Report Titled

SWIFTY DRAW: A SKETCH RECOGNITION GAME

Submitted

in partial fulfilment of the requirements of the degree of

Bachelor of Technology

(Information Technology)

by

Adway Kirwe(151080974)

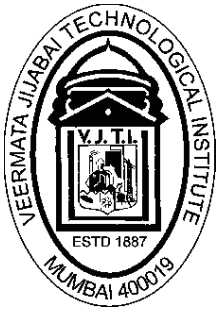
Kaustubh Phulkar (141080011)

Pramod Choudhari (141080065)

Prasad Shirvandkar (151080970)

Under the guidance of

Dr. V.B.Nikam



Department of Computer Engineering and Information Technology

Veermata Jijabai Technological Institute

Mumbai 400019

May 2018

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources.

We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea / data / fact / source in our submission.

We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

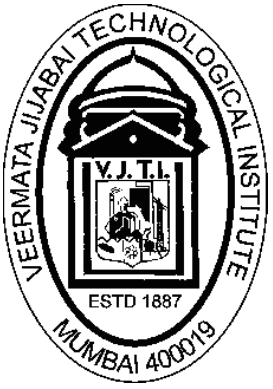
Adway Kirwe	Kaustubh Phulkar	Pramod Choudhari	Prasad Shirvandkar
151080974	141080011	141080065	151080970

Date - May 2018

DEPARTMENT OF COMPUTER ENGINEERING AND INFORMATION TECHNOLOGY

VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE

MUMBAI - 19, INDIA



CERTIFICATE

This is to certify that **Adway Kirwe[151080974], Kaustubh Phulkar[141080011], Pramod Choudhari[141080065], Prasad Shirvandkar[151080970]**, the students of **Bachelor of Technology in Information Technology**, have completed the report entitled "**SWIFTY DRAW: A SKETCH RECOGNITION GAME**" to our satisfaction.

Dr. V.B.Nikam
Associate Professor
Project Guide
Date - May 2018

Dr.V.B.Nikam
Head
Department of CE & IT
Place: VJTI, Mumbai

Abstract

Traditional methods on sketch recognition generally follows the conventional image classification paradigm that is, extracting hand-crafted features from sketch images followed by feeding them to a classifier. Most hand-crafted features traditionally used for photos such as HOG, SIFT and shape context, have been employed, which are often coupled with bag of visual words (BoW) to yield a final feature representation that can then be classified.

However, existing hand-crafted features designed for photos do not account for the unique traits of sketches. More specifically, they ignore two key unique characteristics of sketches. First, a sketch is essentially an ordered list of strokes; they are thus sequential in nature. In contrast with photos that consist of pixels sampled all at once, a sketch is the result of an online drawing process. Second, free-hand sketches can be highly abstract and iconic and, coupled with varying drawing skills among different people, intra-class structure and appearance variations are often considerably larger than photos.

Existing hand-crafted features such as HOG and classifiers such as SVMs are suited neither for capturing this large variation of abstraction and appearance variations, nor exploiting the ordered stroke structure of a sketch. But these features are sensitive to the view perspectives and some appearance cues of drawn sketches. Furthermore, the ability of learning algorithms to train the classification models are also influenced by the handcrafted features and the capacity of classifiers like SVM to memorize feature information. Also, most of the sketch recognition system using advanced technology for classifying hand drawn sketches are just good at classifying or recognizing what is being drawn and improve the drawing with predefined sketch model but hardly any systems are developed that carries the power of hand drawn sketch classification to showcase that it can be used in a fun way or in a form of a game. A game containing game logic with sketch recognition system using machine learning frameworks.

There does not exist a system that rewards the user on successful classification and uses a sequence of sketches as task to be performed by user and make the user work for it in a fun way. Convolutional Neural Network has been a goto framework for feature representation and recognition and for a dataset like hand drawn sketches, it is important for the small features to be correctly identified. Traditional CNN having multiple advantages in terms on performance and accuracy. However, it is not very good when it comes to portability i.e. the trained model size is high and time for execution.

Portability is the biggest factor when it comes to using trained model for classifying data on portable devices like smartphones because of its limited computational resources. To overcome the barrier of portability, a different type of CNN has emerged as the favourite framework that uses the power of Convolutional Deep Neural Network at minimal effect on accuracy with high portability and less execution time, which is named as Depthwise Separable Convolution. The Depthwise Separable Convolution has become the easy choice for image classification to be used on portable devices like smartphones.

The current state-of-the-art is the MobileNet Model which uses Depthwise Separable Convolution for classification and feature extraction at much lower cost than traditional or standard CNN.

Acknowledgements

We would like to thank our project guide Dr.V.B.Nikam for contributing his time and effort to help us with our project. His suggestions and guidelines have been of a great help during the entire course of our project. He has always been involved by discussing our project at each phase to make sure that the experiment is designed and carried out in an appropriate manner and that our conclusions are appropriate, given their results. His constant support and interaction have been a driving force which has constantly motivated us to explore the different aspects of our project.

Contents

List of Figures

Chapter 1

Introduction

Sketches are very intuitive and have been long used as an effective communication channel. It can be seen as an abstract representation of collection of ideas and well structured gestures.

Sketch Recognition attempts to recognize the intent of the user what the user is trying to draw while allowing the user to draw in unconstrained manner and target s on classifying human drawn sketches into categories. As a sub-task of object recognition, it remains a degree of uniqueness because the sketches usually lack color and texture information compared to photos, which makes classification a non-trivial problem.

Recognizing free hand human sketch is an extremely difficult / challenging task. This is due to no of reasons:

1. Sketches are highly iconic and abstract.

2. Different Human have different perspective and style of the same object.

The Objective is to train machines to recognize and understand abstract human concepts / metaphor in similar manner to how human does it and using the same understanding of sketch in a game that can be played.

There are many applications available that uses the power of machine learning for recognizing human drawn sketches with the aim to make them easy to understand and

help human to get the job done and very few works have been done that try to use the power of machine learning in a game format.

In this project, we used the power of sketch recognition in an interactive sequence of frames designed to be played in a fun way using a sequence of tasks and points rewarded for the task. The game basically is a set of this sketch tasks that a user needs to be completed in time and the neural network tries to guess the sketch. If it finds it to be right, then user is rewarded with points based on what how the sketch is drawn.

Convolutional neural networks have become a standard in computer vision ever since AlexNet popularized deep convolutional network by winning the ImageNet Challenge in 2012. The trend since then is to develop or make deeper and more complicated neural networks for higher accuracy and minimize error as much as possible. This is why CNN is the go-to choice for recognizing different characteristics of human drawn sketches.

We have proposed a system which is able to recognize various human drawn sketches using Neural Network giving accurate results and developed an interactive system that communicates with the users and rewards them for their work.

The Motive is to showcase that machine learning is not only used for solving real world problems in dynamic way but can also be used in a fun way.

Chapter 2

Literature Review

Convolutional Neural Network (Also called ConvNets or CNNs) are a category of Neural Networks that have proven very effective in areas such as image recognition and classification. ConvNets have been successful in identifying faces, objects and Traffic signs apart from powering vision in robots and self-driving cars.

With the advent of autonomous driving, image-related tasks are now quite common in data science workflows. CNNs are algorithms for transforming an image into some output, like a category (e.g., "picture contacting a person" or "dogs playing on beach"). Intelligent photo storage and indexing tools can use CNNs to determine which photos in your library contain images of you with your dog. Automatic Teller Machines (ATMs) can use CNNs to determine the amount of money a handwritten check is for. Autonomous vehicles can use advanced versions to determine whether a sign down the road is indicating "Stop" or "Silent Zone".

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

2.1 Basic building blocks of Convolutional Neural Networks

A CNN architecture is formed by a stack of distinct layers that transform the input volume into an output volume (e.g. holding the class scores) through a differentiable function. A Convolutional Neural Network consists of following five basic components :

1. Convolutional Layer
2. Pooling Layer
3. ReLu Layer
4. Fully connected Layer
5. Loss Layer

2.1.1 Convolutional Layer

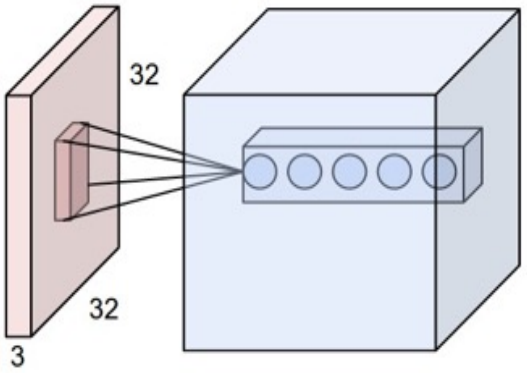


FIGURE 2.1: Convolution layer

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

2.1.1.1 Local Connectivity

When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume because such a network architecture does not take the spatial structure of the data into account. Convolutional networks exploit spatially local correlation by enforcing a local connectivity pattern between neurons of adjacent layers: each neuron is connected to only a small region of the input volume. The extent of this connectivity is a hyperparameter called the receptive field of the neuron. The connections are local in space (along width and height), but always extend along the entire depth of the input volume. Such an architecture ensures that the learnt filters produce the strongest response to a spatially local input pattern.

2.1.1.2 Hyperparameters

Three hyper-parameters control the size of the output volume of the convolutional layer:-

1. Depth : The depth of the output volume controls the number of neurons in a layer that connect to the same region of the input volume. These neurons learn to activate for different features in the input. For example, if the first convolutional layer takes the raw image as input, then different neurons along the depth dimension may activate in the presence of various oriented edges, or blobs of color. The number of filters decide the depth of the output volume.
2. Stride : Stride controls the width and height of the output volume. When the stride is 1 then we move the filters one pixel at a time. This leads to heavily overlapping receptive fields between the columns, and also to large output volumes. When the

stride is 2 (or rarely 3 or more) then the filters jump 2 pixels at a time as they slide around. The receptive fields overlap less and the resulting output volume has smaller spatial dimensions. Thus, using small strides can help preserve input features but increases the computation cost whereas large strides can help reduce computation cost but may result is loss of input features.

3. Zero-padding : Sometimes it is convenient to pad the input with zeros on the border of the input volume. The size of this padding is a third hyperparameter. Zero-padding helps to preserve features at the edges and the corner of the images. Generally, a padding of size 1 or 2 is used in most of the cases.

The spatial size of the output volume can be computed as a function of the input volume size **W**, the kernel field size of the Conv Layer Neurons **K**, the stride with which they are applied **S**, and the amount of zero padding **P** used on the border. The formula for calculating how many neurons "fit" in a given volume is given by

$$(W - K + 2P)/S + 1 \tag{2.1}$$

. If this number is not an integer, then the strides are set incorrectly and the neurons cannot be tiled to fit across the input volume in a symmetric way. In general, setting zero padding to be

$$P = (K - 1)/2 \tag{2.2}$$

when the stride is **S=1** ensures that the input volume and output volume will have the same size spatially. Though it's generally not completely necessary to use up all of the neurons of the previous layer, for example, you may decide to use just a portion of padding.

2.1.1.3 Parameter Sharing

A parameter sharing scheme is used in convolutional layers to control the number of free parameters. It relies on one reasonable assumption: That if a patch feature is

useful to compute at some spatial position, then it should also be useful to compute at other positions. In other words, denoting a single 2-dimensional slice of depth as a depth slice, we constrain the neurons in each depth slice to use the same weights and bias.

Since all neurons in a single depth slice share the same parameters, then the forward pass in each depth slice of the CONV layer can be computed as a convolution of the neuron's weights with the input volume (hence the name: convolutional layer). Therefore, it is common to refer to the sets of weights as a filter (or a kernel), which is convolved with the input. The result of this convolution is an activation map, and the set of activation maps for each different filter are stacked together along the depth dimension to produce the output volume. Parameter sharing contributes to the translation invariance of the CNN architecture.

Sometimes the parameter sharing assumption may not make sense. This is especially the case when the input images to a CNN have some specific centered structure, in which we expect completely different features to be learned on different spatial locations. One practical example is when the input are faces that have been centered in the image: we might expect different eye-specific or hair-specific features to be learned in different parts of the image. In that case it is common to relax the parameter sharing scheme, and instead simply call the layer a locally connected layer.

2.1.2 Pooling Layer

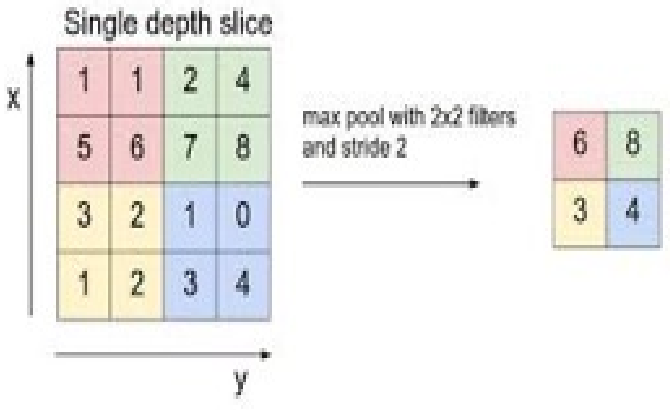


FIGURE 2.2: Pooling layer

Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. The intuition is that the exact location of a feature is less important than its rough location relative to other features. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network, and hence to also control overfitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture. The pooling operation provides another form of translation invariance.

The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 down samples at every depth slice in the input by 2 along both width and height, discarding 75% of the activations. In this case, every max operation is over 4 numbers. The depth dimension remains unchanged.

In addition to max pooling, the pooling units can use other functions, such as average pooling or L2-norm pooling. Average pooling was often used historically but has recently fallen out of favor compared to max pooling, which works better in practice.

More generally, the pooling layer:

1. Accepts a volume of size $W_1 H_1 D_1$
2. Requires two hyperparameters:
   (a) their spatial extent F,
   (b) the stride S,
3. Produces a volume of size $W_2 H_2 D_2$ where:
   (a) $W_2 = (W_1 F)/S + 1$
   (b) $H_2 = (H_1 F)/S + 1$
   (c) $D_2 = D_1$
4. Introduces zero parameters since it computes a fixed function of the input

5. Note that it is not common to use zero-padding for Pooling layers

It is worth noting that there are only two commonly seen variations of the max pooling layer found in practice: A pooling layer with F=3, S=2(also called overlapping pooling), and more commonly F=2, S=2. Pooling sizes with larger receptive fields are too destructive

2.1.3 ReLu (Rectified Linear Unit) Layer :

ReLu refers to the Rectifier Unit, the most commonly deployed activation function for the outputs of the CNN neurons.The ReLU is the most used activation function in the world right now.Since, it is used in almost all the convolutional neural networks or deep learning.The function and its derivative both are monotonic.

But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately. Mathematically, it's described as:

$$f(x) = max(0, x) \tag{2.3}$$

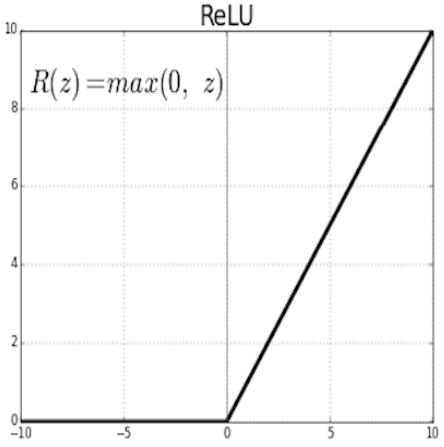


FIGURE 2.3: ReLu activation

It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

Other functions are also used to increase nonlinearity, for example :

1. Saturating hyperbolic tangent:

   Tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped).The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.The function is differentiable.

   The function is monotonic while its derivative is not monotonic.The tanh function is mainly used classification between two classes. Tanh activation functions are used in feed-forward nets.

$$f(x) = |tanh(x)| \tag{2.4}$$

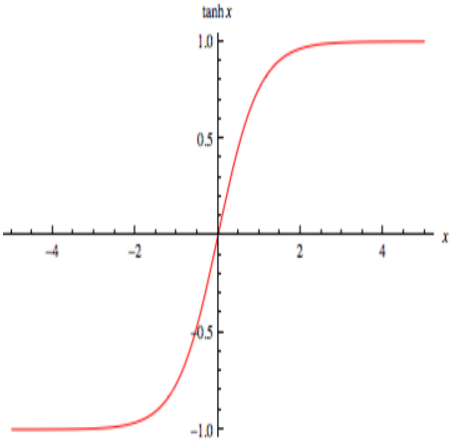


FIGURE 2.4: Tanh activation

2. Sigmoid function : The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output.Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.The function is differentiable.That means, we can find the slope of the sigmoid curve at any two points.The function is monotonic but

function's derivative is not. After several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

$$f(x) = 1/(1 + e^{(} - x)) \tag{2.5}$$

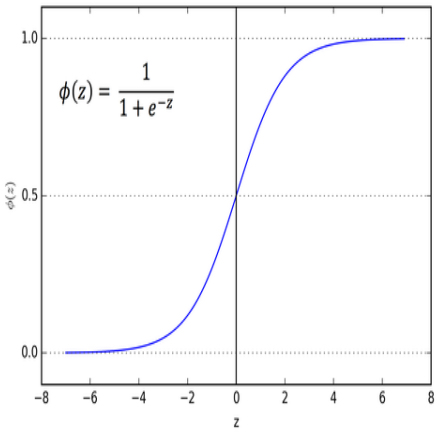


FIGURE 2.5: Sigmoid activation

2.1.4 Fully Connected Layer

The Fully Connected layer is configured exactly the way its name implies: it is fully connected with the output of the previous layer. Fully-connected layers are typically used in the last stages of the CNN to connect to the output layer and construct the desired number of outputs.A linear operation in which every input is connected to every output by a weight (so there are n-inputs * n-outputs weights). Generally followed by a non-linear activation function

2.1.5 Loss Layer

The loss layer specifies how training penalizes the deviation between the predicted and true labels and is normally the final layer. Various loss functions appropriate for different tasks may be used there.SoftMax loss is used for predicting a single class of K

mutually exclusive classes. Sigmoid cross-entropy loss is used for predicting K independent probability values in [0,1].

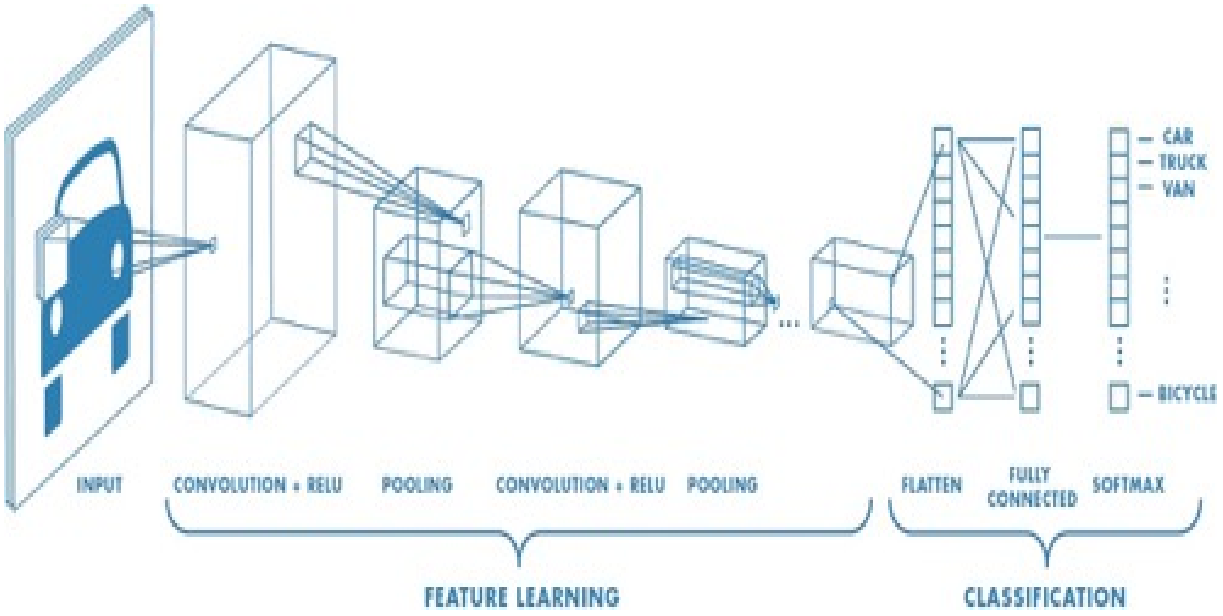


FIGURE 2.6: General Structure of Convolutional Neural Network

2.2 Choosing hyperparameters

If hyperparameters are tuned correctly then the ConvNet Model will give accurate results both on training as well as validation set. Following hyperparameters should be considered while optimizing the Convolutional Neural Net model :

2.2.1 Number of filters

Since feature map size decreases with depth, layers near the input layer will tend to have fewer filters while higher layers can have more. To equalize computation at each layer, the feature x pixel position product is kept roughly constant across layers. Preserving more information about the input would require keeping the total number of

activations (number of feature maps times number of pixel positions) non-decreasing from one layer to the next.

Number of Filters Depends upon number of training examples available and the complexity of the task. If the complexity of the task is high (for example, Object detection) then in that case more filters are required since number of features to be detected are more. Generally, as we go deeper into the model the depth of filter goes on increasing.

2.2.2 Filter shape

The shape of filter varies from dataset to dataset. Generally, the shape (i.e. width and height) of the filter is decreased gradually as we go deeper into the model.

2.2.3 Max pooling size

Typical values are 2x2. Very large input volumes may warrant 4x4 pooling in the lower-layers. However, choosing larger shapes will dramatically reduce the dimension of the signal, and may result in excess information loss. Often, non-overlapping pooling windows perform best.

2.2.4 Dropout for regularization

Dropout is a preferable regularization technique to avoid overfitting in deep neural networks. The method simply drops out units in neural network according to the desired probability. A default value of 0.5 is a good choice to test with

2.2.5 Learning rate

Learning rate controls how much to update the weight in the optimization algorithm. We can use fixed learning rate, gradually decreasing learning rate, momentum-based methods or adaptive learning rates, depending on our choice of optimizer such as

SGD, Adam, Adagrad, AdaDelta or RMSProp. AdaDelta is the most used optimizer in modern day use.

2.2.6 Number of epochs

The number epochs depend on dataset. Using very high value of epoch number may lead to overfitting and small value of epoch number may lead to underfitting. If we have very large dataset then we can used bigger value of epoch number to fit the model on the dataset.

2.2.7 Batch size

Mini-batch is usually preferable in the learning process of convnet. A range of 16 to 128 is a good choice to test with. We should note that convnet is sensitive to batch size. If the model is overfitting on the dataset the reducing the batch size may prevent overfitting and vice versa.

2.3 Convolution Models

2.3.1 Standard Convolution

A standard convolutional layer takes as input a $D_F\ X\ D_F\ X\ M$ feature map F and produces a $D_G\ X\ D_G\ X\ N$ feature map G where D_F is the width and height of a square input feature map, M is the number of input channels, D_G is the width and height of a square output feature map and N is the number of output. Convolution on kernel K of shape $D_K\ X\ D_K\ X\ M$ with input feature map F resulted in output of $D_G\ X\ D_G\ X\ 1$. If N such kernels are applied on input, it produces an output volume G of shape $D_G\ X\ D_G\ X\ N$ where D_G is width and height of an assumed square output. The standard convolutional layer is parameterized by convolution kernel K of size $D_K\ X\ D_K\ X\ M\ X\ N$ where D_K is the dimension of the kernel assumed to be square. The output feature map for standard convolution assuming stride one and padding is computed as:

Standard convolutions computational complexity:

1. Single convolution: $D_K\ X\ D_K\ X\ M$
2. Convolution of 1 kernel over input feature map F: $D_G\ X\ D_G\ X\ D_K\ X\ D_K\ X\ M$
3. Convolution of N kernels over input feature map F: $N\ X\ D_G^2\ X\ D_G^2\ X\ M$

where the computational cost depends multiplicatively on the number of input channels M, the number of output channels N, the kernel size $D_K\ X\ D_K$, the output feature map G size $D_G\ X\ D_G$.

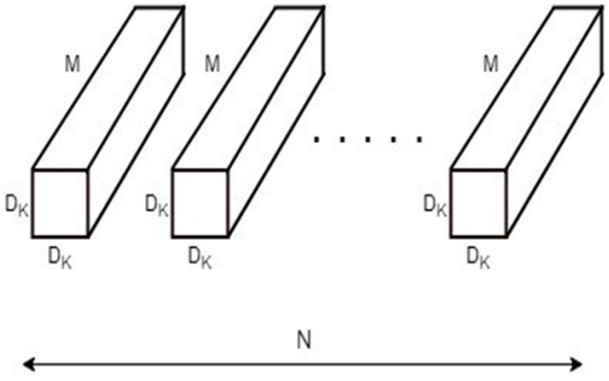


FIGURE 2.7: Standard Convolution Filters

2.3.2 Depthwise Separable Convolution

Depthwise Separable Convolution address each standard convolution heavy computational terms and their interactions. First it uses depthwise convolutions to break the interaction between the number of output channels and the size of the kernel and then it uses pointwise convolutions to combine the broken interactions in additive manner resulting in reduced multiplicative cost of standard convolution.

Depthwise separable convolution steps:

1. Depthwise convolution It takes as input a $D_F X D_F X M$ feature map F where D_F is the width and height of assumed square input feature map. Convolution on kernel K of shape $D_K X D_K X 1$ over input feature map F of shape $D_F X D_F X M$ results in

output feature map G of size $D_G X D_G X M$ where DF is the width and height of input image, D_G is the width and height of output image and M is the number of input channels

2. Pointwise convolution It takes input from depthwise convolution's output feature map as input feature map G of shape $D_G X D_G X M$. The filter used in pointwise convolution is $1X1XM$ which is basically 1X1 convolution operation over all M layers. If N such filters are applied on input feature map F, it results output feature map G of shape $D_G X D_G X N$.

Depthwise convolution computational complexity:

1. Depthwise operation (Filtering)
   (a) Single convolution: $D_K X D_K$
   (b) Convolution of 1 kernel over input feature map F for 1 channel: $D_G \ X \ D_G \ X \ D_K \ X \ D_K$
   (c) Convolution of 1 kernel over input feature map for M channels: $D_G \ X \ D_G \ X \ D_K \ X \ D_K \ X \ M$
2. Pointwise operation (Combining)
   (a) Single convolution: M
   (b) Convolution of 1 kernel over input feature map for 1 channel: $D_G \ X \ D_G \ X \ M$
   (c) Convolution of N kernels over input feature map for M channels: $N \ X \ D_G \ X \ D_G \ X \ M$

Total Computational Complexity

$$M \ X \ D_G \ X \ D_G \ X \ D_K \ X \ D_K \ + \ N \ X \ D_G \ X \ D_G \ X \ M$$

where the computational cost depends multiplicatively on the number of input channels M, the number of output channels N, the kernel size $D_K X D_K$, the output feature map G size $D_G X D_G$.

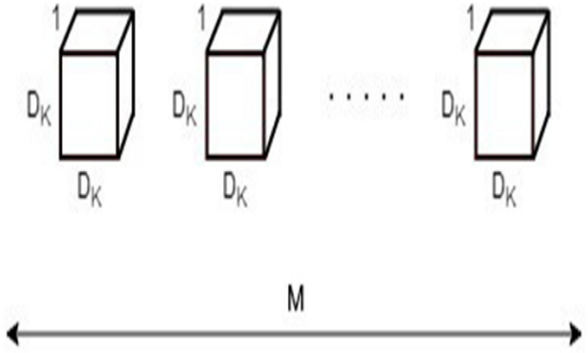


FIGURE 2.8: Depthwise Convolution Filters(Depthwise Separable)

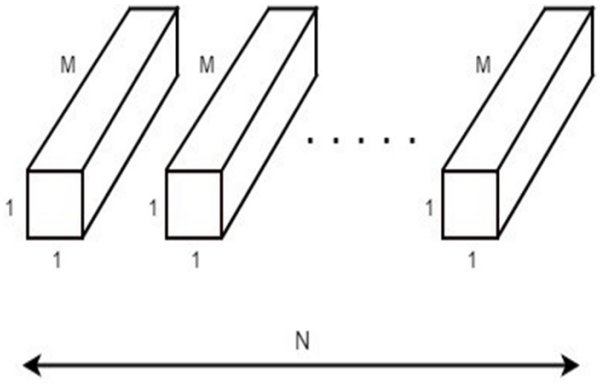


FIGURE 2.9: Pointwise Convolution Filters(Depthwise Separable)

2.3.3 Comparison of Standard Convolution and Depthwise Separable Convolution

By expressing convolution as a two-step process of filtering and combining we get a reduction in computation of :

$$\frac{\textit{No. of mult in depthwise separable convolution}}{\textit{No. of mult in standard convolution}}$$

$$= \frac{M \, X \, D_K \, X \, D_K \, X \, D_G \, X \, D_G + D_G \, X \, D_G \, X \, M \, X \, N}{N \, X \, D_G \, X \, D_G \, X \, D_K \, X \, D_K \, X \, M}$$

$$= \frac{1}{N} + \frac{1}{D_K^2}$$

MobileNet uses 3 X 3 depthwise separable convolutions which uses between 8 to 9 times less computation than standard convolutions at only a small reduction in accuracy. For e.g. Consider an output feature volume N = 1024 and DK of size = 3, plugging in the values in convolution expression, we can see,

$$= \frac{1}{N} + \frac{1}{D_K^2}$$

$$= \frac{1}{1024} + \frac{1}{3^2}$$

$$= \frac{1033}{9216} \;=\; 0.112$$

This indicates that depth separable convolution 8 – 9 times faster with respect to standard convolution in terms of computational complexity.

2.4 MobileNet Model (Depthwise separable CNN)

2.4.1 Introduction

MobileNets, a family of mobile-first computer vision models for TensorFlow, designed to effectively maximize accuracy while being mindful of the restricted resources for an on-device or embedded application. MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of a variety of use cases. They can be built upon for classification, detection, embeddings and segmentation similar to how other popular large scale models, such as Inception, are used. MobileNets are based on a streamlined architecture that uses depth-wise separable convolutions to build light weight deep neural networks.

Following are advantages of MobileNet :

1. MobileNets are very small, they consist of very few parameters to train.
2. MobileNets are very fast to train.
3. Mobilenets provide very good accuracy. The accuracy is almost similar to other Standard Convets like VGG16, AlexNet, etc.
4. MobileNets are optimized for mobiles. They are easy to tune for resources vs accuracy.

2.4.2 Architecture of MobileNet

Table 1. MobileNet Body Architecture

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	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
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	$1 \times 1 \times 1000$

FIGURE 2.10: MobileNet architecture

The softmax function is given as :

$$f_j(z) = \frac{e^{z_j}}{\Sigma_k e^{z_k}}$$

where $j \in [1, N]$ and N denotes the number of classes, ie the number of nodes in the final output layer.

The MobileNet structure is built on depthwise separable convolutions as mentioned in the previous section except for the first layer which is a full convolution. Counting depthwise and pointwise convolutions as separate layers, MobileNet has 28 layers. The figure shows the layers in Standard and MobileNet Convolution Layers. It is not enough to simply define networks in terms of a small number of Mult-Adds. It is also important to make sure these operations can be efficiently implementable.

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

FIGURE 2.11: Depthwise Separable vs Full Convolution MobileNet

As we can see in the above figure, the deeper we go into the model, the depth of feature image goes on increasing, but the width and height of image goes on decreasing.

2.4.3 Network Training

MobileNet models were trained in TensorFlow with asynchronous gradient descent similar to Inception V3. When training MobileNets side heads or label smoothing were not used and additionally reduce the amount image of distortions by limiting the size of small crops that are used in large Inception training. The layers of CNN mentioned in Figure 2.12 can be repeated as Standard CNN, they can contain multiple convolutional and pooling layersbut in MobileNet, there are multiple 1*1 CNN depthwise layers and only single layer of pooling.MobileNet models were trained in TensorFlow with asynchronous gradient descent similar to Inception V3.

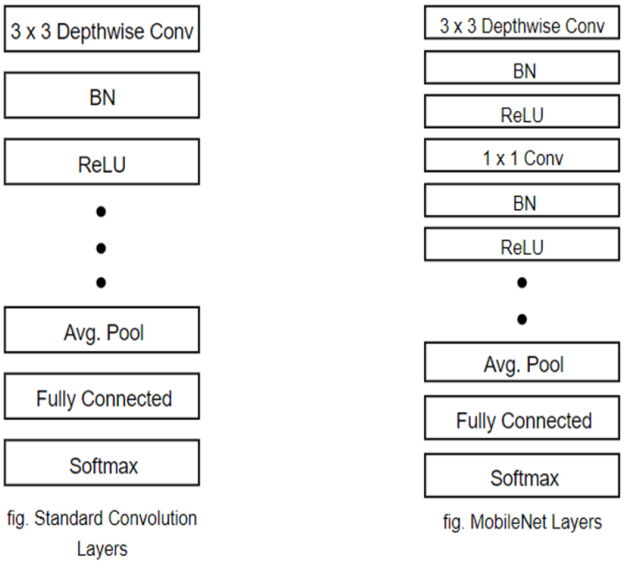


FIGURE 2.12: Standard Convolution and MobileNet Layers

2.4.4 Width Multipler in MobileNet Model

Although the base MobileNet architecture is already small and low latency, many times a specific use case or application may require the model to be smaller and faster. In order to construct these smaller and less computationally expensive models there is a very simple parameter called width multiplier. The role of the width multiplier is to thin a network uniformly at each layer. The computational cost of a depthwise separable convolution with width multiplier is:

$$D_K \; X \; D_K \; X \; \alpha M \; X \; D_F \; X \; D_F + \; \alpha M \; X \; \alpha M \; X \; \alpha N \; X \; D_F \; X \; D_F$$

where $\alpha \in \; (0, 1]$ with typical settings of 1, 0.75, 0.5 and 0.25. Width multiplier has the effect of reducing computational cost and the number of parameters quadratically by roughly α^2.

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

FIGURE 2.13: Mobilenet Width Multiplier

2.4.5 Resolution Multipler in MobileNet Model

The second hyper-parameter to reduce the computational cost of a neural network is a resolution multiplier . The computational cost for the core layers of our network as depthwise separable convolutions with width multiplier and resolution multiplier ρ :

$$D_K \; X \; D_K \; X \; \alpha M \; X \; \rho D_F \; X \; \rho D_F + \; \alpha M \; X \; \alpha M \; X \; \alpha N \; X \; \rho D_F \; X \; \rho D_F$$

where $\rho \in (0, 1]$which is typically set implicitly so that the input resolution of the network is 224, 192, 160 or 128. Resolution multiplier has the effect of reducing computational cost by 2.

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

FIGURE 2.14: MobileNet Resolution

Chapter 3

Analysis and Design

3.1 Use case specification

- Brief Description :

  The following use-case describes the process in which the user will be given a task to draw a sketch on the screen and system will predict the result based on the drawn sketch.

- Actors : Normal Person

- Pre-condition :

  1. The user used should know how to draw basic sketches.
  2. The user should try to draw only the those sketches that are asked by the system.

- Basic Flow of Events:

  1. The use-case begins with the user opening the interface.
  2. He/She selects the play button.
  3. The system asks the user a question related to sketch.
  4. The user types in the answer.
  5. The user submits the answer.
  6. The person draws the sketch, which was the answer to the previous question.

7. .The system predicts the result based on the drawn sketch.
8. Use case ends successfully.

- Alternate flow of events:

  For step 4 in the basic flow, the user types incorrect answer :

  1. The User clicks on hint button
  2. The system displays the hint
  3. User types the answer based on given hint
  4. Go to step 5 for basic flow of events

  For step 7 in the basic flow, the system predicts that sketch drawn by the user does not match :

  1. The user clicks on undo button as many times he/she wants.
  2. The user rethinks on how he/she should draw the sketch.
  3. Go to step 6 for basic flow of events.

- Post-condition :

  1. The sketch is recognized by the system successfully.
  2. The system rewards the user with points according to the perfection of the sketch drawn by him/her.

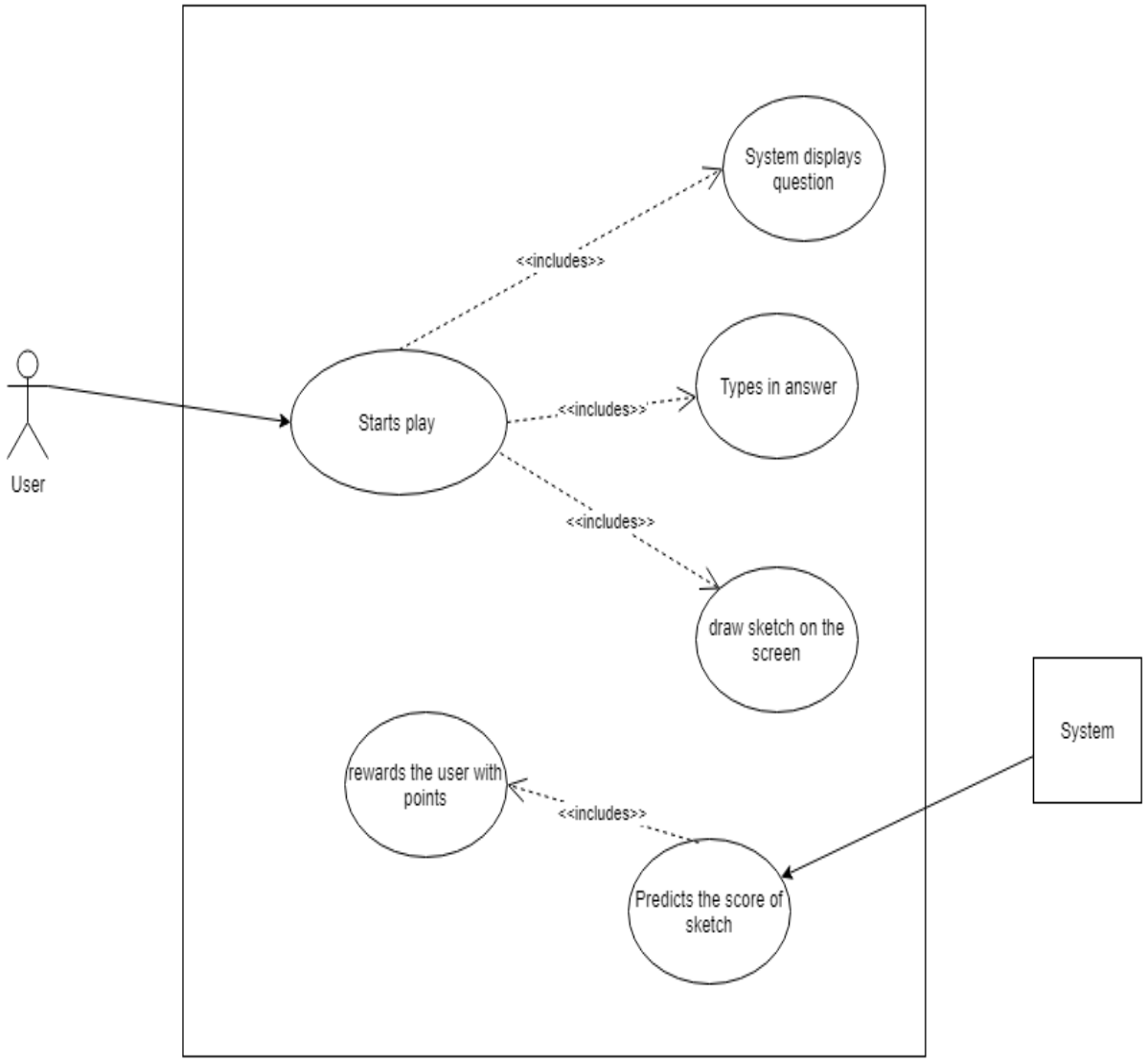


FIGURE 3.1: Use case diagram

3.2 Architecture Diagram

The advantage of a three tier system from a project communications standpoint is that the three tiered architectural is relatively straightforward to non-technical people. It would only take a couple of minutes to explain how the system works. The three tiered architecture shown below has the following major components:

1. Presentation Layer :

   This is basically what the normal user of the system will interact with or see. The User will interact with the system with the following logic and physical components.

   (a) Logical Components:

   User will interact with the system with a Mobile App interface developed in XML. This UI will use the physical components to accept the input and displaying the output processed from the lower layers.

   (b) Physical Components:

   This is the hardware that the user will use the system on. In this case, it is the smartphone which everyone carries around. Every interaction of the user with the system will be displayed on the smartphone.

2. Logic Tier:

   This layer will deal with logical part of the system that are the different modules and algorithms that will be used to process the data of input and output. There are multiple modules used in this system from image - bytes array conversion to neural network algorithm for sketch classification and database storage for game score and user information.

3. Data Tier:

   This layer will deal with the database and neural network model in the system. Database mainly consisting of user score data and other information and Neural Network for classifying sketches that will be fed to it.

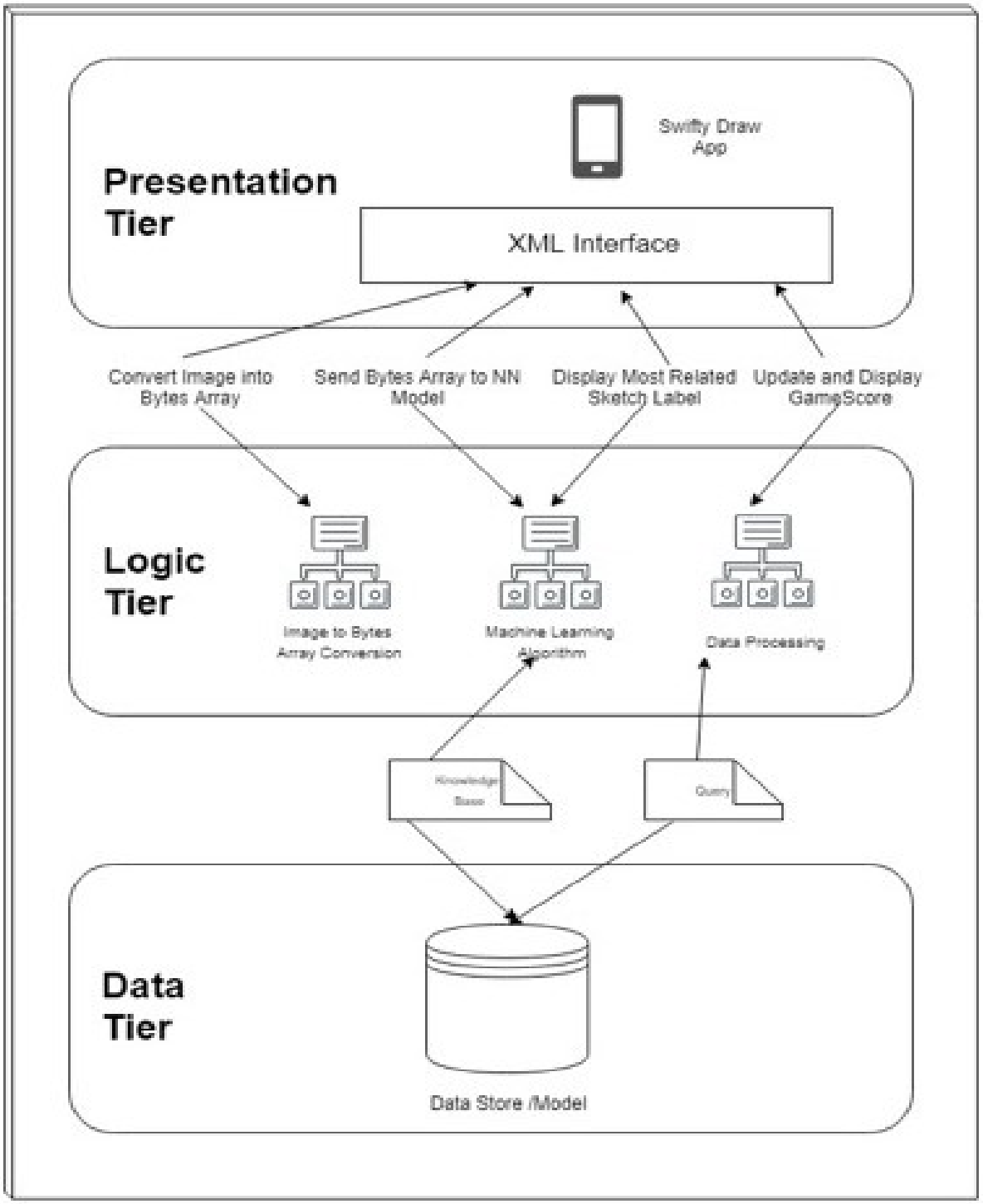


FIGURE 3.2: Architecture Diagram

3.3 Sequence Diagram

Sequence diagram shows the life lifeline of each actor and sequence of events. It consits of two actors and system. The user initiates the call to start the system.The bar shows the lifeline of the actor and arrows indicate the next step after the action.

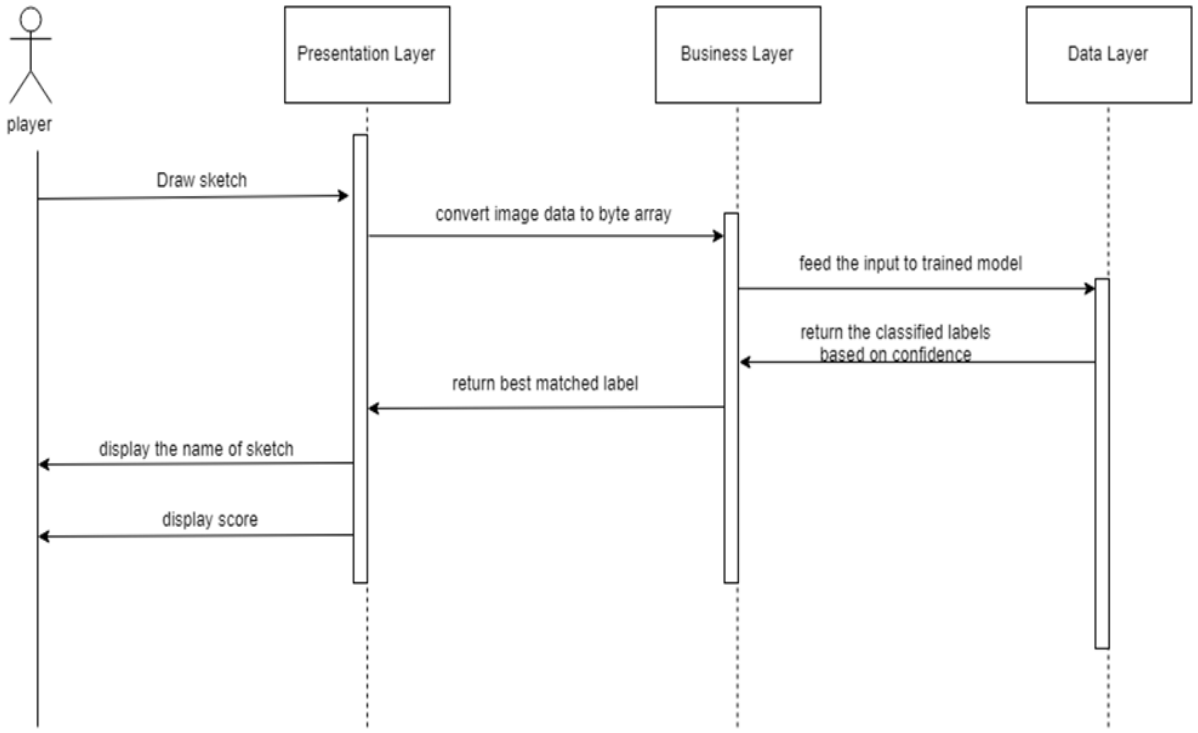


FIGURE 3.3: Sequence Diagram

3.4 Activity Diagram

Activity diagram shows various activities during recognition process. It depicts all the steps involved in the depiction process. It also states, if a step does not work what is the result of it. The rectangles, curved rectangles represent the action and the triangles represent the condition.

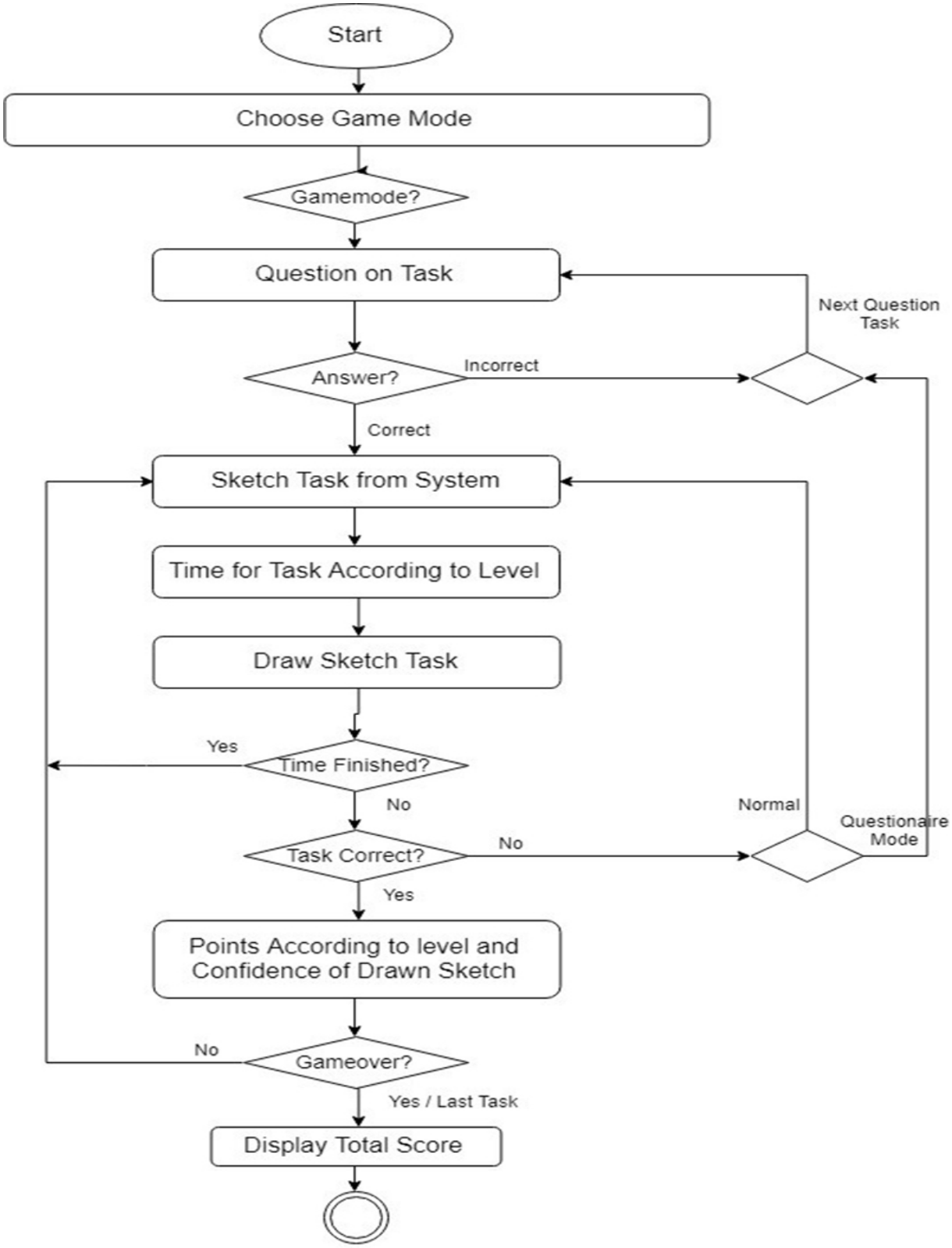


FIGURE 3.4: Activity Diagram

3.5 Model Diagram

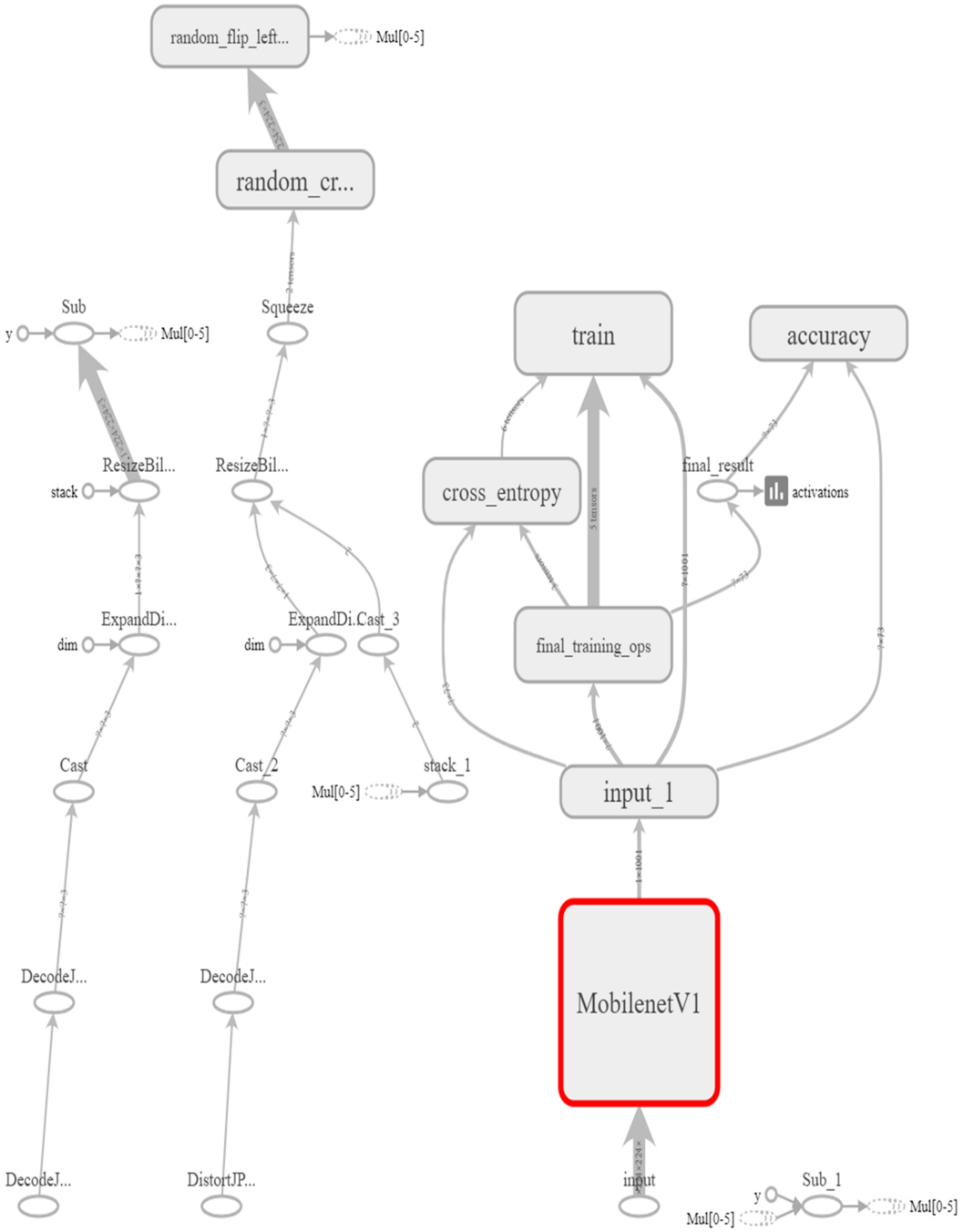


FIGURE 3.5: Model Diagram

Chapter 4

Implementation

4.1 Proposed System

The figure below shows a flow diagram of a proposed system for Sketch classification. Sketches drawn by the users are captured into an image format. The captured image is fed to the pretrained CNN Model. The CNN model used for this research is explained in Section. Figure 4.1 explains the basic model of how the user drawn sketch is processed and classified by the neural network and label is selected by the application.

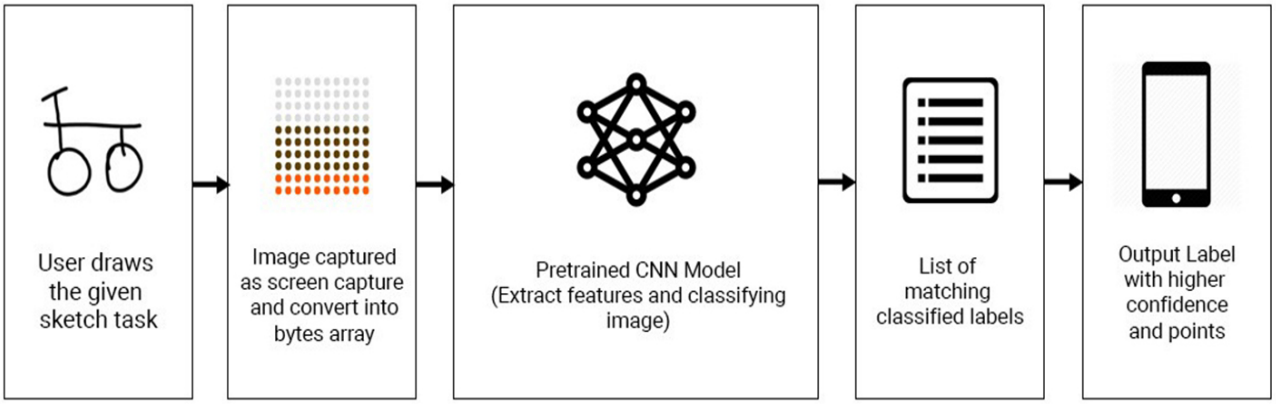


FIGURE 4.1: Process of Classification of Sketch

On selecting the game mode, a user is given a set of tasks to be performed or drawn. User draws the given sketch task and a 28 x 28 scaled image of the hand drawn sketch task is generated which is then converted into byte array which is fed to the pretrained neural network. The pretrained neural network performs the convolution operations on the bytes array data and provides a set of labelled outputs sorted by confidence for the byte array. According to the level of the task assigned and the threshold allowed on the level, the label at the top of the matched label list is selected and points are assigned to the task based on the confidence of the image.

Everything is repeated for a finite time with different tasks and different levels for the tasks.

4.2 Dataset and Neural Network details

4.2.1 Dataset

We are using QuickDraw dataset open-sourced by google which is a huge collection of free hand human drawn sketches of real world objects. It contains about 5 million sketch images totalling its size to 3 TB of data but we are using on part of data about 75,000 images for projects because of size constraints and low computational resources on Mobile Devices.

The data is categorized in raw-data and preprocessed data. The data is captured in time stamped vector format drawings and is available in formats - NDJSON, NPY, BIN. The dataset is freely available to be downloaded and used for development purposes.

The quickdraw dataset is in the form .npy files and it is preprocessed. The npy files are provided as input to convert npy-to-jpg.py python script which will generate about 1000 images for each category of sketches.

NOTE: Before running the file, check or change the input and output location according to your directory location in the script.

Run the processing script :

python3 convert `npy_to_jpg.py`

The images are the input to the neural network training.

4.2.2 Splitting Dataset into training set and test set

The QuickDraw dataset has 75,000 human drawn sketches. We split the dataset in 70:30 format. The training set contains 70% of the total images, which translates to 52,500 images in the test set. The test set contains 30% of the total images, which translates to 23,500 images in the test set.

4.2.3 Training using MobileNet

We are using MobileNet convolution model to train the images and generate pretrained model. TensorFlow comes packaged with great tools that you can use to retrain MobileNets without much code to write. The pretrained model uses parametric arguments to start training the dataset.

For training our image classifier, we are going to use the transfer learning concept. Transfer learning basically refers to a supervised learning technique that takes advantage of an already existing trained model that solves a similar problem. For our purpose, we will take TensorFlow's fully trained model for Imagenet and retrain just the last layer of the neural network on our QuickDraw dataset. Though this approach is not as powerful as a fully trained model, but it can provide a surprisingly high accuracy for most tasks that are related.

As we are only training the final layer of the neural network, the training will end in reasonable amount of time. TensorFlow's retraining procedure allows you to optimize the training procedure by tweaking certain parameters. The pretrained model uses parametric arguments to start training the dataset.

```
convnet = input_data(shape=[None, 28, 28, 1], name='input')
convnet = conv_2d(convnet, 32, 2, activation='relu')
convnet = conv_2d(convnet, 32, 2, activation='relu')
convnet = conv_2d(convnet, 64, 2, activation='relu')
convnet = conv_2d(convnet, 64, 2, activation='relu')
convnet = max_pool_2d(convnet, 2)
convnet = conv_2d(convnet, 128, 2, activation='relu')
convnet = conv_2d(convnet, 128, 2, activation='relu')
convnet = conv_2d(convnet, 256, 2, activation='relu')
convnet = conv_2d(convnet, 256, 2, activation='relu')
convnet = max_pool_2d(convnet, 2)
convnet = fully_connected(convnet, 1024, activation='relu')
convnet = dropout(convnet, 0.25)
convnet = fully_connected(convnet, len(classes), activation='softmax')
convnet = regression(convnet, optimizer='adam', learning_rate=LR, loss='
   categorical_crossentropy')
model = tflearn.DNN(convnet, tensorboard_verbose=3)
train = trainingData[:-6490]
test = trainingData[-6490:]
print('Size of Train: {} and Test: {}'.format(len(train), len(test)))

X = np.array([i[0] for i in train]).reshape(-1, 28, 28, 1)
Y = [i[1] for i in train]
    test_x = np.array([i[0] for i in test]).reshape(-1, 28, 28, 1)
    test_y = [i[1] for i in test]
    model.fit({'input': X}, {'targets': Y}, n_epoch=60, validation_set=({
   'input': test_x}, {'targets': test_y}),snapshot_step=500, show_metric=
   True, run_id=MODEL_NAME)
```

```
    del tf.get_collection_ref(tf.GraphKeys.TRAIN_OPS)[:]
    model.save("model_test_cnn")
```

The retrain script can retrain either Inception V3 model or a MobileNet.In this project, We will use MobileNet. The principal difference is that Inception V3 is optimized for accuracy, while the MobileNets are optimized to be small and efficient, at the cost of some accuracy.

Inception V3 has a first-choice accuracy of 78% on ImageNet, but is the model is 85MB, and requires many times more processing than even the largest MobileNet configuration, which achieves 70.5% accuracy, with just a 19MB download.

To start the training, just enter:

```
python3 -m retrain \
  --bottleneck_dir=TrainingResult/bottlenecks \
  --how_many_training_steps=7000 \
  --model_dir=TrainingResult/models/ \
  --summaries_dir=TrainingResult/training_summaries/"mobilenet_1.0_224" \
  --output_graph=TrainingResult/retrained_graph.pb \
  --output_labels=TrainingResult/retrained_labels.txt \
  --architecture="mobilenet_1.0_224" \
  --flip_left_right=true
  --image_dir=Output
```

NOTE: Image Dir Output to be replaced by your Images Directory.

The Training will take some time depending upon the dataset and the processing power of your machine.

When this finishes, we will have two files :

- retrained_graph.pb : The newly generated protobuf graph of our neural network which will be used in Android App for recognizing images.

- retrained_labels.txt : The newly generated labels file which contains names for images trained.

4.2.3.0.1 Model Training Parameters and Time Requirements:

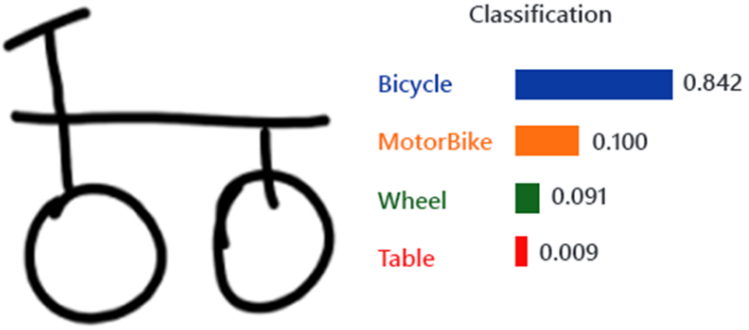
- training steps = 8000
- architecture = MobileNet_1.0_224
- learning rate = 0.01
- train batch size = 512
- output graph = retrained_graph.pb

Configuration	Time Required
Dual Core i3 CPU, 8GB RAM	28 Min 45 Sec
Quad Core i5 CPU,8 GB RAM, 2GB 940MX GPU	20 Min 9 Sec
Octa Core Xeon CPU, 10 GB RAM	22 Min 17 Sec

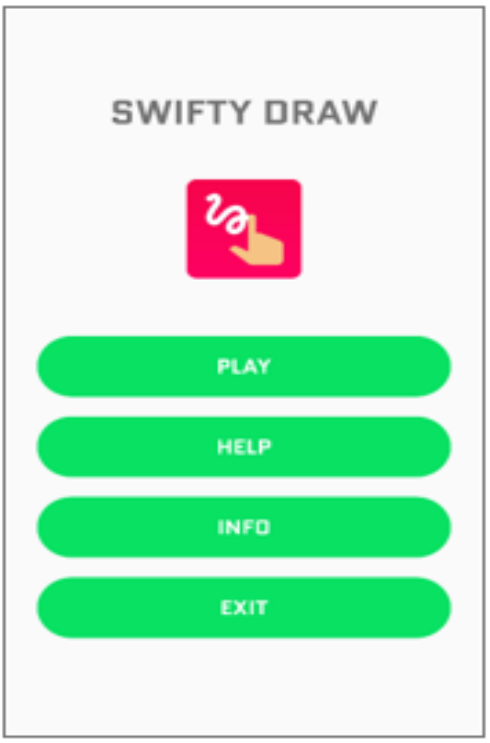
4.2.3.0.2 Model Accuracy Size The final training accuracy using MobileNet with 1.0 width multiplier with 224 resolution, learning rate 0.01 was 98% and validation accuracy was 84.7%.

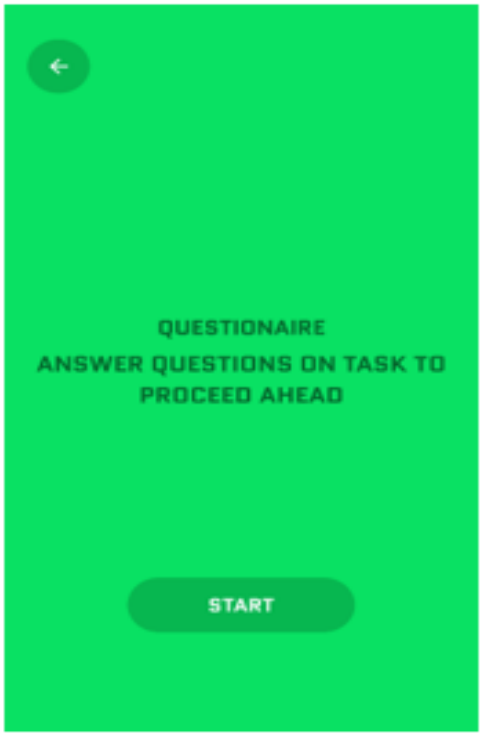
The trained frozen graph model size was only 16.5 MB which is very small in size compared to standard CNN's frozen graph which was 98.4 MB.

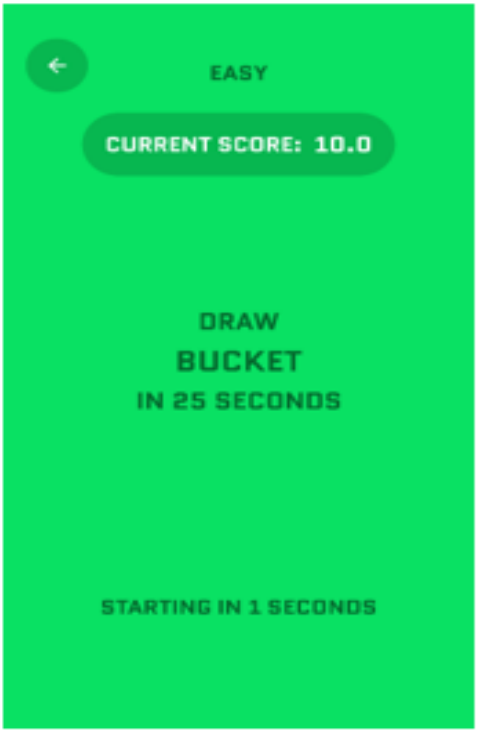
The trained model is used the application to recognise hand drawn sketches.



4.3 Screenshots of System







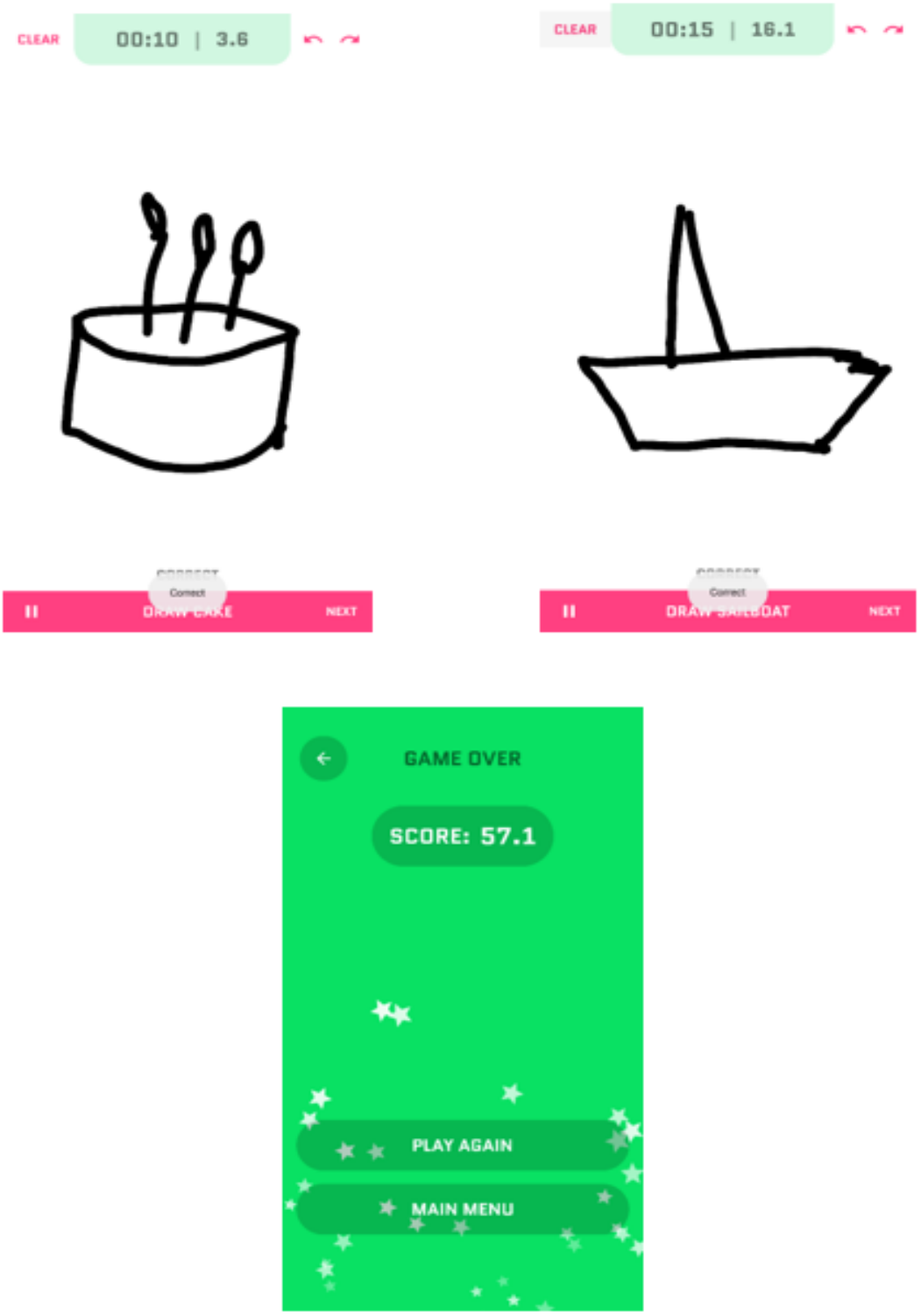


FIGURE 4.2: Screenshots of System

4.4 Gameplay

A task i.e. a sketch to be drawn is provided to the user that needs to be completed or drawn in the time provided for points.

Based on the what and how image is drawn for a given task, the app calculates a score for the image drawn and based on it and the level at which user is playing, an overall score is calculated for the task.

The task becomes harder and harder as you proceed through the levels, the harder task you solve, the more points you get.

The recognition threshold increases gradually and looks for more correct depiction of the given task gradually. The Game provides four modes for playing:

- Normal Mode
- Arcade Mode
- Intime Mode
- Questionnaire Mode

4.4.1 Normal Mode

In Normal Mode, a set of tasks ranging from easy to hard is provided and the user has to complete the particular task in the provided time based on the level like for easy it will be 25 seconds, medium 15 seconds and hard 10 seconds. Based on the sketch drawn by the user i.e. how good or bad the sketch is drawn, points are associated to the sketch task as mentioned below. In this mode, if the user can't complete the task, the game proceeds with the next task unlike the arcade mode. Once a particular set of tasks given are completed, an overall score is calculated which is additive of set of tasks which indicates the gameover. In this as the game progresses, the tasks become harder and harder to solve as there are constraints applied on time and the complexity of sketch task but more points onboard.

4.4.2 Arcade Mode

In Arcade Mode, a set of tasks are provided in similar manner to Normal Mode and based on the sketch drawn by the user, points are associated to task. But unlike Normal Mode, the points associated across all tasks are same and doesn't depends on the level of tasks. Also, if the user can't complete the, gameplay stops and the additive score is calculated as total score which indicates gameover. The Max Points assigned in Arcade Mode for task is 30 Points, the time for tasks in Arcade Mode is 18 seconds.

4.4.3 Intime Mode

The In-Time Mode is very similar to arcade mode but the difference is that a constant global time is assigned which is 3 minutes and in 3 minutes the users has to complete maximum number of tasks as possible. This mode doesn't associate any points for the completed tasks instead it calculates the number of tasks completed out of the given tasks once the time which is 3 minutes is over.

4.4.4 Questionnaire Mode

The Questionnaire Mode is similar to Normal Mode, but the difference is that in Questionnaire Mode instead of directly providing the sketch task, a question is given and the user needs to answer the question correctly. If the answer is correct, the sketch task is provided to the user to perform. For each successful answer to the question, 10 points are given to the user which is added with the successful completion of task. This mode also provides with the option of hint which can be used if the user can't guess the answer of question in the given time. If hint is used by the user, 5 points are deducted from the 10 points given to the question. The Question needs to be answered in 45 seconds and if user fails to answer the question in 45 seconds, the next question task is provided to the user.

For e.g. the user will be given question like "What do we use to protect ourselves from Rain?" and the answer is Umbrella then the sketch to be drawn is Umbrella.

Chapter 5

Result analysis

The proposed CNN model is applied to the QuickDraw dataset. The dataset conatins 71 output classes. Each class contains around 1000 images for training. All the images were of size 28 * 28 pixels. Following are the results that we achieved after training the dataset using Standard CNN model as well as MobileNet model.

5.1 Accuracy of Standard CNN model

We trained our dataset with the Standard CNN model on a subset of dataset conatining around 15000 images. The images were included from every class of the dataset. We applied batch size of 512. The training was performed on 8-core Xeon processor with 12 gigabytes of RAM on Google Cloud Engine with Ubuntu 16.10 VM Instance and 10GB persistent solid state storagefor high training performance . We applied 60 epochs on the dataset.

1. Training time : 4hrs 30 mins
2. Training accuracy : 98.6
3. Test accuracy : 87.6
4. Model size :25.6 megabytes

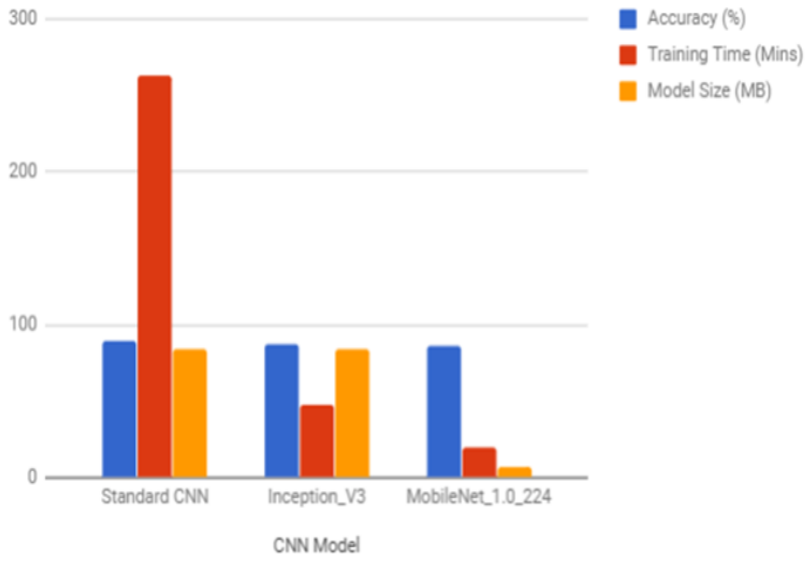


FIGURE 5.1: Standard CNN vs MobileNet comparison

5.2 Accuracy of MobileNet model

We trained our dataset with the MobileNet model on a subset of dataset conatining around 15000 images. The images were included from every class of the dataset. We applied batch size of 512. The training was performed on 8-core Xeon processor with 12 gigabytes of RAM on Google Cloud Engine with Ubuntu 16.10 VM Instance and 10GB persistent solid state storagefor high training performance. We applied 60 epochs on the dataset.

1. Training time : 4hrs 30 mins
2. Training accuracy : 97
3. Test accuracy : 82.3
4. Model size :6.4 megabytes

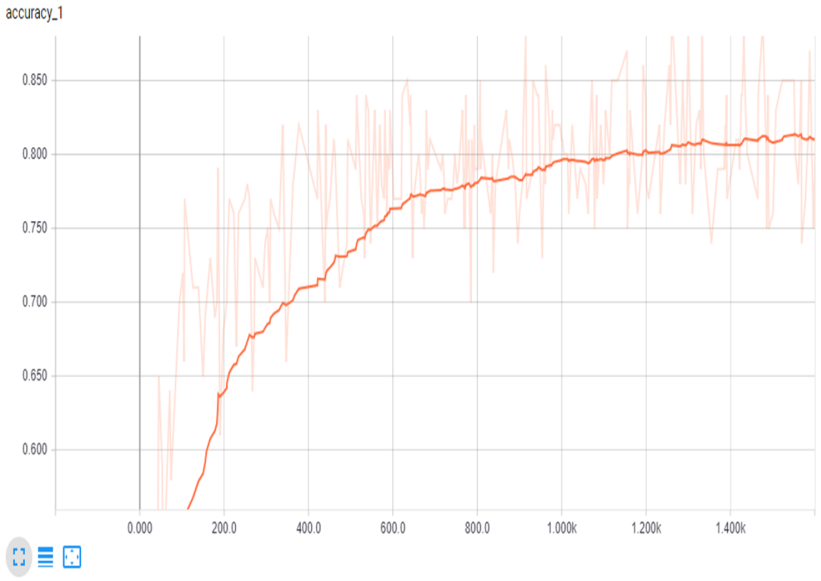


FIGURE 5.2: MobileNet accuracy

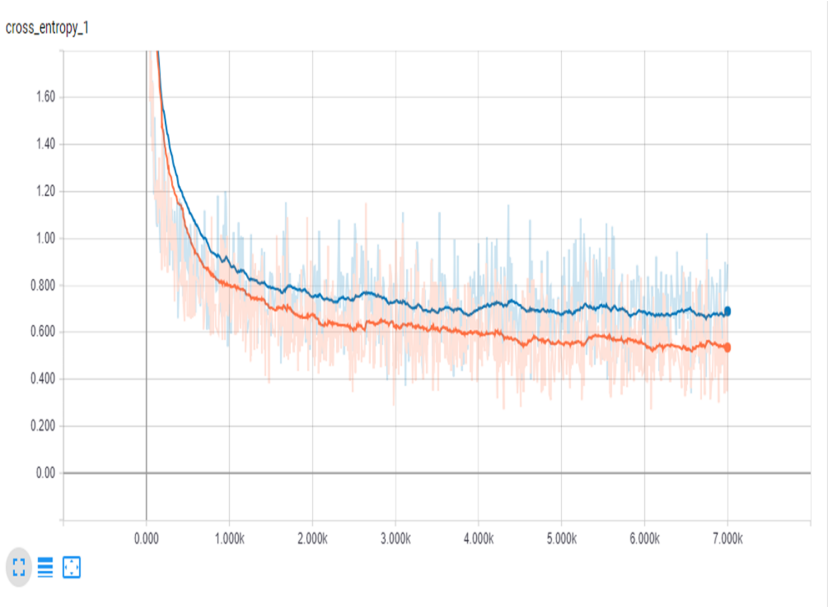


FIGURE 5.3: MobileNet crossentropy

Chapter 6

Conclusion and Future enhancements

6.1 Conclusion

The Histogram or Oriented Gradient based method discussed in prior work used for the recognition of sketches has a advantage of being simple and robust for photos classification but not so good to take into account the unique traits the hand drawn sketches possess and the different features and characteristics that the sketches represents.

This is why in this project, we tried to represent the different CNN models and how Neural Network can work with free drawn sketches which is a dramatical improvement over the traditional method and we got validation accuracy of 87.8% but even though CNN is correctly able to classify sketches by extracting features and feeding to Neural Network. It is to be noted that the neural network is used on a portable device like smartphone and compressing such a huge neural network on a portable device with low computational resources is a huge task.

For using the neural network model with the same traits of CNN's characteristics of accurate sketch classification with portability, we used the MobileNet Model which is based on Depthwise Convolution Methodology which greatly helps in reducing the neural network model size i.e. make it portable enough to be used on portable devices like smartphone which has low computational resources to work with without much sacrificing on the accuracy of classification.

The current system uses the MobileNet model trained on 72998 28x28 images and is able to classify on 74 different hand drawn sketches with validation accuracy of 82.1% embedded into a simple easy to use app where the sketch classification is used and transformed it into a Game which can be played to have fun and engage into the interactive system as explained in chapter 4 and it does not need any special hardware to run it. User just need to have a smartphone running Android 5.0 or greater.

The main aim of developing such a system is to represent how Machine Learning is not only used for problem solving but it can also be used in a fun way to make the Power of Machine Learning more Intuitive and Attractive

6.2 Future Enhancements

1. Recognizing sketches using smartphone camera drawn on physical paper instead of app. There are some people who prefer drawing on paper. Or some are just more comfortable drawing on a physical paper than on a screen, part of a reason being drawing on a per gives you more flexibility and accuracy in drawing.
2. Sketch Task Challenges among friends using multiplayer to make game more engaging.This will allow you to compete with your friends and challenge them to better your score.
3. Learning user behaviour of drawing sketches and dynamically assigning easy or hard task and points based on how user is playing. This ensures adaptivity. The system learns how the user is playing. If he/she is performing poorly, the app will give him/her some easy sketches and once he starts scoring, the app will increase the difficulty level a little bit.
4. More Accurate Sketch Classification and Improved Overall Performance. The better the accuracy, the more is the probability of our app classifying an object correctly.
5. Global leader board of the top high scorers. This will let you know how other users of the app are scoring.

Bibliography

1. Hand Drawn Sketch Classification suing convolutional neural networks – Habibollah Agh Atabay–http://www.iioab.corg/articles/IIOABJ-7.S5-337-341.pdf

2. Free hand sketch recognition classification – Stanford University http://cs231n.stanford.edu/report/2017/pdfs/420.pdf

3. Free hand drawn sketch segmentation – Brain like computing lab, Shanghai Jiao, Tong University, P.R. China – http://bcmi.sjtu.ed.cn/ zhangliqing/Papers/2012ECCV-Sun-Sketch.pdf

4. ImageNet Classification witg Deep Convolutional Networks – University of Toronto – http://papers.nips.cc/paper/4824-imagenet-classification-with -deep-convolutional-neural-networks.pdf

5. SketchNet: Sketch Classification with WebImages – School of Computer Science and Technology, Tianjin University – https://pdfs.semanticscholar.org/b5b6/20774304e6245a660b14c1207386d3abad17.pdf

6. Sketch-a-Net that Beats Humans – School of Electronic Engineering and Computer Science, Queens Mary – https://arvix.org/pdf/1501.07873.pdf

7. Sketch-based Object Recognition – Standard University – http://cvgl.stanford.edu/teaching.cs231a-winter1415/prev/projects/final-report.pdf

8. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision for Mobile Vision Applications – Andrew G. Howard, Menglong Zhu, Bochen, Dmitry Kalenichenko, Weijun Wang, Tobbias Weyand, Marco Andreetto, Hartwig Adam - https://arxiv.org/pdf/1704.04861.pdf

9. Feature Representation in Convolutional Neural Networks – Ben Athiwaratkun, Keegan Kang – Cornell University - https://arxiv.org/pdf/1507.02313.pdf

10. Using Convolutional Neural Networks for Image Recognition – Samer Hijazi, Rishi Kumar and Chris Rowen, IP Group, Cadence https://pdfs.semanticscholar.org/bbf7/b5bdc39f9b8849c639c11f4726e36915a0da.pdf

11. Creating insanely fast image classifiers with MobileNet in Tensorflow – https://hackernoon.com/creating-insanely-fast-image-classifiers-with-mobilenet-in-tensorflow-f030ce0a2991

12. Inception - https://medium.com/initialized-capital/we-need-to-go-deeper-a-practical-guide-to-tensorflow-and-inception-50e66281804f