IFT 6390 - FONDEMENTS DE L'APPRENTISSAGE MACHINE
FALL 2018

HOMEWORK - 4 : KAGGLE COMPETITION

CONVOLUTIONAL NEURAL NETWORKS FOR CLASSIFICATION OF NOISY HAND DRAWN IMAGE

TEAM NAME

ShadowBrokers

TEAM MEMBERS

Anirudh K (20093112)
Marulasidda Swamy K S (20116267)
Naveenkumar A R (20098468)
Pravish Sainath (20125633)

1 Introduction

The goal of the homework was to implement a classification algorithm that can automatically predict the categories given human drawn images. The given dataset consists of 10,000 images in each of the training and testing sets. Each image is of size 100px x 100px and these images have been modified by transformation with added noise in the images. There are 31 categories in total.

We tried different baseline linear classifiers like linear SVM and logistic regression but the SVM alone performed better among the several baseline classifiers with 4.76% as test accuracy. We also worked on random forests algorithm and a CNN model. The CNN model performed better than any other machine learning algorithms with the test accuracy of 76.73% as test accuracy. Additionally, we tried denoising the input images for improving the performance of the CNN which was marginal.

2 Feature Design

- Each item of the input data was reshaped to 100 x 100 arrays
  Reason : To maintain the spatial relationship between the pixel values.

- Conversion to Python Imaging Library (PIL / Pillow) format
  Reason : To transform the data to be interpreted as images for the CNN training

- Data augmentation was done with a variety of operations :
  - Random horizontal flipping
  - Random vertical flipping
  - Random rotations
  - Random rescale

  Reason : To enrich the dataset with possible variants of the input images in order to allow the CNN filters to be robust enough to learn and predict the labels even if the images are slightly rotated or flipped or in a different size.

  CNN models are translation invariant but are prone to variation in rotation and Scaling. So the networks need to

- Normalization of image pixel intensity values :
  The pixel intensity values of the images are subtracted by their sample means values and scaled by their sample standard deviation values.

  Reason : To transform the input pixel values to a reasonable range to help the neural network to learn faster since gradients act uniformly.

- As a CNN model is being used, the features are not hand-designed and the features with their hierarchies are learned by the filters of the convolutional layers of the CNN during the training process.

- The input images were denoised using Rudin-Osher-Fatemi (ROF) algorithm to test with the CNN.

3 Algorithms

Given below are overviews of the learning algorithms used without going into too much detail.

3.1 Linear Support Vector Machines (L-SVM)

Given a training dataset of n points : $\{(x_i, y_i)\}_{i=1 \dots n}$ where $y_i = 1$ or -1 class labels

- Find the "maximum-margin hyperplane" that separates the data points belonging to each of the two classes. This optimal plane is the one that maximizes the distance between the hyperplane and the nearest point from either group as the set of points ***x*** satisfying :

$$\vec{w} \cdot \vec{x} - b = 0$$

w is the normal vector to the hyperplane.

- For soft-margin SVM, the hinge loss is defined as : $\max\left(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b)\right)$
- Training involves solving $\min_w \left[\frac{1}{n}\sum_{i=1}^{n} \max\left(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b)\right)\right] + \lambda\|\vec{w}\|^2$

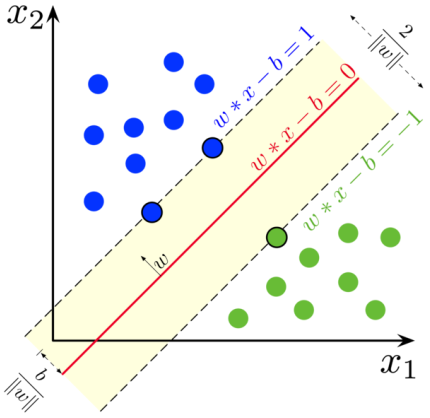


Figure 1 : Optimal hyperplane of an SVM, Source : Wikipedia

3.2 Convolutional Neural Networks (CNN)

- Given a set of images, design a sequence of convolution and pooling layers with respective activations.
- For each set of input(s), perform forward propagation performing the convolution and pooling operations to find the output.
- Compute the cost using the computed output and actual target of the input data and backpropagate the gradients to update the weights.

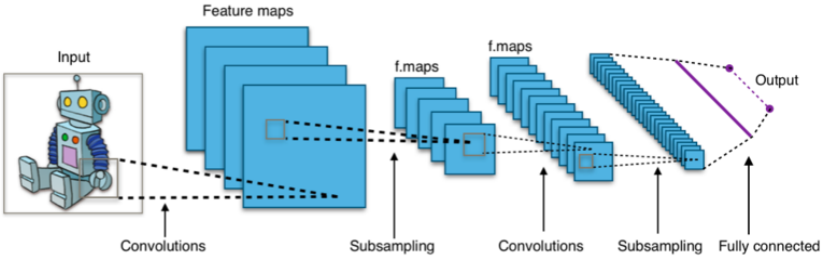


Figure 2 : General architecture of a CNN, Source : Wikipedia

4 Methodology

4.1 Training / Validation Split

The following was the split of the given data containing 10000 images :

training samples = 7000 images

validation samples = 3000 images

As there are good enough samples across all 31 classes of drawings, this split was good enough.

4.2 Network Architecture

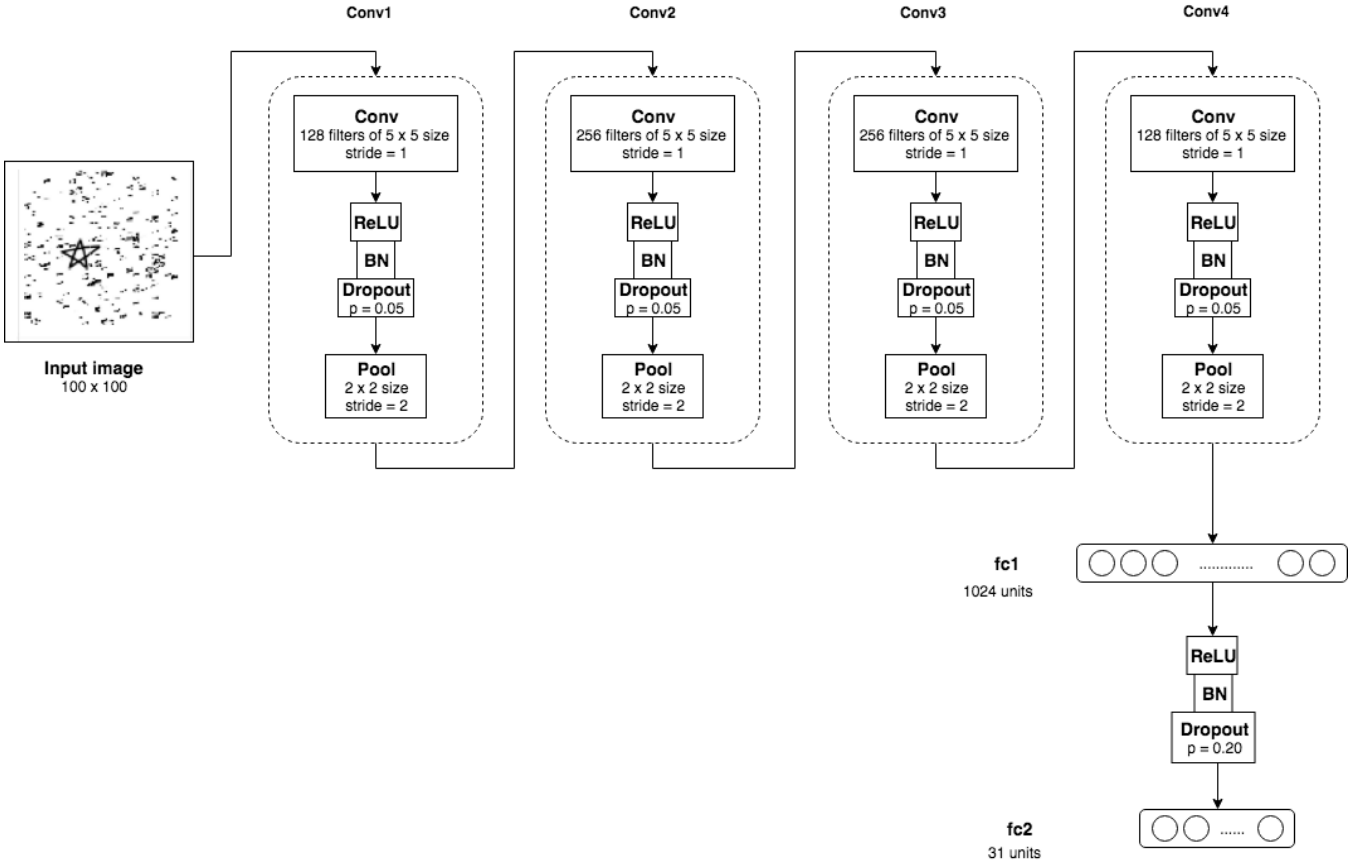


Figure 3 : Architecture of the best performing CNN

4.3 Hyper-parameters

The following hyperparameters were tuned using either cross-validation, random search or inspiration from the existing benchmarks of the Google QuickDraw dataset.

- Number or layers = 4 (Conv + Pool) layers + 2 fully-connected layers
- Number of weights / connections / filter sizes as indicated in Figure 3.
  Conv layers : 128 (5 x 5) filters -> 256 (5 x 5) filters -> 256 (5 x 5) filters -> 128 (5 x 5) filters
  (Inspired by CNNs for QuickDraw but added one more layer for improvement)
- Pool layers : Max Pooling with 2 x 2 filters with stride 2
  (Common choice for natural images)
- Number of fully-connected neurons = dimension of the final output from Conv + Pool layer
- Batch size = 150 (Set by trial-and-error performances)
- Learning rate = 0.025 (Tuned randomly with multiple trials)
- Momentum = 0.9 (Tuned randomly for fastest convergence)

4.4 Optimization Tricks

- Cost function : Cross Entropy Loss (CE) between the output and target label vector
- Optimizer : Stochastic Gradient Descent (SGD) optimizer with momentum was used.
  (to achieve more efficient optimization)

4.5 Regularization Strategy

- Batch Normalization : Adjusting and rescaling the activations to avoid covariance shift over consecutive layers by normalizing over batch of samples used
- Dropout : Randomly ignores certain weights with probability p, 0.05 for Conv layers and 0.2 For fully-connected layers.

5 Results

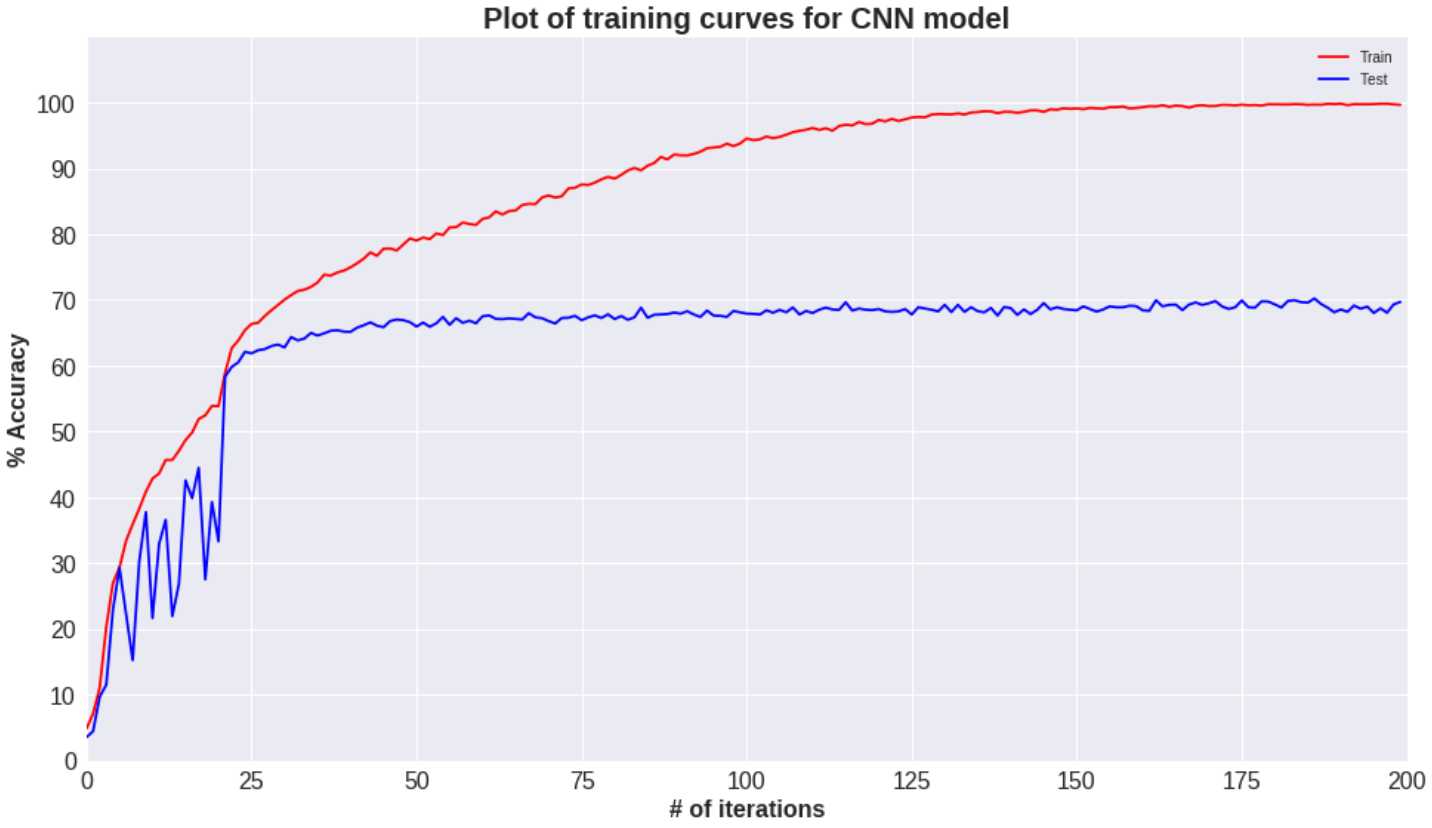
We present our results one the simple baseline and the method that gave us the best performance.

5.1 Baseline Model : Linear SVM

```
Train accuracy using Linear SVM (C = 1000) =  6.28
 Test accuracy using Linear SVM (C = 1000) =  5.04
```

5.2 Best Model : CNN

```
Train accuracy using CNN = 99.65714285714286
Test accuracy using CNN  = 69.69999999999999
```



5.3 Results Comparison : SVM and CNN variants

Method	Validation Accuracy	Public Score	Private Score
Baseline Linear SVM (C = 1000)	0.50400	0.04557	0.04766
CNN (200 epochs)	0.69699	0.77128	0.76733
Denoising + CNN (200 epochs)	0.71333	0.77142	0.76733

6 Discussion

We discuss why your best method performs better than the simple baseline and which design choices where the most impactful on performance.

- Higher Capacity :
  The CNN used for our model has a higher capacity than Linear SVMs.
  It is evident from the generalization error of the Linear SVM model that it underfits the given data that is complex and multi-dimensional.
  Also, the given data is not necessarily linearly separable in the very high dimensional space which will essentially cause the Linear SVM to fare poorly.
  The CNN has multiple neurons and layers and had much higher capacity than the baseline.

- Representational Power :
  The CNN model learns the representations of the image over the layers in the training procedure. The initial layers learn lower level features like edges, corner and the higher layers compose these to identify contours, boundaries, segments and objects.
  On the other hand, Linear SVM was given the array of raw pixels of length 10000 as inputs and it merely tried to optimize the hinge loss and fit a hyper plane than learn features.

  As the input data samples are images, CNN is a good choice for a model.

- Regularization :
  In the CNN model implemented, regularization was achieved by using dropouts and batch normalization. This ensured that the model did not end fitting the noise more than the data, especially given that the input images samples are distorted with some noise.
  In the case of SVM, the feature space is very huge and consists of 10000 dimensions and with the given data, it might have generalized to the noise present in the images in the absence of mechanisms to enforce regularization.

7 Statement of Contributions

The implementation of the CNN model and coding the solution for the same was done by both Marulasidda Swamy and Pravish Sainath. The data analysis and implementation of baseline models was done by Pravish Sainath, Anirudh K and Naveenkumar A R. In addition, the preparation of the report was done by both Pravish Sainath and Naveenkumar A R. In all, it was a team effort.

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

8 Acknowledgements

First and foremost, we would like to extend our gratitude to our instructor Prof. Ioannis Mitliagkas and all the Teaching Assistants involved in managing the logistics and setup for organizing this very cool Kaggle contest. It was a wonderful learning experience.

We are grateful to Google Inc. for their generous provision of access to GPUs on the Google Colab platform. It immensely helped us to improve our models and be productive.

9 References

http://www.isikdogan.com/blog/how-to-design-a-convolutional-neural-network.html
https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c
https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5
https://stackoverflow.com/questions/49035156/pytorch-how-to-use-topilimage-correctly
https://www.coursera.org/lecture/deep-neural-network/dropout-regularization-eM33A

http://www.cs.utah.edu/~ssingla/AdvIP/Image%20Denoising/Rof.html#Implementation
http://www.ipol.im/pub/art/2012/g-tvd/article_lr.pdf