PEDRAM GHAZI

SMILE RECOGNITION IMPLEMENTATION ON EMBEDDED PLATFORMS

Master of Science Thesis

Examiner: Assoc. Prof. Heikki Huttunen
Examiner and topic of this thesis approved on 8 August 2018

ABSTRACT

PEDRAM GHAZI: Smile Recognition Implementation on Embedded Platforms
Tampere University of technology
Master of Science Thesis, 48 pages
August 2018
Master's Degree Programme in Information Technology
Major: Data Engineering and Machine Learning
Examiner: Associate Professor Heikki Huttunen

Keywords: Smile recognition, Smile detection, Facial expression, Machine Learning, Neural Network, Deep Learning, Multithreading, Asynchronous communication, Parallelization, Embedded implementation, Nvidia Jetson

In this work, our focus is on the real-time development of a smile recognition system on low resource computational devices utilizing deep learning algorithms which could be simply further developed to address issues in mentioned areas.

We have primarily used the Looking at People (LAP) dataset for training and testing various neural network architectures. Images in this dataset have been pre-processed at first by acts of cropping around the facial area and face alignment. Then six pre-trained deep learning network architectures were finetuned for this purpose.

The fine-tuned models were deployed on Nvidia's embedded platform and we were employing an asynchronous design to provide smoother frame rate through parallelization and multithreading. Accuracy and speed of these models were retrieved letting us compare them to each other and choose the most suitable ones for this task. Our research shows that modern low complexity architectures could almost reach the older or bulkier ones' performance.

PREFACE

"The face is the mirror of the mind, and eyes without speaking confess the secrets of the heart."
— St. Jerome

It is my passion to implement systems that are applicable to real-world issues. The need for such systems that can recognize facial expressions has emerged during the past decade and it is growing rapidly due to its wide applications and fast advancements in artificial intelligence. In this thesis, once again I followed my desired path and tried to give solutions to the recognition of facial expression within low resource environments.

The truth is that I could not have achieved the current level of success without the support of many people. First, I want to express my deepest gratitude to Assoc. Prof. Heikki Huttunen, who always supported me with his inspiring thoughts and delightful personality. And secondly, my beloved parents, I could never thank you enough for guiding me in the right direction. Without you, I would have never made it this far. Finally, I want to thank my dear friends for our unforgettable moments. Thank you all for your unwavering support.

Tampere, 2.07.2018

Pedram Ghazi

CONTENTS

1. INTRODUCTION 1
  1.1 Background & problem overview 1
  1.2 Related works & applications 2
  1.3 Contributions 4
2. THEORY 6
  2.1 Convolutional neural network 6
  2.2 Layers in CNNs 9
  2.3 Major building blocks of CNNs 16
  2.4 Mostly used neural network architectures 20
3. TOOLS AND METHODS 27
  3.1 Frameworks and APIs 27
  3.2 Image pre-processing (geometrical transformations) 33
  3.3 System level architecture 35
4. EXPERIMENTS AND RESULTS 37
  4.1 Datasets 37
  4.2 Setup 38
  4.3 Evaluation 39
  4.4 Results 39
5. CONCLUSIONS AND POTENTIAL FUTURE WORK 42
REFERENCES 44

LIST OF FIGURES

Figure 1. Sample outcome from the implemented syste 2
Figure 2. 3-D tensor for an RGB image. 7
Figure 3. Visual comparison between convolution and cross-correlation operations 8
Figure 4. Left: A regular NN. Right: A CNN 9
Figure 5. Left: An image tensor and a set of neurons in the convolution layer. Right: Performing a dot product operation by the neurons 10
Figure 6. Sample convolution operation 11
Figure 7. Left: Passing the input tensor through pooling layer. Right: Downsampling example. 14
Figure 8. Sigmoid activation function 14
Figure 9. Convolving example 16
Figure 10. Up: Depthwise convolution. Down: Pointwise convolution 17
Figure 11. Inception module 18
Figure 12. Left: ResNet block with 2 layers. Right: ResNet block with 3 layers 19
Figure 13. VGG-16 structure 20
Figure 14. Inception network and its modules 23
Figure 15. Left: Resnet building block. Right: Connections in the residual block 24
Figure 16. Inception-resnet V2 architecture 25
Figure 17. Xception module 26
Figure 18. Xception architecture 26
Figure 19. Data flow graph. 29
Figure 20. Use of Keras framework among researchers 32
Figure 21. Pairs of symmetric landmarks 34
Figure 22. Left: Aligning images without restrictions. Right: with restrictions. 34
Figure 23. Software architecture diagram for the smile detector. 35
Figure 24. Sequence diagram for the system architecture. 36
Figure 25. ACC and AUC measures for trained models 41

LIST OF TABLES

Table 1. Structure of MobileNet 21
Table 2. Ranks in TensorFlow 28
Table 3. Shapes in TensorFlow 28
Table 4. Running speed and network size of different networks 39
Table 5. Comparing a Mobilenet's variant with Smile-Net network 40

LIST OF SYMBOLS AND ABBREVIATIONS

ACC: Accuracy

ANN: Artificial Neural Networks

AUC: Area Under Curve

CNN, ConvNet: Convolutional Neural Network

FC: Fully connected

FN: False Negative

FP: False Positive

FPO: Floating Point Operations

FPS: Frame Per Second

ML: Machine Learning

NN: Neural Network

ROC: Receiver Operating Characteristic

TF: TensorFlow

TN: True Negative

TP: True Positive

1. INTRODUCTION

1.1 Background & problem overview

An assessment of emotions in a person can be obtained from observing his//her facial expressions, so recognition of them would come handy and are applicable in many areas like making safer assisting cars, emotion detection during interviews, diagnosing psychological problems and disorders, consumer behavior research, market research, etc. Therefore, detecting facial expression from real-time image data streams has become one favored research topic in computer vision during few past years. Many recent commercial technologies including Google Cloud Vision API[1], Amazon Rekognition[2], and Microsoft Azure Face API[3] are primarily using cloud computing and hence unsuitable in cases that should provide low latency and are meant to be real time. Further, like other image analysis tasks changes in direction, brightness, illumination, and other imaging conditions would have effects on the results and makes it more challenging and difficult to achieve desired results in real-time embedded systems.

Nowadays, deep convolutional neural networks have become a major tool in many machine learning areas like computer vision and image analysis [1]. Since Convolutional Neural Networks(CNN) emerge, most of the scientists have been working to increase the accuracy of deep neural network architectures and it is only recently that researchers have partially directed their focus towards more efficient and faster implementations [2, 3] that could be executed on embedded platforms with limited resources. Furthermore, most research focus on improving efficiency of individual parts of different systems and only a small number of them study through system level implementations.

This thesis explains a system level architecture employing deep learning algorithms for smile recognition, which has several applications such as customer satisfaction measurement [4] and clinical quantification of emotional state in patients [5]. This system also could be considered as an ideal case for designing a system level architecture. In this system, we would have several modules with various computational complexities such as frame grabber, face locator, smile recognizer, and a module for drawing and showing results. However, deep learning models might not keep up with other modules' in terms

---

[1] Google Cloud Vision API. Available: https://cloud.google.com/vision/.
[2] Amazon Rekognition, Amazon Rekognition announces real-time face recognition, text in image recognition, and improved face detection, https://aws.amazon.com/rekognition/,
[3] Microsoft Azure, Face API, https://docs.microsoft.com/enus/azure/cognitive-services/face/

of frame rate processing, the outcome from the system should still seem to be real-time while depicting it. A sample output of the implemented system is illustrated in Figure 1.

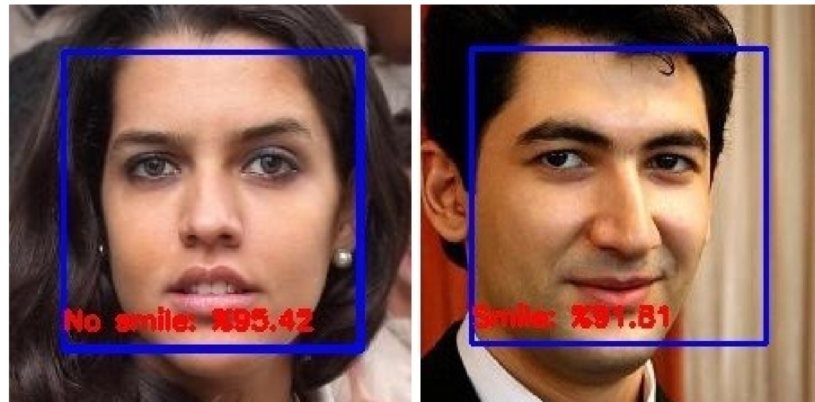


Figure 1. *Sample outcome from the implemented system. A none-smiling face with 95% recognition accuracy and a smiling face with 92% recognition accuracy. [6]*

1.2 Related works & applications

1.2.1 Related works

First attempts for robust and fast detection of faces were done in early years of this century [7]. And yet many real-time face detection algorithms [8-10] are using the OpenCV library which is employing the Viola-Jones algorithm [7]. By the rise of deep learning, methods have been reported for detecting and recognizing faces with much better accuracy and in the past few years, various algorithms based on CNNs have been developed for face detection [11-15] and recognition [16-19].

First implementations of facial recognition systems were based on locating several landmark points, e.g. eyes, ears, nose, lips, etc. Then, the relative location of these points was utilized for classification problems, for example, categorizing smiling and non-smiling faces. Recent deep learning algorithms give unprocessed pixel data from face images to a deep CNN as input. This representation from a cropped face area carries more information with itself since it is retrieving information from the whole face region.

Many works around detection of age, gender and mood employing deep CNNs have been done, for example [20], smile detection for image data of young children [21], and recently a neural network architecture known as SmileNet [22] for smile detection. Another multipurpose framework HyperFace uses deep learning and is able to offer landmarks localization, face detection, and gender detection all at the same time [23].

Even so, we are facing the challenge of achieving real-time execution speed for continuous data where different components of our system have distinctive execution times. Sim-

ultaneously, this system should still maintain working at an acceptable accuracy in comparison with post-processing systems. Traditionally Haar feature-based cascade classifiers [7] is used for face and especially object detection. Besides those kinds of detectors, there are also several customized deep network structures only for face detection purposes which can run on embedded platforms smoothly. FaceBoxes is an example for detecting faces with high accuracy executing real-time using CPU [24], whilst LCDet provides an 8-bit fixed-point TensorFlow detector running on top of embedded platforms [25] and OpenFace introduces a general-purpose library for face recognition tasks in mobile applications [26].

1.2.2 Applications

Facial expression recognition has been already employed by several organizations to measure users' feelings about their services in both digital and real worlds. In real-world scenarios, customers interact with products and services in different environments and these interactions are still challenging for assessing their responses automatically. Retrieving emotions out of facial expressions making use of ML techniques could be a feasible choice for automatically measuring customers' engagement with their services. Here, we explain and give examples how emotion recognition -which is the general case of smile recognition- can be used to address various real-world use cases competently.

Healthcare

From the health-care point of view, the inference of this technology is that machine learning would be an aid for doctors to trace their patients' wellbeing by quantizing emotional state of patients [5]. Other possible applications in medicine include: help monitoring for elderly people, retrieving patient's feelings about treatment, helping in perceiving facial expressions from children with autism [27].

Marketing

Before this, verbal practices like surveys was used to do researches in marketing for finding the customer's needs. Although, these practices expect that customers are able to express their desires verbally and the those relate to their upcoming activities which might not always be true.

Nowadays, employing behavioral practices are employed as another method in the market research industry for observing customers' response towards a product or service. We believe these techniques are more objective than verbal ones. Video streams of customers could be employed while interacting with the service and then put into further inspection for studying their responses and feelings. Facial expression detection systems can automatically measure states of facial emotions and interpret them into meaningful and sensible results [4].

Law

There are situations we need to estimate emotions level in people, for example, a system could be employed as an integration tool with suspects of criminal activities or candidates for sensitive sector employment. Police departments generally are interested in using such systems to interrogate suspects and also analyze employees. An interviewee ‘s communication is vulnerable to many parameters like language interpretation, cognitive biases, and the lying context in between. AI might come handy here, it would be able to assess interviewee’s facial expressions to understand his/her emotions and based on them determine their personality characteristics, moods, and many other needed information.

This system can also help recruiters. It can evaluate the overall confidence level of an interviewee which enables employers to decide if the interviewee is well-suited for the current job position or not. Such a system can be also used to find out whether or not the person is answering all questions honestly by screening changes in his/her emotions during the interview. This system can also help with estimating employees’ morale by checking and scanning their interactions.

Monitoring

Globally automobile manufacturers have made the production of safer and personalized cars a focus of attention. For making a car with smarter features, they make use of AI in perceiving our emotions. Detection of facial expressions let these smart cars notify passengers whenever the driver is feeling sleepy or suspicion of driving under the influence.

Errors in driving cause around 95% of most fatal road accidents in the US[28]. Facial expression recognition would be able to detect slight changes in micro facial expressions that come before sleepiness, warn the driver and ask him/her to pull over for a short nap or coffee.

1.3 Contributions

In this thesis we are offering an architecture implemented on top of low resource computational environments like NVidia Jetson embedded platform and this architecture would be suitable for achieving real-time execution of deep learning methods in such systems. We deployed a smile recognition system as a sample use case and tested it employing thirteen deep CNN architectures using images from three well-known image datasets. To be specific, we studied their performances using three different computation resources: a desktop CPU, a desktop GPU, and Nvidia Jetson, and then we analyzed the tradeoff between their speed and accuracy. It is obvious that different network structures require significantly different computational power, but consequently, we concluded from our examinations that accuracies for many models are relatively close to each other. On one hand, in the image datasets we were working with the ground truth could be corrupted

since categorizing distinctive levels of smiling into just two 'smiling' and 'non-smiling' categories are difficult, and this might raise some questions about how significant they affect the observed accuracies. On the other hand, someone would hold that more recent deep CNN architectures like Mobilenets surpass the earlier ones (like VGG) in speed/accuracy tradeoff, and they should not be used for real-time purposes.

We implemented an asynchronous system capable of integrating other detection modules easily. The suggested system level design on top of NVidia Jetson let us reach the execution time of 27.3 fps in average which could be considered as real-time performance.

The proposed method and implementation would offer almost three times faster processing time than the mentioned modern smile detectors. In addition, our system's accuracy is so similar to that of the modern approaches.

2. THEORY

2.1 Convolutional neural network

Convolutional neural networks (ConvNet or CNN) are a part of Machine Learning (ML) and more specifically a subset of deep, feed-forward artificial neural networks. They are majorly used in computer vision tasks, natural language processing, and recommender systems.

ConvNets employ various multilayer perceptrons, besides they are recognized as space invariant or shift invariant ANNs because of their architecture sharing weights and supporting translation invariance. The idea behind the CNNs originally came from biological processes in which the associations and links between neurons are attempting to mimic the structure of living organisms' visual cortex. CNNs are designed to use fairly small pre-processing in comparison with other methods developed for image classification. This is the same as the fact that in neural networks, filters are learned and eventually tend to their optimum while in older methods they were hand-engineered. [29]

ConvNets are similar to ordinary ANNs in the manner that they are made from neurons which are able to learn and change for new weights and biases. Every neuron accepts some inputs then performs a dot product operation on them and after that, a non-linearity module may come optionally. This kind of networks defines differentiable score functions in such a way that raw images, in numerical form, enter as inputs from one end and the function generate class scores at the other end. In CNNs There exists a loss function in their last FC layer [29]. As mentioned earlier CNNs structure assume that the inputs for the model are images. This fact lets us encode particular configurations into their structure and make more efficient implementation of the forward functions, then, as a result, reduce the number of learnable parameters enormously.

2.1.1 Images as three-dimensional tensors

CNNs deal with images as tensors, this means they consider images as matrices of numerical values that might have more than two dimensions. One number is also called a scalar a and list of numbers is called a vector. A matrix consists of several vectors, with the same length, concatenated in the second dimension which makes a grid of scalars. We assume that a scalar has zero dimensionality, a vector has dimensionality one, a matrix is a plane and has two dimensionalities, and consequently, a bunch of matrices makes a cube which is three-dimensional. The fourth dimension could be defined when each element of these matrices is a stack of matrices. This way of nesting can continue infinitely making any desired number of dimensions which we are not capable of visualizing. CNNs mostly

deal with three-order (3-Dimensional) tensors and in Figure 2 you can see three different channels of a color image are presented as the depth dimension.

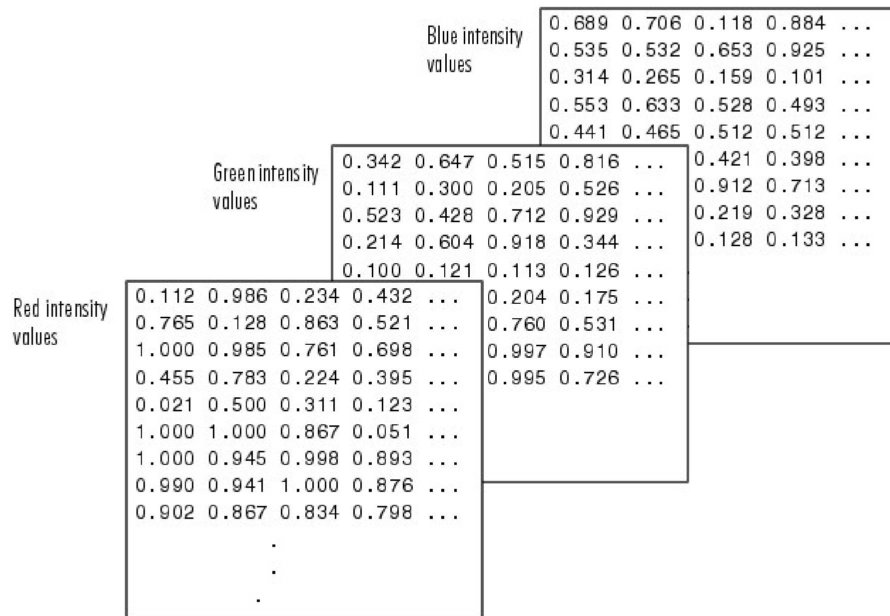


Figure 2. *3-D tensor for an RGB image.*

Images have height, width, and depth. It is easy to understand their width and height and the depth means layers or channels, i.e. in RGB(Red-Green-Blue) case images are consisting of three channels. Through convolution, every channel(layer) makes a bunch of feature maps (clarified in CONVOLUTION section), which exist in the third dimension.

2.1.2 Definition of convolution operation

Convolution is a mathematical operation between two functions and gives another function which outputs the integral from the pointwise multiplication of the two functions where one of the two first functions is translated. In another word, it is the integral of two overlapping functions when one passes over the other one. [29]

$$f * g = \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau = \int_{-\infty}^{\infty} g(\tau)\, f(t-\tau)\, d\tau$$

Mathematically convolution is considered close to cross-correlation. Convolution operation on real-valued signals can be performed by flipping one of the sequences before performing the correlating. A visualization of this is illustrated in Figure 3.

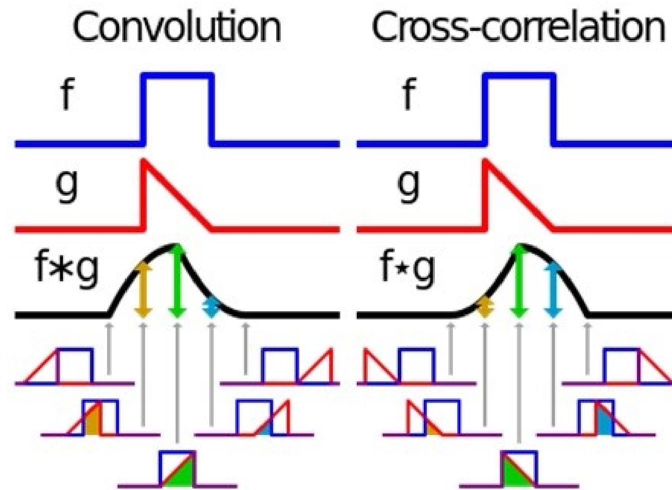


Figure 3. Illustration of comparison between convolution and cross-correlation operations. [30]

This operation has applications in many areas of science including probabilities and statistics, image and signal processing, computer vision, natural language processing, etc.

In image analysis tasks the input image being analyzed acts as the underlying function which is static and the second function which passes over the first one is known as the kernel or filter. This enables us to retrieve a particular feature or signal from the input image.

It should be denoted that in convolutional networks many different kernels are passed over an individual image and each one of them is meant to retrieve a different feature. In the first few layers, someone could consider passing kernels associating with vertical, horizontal, diagonal lines to get a view from edges in the input image.

CNNs get a map of places that a specific feature occurs by applying kernels on various parts of the image's feature space. By learning different portions of a feature space, convolutional nets allow for easily scalable and robust feature engineering.[29]

2.1.3 Structure

Regular NNs receive an individual vector as input and process it through some hidden layers [29]. In every hidden layer, we would have artificial neurons, where each of them is connected to all other ones in the former layer and every neuron in any layer would function independently. They are not able to scale well to full large images. For example, consider a 224×224×3 image, it would result in neurons with 224*224*3 = 150,528 number of weights. This kind of full connectivity between subsequent layers is a waste of resources and this enormous number of parameters would most probably cause overfitting.

CNNs are made of an input and an output layer, and a number of hidden layers in between. Typically, hidden layers in a CNN include convolutional layers, pooling layers, fully connected layers, and normalization layers. They take this into account that inputs are images, therefore their architecture is reformed in a sensible way to serve tasks related to images. In general, layers in a CNN are consisting of neurons organized in three dimensions: width, height, depth. And we will explain soon that neurons in one layer will be connected just to a small area of the previous layer unlike the full connected manner in normal NNs[29]. The structure of regular NNs and CNNs is shown in Figure 4.

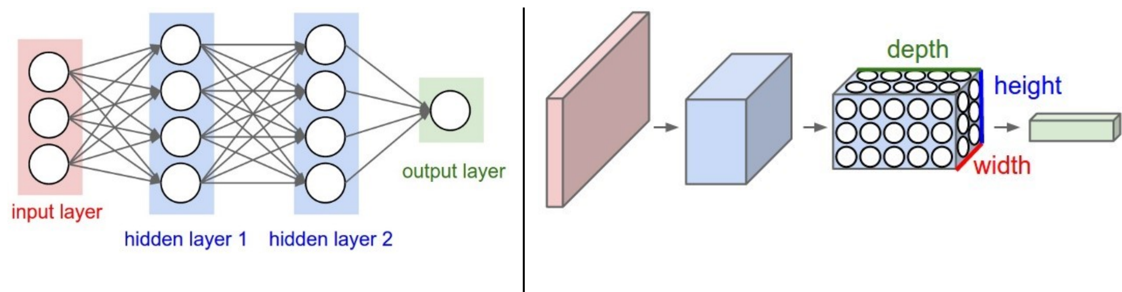


Figure 4. *Left: A regular NN with 2 hidden layers. Right: A CNN organizing its neurons in three dimensions (width, height, depth), as drawn in the right hidden layer. CNNs transform a three-dimensional volume to a three-dimensional volume of neuron activations in every hidden layer. In this figure the image is depicted in red as the input layer, so the dimensions of the image are the input layer width and height, and its depth is 3 because of RGB channels. [31]*

2.2 Layers in CNNs

2.2.1 Convolutional Layer

This layer is the major building block of a CNN which is responsible for almost all of the heavy computations.

The learnable parameters in a convolutional layer are actually making filters or kernels. Each kernel is small spatially in terms of width and height although it is extended as the input's depth. As an example, a typical kernel size in the first hidden layer of a CNN would be 5×5×3, meaning it has width and height of five pixels and depth of three which is same as number of the color channels. For passing tensors forward between layers, every kernel is convolved with the width and height of the input tensor by sliding the kernel across the current tensor and computing products of the elements of the kernel and the tensor at any position. When moving the kernel all over the width and height of the tensor a 2-D activation map is created by collecting responses from performing that kernel at every position. By this, the network is able to learn kernels capable of recognizing particular visual features like edges in the horizontal direction or a mark with a specific color in the first few layers, and eventually a complete object or more generally a pattern in the layers that come later network. In CNNs a collection of kernels in every convolution

layer exists and each is producing a 2-D activation map. By stacking these activation maps after each other the output tensor from the convolution layer is made. [29]

As we explained it is inappropriate to have connections between neurons in one layer and those in the following/previous in places that we are facing inputs with high dimensionality like images. Instead, it is better to have the connection from each neuron to a smaller area of the input tensor. It is up to us to choose the spatial extent of these connections and we call this hyperparameter neuron's receptive field (size of kernel). The depth parameter for this hyperparameter is always the same as the depth of the input tensor.

As an example, consider an input tensor with 32×32×3 size and kernel size (receptive field) of 5×5. Then in the convolution layer, every neuron would have weights associated with a region, with 5*5*3 = 75 (plus 1 bias parameter) parameters, from the input tensor. Because of the depth of the input tensor which is 3, the depth of this connection should also be 3.

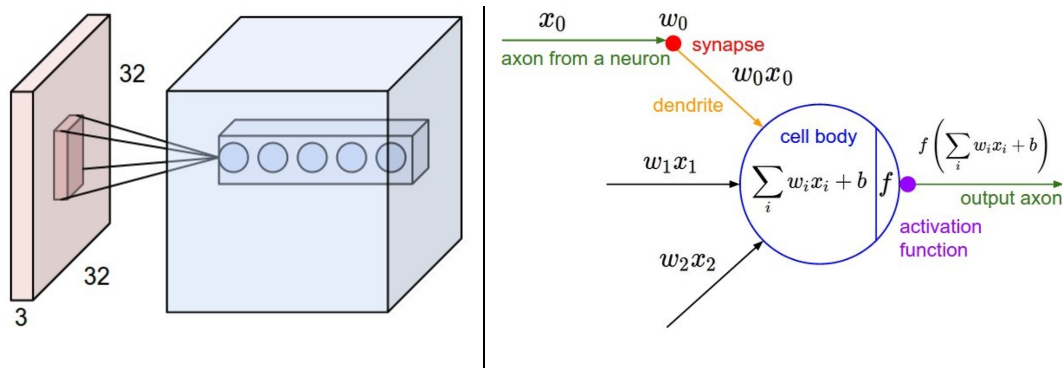


Figure 5. *Left: A sample image tensor in red with 32×32×3 size and a set of neurons in the conv layer (in blue). Each neuron in this layer has connections to the full depth but only with a small area in the input tenor. It is notable that as in this example a number of neurons, in here 5, are all operating at the same area in the tensor. Right: The neurons still perform a dot product operation on the input and their weights and then a non-linearity would follow it. [31]*

It's time to explain how neurons in the output tensor are arranged and how they affect the size of the output tensor. There are three hyperparameters involved in changing the size: the depth, stride, and zero-padding.

1. The **depth** would be equal to the number of kernels we use and each of them would be able to look for some specific pattern in the input tensor. For example, if the input for one conv layer is an unprocessed image then various neurons in the depth axis might sense the presence of edges with different directions or color.
2. **Stride** is the number of pixels that we slide the kernel at a time, e.g. when it is 1 the kernel slides 1 pixel and when the stride is 2 then it jumps over 1 pixel (slide 2 pixels). By choosing the stride more than one then a smaller output tensor would be produced spatially.

3. Sometimes having zeros around the input tensor border is more convenient for processing the input. In here, the size of the padded area with zero (zero-padding) acts as a hyperparameter. **Zero-padding** enables us to control the size of the output tensor and in most cases, we try to keep its size the same as size of the input.

The size of the output tensor could be computed using the parameters: the filter size in convolution layer (F), the stride size (S), the size of zero-padding used (P), and the input tensor size (W). We can reach the formula for computing the output tensor size: $\frac{W-F+2P}{S}+1$. As an example, we can consider images with size 227×227×3 in the first layer. Then in the next layer (which is a conv layer), assume neuros have filter size (receptive field) F=11, stride size S=4, and without zero-padding (P=0). Output size of the first convolution layer will be a tensor with (227 - 11)/4 + 1 = 55 size spatially. Again, assume the depth of the convolution layer is K=96, therefore, the output tensor would have the size of 55×55×96 and then each neuron (among these 55*55*96 neurons) would be connected to a part of the input tensor with the size of 11x11x3. Figure 6 is an illustration of a convolution operation with W=5, F=3, S=2, and P=1.

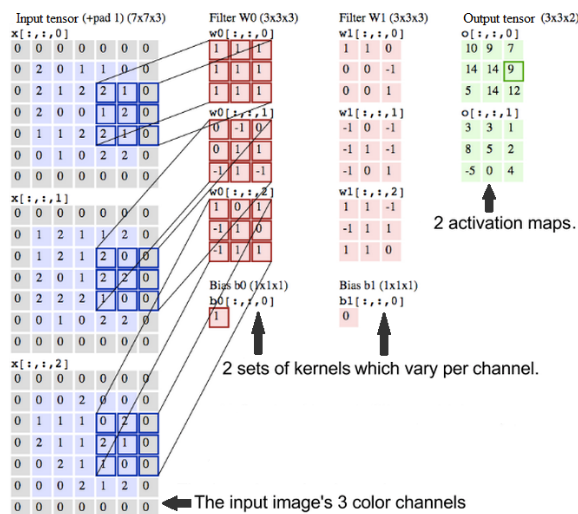


Figure 6. *Applying convolution operation on an input tensor with size 5×5×3 (with padding it is 7×7×3) and the output tensor will be in 3x3x2 size since number of filters (each filter is in 3×3×3 size) are 2. [31]*

Now we can discuss parameter sharing in convolution layers which helps a lot in reducing the number of parameters. In the above example, in the first convolution layer, we had 55*55*96 = 290,400 neurons and each of them has 11*11*3 = 363 weights (plus one bias). Easily we can see only on the first layer of the convolution layer we have 290,400 * 364 = 105,705,600 parameters which obviously is a very large number. [29]

We would be able to reduce the number of parameters to a considerable extent by making only one simple assumption and that would be: constraining neurons that are in one depth slice to have the same weights. This is originated from the idea that if a feature is useful at a specific position, then it is also useful in other positions. For example, consider a 2-D slice depth wise (e.g. we have 96 depth slices in a tensor with 55×55×96 size and

every one of them is in 55×55 size). Now in the last example and in the first convolution layer, for all depth slices, we would have 96*(11*11*3 +1) = 34,944 parameters. [29]

It is notable that assuming parameter sharing would not always be the best choice. Especially in situations that input images have certain information in specific locations, e.g. where images consist of faces in the center of them. We might expect that the network might learn facial features and their existence in specific places. In here, it is usual to relax parameter sharing meaning that allowing the network to search for a feature only in a particular area.

Backpropagation is another term in convolution layers that needs explanation. ANNs are similar to newborn babies in the essence of the fact they are uneducated and untrained in the first place, and when they are exposed to the outside world it is then that they eventually become aware and skilled. The world is expressed in form of data. By attempting to train an ANN with a specific dataset, indeed we are trying to reduce its ignorance and we can assess its progress by calculating the number of errors it made. [32]

It is an ANN's weights that enables it to grasp knowledge about the world, they modify input tensor's information while it's moving towards the network's last layer who is responsible for making a decision about the first input tensor. When the network is immature, its decisions are mostly wrong since the weights, that is altering the input in different layers, are not optimized. It is clear that these parameters (weights) are correlated with the error rate in the network and by changing them the error rate changes as well. An optimization algorithm, gradient descent, is employed to alter these parameters. This method comes handy when we want to find the minimum of a function. Our main intention here is to reduce the error as much as possible, which is a function called loss function or objective function. The input data is sent through the network's weights towards its last layer for decision making, and after that network sends back (backpropagate) error information through the network towards its first layers for modifying its weights. [32]

A gradient is a slope that can be shown a relation among 'x' and 'y' and we can estimate its slope. In here, 'x' is dependent on the weights and 'y' would be the error made by the network. By this, we will have changes of 'y' with respect to 'x' and will be able to get closer to the minimum value for the error by using differential calculus and taking the partial derivatives of each parameter contributing in error changes.

Deep CNNs are prone to suffer for the vanishing gradient problem. This would happen when we want to train an ANN model employing optimization techniques based on the gradient. In general, when we add more hidden layers to the network, it helps the network in learning more complex functions and patterns and then letting it predict results with better precision. Then we employ the back-propagation method to calculate gradients of error with respect to the weights, the gradients will eventually get smaller when moving backward in the network. This causes the learning rate of neurons in the preceding layers

to be small in comparison with the neurons in the later layers. Neurons in the preceding layers are important to the model since they detect and learn the simpler patterns. So, if they generate inaccurate outputs then the next layers and therefore the whole network will not be able to predict accurately. [33]

1×1 convolution is the last idea from convolution layers we cover in this part**.** Some people, especially those with signal processing background get confused when they are told about 1×1 convolution. The reason is that typically a signal is 2-D, so 1×1 convolution operation on a 2-D signal would be equivalent to normal multiplication. Although in convolution networks, it is different since the operation is performed on 3-D tensors, which means the depth for the 1×1 convolution would be extended as the depth of the input tensor and as discussed before, the output tensors depth would be equal to the number of filters convolved with the input tensor.

2.2.2 Pooling Layer

Pooling layers could be inserted periodically between convolution layers in a network. They gradually decrease the size of the input in order to have fewer parameters, as well as avoiding overfitting. This Layer operates individually on all depth slices of the input tensor and resizes it, usually using the MAX operation for this purpose. A pooling layer with 2×2 sized windows (filters) with stride 2 is the most common one which downsamples the input tensor in all of its depth slices by 2 along both width and height (keeping only 25% of the activations). In this case, the third dimension (depth) does not change and the MAX function would be operated over a small 2×2 area [31]. Some details about the pooling layer could be listed as below:

- Their input is a 3-D tensor in $W_i \times H_i \times D_i$ size.
- They have two hyperparameters: their spatial size (F) and stride (S).
- They will make a tensor in size $W_o \times H_o \times D_o$:
  - $W_o = \frac{W_i - F}{S} + 1$
  - $H_o = \frac{H_i - F}{S} + 1$
  - $D_o = D_i$
- It does not add any parameters.

Other than max pooling function, there are other functions that a pooling layer can perform like average pooling or L2-norm pooling. It has been shown that max pooling operation gives more accurate results in practice than the average pooling was employed traditionally. In Figure 7 you can see some visual demonstration from pooling layers.

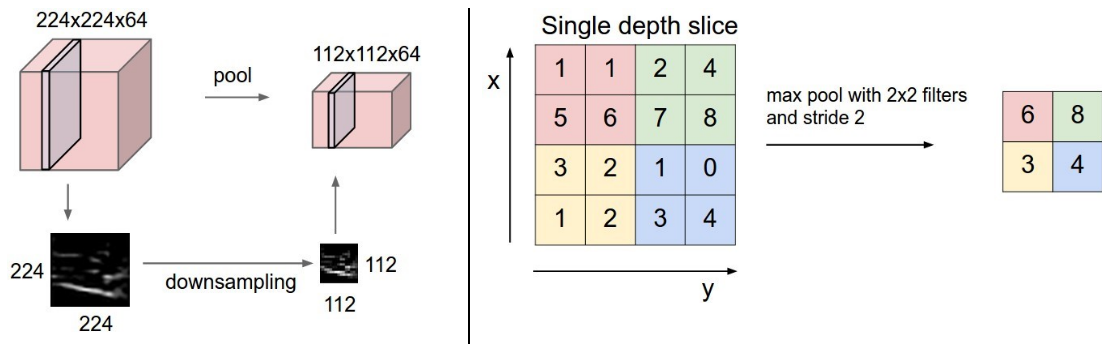


Figure 7. *Left: The input tensor with 224×224×64 size is passed through pooling layer with F=2 and S=2, as a result, the output tensor will be in 112×112×64 size. Right: The most used downsampling method is the max, and now it is used to pool the input with again F=2 and S=2. Then max of every 2×2 square is computed and replaced with that square. [31]*

2.2.3 Activation function and ReLU layer

Like other ANNs, in CNNs we also employ activation function to produce a non-linear output. In a CNN, the output tensor from convolution layers is sent to the activation function, e.g. ReLU, Sigmoid, and SoftMax activation functions.

Rectified Linear Units or ReLU layer performs a non-saturating activation function, which is defined as f(x)=max(0,x), and the size of the tensor would also be kept same. This helps in having a decision function and overall the network with more nonlinear characteristics. ReLU is used more than other activation functions since it is much faster, and it does not have a major drawback or effect on generalization factor. [31, 34]

The Sigmoid function generates a curve, which visually looks like a ‘S’. This function is used mainly for the reason that it always exists between zero and one. So, it is useful particularly when we out to predict the probability of an event. In Figure 8 the sigmoid function curve and its mathematical formula are presented.

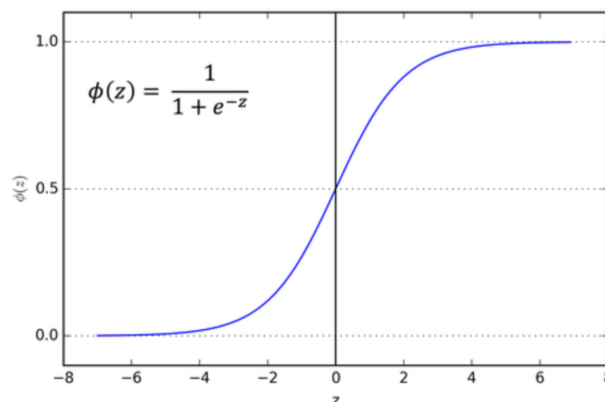


Figure 8. *Sigmoid activation function. [35]*

The Sigmoid function is differentiable, which enables us to find the Sigmoid curve's slope at any position.

It is worth mentioning here that the SoftMax activation function is a more generalized form of the Sigmoid activation function and it is employed for multiclass classification.

2.2.4 Normalization Layer

In image process and signal processing field, 'normalization' is also referred as histogram stretching or contrast stretching. Several kinds of normalization layers have been employed and suggested to normalize the input in CNN architectures. These layers are not so popular anymore since in practice they have very little contribution. [31]

2.2.5 Fully-connected layer

In fully-connected (FC) layer we are evaluating the class scores vector, which is the same length as the number of classes. And like regular ANNs, every neuron in the FC layer has connections with all the activations in the former layer. Therefore, their activations are calculated using matrix multiplication. [31]

These transformations from input images to vector of class scores in FC layers are a function of both activations in the input tensor and also weights of neurons. Gradient descent is used in this layer for learning the weights and biases, so the class scores evaluated by the CNN are compatible with the labels for the images in the training set.

So, Convolutional Neural networks could be summarized as this:

- CNN structures are based on multiple layers stacked along each other which ins simplest case is transforming the image input into an output tensor like a vector of class scores.
- Various kinds of layers exist, and the most common ones are Convolution, Normalization, Activation(ReLU), Pooling, and Fully connected layers.
- Some layers have parameters, and some do not. For example, convolution or FC layers have and activations (ReLU) or pooling do not.
- Some layers have additional hyperparameters, and some do not. For example, convolution, FC, and pooling layers have and activations (ReLU) does not.
- Layers receive a 3-D tensor as input, process it and produce another 3-D tensor as output.

2.3 Major building blocks of CNNs

As described before we can think of Convolutional Neural Networks as a specific type of ANNs with multiple layers which are meant for computer vision tasks like recognition of visual patterns in RGB images with minimum effort and pre-processing.

A major part of recent progress in deep learning algorithms related to computer vision area could be outlined as several neural network architectures. So, in this section, we will give an adequate explanation about different blocks contributing in a few numbers of these CNN architectures such as ResNet, Inception, VGG, Xception, Inception-resnet, and in more efficient ones like Mobilenet that we used in our experiments.

In this part, we explain what the most common building blocks of the efficient ANN architectures before are before explaining any particular CNN architecture. We would see the computational cost of these blocks and check how they differ from the original convolution operation.

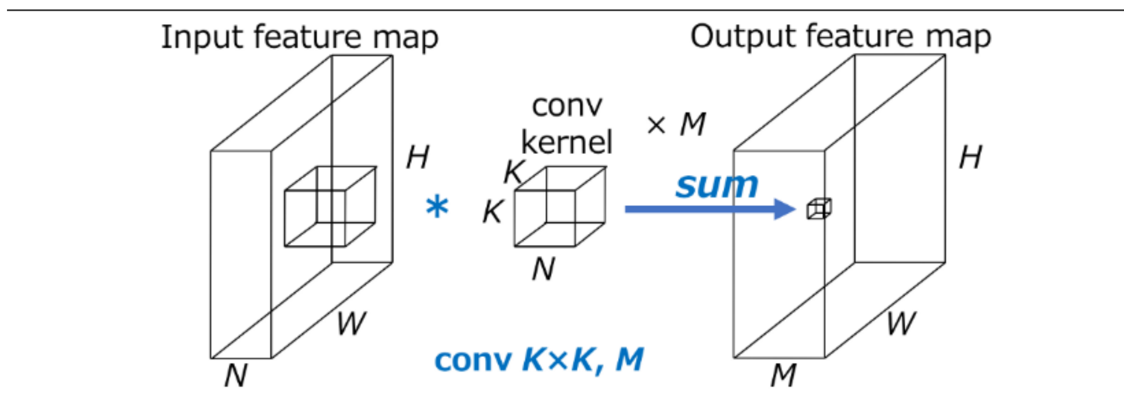


Figure 9. *Convolving a= 3-D input tensor with a 3-D filter. [36]*

In Figure 9 the size of the input, a tensor is equal to H×W which has N channels. Consider M number of convolution kernels in size of K×K with depth size of N. the resulting output tensor from this convolution operation will be in size of H×W with M channels. This is a normal convolution operation which would have the computational cost of $H*W*N*K^2*M$.

The key point in the standard convolution operation is that the computational cost is proportional to the size of the input tensor (H×W), the size of the filter (K×K), depth of these two (N), and the numbers of filters (M). We can speed up this process in CNNs by factorizing this convolution which will be explained soon.

2.3.1 Depthwise convolution

In the standard convolution, we perform the operation over multiple channels by making use of filters with the same depth as the input tensor. In the Depthwise we convolve 2-D

filters with each depth slice of input tensor individually[37]. This is shown the upper part of Figure 10.

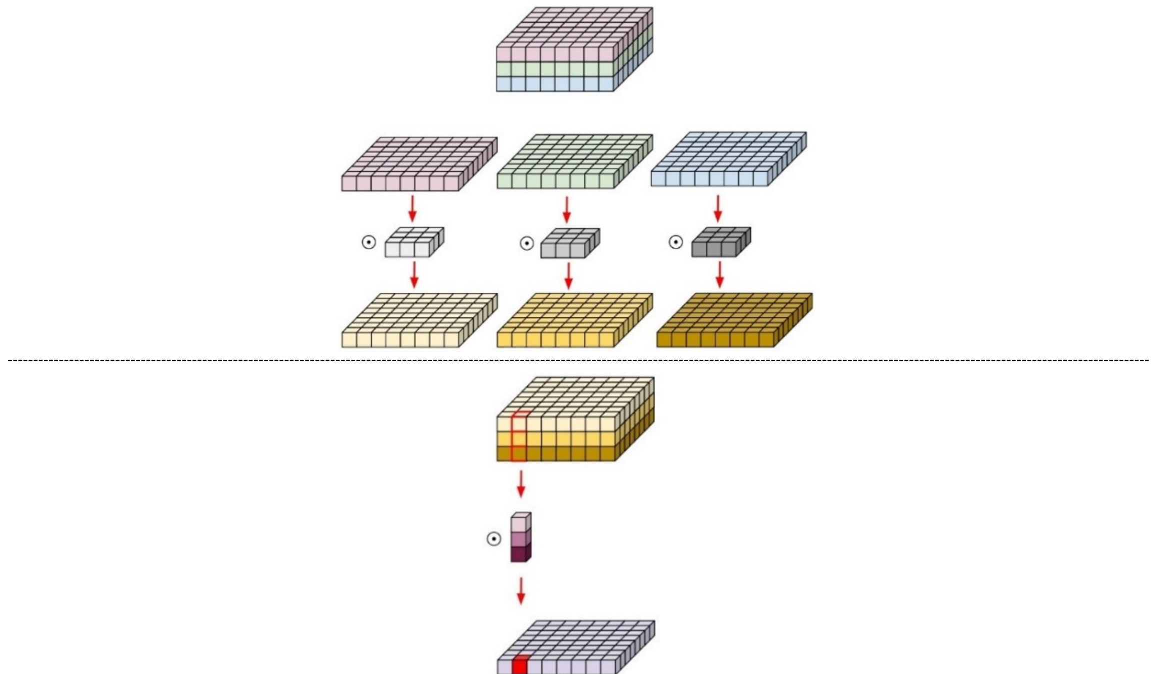


Figure 10. *Up: Depthwise convolution. Down: Pointwise convolution. [37]*

2.3.2 Pointwise Convolution

Pointwise convolution is same as 1×1 convolution where as shown in downer part of Figure 9, we convolve each pixel (1×1 element) including all of its channels with a filter in size 1×1 with the same number of channels. Primary characteristics of this operation: Changing the dimensionality and also enables us to apply nonlinearity afterward again.

An example might make it clearer: if we have an input in size 224×224×3 and we apply pointwise convolution with 10 of the 1×1 filter, where each has depth 3, then we will have output with 224×224×10 size.

2.3.3 Depthwise separable convolution

This type of convolution which is used in many architectures is consist of both depthwise convolution and respectively pointwise convolution. Multiple outputs from the depthwise convolution layer would be stacked to each other making a 3-D tensor which is the input for the pointwise convolution layer. Then the pointwise convolution layer takes this 3-D tensor and applies coevolution operation with multiple 1×1 filters resulting in an output with the same size but with the desired depth. The block which is used in Inception network is also similar with this block, the difference is that in here at first, we apply depthwise and then pointwise convolution, while in Inception module we apply the pointwise

convolution first. Usually, depthwise separable convolutions are used without non-linearities [37]. Recently this method has become interesting to researchers working in deep ANNs area since:

1. It will produce fewer parameters than standard convolutional operation, so it needs fewer computations and thus it would be faster and cheaper.
2. Since it has fewer parameters, the model will be less prone to overfitting.

Now we can check the difference between the computation cost of standard and this type of convolution. We assume:

- $H \times W$: Input's dimensionality.
- K: Width/height of the filter.
- N: Number of input channels.
- M: Number of output channels.

As mentioned before, in a standard convolution the computational cost is $H*W*N*K^2*M$. In depthwise convolutions, for each channel, the computational cost is $H*W*K^2$ and then in total in the depthwise layer, the number would be $H*W*N*K^2$. And in the pointwise convolution layer, the computational cost would be $H*W*N*M$. It is obvious that the sum of these two is significantly smaller than in standard convolution: $H*W*N*K^2*M >> H*W*N*K^2 + H*W*N*M = H*W*N*(K^2+M)$.

2.3.4 Inception module

The idea originally comes from the fact that we have to choose what type of convolution should be used in each layer, i.e. a 3×3, a 5×5 or 7×7? Another solution is that to have all of them and let the model choose between them. This would be possible by performing each of those convolutions in parallel and then stack results to make a new feature map as input for the next layer [38]. A single inception module is shown in Figure 11.

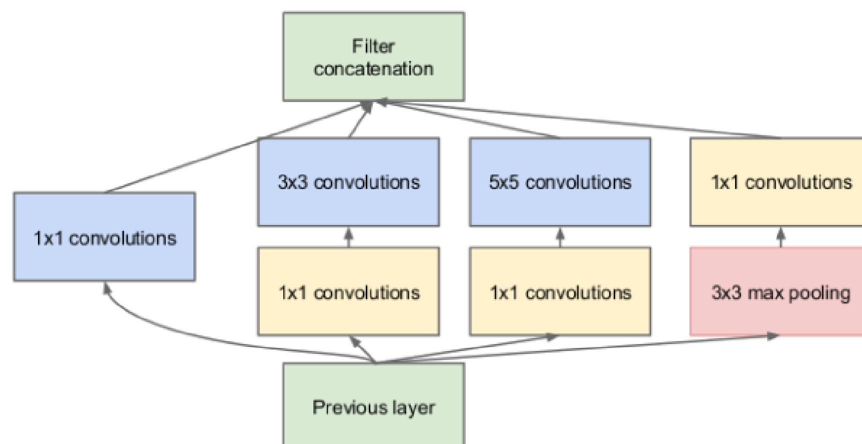


Figure 11. *Inception module.[38]*

As you can see we have various convolutions, e.g. we would use a 3×3 max pooling beside 1×1, 3×3, and 5×5 convolutions. The max pooling operation is part of the Inception module just because traditionally accurate networks have pooling. When convolutions become larger, they also become more computationally expensive, so in the Inception module at first, a 1×1 convolution operation is performed to reduce the dimensionality of the input feature map, and then pass it through a ReLU. Now we are able to perform the larger convolution with less computational cost on the result.

2.3.5 Residual block

When the depth of a network increases, its accuracy gets worse and then it reduces quickly. surprisingly this reduction in accuracy is not because of overfitting and stacking more layers to such models would result in a higher error during training.

The inventors of ResNet try to address this issue by making a single hypothesis: it is difficult to learn direct mappings. So they suggested: Instead of expecting that some layers could estimate the desired underlying mapping function, they offer that these layers estimate a residual function F(x) = H(x)-x. therefore the original mapping could be derived by a simple adding operation: F(x)+x. The idea behind this method is that it is easier to optimize this residual mapping than the original one. As the extreme case consider that for making an optimal identity mapping, it is more efficient and accurate to move the residual to zero than approximating an identity mapping by several nonlinear layers[39]. The two types of this block are depicted in Figure 12.

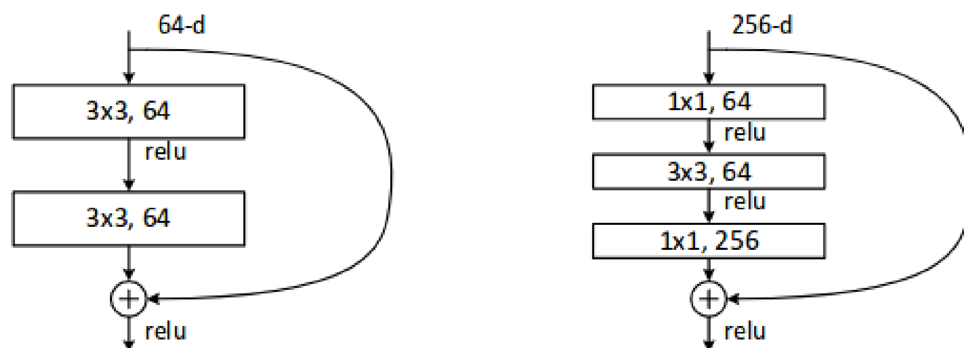


Figure 12. *Left: ResNet block with two layers (ResNet 18, 34)*
Right: ResNet block with three layers(ResNet 50, 101, 152)

Acknowledging the mentioned core blocks of neural networks, now we discuss mostly used neural network structures.

2.4 Mostly used neural network architectures

2.4.1 VGG16

This architecture of neural networks makes some improvement over the state-of-the-art method prior to it, Alex Net, by substituting large filters, i.e. size of 11 and 5, with several 3×3 filters. When the receptive field is clear then stacking a number of smaller kernels works better than a larger size one since they increase the model depth and therefore allowing it to learn more complex features efficiently [40]. The network architecture is shown in Figure 13.

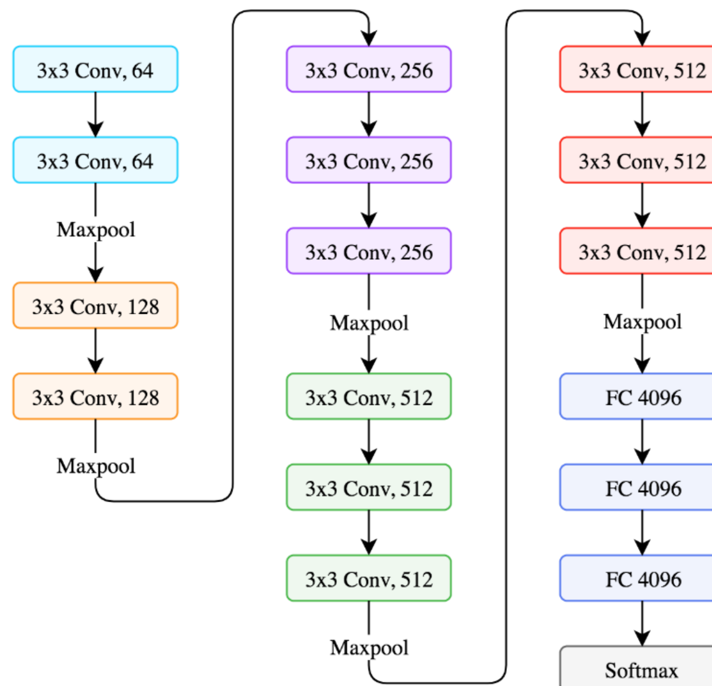


Figure 13. *VGG-16 structure.[41]*

VGG-16 [40] architecture is simpler than ones developed recently as it does not have many hyperparameters. In convolution layers, it is mostly employing 3×3 sized kernels with stride 1 and in pooling, layers 2×2 kernels with same padding and stride 2. At the beginning of VGG structure hierarchy, the width of the network is 64 which is relatively small and this increases after each pooling layer by factor of two and three fully connected layers come in the end after all the convolutional layers. This structure achieved the accuracy of 92.3 % on ImageNet.

2.4.2 Mobilenet

The core block in the MobileNet [3] structure is depthwise separable kernels that we introduced earlier as Depthwise Separable Convolutions. The structure of the network itself is playing a significant role in boosting the performance and it is shown in Table 1.

There are two additional hyperparameters in Mobilenet: Width Multiplier which is employed to reduces the number of the channels and the Resolution Multiplier which is employed to reduce the spatial size of the input image [3]. There is a trade-off between accuracy and latency and we can tune those two hyperparameters to achieve a balanced result.

Table 1. *Structure of MobileNet. [42]*

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1 Conv / s1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$ $14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

In summary, Mobilenet structure is mainly contributing in two ways:

- MobileNet architecture is suitable for computer vision applications executing on mobile phones or embedded devices since it is an efficient deep neural network structure based on a streamlined architecture making use of depthwise separable convolutions.
- MobileNet architecture has two simple hyperparameters that establish an efficient trade-off between accuracy and latency.

2.4.3 Inception V3

An ANN architecture was constructed by using the inception module. If we assume we have an Inception module in the next layer as well, then as explained before, each feature map would be passed through a combination of convolution operations and all results

would be concatenated to make an input for the next layer. This structure of a network would enable the model to understand both high abstracted features and local features by making use of bigger convolution filter sizes and smaller convolution filters respectively.

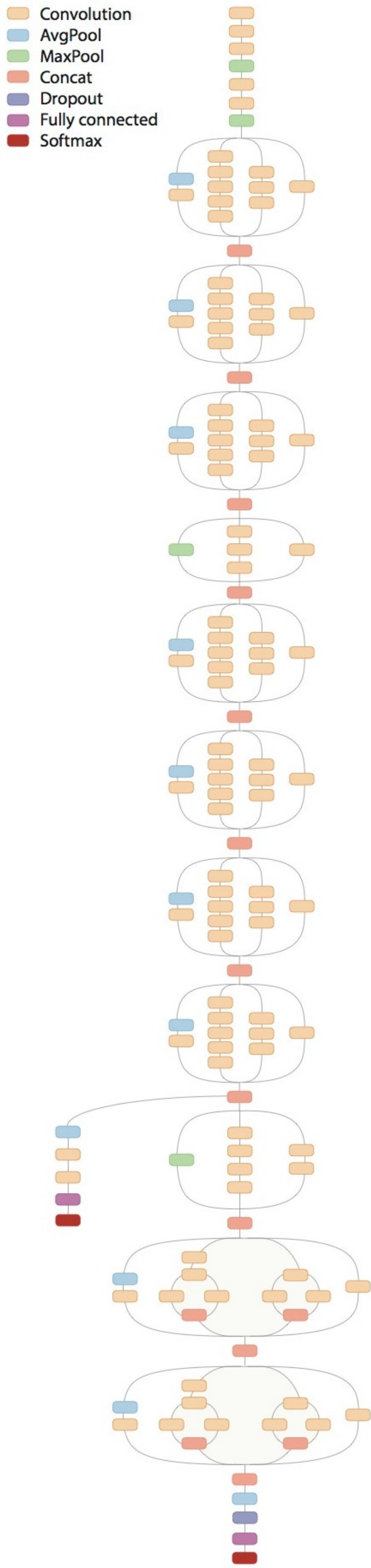
It is obvious that the Inception V3 [43] has a pretty deep structure and like any other very deep network, it is prone to the issue of vanishing gradient.

In Figure 14 A you can see the structure of this model. Authors [43] presented an auxiliary classifier. They performed SoftMax to the result from one of the inception modules located about in the middle of layers' hierarchy and calculate its loss which we can call auxiliary loss. The total loss function would be calculated as a weighted sum of the real loss in the last layer and the auxiliary loss.

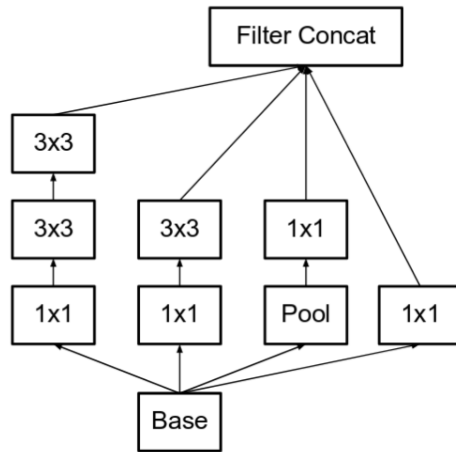
For reducing the computational costs, in this structure, 7×7 and 5×5 convolutions are factorized to a couple of smaller convolution operations. This might look counterintuitive but a single 5×5 convolution operation is approximately 2.8 times slower than one 3x3 convolution. Therefore, concatenating two separate 3×3 convolution operations helps in boosting the performance. This is depicted in Figure 14 B.

As shown in Figure 14 C convolution of a filter with n×n size is factorized to a mixture of n×1 and 1×n convolutions. For instance, a 3×3 convolution operation is the same as performing a 1×3 convolution at first and then a 3×1 convolution operation on the previous result. This way of expanding convolution is approximately 33% faster than one 3×3 convolution.

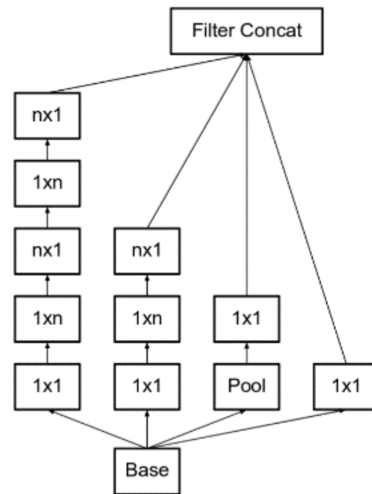
In Figure 14 D you can see that this model is expanding the filter banks by making them wider, i.e. not deeper, to avoid loss of data. It means if the module becomes deeper, the model will suffer from excessive dimension reduction which is equivalent to information loss.



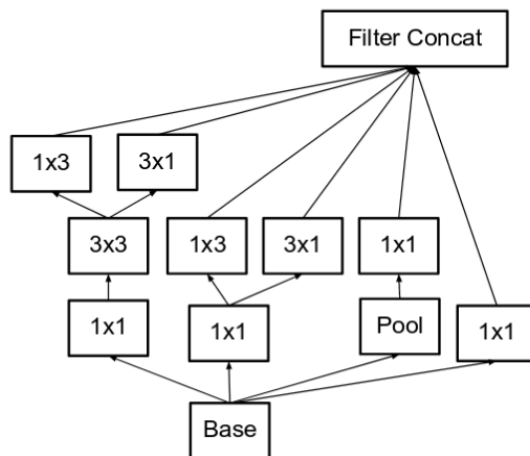
A. *Inception V3 structure.*



B. *5×5 convolution factorized to two 3×3 convolutions.*



C. *Converting the n×n convolution to 1×n and n×1*



D. *Making filter banks wider rather than deeper*

Figure 14. *Inception network and its modules.[38, 44]*

2.4.4 ResNet-50

Looking at the residual block that is employed in ResNet-50 [45] is a good place to start studying this model and comparing it to other ones.

As shown in Figure 15, residual blocks in this model are made of 1×1, 3×3, and 1×1 convolution filters. At the beginning of this block, a 1×1 convolution reduces the dimensionality of the input tensor, which helps a lot in the computational cost of the following 3x3 convolution which is relatively expensive. The last 1×1 convolution is responsible for recovering the dimensionality of the output tensor.

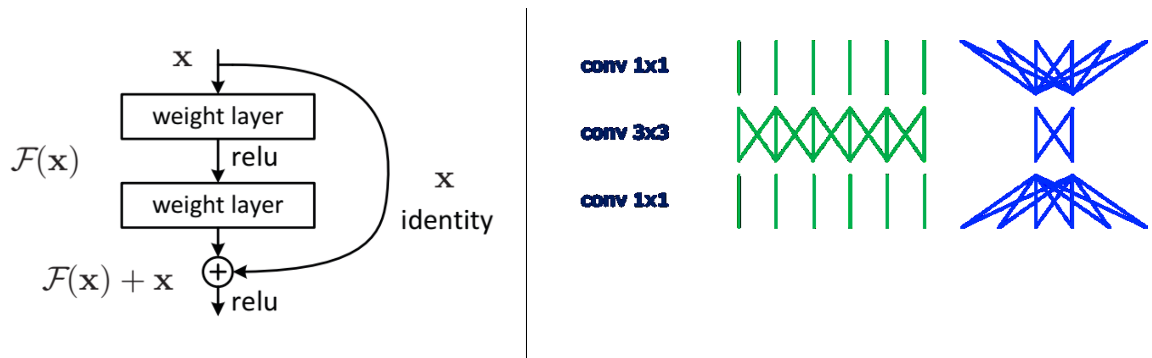


Figure 15. *Left: Resnet building block. Right: Connections in this block. [36, 45]*

Each of these blocks in this architecture is made up of a sequence of layers and there is a shortcut adding the input tensor to the output tensor. While performing this element-wise add operation if the input size is different than the output size, 1×1 convolutions or zero-padding should be employed to match their dimensions.

In practice, this method works notably well. Before this deep ANNs usually suffered from the vanishing gradients issue, where gradients from the loss function reduce exponentially while they were backpropagated to preceding layers in the hierarchy. In fact, in practice when the errors moved backward to the preceding layers, they have become such small that makes the learning procedure impossible for the network.[45]

The gradient in ResNet architecture is able to move back to preceding layers using the shortcuts, and this enables us to build deeper networks, i.e. with 50, 100, 150, and even 1000+ layers, that still operate stunningly good.

This architecture of ANNs was a major breakthrough after its previous state-of-the-art method, GoogleNet/Inception V1, which had 22 layers and won ILSVRC 2014 challenge. It fundamentally changed our understanding of ANNS and their learning mechanism.

2.4.5 Inception-resnet V2

Inception-ResNet V2 [46] is a CNN engineered by Google researchers in hope of progressing in the field. This model actually reached higher accuracy on the ILSVRC

image classification benchmark. In short, it is a version of the Inception V3 architecture that is borrowing the main idea of residual blocks. This idea enables shortcuts between some layers and therefore training deeper and deeper ANNs with better performance [46]. This has also helped the Inception blocks to be simplified.

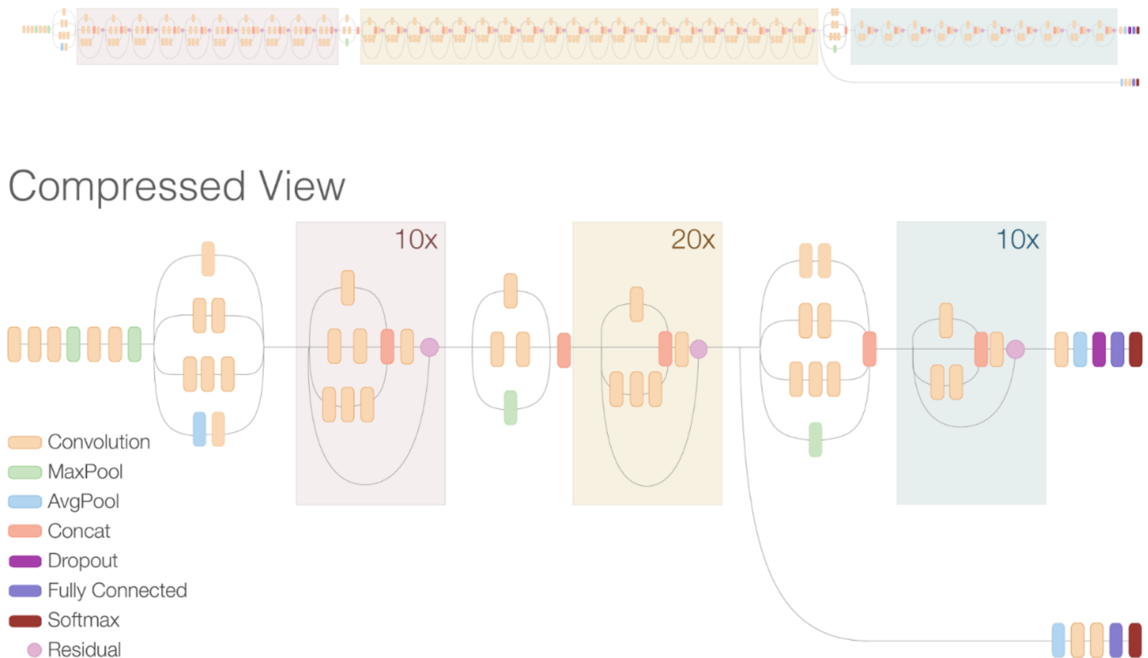


Figure 16. *Inception-resnet V2 architecture. [44]*

You can see the complete network on top of Figure 16, which is noticeably deeper than the original Inception V3. You can also see in the downer part of this figure, which has the residual blocks compacted in one block, that the inception blocks are simpler than the original Inception V3 and they have fewer parallel connections.

The authors claimed that this model is more accurate than its previous state of the art model while it only needs about twice the computational power and memory than the original Inception V3.

2.4.6 XCeption

The module shown in Figure 17 is the major building block of Xception [47]. This module there is a pointwise convolution and then a separate convolution on each depth slice. This process is almost same as depthwise separable convolution and there exist only two slight differences between them: First is the order of the two operations and the second one is that in this module these operations are followed by ReLU non-linearity. Having convolutions for every output channel of the 1×1 convolution could be considered as an extreme version of Inception module and that is the underlying idea for naming ‘Xception’ architecture which means ‘Extreme Inception’. [47]

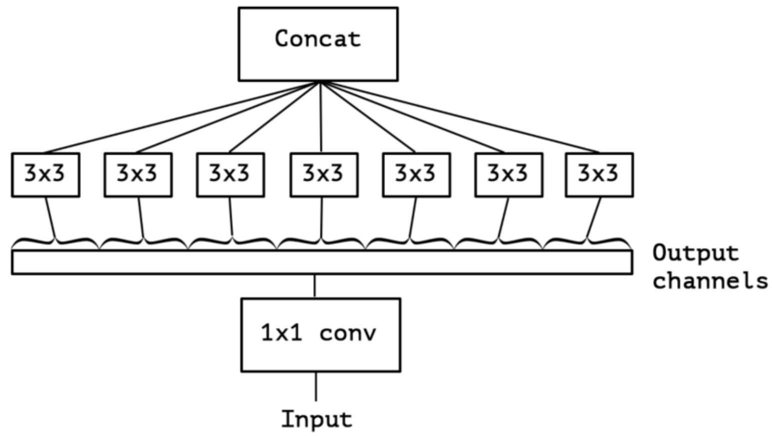


Figure 17. *Xception module. [47]*

The Xception architecture is illustrated in Figure 18 which shows this model is simply series of separable convolution layers which are also bundled with residual connections.

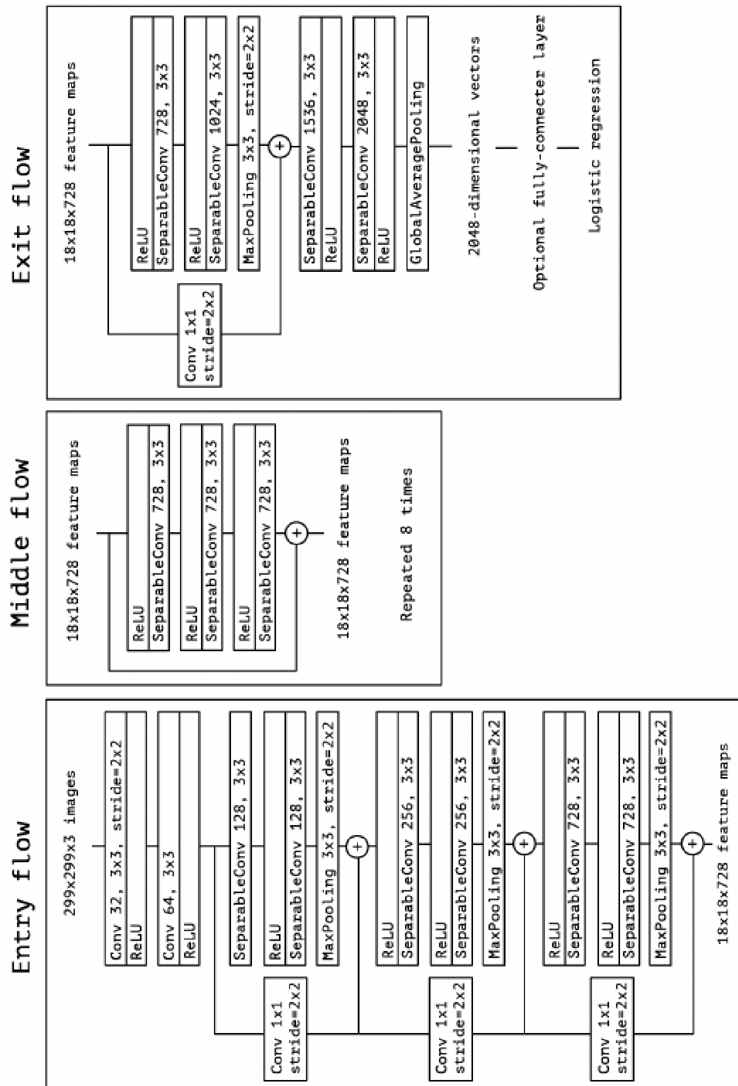


Figure 18. *Xception architecture. [48]*

3. TOOLS AND METHODS

In this study, we have used TensorFlow framework as the backend and on top of that, Keras API for developing and implementing our smile recognition system, with Python programming language, in low resource environments and devices. In this section at first, we will explain these two popular libraries and then describe two other steps of preprocessing images and system level architecture.

3.1 Frameworks and APIs

3.1.1 TensorFlow

TensorFlow[1] is a framework developed by Google for machine learning purposes and more specifically designing and building of deep learning models. The first client programming language that TensorFlow supports was Python but currently, many of its functionalities have been moved into the TensorFlow core which is implemented in C++, therefore other programming languages could be used through their foreign function interfaces to call the C API for having TensorFlow functionality.

TensorFlow at its core is a library for dataflow programming. It is employing several optimization techniques to transform mathematical operations and calculations into more efficient and easier ones. The main task of TensorFlow library is to do numerical computations making use of data flow graphs in which nodes are representing mathematical operations and the data, which mostly is multidimensional tensors, is represented as the edges which are the connections between nodes. TensorFlow key features could be summarized as this:

- It is capable of doing mathematical operations efficiently on tensors or multi-dimensional arrays.
- Supports deep ANNs and other machine learning methods widely.
- It is capable of executing a program on both GPU and CPU.
- It is highly scalable in terms of splitting computations across multiple machines and also extra-large sets of data.

As you probably have noticed, Google got the name for ‘TensorFlow’ from the operations that are performed on tensors in ANNs. And now we explain basic concepts involved in TensorFlow.

[1] https://www.tensorflow.org

Tensors

A tensor could be considered as a general form of vectors and matrices which might have high dimensionality. Internally, TensorFlow represents tensors as n-dimensional arrays of base datatypes. Tensors major specifications are explained below.

1. Rank

The number of a tensor's dimensions is its rank which is also called degree, order, and n-dimension. This is important to notice that rank in TensorFlow tensor is not equivalent to the rank defined for matrices in mathematics. Table 2 is showing that tensor's different ranks in TF correlate to which mathematical concept:

Table 2. *Ranks in TensorFlow*

Rank	Mathematical form
0	Scalar (magnitude only)
1	Vector (magnitude and direction)
2	Matrix (table of numbers)
3	3-dimensional Tensor (cube of numbers)
n	n-dimensional Tensor

2. Shape

The number of elements in every dimension of a tensor is called its shape. Shapes of tensors are calculated during construction of graphs in TensorFlow. In the TensorFlow documentation, three notations are used to express dimensionality: shape, rank, and dimension number. Table 3 explains the correlation between them.

Table 3. *Shapes in TensorFlow*

Rank	Shape	Dimension number
0	[]	0-D
1	[D0]	1-D
2	[D0, D1]	2-D
3	[D0, D1, D2]	3-D
n	[D0, D1, ... Dn-1]	n-D

3. Data types

In TensorFlow, a data type is assigned to tensors and you cannot have multiple data types for a tensor, however, it is possible to convert different data types to strings and store them a tensor. You also can cast tensors one datatype to another, inspect a tensor data type, and specify the datatype when making a new tensor and if you don't, a datatype by which the data could be represented would be chosen automatically by TensorFlow. TensorFlow transforms Python integers to a type defined by TensorFlow itself, tf.int32, and floating points to tf.float32 TensorFlow type and other data types are converted with the same rules as are used for array conversions in Numpy library.

Data Flow Graphs

TensorFlow Core includes two separate parts. First building the computational graph and then by using a session it runs that computational graph. A computational graph has been illustrated in Figure 19.

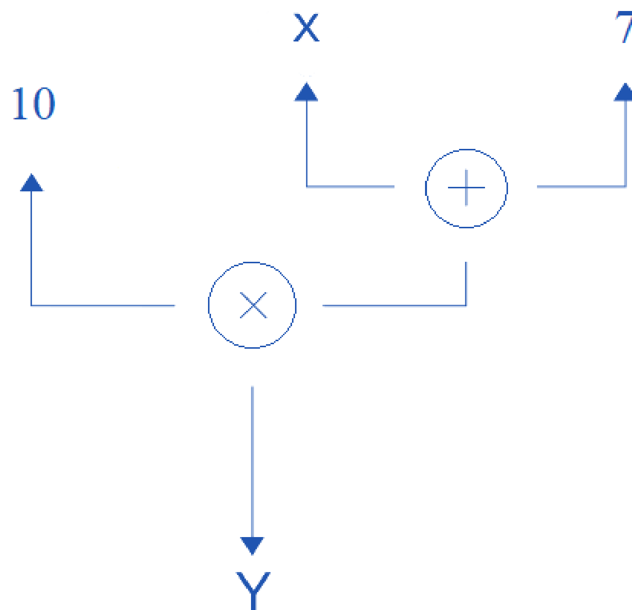


Figure 19. *Data flow graph.*

Dataflow graphs in TensorFlow are one big advantage in TensorFlow which is allowing separation of the execution model from the execution itself, which could be performed on GPU or CPU. It is also good that there is no explicit need to instantiate Graph objects when building the computation graph. As mentioned earlier a computational graph is a number of operations formed as a graph. This graph is made of two kinds of objects.

- Operations that consume and produce tensors and perform calculations. They are nodes of the graph.
- Tensors that flow through the graph and represent the values. They are edges in the graph.

Now let's first look at few lines of code in TensorFlow:

```
import tensorflow as tf

# Initializing two constants
C1 = tf.constant([1,2,3])
C2 = tf.constant([4,5,6])

# Add
R = C1 + C2

# Printing the result
print(R)
```

In the above code snippet, we have defined constants. There are two other value types in TensorFlow. Firstly, the variables that their values can be changed and secondly placeholders that are unassigned values and would be initialized when we run the session. Notice that we need to feed the placeholder tensor with data during the session runtime and if we forget to do so the placeholder generates an error. The major advantage of using placeholders is that they enable developers to create operations and therefore the computational graph itself; So, there would be no need in delivering the data to the graph in advance and TensorFlow can fill with the data received from external sources during execution time.

TensorFlow has a lazy evaluation and it means the output of these lines of code is an abstract tensor from the computational graph and the result has not been calculated actually. The code has defined the model only and does not compute the result. For retrieving the actual result, we should run this code within a session as demonstrated here:

```
import tensorflow as tf

# Initializing two constants
C1 = tf.constant([1,2,3,])
C2 = tf.constant([4,5,6])

# Add
R=C1,C2

# Initializing the Session
S = tf.Session()

# Printing the result
print(S.run(R))

# Closing the session
S.close()
```

"A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated." is the explanation given about sessions in the TensorFlow documentation.[1]

---

[1] https://www.tensorflow.org/api_docs/python/tf/Session

3.1.2 Keras

The main idea behind creating Keras[1] was enabling fast experimentation leading to the transformation of ideas to results with no frustrating delay. So, it could be explained as a high-level API for neural networks in Python programming language which runs on top of different backends including TensorFlow, Theano[2], Microsoft Cognitive Toolkit (CNTK)[3], and Amazon is working on creating a MXNet[4] backend currently. Keras employs TensorFlow by default for manipulating tensors.

There are plenty of deep learning frameworks so why does someone prefer to use Keras than other alternatives? Here we mention some areas in which Keras is favored among developer.

- Keras is a user-friendly API designed for human beings. It provides simple and consistent APIs and clear feedbacks for user errors. That is why we say Keras is easy to use or learn. Users would become more productive since Keras allows faster implementation of their ideas.
- It supports modular programming. In Keras a model is assumed as a series of modules which are fully configurable and able to be attached together with the least possible restrictions. In total, all layers including neural layers, loss functions, activation functions, regularization schemes, optimizers, and initialization schemes are independent modules that could be mixed in diverse ways to develop new architectures of ANNs.
- It is easy to extend which means we can simply add new modules, as new functions or classes.
- It is flexible in terms of integrating with lower-level deep learning APIs, in particular, TensorFlow, and this lets us develop things that one could build using the lower-level API. In addition, A Keras model would be portable across all backends just if it uses built-in layers. This means if we train a model using one backend then we would be able to load it with another.
- Models are expressed in Python language which makes the debugging process easier and also there would be no need for individual configuration files for declaring models.
- It is easy to deploy the models created in Keras across a wide range of platforms: on iOS using Apple's CoreML[5], on Android using the TensorFlow Android runtime, on browsers using GPU-accelerated JavaScript runtimes such as Keras.js[6] and WebDNN[7], on Google Cloud using TensorFlow-Serving[8], on a Python web application backend like a Flask application, on the JVM using

[1] https://keras.io/
[2] http://deeplearning.net/software/theano/
[3] https://www.microsoft.com/en-us/cognitive-toolkit/
[4] https://github.com/awslabs/keras-apache-mxnet
[5] https://developer.apple.com/documentation/coreml
[6] https://transcranial.github.io/keras-js
[7] https://mil-tokyo.github.io/webdnn/
[8] https://www.tensorflow.org/serving/

DL4J model import provided by SkyMind[1], and on embedded devices such as Raspberry Pi[2] and NVIDIA Jetson.

- We can train Keras models on different platforms including CPUs, NVIDIA GPUs, Google TPUs using TensorFlow backend and Google Cloud, and using the PlaidML[3] Keras backend with OpenCL-enabled GPUs.
- Keras supports training via distributed platforms and multiple GPUs, e.g. it has built-in support for data parallelism on multiple GPUs, or we can convert models in Keras to TensorFlow Estimators and then train them using clusters of GPUs, and it can also be run on top of Spark via Elephas[4] and Dist-Keras[5].

By November 2017 Keras had over 200,000 individual users which rank second after TensorFlow itself in terms of being employed among both the industry and the research community, and it should not be neglected that Keras is usually used with TensorFlow backend. [49]

Keras is used by many famous products like Netflix, Uber, Yelp and by many startups using deep learning. Figure 20 is showing that Keras ranks second in terms of a total number of mentions in scientific papers published in arXiv[6] and therefore is also popular among deep learning scientists. [49]

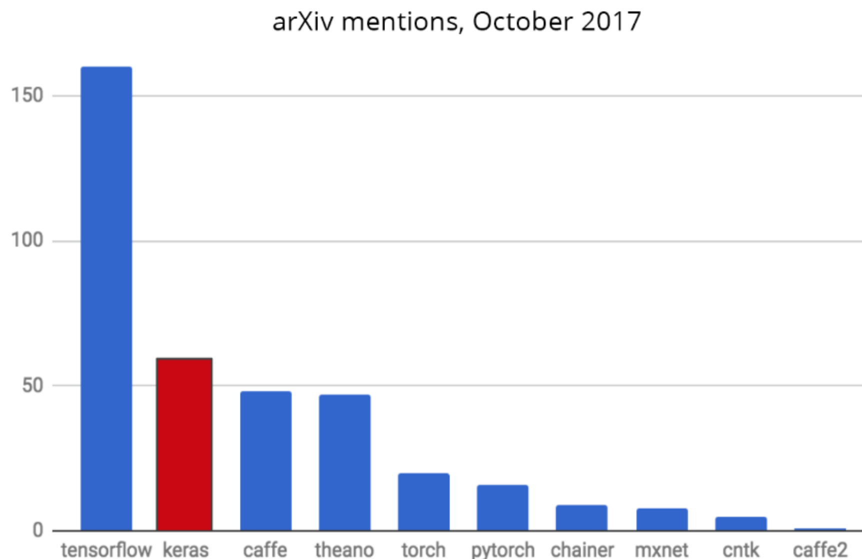


Figure 20. *Use of Keras framework among researchers. [49]*

[1] https://deeplearning4j.org/model-import-keras
[2] https://www.raspberrypi.org/
[3] https://github.com/plaidml/plaidml
[4] https://github.com/maxpumperla/elephas
[5] https://github.com/cerndb/dist-keras
[6] https://arxiv.org/

3.2 Image pre-processing (geometrical transformations)

For training, any machine learning model data is needed and as mentioned earlier we employed a couple of image datasets for training different neural network architectures in order to create models capable of smile detection. All the images we used were not fully annotated and therefore need pre-processing. This made us perform firstly face detection algorithms on the face images enabling us to retrieve salient parts in the images, i.e. the rectangles surrounding faces. Another employed method that helped us to achieve higher accuracy in the final result was face alignment. In this section, we would explain these two steps of image processing in more details.

1) Face Detection: First step in detecting smile is, of course, finding the location of the face in an image. Because of computational power restrictions we had on the chosen platform, we decided to use the classical and fast Viola-Jones detector [7] in the deployment stage. As far as our goal was to implement a real-time detector, lower accuracy of this method would not be a significant issue since a few numbers of missed detections is not critical within a sequence of frames.

It should be considered that it was not crucial to us to reach a good execution speed during the training procedure, therefore we employed a more sophisticated face detector [50] in this phase in order to take the most out of the training images. Using two different algorithms for training and deployment stages would not make any problem since this step is followed by the alignment step which would compensate for their different behavior.

2) Face Alignment: An essential part of the proposed smile detection system is face alignment. Face alignment is an act of normalizing the input face image in a way that various parts of the face are always at the same relative location. This lets the network to trust that for example lips are all the time at the same location in different images. This method also takes care of distortion issues images might suffer from such as rotation and scale which most probably would lead to drops in the accuracy.

The employed alignment method is well suited for real time applications. For reducing consumption of computational power in this method, we used a landmark-based approach [51] which locates the position of a predefined number of landmarks with an ensemble of regression trees and then matches a reference landmark set [52] with the obtained ones. Figure 21 is depicting the landmark template set that we used here, which was retrieved from the landmarks of a randomly selected training image and then normalized to have horizontal symmetry. A symmetric landmark set allowed data augmentation by left right-flips for training images. The alignment procedure could be described as an affine transformation matrix which would look like this in homogeneous coordinates:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

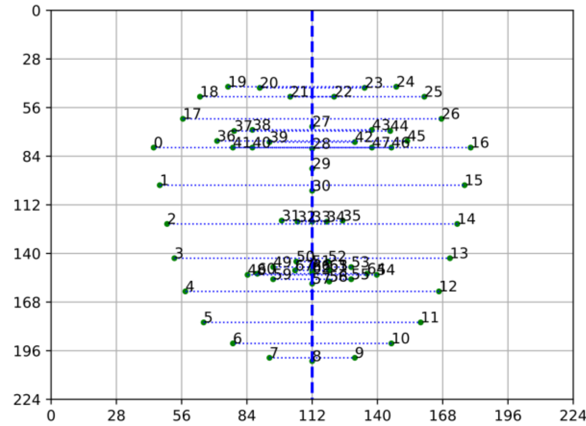


Figure 21. *Pairs of symmetric landmarks for alignment are connected by dashed lines.*

As you can see in Figure 22, if an affine transformation is too intense visually we have found out the geometry of faces in output image could suffer from distortions which would lead to degradation in accuracy. Therefore, a suitable similarity transformation is needed that has more restrictions than a full affine transformation. The allowed transformation in the proposed alignment method are an only rotation, scaling, and translation and not shearing. Moreover, we avoid considerable extent of distortions by calculating the ratio of the maximum and minimum eigenvalues from the employed transformation matrix. So, if the ratio is more than 2 we do not perform the transformation [52].

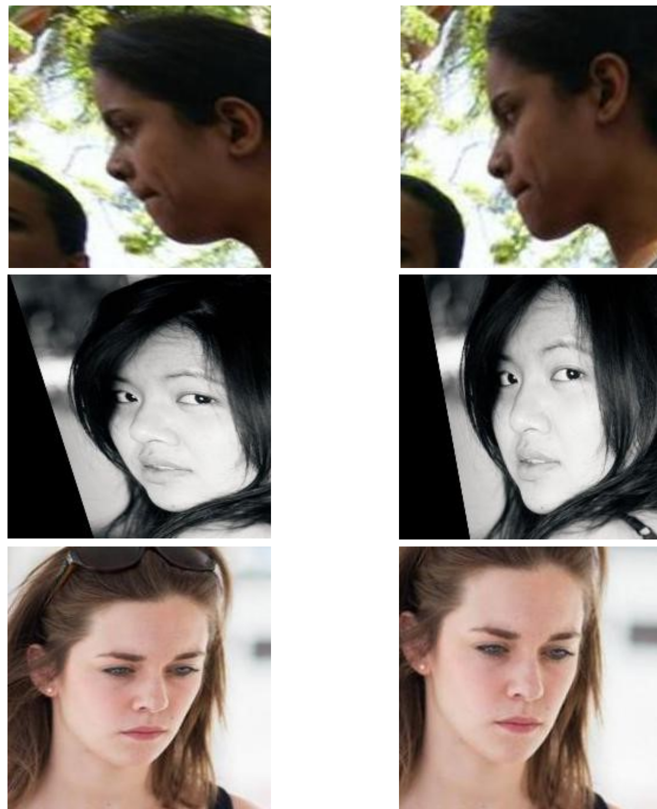


Figure 22. *Left: Aligning images without restrictions.*
Right: Face alignment with restrictions.

3.3 System level architecture

Our target platform for the smile detection system is an embedded device named NVidia Jetson TX2 with six ARM CPU cores and an embedded GPU with 256 CUDA cores. In addition to this device, we deployed the system on a Desktop computer with and without a GPU for the sake of comparison whose results and benchmarks are given in the experimental section.

Implementation of the smile detection system is based on asynchronous communications and multithreading which allows computation parallelization. The software architecture of our system, which is shown in Figure 23, includes a main thread and couple of worker threads, each of which is responsible for a specific task in the pipeline, i.e. grabbing, face detection, smile detection, etc. All worker threads are instantiated within the main thread and asynchronously poll frames from a stresam of frames. When a worker thread requires a frame then it would get access to the most recent one provided by the main thread through a shared data structure between threads. Additionally, all grabbed frames have to meet the necessary prerequisites, e.g. all frames should be processed during the face detection step and the passed to other threads. Worker threads would attach the output of their process into the frame.

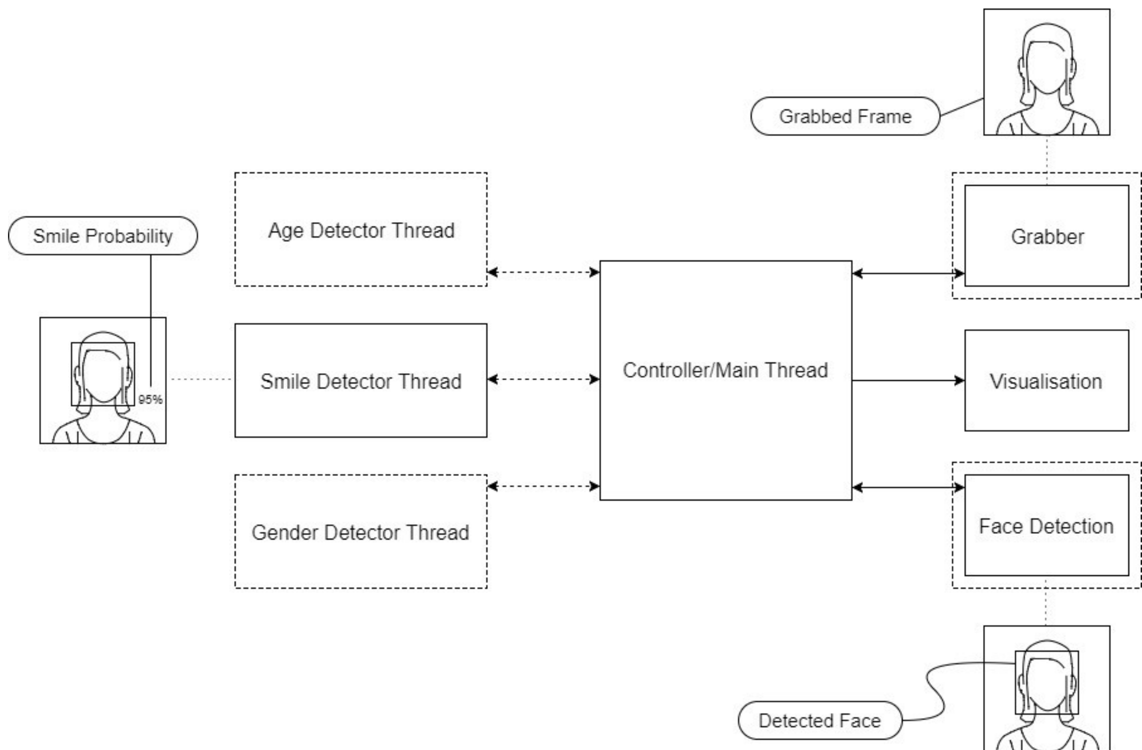


Figure 23. *Software architecture diagram for the smile detector.*

The boxes drawn with a dashed outline, in Figure 21, are some potential future extensions, such as age detection, that could be added to the system easily thanks to its modular architecture. The modularity architecture also allows balancing the task loads by prioritizing the threads and by allocating more workers for a crucial task.

As said earlier threads communicate with each other asynchronously and therefore there would be no need that they start and finish computations at the same time. As far as all threads are performing almost at the same speed, the system is stable and is able to grab and draw frames between 20 and 25 fps, which, as we experienced, is just a little below inference rate of more efficient network architectures as Mobilenet. Figure 24 is illustrating the sequence diagram for the proposed thread communications.

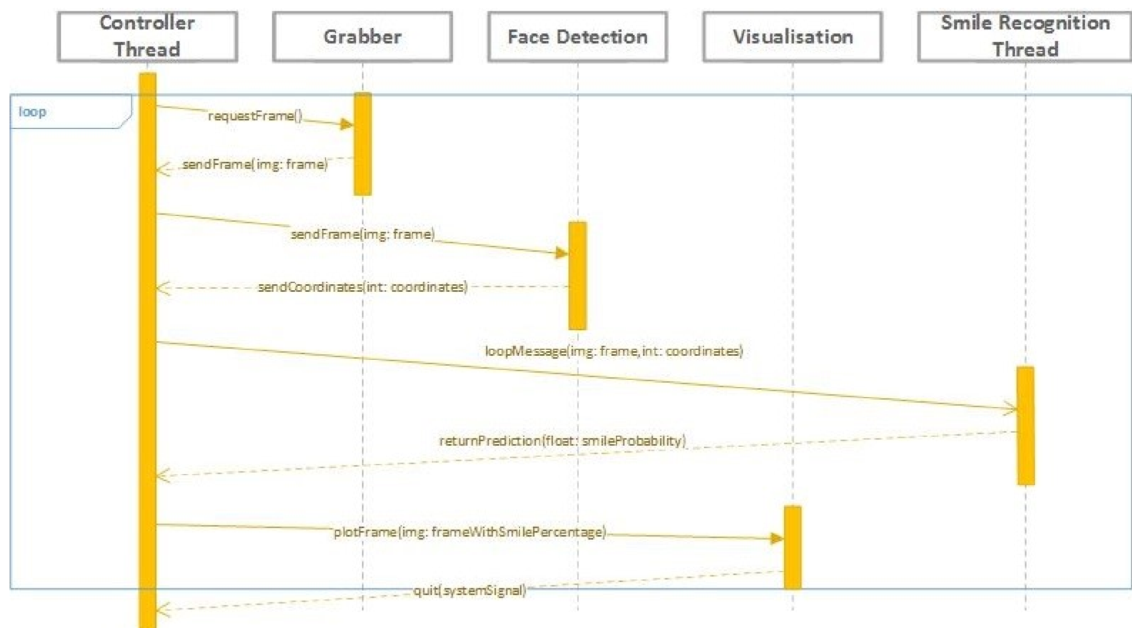


Figure 24. *Sequence diagram for the system architecture.*

4. EXPERIMENTS AND RESULTS

4.1 Datasets

Analysing people's postures and facial expressions is considered as a heavy computer vision or machine learning (ML) task which requires many distortions to be taken care of since human body and facial arrangements make a vast space area: changes in illumination, blocking, changing the viewpoints, or deformations, etc and that is why the field of Computer Vision and Machine Learning has grasped so much attention from researchers recently. But before further investigation, we need information and its where we understand the importance of datasets. A dataset is a collection of data which is gathered for a particular use. It should also come proper documentation files well describing its usage. We mainly used three different face image datasets in our experiments: 1) ChaLearn Looking at People (LAP) 2016 dataset[1], 2) GENKI-4K dataset[2], and 3) CelebFaces Attributes dataset[3].

4.1.1 LAP

ChaLearn team claims that 'Looking at People' was a challenging research area since it mainly focuses on human detection and recognition in images, including detection of different human body parts, recognizing actions or gestures from images or motion pictures, and taking into account multimodal data. Field of 'Looking at People' is associated with any case of visual human analysis.

Multiple sub-fields have been defined recently within the LAP like Affective Computing, Social Signal Processing, Human Behavior Analysis, or Social Robotics. The effort that has been made in this field will pay off shortly since it has a prospective future and even currently it has many potential applications in markets such as TV production, analysis of multimedia content, educational purposes, researches in sociology, security and surveillance, giving automatic assistance, monitoring, etc.

The subset of the LAP dataset that we used is annotated images of face initially presented for the 'Looking at People' competition in ICCV2015, which is also open for public use. We used the CVPR 2016 version of that dataset which contains 7,591 images. Facial images in this dataset in comparison to the similar datasets are captured in unconstrained environments, therefore their backgrounds are more natural and diverse.

---

[1] http://chalearnlap.cvc.uab.es/
[2] http://mplab.ucsd.edu
[3] http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

4.1.2 GENKI-4K

GENKI database from the MPLab is a growing collection containing face images with various ethnic groups, geographical places, personal identities, and illumination environments. Every updated version includes all images from the previous versions, therefore it is backward compatible. This dataset is partitioned to overlapping subspaces and every one of them has its specific descriptions and labels. For instance, the GENKI-4K batch has four thousand images of faces categorized by human as ‘smiling’ or ‘not smiling’. Almost all faces’ postures found to be frontal with help of an automatic face detector. The GENKI-SZSL batch has 3500 face images which have information about faces location and size. All the images are allowed for public use.

In our experiments, we used GENKI-4K dataset which, as mentioned, is a subset of MPLab GENKI image dataset. This dataset is including 4,000 images of faces which are subject to vast selection of illumination, facial appearance, geographical places, background environments, and camera types. Images are categorized into two categories of ‘smiling’ and ‘not smiling’ and they also have information on the Head posture and rotation (yaw, pitch, and roll parameters, in radians).

4.1.3 CelebFaces

CelebFaces Attributes Dataset (CelebA) is a huge dataset of face postures which has more than 200,000 images from celebrities worldwide, and every image has annotations for 40 characteristics. CelebA dataset contains images spanning a wide range of postures and backgrounds. This dataset is truly diverse, huge, and rich in terms of annotations. It contains 202,599 images from faces, 10,177 different identities, and 5 landmark locations. Each image has 40 binary attributes annotations and can be used for computer vision problems such as facial expression recognition, face detection, and landmark localization. The dataset is allowed only for non-commercial research purposes.

MMLAB declares that all images in this dataset are retrieved from the Internet, therefore images are not owned by the MMLAB and it is not responsible for the content nor the meaning of these images.

4.2 Setup

For implementing our system, we employed ANN structures which were pre-trained on ImageNet dataset. For this purpose, we removed the last layer from the pre-trained network and replaced it with a fully connected layer, a dense layer with sigmoid activation function allowing a binary classification. We experienced that binary classification produced slightly more accurate results.

Moreover, we performed data augmentation for generalizing the trained model more and better. The methods for augmenting the input images include zooming, shearing, and flipping horizontally.

4.3 Evaluation

For measuring our system's running speed in addition to the embedded platform Nvidia Jetson TX2 we deployed the system on two more platforms: Two desktop computers, one equipped with Intel Core i5-6200U CPU and 8G of RAM and another one with K40 Desktop GPU. We employed these three different measures (1) model size (number of parameters), (2) frames per second (FPS), and (3) the number of floating-point operations (FLO) for assessing the computational performance of our smile detector system utilizing the multiple network topologies.

Furthermore, we carried out performance of the system in terms of accuracy by means of two other metrics. First, the accuracy which is calculated as this:

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

and then AUC which is the area under the ROC curve. These two measures are effective ways of summarizing the accuracy of smile detection in our system.

4.4 Results

Table 3 is summarizing the running speed of the compared models on different platforms. Order of the tested networks in this table is by the number of required FPOs for a single frame to process.

Table 4. *Running speed and network size of the experienced trained network architectures for smile detection. The network structures are ordered by the number of floating point operations per frame (FLO column).*

Network Model	FPS (Jetson)	FPS (CPU)	FPS (GPU)	Size $\times 10^6$	FLO $\times 10^9$
Mobilenet ($\alpha = 0.25, \rho = 0.714$)	**28.1**	**31.5**	**164**	**0.21**	**0.02**
Mobilenet ($\alpha = 0.25, \rho = 1$)	27.3	19.6	158	0.21	0.03
Mobilenet ($\alpha = 0.5, \rho = 0.714$)	26.1	18.5	159	0.83	0.07
Mobilenet ($\alpha = 0.5, \rho = 1$)	25.6	9.0	154	0.83	0.14
Mobilenet ($\alpha = 0.75, \rho = 0.714$)	24.2	12.2	153	1.8	0.16
Mobilenet ($\alpha = 1, \rho = 0.714$)	22.5	9	146.8	3.2	0.28
Mobilenet ($\alpha = 0.75, \rho = 1$)	23.5	6.5	149	1.8	0.31
XCeption	9.6	0.9	28	20.8	0.35
Mobilenet ($\alpha = 1, \rho = 1$)	22.3	4.9	146.1	3.2	0.55
Inception V3	7.9	2.9	22	21.8	2.8
ResNet-50	8.4	1.5	27	23.5	3.8
Inception-resnet V2	*	1.5	11	54.3	6.4
VGG-16	*	0.9	25	134	15

There are huge differences, as you have probably noticed, between models: VGG16 as the bulkiest network owns about 750 times more FPOs than the lightest variant of Mobilenet. Since processors are performing computation parallelization, it is noticeable that other performance measures are not so distinct from each other. Moreover, we were not able to execute huge models on Jetson TX2 device because of memory limits on this platform.

For enabling comparison between our approach in the implementation of this smile detection system and state of the art methods we had to think of more steps. For example, in the SmileNet [22] as a complete smile detection pipeline, face localization is also performed. Therefore, for assessing models within a complete pipeline, we add a ViolaJones face detector step before the smile detector stage. With this setup, the execution speed of our system on the K40 GPU would reach between 40 and 60 FPS, which is also depended on the input image resolution and is about twice faster than that of the SmileNet which was reported as 21.15 FPS[22]. It is true that our approach could not reach the accuracy shown for the state-of-the-art methods, such as the one shown in Table 5, but it should also be noticed that these algorithms are cannot run in real time or even are not able to run on low resource devices like Jetson TX2 embedded devices because of the computational power and memory limitations we have on these platforms.

Table 5. *Comparison of Mobilenet (α = 1; ρ = 1) with Smile-Net method in terms of accuracy*

	GENKI-4K	CelebA
SmileNet	95.76%	92.81%
Proposed model	93.6%	88.5%

Even on platforms with limited resources, it makes sense to perform face localization since it improves the accuracy of the smile detection procedure. Figure 25 is showing the smile detection accuracies, after locating faces, on the GENKI-4K dataset. Summary of these results was shown in Table 5, which was comparing the proposed method and Smile-Net on images from CelebA and GENKI-4K datasets. As it is clear the performance of the smile detection method from our experiments is so close to the recent detectors in terms of accuracy while it runs much faster than them. Here is good to emphasise this again that networks such as Smile-Net are not able to run smoothly or real-time on an embedded device because of their high memory consumption.

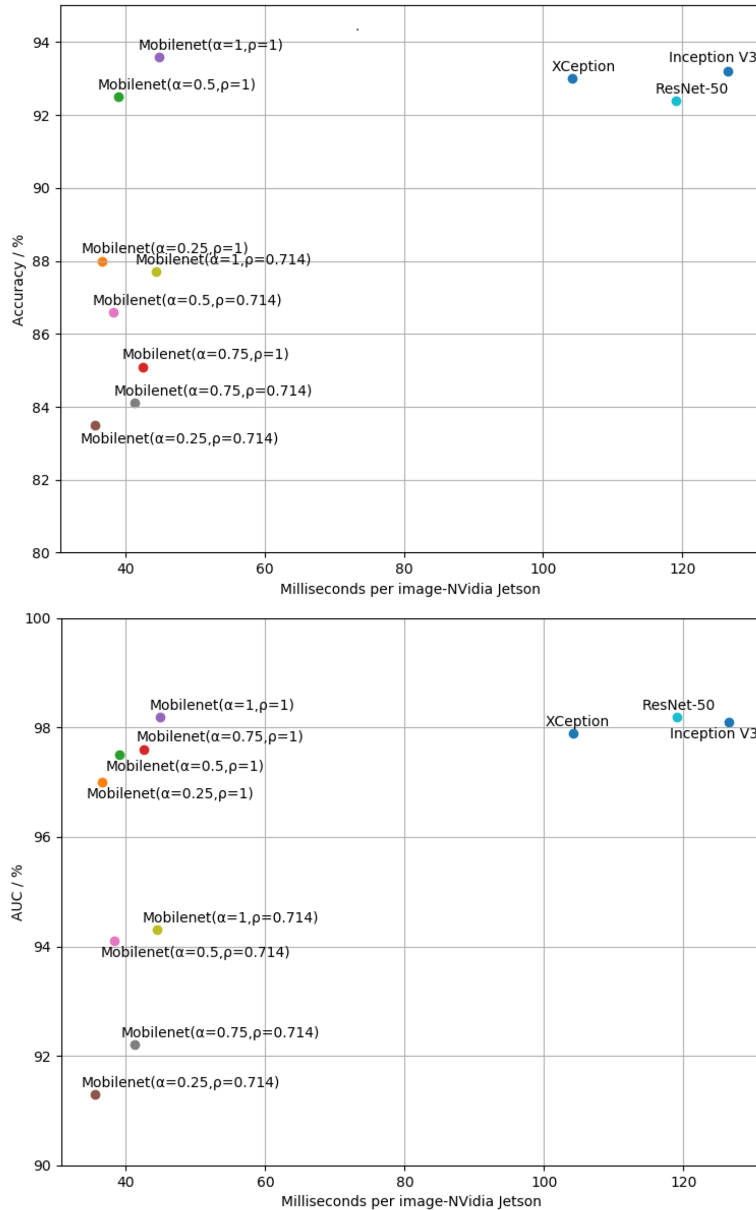


Figure 25.
ACC and AUC measures. They are drawn with respect to the running time for the studied models. Results from VGG-16, ACC: 91.2% and AUC: 90.5%, and Inception-resnet-V2, with ACC: 92.8% and AUC: 98.0%, models are not shown since their large memory consumption does not let us deploy them on NVidia Jetson-TX2.

5. CONCLUSIONS AND POTENTIAL FUTURE WORK

In this work, we have proposed suitable deep learning architectures for real-time deployment of computer vision related applications on low resource platforms such as NVidia Jetson TX2. We implemented and deployed a smile detector system as a use case of this purpose and assessed it by utilizing thirteen different deep CNN topologies and three image datasets. Particularly, we compared their execution speeds on three devices: Nvidia Jetson, a desktop CPU, and a desktop GPU. And at last, we studied the tradeoff between speed and accuracy of these thirteen models and a state-of-the-art method and showed that these models produce comparable results in terms of accuracies while they are running at significantly different speed because of the dramatic difference in their computational demand.

The ground truth in our experiments the could be considered rather noisy, i.e. smile is not easy to annotate because there are multiple stages between no smile and a full smile and as a result the importance of the differences in prediction accuracies are questionable. Considering everything, we could claim that the Mobilenets excel in accuracy/complexity tradeoff and the earlier CNN architectures, like VGG variants, or even recent networks, like SmileNet, should not be utilized in a real-time use cases.

Moreover, we implemented an asynchronous software system which allows easy expansion and of the entire system by simple integration with future detection modules. The proposed pipeline which originally was planned to be deployed on NVidia Jetson embedded platform would let us reach the execution time of 27.3 FPS on average. This is considered as real-time performance to humans' visual system.

We have considered integrating other detection modules, such as age and gender detection, with this system as the first future supplement. In addition to this, the next addition to the current system would be the integration of the smile detector into a software framework, such as PRUNE [53], that handles data transfers and synchronization between CPU and the GPU cores. Utilizing a framework like this would improve the software modularity and in addition, allows us to concentrate on the algorithms instead of multiprocessing during the development stage. This would also enable even higher gains in processing time on a platform specifically designed for streaming data processing. Another potential future addition in this research would be attempting to reduce the encoding precision of neural network weights or activations, e.g. by having a reduction in the number of bits or employing fixed point instead of floating points. Matrix multiplication operations could be quite expensive to implement and need several logic gates. So, devices with limited

resources might not be able to implement floating point operations efficiently. It is obvious that using, for example, 8-bit values instead of 32-bit ones would reduce the memory consumption and computation costs. Therefore, for weakening the mentioned issue, we can compress and map the floating-point input or output values into a fixed-point representation.

According to our smile detector application, the proposed approach would provide almost even three times faster average processing times on GPU than the reported state–off–the–art smile detectors. Besides, the smile detection accuracy of our method is close to that of the state–off–the–art methods.

REFERENCES

[1] F. Chollet, Deep Learning with Python, Manning Publications Co., 361 pages, 2018.

[2] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer, SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size, 2016, .

[3] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, 2017, .

[4] A.S. Patwardhan, G.M. Knapp, Multimodal Affect Analysis for Product Feedback Assessment, 2017, .

[5] S. Turabzadeh, H. Meng, R. Swash, M. Pleva, J. Juhar, Facial Expression Emotion Detection for Real-Time Embedded Systems, Technologies, Vol. 6, Iss. 1, 2018, pp. 17.

[6] P. Ghazi, A.P. Happonen, J. Boutellier, H. Huttunen, Embedded Implementation of a Deep Learning Smile Detector, 2018, .

[7] P. Viola, M.J. Jones, Robust Real-Time Face Detection, International Journal of Computer Vision, Vol. 57, Iss. 2, 2004, pp. 137-154. Available (accessed ID: Viola2004): https://doi.org/10.1023/B:VISI.0000013087.49260.fb.

[8] J. Chatrath, P. Gupta, P. Ahuja, A. Goel, S.M. Arora, Real time human face detection and tracking, 2014 International Conference on Signal Processing and Integrated Networks (SPIN), IEEE, pp. 705-710.

[9] A. Tsai, T. Lin, T. Kuan, K. Bharanitharan, J. Chang, J. Wang, An efficient smile and frown detection algorithm, 2015 International Conference on Orange Technologies (ICOT), IEEE, pp. 139-143.

[10] N. Kazanskiy, V. Protsenko, P. Serafimovich, Performance analysis of real-time face detection system based on stream data mining frameworks, Procedia Engineering, Vol. 201, 2017, pp. 806-816.

[11] C. Garcia, M. Delakis, Convolutional face finder: a neural architecture for fast and robust face detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 26, Iss. 11, 2004, pp. 1408-1423.

[12] N. Farrugia, F. Mamalet, S. Roux, F. Yang, M. Paindavoine, Fast and Robust Face Detection on a Parallel Optimized Architecture Implemented on FPGA, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 19, Iss. 4, 2009, pp. 597-602.

[13] F. Mamalet, S. Roux, C. Garcia, Real-Time Video Convolutional Face Finder on Embedded Platforms, EURASIP Journal on Embedded Systems, Vol. 2007, Iss. 1, 2007, pp. 1-8.

[14] S. Yang, P. Luo, C.C. Loy, X. Tang, Faceness-Net: Face Detection through Deep Facial Part Responses, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017, pp. 1.

[15] H. Jiang, E. Learned-Miller, Face Detection with the Faster R-CNN, 2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017), IEEE, pp. 650-657.

[16] L. Chi, H. Zhang, M. Chen, End-To-End Face Detection and Recognition, 2017, .

[17] M. Arsenovic, S. Sladojevic, A. Anderla, D. Stefanovic, FaceTime - Deep learning based face recognition attendance system, 2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY), IEEE, pp. 53.

[18] Y. Byeon, S. Pan, S. Moh, K. Kwak, A surveillance system using CNN for face recognition with object, human and face detection, Lecture Notes in Electrical Engineering, pp. 975-984.

[19] G. Hu, Y. Yang, D. Yi, J. Kittler, W. Christmas, S.Z. Li, T. Hospedales, When Face Recognition Meets with Deep Learning: An Evaluation of Convolutional Neural Networks for Face Recognition, 2015 IEEE International Conference on Computer Vision Workshop (ICCVW), IEEE, pp. 384-392.

[20] M. Uricar, R. Timofte, R. Rothe, J. Matas, L. Van Gool, Structured Output SVM Prediction of Apparent Age, Gender and Smile from Deep Features, 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE, pp. 730-738.

[21] Y. Xia, D. Huang, Y. Wang, Detecting Smiles of Young Children via Deep Transfer Learning, 2017 IEEE International Conference on Computer Vision Workshops (ICCVW), IEEE, pp. 1673-1681.

[22] Y. Jang, H. Gunes, I. Patras, SmileNet: Registration-Free Smiling Face Detection In The Wild, 2017 IEEE International Conference on Computer Vision Workshops (ICCVW), IEEE, pp. 1581-1589.

[23] R. Ranjan, V.M. Patel, R. Chellappa, HyperFace: A Deep Multi-task Learning Framework for Face Detection, Landmark Localization, Pose Estimation, and Gender Recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017, pp. 1.

[24] S. Zhang, X. Zhu, Z. Lei, H. Shi, X. Wang, S.Z. Li, FaceBoxes: A CPU Real-time Face Detector with High Accuracy, 2017, .

[25] S. Tripathi, G. Dane, B. Kang, V. Bhaskaran, T. Nguyen, LCDet: Low-Complexity Fully-Convolutional Neural Networks for Object Detection in Embedded Systems, 2017

IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE, pp. 411-420.

[26] B. Amos, B. Ludwiczuk and M. Satyanarayanan, ”OpenFace:
A general-purpose face recognition library with mobile applications”, CMU-CS-16-118, CMU School of Computer Science, 2016, https://cmusatyalab.github.io/openface/, [Accessed: 8-Feb-2018].

[27] T. Guha, Z. Yang, A. Ramakrishna, R.B. Grossman, D. Hedley, S. Lee, S.S. Narayanan, On quantifying facial expression-related atypicality of children with Autism Spectrum Disorder, 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, United States, pp. 803-807.

[28] Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey, https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812115.

[29] I. Goodfellow, Y. Bengio, A. Courville, Deep learning, MIT Press, Cambridge, MA, 2017, 775 page p.

[30] En.wikipedia.org. “Cross-correlation”. Available at: https://en.wikipedia.org/wiki/Cross-correlation [Accessed 22 Aug. 2018].

[31] Cs231n.github.io. “CS231n Convolutional Neural Networks for Visual Recognition”. Available at: http://cs231n.github.io/convolutional-networks/ [Accessed 22 Aug. 2018].

[32] En.wikipedia.org. “Backpropagation”. Available at: https://en.wikipedia.org/wiki/Backpropagation [Accessed 22 Aug. 2018].

[33] Medium. “The Vanishing Gradient Problem – Anish Singh Walia – Medium”. Available at: https://medium.com/@anishsingh20/the-vanishing-gradient-problem-48ae7f501257 [Accessed 22 Aug. 2018].

[34] En.wikipedia.org. “Convolutional neural network”. Available at: https://en.wikipedia.org/wiki/Convolutional_neural_network [Accessed 22 Aug. 2018].

[35] Towards Data Science. “Activation Functions: Neural Networks – Towards Data Science”. Available at: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 [Accessed 22 Aug. 2018].

[36] Medium. “Why MobileNet and Its Variants (e.g. ShuffleNet) Are Fast”. Available at: https://medium.com/@yu4u/why-mobilenet-and-its-variants-e-g-shufflenet-are-fast-1c7048b9618d [Accessed 22 Aug. 2018].

[37] Eli.thegreenplace.net. “Depthwise separable convolutions for machine learning - Eli Bendersky's website”. Available at: https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning/ [Accessed 22 Aug. 2018].

[38] Towards Data Science. “A Simple Guide to the Versions of the Inception Network”. Available at: https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202 [Accessed 22 Aug. 2018].

[39] Medium. “Understanding and Implementing Architectures of ResNet and ResNeXt for state-of-the-art Image…”. Available at: https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624 [Accessed 22 Aug. 2018].

[40] K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014, .

[41] Towards Data Science. “Applied Deep Learning - Part 4: Convolutional Neural Networks”. Available at: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2 [Accessed 22 Aug. 2018].

[42] Towards Data Science. “Neural Network Architectures – Towards Data Science”. Available at: https://towardsdatascience.com/neural-network-architectures-156e5bad51ba [Accessed 22 Aug. 2018].

[43] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the Inception Architecture for Computer Vision, 2015, .

[44] Google AI Blog. “Improving Inception and Image Classification in TensorFlow”. Available at: https://ai.googleblog.com/2016/08/improving-inception-and-image.html [Accessed 22 Aug. 2018].

[45] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, pp. 770-778.

[46] C. Szegedy, S. Ioffe, V. Vanhoucke, A. Alemi, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, 2016, .

[47] F. Chollet, Xception: Deep Learning with Depthwise Separable Convolutions, 2016, .

[48] Vitalab.github.io. “Xception: Deep Learning with Depthwise Separable Convolutions”. Available at: https://vitalab.github.io/deep-learning/2017/03/21/xception.html [Accessed 22 Aug. 2018].

[49] Keras.io. ”Why use Keras - Keras Documentation”. Available at: https://keras.io/why-use-keras/ [Accessed 22 Aug. 2018].

[50] Y. Sun, X. Wang, X. Tang, Deep Convolutional Network Cascade for Facial Point Detection, 2013 IEEE Conference on Computer Vision and Pattern Recognition, pp. 3476-3483.

[51] V. Kazemi, J. Sullivan, One millisecond face alignment with an ensemble of regression trees, 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1867-1874.

[52] Y. Bai, S.S. Bhattacharyya, A.P. Happonen, H. Huttunen, Elastic Neural Networks: A Scalable Framework for Embedded Computer Vision, EUSIPCO, 2018, .

[53] J. Boutellier, J. Wu, H. Huttunen, S.S. Bhattacharyya, PRUNE: Dynamic and Decidable Dataflow for Signal Processing on Heterogeneous Platforms, IEEE Transactions on Signal Processing, Vol. 66, Iss. 3, 2018, pp. 654-665.