A Simplified Version of Google's "Quick, Draw!" Game

Uroš Ogrizović

Software Engineering and Information Technologies, Faculty of Technical Sciences, University of Novi Sad

Introduction

Google's "Quick, Draw" is an online game developed by Google that challenges players to draw a picture of an object or idea and then uses a neural network artificial intelligence to guess what the drawings represent [1].
In this, simplified version of the game, only six types of drawings (i.e. six classes/labels) can be predicted:

- airplane
- alarm clock
- ant
- axe
- bicycle
- The Mona Lisa

Additionally, the prediction isn't done in real-time, but rather after clicking a button.
Three models give their predictions for each drawing that gets submitted by the user.

Models

- Vanilla CNN - a convolutional neural network consisting of 13 layers, excluding the input layer. Dropout was used to avoid overfitting. The kernel's dimensions are 3x3, which is an often-used kernel size. The architecture visualization for this model can be seen in Fig. 1.

  It was trained on both 10,000 images per label and 100,000 images per label. The latter case brought no noticeable improvement.

  Train/validation accuracy/loss plots for this model can be seen in Fig. 2.

- SVM - a value of 1 was used for C, the penalty parameter of the error term, which denotes the amount by which misclassification should be avoided, thus regulating the width of the hyperplane. 'rbf' was the value used for the parameter whose name is *kernel*. The RBF kernel was chosen because, unlike the linear kernel, it can handle the case when the relation between class labels and attributes is nonlinear, whereas it has fewer hyperparameters than the polynomial kernel. A value of 'scale' was used for *gamma*, the RBF kernel coefficient, which defines how influential each training example is.

- VGG19 - a convolutional neural network consisting of 24 layers, excluding the input layer. However, instead of using VGG19's fully connected layers, I used my own, because my problem doesn't have 1000 classes. Additionally, I had to pad Google's 28x28 images to 32x32 images, because this model doesn't accept images smaller than 32x32.

  This model uses 3x3 convolution filters. Its predecessor, VGG16, achieved state-of-the-art results in the ImageNet Challenge 2014 by adding more weight layers compared to previous models that had done well in that competition.

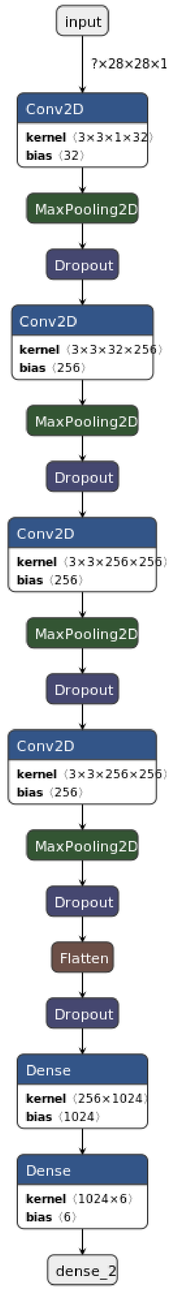

Fig. 1: Vanilla CNN model plot

Results

Looking at the 8th epoch of the vanilla CNN 10k model loss plot, one can conclude that the model suffers from underfitting, seeing that the training loss is ~ 0.8, and the validation loss is ~ 0.3, i.e. the training loss is significantly higher than the validation loss.
Similarly, looking at the 7th epoch of the vanilla CNN 100k model loss plot, one can conclude that the model suffers from underfitting. However, after the 7th epoch, the validation loss for the 100k model explodes, i.e. the model overfits. Luckily, thanks to using ModelCheckpoint, only the best model (i.e. the one with the smallest validation loss) is saved, so the saved model doesn't actually suffer from overfitting.

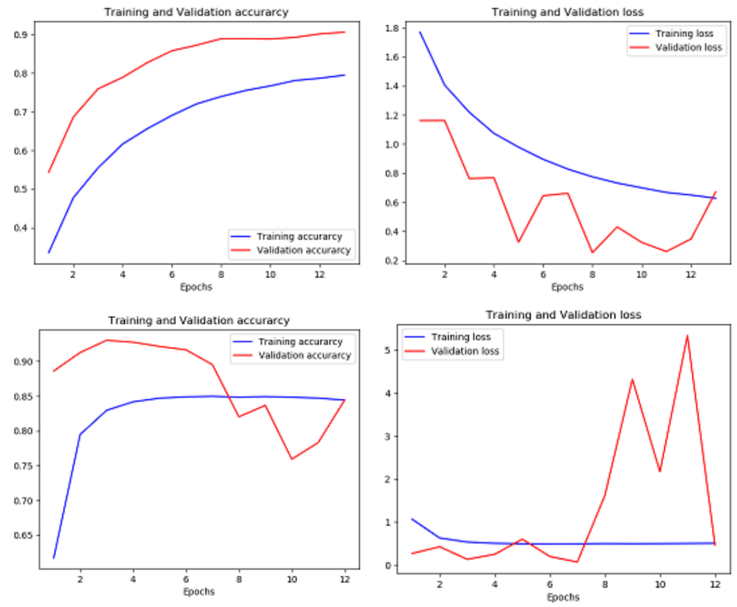

Fig. 2: Train/validation accuracy/loss for 10k (top) and 100k (bottom) versions of the Vanilla CNN.

As for the SVM model, the training was very slow; from docs: *"The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples."* Although the training time was decreased by scaling the pixel values to a [0, 1] interval, the training was still slow. This particular model doesn't work well on this problem.

Discussion

The vanilla CNN model has proved to be too simple for a problem this complex. The same applies to the SVM model. VGG19 gives the best results.
Perhaps the performance of the vanilla CNN model could be improved by adding skip connections [2].
The performance of the SVM model might be improved by performing a "grid search" on C and *gamma* using cross-validation [3].
In conclusion, it seems that the problem can best be tackled by using complex CNNs similar to VGG, such as Inception or ResNet.

References

References

[1] "Quick, Draw!" wiki

[2] Deep Residual Learning for Image Recognition (2015)

[3] A Practical Guide to Support Vector Classification (2003)

[4] GitHub repository