UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

FACIAL EXPRESSION CLASSIFIER BASED ON MACHINE LEARNING ALGORITHMS

VALERO PUCHE, AARÓN

Tutor: Llobet Azpitarte, Rafael

This bachelor's thesis is presented at the Telecommunication engineering school at Polytechnic University of Valencia, to complete and certify the bachelor's degree in Telecommunications Engineering.

Academic course 2017-18

Valencia, 1st of August 2018

Abstract

Sentiment analysis is a hot topic nowadays, an algorithm extracts subjective information from users, and helps a business to monitor the social reaction toward certain product or service. Sentiment can be extracted from different sources, e.g. speech, text, images. In this document, we are going to focus on computer vision algorithms using images as input data. In recent years, convolutional neural networks (CNN) have become the state-of-art deep learning models in image recognition. They have the power of deep learning as well as optimized feature extraction capabilities. CNNs are considered the top most accurate models in last decade, and new architectures are constantly being developed. This paper proposes several convolutional neural network designs, trained with the greatest public dataset in facial expression, AffectNet, with over 420K manually labeled images. Once the model is trained and tested, the most accurate model is deployed in Affective, a photo manager application capable of classifying stored images in any android device according to 8 facial expressions.

Resumen

El análisis de sentimientos es un tema en auge hoy en día, Las empresas son capaces de extraer la información subjetiva proporcionada por los usuarios, y así tener una opinión acerca de un producto o servicio. Los sentimientos pueden extraerse de muchos formas, la naturaleza de los datos puede ser; el habla, escrito, imágenes. En este documento nos centramos en los algoritmos de visión artificial usando imagines como datos de entrada. En los últimos años, las redes neuronales convolucionales se han convertido en la piedra angular del reconociendo de imágenes. Estas redes están consideradas las más precisas de última década, y los desarrolladores de esta tecnología crean nuevas arquitecturas constantemente. Este documento propone varios diseños de redes neuronales y entrenados con el conjunto de datos más grande hasta la fecha en análisis de sentimientos atreves de imagen, AffectNet, con más de 420 mil imágenes etiquetadas manualmente. Una vez entrenado y testeado, el diseño más optimo se desplegará en Affective, una aplicaciones de gestión de imágenes capaz de filtrar 8 expresiones faciales sobre imágenes guardadas en cualquier dispositivo Android.

Resum

L'anàlisi de sentiments és un tema en alça hui en dia, Les empreses són capaços d'extraure l’informació subjectiva proporcionada pels usuaris, i així tindre una opinió sobre un producte o servici. Els sentiments poden extraure's de moltes formes, la naturalesa de les dades pot ser; la parla, escrit, imatges. En este document ens centrem en els algoritmes de visió artificial utiliçant imatges com a dades d'entrada. En els últims anys, les xarxes neuronals convolucionals s'han convertit en la pedra angular del detecció d'imatges. Estes xarxes estan considerades les més precises de l'última dècada, i els desenvolupadors d'esta tecnologia creen noves arquitectures constantment. Este document proposa diversos dissenys de xarxes neuronals i entrenats amb el conjunt de dades més gran fins a la data en anàlisi de sentiments atrevixes d'imatge, AffectNet, amb més de 420 mil imatges etiquetades manualment. Una vegada entrenat i probat, el disseny més òptim es desplegarà en Affective, una aplicacions de gestió d'imatges capaç de filtrar 8 expressions facials sobre les imatges emmagatzemades en qualsevol dispositiu Android

Table of content

1 Objectives 3
2 Introduction 4
2.1 Basic knowledge in Machine learning 4
2.1.1 Neural networks 5
2.1.2 Convolutional neural networks 8
2.2 Developing tools 10
2.2.1 Facial expression classifier. 10
2.2.2 Android application 11
2.3 Resources 11
2.3.1 Datasets 11
2.3.2 Hardware 13
3 Methodology 15
3.1 Reference models 15
3.1.1 Simple Baseline 15
3.1.2 State of art architecture 16
3.2 CNN design 18
3.2.1 Parameters 18
3.2.2 Architecture designs 18
3.3 Training workflow 23
3.3.1 Input stage 23
3.3.2 Training 25
3.3.3 Testing. 25
3.3.4 Export 26
3.4 Android app 26
3.4.1 Libraries 26
3.4.2 User guide 27
3.4.3 Development 29
4 Results 31
4.1 Classifier 31
4.2 App performance 34
5 Cost summary 36
6 Conclusion 37
7 References 38

1 Objectives

The main target of this project is to achieve a competitive supervised algorithm for facial expression recognition, with an accuracy of 70% or more, which is able to identify the predominant facial expression in any picture that contains a centered face in it, however not only a global accuracy of 70% but a decent accuracy per each classification label, over 50%. As mentioned in the abstract, the selected technology to carry out the classification is the CNN, with great performance in image recognition. Training a model and improving its classification require considerable time, and under certain conditions it can be exponentially increased; such as large datasets and big architectures. The main goal of this project is to face up a huge dataset with remote hardware, reducing the developing time and thus decreasing the cost. The final application of this model is an android application. Mobile devices cannot operate as fast as a server or domestic pc and neither store same amount of data, therefore, the final model needs to be smaller than well-known architectures, such as GoogleNet (23 million of parameters) or VGG (138 million of parameters). At the final model we will consider 10 million as maximum number of trainable parameters.

In terms of the android application, the photo manager requires several operations to classify a single image: facial detection and its location, image preprocessing, and executing the classifier with the normalized tensor. These three steps in a mobile can take longer time than expected, since the mobile resources are limited. Hence the application needs to be evaluated in the time domain. We consider a time of 0.5s per image classified as the maximum time that a user would tolerate.

2 Introduction

First, before starting to review the methodology and the results of the classifier, it would be important to cover the tools and resources used to carry out both the development of machine learning algorithms and android app development. Furthermore, this section contains a basic knowledge explanation about neural network and convolutional neural networks.

2.1 Basic knowledge in Machine learning

Machine learning (ML) is a subset of a more abstract and well-known concept, Artificial Intelligence (AI). It consists of data analysis techniques and statistical models that give computer the ability to "learn", without being explicitly programmed. Machine learning models apply a bunch of statistical mechanisms that allow the model to explore data, correcting its parameters to obtain better accuracy. With the appearance of internet and social networks, the amount of data was exponentially increased. Thanks to ML algorithms, and being allowed by the hardware speed and capacity enhancement, machine learning algorithms are presented in countless business areas and have changed daily's life of society.

There are subgroups among this technology, ML algorithms are commonly divided into categories according to their purposes and implementations.

- Supervised learning

It is a type of ML model that requires a labeled output to be trained. In order to train the model, first it processes the input data and makes a prediction, while it compares the predicted output (label) right after. The loss function reveals how accurate was the prediction, and finally, an optimization function adjusts the weights of the algorithm based on the cost.

- Unsupervised learning

At this group of ML, the data is not required to be labeled. This type of task cannot be learnt like a regression model or classifier, but instead, it can be used to discover new structures or patterns in the data.

- Semi-supervised Learning

It is generally a combination of previous two subgroups. First, a supervised classifier or regression model is trained with labeled data. After that, the pre-trained model is then applied to the unlabeled data to generate more labeled data as inputs for the supervised learning model. Data augmentation is a typical procedure to expand the dataset and reduce over fitting.

- Reinforcement Learning

It is an area of ML, inspired by behaviorist psychology. It is concerned with how an agent ought to take actions in an environment so as to maximize the rewards, which is evaluated based on the agent target.

Once the main ML groups are listed, let's continue with a brief introduction to neural networks and convolutional neural networks; The state-of-art models of supervised learning.

2.1.1 Neural networks

Neural network is a supervised model; it is a computing system inspired by biological neural networks. Neural networks are widely used because of the robustness and accuracy for classification tasks, by reaching great performance for most of recent data analysis problems. Multiple designs and architectures are being upgraded nowadays. Depending on the nature of the data, machine learning experts readapt these adjustable structures to the specific need of the data. For instance, convolutional neural networks use a simple neural network plus a set of convolutional layers that extract the features of the images. Or, another example would be a recurrent neural network, that provides to the network the ability to keep certain memory of past stages.

A neural network is a set of interconnected layers, each layer contains a vector of perceptron or neurons.

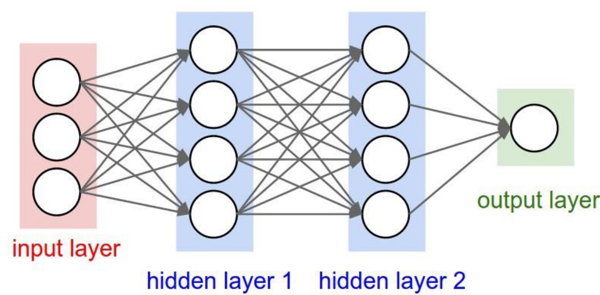

Figure 1 ***a simple neural network with; an input layer, two hidden layers and the output layer***

It is obvious to notice that the input layer will receive the raw input data. Then, the data is transmitted to the following layers. In most of the cases, each perceptron is connected to every perceptron of the next layer, which will apply a linear combination and normalization to the input values. Once the process is completed throughout the entire hidden layers, the data ends up at the output layer (*forward propagation*). The output layer is the responsible of yielding the prediction of the input data when prediction mode is activated, and it is the start point of the *back propagation task,* which will be explained in this section below.

Focusing on a single neuron, the elemental asset of the network, is basically a summation of the weighted input values. After summing up those inputs, a value called bias is added to the summation result in order to reduce offset deviation (Bias). Finally, the value is passed to an activation function.

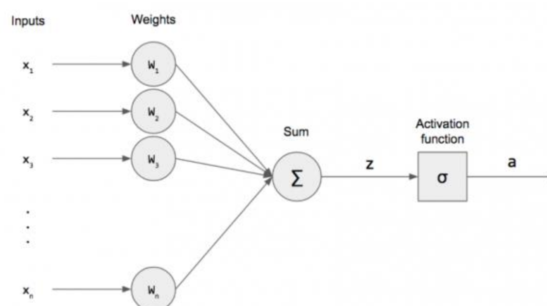

Figure 2 Diagram of a single perceptron/neuron. Weighting the input, summing them and then applying an activation function.

The activation or trigger function will normalize this value according to the shape of the chosen function, and it takes responsibility for designing to choose the best function for the specific case. This function basically will give a binary or fuzzy binary output depending on the magnitude of the input value. There are a bunch of activation functions, some of the them are explained below:

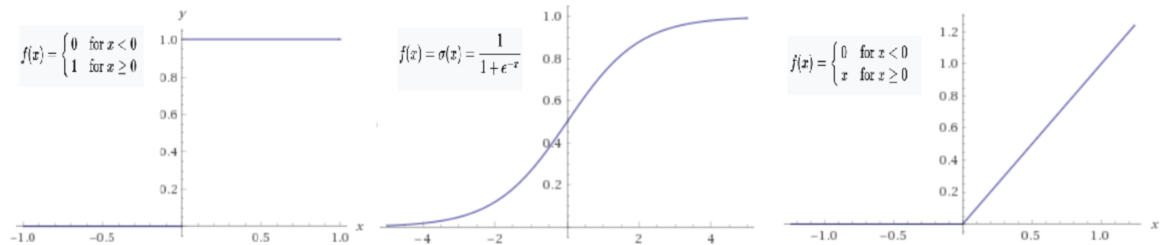

Figure 3 From left to right, Step function, sigmoid function and RELU function

It is easy to notice, the step function, which is the simplest one, and widely used in digital electronics as a trigger function. This function is not a good option; since a small variation in the input value could flip the output value, by affecting to the final prediction. An optimal threshold point can be found, but the stated issue is unavoidable for those values that fall around the boundary of the step. On the other hand, sigmoid functions solve this issue, by using a logistical function, which smooths out its behavior. Last activation function depicted is the Rectified linear unit. It is a simple and efficient function for hidden layers and specially used in CNNs because of its simplicity, in addition, it allows to the model to be trained with less operations than a typical sigmoid function achieving similar performance.

The neuron's input data is weighted, but which weight/bias values are assigned? When a model is trained from scratch, typically all these parameters are randomly selected. Obviously, any prediction carried out will wrongly predict the label of the input data. Then is when the forward propagation and backwards propagation comes in.

Forward and back propagation are the two main tasks of the training process of any neural network. Forward propagation is understood as the prediction task, the input data is ingested into the input layer, propagated throughout all layers with the regarding weights and biases until reaching the output layer.

Once the prediction is finalized, a loss function will evaluate how far the prediction was from the actual value. Since is a supervised learning, the actual value is provided in the training set. Lastly, based on the cost calculated, the optimization function, which is a gradient function, will find the path of the minimum error. Layer by layer from output to input, it will readjust all weights and biases to a more optimal value. This is commonly known as back propagation. There are several functions and tuning parameters to be chosen.

Down below are some of the most common loss and optimizer functions that Keras and TensorFlow offer:

Loss function

- Categorical cross-entropy: calculates the entropy between two distributions, particularly the real distribution and the predicted by the algorithm. It works with a one hot configuration.

$$H(y, y') = \sum_i yi \log \frac{1}{yi'} = -\sum_i yi \log yi' \quad \textbf{(1)}$$

- Sparse categorical cross-entropy: Same as previous cross-entropy but output values are integer encoded.
- Binary cross entropy: It only works for binary output.

Some of these functions are ideal for regressions like: mean squared error or mean absolute error. Although, it is better to use categorical functions because of the binary nature of the output.

Regarding the optimization function, here we have the most relevant:

- Stochastic gradient descent (SGD): computes the gradient of the cost function for the entire training dataset. SGD has fluctuations and it often miss the optimal adjustment.

$$v_{t\theta}= \gamma v_t - 1 + \eta \nabla_\theta J(\theta) \quad \textbf{(2)}$$

$$\theta = \theta - v_t \quad \textbf{(3)}$$

  Where η is the learning rate(lr), θ is the function, J() is the loss function and γ is the momentum. In Keras, Lr = 0.01 and momentum = 0 as default value.

- Adagrad: is an algorithm for gradient-based optimization that does just adapt the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. In Keras, Lr = 0.01 as default value.

- Adadelta: is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size w. In Keras, Lr = 0.01 and momentum = 0 as default value.

- Adaptive Moment Estimation (Adam) [4]: is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients, similar to momentum value.

  The author of adam's recommends as well as Keras, Lr = 0.001, β_1= 0.9 and β_2= 0.999 as default value. They empirically show that Adam works well in practice and can be compared favorably to other adaptive learning-method algorithms.

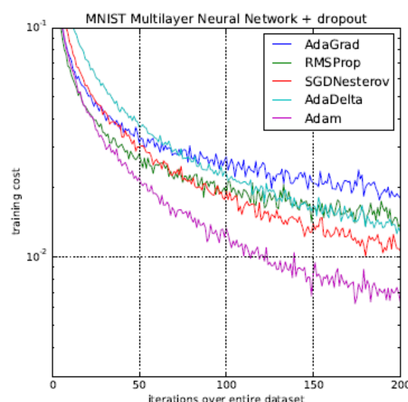

Figure 4 Optimization function comparison, this plot represents the cost function while training with MNIST, a number handwritten dataset.

As we can see, Adam optimizer is empirically the best optimization function. It is highly recommended to use the author's proposed default values.

2.1.2 Convolutional neural networks

The aim neural network has to deal with images as an input data. Since input neurons can handle only one value, the simplest approach would be assigning each neuron to each one pixel of the image, although it does not perform well. It is concluded that a small variation of the pixel value, in terms of brightness, contrast or any image parameter can affect our prediction. Namely, even there are two images with the same shape but different brightness, a regular neural network could predict different results just because each neuron is focusing on pixel level. The main goal to solve this issue is to extract the features of the images locally but not point by point. Images have only meaning when they are taken in groups of pixels and increasing the receptive field, defining the receptive field as; the minimum pixels change in an image that provokes a change at the particular layer neuron.

CNNs allows us to extract the features of the image, and then process it like a traditional network with a fully connected architecture. CNNs uses a variation of multilayer perceptron design to require minimal preprocessing, by using convolution and pooling operations. They are also known as shift invariant or space invariant artificial neural networks (SIANN), It is also computationally efficient and accurate. These facts make convolutional neural network universally attractive.

A common architecture followed to design a convolutional net is depicted in the next figure:

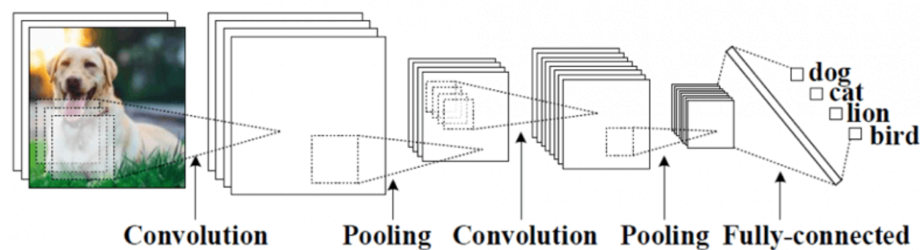

Figure 5 Simple convolutional neural network, it is composed by 2 convolutional layers and 2 pool layers. And finally connected to a traditional fully connected neural net

The input image is generally normalized to reduce computation cost; the image is typically normalized with a Gaussian distribution or simply using a linear normalization. First layer is usually a convolutional layer and depending on the designer and the architecture, it can be concatenated with several conv. layers. Convolutional layers are usually combined with pool layers. At the last part of the network, there is a fully connected block. The following paragraphs give a brief description the relevant layer in CNNs.

A convolutional layer is basically a sliding window that applies convolutions to the input matrix and so getting a convolved image. The window, or commonly called as kernel, is a weight matrix that operates a dot product against the input region. Namely, it multiplies and sums up a region of the input image with the kernel, obtaining a representative scalar value. Once the kernel operates the dot product to certain region, it jumps to the next region according to a configuration step parameter called *Slide.* Additionally, there is another parameter called *padding*, as you might guess is a 0's frame that wraps the input image or feature map. A feature map is the resulting matrix after applying an operational layer such as convolutional layer.

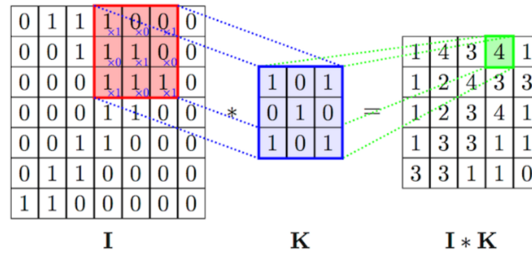

*Figure 6 I matrix is the input image, K is the kernel and the I*K is the resulting feature map. The convolutional layer applies the dot product of the sliding window(blue) to a region of the input image (red). As a result, a single value is obtained (green value)*

Another important layer is the pool layer. The aim of pool layers is to reduce the information by applying pixel region to average or max value selection, increasing the receptive field.

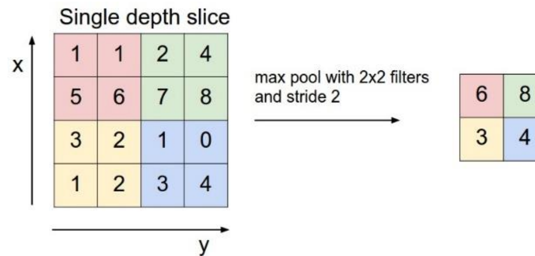

Figure 7 Max pool layer with stride. The left matrix is the input matrix and the right one corresponds to the output matrix. Each operation or window slide is depicted with a unique color.

Dropout layer [6] is more recent layer than the other cited layer. It was first used in AlexNet model, a model which will be briefly introduced later in this document. A dropout layer randomly disables neurons with certain probability. When a neuron is disabled, the training session forces other neurons to be readjusted, and as a consequence, it reduces the over-fitting and allows the model to reach more optimal learning.

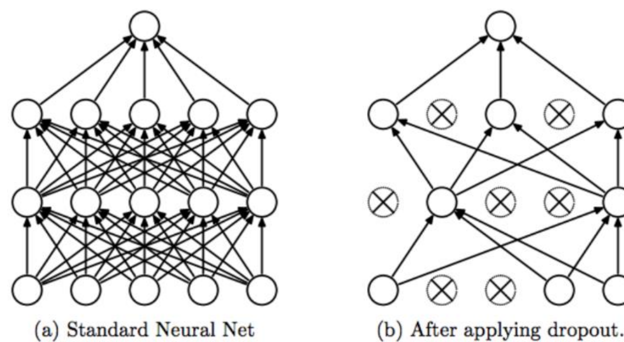

Figure 8 (a) diagram shows a typical fully connected network, while (b) is the result network when a random dropout with certain probability is applied

2.2 Developing tools

Both the machine learning model and the application are completely independent to each other, even the packages and the programming languages used are different. This section sketchily explains which environments were used in the implementation for this project.

2.2.1 Facial expression classifier.

Anaconda is a free and open source distribution of the Python and R programming languages for data science and machine learning applications (large-scale data processing, predictive analytics, scientific computing), that aims to simplify package management and deployment. Anaconda distribution comes with more than 1,000 data packages as well as virtual environment manager, called Anaconda Navigator, so it eliminates the need to learn to install each library independently. Anaconda contains several applications that can be installed from the Navigator, particularly we are going to focus on Jupyter Notebook using Python as programming language.

Jupyter Notebook [15] is a server-client application that allows notebook documents to be edited and run via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet. In addition to executing notebook files, the Jupyter Notebook App has a "Dashboard" (Notebook Dashboard), a "control panel" showing local files and allowing to open notebook documents or shutting down their kernels. One of the most interesting features is that cells can be easily converted to Markdown cells, a markup language that is a superset of HTML which allows to attach images or text.

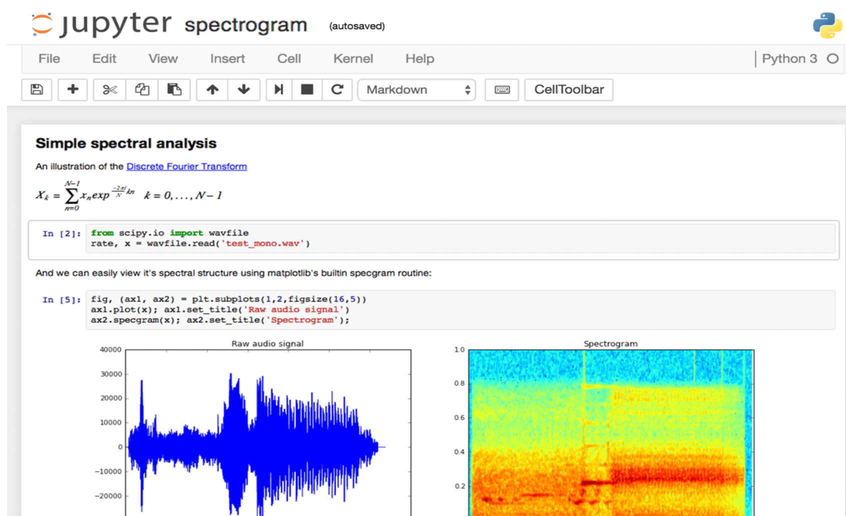

Figure 9 Screenshot of a Jupyter notebook python code.

The following packages were essential in the development of the classifier and were installed through the Anaconda environment manager:

- **TensorFlow** [17], is an open-source machine learning library for research and production. TensorFlow offers an APIs for beginners and experts to develop for desktop, mobile, web, and cloud. TensorFlow is a software library for numerical computation using data-flow graphs. It was originally developed for machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

- **Keras** [18], is a high-level API to build and train deep learning models. It is used for fast prototyping, advanced research, and production. User friendly, modular, compensable and easy to extend are few of the advantages of using Keras. It is a higher abstraction of TensorFlow or Theano.
- **Pandas** [19], is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools.
- **OpenCV** [20], is an open source computer vision and machine learning software library. In this project, it is used for image preprocessing, resizing, as gray scale converter and other image functions.
- **Numpy** [21], is fundamental package for scientific computing with Python. NumPy can also be used as efficient multi-dimensional container of generic data and can apply complex mathematical functions.

2.2.2 Android application

Android Studio [16] was first announced at a Google I/O conference in 2013 and was released to the general public in 2014 after various beta versions. Android Studio is the official IDE for Android application development, based on IntelliJ IDEA. This IDE makes life significantly easier in comparison with non-specialist software. Even though the IDE is free to all developers, is a professional tool with advanced features. The next marks are some of the great points of using this development kit:

- Flexible Gradle-based build system.
- Rich layout editor with support for drag and drop theme editing.
- Debugging and performance, the tool can create a virtual device or connect your own device and using inline debugging to enhance your code walk-throughs.
- Memory and CPU monitor.
- Intuitive code completion and code analysis.

2.3 Resources

2.3.1 Datasets

Convolutional neural network is a supervised learning model, which means, it requires labeled data to be trained (back propagation philosophy). It is important to find decent dataset with a great number of labeled images. Two main datasets are considered to train the classifier: Facial Expression Recognition Challenge held by Kaggle and AffectNet, supported by University of Denver.

Kaggle is a website that posts challenges and competitions which are economically awarded, and thousands of datasets as well as resources are available for all users for free. It is a competitive website where the biggest teams and scientists compete to get the highest accuracy. Particularly, Kaggle provides a past challenge's dataset about facial expression recognition with 35K labeled images. There are 7 different labels and the images are 48x48x1 (grayscale).

The dataset is stored as a csv file with 2 columns, one column for the pixel images and another column for the label.

Label	Amount
Angry	4953
disgust	547
fear	5121
happy	8989
sad	6077
surprise	4002
neutral	6198
Total	35887

Table 1 Kaggle dataset summary.

Figure 10 Kaggle's examples.

Because of the small number of training samples and the size of the images, the model could not reach high accuracy. In addition, well-known architectures cannot be used as baseline because of the size of the images, since these state-of-art designs are prepared for 224 pixels image. Another alternative would be to resize those images, however when resizing method is applied, the outcome images are unclear, and the extra pixels are obtained through extrapolation. This noise can affect the training and over fitting.

As a result of the problems arisen in the preceding dataset, I decided to find an alternative dataset.

AffectNet is the greatest open source facial expression dataset, supported by University of Denver. Here are some of the features that are provided with the dataset:

- Images of the faces
- Location of the faces in the images
- Location of the 68 facial landmarks
- Eleven emotion and non-emotion categorical labels (Neutral, Happiness, Sadness, Surprise, Fear, Disgust, Anger, Contempt, None, Uncertain, No-Face)
- Valence and arousal values of the facial expressions in continuous domain

Label	Amount	UI
Neutral	74874	😐
Happy	134415	😄
Sad	25459	😭
Surprise	14090	😯
Fear	6378	😱
Disgust	3803	😖
Anger	24882	😡
Contempt	3750	😩
None	33088	-
Uncertain	11645	-
Non-Face	82414	-
Total	414798	-

Table 2 AffectNet labels summary

The size of the images goes from around 40 pixels width to high resolution image with over 2000 pixels width. The size of the database is about 122GB, this fact would mean a challenge for the training workflow.

Figure 11 AffectNet's examples.

Since the model will always be accompanied by a facial detector, we will not consider the label *Non-face*. Furthermore, based on the paper written by the annotators, *uncertain, none* labels were assigned when they were not sure about the classification. Therefore these labels and the regarding images will be discarded.

2.3.2 Hardware

It turns out that Kaggle dataset, which is a small dataset of 294 MBs, takes around 265 seconds in average to train all images in a i-5 CPU and 8Gb of RAM with a 100 million parameters CNN, whereas, training AffectNet under the same conditions, it takes approximately 15 hours for the whole dataset.

As a result of the above issue, it is necessary to use advanced GPUs to speed up the process and be able to redesign until the model reaches a minimum accuracy. With the increase of the machine learning activities, numerous companies offer cloud computing services for machine learning purposes. I decided to make use of the FloydHub services. And it is mainly because of the following reasons; all packages are preinstalled, with a wide variety of environments depending on your objectives, it is also simple to use and can be opened with Jupyter notebooks, that allows the developer to edit the script and debug it. The following table represents all services that were used to build the models.

Processing unit	Memory	AffectNet/ epoch	Kaggle dataset/ epoch
GPU Tesla K80, 12GB Memory	61GB RAM, 100 GB SSD	1 hour 10 min	29s
CPU Intel Core i5, 27 GHz	8GB DDR3	15 hours	265s

Table 3 Available hardware with timings measured with the CNN baseline described in Figure 12

The developer enhances the CNN model to obtain better accuracy, nevertheless, it is not strictly compulsory to run over the whole dataset, but reducing the dataset to a smaller size without losing the essence of the dataset can help to speed up the training. Since the training session takes approximately an hour when using GPUs with AffectNet, as well as the fact that an increase on training time means an increase on the economic cost, AffectNet was reduced to 10% of the whole data, namely, it was approximately shrunk from 420k to 42k. The reduction was carried out proportionally according to the number of images per label.

On the other side, regarding the android application, it is tested and executed on an android device, a Samsung galaxy J3, with 2GB RAM, Exynos 7570 Quad and 8GB internal storage.

3 Methodology

As declared above in this document, the considered algorithm to perform the facial expression classification is the current leader in image recognition; the convolutional neural network. At this section, it is detailed all about the designing procedure as well as the architectures that have been considered as references for the best CNN model. It will be compared to several baselines, including famous architectures to measure the competitiveness and performance. After the designing section, the actual preprocessing, training, testing and exporting tasks will be thoroughly explained.

Regarding the android application, it shortly explains how to deploy the TensorFlow model into the mobile operative system as well as how to run the models with an asynchronous task.

3.1 Reference models

The baselines are the point of departure, a solid architecture that will be used as comparison, the main goal is to overcome these reference models and with the aim of achieving better results.

3.1.1 Simple Baseline

The following model is the typical CNN that can be applied successfully to any purpose as far as the input and output layers are correctly adapted. This simple architecture will be the foundation upon the final architecture.

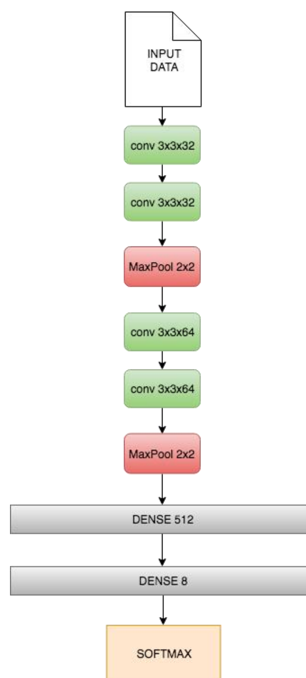

Figure 12 Baseline diagram.

As is depicted in the Figure 12, it is composed by two identical blocks and then coupled to the output block. Each block contains two conv layers in cascade, adding a max pool layer at the end.

The output block contains a fully connected layer of 512 neurons connected to the final activation function, typically a SoftMax(Sigmoid) function. The output size is 8 in the case of AffectNet. A one-hot configuration is used to identify a label with an output. It could be used the same ending neuron for all labels, although that would add sequential dependency between labels, namely 1 is closer to 2 than 5, this is translated into the dataset as; happy is closer to sad than disgust which is not correct.

Kernels are defined as RELU activation functions at the convolutional layers and with no padding or strides. The first convolutional block has 32 channels, whilst in the second block is extended to 64 channels. It is proved that, the increase of the channels promotes to the model with more variability as heading to deeper hidden layers, and thus it provides worthier predictions. To sum up, a progressive channel rise will be considered, in order to build more robust model.

3.1.2 *State of art architecture*

As mentioned before, there are periodically competitions and challenges worldwide. Here it is some of the most relevant architectures that gave a new approach to how CNNs are implemented.

AlexNet [8] was one of the first deep networks to push ImageNet Classification accuracy by a significant stride in comparison to traditional methodologies. It is composed of 5 convolutional layers followed by 3 fully connected layers, as depicted in the figure 13.

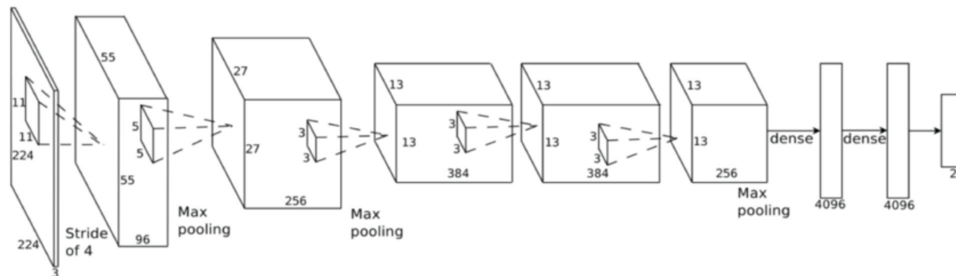

Figure 13 AlexNet architecture.

Although the model was later updated to two steams net giving a new level of accuracy, it was originally split into two streams because the computation time was too expensive, so that the developers decided to train onto 2 GPUs:

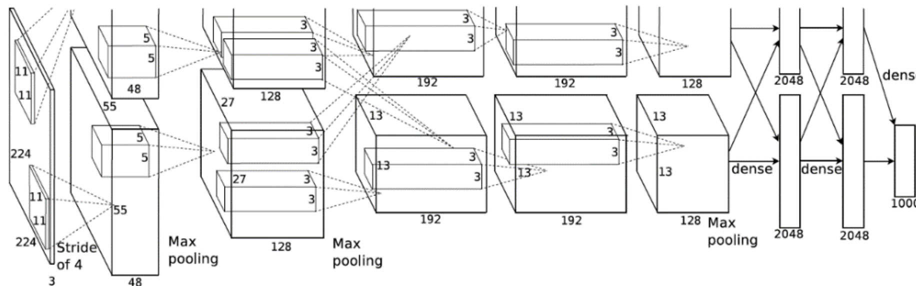

Figure 14 AlexNet with 2 streams architecture.

The advantage of the RELU over sigmoid achieved much faster trainings, additionally Another problem that this architecture solved was reducing the over-fitting by using a dropout layer after every FC layer.

GoogleNet [7] is a 22-layer CNN and winner of ILSVRC 2014 with a top 5 error rate of 6.7%. GoogleNet added a new concept for CNNs; *inception.*

The module basically acts as multiple convolution filters, that are applied to the same input, with some pooling. The results are then concatenated. This allows the model to take advantage of multi-level feature extraction. For instance, it extracts general (5x5) and local (1x1) features at the same time. The following figure depicts the dimensional reduction module.

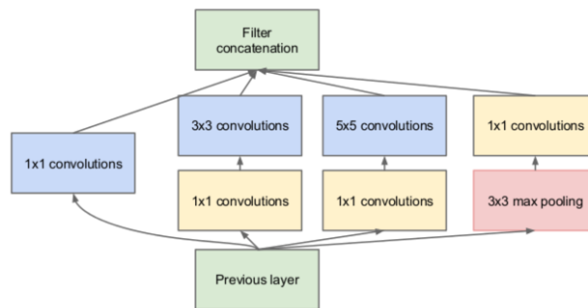

Figure 15 Inception module, dimensional reduction type.

Nowadays there are new versions and more accurate models, however we will not pay attention to all of them, these models will simply be used as inspiration for new models. The next figure represents the most innovative and accurate architectures build so far.

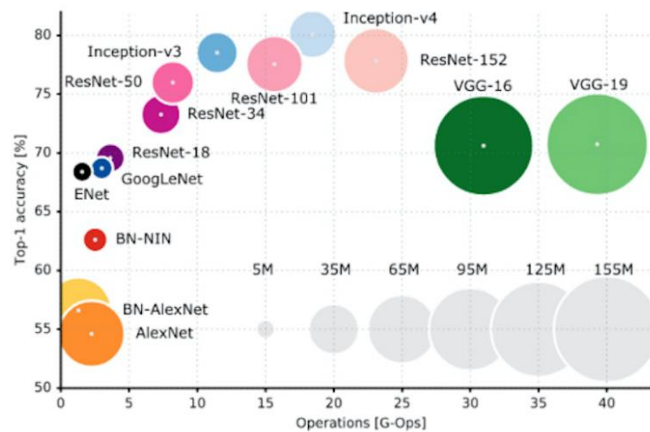

Figure 16 a Accuracy vs computational cost plot, the top most accurate models of recent years .

3.2 CNN design

The main questions at this point are, which parameters or functions should be chosen? which criteria should be followed to create an accurate architecture? The answer to this question will be split in two different topics, tuning parameters and architecture.

3.2.1 Parameters

On one hand, any sort of neural networks has certain parameters or functions that has to be assigned, for instance; the learning rate or the loss function. Worldwide researches have studied about the optimal parameters and is responsibility of the developer to wisely choose those parameters. Based on previous section *2.1.1 Neural Networks*, these are the selected features for the base model:

- Loss function: Categorical Cross-Entropy is a great option for a one hot and multi-category labels.
- Optimizer: Adam seems to have the best performance. It has four parameters that can be tuned, although the authors, as well as Keras API recommends to keep the values as default.
- Metrics: Since the model is a categorical function, the accuracy of the predictions is an essential parameter for evaluating the model, additionally the loss value is visualized while training.

3.2.2 Architecture designs

There are infinite degrees of freedom when a neural network is designed, so that during the developing course, several models are proposed for later training and testing. Finally, the best model in terms of the mentioned metrics will be selected. Leslie N. Smith and Nicholay Topin, researchers in convolutional networks gathered the most meaningful design patterns, taking under consideration those awarded models in order to find a solid criterion to build CNNs. They suggest some of the following criteria in their paper [2]:

- Proliferate Paths: GoogleNet, AlexNet, ResNets have used parallel paths in order to get better variability and so better results.
- Strive for Simplicity: Make it simple as possible
- Increase Symmetry: When more paths are parallelized is better to follow a symmetric pattern.
- Pyramid Shape: AlexNet suggests as well as other models that increasing the channel while the net is getting deeper can achieve good results
- Normalize Layer Inputs: Using Batch normalization and dropout layers allow to the network to reach new learning states. Avoiding to the training session to be stuck.
- Summation Joining: When several paths are paralleled they most likely end up merged. There are different types of merging functions; adders, concatenations, multipliers, averaging…
- Down-sampling Transition: It is recommended to use down sampling layers (pool layers), even with greater stride than 1, in addition to a concatenation merging layer.

Those criteria proposed by Leslie and Nicolay are the fundamentals of the models that are displayed in this report. Starting by the simple baseline as a foundation model.

The first improvement added to the baseline is the batch normalization and the dropout layers. Whenever an input image is ingested in a neural network, the pixel values are normalized to the [0,1] scale. The idea of batch normalization [5] is to additionally normalize the data in a hidden layer. It is proof that this technique gives a great help to reduce covariance shift, helping the activation function to decrease the effect of extreme values. On the other side, Dropout layers helps the net to get deeper learnings, through random neuron is disabling methodologies, explained at *2.1.2 Convolutional neural networks*.

The normalization layer is normally located before the sigmoid function although at the feature extraction it has better performance after a RELU activation function. It is also added before the SoftMax function that yields the output predictions. To make it simple and symmetric the batch normalization is located between the conv layers and the max pool of the baseline for all feature extraction blocks. At the end of each block a dropout is added. As said earlier, the dropout layer helps in learning, the deeper the layer is, the less the layer will learn, the receptive field increases, and the information is lost in the process. Therefore, and in order to aid in the learning process, the first part of the convolutional layer uses a 25% dropping probability while at the last layers, it is increased to 50%. A final dropout layer is added between the fully connected layers.

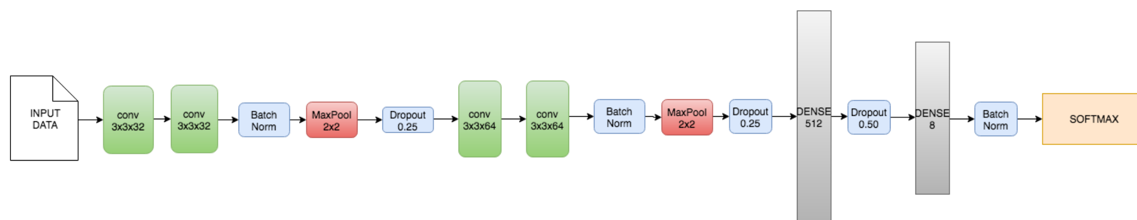

Figure 17 Baseline enhanced, adding batch normalization and dropout layers.

Xudong Cao winner of National data science bowl is a Kaggle competition [1], has some suggestions to build deep networks, multiple network follow this pattern such as ResNets and VGGs. He gives some tips that allows any sort of convolutional network to get great performance. One of the main points is to reduce the feature maps until the size of 5x5, 6x6, 7x7 or similar size, to reduce over fitting in later layers. Additionally, he suggests removing fully connected layers. Moreover, the paper defends the idea of making wider channels while getting deeper in the network as well as Leslie's papers recommends. And finally, it is important to increase the receptive view in the topmost layers in order to get wider view at the beginning of the network, this paper prove that pool layers with greater stride than 1 is a decent manner to do so.

The following model adapts the enhanced baseline to these new principles.

Figure 18 Model adapted based on Xudong Cao's paper (it is identified in this report as X.Cao), the right diagram would be a 2 streams model following Leslie and Nicholay suggested patterns.

As the figure 18 depicts, several layers have been updated to this new model. First of all, the first Pool layer will jump with stride of 2 in order to increase the receptive view in the topmost layers. Since the last layers of the feature extraction segment should be highly reduced to 5x5 kernel size, more pool layers have been added to reduce the size. At Xudong Cao paper is presented a bunch of examples that accomplish his criteria. Following his criteria, the number of channels could easily overcome 1000. In this project a small CNN is demanded, and so it shall be light-weight to be suitable for mobile device. To overcome this issue, the topmost channels start with few channels, e.g. 16, and while the feature map is reduced each layer, the channel is increased. As a result, the network size is balanced since the features maps proportionally reduced and augmented.

Proliferate path are supported by GoogleNet, AlexNet etc.… almost all recent models include certain parallel paths. The next model, depicted at figure 18 (right side), is similar to the two streams AlexNet version. By adding to the previous model (figure 18 left size) a symmetrical path. As the sources suggest, both symmetrical paths are combined with a merging layers (Adder) at the middle point of the feature extraction block.

Finally, the last model and endorsed by Leslie's paper is a fractal architecture. The following diagram extracted from the paper, represents a typical fractal shape.

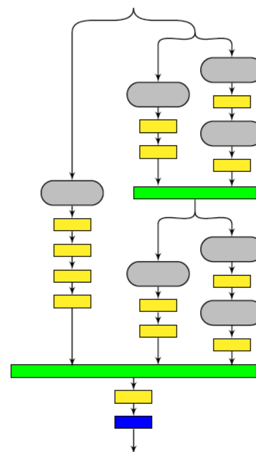

Figure 19 Fractal architecture proposed in the paper written by Leslie and Nicholay. The gray ellipses are convolutional block, this block could be any convolutional architecture, yellow boxes are pooling layers, green boxes are merging layers and at the bottom, the blue box is a SoftMax activation function.

The example shown at the above image has four pooling layers in cascade in the left stream, increasing the receptive field four times in a row. This pool cascade could skip meaningful patterns that a convolutional layer could identify, so that the fractal model replaces these cascade configurations by a more conventional block as shown at figure 19. The Fractal model uses 4 blocks, each block composed by 2 convolutional layers, one pool layer and optionally adding batch normalization and dropout layers. As a fractal architecture is desired, these blocks are replicated in a secondary path reducing symmetrically the number of block and adding concatenation layers. At the end of the network, a maximum merging layer is used to increase the exclusiveness and a fully connected layer with option to add batch normalization and dropout layers as the networks.

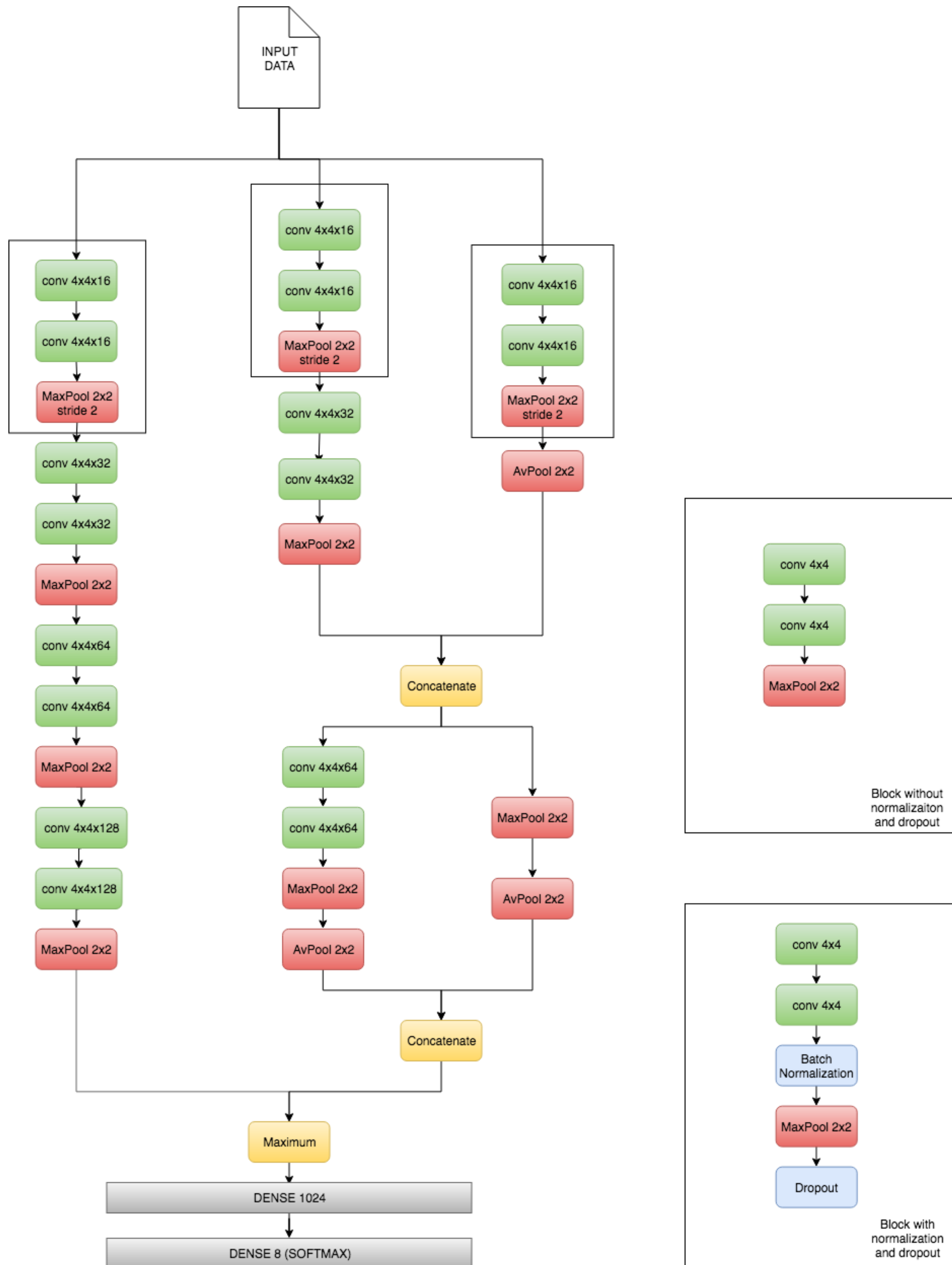

Figure 20 Fractal version of the previous model, the displayed diagram does not use the batch normalization and dropout layers, although it can be simply exchanged with the other block.

3.3 Training workflow

It is time to develop the training script. Classically, every data processing requires three tasks. One task for preparing input data. For this case it will be assumed in two different steps, data ingestion and preprocessing. Afterwards, the processing, for normal machine learning process is equivalent to training. And Finally, evaluating the model.

3.3.1 Input stage

Before caring about preprocessing and adapting the images to the model which will be completed in each batch, it is important to think about the data. As said in the introduction *2.3.1 Datasets*, I decided to prune the dataset removing certain labels and the associate images, as these labels do not provide valuable information to the classifier purpose. After pruning, the resulting dataset becomes lighter. Another issue to be aware of is the variable size of the images, since the model has a constant input size, it is recommendable to resize the images to a fix dimension. After the pruning, the dataset passed from approximately 80GB to around 10GB, easing the training task. Remember that the dataset will be executed in the cloud, the heavier the dataset, the more time to process and thus more cost added to the budget. The storage is also limited, and when more storage is added, more fees will be charged.

The dataset must be divided in two pieces, training set and testing set. It is not a simple task, since the training set will define the network, but on the other hand, the testing set will evaluate how reliable the network is. Therefore, both sets have to be wisely selected and proportional according to the objective. First issue is to decide which percentage of data is destined for training/testing. A 90% for training and 10% for testing was decided. Next is choosing how to split the information. Cross validation is a great idea to find the optimal, it basically splits the data into k subsets, and train on k-1 of those subsets and leave the last subset (or the last fold) as test data. Then it averages the model against each of the folds. However, this technique would imply to train the model K times to find the most accurate split among the K chunks. This is not possible based on my resources, the execution time would require long time and cost. As a consequence, a random sampling was chosen.

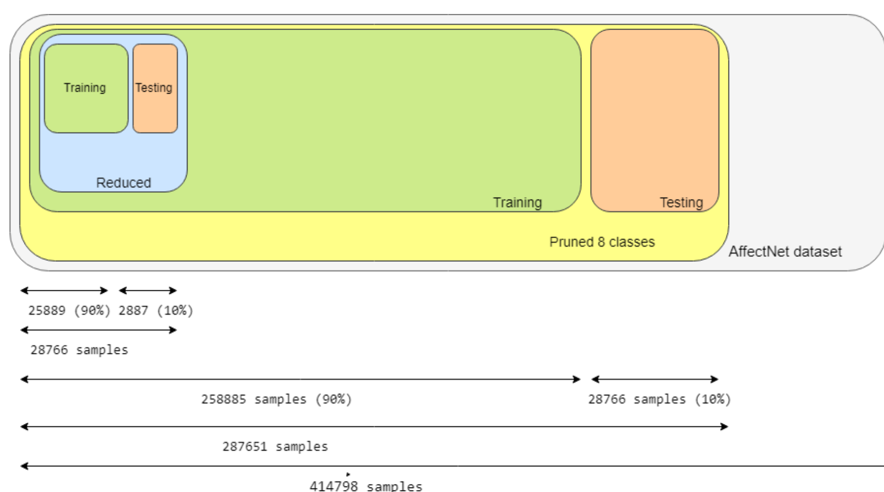

Figure 21 AffectNet's dataset. The dataset is pruned with 8 labels, it is also split in training and testing set. Since it's a big dataset, the designs can be customized with a reduced version. This reduced version is a 10 of the pruned versions.

Once the dataset is pruned, resized and divided into test and train sets, the images need to be adapted to the input layer. There are several preprocessing techniques that can be applied to the images, nevertheless, none of them worked to get higher accuracy. In the attempt to improve the results, several image preprocessing techniques were tried out to the Kaggle dataset. To evaluate the performance of the preprocessing step, the testing procedure uses the same CNN design, the baseline described at the figure 12 with the same configuration.

First of all, a histogram equalization was tried, the aim is to normalize the histogram of each image, so that all images are treated under same histogram conditions, and then extracting only the features related to the facial expression.

The second approach was DoG, difference of Gaussians, this method is a feature enhancement algorithm that involves the subtraction of one blurred version of the original image with a different blurred version of the image, by using different standard deviation for each version. In the end, the resultant image only remains the contours and high contrast parts. It seems useful since the facial expression can be detected by the contour of the face, although, this methodology might remove crucial information that the CNN can get profit.

In third place, Gamma correction is typically used in image processing to equalize the light intensity of the image. Defined with the next expression

$$V_{out} = A\ Vin^{\gamma} \quad \textbf{(4)}$$

Where Vin is a non-negative real input value and γ is a real value sometimes called encoding gamma. When $\gamma < 1$, it generates lighter image, on the other hand $\gamma > 1$, it generates darker image. It is useful technique but finding an optimal gamma value for the whole dataset is complicated and not efficient.

The next table represents the best performances achieved per each preprocessing technique, under the same scenario.

Preprocessing task	Accuracy	Loss
No preprocessing	**0.575**	**1.148**
Equalization	0.5649	1.20
DoG	0.24	1.957
Gamma correction, γ = 0.75	0.549	1.187
Resize to 98x98	0.5291	1.391

Table 4 Training results after applying preprocessing methods to a Kaggle dataset.

Table 4 perfectly reflects the fact that, a preprocessing task prior the feature extraction can only difficult the task of the back propagation. Thanks to the back propagation, features extraction in CNNs is optimized, capable of readjusting the weights of the convolutional masks per execution. In addition, all preprocessing techniques has to be carried out before inserting the image to the network, therefore it takes more time from the deployed device.

There is one necessary preprocessing task to be considered; facial expression does not require color images, so that all images will be converted to gray beforehand. Although, other architectures use 3 channels, such as inception, that is why it is necessary to store the images with 3 channels.

3.3.2 Training

At the training session the model will be configured, compiled and training images will be inserted in batches for optimizing. It is relevant to mention that in most of the cases, the dataset can be loaded in memory, however for AffectNet, it is inefficient to load all 10GB in memory, and most cases it raises memory exceptions. To solve this problem, Keras library offers a way to read the information in runtime, it preprocesses it and insert the images in batches, it is called as batch generator. A batch size is defined by the developer depending on the hardware specifications.

Another important concept is *epoch*, defined as 1 forward propagation and 1 back propagation for the whole dataset. The number of epochs depends on the accuracy/loss improvement, when the model reaches the minimum loss value, both metrics, the accuracy and the loss will remain static. It is a great indicator to know whether the training is optimized to the maximum with the current configuration and architecture. For this case, 10 epochs will be enough.

The next step is to define the model. The model can be described and randomly initialized or can be imported as well. Once the training is completed, it is recommendable to export the model and the weights for future usage.

Once the model is correctly initialized, it is time to build it with the configuration cited in section *3.2.1 Parameters*. Afterwards, the *fitting* function is executed, passing as argument the method that executes the batch generation. This batch generator consists of a loop that runs over the dataset, yielding batches of the data. The implemented method first loads the images in the correct format (Gray scale for the final model), then, the images are normalized by dividing the whole image by 255. Finally, it retrieves the associate labels and yields both the images and the labels into the model. TensorFlow under the hood will perform the forward and back propagation for each single image in the batch. Since the process could take hours or even days, it is important to configure the verbose to monitor the process.

After the training is completed, the model's definition and the optimized weights shall be exported. The history of the training session can also be exported or plotted to check the progression of the loss and accuracy over epoch.

3.3.3 Testing.

Testing subset will evaluate the model performance, by predicting non-trained images. Beforehand, as mentioned earlier in this report, the dataset is split into two subsets. The testing subset remains as 10% of the data.

The testing procedure is really simple, the built algorithm predicts the output as forward propagation, this time no back propagation is accomplished. It will predict all elements in the testing set, and at the end, the demanded metrics reveals how accurate the model is. The most common metric is the accuracy. It basically calculates the ratio between correct classification and misclassification providing a success rate, additionally, the categorical cross entropy used as cost function is shown.

Training session is constantly logging the status at the console, although the accuracy is not a correct representation of the final accuracy unless a piece of the test subset is predicted at the end of each epoch. The training session evaluates the accuracy from the training set and that gives a wrong estimation during the training process. In this case and in order to fix misleading accuracy, a small validation set chosen from the test set are used to give a more reliable accuracy information in runtime. This is important debugging procedure, when long training sessions seem stuck in certain metric value for a relative long time (Several epochs), it is solid indicator to stop training.

3.3.4 Export

A model can be exported in numerous formats, for instance JSON and h5. The structure of the model is stored with JSON format and the weight are exported separately in a h5 file. In order to use the model again, it is obvious that the structure and the weight should be again imported. However, those files cannot be easily executed from an external software, since the classification service requires that the external software builds together both files and translate it into a sequence of operations.

An additional format can be used to avoid the above problem. A Protocol Buffers, or commonly called Protobufs, generates a graph where all operations are structured in nodes. When the file is exported, the weights passes from training variables to constants, that is why the exporting task is called; freeze graph. Now the structure and the frozen weights are arranged with a defined structure of nodes and operations. The protobuf format is essential to be loaded in external software such as a Java machine.

3.4 Android app

Now is time to discuss the real user case of the neural network. The objective is to build a simple android application using TensorFlow libraries to run the exported frozen graph and predict the facial expressions for a useful purpose. The developed app demo is a photo manager; it is a regular gallery although it can filter stored images with the trained model, ideally it could be extended to more filters. The app can perform this filtering in two different ways; all images or one single image.

3.4.1 Libraries

The whole development was carried out in android studio, this IDE and as well as Eclipse are the main environment to develop android software. Android software is compiled with a Java virtual machine, however, the source code of the TensorFlow graph is implemented in C++. So, in order to use a C++ object in a java for android, some extra toolsets shall be included in the project. The next images easily show how is the communication between the SDK (software development kit), which enables developers to create applications for the Android platform, and the NDK (Native development kit), a tool that allows the designer to program in C/C++ for Android devices.

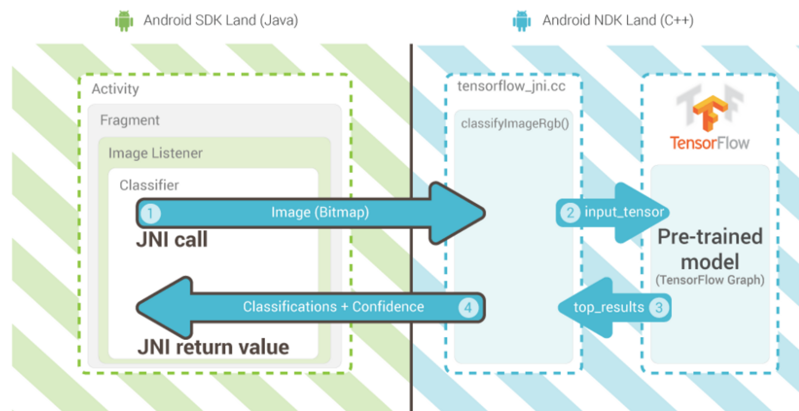

Figure 22 A time diagram that shows a usual prediction task. (1) the java application sends the bitmap image to the NDK, (2) Initialize the model and pass the data to the graph in a tensor format. (3) The graph gives the classifications back to the TensorFlowInterface. (4) Finally, the JNI retrieves the label and the confidence of the classification.

As it is shown at the above figure, the first step is passing the image as a Bitmap to the NDK, using Java Native Interface (JNI) as bridge between SDK and NDK. Then at the core of NDK with the TensorFlow dependencies, the TensorFlow API will convert the image into a tensor and execute the prediction at the exported graph. Next step is to retrieve the recognition results and bring them back to the JNI.

In 2017, an android specific contribution was added to TensorFlow which allows native binaries and java code to be built into a single library (packaged as an AAR file). After that fact, the task was simplified to just two simple dependencies.

- Include the compile *'org.tensorflow:tensorflow-android:+'* dependency in your build.gradle. Gradle is the compilation system that android studio uses.

- Use the *TensorFlowInferenceInterface* to interact with your model.

Before detecting the facial expression, the application needs to guarantee that there is a face in the picture. There are a lot of open source libraries offered for android, which can find a face as well as yielding its coordinates.

- *com.google.android.gms:play-services-vision:9.8.0*

A grid view is a layout control that shall hold the images in the main panel. After several attempts loading the retrieved images from the system directly into the grid. The app generated a memory overload in the testing device. The reason to this fact is that the device cannot keep all images loaded in RAM. The solution to this problem is using asynchronous libraries able to load the images dynamically. In essence only, the images that are visualized in the screen will be loaded, whereas, the rest of the images are waiting until the user updates the grid by scrolling the control. The chosen library is called *Picasso.*

- com.squareup.picasso:picasso:2.71828

3.4.2 User guide

The proposed image manager application will be capable of filtering stored images at the device. The main activity is divided in 3 panels. 80% of the screen is responsible of showing the gallery images, so that we can see the images that the application is working with and in case the

user wants to select a single image. The beneath panel has two buttons; *Filter all* and *back*. *Filter all* button runs the model through the entire gallery and updates the gallery panel showing under each image the available facial expression, represented with *emoji*. Once the filtering is executed, *back's* button will restore the gallery panel to the initial state. The last control is a horizontal scroll with all labels. Each label is associate to a different representative emoji (table 2).

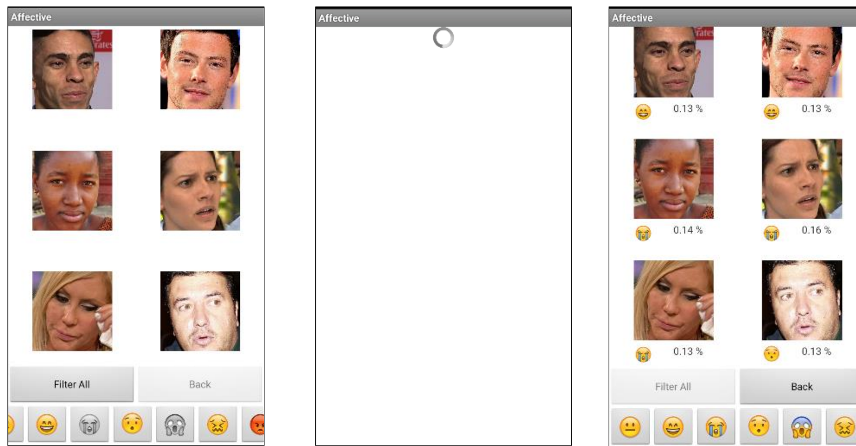

Figure 23 Main activity layout of Affective app. The first image starting from the left is the initial stage, where stored images are visualized. Under the grid view panel, it is placed the buttons and the filters that can be applied (Sad and fear are disabled). At the second picture, the app remains in loading stage until all images are filtered. The last screenshot depicts the images that accomplish the enabled filter, with the regarding classification and a confidence value.

The application filters could filter multiple images or single images. As explained in the previous paragraph, the multi-image classification is executed with the Filter all's button. To recognize a simple image, the user needs to tap on any image, the app automatically creates a new activity with the selected image centered. Afterwards the user could press whether *filter* button or *back* button, similarly as previous functionality. Filtering button classifies the image giving a short description of the classification as well as the probability, whilst the back button will bring the current app state to the initial activity.

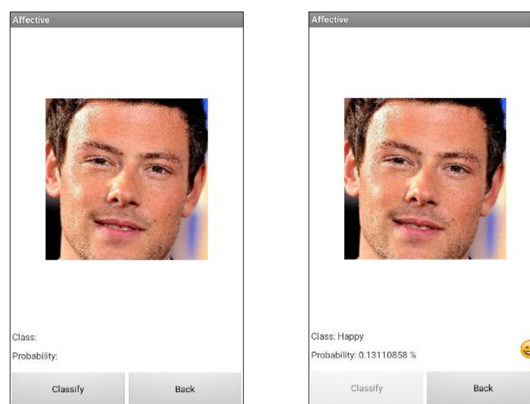

Figure 24 Secondary activity that is raised when a picture is selected from the grid view. The app can perform individual classification.

Images that contains more than one face detection, they will be classified as special label. On the other hand, if there are not detected faces at the picture, the image will not be included when a filter is applied.

3.4.3 Development

Once all libraries are added to the compiled system Gradle, it is time to develop the the application. First, it is important to know the schedule flow chart of an android activity, depicted in the figure 24.

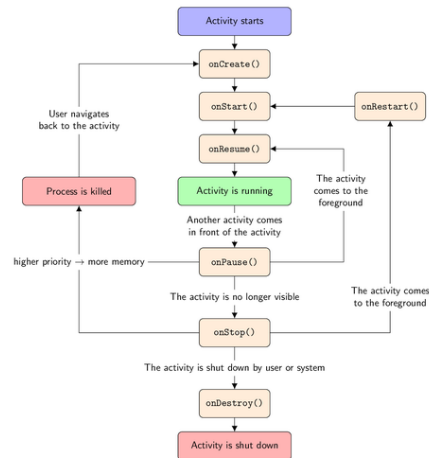

Figure 25 Schedule flow chart of any android activity.

The activity shall inherit from an android preinstalled base Activity class, which offers some abstract methods to be overridden, which are displayed at the figure 25. There are 2 main methods that holds most of the functionalities of the application;

onCreate: When an activity is started for the first time, *onCreate* method will be instanced. It is the right moment to initialize all the dependencies that the activity needs as any constructor would do:

1. Views; it creates the views in the class and link them with the layout controls. The initialized views are the the buttons, and the horizontal filter scroll with its elementary image views. The buttons are assigned to a global event handler method, called OnClickListener. It raises when a button is clicked. Grid view is explained at the third point.
2. Stored image; Retrieve all images from the gallery and store the paths in a list.
3. Grid View; a view that displays the retrieved images, it requires an adapter in order to load the images into the grid view. This adapter loads the images asynchronously as cited at the beginning of this section.
4. TensorFlow classifier; The classifier class is created and initialized with the features of the expected network, which are; image size, input node name, output node name and labels.

OnClickListener: this method is an event handler method that is waiting until the user executes a command from a button. The main commands of the activity are:

1. Filter All; this command creates an asynchronous thread that executes the classification to all retrieved images. While iterating over all images, one by one, this secondary thread finds any face at the picture and in case a picture a face is detected, the method crops, resize and normalize the image in gray scale. Finally, it is converted into an array and inserted in

the model. After that the method retrieved the highest predicted label and updates the current state of the image. This thread repeats the mentioned steps through all images and finally updating the grid view.

2. Tapping single image; when an image at the grid view is selected, a new activity is created focusing only on the selected item. The new activity shall classify individually the selected picture, following the same steps listed in previous point
3. Back; takes the current state of the application to the initial stage.

4 Results

4.1 Classifier

Previous models have been trained and tested to get the optimal classifier. The key point of the classifier is to be small in terms of trainable parameters, the smaller number of parameters, the less time takes the classifier to be trained and predict. It also means lighter export file, relevant when multiple classifiers are imported to the app. The criteria to select the final model also depends on few metrics, accuracy, loss. It is important to remember that the selection process is trained with the reduced version of AffectNet.

The next table depicts the scores of all mentioned model at the methodology section:

CNN	Accuracy	Loss	Number of parameters	Time per prediction(ms) CPU intel i5
Baseline	0.6276	3.0361	102,839,056	11.7
GoogleNet	0.6893	1.3067	23,851,784	14.1
MobileNet	0.4730	1.4600	**4,253,864**	10.0
Baseline enhanced	0.6300	1.3137	102,830,856	15.2
XCao adapted to 224x224	0.6861	0.9185	6,970,728	**7.2**
X.Cao with 2 streams	**0.7540**	**0.6863**	7,509,672	13.1
Fractal with batch Nom. and dropout	0.6923	0.991	26,399,016	28.0

Table 5 Testing results of the all models presented at: 3.2 CNN design. Trained with the reduced version of AffectNet and Tested with 25K samples from the test set split of the whole pruned version (See Figure 21).

The following graph compares the accuracy and the loss per epoch of all models while training with reduced AffectNet.

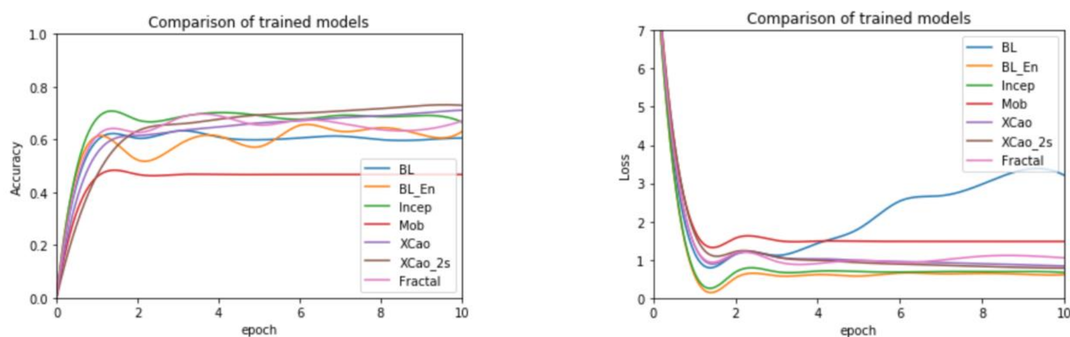

Figure 26 Accuracy and cross-entropy vs epoch comparison, executed with reduced version of AffectNet. This model is validated with a small test set of the reduced version of AffectNet, 2.8k images (See figure 21)

First of all, the baseline was used as foundation of the next designs, whereas the reference models; GoogleNet and MobileNet [9] were considered as objectives. Once said this, the baseline defines the inferior limit of accuracy, derived design should show improvements in terms of accuracy, loss and weight. The simple baseline reached 62.76% of accuracy, trained with the reduced version of AffectNet and tested with the testing set of the whole pruned dataset. It is a simple structure, but because of the thickness of the convolutional layers (32 and 64 channels) the number of weights makes it the heaviest weight among all models. The loss results of the baseline are significantly greater than the remaining models; this fact is caused by overfitting on the training data, thus becoming extremely good at classifying the training data but generalizing poorly and causing the classification of the validation data to become worse. Switching the topic to the target models, GoogleNet obviously achieves better performance than the proposed baseline, it also reduces the amount of parameter plus deeper structure (159 levels), reducing over fitting. On the other hand, MobileNet was suggested because it's a model prepared for mobile devices and so it is the smallest model with 4.2 million parameter and 88 levels deep, nevertheless, MobileNet has the worse performance among all architectures. GoogleNet and MobileNet can also be affected by the fact that they were originally designed to work with 299x299x3 input images and 1000 output classes. Since the dataset is resized to 224 and only 8 target labels, the input and the output of both models need to be modified.

The first enhancement added to the base model is the batch normalization and dropout layers. As it is shown at the table 5, the accuracy of the enhanced baseline is slightly improved and additionally the batch normalization and dropout layers lowered the over fitting. As a consequence of the additional layers, the prediction time was augmented.

The next model supposed a big improvement of the performance, not only in accuracy but also in number of parameters, and consequently, a prediction time reduction. This model based on Xudong Cao's paper allows deeper architecture. The number of channels starts by 16 and end up with 128 channels. Cao also recommends to increase the receptive view at the first input layers, so that a pool layer with stride 2 was added. In addition, the feature map was shrunk until 7x7x128. The accuracy of this model almost reaches the performance achieved by GoogleNet for this specific task, but it has better results in terms of loss value and number of parameters. This architecture significantly overcomes the baseline model in all cases.

The best performance among all models presented at this document for this specific scenario, is the *X.Cao* based model with 2 streams. The model based on AlexNet (*3.1.2 State-Of-Art architecture*) reached 75% of accuracy with 0.68 loss, validated with the 25K test samples. Because of the additional path, the number of parameters of this model was augmented and, as a consequence the prediction time was also increased.

The last model to train and test was the fractal model. this architecture had better performance in terms of accuracy and loss than GoogleNet, it has similar number of parameters and it is the slowest model among all algorithms.

Based on the results of the table 5 and the training plots of the figure 26, I conclude that the 2 stream's model depicted at the right of the figure 18, is the optimal classifier to carry out the classification task.

Once the optimal classifier is selected, the model is trained with the complete version of AffectNet, with around 258K images for training and 28K for testing. The following table depicts the result of this training session:

Epoch	Accuracy	Loss
1	0.7212	0.8129
2	0.7347	0.7653
3	0.7287	0.7446
4	0.7347	0.7307
5	0.7427	0.7180

	Accuracy	Loss
Test set	**0.7510**	**0.7051**

Table 6 X.Cao with 2 streams training results, it is trained with pruned AffectNet dataset. The testing data is the regarding test split of the pruned version of AffectNet (28K images) and the validation data per epoch is a small chunk of this data (10% of the test data; 2.8K images), check Figure 21.

The information shown and the table 7 reveals that the training session with 10 times more images did not affect to the accuracy and the loss. This implies that the reduced AffectNet dataset was correctly selected.

Now we take a look to the actual prediction, a confusion matrix will tell us which categories are more likely to be correctly predicted and which categories has less percentage of accuracy. Since a great portion of the dataset are *happy* or *neutral* emotions, it is obvious that high accuracy values shall correspond with good precision to detect those categories.

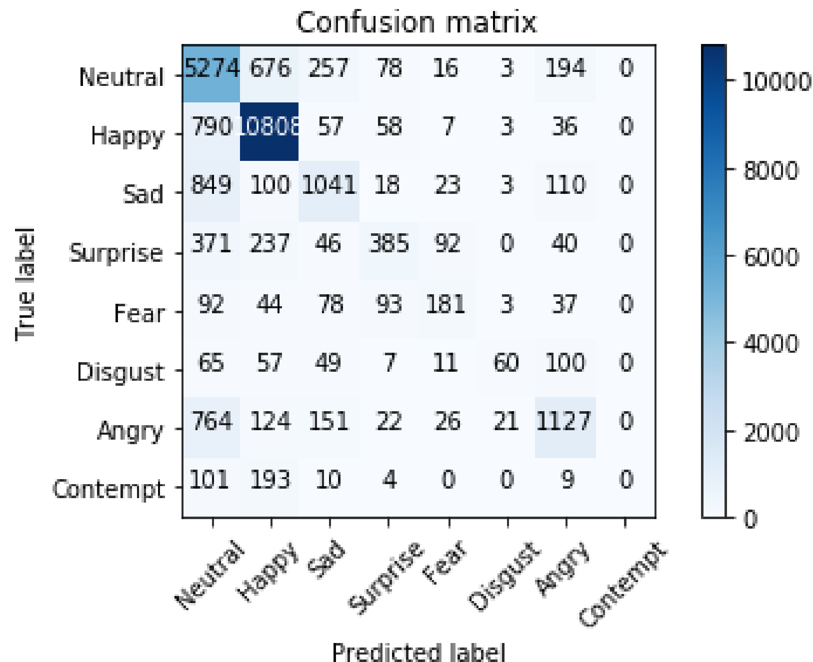

Figure 27 Confusion matrix of 25K testing images by using 2 streams X.Cao model.

Label	Correct classifications	Misclassifications	Accuracy (%)
Neutral	5274	1224	81
Happy	10808	951	91
Sad	1041	1103	48
Surprise	385	786	32
Fear	181	347	34
Disgust	60	289	17
Anger	1127	1108	50
Contempt	0	317	0

Table 7 Accuracy per label.

These tables expose an important issue, the final accuracy does not represent the accuracy per each category. The accuracy of each label is then highly correlated to the number of images trained per category, namely, happy and neutral are the most populated pictures among the dataset with 134k and 74.8k, these categories achieved a high accuracy of 91% and 81%. On the other hand, fear and disgust with 6.3k and 3.8k images obtained lower accuracy than 50%. Although, the most concerning point comes with *Contempt's* category results. It is the scarcest category among the existing, it is similar number as disgust, but zero times was predicted throughout a 25k testing images. This behavior is not particular of this model but it occurs for all models presented in this paper.

4.2 App performance

The android application needs to be analyzed in the time domain. When a classification task is carried out, this task can be summarized with five steps; retrieving the bitmap for each image, detecting the fase and in case there is one face detected, the cropped picture is then resized to 224x224. Afterwards the images is then converted into a float tensor with gray format (1 channel) and finally and the most important, the tensor is passed to the exported graph, all images are all ran in a secondary thread, meanwhile the main thread is waiting until the classification is completed.

In order to measure the timing of cited steps, 25 images have been selected and loaded into the device from the test data. Among those images, 20 images contain one face, 3 contains no face and 2 contain more than one face.

Steps	Average Time (ms)	Standard deviation (ms)
Retrieving Bitmap	57.88	140.5
Face detection	**1601.56**	**3846.5**
Resizing	1.5	2.7
Gray conversion	126.8	33.5
Classifying	384.1	315.35
Total	2171,9	4338,6

Table 8 Average time and standard deviation taken per each step to classify an image in Affective

Considering the previous table, Face detection and classification takes the biggest part of the processing time. The rest of the preprocessing steps: retrieving bitmap, resizing and gray conversion takes less than 200 miliseconds. In addition the deviation is low regarding the detecting steps.

Focusing on the detection steps, it is easy to note that face detection step (using Android library) spends a wide spectrum of time. Normally operates in 500ms although, in certain images it can be extended for couple seconds in a single image. This fact increases the standard deviation. About the facial expresion classifier exported, it takes approximatelly 300ms, however same issue occurs for this classifier, it is constant in time but for certain images, the classifiers has difficulties or *get stuck* by processing a single image.

After several attempts, the application running on the testing device, mentioned in *2.3.2 Hardware*, processing takes around 2 seconds per image in average. This implies that 30 images can take 1 minute to be processed. This time is too long time for a simple classification task and do not fulfil the mínimum requirement presented at the *1. Objectives* section.

5 Cost summary

Online services are being used for training models with powerful GPUs, which are specialized in machine learning. FloydHub is a Platform-as-a-Service for training and deploying your deep learning models in the cloud. FloydHub service was only used while training with AffectNet dataset, since it takes a lot of resources and is time consuming. On the other hand, Kaggle facial expression dataset could be ingested and trained in a domestic CPU since it takes relatively short period of time compared to AffectNet. The next table summarizes the services used from FloydHub.

Item description	Duration of service	Quantity	Unit Price €	Extended price €
Base subscription with 100 GB of storage	1 month	2	9	18
Powerups GPUs	10 hours	3	12	36
			Subtotal –	54
			Total –	54

As a result of the AffectNet reduction, the model under test was fitted with a 10% of the real data, which was translated in a faster training and so less economical cost per training session. The designing took around 2 months. The optimization of the networks consumed 30 hours of GPUs, of which 20 hours were assigned for developing with reduced AffectNet and 10 hours for training the resulting model.

6 Conclusion

In light of the results presented, the proposed CNN model shows how a light model with barely 7.5 million trainable parameters can compete against reputable CNN architectures. Thanks to the machine learning society and researchers, non-experts can easily build a decent convolutional neural network, although, the process and the criteria to be followed is not clear yet. At the classifier results section, certain categories were not well predicted, and in the case of *contempt* category, it was never predicted by the algorithm even though the total accuracy of the model was around 75% for the optimal model. Therefore, it would be interesting to work with an additional metric that offers the accuracy depending on the sample size, or a metric per category to identify misclassifications for certain cases. For this case, one possible solution would be to adding new relevant information to the input data to help the classifier, such as, facial key points or facial emotion intensity. That information is also provided at the dataset metadata.

There are certain ways to improve the final model. First improvement would be using data augmentation approach. Data augmentation expands the dataset by applying a random modification to random images in the batch, such as flipping, inclining, shifting, Gaussian noise, resizing etc.… It helps the model to "*get used to*" unexpected images and so reducing the over fitting. AffectNet is already full of dataset and since my resources are limited, I preferred not to enlarge it more. Another important topic is the preprocessing. There might be a way of preprocessing that helps the CNN's feature extraction to get higher accuracy.

In terms of the deployment, the demo ended up being too slow for regular tasks, users would not tolerate to wait some minutes just for a simple classification task. Here is where servers become essential. TensorFlow models can be deployed in servers and even run in browsers with new JavaScript libraries which are developed by Google. Being on the internet, and sharing models, allows the network to learn continuously and readjust the neural network for better service. In addition, servers would incorporate more models (more capacity), not only facial images but alse other type of labels or even different type of data such as speech.

One of the most important matters about machine learning and its deployment is the independence between the model and its service, as far as they are compatible. Both projects are totally isolated and the classifier could be reused, replaced or updated to a new version regardless the application or vice versa.

7 References

[1] Xudong Cao. *A practical theory for designing very deep convolutional neural networks*

[2] Leslie N. Smith, Nicholay Topin. 2016. *DEEP CONVOLUTIONAL NEURAL NETWORK DESIGN PATTERNS*

[3] S.H.Hasanpou, M.Rouhani, M.Fayyaz, M.Sabokrou. 2018. *LET'S KEEP IT SIMPLE, USING SIMPLE ARCHITECTURES TO OUTPERFORM DEEPER AND MORE COMPLEX ARCHITECTURES*

[4] Diederik P. Kingma. Jimmy Lei Ba. 2017. *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION*

[5] Sergey Ioffe, Christian Szegedy. 2015. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*.

[6] N.Srivastava, G.Hinton, A.Krizhevsky, I.Sutskever, R.Salakhutdinov. 2014. *A Simple Way to Prevent Neural Networks from Overfitting*.

[7] Szegedy, Liu, Jia, Sermanet, Reed, Anguelov, Erhan, Vanhoucke, Rabinovich. 2014. *Going deeper with convolutions.*

[8] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. 2012. I*mageNet Classification with Deep Convolutional Neural Networks*

[9] Howard. Zhu, Chen, Kalenichenko, Wang, Weyand, Andreetto, Adam. 2017. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision*

[10] Arden Dertat. Applied Deep Learning [https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks 584bc134c1e2] accessed August, 3 2018

[11] Michael Nielsen. Using neural nets to recognize handwritten digits [http://neuralnetworksanddeeplearning.com/chap1.html] accessed August, 3 2018

[12] Adit Deshpande. A Beginner's Guide To Understanding Convolutional Neural Networks [https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/] accessed August, 3 2018

[13] Wikipedia. Artificial neural network. [https://en.wikipedia.org/wiki/Artificial_neural_network] accessed August, 3 2018

[14] Sebastian Ruder. An overview of gradient descent optimization algorithms [http://ruder.io/optimizing-gradient-descent/] accessed August, 3 2018

[15] Jupyter Notebook documentation. 2015. [http://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html] accessed August, 5 2018

[16] Android's documentation [http://androiddoc.qiniudn.com/tools/studio/index.html] accessed August, 6 2018

[17] Tensorflow API and documentation [https://www.tensorflow.org/] accessed August, 6 2018

[18] Keras API and documentation [https://keras.io/] accessed August, 6 2018

[19] Pandas API and documentation [https://pandas.pydata.org/] August, 14 2018

[20] OpenCV API and documentation [https://opencv.org/] August, 14 2018

[20] Numpy API and documentation
[https://numpy.org/] August, 14 2018

[21] TensorFlow Documentation with some code examples.
[https://www.tensorflow.org/mobile/mobile_intro] Accessed August, 1 2018

[22] Dan Jarvis. Using a Pre-Trained TensorFlow Model on Android
[https://medium.com/capital-one-developers/using-a-pre-trained-tensorflow-model-on-android-e747831a3d6] accessed August, 1 2018

[23] Jay Alammar. Supercharging Android Apps With TensorFlow
[https://jalammar.github.io/Supercharging-android-apps-using-tensorflow/] Accessed August, 1 2018