Free Your Mind - AI Platformer Game

Iris Kotsinas[1], Philip Ngo[2]

Abstract
Games and AI are becoming more and more integrated to give the user a better experience and make games more fun to play. The aim of this project was to make a 2D platformer game, where the player has access to an artificially intelligent book that can classify doodle drawings using a trained neural network. This is a game that challenges your creativity and gives you a freedom to play the game however you want to play it and solve problems however you want to solve it. The project starts with an attempt to build our own CNN that uses Google's "Quick, Draw!" dataset. Due to lack of access to powerful GPU/TPUs to train the large dataset, the number of classes was reduced to 50 instead of 345 and only contained animals. However, the final game uses a pre-trained model called DoodleNet to ensure fun and high quality gameplay.

Source code: https://gitlab.liu.se/iriko934/TDDD23_2D_game

Video (Game Trailer): https://www.youtube.com/watch?v=sAoJyfAJC10&ab_channel=IrisKotsinas

Authors
[1] *Media Technology Student at Linköping University, iriko934@student.liu.se*
[2] *Media Technology Student at Linköping University, phing272@student.liu.se*

Keywords: Deep Learning — CNN — Unity — Game

Contents

1 Introduction 1
2 Theory 1
2.1 Neural Networks 2
Perceptrons • Multi-layer Perceptrons
2.2 Convolutional Neural Networks 2
Convolutional Layer • Pooling Layer • Fully-Connected Layer
3 Method 3
3.1 Training of the model 3
Preprocessing • Architecture • Training
3.2 Game implementation 4
3.3 DoodleNet 4
4 Result 4
4.1 Our Model 4
4.2 The Game with DoodleNet 4
4.3 Accuracy of the models 5
5 Discussion 5
5.1 Our model vs DoodleNet 5
5.2 Future work 5
6 Conclusion 5
References 5

1. Introduction

The ability to recognize and classify doodles has important implications in computer vision and pattern recognition, especially in relation to high noise datasets. Games and AI are becoming more integrated to give the user a better experience, and the classification of doodles can be implemented into a game for a fun experience.

The aim of this project is to make a 2D platformer game, where the player has access to an artificially intelligent book that can classify doodle drawings using a trained neural network. The player should be able to draw an object, which then is sent to the model where predictions are made. These predictions should be sent back, and the player should have the option to choose the right prediction out of five different ones. The prediction should then appear in the game as a spawned object.

In this project, Google's "Quick, Draw!" dataset is used to train the model. The model is then evaluated and implemented into the game, and compared to another pre-trained model obtained from DoodleNet.

2. Theory

In order to understand how artificial intelligence can be applied to the classification of doodles, one needs to understand the basics of neural networks.

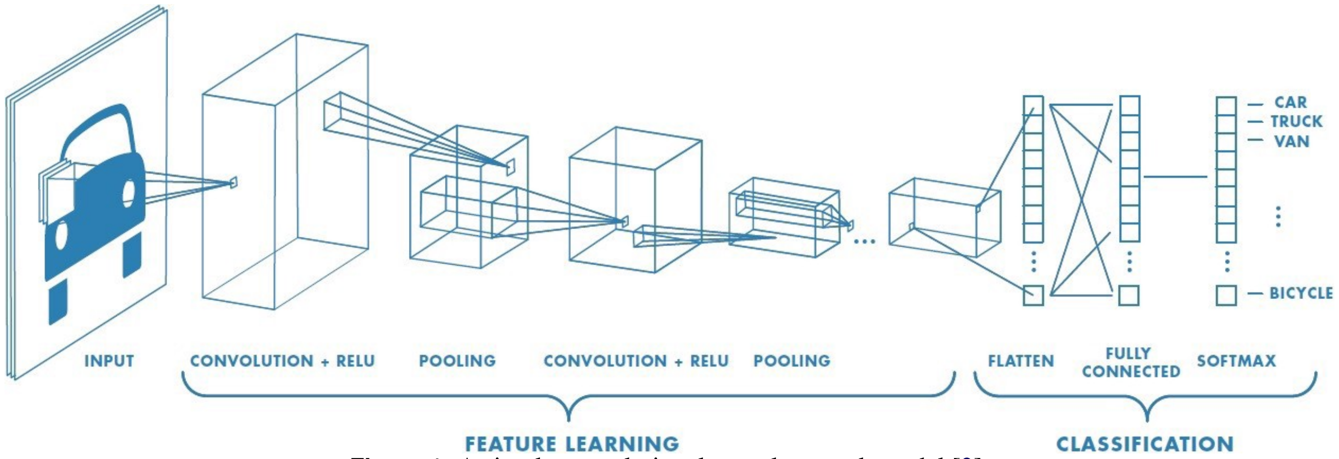


Figure 1. A simple convolutional neural network model [2].

2.1 Neural Networks

A neural network is a network of neurons composed of artificial neurons or nodes. Neural networks reflect the behavior of the human brain, which allows computers to recognize patterns and solve problems. By labeling or clustering raw input, they can interpret sensory data. They can recognize numerical patterns in vectors, into which real-world data is translated.

A neural network consists of many different layers of neurons, where each layer receives inputs from the previous layers, and passes outputs to further layers. They group unlabeled data by comparison of the example inputs, and classify data when a labeled dataset is available to train on [1].

2.1.1 Perceptrons

A perceptron is a type of artificial neuron, which takes different binary inputs and produces a single binary output. Weights can be applied to explain the importance of each input to the output. The neuron's output is determined by whether the weighted sum is less or greater than a set threshold value. Any layers in the middle are called *hidden layers* since their neurons are neither inputs nor outputs [9]. A perceptron with only one hidden layer is called a single-layer perceptron.

2.1.2 Multi-layer Perceptrons

Perceptrons with multiple hidden layers are called *multi-layer perceptrons* (MLPs). Such multi-layer perceptrons are designed to approximate any continuous function and can solve problems for non-linearly separable data. MLPs use a method called *backpropagation*, which is to model a given function by modifying the weights of the inputs to produce an expected output [7]. In Figure 2 an example of a multi-layer perceptron can be seen, with its inputs, hidden layers and output.

2.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a Deep Learning algorithm, which takes an input image and assigns importance to the various aspects and objects in it. It can then differentiate them from one another by constructing complex patterns using local features.

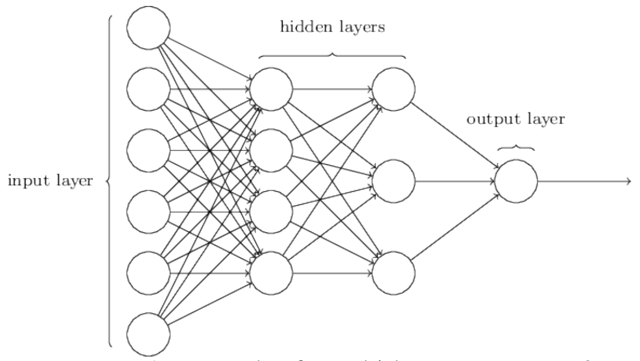


Figure 2. An example of a multi-layer perceptron [9].

A convolutional neural network requires less pre-processing compared to other classification algorithms, and can successfully capture spatial and temporal dependencies in an image though the application of filters. CNN reduces the images into a form which is easier to process, without losing features critical for getting a good prediction, and is therefore scalable to large datasets. A convolutional neural network allows better fitting of the image dataset due to the reduction in the number of parameters and reusability of weights [2].

A CNN consists of a sequence of layers and three main types of layers are used to build CNN architectures: Convolutional Layer, Pooling Layer and Fully-Connected Layer. An example of a simple CNN-model is shown in Figure 1.

2.2.1 Convolutional Layer

Convolutional layers are the major building blocks used in convolutional neural networks. Since regular neural networks require a large number weights, they do not scale well on images. Therefore, convolutional kernels can be used to extract the high level features, such as edges, from the input image [2].

A convolution is the simple application of a filter to an input that results in an output. A filter smaller than the input data is used as it allows the same filter to be multiplied by the input array multiple times at different points on the input, using dot product. A repeated application of the same filter to an input results in a two-dimensional array of output values, commonly called a *feature map*, where locations and strengths of a detected feature is enhanced. A large number of filters are automatically learnt in parallel, specific to a training dataset under the constraints of a specific predictive modeling problem. After the creation of the feature map, an activation function is used where each value in the feature map is passed through a non-linearity, such as the Rectified Linear activation function (ReLu) [5].

2.2.2 Pooling Layer

The output feature maps obtained from the convolutional layers are problematic in the sense that they are sensitive to the location of the features in the input. By down sampling the feature maps, this problem can be addressed. This results in the down sampled feature maps becoming more insensitive to changes in the position of the feature in the image, sometimes referred to as *local translation invariance* [4]. Pooling layers can be used in order to down sample feature maps. This is done by summarizing the features in patches of the feature map. Furthermore, they are useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model [2].

There are two common pooling operations: average pooling and max pooling. Average pooling summarizes the average presence of a feature from a patch of the image covered by the filter, and max pooling summarizes the most activated presence of a feature [4]. Max pooling is often used with CNN since it also performs as a noise suppressant. In Figure 3 max pooling and average pooling is performed with a 2x2 filter and a stride 2, which corresponds to the number of pixel shifts over the input [2].

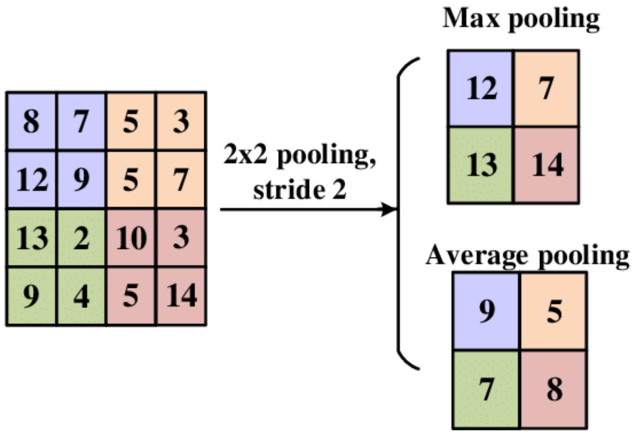


Figure 3. Pooling layer operation approaches: max pooling and average pooling [6].

2.2.3 Fully-Connected Layer

The Fully-Connected Layer is applied as one of the final steps in the CNN-model. Its input is the final output from the feature learning where convolutional and pooling layers are applied. After the feature learning the data needs to be classified into various classes, which can be obtained with the use of a fully-connected layer. The input is flattened into a vector before it is sent into the layer. A fully-connected layer connects every node to the nodes in the layers before and after it. After passing it through the fully-connected layers, the final layer applies the *softmax activation function*. This is used to obtain classification probability values of the input [11].

3. Method

The process was split into two different parts, the training of the model and the implementation of the model in the game.

3.1 Training of the model

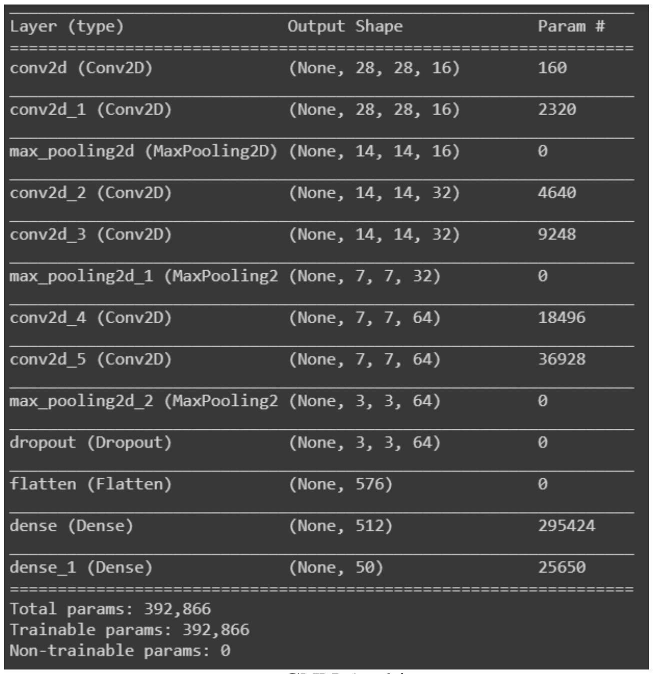
The model required preprocessing of the data, which was performed before the training of the model.

3.1.1 Preprocessing

The dataset was downloaded from Google's ”Quick, Draw!” dataset API, and then loaded into a Python script. There were different versions of the dataset to download, but for this project the *numpy bitmap format* was chosen. This is because all the drawings were simplified and rendered into a 28x28 grayscale bitmap in numpy which made it easier and less heavy to work with. The images were then normalized and divided into 90:10 ratio for training and testing.

3.1.2 Architecture

The CNN architecture can be seen in Figure 4 and was inspired by Yining Shi's DoodleNet [10], which will be discussed later in section 3.3.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 16)        160
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 16)        2320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 32)        4640
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 64)          18496
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 7, 64)          36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 64)          0
_________________________________________________________________
dropout (Dropout)            (None, 3, 3, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 512)               295424
_________________________________________________________________
dense_1 (Dense)              (None, 50)                25650
=================================================================
Total params: 392,866
Trainable params: 392,866
Non-trainable params: 0
```

Figure 4. CNN Architecture

The idea behind this architecture is to first run the image-tensor through two 2D-convolution layers to get the high level features such as edges as stated in section 2.2.1 and use max-pooling to down sample these images to remove the problem with sensitivity as stated in section 2.2.2. The process of 2D-convolution and down sampling is then repeated 2 more times. By adding multiple convolutional layers and max-pooling layers, the image is processed for feature extraction. The image is later run through a dropout layer which disables a percentage of neurons chosen at random. The dropout rate for this model is 10%. The tensor is later flattened and reduced to the 50 classes. The output is a softmax of the prediction for the classes which is based on section 2.2.3.

3.1.3 Training

The model was trained with a batch size of 256 and with 5 epochs with the Adam optimizer [8] set to the default learning rate of 0.001. The Adam optimizer is an extension of the standard stochastic gradient descent and has more benefits.

The training of the model was performed in *Google Colab*. Google Colab is a platform for execution of Python code in the browser, where one can utilise their GPU/TPU. However, Google Colab currently only supports a certain amount of memory and RAM, unless you buy the Google Colab Pro, which is currently only available in the US. This limited the training of the model to 10 000 images for 50 different classes. The total number of classes in the dataset is 345 but due to the lack of resources we chose 50. The classes chosen were only of type ”animals”, and was chosen for no particular reason, mostly because there is variation and fun to compare.

3.2 Game implementation

An API was built in order to enable the implementation of the model into the game. The API then called the model and sent back the prediction. This was built using *Node.js*, *Express* and *TensorFlow* for Node. The API takes in a base64 encoded string that represents the image and returns a top 5 prediction where the user can choose the correct one.

Inside the API, the image is converted into an image tensor and inverted since the drawing strokes should have the value 1 instead of the white background. The game calls the API every frame and sends the image to get a live update of the predictions.

3.3 DoodleNet

DoodleNet is a CNN doodle classifier, which our model is heavily inspired by. This is an existing pre-trained model that is trained with 50k images for all 345 classes of the ”Google Quick, Draw!” dataset. The model uses the same architecture described in Section 3.1.2.

4. Result

In this section, the result of the project is presented, which includes an evaluation of the model and the game application.

4.1 Our Model

While training and testing our model, a test accuracy of 90.65% was obtained after evaluating the model. The training took about 15 minutes per epoch and gave us a top accuracy of 91.34% after the final epoch.

4.2 The Game with DoodleNet

The final game uses DoodleNet instead of our model to make the game more enjoyable with more accurate results and more classes. The result of the game can be seen in Figures 5-8 and a link to the game trailer is provided in the abstract.

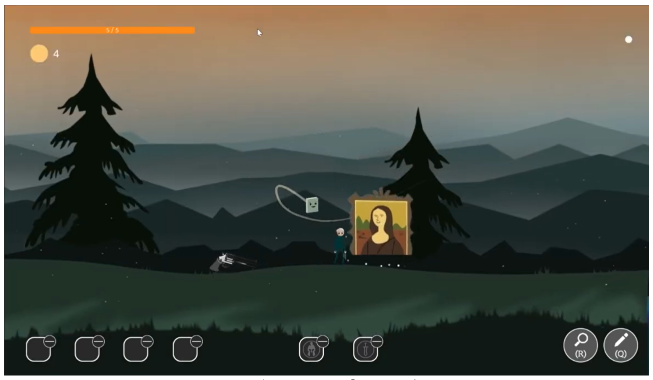


Figure 5. A scene from the game.

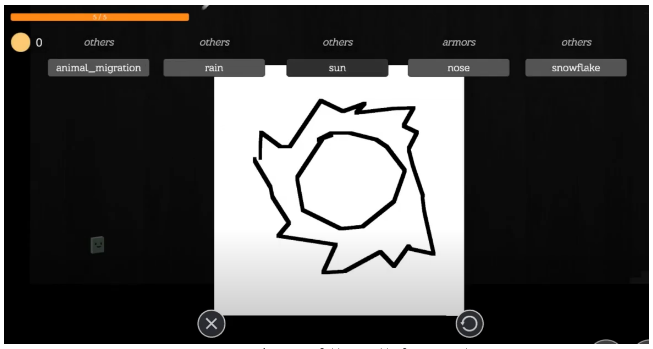


Figure 6. Drawing of ”sun” from the game.

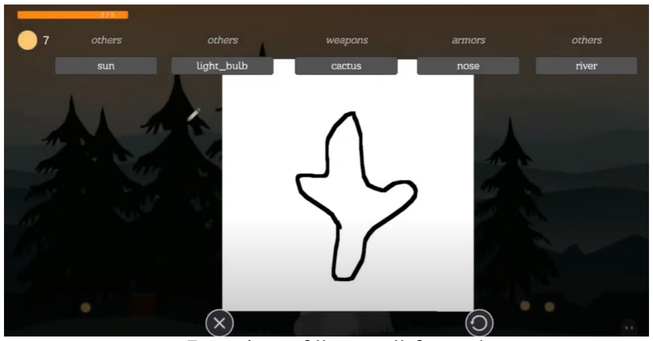


Figure 7. Drawing of ”cactus” from the game.

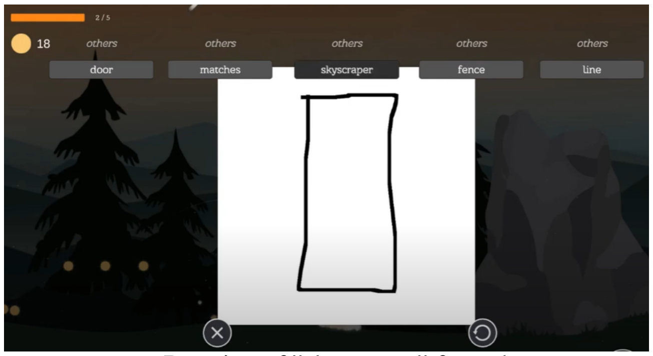


Figure 8. Drawing of ”skyscraper” from the game.

4.3 Accuracy of the models

The predictions from our model and DoodleNet were tested and noted. Three different objects were drawn and tested, and the predictions can be seen in Table 1.

Table 1. Predictions of a doodle depicting a horse and flamingo.

Model	Object	Prediction
Our model	Horse	0.5189
DoodleNet	Horse	0.1307
Our Model	Flamingo	0.3333
DoodleNet	Flamingo	0.003548

5. Discussion

5.1 Our model vs DoodleNet

As can be noted in Table 1, our model predicted animals better than DoodleNet's model. It can also be noted that DoodleNet predicted objects other than animals with higher accuracy compared to our model. The reason for this is that our model was trained only on animals, and therefore obtained a higher accuracy for that group, while it also produced a lower accuracy for the other groups. Since the training of our model focused on only one group and therefore could train on a higher amount of images, compared to DoodleNet, the accuracy increased.

5.2 Future work

Since the aim of the project was to have an interactive game that incorporates AI and gives the user a fun experience, we wanted to utilise the state of the art models. Even though they may not be developed by us, we understand the underlying idea but lack the resources to train the model to our liking.

One way to improve the current pre-trained model, DoodleNet, is to have a larger dataset since it currently only uses 34.5% of the full dataset of 50 million images. Another way would be to have a different model that takes into account the order of the strokes the user takes, since that information is available in the dataset. The data we are currently using to train the model is a simplified version.

As of now the game provides the player with the five most likely predictions. The use of a more accurate model would speed up the game by giving the player the right prediction at once, but this is a future improvement which requires a model with higher accuracy.

Something more we explored that would be fun is to have a speech comprehension into the game as well. We currently let the user talk as well but it only reacts to keywords and spawn items if a keyword is triggered. A future approach to make the game more enjoyable would be to let the dialog flow naturally between the user and the AI book using maybe OpenAI's GPT-3 [3] if it becomes available to the public.

6. Conclusion

In conclusion, we believe that we successfully created a fun and unique game, with an experience not seen before, by using a pre-trained state of the art model. Though there are improvement left to be done, we believe with the time frame given, we produced great results.

References

[1] Classification using neural networks. `https://towardsdatascience.com/classification-using-neural-networks-b8e98f3a904f`. Accessed: 2021-10-26.

[2] A comprehensive guide to convolutional neural networks. `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`. Accessed: 2021-10-26.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. 2020.

[4] J. Brownlee. A gentle introduction to pooling layers for convolutional neural networks. `https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/`, 2019. Accessed: 2021-10-27.

[5] J. Brownlee. How do convolutional layers work in deep learning neural networks? `https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/`, 2019. Accessed: 2021-10-27.

[6] K.-Y. L. Huo Yingge, Imran Ali. Deep neural networks on chip - a survey. `https://towardsdatascience.com/classification-`

`using-neural-networks-b8e98f3a904f`. Accessed: 2021-10-26.

[7] R. Keim. Understanding training formulas and backpropagation for multilayer perceptrons. `https://www.allaboutcircuits.com/technical-articles/understanding-training-formulas-and-backpropagation-for-multilayer-perceptrons/`. Accessed: 2021-10-27.

[8] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[9] M. A. Nielsen. Neural networks and deep learning. *Determination Press*, 2015.

[10] Y. Shi. doodlenet. `https://github.com/yining1023/doodleNet`, 2019.

[11] S. P. Singh. Fully connected layer: The brute force layer of a machine learning model. `https://iq.opengenus.org/fully-connected-layer/`. Accessed: 2021-10-27.