**HAND-DRAWN PREDICTION APPLICATION WITH QUICK DRAW DATASET**

*Nguyen Dang Duy Tan*

*Bachelor of Engineer, Robotics and Artificial Intelligence,*

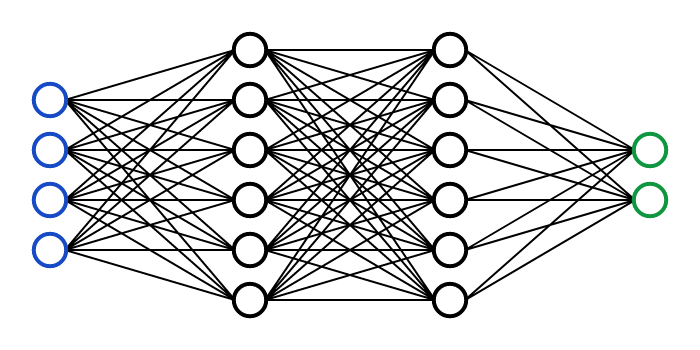
*HCMC University of Technology and Education, Ho Chi Minh City, Viet Nam*

*201340007@student.hcmute.edu.vn*

***Abstract******:*** *I present a hand-drawn prediction application that is based on a neural network model trained on the Quick Draw dataset. The model is able to achieve high levels of accuracy on the Quick Draw dataset, and demonstrates good generalization to other datasets as well. In addition, we discuss the strengths and limitations of the model, as well as potential improvements that could be made. Overall, the results suggest that the neural network model trained on the Quick Draw dataset is a powerful tool for hand-drawn image recognition tasks.*

**1. Introduction**

Neural networks are a type of machine learning algorithm that are inspired by the structure and function of the human brain. They consist of interconnected "neurons" that process and transmit information, allowing them to learn and adapt to new data over time. In recent years, neural networks have become widely used for a variety of tasks, including image recognition, language translation, and even playing games.



***Fig 1.*** *Illustrate the structure of a neural network*

One key aspect of neural networks is their ability to automatically learn features from raw data. This allows them to extract useful patterns and structures from large datasets, without the need for manual feature engineering. This is particularly useful for image recognition tasks, where it can be challenging to manually design features that capture the relevant information in an image.

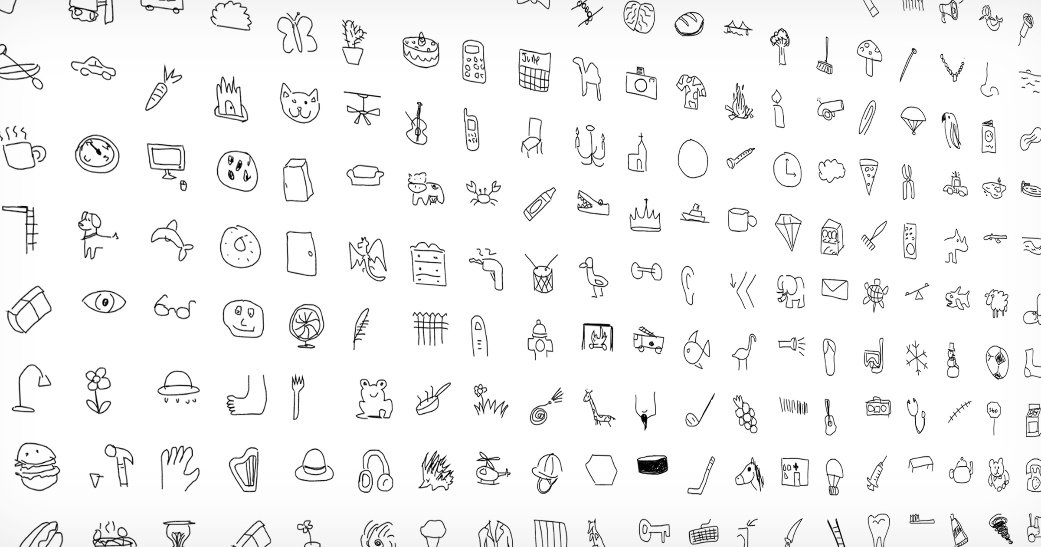
In this project, we developed a hand-drawn image recognition system using deep learning techniques. The system allows users to draw an image on a canvas, and then predict the object represented by the drawing using a pre-trained convolutional neural network (CNN). The system was developed using Python and its popular deep learning libraries such as Keras and TensorFlow. The goal of this project is to create an interactive and user-friendly application that allows users to easily draw an image and quickly get a prediction of the object represented by the drawing. The system can recognize up to 80 different objects. In this report, we will provide an overview of the related work in this field, describe the dataset used for training, and detail the architecture and performance of the CNN model used for prediction.

**2. Related work**

One of the most popular and well-known examples of hand-drawn image recognition is the Quick Draw application developed by Google. This application uses a similar approach to our project, where users can draw an image on a canvas and then receive a prediction of the object represented by the drawing. However, Quick Draw uses a different dataset and a different type of neural network, called a recurrent neural network (RNN), for its predictions. Despite the differences in implementation, the overall goal of Quick Draw and my project is the same: to predict the hand-draw image.

**3. Dataset**

The dataset used for this project is the Quick, Draw! dataset, which was developed by Google. The dataset contains more than 50 million drawings from 345 categories, such as animals, vehicles, and everyday objects. Each drawing is a 28x28 matrix in greyscale. The dataset is publicly available on the Google Cloud Platform and can be downloaded in a variety of formats, including .csv, .json and .npy.



**Fig2.** All categories of Quick Draw dataset

One of the advantages of using this dataset is the large number of drawings available for each category, which allows for a high level of diversity in the training data. This diversity is important for a deep learning model as it allows the model to learn a wide range of variations and variations of the objects within each category. Additionally, the dataset has been pre-processed by Google, which means that it is already in a format that can be easily used for training deep learning models.



**Fig3.** Sample doodles of a sock, elbow, and carrot (left to right) from the training dataset.

However, the Quick, Draw! dataset also has some limitations. One limitation is that the drawings are relatively small, with a resolution of 28x28 pixels. This may not be sufficient for more complex drawings, and it may also be challenging for the model to accurately recognize fine details in the images. Additionally, since the dataset was collected from users drawing images in a web-based game, the images have a wide range of variations in quality and style, some of them may be crude, others may be more detailed, which could be an additional challenge for the model.



**Fig4.** A dog might look like a backpack

Despite these limitations, the Quick, Draw! dataset is a good choice for this project as it allows us to train a deep learning model with a diverse set of data and it has a good size and amount of categories that is enough to our needs.

*Data organization*

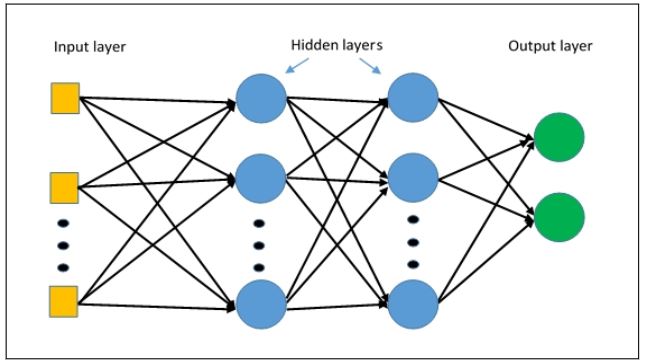
I split data into two part 70% for trainning and 30% for validation

**4. Model**

In this project, I will use the model to categorize 80 different types of hand-drawn images. This training will use 6000 photos from each class.

**4.1. MLP (Multilayer Perceptron)**

A Multi-Layer Perceptron (MLP) is a type of Artificial Neural Network (ANN) that uses supervised learning technique, backpropagation, to train a neural network. An MLP is composed of multiple layers of artificial neurons, also known as perceptrons, that are connected in a directed acyclic graph, with no cycles. It usually includes an input layer, one or more hidden layers, and an output layer. MLPs are good at learning non-linear function and can be used to solve a wide range of problems such as image recognition, natural language processing, and supervised learning problems such as classification and regression. They are relatively simple to understand and easy to implement but their performance depends on the data quality and problem complexity.



**Fig5.** Basic structure of MLP

*Model deployment*

In this model, I will use A ReLU Activation function because of its advantages in filter data and a Cross-Entropy Loss function for loss calculation

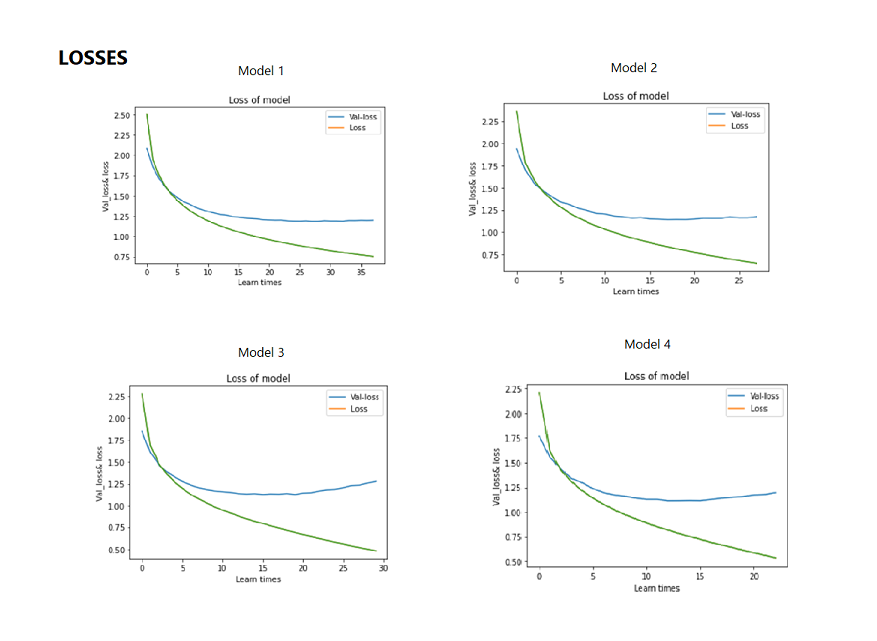
Next, use some MLP architectures to evaluated , from that find out the best performance.

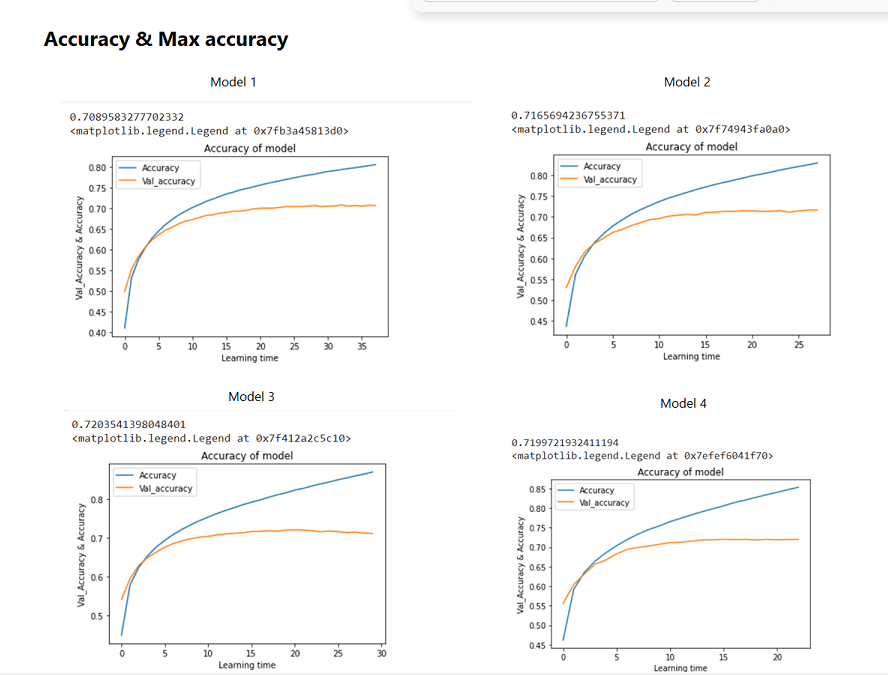
First, I create a basic model with only three layers, and evaluate the dataset. Later on, I increase the complexity of the network trying to obtain a better performance.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **MLP ARCHITECTURE** |  |  |
| **Model** | Model 1 | Model 2 | Model 3 | Model4 |
| **Learning rate** | 0,0001 | 0,0001 | 0,0001 | 0,0001 |
| **Optimizer** | Adam | Adam | Adam | Adam |
| **n layers** | 3 | 4 | 5 | 6 |
| **n epochs** | 100 | 100 | 100 | 100 |
| **Layer 1** | (784, 500) | (784,500) | (784,500) | (784,500) |
| **Layer 2** | (500, 256) | (500,500) | (500,500) | (500,500) |
| **Layer 3** | (256, 80) | (500,256) | (500,500) | (500,500) |
| **Layer 4** |  | (256,80) | (500,256) | (500,500) |
| **Layer 5** |  |  | (256,80) | (500,256) |
| **Layer 6** |  |  |  | (256,80) |

**Table 1**

Collect all result for better comparision:





The highest level of accuracy achieved using these architectural designs was 72.03% in Model 3.

**4.2. Convolutional Neuron Network (CNN)**

In order to improve the accuracy of the model for training on the Quick Draw dataset, I decided to used Convolutional Neural Network because of it advantages.

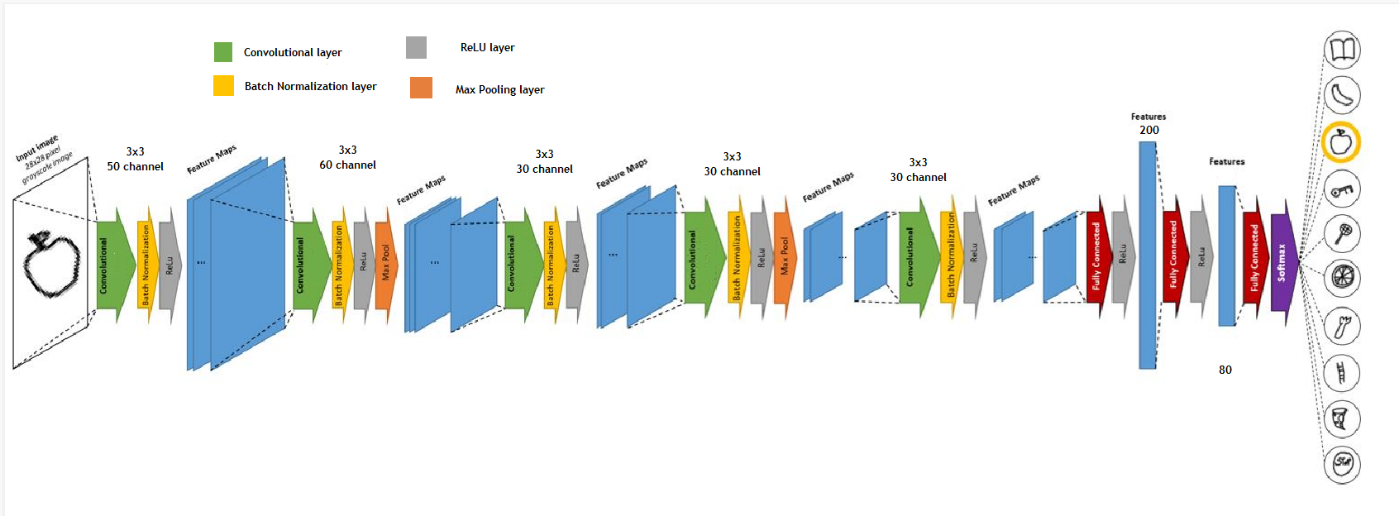
Specifically, CNNs are particularly well-suited for image classification tasks due to their ability to automatically and adaptively learn spatial hierarchies of features through backpropagation. This allows CNNs to effectively identify and extract features such as edges, textures, and shapes from images, which are crucial for recognizing and classifying the different drawings in the Quick Draw dataset. Additionally, CNNs can also take into account the spatial relationships between features, allowing them to effectively classify images even when the objects in the images are translated or rotated.



**Fig6.** Basic structure of CNN

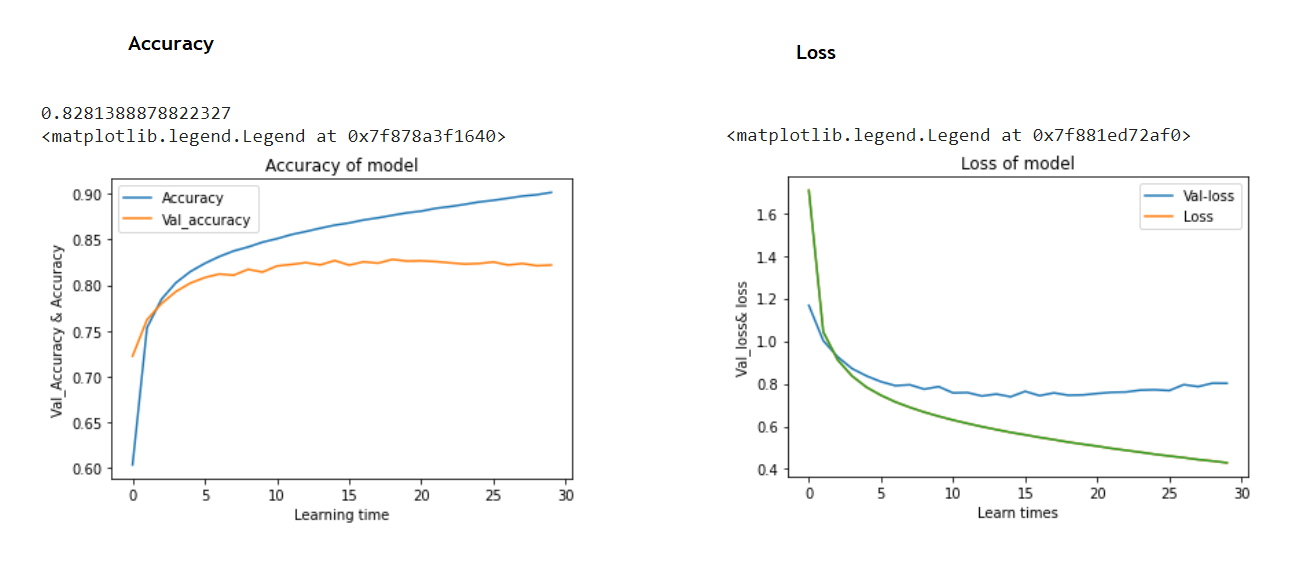
In the research phase, I studied different architectures and learned how to improve the performance by using techniques such as data augmentation, transfer learning, and feature extraction from pre-trained models.

After a lot of testting I come up with the following structure that resulted to be excellent in terms of performance. This final architecture, which will be followingly explained, consists basically on alternating 5 convolutional layers (followed by a non-linearity) with 2 max-pooling layers and, ending with 3 fully connected layers also followed by non-linearity.



**Fig7.** Suitable structure of CNN model

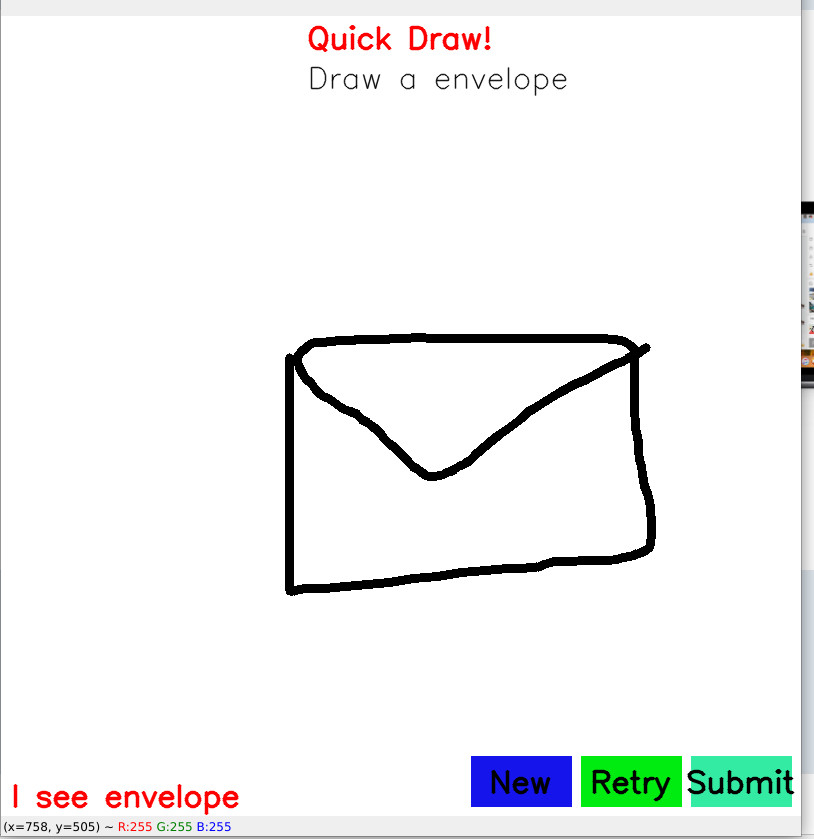
With the model above, I got the result as below:

**

After implementing the CNN architecture, compared the results with previous used MLP model and found that CNNs significantly improved the accuracy. I obtained an accuracy of 82.81% on the test set using the above model.

**5. Testing and conclusion**

I have developed an application that utilizes user input by allowing them to draw according to given prompts. The drawn images are then processed using basic image processing techniques before being passed through the trained model for evaluation. This process enables us to assess the stability and accuracy of the model, ensuring that it is suitable for the task at hand. This application not only allows users to have fun with the drawing task but also provides a way to check the performance of the model and make improvements if necessary.

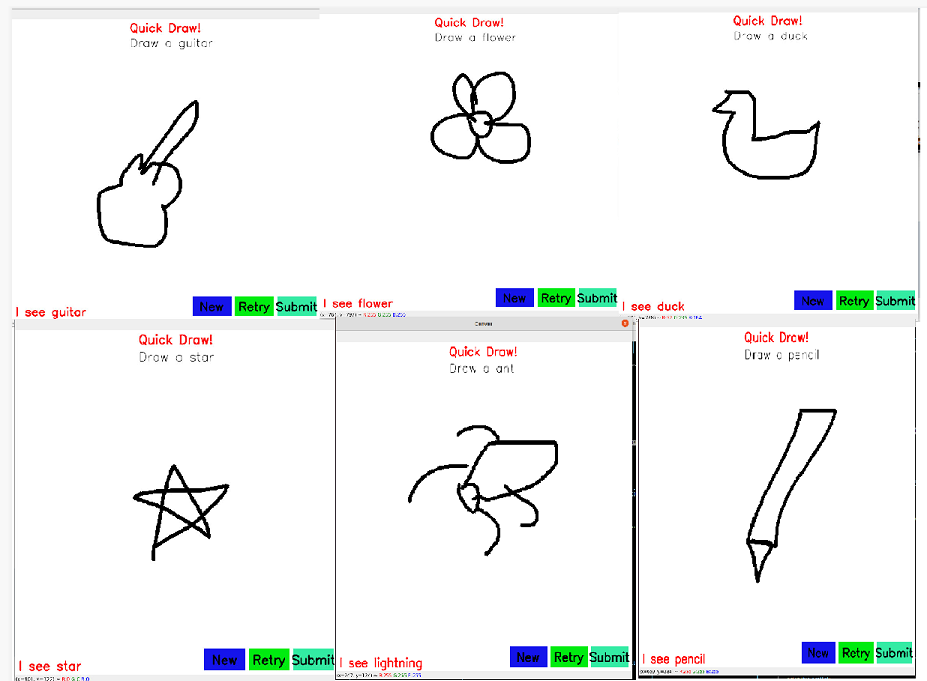


**Fig8.** *Application interface*

It features three buttons for the user to interact with, each serving a specific purpose. The first button is for requesting a new drawing assignment, the second button is for retrying a drawing task in case the machine fails to recognize it, and the third button is for submitting the drawing for the machine to evaluate and guess what the user has drawn.

**5.2. Result**

After several tests, the machine has shown to give quite accurate results, however, it still has a certain margin of error. This is due to the limitations of the model and the fact that the Quick Draw dataset used for training the model, while diverse, also contains some errors. These limitations and errors can cause the model to make incorrect guesses. To further improve the model's performance, it would be necessary to clean and pre-process the data more thoroughly, and also apply more complex architectures or techniques, such as transfer learning, to the model. These steps would help to reduce the number of errors and increase the overall accuracy of the model.

****

**Fig9**. Wrong and right guess of the model

**5.3. Conclusion**

This project marked my first experience with a deep learning problem. It help me encountered common deep learning challenges such as overfitting and hyperparameter tuning. Through this process, I have found that the CNN model performed the best with an accuracy of 83%..

In future research, I plan to test advanced CNN architectures like VGG-Net and ResNet for image classification in sketches. I also intend to use the full Quick, Draw! dataset. Additionally, I am interested in exploring ensembling techniques, particularly for lightweight methods like KNN, to improve accuracy.

# REFERENCE

**[1]** game Quick, Draw!; A.I Experiment, made by Google team: https://quickdraw.withgoogle.com/

**[2]** Understanding the effect of the Batch Normalization layers: <https://papers.nips.cc/paper/7996-understanding-batch-normalization.pdf>

[3] A CNN implementation of Google's "Quick, Draw!":

<https://github.com/ck090/Google_Quick_Draw>

[4] Quick, Draw, Keras! CNN model:

https://www.kaggle.com/code/amneves/quick-draw-keras-cnn-model