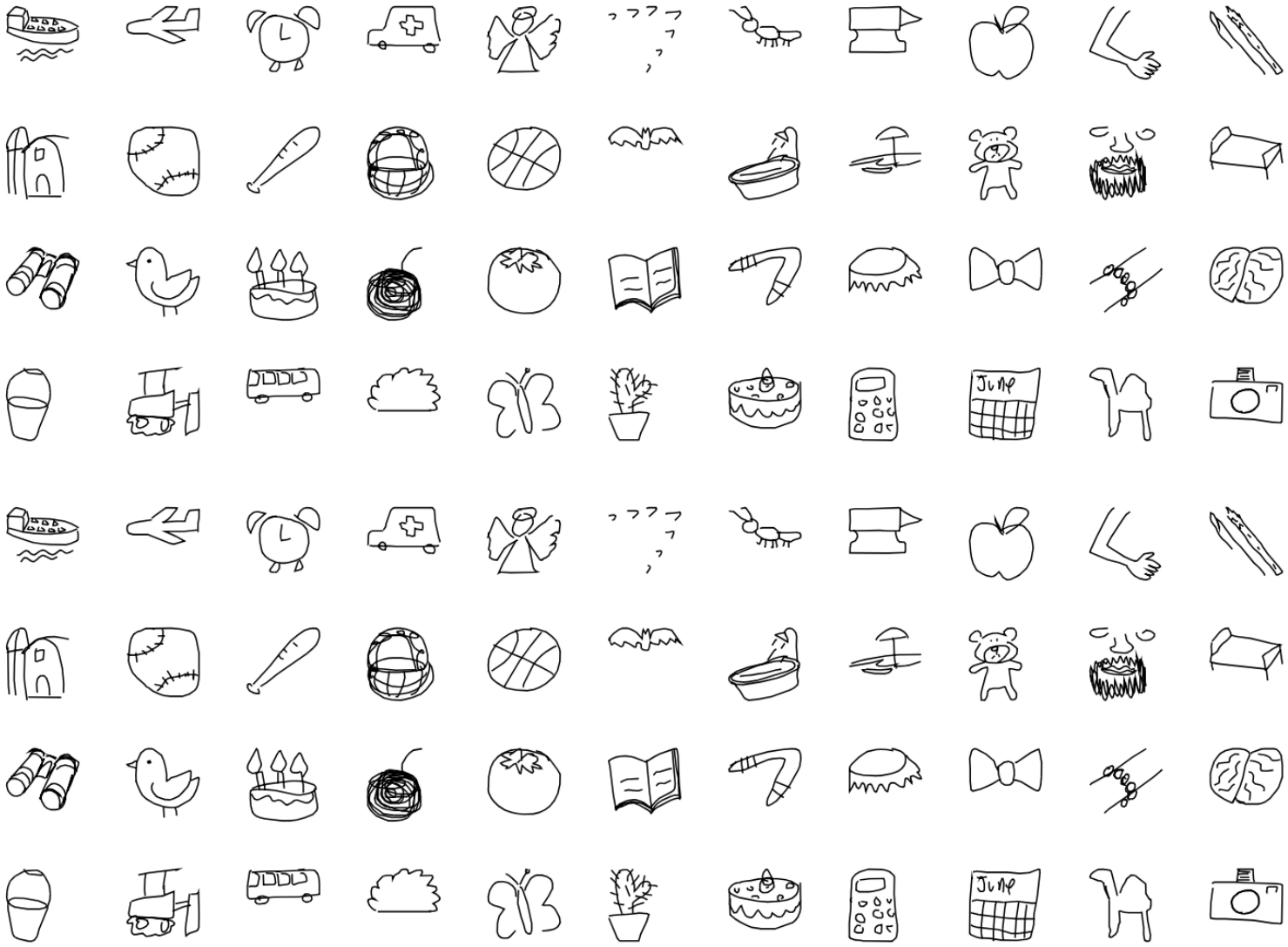

# QUICK, DRAW! DOODLE RECOGNITION

Project Report

by Aleksandra Deis
lexie@live.ru
https://lexie88rus.github.io

CONTENTS

***DEFINITION*** ..... ***2***

**Project Overview** ..... **2**

**Problem Statement** ..... **2**

**Metrics** ..... **2**

***ANALYSIS*** ..... ***2***

**Input Data** ..... **2**

**Data Exploration** ..... **2**

***METHODOLOGY*** ..... ***5***

**Data Preprocessing** ..... **5**

**Implementation** ..... **5**

Building the Model ..... 5

Building the Web-application ..... 6

**Refinement** ..... **8**

***MODEL EVALUATION*** ..... ***9***

**Justification** ..... **10**

***CONCLUSION*** ..... ***10***

**Summary** ..... **10**

**Improvement** ..... **11**

# DEFINITION

## Project Overview

The Quick Draw Dataset is a collection of 50 million drawings across 345 categories, contributed by players of the game Quick, Draw!. The player starts with an object to draw (for example it may say "Draw a chair in under 20 seconds"). Then the player has twenty seconds to draw that object. Based on what they draw, the AI guesses what they are drawing.
Research in recognition of images drawn by humans can improve pattern recognition solutions more broadly. Improving pattern recognition has an impact on handwriting recognition and its robust applications in areas including OCR (Optical Character Recognition), ASR (Automatic Speech Recognition) & NLP (Natural Language Processing).
In this project I analyzed the drawings and tried to build a deep learning application to classify those drawings.

## Problem Statement

Recognition of a drawing is a classification problem. I have to build a solution, which classifies input images. I split the whole problem of recognition of drawings into the following tasks:

- Input data analysis and preprocessing;
- Building a model to classify drawings;
- Evaluation of the model concerning chosen metrics;
- Building a web-application to demonstrate the results.

## Metrics

I chose accuracy as a metric to evaluate the results. Because of the rules of the game, we mostly care about how many times did the AI recognize the drawing correctly, and this is just the accuracy of the model.

# ANALYSIS

## Input Data

The whole dataset consists of:

- Raw moderated dataset, which contains the raw data in .ndjson format;
- Preprocessed dataset, which contains simplified images. Simplified dataset offers three options with different file formats:
    - Simplified .ndjson images;
    - The simplified drawings and metadata are also available in a custom binary .bin format for efficient compression and loading.
    - Simplified drawings in 28x28 grayscale bitmap in numpy .npy format. The files can be loaded with np.load(). These images were generated from the simplified data but are aligned to the center of the drawing's bounding box rather than the top-left corner.

I chose the simplified dataset with images in .npy format. This format is the easiest to use, preprocess, and to produce by a web application.

## Data Exploration

These are the examples of images from the dataset loaded by the Matplotlib plotting library:

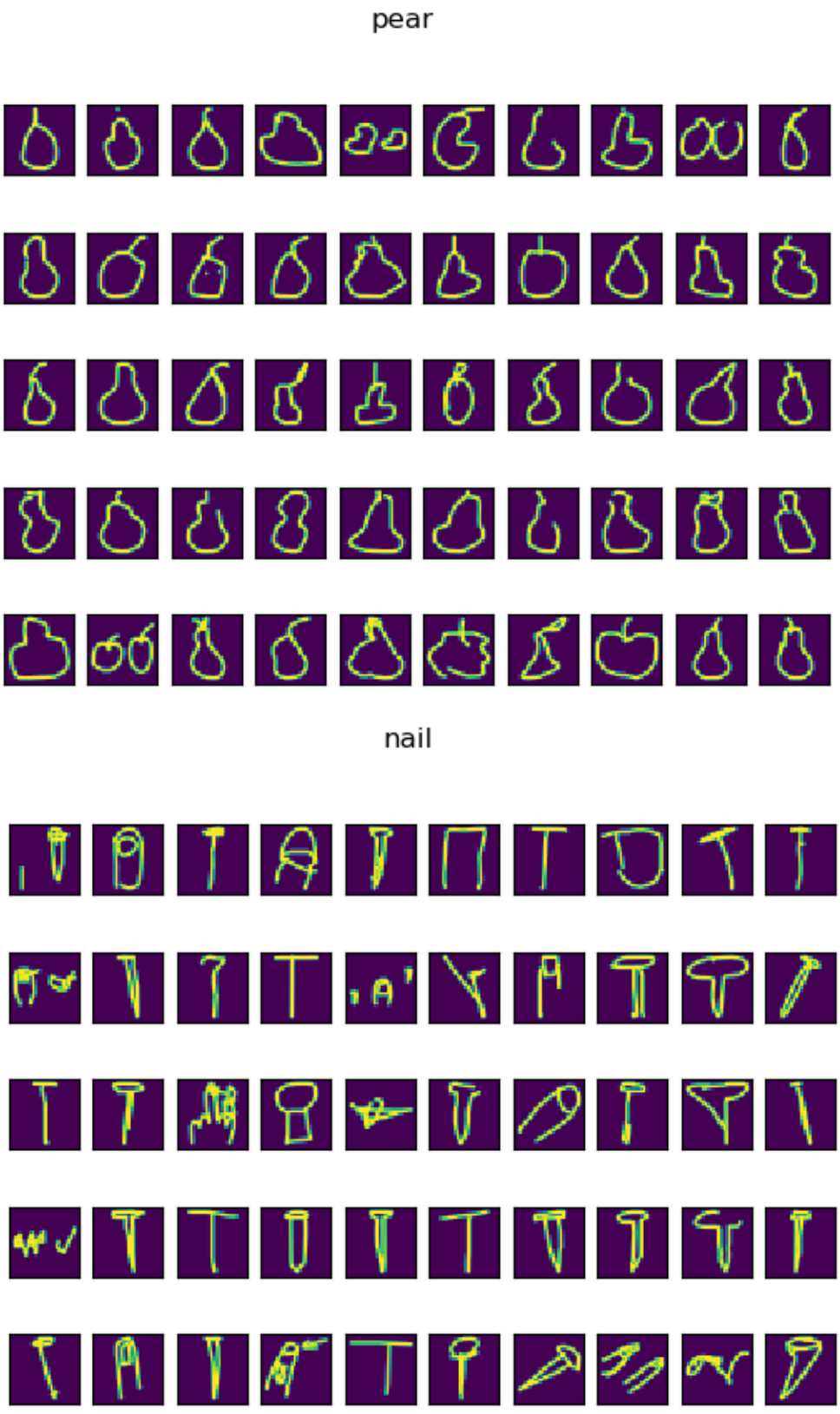


*Figure 1 Examples of images from the dataset*

The figure above demonstrates drawings from the dataset. We can see that dataset contains simple classes like “pear” as well as more complicated ones like “nail,” which can have several different meanings (small metal spike or a body part).

It is also interesting to look at the generalized heatmaps for different classes of drawings. I created heatmaps for each class by calculating the average brightness for a pixel for all drawings belonging to one class.
The examples of heatmaps for different classes:

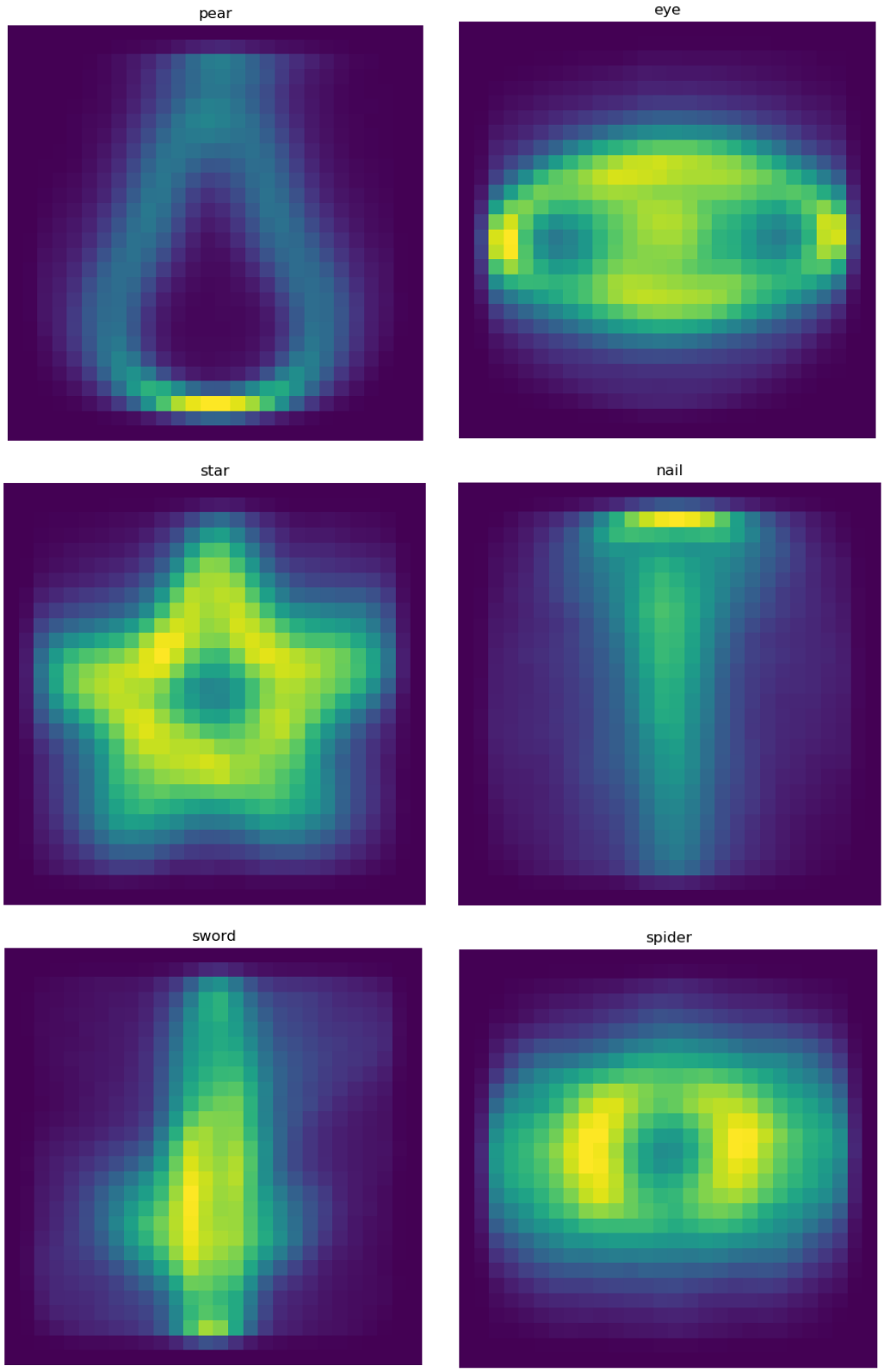


*Figure 2 Examples of Heatmaps for Image Classes*

The heatmaps demonstrated on the figure above, in fact, represent the generalized idea of each class. Looking at the heatmaps, we can make a lot of interesting observations:

- People who play the game give the star a five-pointed representation.
- People who play the game represent nail as a metal spike (not as a body part).
- Game players generally draw the sword pointed upwards.

## METHODOLOGY

### Data Preprocessing

I chose already preprocessed dataset with images which are already cropped and resized to 28 to 28 pixels. However, I decided also to generate a set of slightly rotated and flipped images. I added these images to the training dataset to reduce the variance of the resulting model.

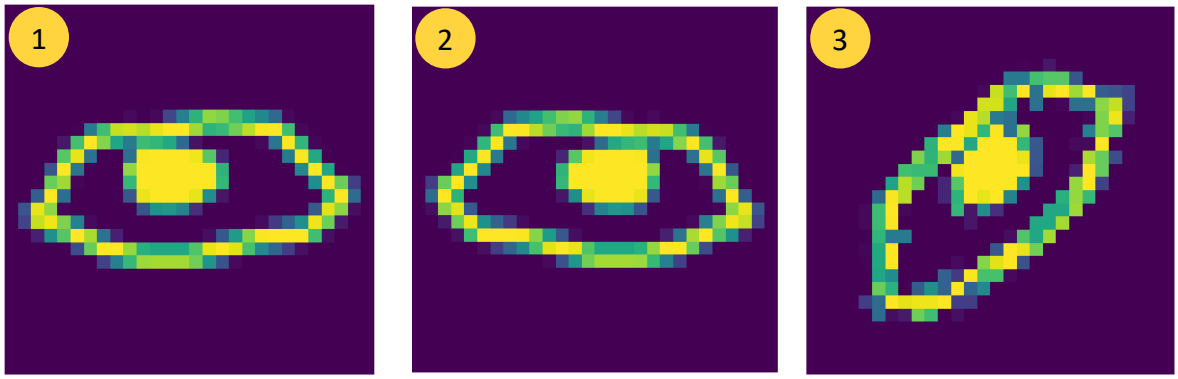


*Figure 3 Image Preprocessing: (1) - original image, (2) - flipped image, (3) - rotated image.*

The figure above gives examples of generated images added to the dataset.

### Implementation

Since there is a lot of data, and I can even generate additional data by flipping and rotating the images, I decided to use deep learning approaches to classify drawings.
The implementation consists of two parts:

1. Building and refining the deep learning model to classify drawings;
2. Building a web application to demonstrate the model.

This GitHub repository contains all code for the implementation.

### Building the Model

To simplify the task a little I chose only ten image classes from the initial dataset.
My goal is to build a model, which takes 28 x 28 pixels image as an input and gives probabilities for each of the possible classes as an output. The figure below demonstrates the desired result:

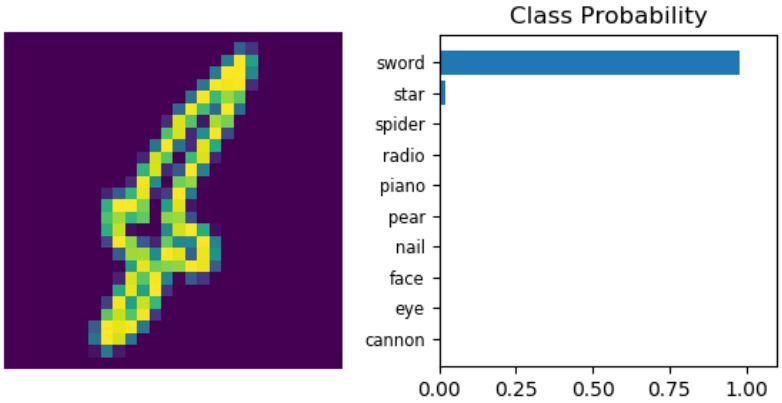


*Figure 4 Prediction example*

I started with a simple fully connected neural network with two hidden layers built with the PyTorch library.
The sizes of the layers are as follows:

- Input layer: 784 (for 28 x 28 images),
- Hidden layer 1: 128,
- Hidden layer 2: 100,
- Output layer: 10 (the number of classes).

For each hidden layer there is:

- ReLU activation function,
- Batch normalization.

The resulting model has hyperparameters as follows:

- Learning rate,
- Dropout for hidden layers,
- Weight decay (L2 regularization),
- Optimizer: Adam or SGD.

The result of building the model part is the buil_model.py script which allows creating, training, and saving the PyTorch deep learning model with the architecture described above. Hyperparameters, as well as the number of epochs for training, may be passed through the command line. The resulting model is loaded and used by the web application described in the next section.

## Building the Web-application

The purpose of the web application is to demonstrate how the model can identify the image drawn by the user of the application. The resulting application should let the user:

1. Draw an image;
2. Submit the drawing and show the response given by the model.

To solve this problem I used Flask, Bootstrap, Plotly, and PIL library for image processing. The web application consists of following parts:

- Front-end:
    - The main page (index.html, see figure below): The main page allows the user to draw an image with HTML canvas and submit the image. The image is encoded into base64 and passed to Flask server.

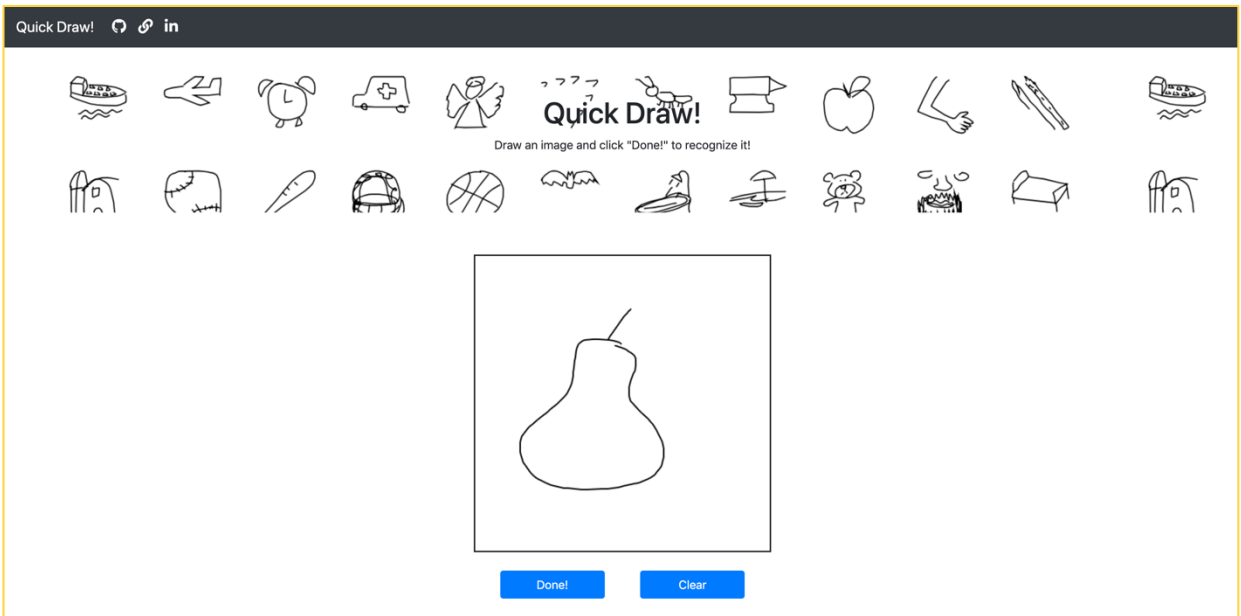


*Figure 5 The application main page*

- The results page (hook.html, see figure below): The results page demonstrates the image class identified by the model along with the Plotly diagram, which shows probabilities for each class.

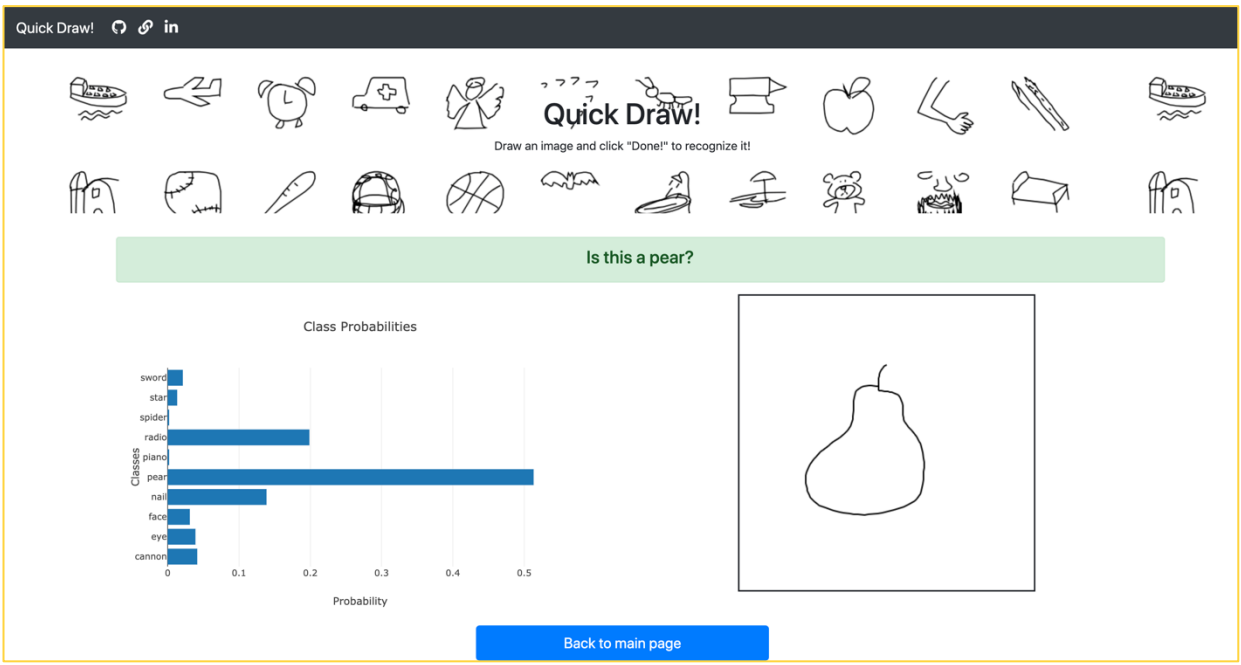


*Figure 6 The application results page*

- Back-end (run.py): The Flask server application, which:
  - Loads the deep learning PyTorch model on start;
  - Receives base64 encoded drawing;
  - Preprocesses the drawing;
  - Passes preprocessed drawing to the model;
  - Creates Plotly graph with probabilities for each class;
  - Renders the results page with the message and the graph.

The application has to preprocess the images, which are drawn by the user:

1. Crop the image to the center;
2. Resize the image to 28 x 28 pixels;
3. Normalize the colors, because the image becomes blurry after the resize.

The following figure shows the steps of the preprocessing.

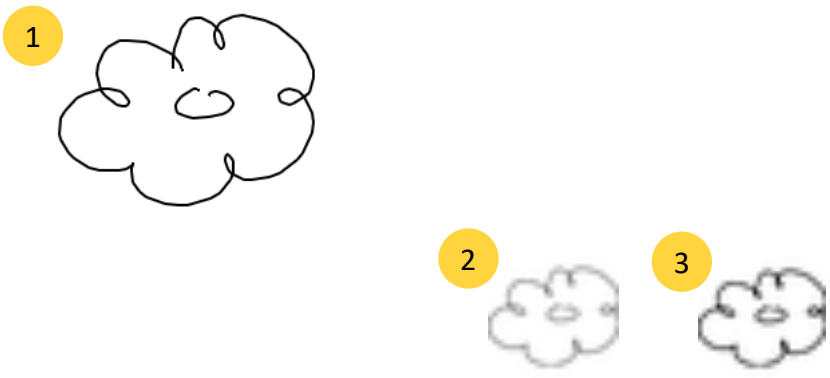


*Figure 7 Preprocessing: (1) – original image, (2) – cropped and resized image, (3) – normalized image*

## Refinement

The following plot demonstrates the accuracy of the basic neural network trained on the original dataset (with no added rotated and flipped images) with following hyperparameters:

- Learning rate is 0.03;
- Dropout is 0.0;
- Weight decay is 0.0;
- Optimizer is SGD;
- No batch normalization applied.

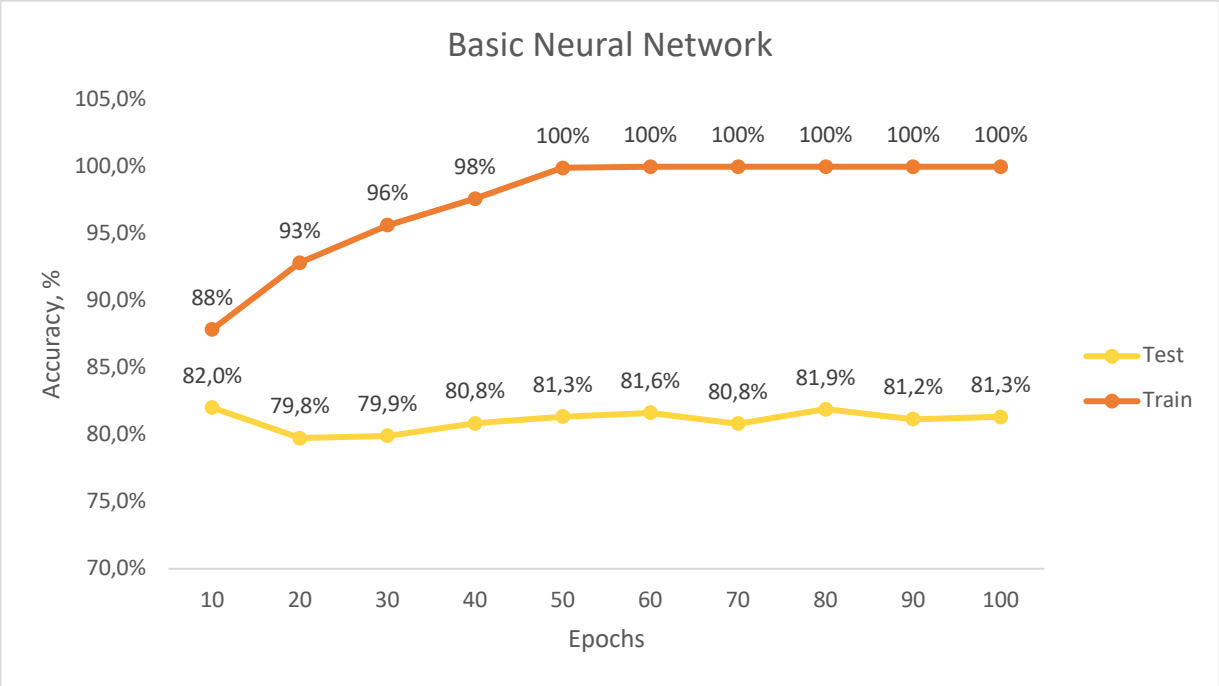


*Figure 8 Accuracy of the basic neural network*

Looking at the evaluation of the basic model above, we can say that we have high variance problem with our model. The model fits well the training set but performs much worse on the test set.

The first thing I tried was to generate more data by adding additional slightly rotated and flipped images to the original dataset. See details in Data Preprocessing section.

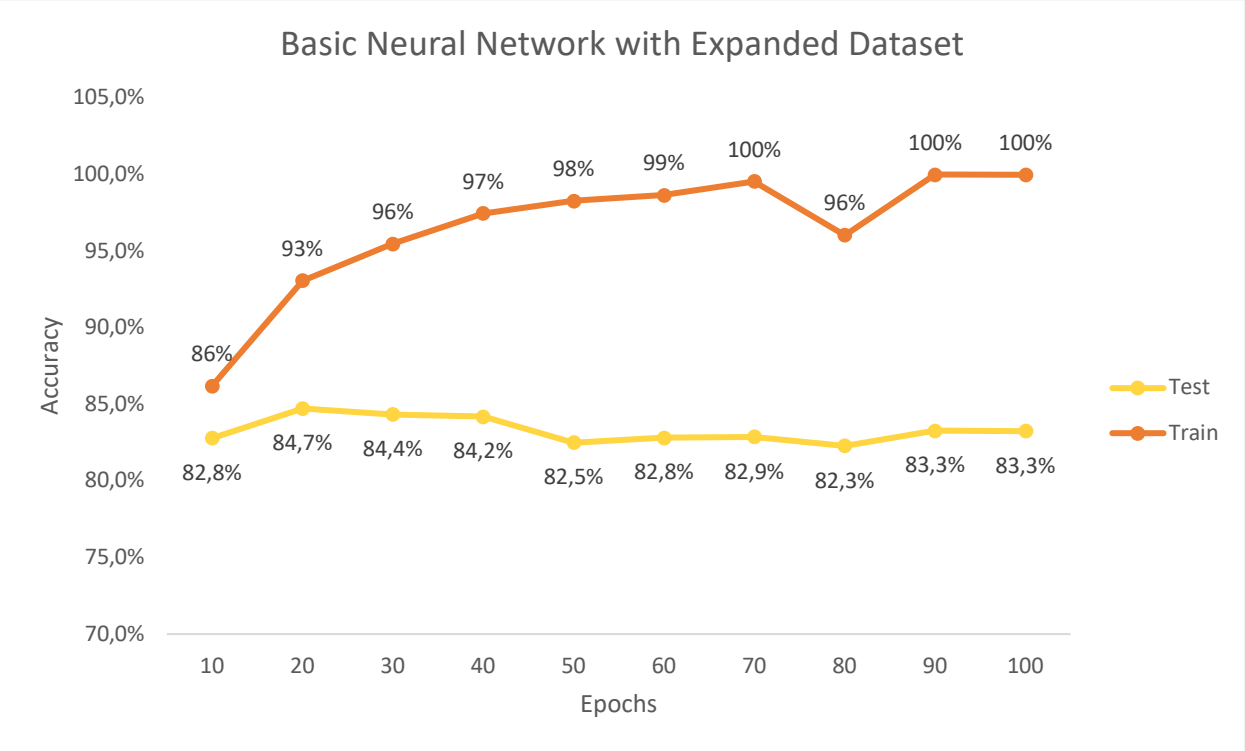


*Figure 9 Accuracy of the basic neural network on the expanded dataset*

The figure above shows the accuracy of the same model on the dataset with rotated and flipped images. We can see that adding generated images to the dataset actually helped to reduce the variance and raise the accuracy score of the model on the test dataset.
But still the variance is very significant, and I have to apply regularization techniques to reduce it.

## MODEL EVALUATION

To reduce the variance and increase the accuracy score on the test dataset I added regularization to the model:

- L2 regularization (weight decay),
- And dropout for each hidden layer.

L2 regularization didn't work out. Using L2 regularization ended with huge bias problems for the model.
Adding dropout to the model actually helped to reduce the variance and increase the test score a little:

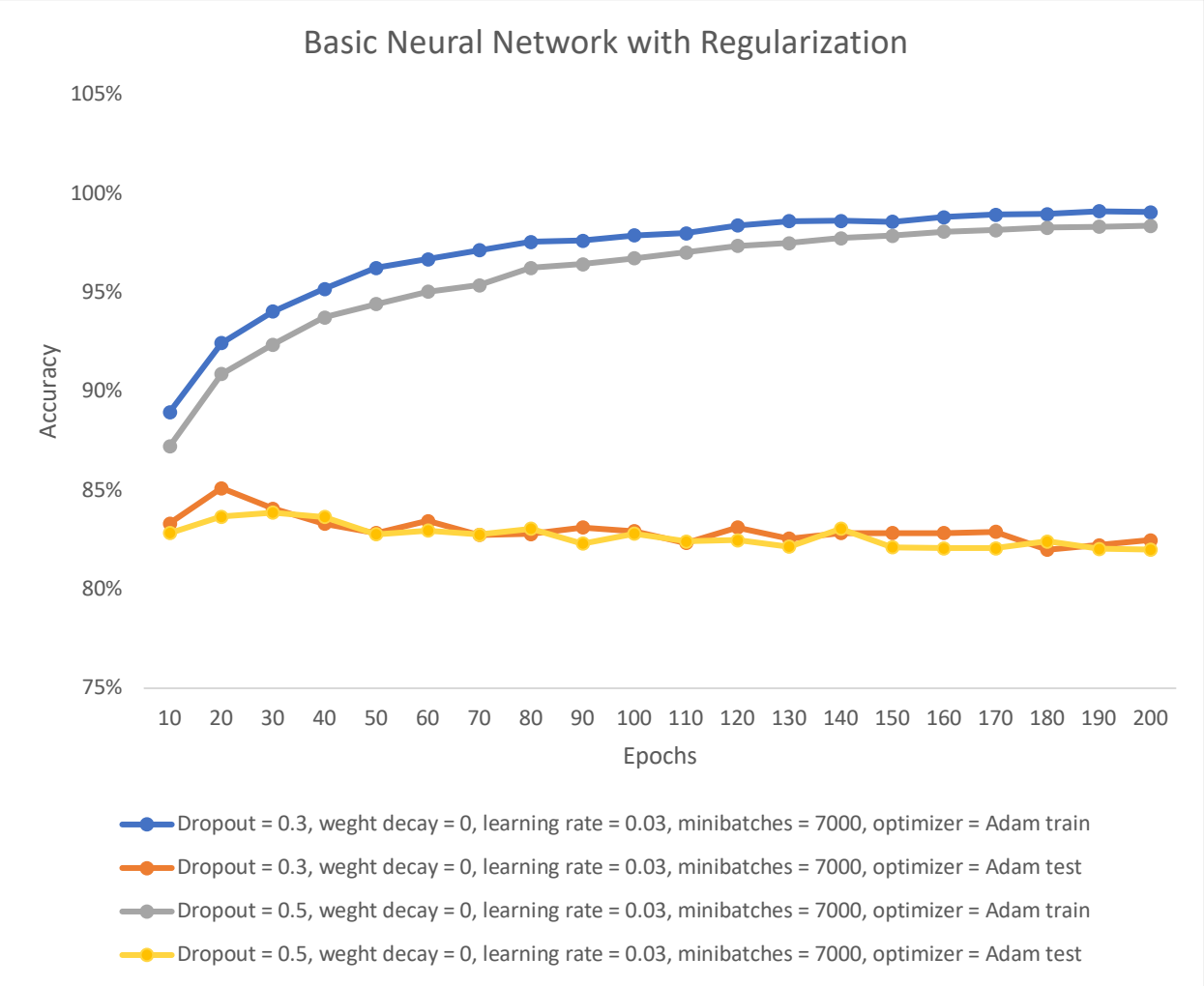


*Figure 10 The accuracy score for model with dropout*

For my final model I used 0.3 dropout. This raised the accuracy up to 85% and reduced the gap between train and test scores.

### Justification

Summary of the final solution:

- Fully connected deep neural network with two hidden layers is used to classify the drawings into ten classes.
- Dropout is applied for each hidden layer to reduce the variance.
- Adam optimizer and batch normalization are used to speed up the computation.
- The test accuracy of the final model is about 85%.
- Web application is created to demonstrate drawings classification by the model.

## CONCLUSION

### Summary

The goal of the project was to build the application to recognize drawing based on Quick, Draw! game dataset. The solution I proposed is as follows:

- The first part of the solution is a deep learning model to recognize images. I used a neural network with several hidden layers to achieve 85% accuracy on the test dataset.
- The second part of the solution was building a web application to demonstrate the ability of the model to recognize the images.

The figure below shows the example of drawing recognition:

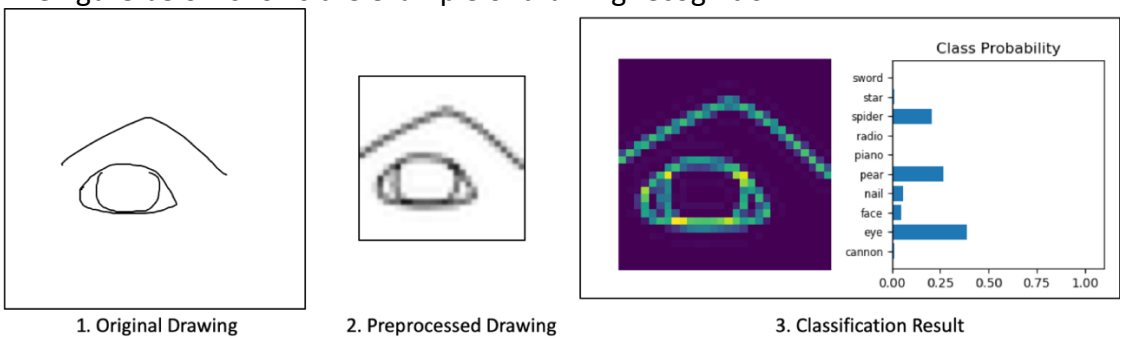


*Figure 11 Example of image recognition*

The most challenging part of the of this project was applying regularization techniques to reduce the variance of the model. I tried several regularization techniques such as using dropout and L2 regularization (weight decay).

## Improvement

The model performs quite well on ten image classes from the simplified dataset, but there is a lot to improve:

- Add more drawing classes;
- Try other architectures: convolutional neural networks;
- Try the full dataset, which contains images with higher resolution and additional information (country, strokes and so on).