.59	0.51
With the 4 features	0.12	0.60	0.59	0.96	...	0.11	0.51	0.07	0.58	0.51

Classification accuracy of the SVM on several classes

For the CNN, we obtained an accuracy of 78.2% on the reduced testing set on the Kaggle's website. This was roughly the same accuracy that we were getting on the validation set. We present the accuracy curves and the average loss curves here:





6 Discussion

Convolutional neural networks are designed to capture local features in images and their hierarchical relationships to form more complex structures. For this reason, they are very good at recognizing images, and therefore this learning algorithm was our method of choice for the present task of drawing classification. Since we had only 10000 data points, we were not sure if other learning algorithms like SVM would not perform better for this task. We were not able to reach the same accuracy with the SVM as with the CNN.

In terms of performance, batch normalization was extremely useful to reduce the training time. In particular, it seemed like a lot fewer epochs were needed to train the model. The learning was more robust to hyperparameter changes. Also, the double convolutions seemed to give slightly better results than single convolutions, although this assertion is uniquely based on a few trials, therefore it might be wrong.
To conclude, we would like to mention a few possible improvements. First and most importantly, our preprocessing methods keeps noise instead of drawings sometimes. A more careful metric to distinguish between noise and drawings would probably give a substantial improvement for the accuracy (or using directly the CNN on the original images). Second, a redressing algorithm could potentially give better results for the SVM. Third, we could have used more data augmentation techniques like cropping or slight rotations. Fourth, we could have trained several different learners and use a combination of them for predictions (ensemble methods). And last, we could have reduced our model slightly since it was overfitting, but it is not clear it would have increased accuracy by much.

7 Statement of Contributions

Adel implemented almost all the preprocessing steps. Adel and ML worked on the SVM and several other classifiers. ML, LFPR and PR played around extensively with the CNNs. In particular, they tried transfer learning for a while since there was some uncertainty if this method was accepted. A post on Studium confirmed that this method is not tolerated. We didn't submit any predictions using a transfer model on Kaggle. The four authors wrote the report.
We hereby state that all the work presented in this report is that of the authors.

8 References

Union Find Structure Design:
http://code.activestate.com/recipes/215912-union-find-data-structure/