Big Data Content Analytics

Sketch Recognition

Kosmidi Antonia BAFT1707
Koutsoukou Dimitra Evangelia BAFT1708
Soukias Nikolaos BAFT1714
Tsopelakou Aglaia BAFT1716
Charalampidou Eleni BAFT1718

September, 2018

Contents

Description

The aim of this project is to use deep learning techniques in order to explore its capabilities in the field of visual recognition. More specifically we will focus on sketch recognition which is the automated recognition of hand-drawn sketches by a computer.

This is a very interesting domain as sketches provide an efficient way of communication among people over various cultures and different languages. Drawings in historic places, that are thousands of years old, show that people can express visual contents of objects using sketches without the need for a structured language. Currently, sketches can be utilized for various applications such as sketch-based retrieval of images and 3D models. The commercialization of touch devices makes it easy to draw a sketch on a smart phone or tablet, and now sketches can be used as tools for communication between humans and machines.

The rapid development of technology has made the implementation of this concept feasible. Nowadays computers can be trained to recognize and classify hand-drawn sketches accurately.

Hand-drawn sketches are ubiquitous in design, arts, education and entertainment. More recently sketching has also been receiving attention as a natural human-computer interaction modality as seen from the continually increasing body of work on automated sketch recognition. Training a computer to identify sketches could be also useful for online shopping, if a customer wanted to draw out a product and have the database recognize it.

Mission

Sketch-based image search is a well-known and difficult problem, in which little progress has been made in the past decade in developing a large-scale and practical sketch-based search engine. 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. Our target is to use this data to implement sketch recognition for educational purposes. Specifically, we want to use symbols or characters for translation.

Problems related to object recognition constitute a broad category of computer vision. Recently, the ability to recognize real world objects using rough hand-drawn sketches has been of particular interest. The MindFinder system performs image search based on a rough sketch provided by the user. The MindFinder system has been built by indexing more than 1.5 billion web images to enable efficient sketch-based image retrieval, and many creative applications can be expected to advance the state of the art.

Our approach to the sketch recognition problem is to train a CNN model using data from the Quick Draw dataset of Google. This model will be trained to identify the class in which the sketch belongs.

Data

The Quick Draw Dataset is a collection of 50 million drawings across 345 categories, contributed by players of the game Quick, Draw!. The drawings were captured as timestamped vectors, tagged with metadata including what the player was asked to draw and in which country the player was located. You can browse the recognized drawings on quickdraw.withgoogle.com/data.

We decided to work on 6 categories, yoga, dog, flipflops, bandage, airplane and bear. Yoga category contains 280.442 sketches. Dog category contains 152.159 sketches. Flipflops category contains 121.518 sketches. Bandage category contains 147.614 sketches. Airplane category contains 151.623 sketches. Bear category contains 134.762 sketches. So the whole dataset consists of 988.118 images.

Sample yoga drawings

Sample dog drawings

Sample flipflops drawings

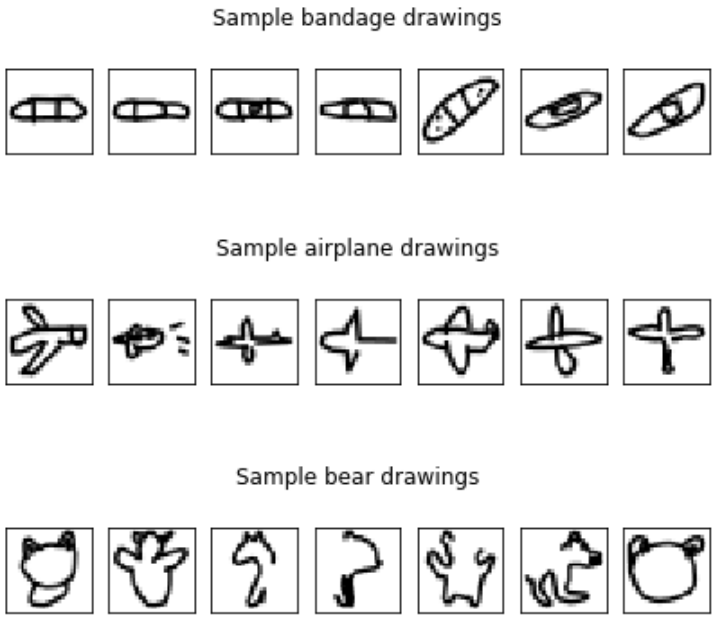


Figure 1

Initially the dataset was made up of the sketches drawn by the users. Google has already processed the data and transformed them in 28x28 grayscale bitmap in numpy format. So, even if there are many pre-processing techniques for image data, such as mean image subtraction, per-channel normalization, per-channel mean subtraction, dimensionality reduction etc. it was not necessary in our case to use them.

Finally, our dataset contains 918.118 images in 28x28 grayscale bitmap in numpy format. In figure 1, we can observe some sample draws of each category of the six and in figure 2 and 3 we can see some specific draws for yoga only.

Figure 2

Figure 3

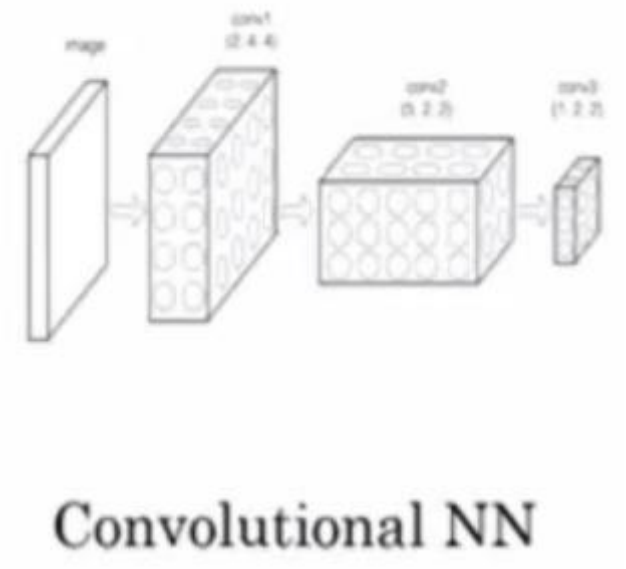
Methodology

In order to train the model we need to split the dataset to train and test. The first one is used to train the model and the second to provide an unbiased evaluation of a *final* model fit on the training dataset. In this case the dataset is split into 70% train and 30% test. The training dataset consists of 42.000 observations while the test of 18.000.

The Model CNN

CNN is the most popular supervised deep learning model for pattern recognition and more specifically image recognition. So we used CNN (convolutional neural networks) because as in other computer vision problems, convolutional neural networks (CNNs) outperform other feature extraction methods significantly in sketch recognition.

CNNs, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. The whole network has a loss function and all the tips and tricks that we developed for neural networks still apply on CNNs. Unlike other neural networks, where the input is a vector, here the input is a multi-channeled image.



The Process of building the Model

In order to build our model we used Tensorflow, a Python library with lots of power and flexibility. We also used Keras which is a high level library and operates over Tensorflow.

Keras offers the option to use the Sequential structure to build our model. By using the Sequential form, the layers are getting stacked linearly and in order (from input to output). While constructing the model, we should specify the input shape, only at first layer (input shape= (1, 28, 28)), because the following layers do automatic shape inference.

In all our model implementations, we compiled the model using :

1. **Loss function** (the objective that the model tries to minimize each time): We used the standard cross entropy for categorical classification.

2. **Optimizer:** we chose Adam optimizer, an improved algorithm of SGD for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments.

3. **Metrics** (the function used to judge the performance of our model): Since, it is a classification problem we used accuracy.

Convolution layers:

The convolution layer is the main building block of a convolutional neural network. It comprises of a set of independent filters. Each filter is independently convolved with the image and we end up with n (n represents the number of filters) feature maps of shape 28*28*1.

Pooling layer:

Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer operates on each feature map independently. The most common approach used in pooling is *max pooling.*

The **FC** is the fully connected layer of neurons at the end of CNN. Neurons in a fully connected layer have full connections to all activations in the previous layer.

Activation function:

An artificial neuron calculates a "weighted sum" of its input, adds a bias and then decides whether it should be "fired" or not.

So consider a neuron.

$$Y = \sum (weight * input) + bias$$

Now, the value of Y can be anything ranging from -inf to +inf. The neuron really doesn't know the bounds of the value.

We decided to add "activation functions" in order to check the Y value produced by a neuron and decide whether outside connections should consider this neuron as "activated" or not.

ReLu

In our case the activation fuction that we used is the ReLu function,

A(x) = max(0,x)

The ReLu function gives an output x if x is positive and 0 otherwise.

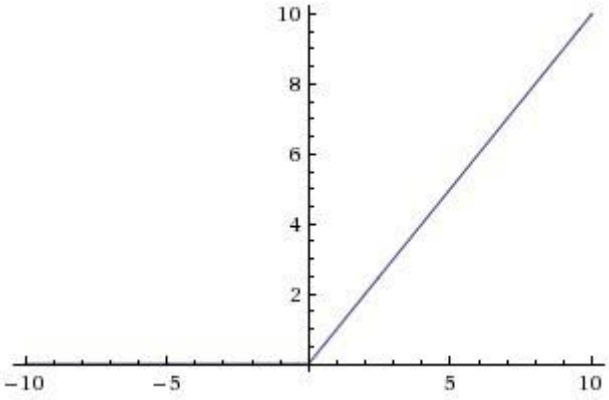


Figure 4

First, ReLu is nonlinear, is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. That is a good point to consider when we are designing deep neural nets.

Architecture of the CNN model

Our model consists of 9 layers. Below, all these layers are analyzed:

- First Layer(Convolutional)

The first layer is convolutional, and its input are images of (28X28), the convolution is 30 filters (5x5) using ReLU activation function and the output are 30 tensors (28x28).

- Second Layer (Pooling)

The second layer is pooling where the input is 30 tensors(28x28) from the previous layer and the output are 30 tensors again but with (2x2) sizing. That is because pooling layer is used to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.

- Third Layer(Convolutional)

The third layer used is again convolutional, with input 30 tensors(2x2) from the previous layer. The convolution is using 15 filters and ReLU activation function. The output is 15 tensors(2x2).

- Fourth Layer(Pooling)

In the fourth layer we used our last pooling layer with input 15 tensors using 1 pooling filter and the output is 15 tensors(2x2).

- Fifth Layer(Dropout)

In this layer the input is 15 tensors with rate=0.2. The rate was used to prevent overfitting. So, in this layer we drop out 20% of input units.

- Sixth Layer(Flatten)

In this layer Flatten was used. Flatten is our regularizer.

- Seventh Layer(Dense)

In this step we used dense function with activation function ReLU. This step returns the output tensor with the same shape as inputs except the last dimension is of size units (units=128).

- Eighth Layer(Dense)

In the eighth layer we did dense again with ReLU but with unit=50.

- Ninth Layer(Dense)

In our last layer we used dense again but with Softmax and unit= number of classes. Softmax was used so as to find the similarity of the input to the classes (6 classes).

Finally, we compile the model using Adam optimizer and the loss function categorical crossntropy. The equation for categorical cross entropy is

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}\mathbf{1}_{y_i \in C_c}\log p_{model}[y_i \in C_c]$$

The double sum is over the observations `i`, whose number is `N`, and the categories `c`, whose number is `C`. The term `1_{y_i \in C_c}` is the indicator function of the `i`th observation belonging to the `c`th category. The `p_{model}[y_i \in C_c]` is the probability predicted by the model for the `i`th observation to belong to the `c`th category. When there are more than two categories as in our case, the neural network outputs a vector of `C` probabilities, each giving the probability that the network input should be classified as belonging to the respective category.

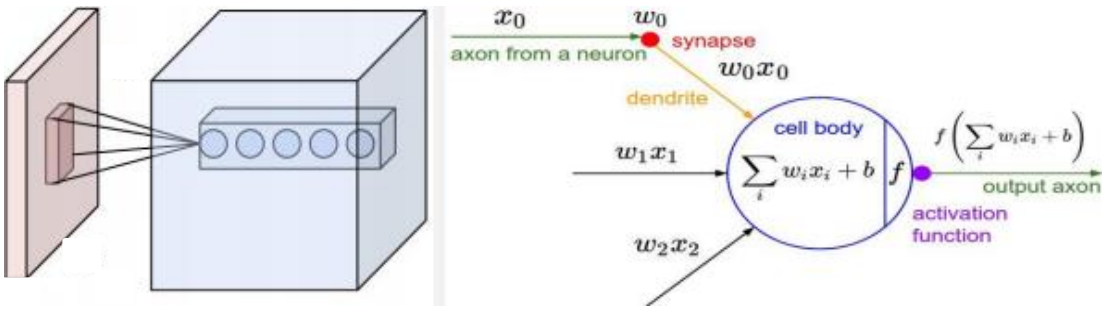


Figure 5

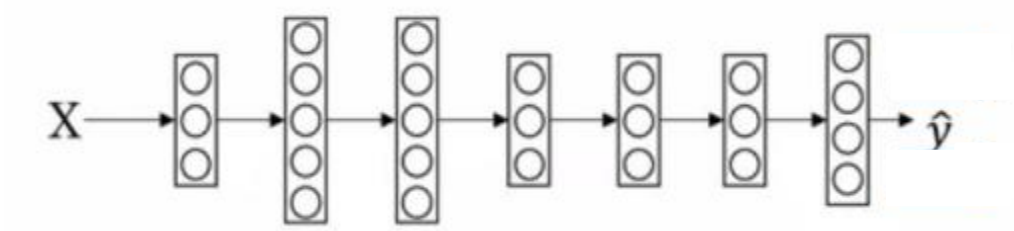


Figure 6

Input features

Feature engineering - is transforming raw data into features/attributes that better represent the underlying structure of your data, usually done by domain experts.
Feature Extraction - is transforming raw data into the desired form.

In our case, our input is feature extraction because our raw data were transformed into the desired form automatically. As mentioned above, Google has already processed the data and transformed them in 28x28 grayscale bitmap in numpy format.

Parameters

The parameters of our neural network model are 59729 and we don't have any non-trainable parameters. Our hyper parameters used for choice activation were the function ReLu and Softmax. The batch size was 50, our training samples were 42000, our epochs were 20 and we did 40 iterations. Since we used Adam evaluation method, we used learning rate a=0.2 and the beta1, beta2 and epsilon were the default settings. So, the values are a=0.2, beta1=0.9, beta2=0.999 and epsilon=10−8.

Evaluation Measures and Optimizers

The choice of optimization algorithm for your deep learning model can mean the difference between good results in minutes, hours, and days. The Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing. So, to optimize our loss function, we used the AdamOptimizer, which is a popular optimizer, like Stochastic Gradient Descent and AdaGrad.

Summary of the Model

The summary of the model is the following. For each epoch and for each batch in our data, we run our optimizer and loss function against our batch of data. To keep track of our loss at each step of the way, we added the total loss per epoch up. For each epoch, we output the loss, which should be declining each time. This can be useful to track, so you can see the diminishing returns over time. The first few epochs should have massive improvements, but after 5, you will be seeing very small, if any, changes.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 30, 24, 24)        780
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 12, 12)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 15, 10, 10)        4065
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 5, 5)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 5, 5)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 375)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               48128
_________________________________________________________________
dense_2 (Dense)              (None, 50)                6450
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 306
=================================================================
Total params: 59,729
Trainable params: 59,729
Non-trainable params: 0
_________________________________________________________________
```

Figure 7

```
Epoch 20/20
42000/42000 [==============================] - 1271s 30ms/step - loss: 0.2377 - acc: 0.9124 - val_loss: 0.
4247 - val_acc: 0.8678
Final CNN accuracy:  86.77777777777777 %
Model is saved
Wall time: 7h 14min 20s
```

Figure 8

Results

Learning Curves

The following graph demonstrates the evolution of the accuracy of the model within the 20 epochs. The blue line represents the train data while the orange the test. The peak performance of the accuracy of the train dataset is approximately 92%, while the same measure for the test dataset is about 86.78%. We observe that the accuracy of the train dataset is higher in general, but this result is unreliable as the accuracy of the train dataset is the result of overfitting. For this reason we focus on the accuracy of the test dataset. It can be seen that for both datasets the accuracy is increasing till the epoch 5, then for the train dataset the accuracy is still increasing but with a

decreasing rate while for the test dataset the accuracy remains stable with few fluctuations.

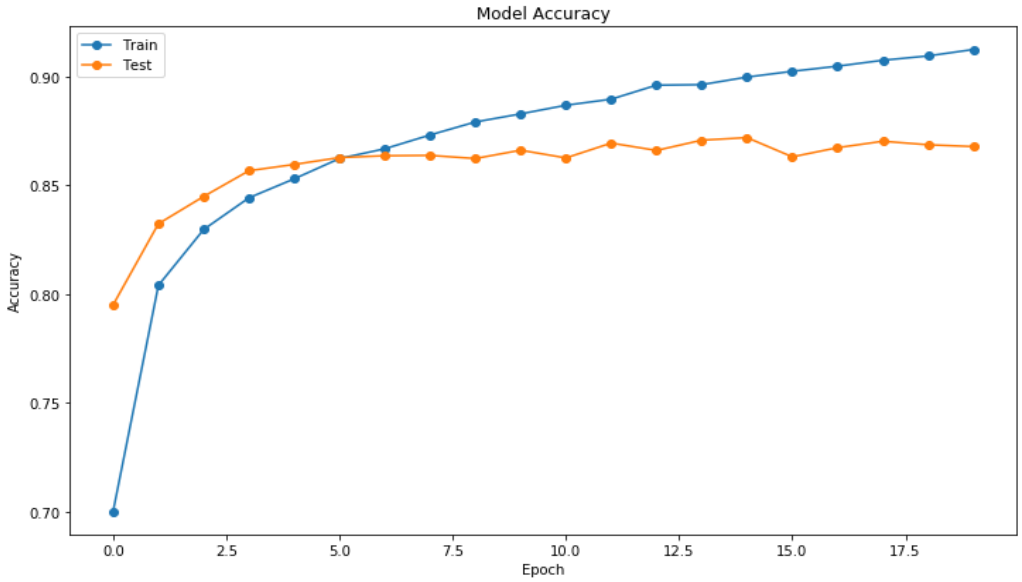


Figure 9

The following graph demonstrates the evolution of the loss of the model within the 20 epochs. The blue line again represents the train data while the orange the test. We observe that the loss of the train dataset is less than the loss of the test. It can be seen that for both datasets the accuracy is decreasing till the epoch 5, then for the train dataset the accuracy is still decreasing but with a decreasing rate while for the test dataset the accuracy remains stable with few fluctuations.

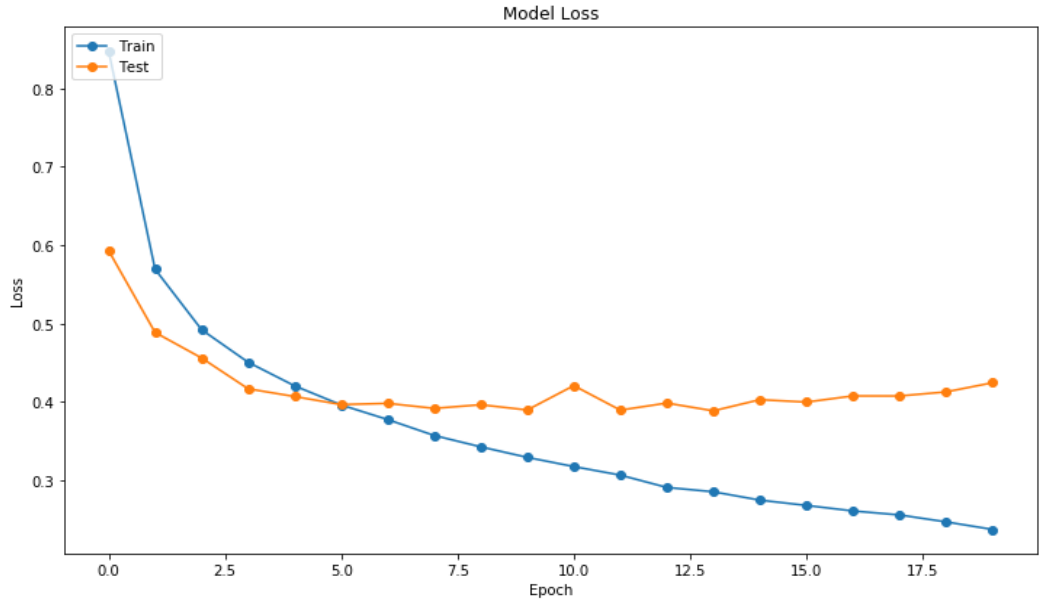


Figure 10

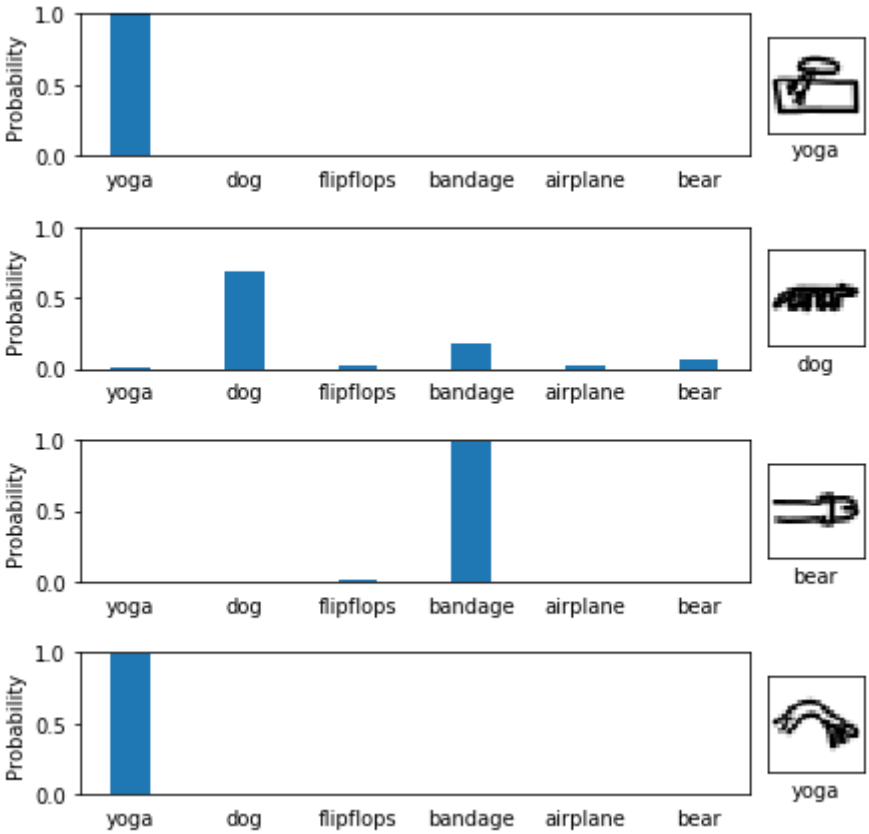
Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. The valid values (those that are correctly classified) are those of the main diagonal. In the following confusion matrix we observe that the majority of the elements is correctly classified and only a small proportion of them is misclassified. For example, 2516 sketches of the test dataset are correctly predicted as yoga, 122 sketches are predicted as dog while they belong to the yoga category, 52 sketches are predicted as flipflops while they belong to the yoga category and so on.

```
Confusion Matrix:
[2516  122   52   75   36  192] (0) yoga
[  80 2381   36   26   31  421] (1) dog
[  48   50 2761   60   16   47] (2) flipflops
[  73   71   81 2687   42   68] (3) bandage
[  63   86   16   42 2761   47] (4) airplane
[  68  311   54   42   24 2514] (5) bear
 (0) (1) (2) (3) (4) (5)
```

Figure 11

Testing the Model



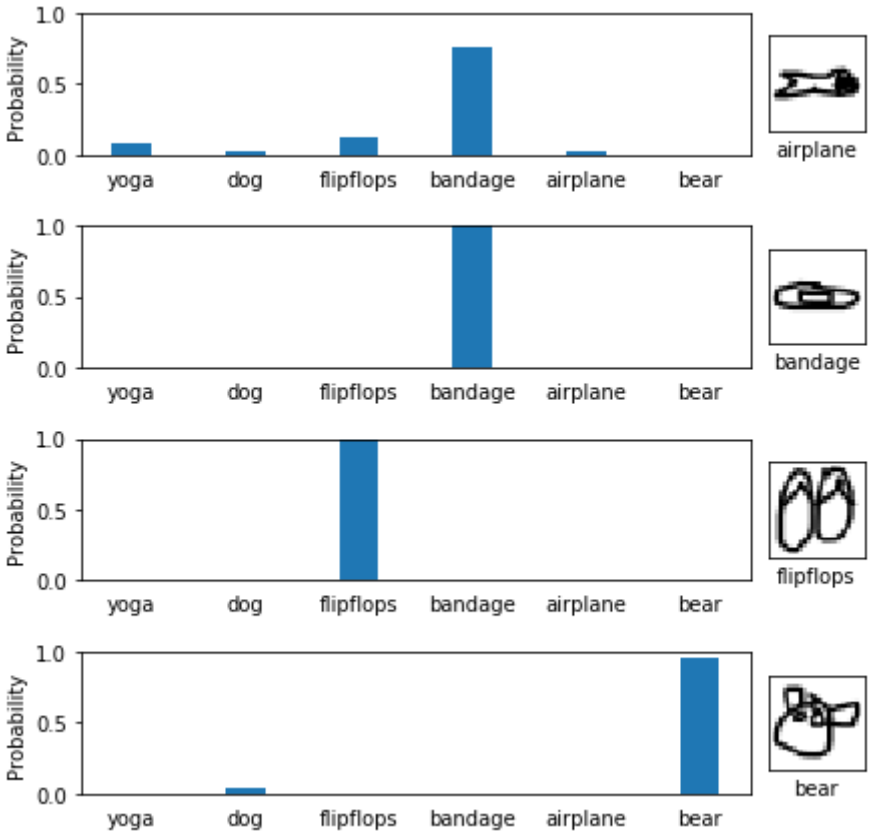


Figure 12

The graph below demonstrates the percentages of classification and misclassification of the first eight sketches.

The first drawing is **yoga** and is identified as **yoga** with a probability of 99.995636940 00244.
The second drawing is **dog** and is identified as **dog** with a probability of 69.02900338 172913. There are also few probabilities of misclassifying dog as bandage.
The third drawing is **bear** and is identified as **bandage** with a probability of 98.695719 24209595.
The fourth drawing is **yoga** and is identified as **yoga** with a probability of 99.7196555 1376343.
The fifth drawing is **airplane** and is identified as **bandage** with a probability of 75.858 59298706055. There also few probabilities of misclassifying bandage as flipflops.
The drawing **bandage** is identified as **bandage** with a probability of 99.994623661041 26.
The drawing **flipflops** is identified as **flipflops** with a probability of 99.998497962951 66.
The drawing **bear** is identified as **bear** with a probability of 95.23402452468872.

General Information

The training of our model lasted 7 hours, 14 minutes and 4 seconds. Each epoch lasted approximately 20 minutes. The length of the dataset is responsible for the duration of the training procedure. This could be considered as a problem because for every little change in our code we needed too much time to check our results.

The code was run in laptop DELL with processor Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, 1992Mhz 4 Cores, 8 Logical Processors.

Members/Roles

Member	Background	Role
Kosmidi Antonia BAFT1707	Economics	Responsible for the business approach of the problem
Koutsoukou Dimitra Evangelia BAFT1708	Mathematics	Responsible for interpreting the results
Soukias Nikolaos BAFT1714	Electrical & Computer Engineering	Responsible for the code
Tsopelakou Aglaia BAFT1716	Financial and Management Engineering	Responsible for the methodology
Charalampidou Eleni BAFT1718	Electrical & Computer Engineering	Responsible for the bibliography and the presentation

Bibliography

1. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/07/00-2016-94-TIP-oracle.pdf
2. https://www.researchgate.net/publication/271710355_Deep_Neural_Networks_for_Sketch_Recognition
3. https://iui.ku.edu.tr/sezgin_publications/2017/SezginYesilbek-CG-2017.pdf
4. http://faculty.cse.tamu.edu/hammond/courses/SR/papers/Yu/Yu2003Domain.pdf
5. http://rationale.csail.mit.edu/publications/Alvarado2004Sketch.pdf
6. https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8
7. https://medium.com/technologymadeeasy/for-dummies-the-introduction-to-neural-networks-we-all-need-c50f6012d5eb
8. https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
9. https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-678c51b4b463
10. https://towardsdatascience.com/quick-draw-the-worlds-largest-doodle-dataset-823c22ffce6b
11. https://www.researchgate.net/publication/271710355_Deep_Neural_Networks_for_Sketch_Recognition
12. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/07/00-2016-94-TIP-oracle.pdf
13. http://www.hrotoday.com/news/engaged-workforce/performance-management-rewards/business-case-recognition/
14. http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf?hc_location=ufi
15. https://towardsdatascience.com/deep-learning-for-image-classification-why-its-challenging-where-we-ve-been-and-what-s-next-93b56948fcef
16. http://faculty.cse.tamu.edu/hammond/courses/SR/papers/Yu/Yu2003Domain.pdf
17. https://github.com/tensorflow/magenta/tree/master/magenta/models/sketch_rnn

Time Plan

- Studying of relative articles for neural networks, planning, splitting the tasks, assigning roles (1 week)
- Finding the dataset and the main area to focus on in our assignment (1 week)
- Methodology & Programming (2 weeks)
- Report & Presentation (1 week)

Contact Person

For any additional information you can contact Dimitra Evangelia Koutsoukou.

E-mail address: dimkoutsoukou4@gmail.com

Phone Number: 6877683602

Comments

Magenta is a research project exploring the role of machine learning in the process of creating art and music. Primarily this involves developing new deep learning and reinforcement learning algorithms for generating songs, images, drawings, and other materials. But it's also an exploration in building smart tools and interfaces that allow artists and musicians to extend (not replace!) their processes using these models. Magenta was started by some researchers and engineers from the Google Brain team, but many others have contributed significantly to the project. They use Tensorflow and release their models and tools in open source on this GitHub. They are training an RNN model for sketch recognition.