UNIVERSITAT DE BARCELONA

Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

IMPLEMENTING A SKETCH PREDICTION APP

Cantarino Sanchez, Jonatan Joaquín

Director: Meysam Madadi
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 10 de juny de 2021

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

Departament de Matemàtiques i Informàtica

A deep neural network approach for sketch recognition

by Jonatan Joaquín Cantarino Sanchez

The main topic of this work is object classification applied to the sketch domain. Starting from a huge set of object drawings, we have studied the problem of recognizing them in order to build a drawing app. This project, then consists in three main blocks, multiplatform app development that let users draw, deep neural network in order to recognize the drawings and an api that recieves the image and outputs the prediction.
The true challenge of this project is then to ensure a relaible recognition alltogether.

El principal tema de este trabajo es la clasificación de objetos en e l dominio de los dibujos. Partiendo de un enorme conjunto de dibujos de objetos, hemos estudiado el problema de reconocerlos y así desarrollar una aplicación de dibujo. Este proyecto, entonces consiste en tres bloques principales, Desarrollo de aplicación multiplataforma que permite dibujar a usuarios, redes neuronal para el reconocimiento de dichos dibujos y una api que recive la imagen y devuelve dicha predicción
La principal aspiración de este proyecto es entonces, asegurar una fiable predicción en su totalidad.

Acknowledgements

First of all, I want to thank my supervisor, Meysam Madadi, who has been steadily providing me an essential advice and prespective through all the project phases.

Starting from theory instruction of neural networks during the research, implementation advices during the development phase and supporting me on the thesis writing.

Secondly, I am grateful for the love of my family and friends, who have been inconditionally helping and encouraging me in every step of my career.

Contents

1. Introduction

1.1. **Motivation**

Sketching is one of the most innate tools for communication and a very intuitive way of understanding others, in fact, based on [1] paper, humans can identify category sketches 73% of the time.

After the constant digital evolution that society is shipped into, people tend to do most of their activities with their own devices. Specially now, due to the pandemic of COVID-19, that has accelerated this connection with devices and now, traditional habits, like drawing with a piece of paper are forgotten.

This leads to a problem with the educational process, where kids need to develop themselves, but they also want to connect with devices. This project would satisfy a proper way to let children grow their imagination [2].
Studying how people interpret and represent objects and their ideas is a field without limits in the understanding of the human mind.

Then, the main objective of this work is to present a system able to interact with people and classify their drawing images. Specifically, we are talking about object classification which is an important problem studied in the field of Computer Vision.

As Big Data has had a big growth in recent years, neural networks have emerged as a powerful tool for pattern recognition. A neural network is based on a set of neurons, that are stacked into layers and transmit signals to the next.

This paper will approach a solution based on deep learning, more concretely convolutional neural networks, which are a type of neural networks that operate with the convolution operation. The network will be fed by sketch drawings which are included in the Google Quickdraw Dataset [11] which contains millions of drawings across 345 different categories.

The dataset [11] offers vectorized and timestamped drawing data, but we will be using the final image instead. Our aim was to study a solution for the shape of the final image and accomplish a pattern recognition independent of the drawing device.

In other words, our project will be able to classify sketches from any type of image source.

Deep Learning is a complex field which contain several development possibilities with modular adaptability and since each use case may contain different requirements, the challenge is to find an optimal approach.

In this paper, we will go through many configurable techniques, which include dataset pre-processing, different Data Augmentation approaches, dynamic learning rate, many model architectures and optimizer configurations.

To present the final approach, we will present a full-stack scheme. This scheme contains a simple drawing application for mobile devices and a basic server API, which for the experiments was set up in a Google Cloud machine to test it remotely. The mobile application then, sends the image to the API which contains the deep learning structure and provides the prediction to the devices.

Basically, our project mainly presents the following contributions:

- An optimal neural network model for sketch recognition.
- An experimental and interactive full-stack application to test the network with sketches in real time.

1.2. **Summaryzing sections**

In this part of the introduction, we are going to introduce the upcoming fields of the thesis and its organization. In chapter 5, we are going to explain the state of the art in each field and related works.

A Definition of the terminology and a deeper dive in the concepts of neural networks can be found in the chapter 6, to get a better understanding of the field and models.

In the 7th chapter we can find a practical analysis and experiments done. Finally, in the last chapter 8th, we will give conclusions and discuss future works.

2. Related works

In this chapter, we are going to introduce different studies with similar problems and different approaches. Every work will be related to sketch recognition, using neural networks.

2.1.1. Handwritten Digits Recognition

This paper [3] presents a solid solution for MNIST dataset recognition.
Using a deep neural network, image pre-processing and data-augmentation.

2.1.2. Hand-Drawn Image Sketch recognition via Transfer Learning

In this study [4], a deep convolutional neural-network-based framework for sketch recognition via transfer learning with global average pooling strategy is proposed.

The dataset of this experiment contains different image sources, such as natural images filtered with different transformations of enhance edges and sketch images gathered by the authors.

Both of their dataset input result in a higher resolution input shape than the proposed dataset of this project.

2.1.3. A Neural Representation of Sketch Drawings

This project [5] presents a neural network solution for sketch reconstruction.

Their model can produce sketches in a vector format. They have constructed the QuickDraw dataset [11], which is the dataset used for our project. Nevertheless, they will be using the vectorized pen strokes dataset version instead of the final image representation.

From a recurrent neural network, their methodology solution provides different ways of finishing the current object being drawn and it also generates possible similar sketches.

2.1.4. Theory Introduction

2.1.4.1. Neural Networks.

A Neural Network is a computational model inspired in brain cells behavior.

How neural cells process information [6] has been inspiring researchers in the computer vision, text processing and industries field and now has become one of the most promising technologies in data science.

2.1.4.2. Convolutional Neural Networks.

Convolutional Neural Networks (CNN), derive their name from the Convolution operator which its main task is to extract features from an input.

CNN can learn complex and high-dimensional features from a large collection of samples. Unlike traditional computer vision models where manually defined features are extracted and used to train a classifier, in this approach the model can learn to extract relevant features.

These networks present invariance with respect to translation or distortions of the input image and can take advantage of the high correlation between nearby pixels of an image and identify simple patterns.

2.1.5. Datasets

2.1.5.1. MNIST

The MNIST (Modified National Institute of Standards and Technology) database is a large image dataset of handwritten digits, commonly used in the computer vision field for training image processing systems.

2.1.5.2. CIFAR-10

CIFAR-10 dataset (Canadian Institute For Advanced Research) is a collection of color images. The dataset contains 10 different classes, and it is also a common tool for computer vision and object classification.

2.1.6. CNN Model Approaches

In this study, different state-of-the-art deep neural networks have been utilized, each of these architectures are described in this chapter.

2.1.6.1. EfficientNet

EfficientNet architecture defines a method [7] that uniformly scales all dimensions of depth/width/resolution by using compound scaling.

2.1.6.2. Resnet50

Residual Networks are an extension of Deep Convolutional Networks wich include 'Skip connections'.
Skip connections layers add the output of previous layers to the next stacked layers, including some additional operations if required.

2.1.6.3. DenseNet

DenseNet, in contrast with ResNets, it never combines features [9] through summation before passing them into a layer. Instead, DenseNet combine features by concatenating them.

2.1.6.4. VGG

VGG, is a convolutional network [10] with an increased depth and small-sized convolution filters and small receptive fields.

3. Methodology

In the previous section, we have described the basic concepts of convolutional neural network (CNN), as well as the different CNN model architectures.
In the following section, we try to discuss the training process of our CNN model.

The training process mainly includes the following steps:

- Data pre-processing
- Parameter Initialization
- Model definition
- Training Performance Monitoring

3.1. Data pre-processings

In this section, we are going to define different preparation techniques which are going to be used before and during the training.

3.1.1. Image pre-processing

Given that all the network applications provided by Keras (Resnet, VGG, …) require a minimum 32x32 input shape and work better with RGB images, we have defined our data input as a 3-channel image of 32x32 pixels.

Then, every image of the dataset [11] will be resized and stacked in 3 channels with the goal of obtaining 32x32 RGB images.

3.1.2. Dataset pre-processing

To avoid resizing the image in every iteration, we have decided to rebuild the dataset and save it into a directory with the pre-processed image definition. Saving it with the directory structure defined by the Keras API, will let us train without loading it all into memory but load from batches with an iterator.

All these pre-processing methodologies have a huge computational load and harness and would delay many days for our larger dataset structure.

This leads us to the last pre-processing technic, which is parallel processing, that will speed up our performance. In our case the time spent was reduced an 80%. This performance depends on depends in the computer characteristics.

3.2. Review on CNN

3.2.1. The Convolution

The convolution operation consists of, given an input tensor and a kernel of fixed size. We slide the kernel over the input tensor and we apply the dot product of each component.

The output tensor this operation gives is called the feature map.

3.2.2. Convolutional Layer Properties

The convolutional layer and the resulting feature map are controlled by three main parameters.

Depth

Depth corresponds to the number of filters or kernels the convolution operation will have.

Stride

Stride is how many positions we skip when sliding the kernel over the input tensor.

Padding

Padding consists in adding zeros around the border of the input. It will let us apply the kernel in the border of the input as we slide over it. Also, it allows us to control the size of the resulting feature map.

RELU

Rectified Linear Unit (RELU), consists of replacing every negative value the feature map contains with zero and all positive values are linear.

3.2.3. Regularization methods

Normalization of the feature map outputs can accelerate the training process. There are several techniques available, we will be using Batch Normalization

and Dropout. Which simply normalizes the values before the next layer and drops some units, respectively.

3.2.4. Pooling Layers

Pooling Layers consists in 'summarizing' the output of a feature map, preserving spatial contiguity. There are many pooling layers types, but in this paper we will be using Max-Pooling.

Max-pooling requires an input pool size, for instance, 2x2 pool matrix. It does divide the feature map into regions of pool matrix size and outputs the maximum values observed in those regions.

We can find an example of Max pooling without padding in the following figure.

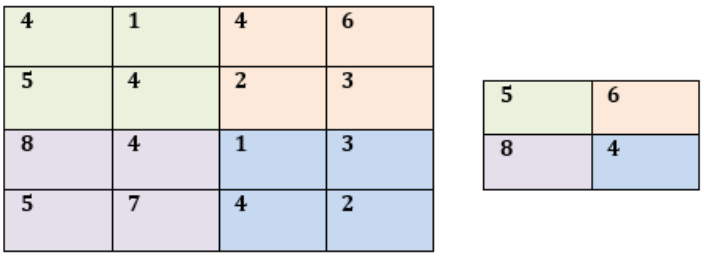

Figure 1 - Max Pooling

3.2.5. Optimizers

This section and the following explanations will be basically based on the book Artificial Intelligence By Example by Denis Rothman [8].

Optimizers are algorithms or methods that update parameters which brings better guidelines during the training and minimize the loss.

In this paper, we used two different optimizers, which are Stochastic Gradient Descent (SGD) and Adam.

SGD

Gradient Descent defines two main concepts which are Gradient and Descent. Gradient is the direction, if it goes up or down and its steep and Descent implies reducing the error level between the current result and the training dataset. In other words, it is a mechanism to optimize the results.

Stochastic Gradient descent consists of calculating gradient descent of random data instead of the whole dataset.

ADAM

The Adaptative Moment estimation optimizer represents an alternative to Stochastic Gradient Descent method which applies the optimizer to stochastic mini-batches of the dataset, making it a version of Stochastic Gradient Descent.

Adam includes root-mean-square deviation to the process by applying per-parameter learning weights and analyzes how fast the means of the weights are changing and adapts the weights.

3.2.6. Softmax

Softmax is a function used with the optimizer, that produces a vector of values in range (0, 1), with the particularity that all the elements sum up to 1.
Each element represents a class, and they can be interpreted as class probabilities.

3.2.7. Batch Size

The batch size parameter is the number of records, samples fed into the network each time. This will not be determined in definition time into the tensor, and it will be 'None' until the training starts.

3.2.8. Epochs

Epochs are the number of times we will run through the entire dataset during training.

3.2.9. Instances per class

We have defined this parameter to limit the input dataset to a smaller collection of samples for each class. During the experiment of this paper, we have been forced to define this parameter to test different techniques and contrast without loading the full dataset which is huge and would mean a significant computational cost.

3.3. Parameter Initialization

For the network weights initialization, we will be using the 'he_uniform' initialization, which randomly draws samples from a uniform distribution of the weights in the input tensor.

3.4. Model Architecture

Our model architecture approach for this paper, will be referred as 'tfgModel' in this paper and can be observed in the following figure.

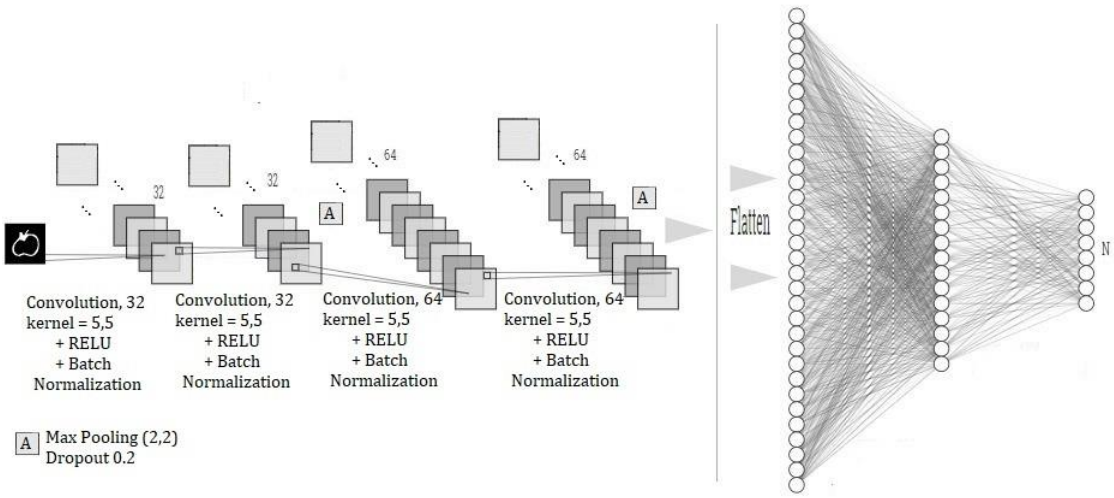

Figure 2 - 'tfgModel' Neural Network Architecture

The model represented in the figure [2], was selected after several simple neural networks architectures tests in Cifar10 and the MNIST datasets.

Since it was giving good results and it had less trend to overfit, we have decided to include it for the later studies of this project dataset.

As shown in the figure 2, it is basically a Sequence model, followed by four main convolution operations, then, after a flatten operation we find two dense fully connected layers, where the last dimension N represents the number of classes.

3.5. Metrics

Both metrics we are about to describe can be observed in every sub-set, such as train dataset or validation dataset.

3.5.1. Accuracy

Accuracy is a way to mesure how often the network classifies a sample correctly. It is formally defined by the following formula:

$$\frac{TruePositives + TrueNegatives}{TruePositives + FalsePositives + TrueNegatives + FalseNegatives}$$

Figure 3 - Accuracy formula

3.5.2. Loss

When we try to minimize the error, we try to minimize the loss function. The loss then, is a scalar calculated by the loss function.

3.6. Training performance monitoring

This section is a key feature of the training process and best performance approach.

Observing the different metrics defined in section 6.5, we can extract conclusions on the training performance. These conclusions let us interact with the training in execution time in several ways.

For instance, we can detect when a model does not improve after an epoch and ignore it by only saving the best model. We consider the best model contains the higher validation accuracy.

Additionally, we can observe if the training is overfitting after many epochs and stop the training.

When training, a period of no grow or decline may happen. This state is called Plateau and negatively affects the training since prolongs the number of epochs without changes.

One solution for this problem is training again with a lower learning rate. Nevertheless, in our project we are including a dynamic learning rate adjustment in time execution.

This adjustment will be applied after we see a plateau phenomenon recurrently for many epochs.

4. Experiments

4.1. Setup

4.1.1. Deep learning scheme

All the experiments about the training process exposed in this paper were run under the same setup configuration. An Intel i5-8600k processor with 16GB of memory RAM and a graphics card NVIDIA Geforce 2070 RTX with a dedicated memory RAM of 6GB, the hard drive was an SSD of 250GB.

About the software used, we were running the code in a version 3.5 of python with the module Keras.

After some research, we have found that in order to get benefit of the GPU processing, we must use the module tensorflow-gpu and use the keras version inside tensorflow instead of the keras module directly.

Then we could take advantage of the graphics card processor by using cuda (cuda_11.3.0_465.89_win10) and cudnn (cudnn-11.3-windows-x64-v8.2.0.53) packages.

4.1.2. API scheme

The API is a basic restful approach which implementation is very simple and only contains one input route. Designed with the Django framework and the same python version.
This API is deployed in a Google Cloud machine with a static IP address, in order to ease the communication from the device.

4.1.3. APP setup

The APP was built using the Ionic Framework, which for the drawing canvas contains an easy html layout and a TypeScript script implementation.

Ionic is a useful tool for app development since its multiplatform and consists in a web project that compiles an APK version for Android versions (IOS also if required).

4.1.4. Datasets

In this section, we describe the collection of the QuickDraw dataset [11] of human sketches, 365 categories and thousands of samples per category.

In the dataset are many different file structures available, we have chosen the numpy bitmap version which consists in, one .npy file for each category. Each file contains all the simplified drawings in a 28x28 grayscale bitmap. These images are aligned to the center of the bounding box, instead of the top-left corner.

We have defined three different subsets of this collection.

- Animal subset, 47 animal categories, evenly distributed.
- Random subset, 47 random categories, evenly distributed.
- Full subset, 365 categories, evenly distributed.

4.2. Results and discussion

4.2.1. Dataset experiments

To test the different subsets, defined in section 7.1.4, we have run a training for each one with the same configuration.

This configuration includes 1000 samples of each class, 64 batch size and 40 epochs.

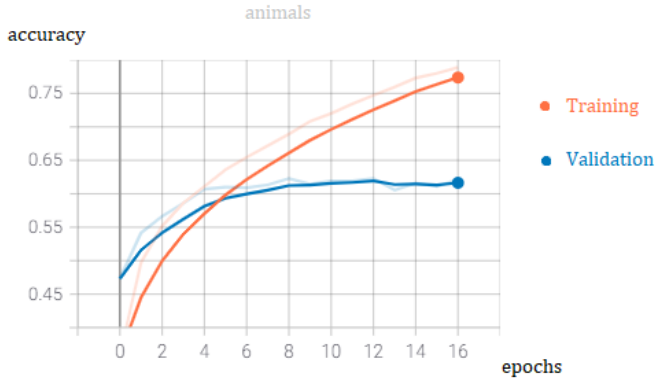

Figure 4 - Animals dataset training

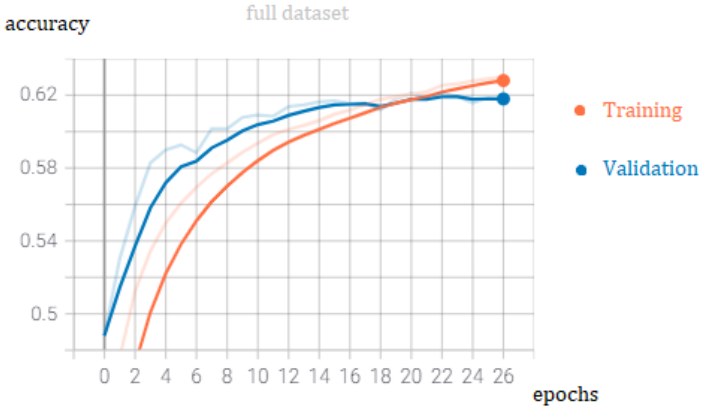

Figure 5 - Full dataset training

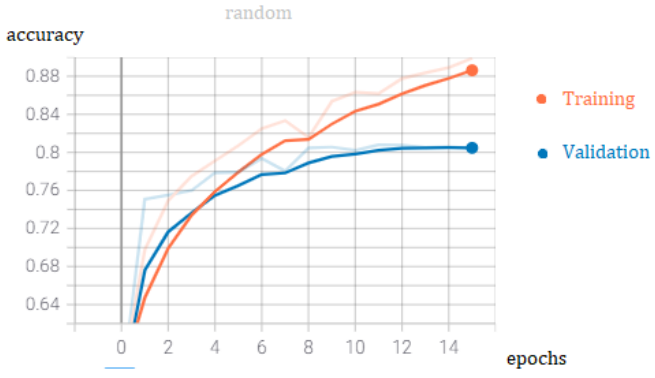

Figure 6 - Random dataset training

4.2.2. Input normalization

We tried to improve the visual perception of the images and improve the boundaries by correcting the resized images of the dataset by implementing a standardization. Which consists in subtracting the mean activity from each pixel and divide them by their standard deviation as explained in [3].

After several iterations and modifying the parameters, we realized this approach was not improving the results, the training accuracy was lower, and it was rejected afterwards.

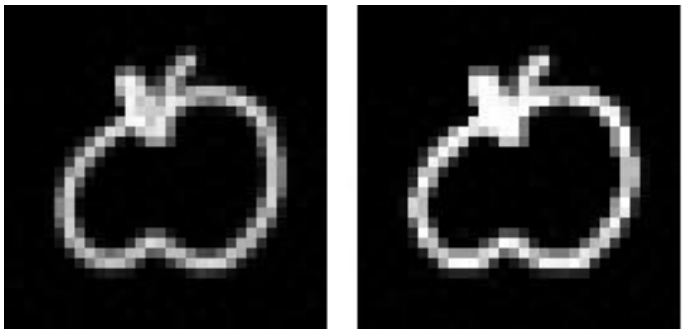

Figure 7 - Image standarization

In figure 7, we can observe a sample of the standardization process results.

4.2.3. Optimizer's test

To find the best optimizer approach, we have trained over the Animals subset with 1000 samples of each class and every neural network architecture defined.

SGD

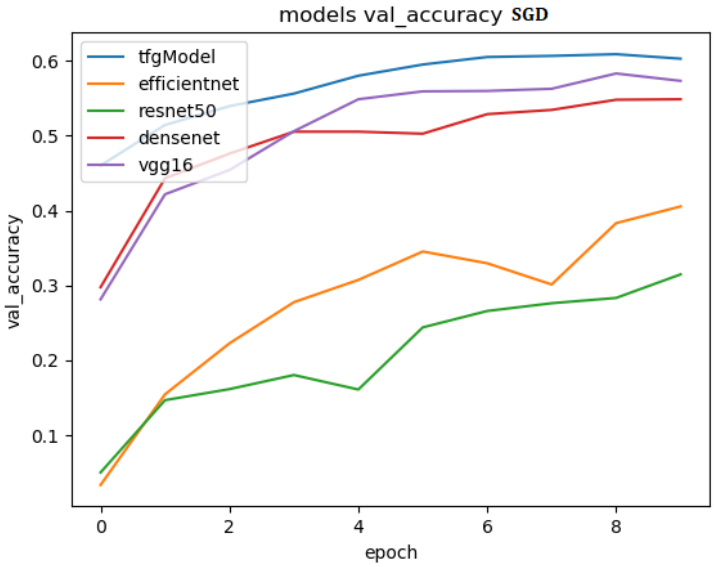

Figure 8 - SGD Optimizer Trainings

ADAM

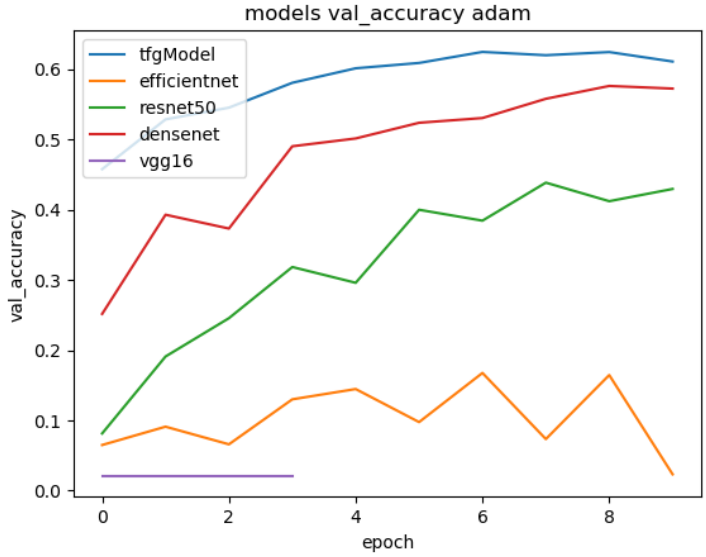

Figure 9 - ADAM Optimizer Trainings

We can observe that for this subset, the configuration of ADAM and the model we have constructed for this paper was the best approach.

After several runs contrasting this configuration, we decided to run the full dataset final training with the ADAM optimizer.

As we can see in the figure 9, 'tfgModel' was giving better results. One explanation could be that since the other model architectures are bigger, tend to be used for more complex problems and sketch classification is simpler.

4.2.4. Training Logs

Another key to monitor a model performance and increase it, is getting a solid log persistence of the training process. We have included features to keep additional metrics that, with the default Keras behavior we do not have.

For instance, we have included TensorBoard tool which checkpoints the performance after every epoch and added a new metric which is not included, the learning rate. This way we can study what happens after a plateau occurs and how the training is affected by reducing the learning rate. We can see an example of this behavior later, in section 7.2.6. Plateau.

4.2.5. Data augmentation experiments

Data augmentation is a powerful tool for many deep learning approaches. Given that the QuickDraw dataset [11] has a lot of samples and is not a limited input, we can assume Data Augmentation will not be needed only to increase the dataset size.

Nevertheless, it could be useful since it can learn from different points of view. For instance, since we realized the dataset has straight drawings, one data augmentation exploration could be rotation over some samples, which would allow new shapes of sketches for the model.

After these considerations and some tests, we have selected a set of data augmentation operations. Which includes, horizontal flip of the image, zooming, rotations, erosions, and dilations.

Some of these operations can crop some parts of the image and create new areas which we will be filling with the value of 0, which represent black.

After running several configurations, the training over best approach resulted in the following figure:

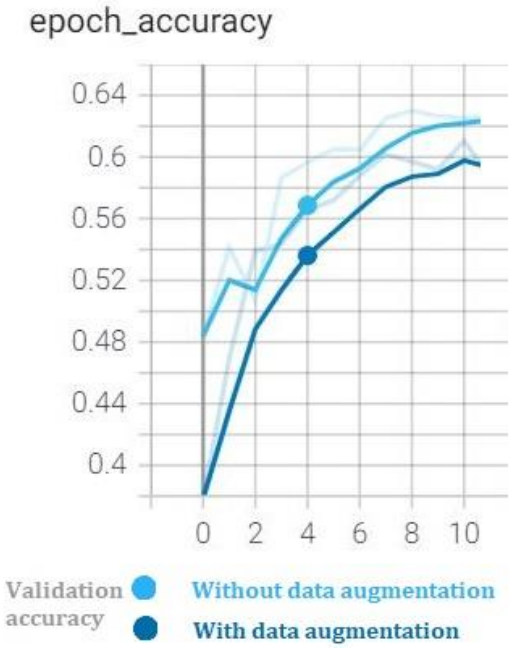

Figure 10 - Performance with Data augmentation

As we can observe in Figure 10, the training process with data augmentation was giving worse results than with the plain dataset.

We concluded then the model was having a better training with the default dataset.

The last data augmentation technic we explored was elastic transform. Elastic transform consists in reshaping some samples with curvatures.

This approach is successfully used in other deep learning problems, such as MNIST dataset, as explained in [3].

Our problem with this approach is that, since there are many similar classes with a close shape distance, for instance, a swan and a bird, a minimum shape modification could remove an essential part of the shape.

After several elastic transform parameter initialization, we found a configuration that we initially thought as a solution for our use case.

An iteration of this configuration and different elastic transform results can be observed in the following figures, which have been transformed, and resized from 32x32 pixels to insert them in this paper.

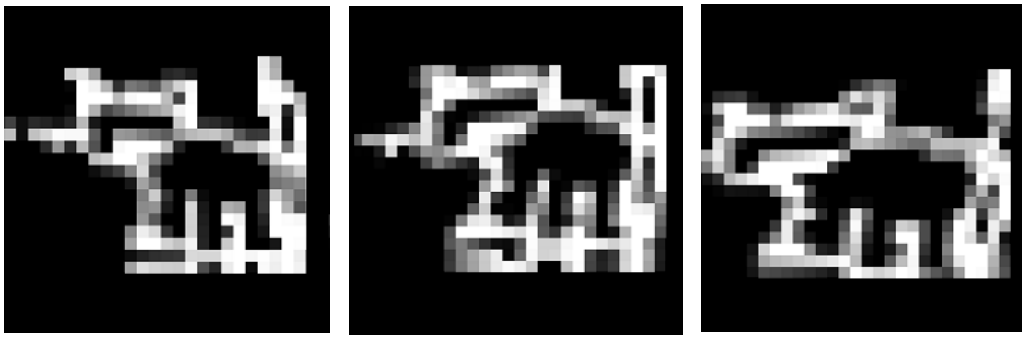

Figure 11 - Elastic Transform Results

The first sample in Figure 11, resulted in a good elastic transform result, which was preserving the shape of a cat. Nevertheless, the second and the third are closer to other animals, for example a tiger and a dog.

Then, we concluded this approach was partially corrupting the training and rejected afterwards.

4.2.6. Plateau

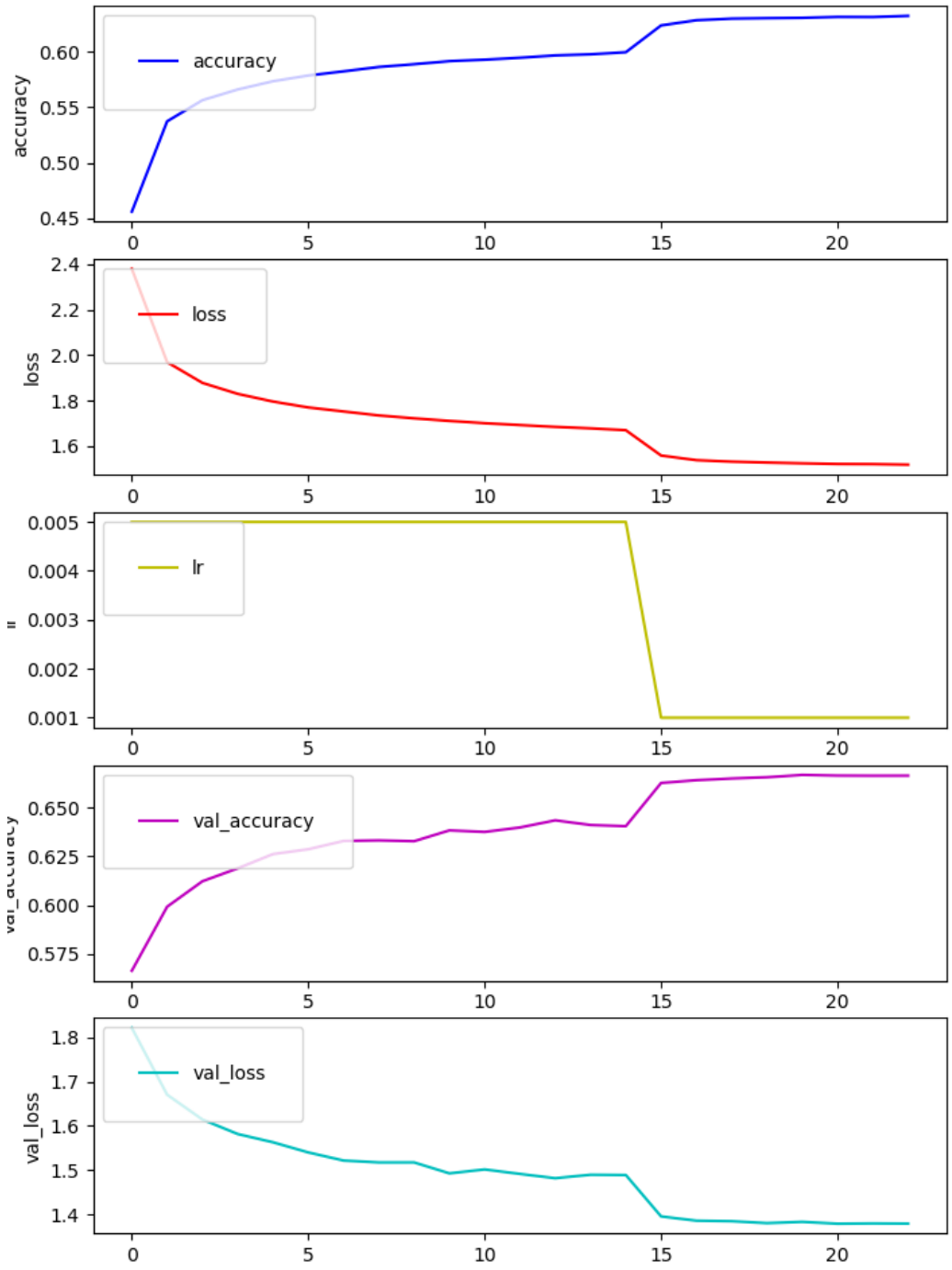

Figure 12 - Dynamic Learning Rate Metrics

A Plateau phenomenon occurs when the loss metric stops reducing and oscillates around a constant value. We can see in Figure 12 that the loss metrics get reduced when the learning rate is reduced, and the plateau is cancelled.

This example was run with the "tfgModel" model and the full dataset with 4000 instances of each class.

After these experiments we concluded the dynamic learning rate was an essential part of our approach.

4.2.7. HyperParameters

One key point for an optimal deep learning approach is finding a good hyperparameter configuration. We have two main configurable parameters which are the batch size and the number of epochs.

Using the animals dataset, described in section 7.1.4, we have run the training on the 'tfgModel' which is specified in section 6.4, over 1000 instances of each class.

In the following figures you can see the results obtained with the different parameter configuration:

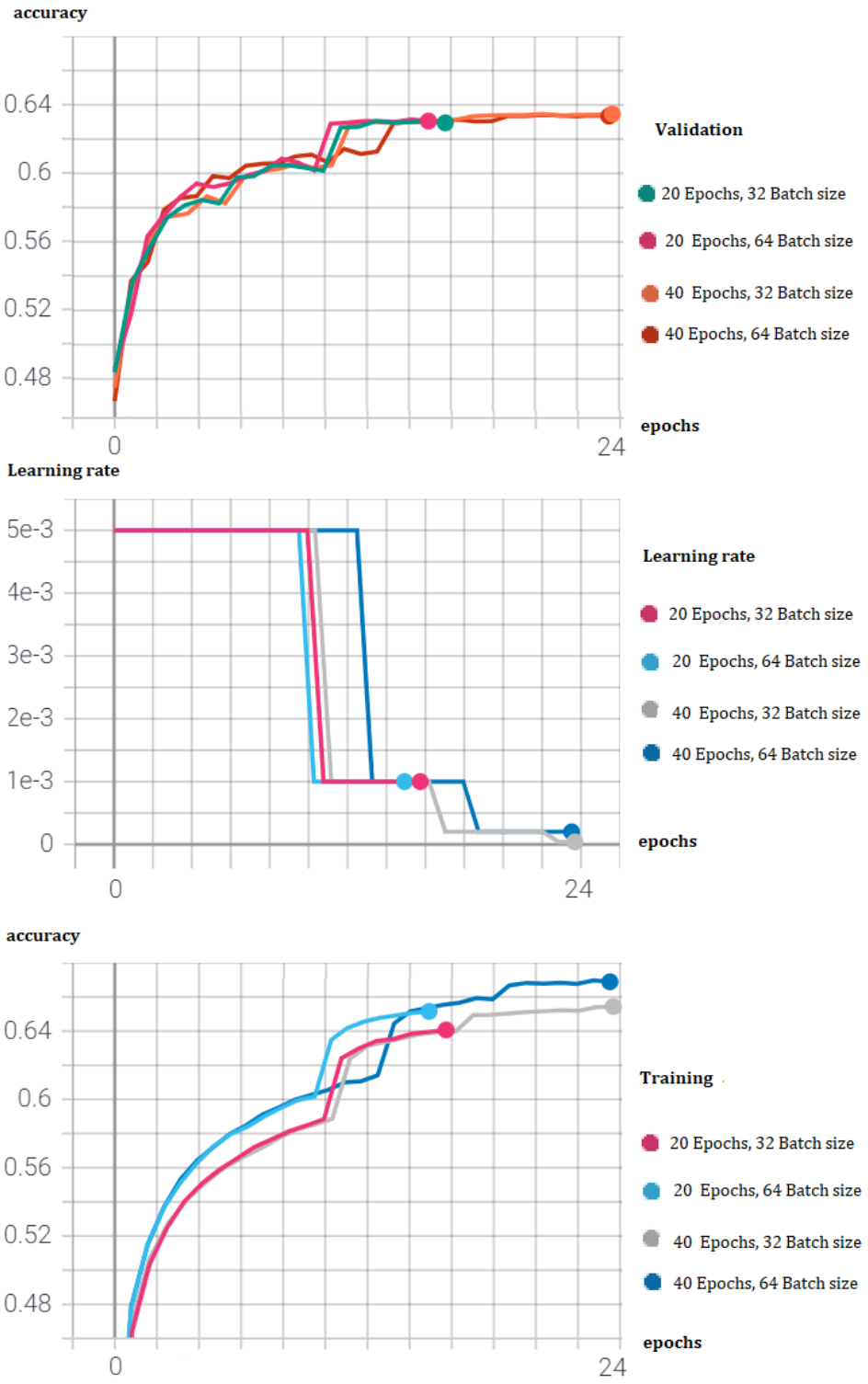

Figure 13 - Hyperparameters Performance

As we can see in the figure 13, the optimal parameter configuration was 40 epochs and 64 of batch size since it allowed the model to grow with a better accuracy.

4.3. App

In this section we are going to explain the evolution of the application development process regarding the results obtained.

When this project was started, it was limited to the neural network solution, our aim was to ensure a good model contribution for this problem.

Nevertheless, after all the training improvements, we wanted to test it with our hands and above it, we wanted to offer a solution to test it.

Usually, in the neural network field it is difficult to see a context where you can test it in real life, excluding image classification with smart devices and some other daily life applications.

Then, when the drawing app idea came onto the table, one step higher was suggested. The drawing application could predict in real time on the flow, as the user draws.

The challenge and the counterpart at the same time in this development was to ensure a fast connection between the device and the neural network. If we wanted to ensure a fast connection, pre-processing the image should be done in the device.

Resizing the image in the application would ensure a lower size http call, and consequently a faster communication.

Figure 14 - Interactive App Predictions

As we can observe in the figures 14, the interactive APP starts with a kangaroo prediction as soon as it's head and belly are drawn, when we keep drawing it's feet and tail it increases the probability.

Nevertheless, if we transform the tail into a squirrel tail it rectifies the prediction and successfully outputs the result.

5. Conclusions and future work

We must consider that, logically, the random subset will reach a higher accuracy since the objects will be more different among them than animals. For instance, within the animals dataset we can find many related classes, such as, tiger, cat, dog or duck, bird, swan, parrot, etc.

In addition, the full dataset will have the lower accuracy since it is a bigger dataset and present the animals subset problem but with other objects.

6. Bibliography

[1] How Do Humans Sketch Objects? ACM Transactions on Graphics (Proc. SIGGRAPH 2012). Mathias Eitz, James Hays and Marc Alexa

[2] Bozzola, E., Spina, G., Ruggiero, M., "Media devices in pre-school children: the recommendations of the Italian pediatric society."

[3] Jay Gupta. Going beyond 99% — MNIST Handwritten Digits Recognition

[4] Shaukat Hayat, Kun She, Muhammad Mateen and Yao Yu, "Deep CNN-based Features for Hand-Drawn Sketch Recognition via Transfer Learning Approach"

[5] David Ha, Douglas Eck, "A Neural Representation of Sketch Drawings"

[6] Ujjwal Karn. "A Quick Introduction to Neural Networks"

[7] Mingxing Tan, Quoc V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks"

[8] Denis Rothman, "Artificial Intelligence By Example"

[9] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, "Densely Connected Convolutional Networks"

[10] Karen Simonyan, Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition"

[11] Google, "Quickdraw Dataset by Google"