

# Definition

## Project Overview

While many machine learning techniques do exceptionally well in classifying photographs, photo-realistic representations are only a small subset of visual information that human encounter in real world. As a matter of fact, it is plausible to think that throughout human history, representing objects in a photo-realistic manner is a rather recent phenomenon, as human beings have utilized visual representation of objects using drawing long before the invention of photographs. Nevertheless, identifying hand-drawn pictures embodies a different set of challenges, for it often contains less information than a feature rich photograph with colors. More specifically, classifying sketches are difficult for the following reasons[2]:

1. Sketches are highly iconic and abstract.
2. Due to the free-hand nature, the same object can be drawn with hugely varied levels of detail/abstraction.
3. Sketches lack visual cues, such as color and texture information.

It is no wonder, then, many recent studies in image classification focuses on identifying sketches made by human. For instance, Eitz and others[1] tried to overcome the difficulties above by translating the sketches into a bag-of-features representation in terms of orientational descriptors, which in turn are represented as a frequency histogram. They then constructed models using the nearest-neighbor algorithm and support vector machine.

## Problem Statement

A Deep Convolutional Neural Network is a powerful neural network algorithm that has achieved much success in image recognition. It functions by creating filters that are adapted to the features of the images the network receives as training inputs. They are then combined to draw probabilistic inferences about pictures they receive. The result is a powerful classifier that works well even in high dimensions.

Despite its success, it cannot be taken for granted that the same performance will hold for recognizing sketches. While sketches are often easily discernible to human eyes and reproducible, they are very different than the pictures that are used to train popular CNN models. They lack the richness of the information found in photo-realistic representation, and they lack the predictable structure of words and numbers. Thus, one problem to be solved is to determine the appropriate input shape for model. While  $224 \times 224 \times 3$  matrices seem to do well for feature-rich images, for black and white sketches this might be overkill. The next problem

to be solve is to determine the number of hidden layers and their relevant parameters. The output layer will ultimately be a 250 node fully connected layer with soft-max activation function, producing a vector of the relative probabilities of the 250 object categories.

In this project, I endeavor to develop a deep CNN that will have a reasonable degree of accuracy of sketch image classification. The basic workflow of the project is to improve the baseline model by experimenting with additional layers and its parameters. As with the benchmark model, I will split the data into the training, validation, and testing sets. However, I intend on using additional convolutional layers for the actual model. I also would add a drop-out layer to prevent over-fitting.

## Evaluation Metrics

For this project, I will use cross entropy as a loss function during training. This seems appropriate since cross-entropy is low when the predicted probability is close to the actual label. Mathematically, cross entropy is defined as:

$$L(p, q) = - \sum_i p(x_i) \log[q(x_i)]$$

where  $p$  is the true distribution of the data and  $q$  is the estimation of the model. The use of cross-entropy loss is justified by its popularity and successes in image classification.

Since the goal is to have a classifier that correctly identifies the object category of a sketch, as an evaluation metric, I will use accuracy of the classification, which is defined as:

$$\frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{True Negative} + \text{False Positive} + \text{False Negative}}$$

Accuracy is chosen over other metrics (such as precision and recall) because in our context there is not distinction between false positive and negative. Also this allows us to compare our models against human accuracy.

# Analysis

## Data Exploration

The dataset for this study originated from [1]. The data set consists of 500MB of png files and there is a total of 20,000 sketches available, divided into 250 categories by their objects. The images are deliberately lacking in features - for instance, objects are not to be situated in an environment, so there is no context that would help identifying the object. Further, they are also lack black areas that help defining the objects.

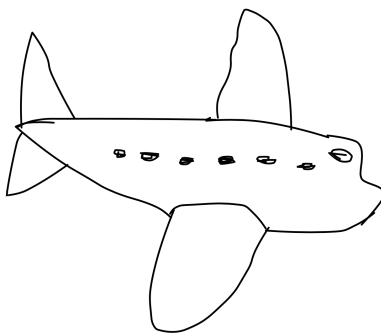


Figure 1: Example Sketch of a Plane

According to the original paper, the taxonomy was the result of a crowd-sourcing effort. They then enlisted users from Amazon Mechanical Turks to draw sketches based on those categories.

The following plot(fig.2 ) is the distribution of images in all of the categories - each of them has 80 images, thus the distribution is perfectly uniform.

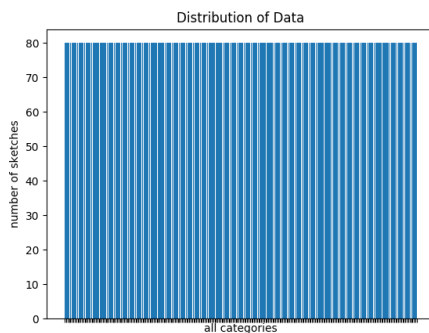


Figure 2: Data Distribution

## Exploratory Visualization

## Algorithms and Techniques

The technique used involved in this study involves constructing a neural network that takes in the images as matrices (that will be smaller than  $(244 \times 244 \times 1)$  for input, through a multiple Convolutional and Pooling layers, and ultimately generates a prediction through a 250 nodes output layers - there are 250 because there are 250 object categories. Data augmentation (discussed below) were used to increase the performance of the network.

### Convolutional Neural Network

A CNN is not much different than typical Neural Networks. Both consist of layers of neurons with trainable weights and biases. A CNN is made of multiple neural network layers, primarily of two sorts: convolutional layers and pooling. The most common architecture is to alternate the two layers, with the final layer being a fully connected one.

One advantage of CNN is its ability to recognize local features. This is accomplished by having the layers, instead of being fully connected, organized in terms of filters which are essentially sub-matrices.

### Convolution Layer

Regular Neural Network takes as its input a vector of values and pass it through a set of fully connected layers called *hidden layers* and finally produce an output with the output layer, which is made of the same number of neurons as inputs. However, regular Neural Networks do poorly with images. Since each parameter has a weight, even a very small image would require thousands of neurons. For instance, a color picture that is  $50 \times 50$  would require 7500 weight. In our case, even though the sketches are black and white, they are  $256 \times 256$  which would require 66536 parameters.

CNN takes a different approach that exploits the 3-dimensional nature of the images (here the 3rd dimension refers to color). Thus, unlike regular Neural Network, the input for CNN is multidimensional and images are not required to be flattened. Furthermore, convolution layers are not fully connected - each neuron is only connected to a small area of the input. These areas are referred to as *filters* and they typically are smaller than the input. These filters, intuitively, act as 'sliding windows' that scan through the local area on pixel of the time. The filter may slide only within the region which is limited by the size of the filter in contrast to the size of the

area, or *padding* can be used to allow the filter moves free within the region, by treating out of bound area as 0s. Roughly speaking, these operations involve the computation the dot-product between the neuron's weight and the input to produce what is called a *featuremap*.

Once the featuremap is generated, to ensure that each output is at least 0, a so-called ReLU(rectified linear unit) function is used as *an activation function*. Mathematically, ReLU is defined as:  $f(x) = \max(0, x)$  or  $\frac{1}{1+e^x}$ .

## Pooling

Often, a convolution layer is followed by a *pooling* layer. The purpose of a pooling layer is to condense the output of the previous layer, resulting an decrease in volume in the dimensions. This is usually done by a summary statistic- two types of which are used in this paper:

1. Max pooling: this layers summarizes the previous convolution layer by its maximal value within a tunable size (for instance  $3 \times 3$ .)
2. Average pooling: this is similar to max pooling, except the average is being used to summarize a sub-region.

The key goal of pooling is ensure that the neural network is insensitive to irrelevant variance of the input (e.g., the object moving slightly to the right), as the pooling layer will produce the same output. This feature is usually called *invariance*.

## Dropout

The dropout layer randomly sets the output of a perceptron to be 0 with a pre-defined probability. The purpose is to add a degree of randomness to each neuron's input. The primary goal of this layer is to alleviate over-fitting.

## Benchmark

### Benchmark 1: Basic CNN

In order to test the performance of the final model, the following basic CNN has been constructed as a benchmark model.

Layer	Output Shape
4x4 Conv2D, 32	256x256x32
2x2 MaxPooling2d	128x128x32
Global Average Pooling	32
Dropout(p=0.2)	32
Fully Connected	250

## Benchmark 2: Human subjects

Another interesting benchmark is the result from the study in [1], in which human subjects are asked to classify four sketches from the same random categories. The accuracy of human perception is 73.1%. This is rather telling because it shows the difficulty of sketch recognition - even human beings cannot obtain a high accuracy.

Further, it will be of interest to compare the categories for which human being have the most and the worst accuracy to our model. Seagulls, for instance, could only be recognized 2.5% of the time while t-shirts were always recognized.













t-shirt  100%	snake  99%	comb  99%	flower  99%	eyeglasses  98%	elephant  98%
seagull  2.5%	panda  11%	armchair  13%	tire  21%	ashtray  24%	snowboard  25%

Figure 3: Human Accuracy: The Best and Worst 6

## Methodology

### Data Preprocessing

- All sketches were rescaled into 255x255.
- The rescaled sketches were transformed into 255x255x1 matrices. Even though they were processed as grayscale pictures. The third dimension only takes 0 and 1 as value, since all sketches were made strictly with solid lines, with no gradation.

The data were augmented randomly in the following way:

- Object may shift horizontally with a range of 0.2.
- Object may shift vertically with a range of 0.2.
- Object may flip horizontally.
- Object may zoom with the range of 0.2

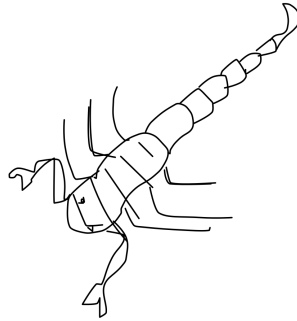


Figure 4: Data Augmentation Example: Original



Figure 5: Data Augmentation Example: Augmented

Lastly, out of the 20,000 sketches, 10% was randomly designated as the test set, and another 10% as the validation set.

## Implementation

- data augmentation was done using Keras' image preprocessing module:

```
datagen = ImageDataGenerator(  
    width_shift_range=0.2,  
    height_shift_range=0.2,
```

```
horizontal_flip = True,
zoom_range = 0.2)
```

- The network was constructed using Keras with a TensorFlow backend.
- Each layer's initial weight was initialized using Keras' default initializer(Glorot normal).
- Padding was enabled, with a kernel size of 4. A max pooling layer is used after every convolution layer
- the ReLU activation function was used for convolution layers.

```
model = Sequential()
model.add(Conv2D(filters=16,kernel_size=4, padding='same',activation='relu',input_shape=train_tensors.shape[1:]))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32,kernel_size=4, padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64,kernel_size=4, padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=128,kernel_size=4, padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=256,kernel_size=4, padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(GlobalAveragePooling2D())
```

- A Dropout layer with  $p = 0.2$  was used.

```
model.add(Dropout(0.2))
```

- the output layer is a fully connected layer with 250 neurons (one for each category).

```
model.add(Dense(250,activation='softmax'))
```

- All models were trained with RMSprop with 0.001 learning rate and 0 decay (default value) with categorical\_crossentropy as the loss function:

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

- The model was fitted with the augmented data generator.
- Validation set was *not* augmented, since it did not appear to improve the performance of the model.
- It was trained on a desktop computer with the following spec:



- CPU: i7-4770K
- Ram: 12GB
- GPU: Geforce 980 with 4 GB Ram

bs = 32

```
model.fit_generator(datagen.flow(train_tensors,
                                y_train, batch_size=bs),
                    steps_per_epoch=train_tensors.shape[0]/bs,
                    epochs=epochs,
                    verbose=1,
                    callbacks=[checkpointer],
                    validation_data=(valid_tensors, y_val)
)
```

## Model 1

Layer	Output Shape
4x4 Conv2D 16	256x256x16
2x2 MaxPooling2d	128x128x16
4x4 Conv2D 32	128x128x32
2x2 MaxPooling2d	64x64x32
4x4 Conv2D 64	64x64x64
2x2 MaxPooling2d	32x32x64
4x4 Conv2D 128	32x32x128
2x2 MaxPooling2d	16x16x128
4x4 Conv2D 256	16x16x256
2x2 MaxPooling2d	8x8x256
Global Average Pooling	256
Fully Connected	250

## Refinement

Initially, the model contained four convolution layers, with roughly 40% accuracy. Different  $p$  value for the Dropout layer was experimented upon, but no substantive change was noticed. Another decision that had to be made is the size of the sketches. It has been reported that a transforming the sketches into 30x30 matrices yielded satisfactory performance[3], but unfortunately this has not been my experience. In our case, the optimal size was found to be somewhere about 250x250.

## Result

### Model Evaluation and Validation

Model	Accuracy
Human Perception	0.73
Model with Data Aug	0.66
Model without Data Aug	0.55
Chance	0.04
Benchmark Model	0.03

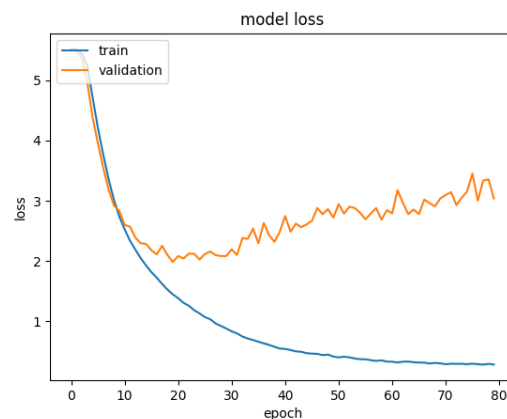


Figure 6: Final Model without Data Aug: Loss vs Epoch

It is clear that without data augmentation, the model suffers from overfitting after 40 epochs (fig 6 and 7). On the other hand, such overfitting is not present when the data is augmented (fig 8 and 9).

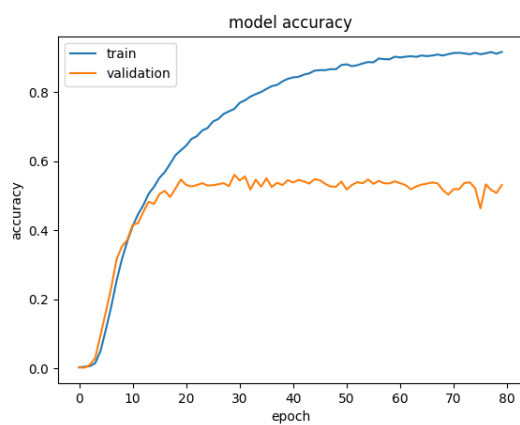


Figure 7: Final Model without Data Aug: Accuracy vs Epoch

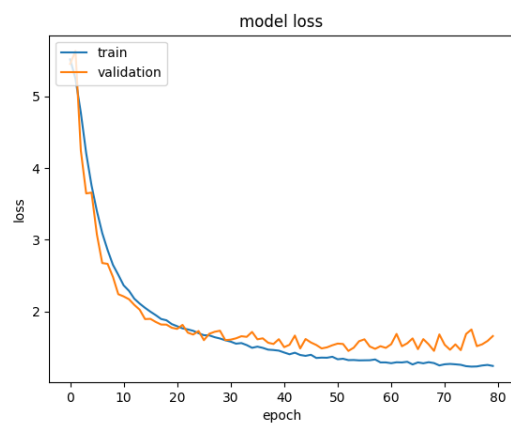


Figure 8: Final Model with Data Aug: Loss vs Epoch

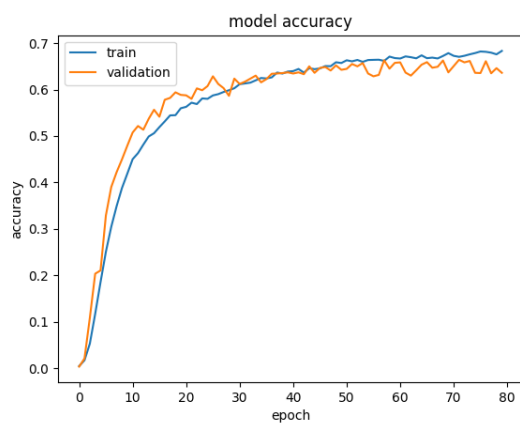


Figure 9: Final Model with Data Aug: Accuracy vs Epoch

## Justification

The model was able to achieve 66% accuracy, which is a huge improvement over the benchmark model, which did worse than chance. Even though it could not perform as well as a human being, the difference is less than 10 %, thus it is reasonable to conclude that the problem is solved. The use of data augmentation was also justified by the fact that it led to a 10% increase in accuracy.

## Conclusion

To get a intuitive sense of the model's performance, I decided to calculate the model's predictive accuracy for each category using the original set. The top 10 is as follows:

### Best 10

Category	Accuracy
Eye	0.99
Sailbot	0.96
Ladder	0.96
Envelope	0.96
T-Shirt	0.96
Candle	0.96
Sponge Bob	0.95
Rainbow	0.95
Smoking Pipe	0.95
Zebra	0.94

The worst 10 is as follow - it is interesting to note that the model has a hard time distinguish different sketches of birds. This may be due the ambiguity of the labels.

### Worst 10

Category	Accuracy
Scorpion	0.37

Category	Accuracy
Armchair	0.39
Bush	0.39
Pigeon	0.40
Standing Bird	0.40
Race Car	0.42
Flying Bird	0.43
Tiger	0.45
Bulldozer	0.46
Loudspeaker	0.49

### Comparing to Human Performance

We can also compare the model’s performance to that of human beings. It is interesting to see that both humans and the CNN struggles on identifying seagulls. Again, it is likely that the label itself is too ambiguous.

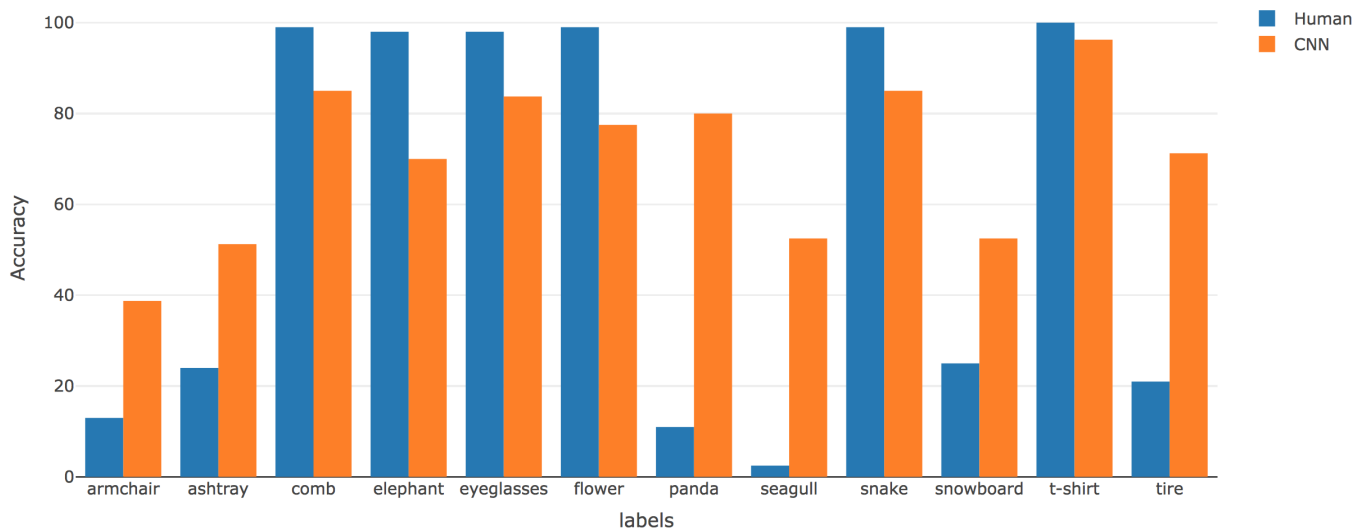


Figure 10: Comparison between Human’s Best/Worst 6 Categories

### End-to-end discussion

My first step toward solving the problem was to prepare the data for the neural network and this turned out to be the most challenging part. First, my computer would frequently freeze when trying to transform the

images into tensors. Initially, two unsatisfactory solutions were attempted: the first was to greatly decrease the size of the training set to 50% of all data and the second was to decrease the target size of the tensors. These solutions were not satisfactory because the sacrifice of performance was too much. Ultimately, the problem was solved by adding 4GB ram.

Once the data was prepared, the process, while time-consuming, was relatively straight-forward. It was time consuming because training the neural network, even with a GPU, takes a bit of time and experimenting with just a few architectures would take a whole day. Once I settled on a model with the best performance, I decided to experiment with data augmentation to see if there was any performance gain.

## Improvement

This implementation of the project was somewhat limited by the hardware available. Especially, the lack of memory was a huge issue: there were many architectures that simply were not implementable, because my GPU only has 4GB of memory.

Another issue is ambiguous labels. It seems that certain labels are simply confusing, e.g., bird vs standing bird vs flying bird vs seagull. This could potentially be improved by using bootstrapping, which has shown to do well against noisy labels. [7]

I am also interested in using transfer learning to improve the model. Preliminary experimentations with transfer learning have not yielded good results - most likely due to the fact that those models were trained on photo-realistic pictures, and not sketches. However, one recent paper[4] contains a very interesting approach to classify sketches by incorporating photo-realistic pictures into training.

## Bibliography

- [1] Eitz, Mathias & Hays, James & Alexa, Marc. (2012). How do humans sketch objects?. ACM Transactions on Graphics. 31. 1-10. 10.1145/2185520.2335395.
- [2] Hand drawn sketch classification using convolutional neural networks,
- [3] [www.iioab.org/articles/IIOABJ\\_7.S5\\_337-341.pdf](http://www.iioab.org/articles/IIOABJ_7.S5_337-341.pdf)
- [4] G. T. Sadouk, Lamyaa and E. H. Essoufi. A novel approach of deep convolutional neural networks for sketch recognition. international Conference on Hybrid Intelligent Systems, 2016
- [5] R. G. Schneider and T. Tuytelaars. Sketch classification and classification-driven analysis using fisher vectors. ACM Transactions on Graphics (TOG), 33(6):174, 2014.
- [6] Zhang, H, Liu S, Zhang C, Ren W, Wang, R, Cao X. [2016], SketchNet: Sketch Classification With Web Images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1105
- [7] Scott Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, Andrew Rabinovichp.[2015], “Training Deep Neural Networks on Noisy Labels with Bootstrapping”, arXiv:1412.6596 [cs.CV]