Athlone Institute of Technology

School of Engineering.

Project Thesis – Academic Year 2018/19

Object Detection

Using

Convolutional Neural Network

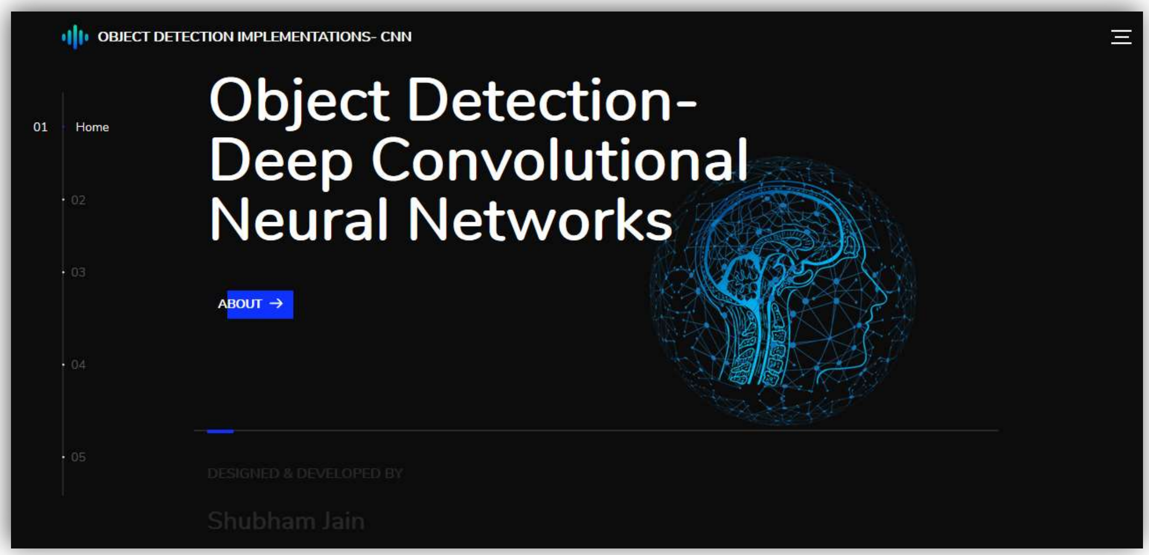
By

Shubham Jain

Bachelor of Engineering (Honours) in Software Engineering

Object Detection using

Convolutional Neural Network



Object Detection-
Deep Convolutional
Neural Networks

DESIGNED & DEVELOPED BY

Shubham Jain

Author: Shubham Jain

Supervisor: Michael Thornton

Project Summary

Convolutional neural networks (CNNs) are widely used in pattern- and image-recognition problems as they have a number of advantages compared to other techniques. This project will discuss the various applications of implementing a Convolutional Neural Network for Image Detection such as Facial Emotional Detection and Sight Prediction. The architecture of ConvNet and different layers would also be outlined. Data loss and efficiency while training will also be discussed. The applicability of final model is portrayed in a website where an interactive web application trains data by capturing images and assigning labels dynamically for emotion detection and saving cursor location as well as the respective image. The dataset is then provided to TensorFlow MobileNet model which processes the image on multiple artificial layers and generates predictions.

Table of Contents

Introduction

Image processing are cost-efficient alternatives to fancy sensors, they analyse an image and retrieve useful information vectors. Image processing converts images to digital images to perform image enhancement or gather useful information by analysing an image. Object detection is one application of image processing where we distinguish image characteristics and find patterns or similarities inside tiny coloured or grayscale dots i.e. pixels of an image.

This report outlines the importance of Convolutional Neural Network for image detection which is a deep learning algorithm. CNN's take an input image and assign weights or biases to varied aspects of an image to differentiate between different objects in an image. Image filtering kernels such as edge detection, sharpening, blurring, embossing and more are hand engineered in primitive image detection techniques which require higher processing power while ConvNet has an ability to learn filters in a multilayer architecture and utilize minimal processing compared to primitive techniques.

Before I started working on deep learning it was immensely important to understand of Neural Network works which I did with an analogy of neurons in human brain; Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area. This architecture proved useful for image classification.

After having decided the architecture for classification I chose TensorFlow MobileNet Neural Network architecture which makes use of a dense neural network approach rather than traditional shallow neural network structure. The multi layered architecture mainly contains four operations Convolution, Non Linearity (ReLU), Max Pooling or Sub Sampling and Classification (Fully Connected Layer). The model for image classification has 300 Million MACs and 3.47 Million parameters needed to perform inference on a single 224×224 RGB image.

The final trained model will be able to generate predictions to detect facial emotions on live webcam feed as well as sight detection using frame by frame analysis. Use of MobileNet brings more processing speed but trades off accuracy as a result the model can be found overfitting when trained with more than 20 epochs. More information about accuracy would be discussed in results section.

Aims and Objectives

The main objective was to learn about Convolutional Neural Networks in more detail, analyse and implement different models and compare general predictions and trade-offs. Other aim was to develop an implementation of learned architecture that can be a potential area of research and increase knowledge base form the existing literature. Facial Emotion Detection and Sight Detection were the applications for my Neural Network, aims and objectives of which are:

Aims:

- A fully functional website to detect human facial emotion.
- An application to detect where human is currently looking at the browser screen.
- To gain understanding of different ConvNet Models.
- Understand previous literatures for emotion detection and sight detection.
- Provide reasonable solutions to existing problems.

Objectives:

- Realtime predictions using webcam.
- Collect data on the website using webcam and label it dynamically.
- To decide model for my Convolutional Neural Network.
- Train data by providing weights and bias dynamically.
- Investigate and compare accuracy with literatures and state-of-the-art accuracy.
- To understand when the model works fine and when is it overfitting.
- Creation of a document to explain this project methodology and results.

Scope

After having decided on the domain and the application I was working on and the approach to begin with, it was necessary to generalize the scope of this project. I chose TensorFlow as TfLearn as my main machine learning libraries and found out the necessary environment and dependencies. I looked at previous literatures and architectures and chose the TensorFlow model that met with my hardware conditions.

The scope of my project was to provide users with a base that detects human emotions and detects sight with easy and efficient training algorithm and provide a web tutorial on how to use it. Also, the scope includes the use of previous approach and improving it to generalize and adapt to various image conditions like background elimination and cropping image to make using of more processing power.

When the website is completed it will be an interactive application where users can train their own emotions and generate real time predictions. This would be the same for sight prediction where an interactive tutorial will guide them towards training data and saving their model in browser local storage and then generating predictions as well as heat maps for understanding those predictions.

My aims and objectives will encompass in the scope of my project. Also, the future scope would be discussed in the later sections of this report.

Build Tools

Hardware & Software

Python 3.7: Python an interpreted language provides various inbuilt libraries which makes it most suitable to work with AI projects.

JavaScript: A programming language for web useful to deploy my AI application on the web.

TensorFlow: A python-based machine learning library which provides abstraction for implementing different algorithms.

TFlearn: Built on top of TensorFlow, a higher-level API facilitates faster processing while still being fully transparent as functions are built over tensors and can be built independently.

OpenSourceComputerVision: Face detection using haar cascades.

Development of AI application using above libraries in a python IDE such as PyCharm can be very processor heavy as it requires at least 8 GB ram, intel i5 processor with NVIDIA GPU along with CUDA toolkit to train data models.

An alternative approach that uses less CPU processing power and can work on minimum hardware configurations without NVIDIA GPU is the use of tensorflow.js in JavaScript web application. It facilitates the use of MobileNet TensorFlow model with consumes less processing power but compromises with the accuracy.

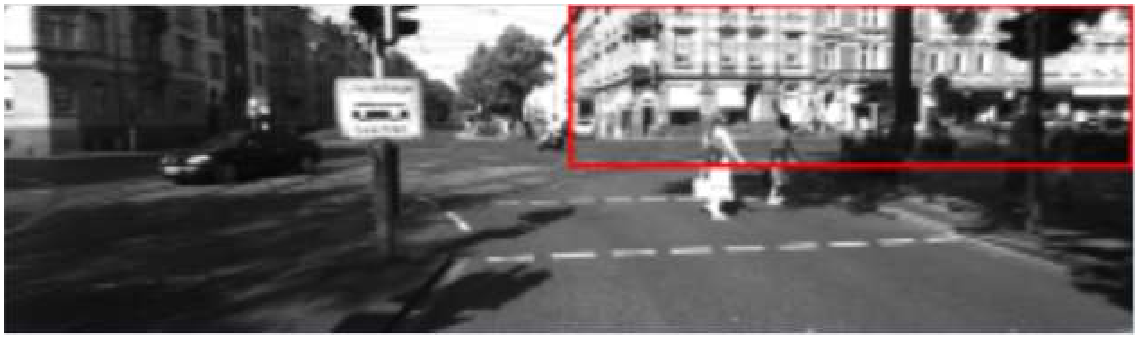
Research Introduction

After deciding my domain of work, I found various approaches that could be implemented to achieve image classification which are defined below

Previous Approaches

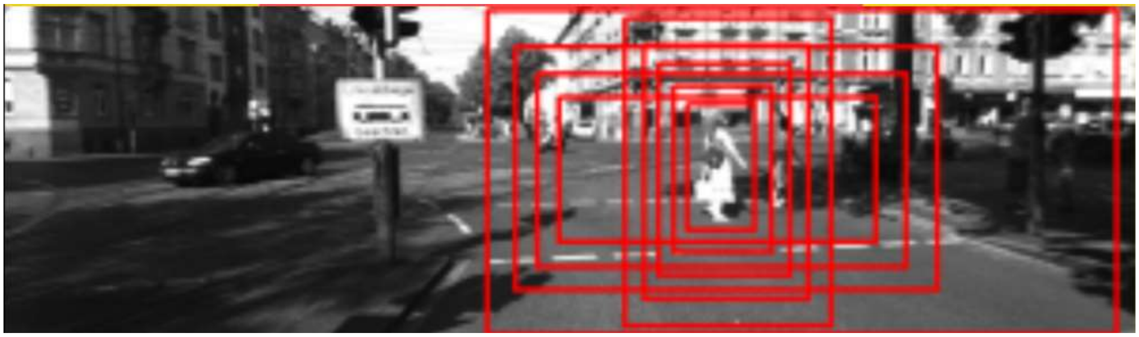
Divide and Conquer

This is a naive way for image classification where you divide the image into four equal parts and then analyse every part generate predictions. But the problem was that the object of interest was one of the four parts of the image and exact location was not identified.



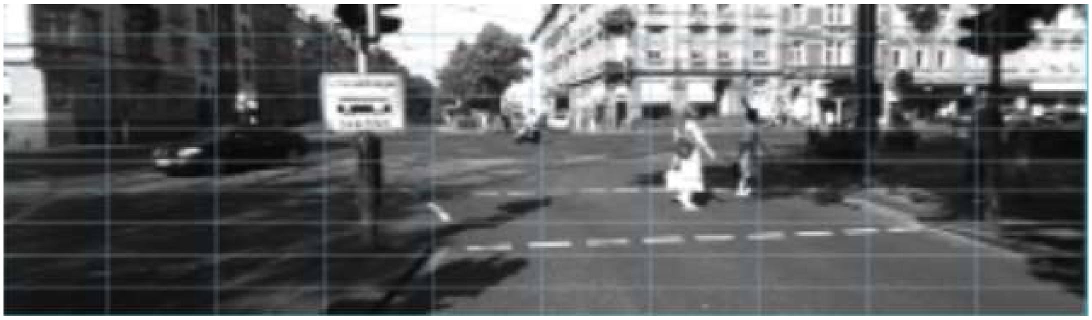
Increase the number of divisions

This approach works better than the previous one as you improve by exponentially increasing the number on patches in an image. But the problem was there were so many bounding boxes with approximated the same thing.

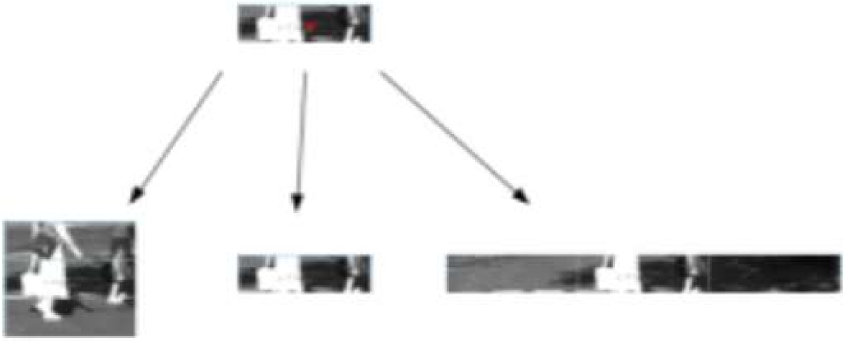


Performing Structured divisions

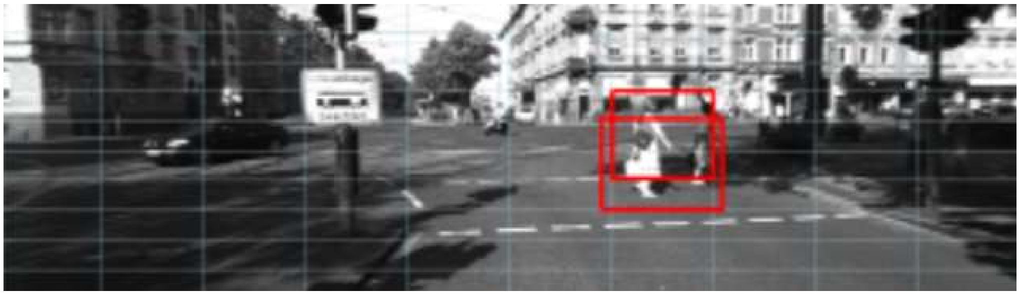
This approach divided the image in a 10x10 grid and assigns a centroid for each patch



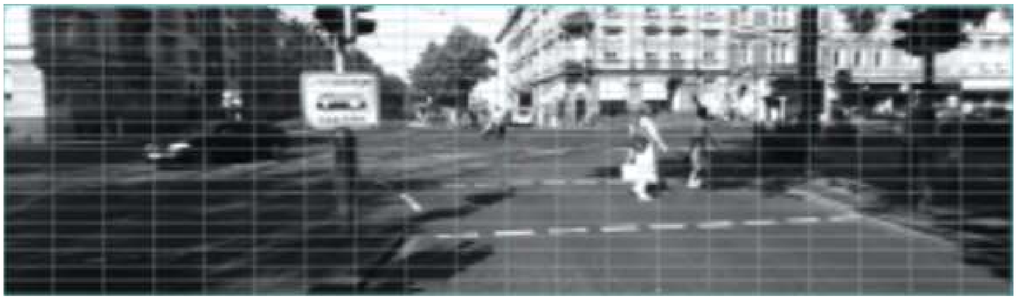
For each centroid we take three different patches of different heights and aspect ratio



Then we pass all the created patches to image classifier and generate predictions.



This approach works even better if we increase the grid size and number of patches within each centroid or take selective patches instead of feeding all images into the classifier.



Challenges

The classifier works fine for predicting images that look similar to the trained dataset but lacks in accuracy when there is,

Variation in Viewpoint

Image classifier is not able to predict the same image if there is variation of viewing angle of that image. As seen in figure above same object has been viewed from different angles but the classifier would not be able to predict all angles as it is not trained for it.

Difference in Illumination

Different lightning conditions make it difficult for the classifier to predict results or filter out the object of attention.

Hidden Images

The classifier was not able to predict if the object of attention was partly hidden or cropped.

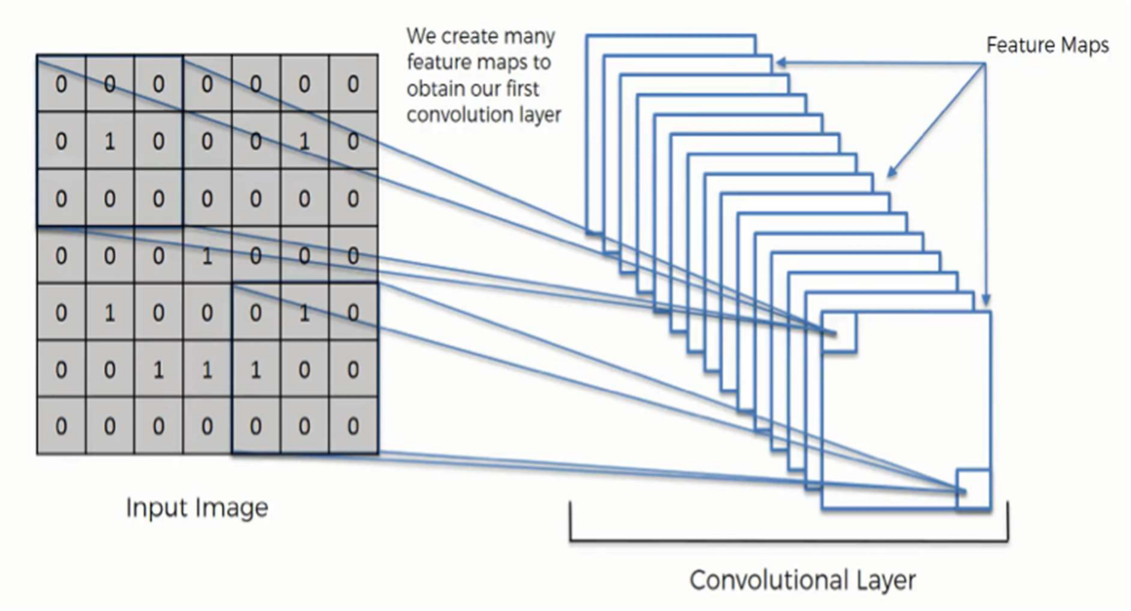
Background Clutter

Background clutter in one another challenge where image classifier is not able to predict results.

Convolutional Neural Network

Convolutional Neural Network provides a multilayer architecture to the image classifier that contain mainly four operations i.e. convolve, activate, max-pooling and fully connected layers. Each layer has its on function on an image which is either an RGB image matrix or a grayscale image matrix. To understand the functioning of each layer I tried implementing them step by step. Below is my process,

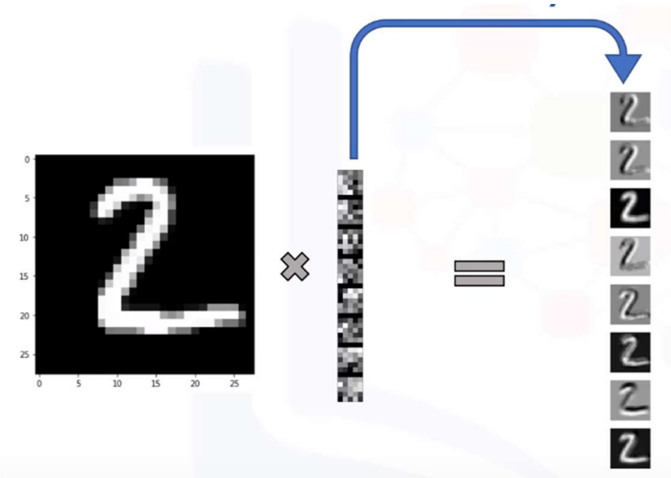
Step 1: Convolve



0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

We create many feature maps to obtain our first convolution layer
Feature Maps
Input Image
Convolutional Layer

We apply various kernels to an RGB or a grayscale image such as edge detection, blurring, sharpening and more. After these filters are applied to matrix they form feature maps. Collection of such feature maps form our first convolution layer.

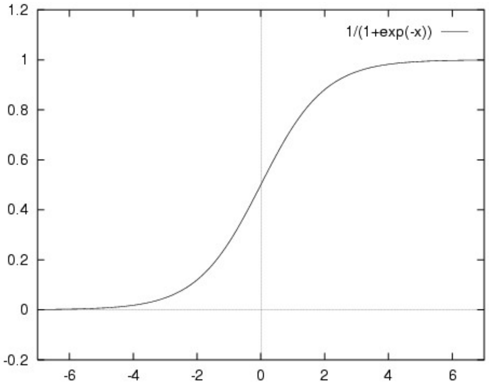


Step 2: Activation

The aim of this step is to bring non-linearity in our model so that is generalizes and adapts to different images in the convolution layer. Hyperbolic tangent (tanh), Sigmoid and Rectified Linear Unit are such functions.

Sigmoid Activation function:

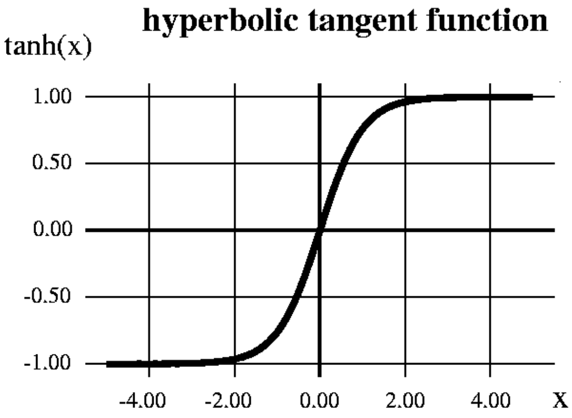
f(x) = 1 / 1 + exp(-x)



Output range for Sigmoid ranges from 0 to 1 so vanishing gradient is a major problem. Also, it makes optimization harder as it is zero centered.

Hyperbolic Tangent function- Tanh:

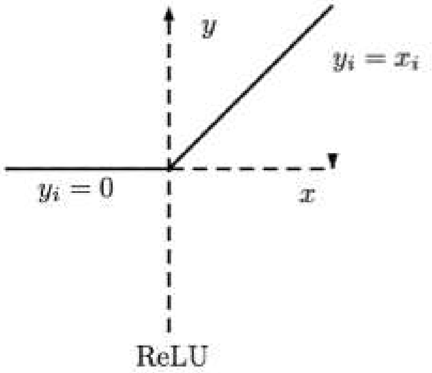
f(x) = 1 — exp(-2x) / 1 + exp(-2x)



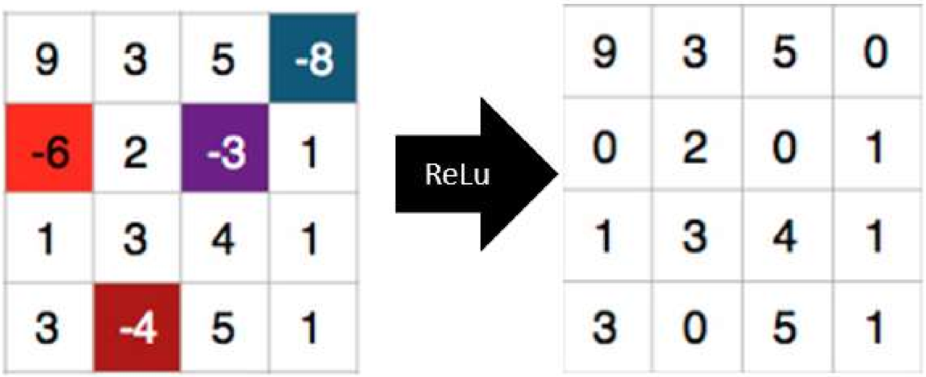
As the range is from -1 to 1 optimization is better in tanh compared to sigmoid but it suffers from vanishing gradient problem.

ReLu- Rectified Linear units:

R(x) = max(0, x) i.e. if x < 0 , R(x) = 0 and if x >= 0 , R(x) = x



This solves the vanishing gradient problem as it is easy for the model to generalize or adapt with variety of data and to differentiate between the output is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero.



9	3	5	-8
-6	2	-3	1
1	3	4	1
3	-4	5	1

9	3	5	0
0	2	0	1
1	3	4	1
3	0	5	1

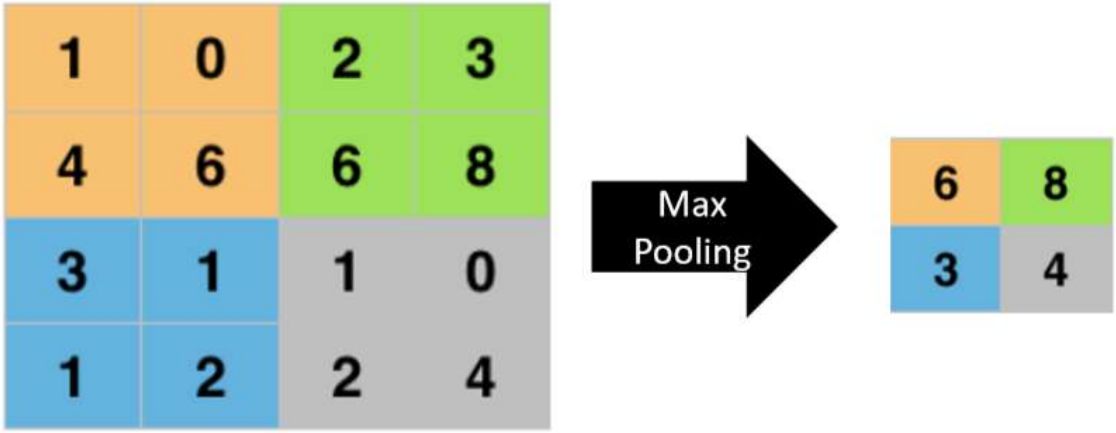
Step 3: Pooling

If the same image is represented with changes dimensions such as rotated from one side or squashed horizontally or vertically for example if there are five different images of a dog where it is standing, sitting, zoomed in or out, different angles and textures, then it is necessary to find a method which can overcome such challenges and still find the required object of attention inside that challenging set of images. Pooling is a solution that teaches our convolutional neural network to recognize object despite of all challenges by working on a property called "spatial variance".

There are mainly three types of spatial pooling that can be implemented after the activation function

- Mean pooling
- Max pooling
- Sum pooling

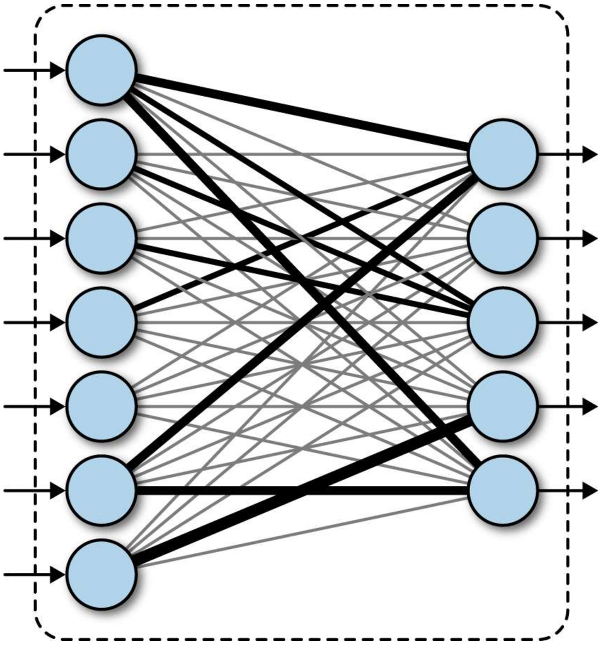
Pooling layers would reduce the number of parameters by using a 2-pixel stride which will return you a 3x3 pooled feature map starting from the top-left corner as you move along the row.



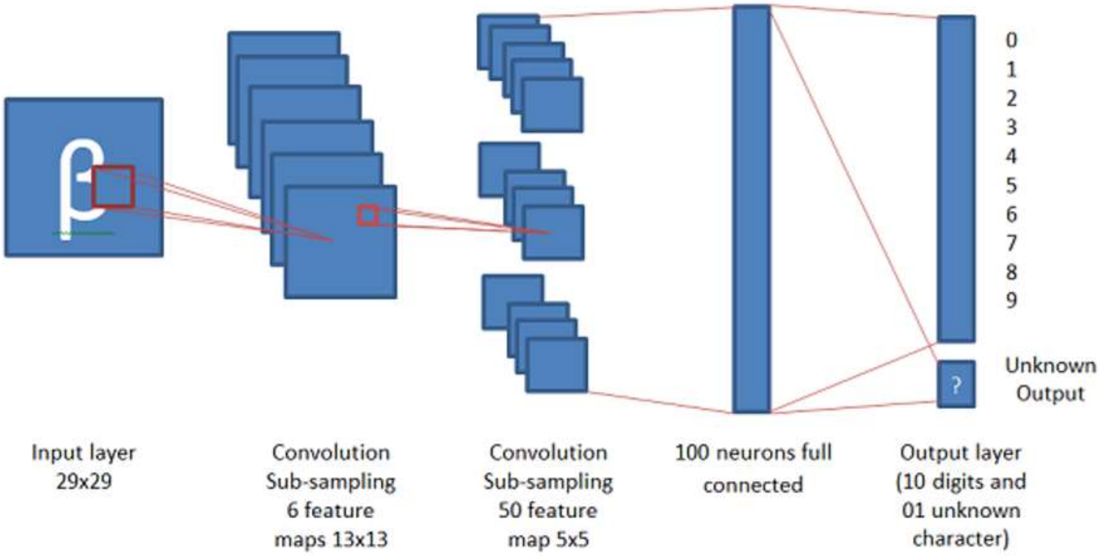
As seen in above figure a 4x4 matrix returns a 2x2 pooled feature map, we begin by using a 2-pixel stride from top-left corner and find out the maximum value which is 6 in our example. We follow the same process row by row until we end up with a pooled feature map.

Step 4: Fully Connected Layer

A fully connected neural network is a series of fully connected layers with a function $\mathbb{R}^m$ to $\mathbb{R}^n$ i.e. each input dimension is responsible for each output dimension.



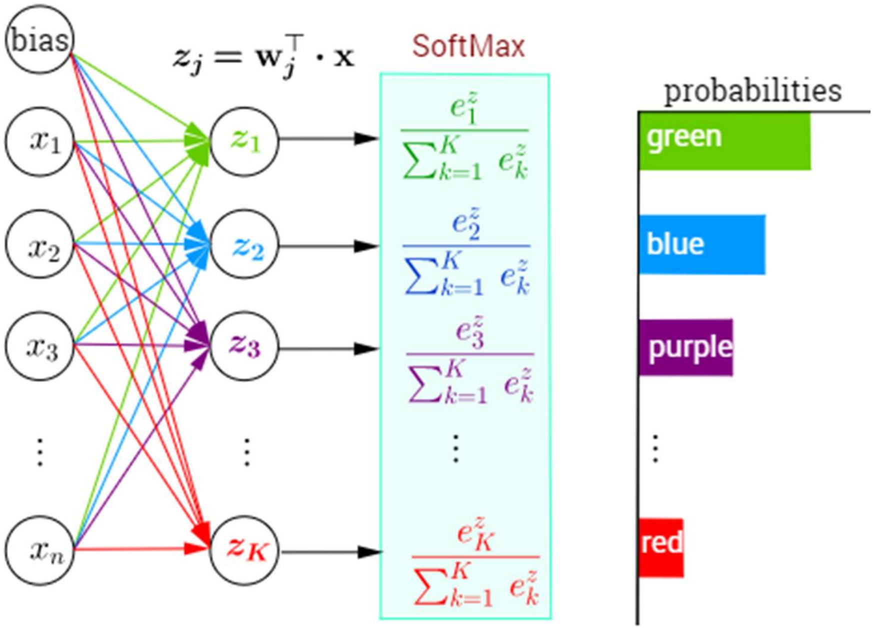
The feature maps from pooling layers are converted to vectors and then feed into fully connected layer. Each node is termed as a neuron which combines together and creates a model which is then applied SoftMax to that it returns results.


β
0
1
2
3
4
5
6
7
8
9
?
Unknown Output
Input layer 29x29
Convolution Sub-sampling 6 feature maps 13x13
Convolution Sub-sampling 50 feature map 5x5
100 neurons full connected
Output layer (10 digits and 01 unknown character)

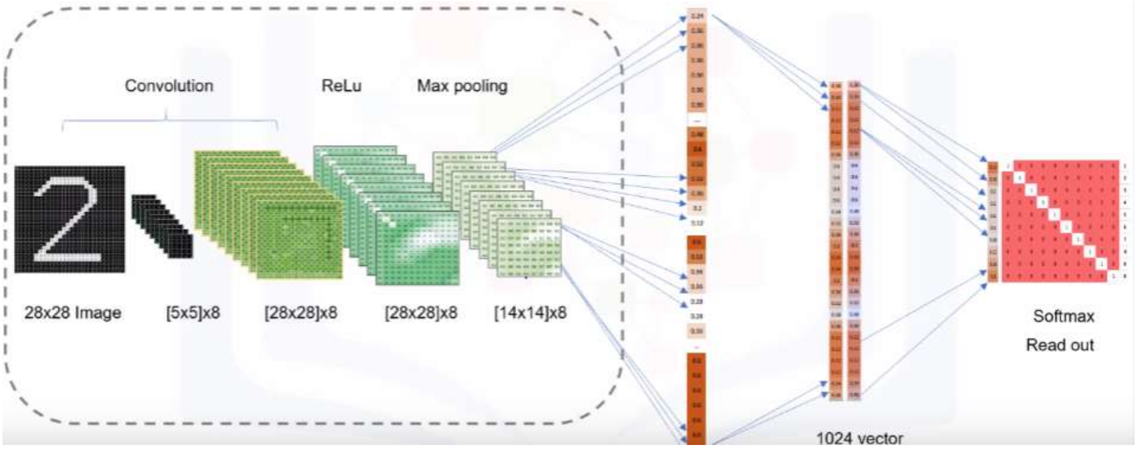
As you can see in this image a 29x29 image generates a fully connected layer with 100 neurons with the help of 50- 5x5 feature maps.

SoftMax

A neural network uses logistic regression to calculate probabilities for predicting target classes. SoftMax calculates the probability of target result over n events with outputs ranging from zero to one and sum of all probabilities being one. As a result it will return the target class with highest probability.

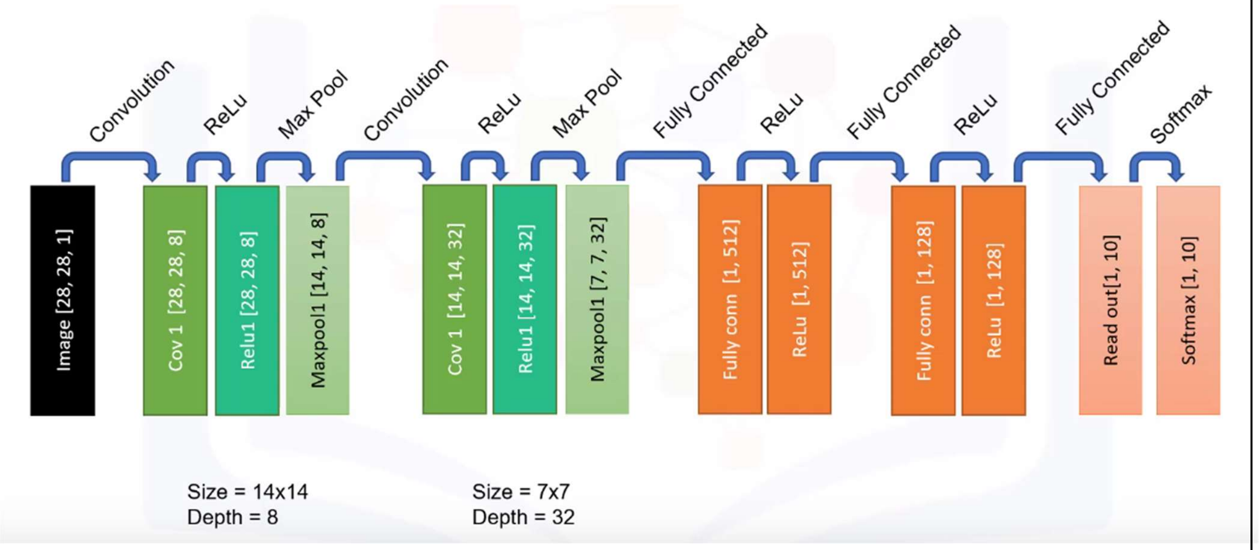

bias
x_1
x_2
x_3
x_n
$z_j = \mathbf{w}_j^\top \cdot \mathbf{x}$
z_1
z_2
z_3
z_K
SoftMax
$\frac{e_1^z}{\sum_{k=1}^K e_k^z}$
$\frac{e_2^z}{\sum_{k=1}^K e_k^z}$
$\frac{e_3^z}{\sum_{k=1}^K e_k^z}$
$\frac{e_K^z}{\sum_{k=1}^K e_k^z}$
probabilities
green
blue
purple
red

Architecture



The architecture can be said to be made up of mainly five layers as seen in above figure. A 28x28 image is convolved using eight 5x5 kernels which returns eight 28x28 convolved images which are activated using the Rectified Linear Unit function and feed to the pooling layer to extract eight 14x14 feature maps. Those feature maps behave as an input for the fully connected layer which then applies Softmax function to return target with highest probability.

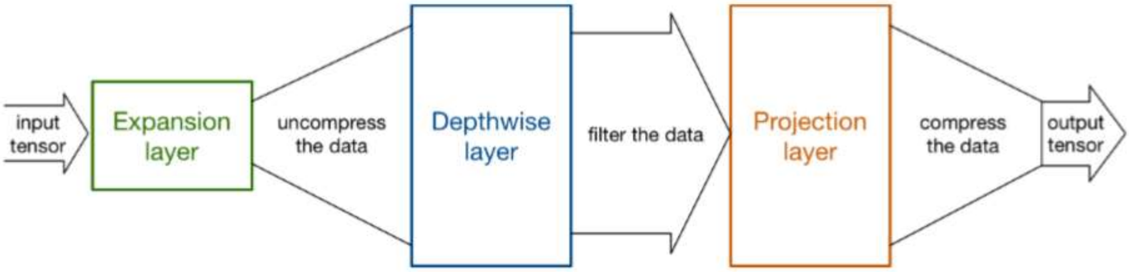
Each of these layers can be used indefinitely as per requirement unless it is found to be over-fitting. As seen in below figure each of the layers are applied twice which creates our architecture.



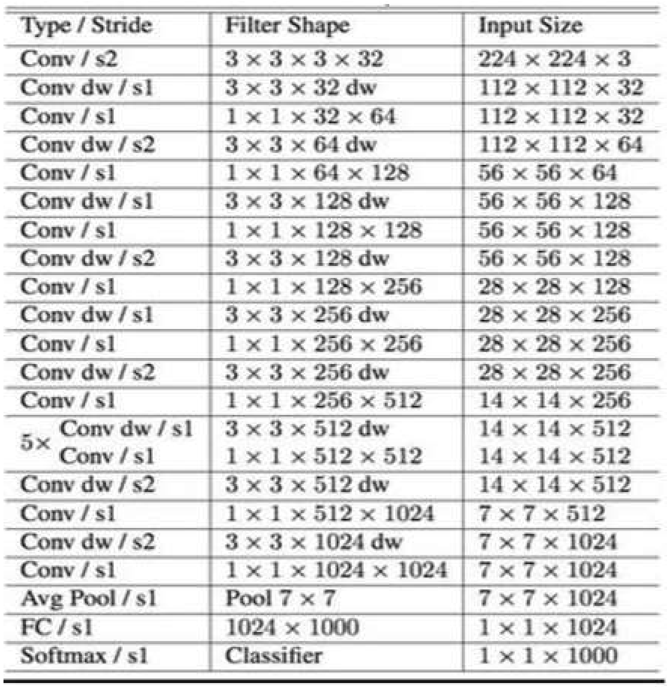
Implementation

MobileNet

MobileNet is a model provided by TensorFlow which provides faster predictions but trades-off accuracy. It can work on low hardware configurations which was the reason I chose this model.



MobileNet layers are divided into three categories which are expansion, depth-wise and projection layers. Each layer has its own set of layers which involve convolve, activation, pooling and fully connected layers. This architecture follows dense approach rather than shallow approach for a convolutional neural network. Below is the architecture of TensorFlow MobileNet.



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

Emotion Detection

The first step for emotion detection was to gather real-time data from webcam which returns a canvas object.

```
function captureWebcam() {
        var video = document.getElementById("main-stream-video");
        var canvas   = document.createElement("canvas");
        var context  = canvas.getContext('2d');
        canvas.width  = video.width;
        canvas.height = video.height;
        context.drawImage(video, 0, 0, video.width, video.height);
        tensor_image = preprocessImage(canvas);

        var canvasObj = {
        canvasElement: canvas,
        canvasTensor : tensor_image
        };
        return canvasObj;}
```

Now we need a trained model to predict the probability of target in canvas object. We follow the below steps

1. Load MobileNet Model
2. Capture Sample
3. Pre-Process Image
4. Add Sample to Tensor
5. Feed tensor to model and train
6. Start prediction

Step 1: Load Model

```
        // load mobilenet from Google
        const mobilenet = await
        tf.loadLayersModel("https://storage.googleapis.com/tfjs-
        models/tfjs/mobilenet_v1_0.25_224/model.json");
        // return the mobilenet model with internal activations from
        "conv_pw_13_relu" layer
        const feature_layer = mobilenet.getLayer("conv_pw_13_relu");
        // return mobilenet model with feature activations from specific layer
        extractor = tf.model({inputs: mobilenet.inputs, outputs:
        feature_layer.output});
```

Step 2: Capture Sample

```
canvasObj = captureWebcam();
canvas = canvasObj["canvasElement"];
tensor_image = canvasObj["canvasTensor"];
var img_id = id.replace("sample", "image");
var img    = document.getElementById(img_id);
img.src    = canvas.toDataURL();
// add the sample to the training tensor
addSampleToTensor(extractor.predict(tensor_image), label);
SAMPLE_BOX[label] += 1;
```

Step 3: Pre-Process Image

```
const tensor = tf.browser.fromPixels(img).resizeNearestNeighbor([224,
224]);
const croppedTensor = cropImage(tensor);
const batchedTensor = croppedTensor.expandDims(0);
//return  Batched Tensor for capture image
return batchedTensor.toFloat().div(tf.scalar(127)).sub(tf.scalar(1));
```

Step 4: Add Sample to Tensor

```
const y = tf.tidy(() => tf.oneHot(tf.tensor1d([label]).toInt(),
NUM_CLASSES));
const oldX = xs;
xs = tf.keep(oldX.concat(sample, 0));
const oldY = ys;
ys = tf.keep(oldY.concat(y, 0));
oldX.dispose();
oldY.dispose();
y.dispose();
// hold each sample as a tensor that
// has 4 dimensions
```

Step 5: Feed Tensor to model and train

```
//Create a Classifier

        classifier = tf.sequential({
            layers: [
                tf.layers.flatten({inputShape: [7, 7, 256]}),
                tf.layers.dense({
                    units: parseInt(hiddenUnits),
                    activation: "relu",
                    kernelInitializer: "varianceScaling",
                    useBias: true
                }),
                tf.layers.dense({
                    units: parseInt(NUM_CLASSES),
                    kernelInitializer: "varianceScaling",
                    useBias: false,
                    activation: "softmax"
                })]});
// create loss visualization


        var lossTextEle = document.getElementById("emotion-loss");
        if (typeof(lossTextEle) != 'undefined' && lossTextEle != null) {
            lossTextEle.innerHTML = "";
        } else {
            var lossText = document.createElement("P");
            lossText.setAttribute("id", "emotion-loss");
            lossText.classList.add('emotion-loss');
            document.getElementById("emotion-controller").insertBefore(lossText,
            document.getElementById("emotion-controller").children[1]);
            var lossTextEle = document.getElementById("emotion-loss");
        }
        classifier.fit(xs, ys, {
            batchSize,
            epochs: parseInt(epochs),
            callbacks: {
                onBatchEnd: async (batch, logs) => {
                    lossTextEle.innerHTML = "Loss: " +
                    logs.loss.toFixed(5);
                    await tf.nextFrame();
                }
            }
});
```

Step 6: Predict

```
var SAMPLE_BOX = {0: 0, 1: 0, 2: 0, 3: 0, 4: 0}
//Define Emotions Class Map
var CLASS_MAP = {
        0: "emoticon-laugh",
        1: "emoticon-excited",
        2: "emoticon-sad",
        3: "emoticon-angry",
        4: "emoticon-sleep"
}
const predictedClass = tf.tidy(() => {
                //Capture Frame
                canvasObj = captureWebcam();
                canvas = canvasObj["canvasElement"];
                const img = canvasObj["canvasTensor"];
                //Extract Predictions
                const features = extractor.predict(img);
                const predictions = classifier.predict(features);
                //Return Predictions
                return predictions.as1D().argMax();
        });

  const classId = (await predictedClass.data())[0];
  predictedClass.dispose();
  highlightTile(classId);
  await tf.nextFrame();
```

Sight Detection

Instead of using the predefined MobileNet model I tried implementing the same architecture in this application. The process includes the following main steps

1. Detect Face and crop eyes
2. Create model
3. Create Dataset
4. Train model
5. Predict

Step 1: Detect Face and Crop Eyes

```
// Given a tracked face using clmtracker library, crops out the eyes and
draws them in the eyes canvas.
const rect = facetracker.getEyesRect(position);
facetracker.currentEyeRect = rect;
const eyesCanvas = document.getElementById('eyes');
const eyesCtx = eyesCanvas.getContext('2d');
// Resize because the underlying video might be a different resolution:
const resizeFactorX =
  facetracker.videoWidthInternal / facetracker.videoWidthExternal;
const resizeFactorY =
  facetracker.videoHeightInternal / facetracker.videoHeightExternal;
facetracker.overlayCC.strokeStyle = 'red';
facetracker.overlayCC.strokeRect(rect[0], rect[1], rect[2], rect[3]);
eyesCtx.drawImage( facetracker.video,
  rect[0] * resizeFactorX,
  rect[1] * resizeFactorY,
  rect[2] * resizeFactorX,
  rect[3] * resizeFactorY,
  0,
  0,
  eyesCanvas.width,
  eyesCanvas.height,
);
```

Step 2: Create Model

```
const inputImage = tf.input({                    //Take Input image
 name: 'image',
 shape: [dataset.inputHeight, dataset.inputWidth, 3],
});
const inputMeta = tf.input({name: 'metaInfos', shape: [4], });
const conv = tf.layers.conv2d({                        //Convolve 2d Layer
  kernelSize: 5,
  filters: 20,
  strides: 1,
  activation: 'relu',
  kernelInitializer: 'varianceScaling', }).apply(inputImage);
const maxpool = tf.layers.maxPooling2d({        //MaxPooling layer
  poolSize: [2, 2],
  strides: [2, 2],}).apply(conv);
const flat = tf.layers.flatten().apply(maxpool);  //Flatten and apply pooling
const dropout = tf.layers.dropout(0.2).apply(flat);
const concat = tf.layers.concatenate().apply([dropout, inputMeta]);
const output = tf.layers                               //Dense Neural Network
 .dense({
  units: 2,
  activation: 'tanh',
  kernelInitializer: 'varianceScaling',
 })
 .apply(concat);
const model = tf.model({
 inputs: [inputImage, inputMeta],
 outputs: output,
});
return model;
```

Step 3: Create Dataset

Capture Image

```
// Capture the current image in the eyes canvas as a tensor.
  return tf.tidy(function() {
   const image = tf.fromPixels(document.getElementById('eyes'));
   const batchedImage = image.expandDims(0);
   return batchedImage
    .toFloat()
    .div(tf.scalar(127))
.sub(tf.scalar(1));
```

Information about meta data

```
// - middle x, y of the eye rectangle, relative to video size
// - size of eye rectangle, relative to video size
// - angle of rectangle (TODO)
let x = facetracker.currentEyeRect[0] + facetracker.currentEyeRect[2] / 2;
let y = facetracker.currentEyeRect[1] + facetracker.currentEyeRect[3] / 2;
x = (x / facetracker.videoWidthExternal) * 2 - 1;
y = (y / facetracker.videoHeightExternal) * 2 - 1;
const rectWidth =
  facetracker.currentEyeRect[2] / facetracker.videoWidthExternal;
const rectHeight =
  facetracker.currentEyeRect[3] / facetracker.videoHeightExternal;
if (mirror) {
  x = 1 - x;
  y = 1 - y; }
return tf.tidy(function() {
  return tf.tensor1d([x, y, rectWidth, rectHeight]).expandDims(0);});
```

Convert RGB to Grayscale

```
// Given an rgb tensor, returns a grayscale value.
  let r = (image.get(n, x, y, 0) + 1) / 2;
  let g = (image.get(n, x, y, 1) + 1) / 2;
  let b = (image.get(n, x, y, 2) + 1) / 2;
  // Gamma correction:
  const exponent = 1 / 2.2;
  r = Math.pow(r, exponent);
  g = Math.pow(g, exponent);
  b = Math.pow(b, exponent);
  // Gleam:
  const gleam = (r + g + b) / 3;
  return gleam * 2 - 1;
```

Add Spatial Information

```
// Convert to grayscale and add spatial info
  const imageShape = image.shape;
  const w = imageShape[1];
  const h = imageShape[2];
  const data = [new Array(w)];
  for (let x = 0; x < w; x++) {
   data[0][x] = new Array(h)
    for (let y = 0; y < h; y++) {
      data[0][x][y] = [
        dataset.rgbToGrayscale(image, 0, x, y),
        (x / w) * 2 - 1,
        (y / h) * 2 - 1,
```

And then add image to either train or val dataset based on usecase

Step 4: Train Model

```
//Check if there is a model present
if (training.currentModel == null) {
  training.currentModel = training.createModel();
 }
//Set Parameters in UI
 let bestEpoch = -1;
 let bestTrainLoss = Number.MAX_SAFE_INTEGER;
 let bestValLoss = Number.MAX_SAFE_INTEGER;
 const bestModelPath = 'localstorage://best-model';
 training.currentModel.compile({
  optimizer: tf.train.adam(0.0005),
    loss: 'meanSquaredError',
    if (logs.val_loss < bestValLoss) {
          // Save model
          bestEpoch = epoch;
          bestTrainLoss = logs.loss;
          bestValLoss = logs.val_loss;
     // Store best model:
     await training.currentModel.save(bestModelPath);
    }
    // Analyse Next Frame
    return await tf.nextFrame();
```

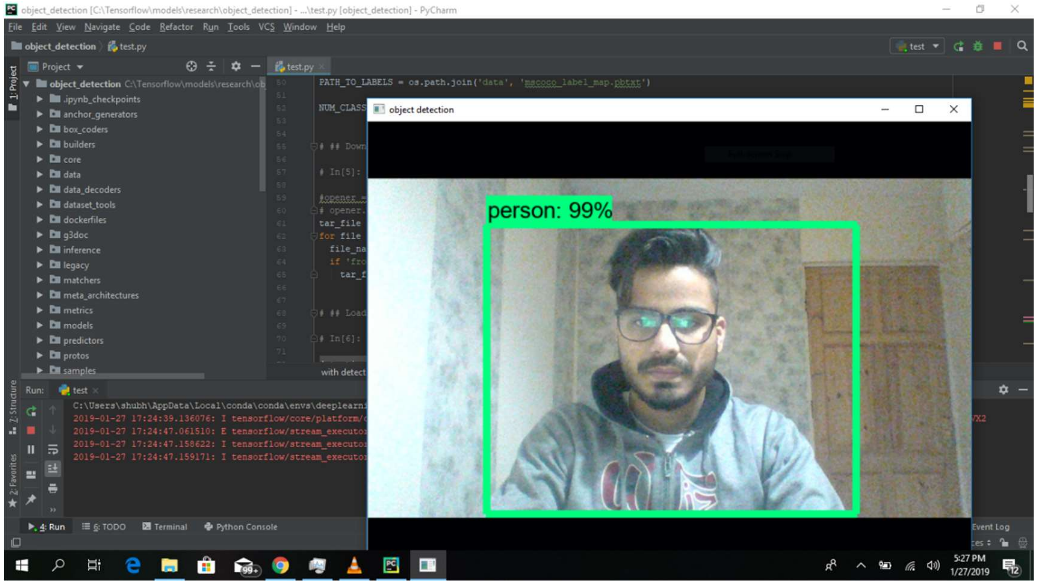
Step 5: Predict

```
getPrediction: function() {
  // Return relative x, y where we expect the user to look right now.
  return tf.tidy(function() {
   let img = dataset.getImage();
   img = dataset.convertImage(img);
   const metaInfos = dataset.getMetaInfos();
   const prediction = training.currentModel.predict([img, metaInfos]);

   return [prediction.get(0, 0) + 0.5, prediction.get(0, 1) + 0.5];
  });
},
```

Design

Basic Object Detection Model

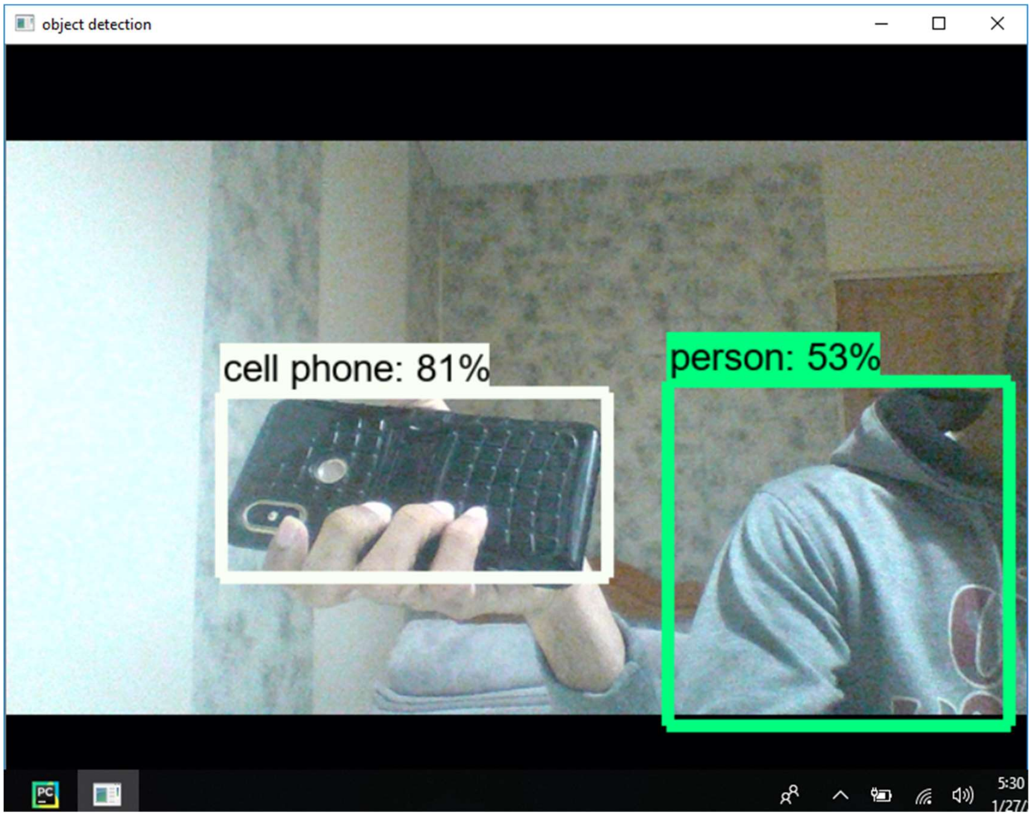
Single Object Detection



```
PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')

NUM_CLASS

# ## Down

# In[5]:

#opener =
# opener.
tar_file
for file
  file_na
  if 'fro
    tar_f

# ## Load

# In[6]:

with detect
```

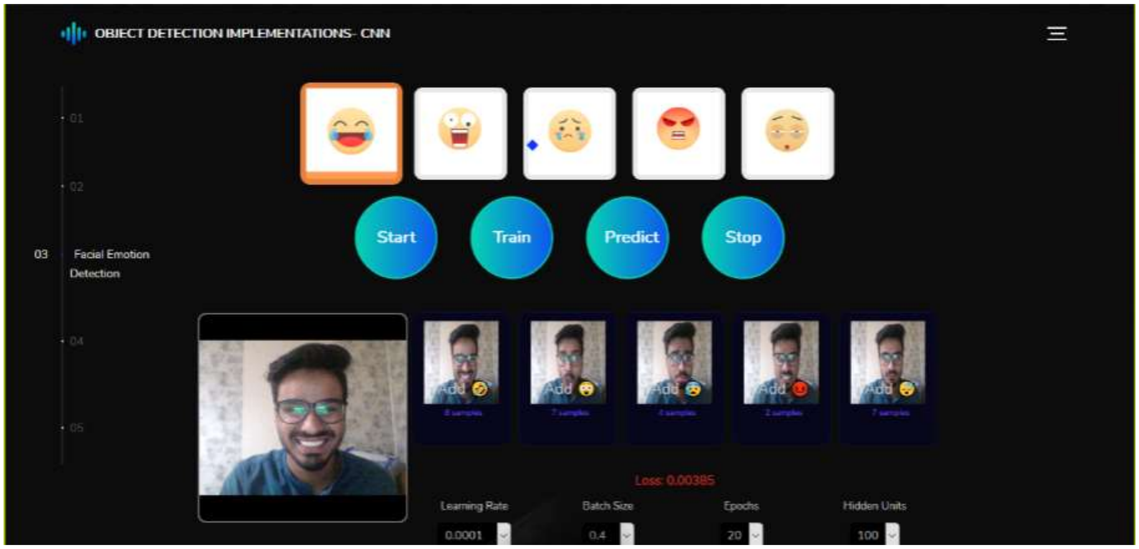
```
C:\Users\shubh\AppData\Local\conda\conda\envs\deeplearni
2019-01-27 17:24:39.136076: I tensorflow/core/platform/c
2019-01-27 17:24:47.061510: E tensorflow/stream_executor
2019-01-27 17:24:47.158622: I tensorflow/stream_executor
2019-01-27 17:24:47.159171: I tensorflow/stream_executor
```

Multiple Object Detection



Emotion Detection

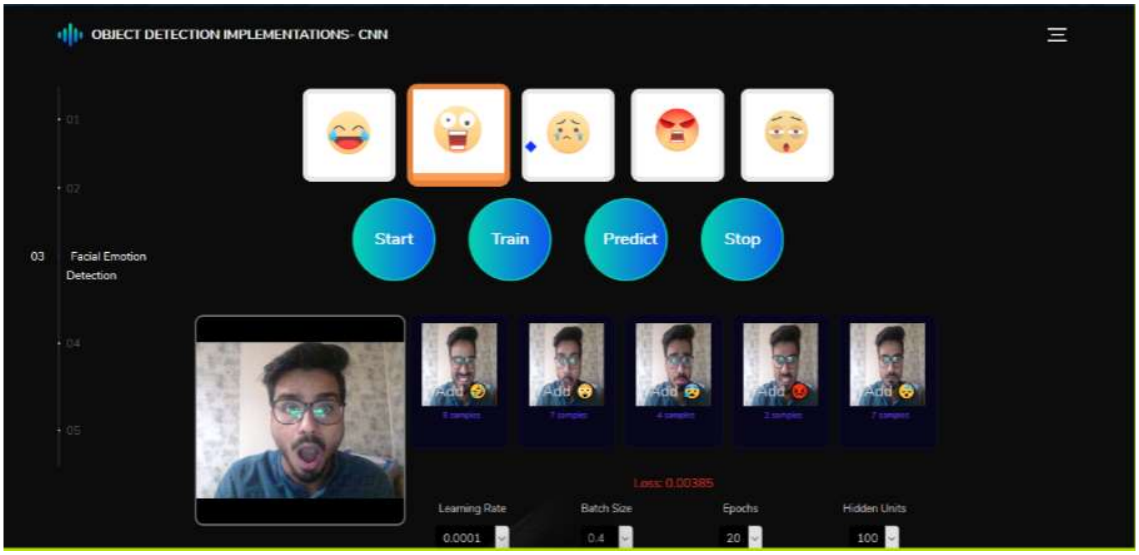
Laugh



Loss: 0.00385

Learning Rate 0.0001
Batch Size 0.4
Epochs 20
Hidden Units 100

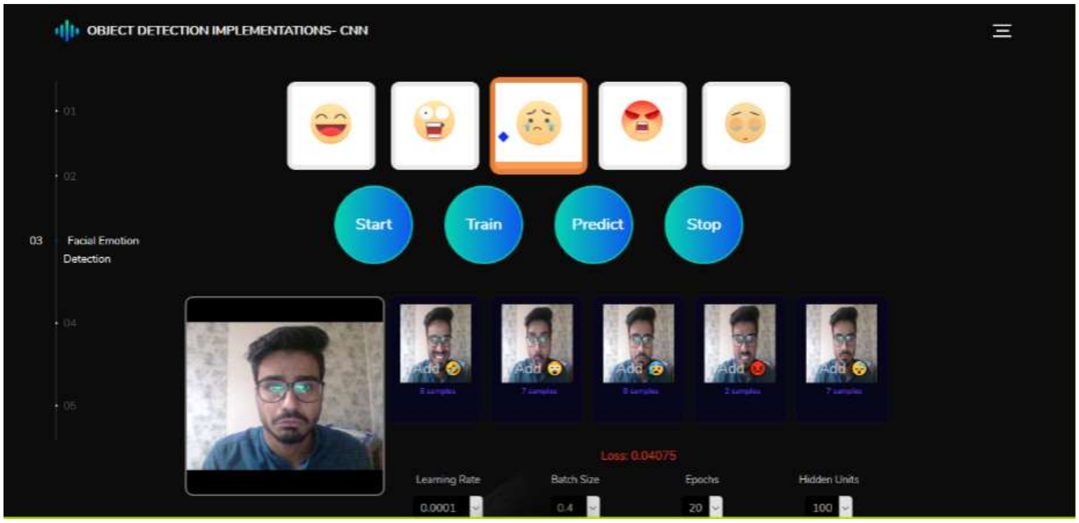
Excited



Loss: 0.00385

Learning Rate 0.0001
Batch Size 0.4
Epochs 20
Hidden Units 100

Sad



Loss: 0.04075

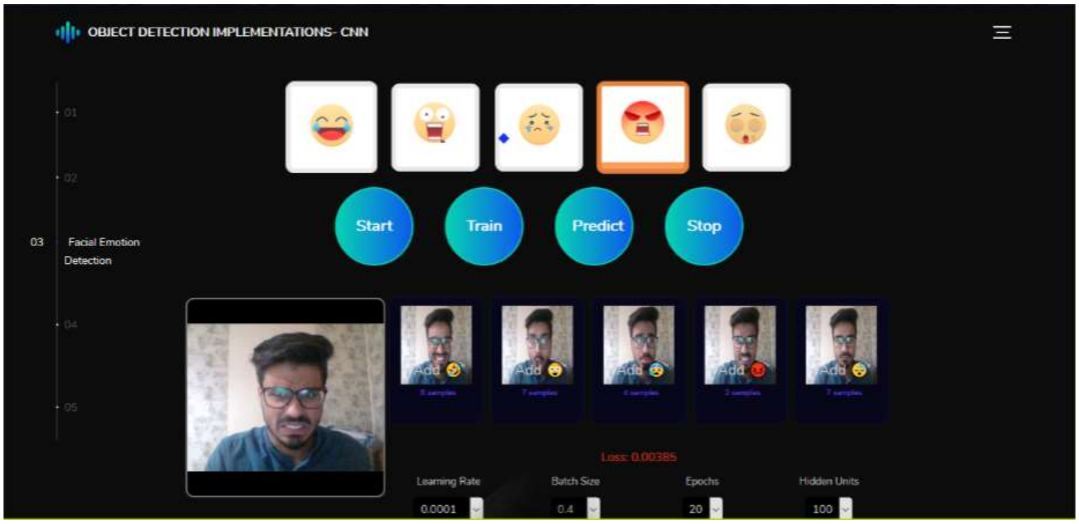
Learning Rate 0.0001

Batch Size 0.4

Epochs 20

Hidden Units 100

Angry



Loss: 0.00385

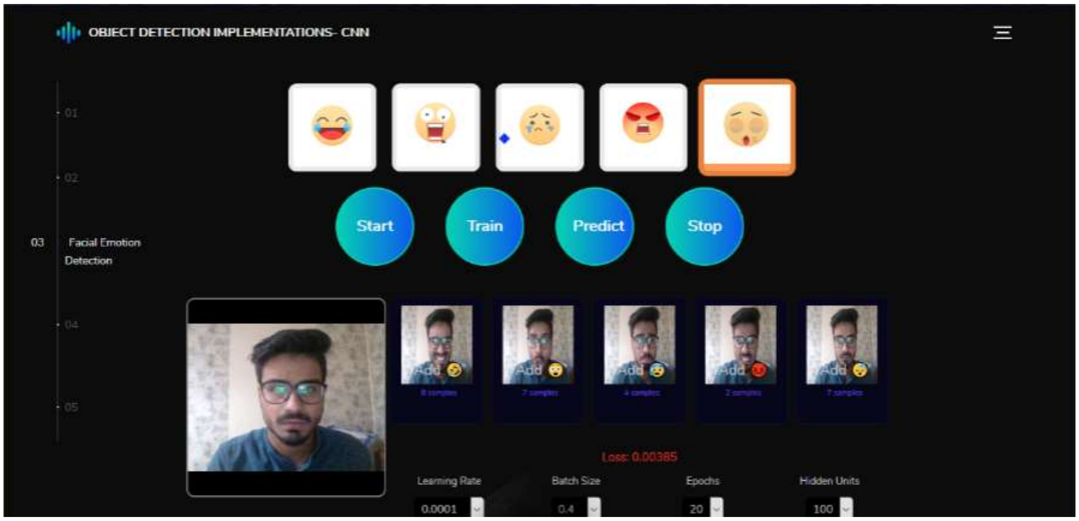
Learning Rate 0.0001

Batch Size 0.4

Epochs 20

Hidden Units 100

Neutral



Loss: 0.00385

Learning Rate 0.0001

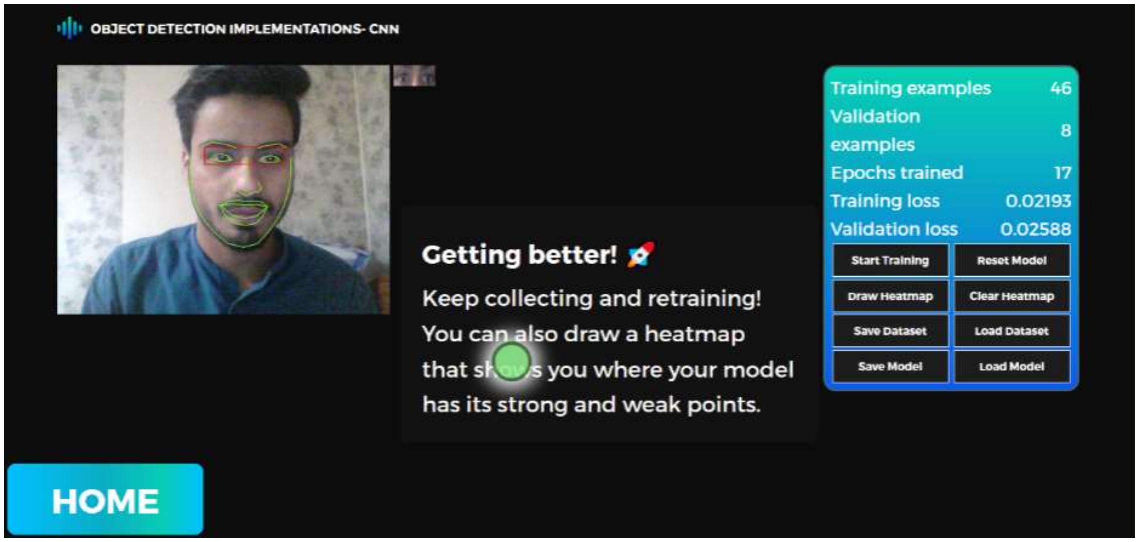
Batch Size 0.4

Epochs 20

Hidden Units 100

Sight Detection

Detection when I was looking at the centre.

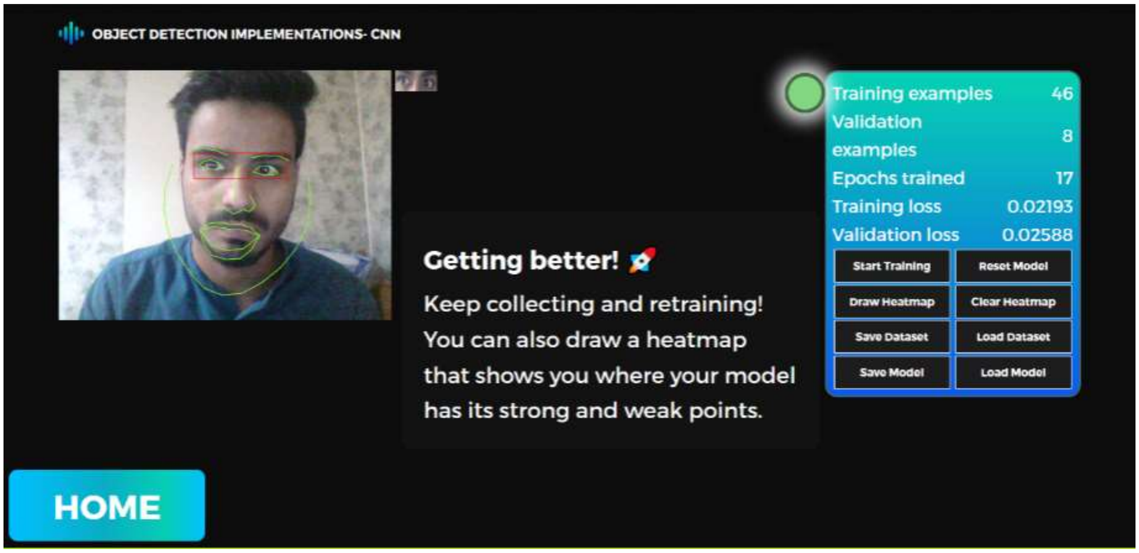


Training examples 46
Validation examples 8
Epochs trained 17
Training loss 0.02193
Validation loss 0.02588

Getting better! 🚀

Keep collecting and retraining! You can also draw a heatmap that shows you where your model has its strong and weak points.

Detection when I was looking on top-right.



Training examples 46
Validation examples 8
Epochs trained 17
Training loss 0.02193
Validation loss 0.02588

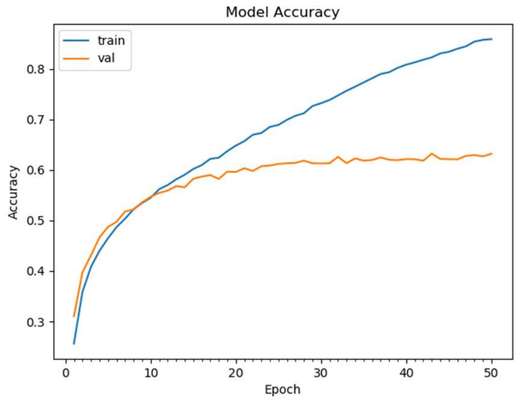
Getting better! 🚀

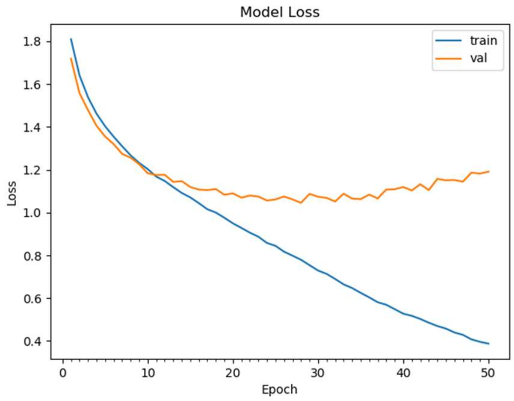
Keep collecting and retraining! You can also draw a heatmap that shows you where your model has its strong and weak points.

Results

The network architecture was able to learn things quickly but converges to an accuracy of 64% which still is quite an achievement looking at the datasets provided. The emotion detection model predicts the target object when trained with at least 20 epochs with 10 images per sample emotion while it was found overfitting when trained with more than 60 epochs where neutral and sad or angry were often confused with. For sight detection the convolutional network was able to achieve accuracy when trained with at least 20 training samples for 40 epochs. When trained with more sample higher accurate predictions were possible but increasing the number of epochs made it overfitting after 80.

Below is a diagram which depicts the model accuracy and model loss for training data and for prediction data. Model accuracy was found increasing for training data with respect to epochs while validating that model after 10 epochs generated no actual increase in accuracy. Model loss kept decreasing with epochs and were found to be constant and fluctuating after 10 epochs.





Conclusion

This project helped me understand the working of convolutional neural network for object detection and classification. The scope of the project was to provide an application that detects facial emotion and detects sight which were accomplished with use of an efficient CNN architecture. Various approaches from the beginning were applied which were studied thoroughly for merits and cons. Alternative for each approach was found step by step by studying various algorithms and techniques for find scope of improvement.

The architecture of model was decided after some research on previous state-of-the-art literatures that provided some efficient architecture design for the current convolution network. Also, the efficiency was optimised based on device configurations as low-quality webcam as well as low configuration processor was utilized for demonstration. Thus, a suitable architecture model that provides accuracy even with standard device configurations was an aim which was achieved through sheer testing of results on different models.

The model predicts emotions and sight with accuracy given good lighting conditions and background eliminations due to the state-of-the-art architecture of MobileNets which uses Depthwise Separable Convolution (DWConvolution) in place of standard Convolution to reduce the size of networks making it possible to work on low processing power devices. I have tested that this model works really well even with a small number of training data. I measured how the accuracy depends on the number of epochs in order to detect potential overfitting problem. I determined that 10 epochs are enough for a successful training of the model. My next step would be to try this model on more data sets such as fer2013, Japanese Female Facial Expression (JAFFE) database and Labeled Faces in the Wild (LFW) dataset for emotion detection. I want to work more on the visualizing the learned filters in depth, and in the future, we also want to take a semi-supervised approach by using the predictions made for the LFW images, to train the network with more data, more filters, and more depth.

Future Scope

Emotion Detection

Facial emotion detection has various use-cases in the world of artificial intelligence and machine learning. This project provided an architecture that would efficiently detect human facial emotions even on devices with low processing powers such as mobile phones. By using this design various applications can be implemented that could minimise the use of costly sensors for home automation devices.

Imagine being remotely able to detect if a patient requires medical assistance just by detecting the facial emotions and trigger a distress signal when a certain emotion is captured for a predicted time interval.

Facial emotion detection can be very useful for self-driving cars, detecting facial emotion of person driving the car or the passengers on other seats can have many security as well as luxurious benefits. Angry drivers have a tendency to drive faster which can result into catastrophic events, detecting this emotion we could cap car's speed limit when driver is angry, or open windows when passenger is not feeling well and various others.

Facial emotion detection can be very safe and stop theft if installed on ATM machines, if the algorithm detects that the person is scared which can mean there is a knife or a gun pointed at him during a pre-decided time interval; it will send a distress signal to the respective garda station and help could be sent out.

Facial emotion detection for customer satisfaction surveys can be very useful. By detecting human faces for satisfaction results after presentations or use of products can provide genuine feedbacks rather than words.

E-learning is a vast area that needs improvement, webcams can constantly detect facial emotions of students to provide feedbacks about how satisfied the student is with the content or lecturer.

Sight Detection

Sight detection has a vast future scope in terms of technology and use. Imagine a world where you walk into your house and with certain gestures such as a 2 seconds wink or rolling up your eyes or just by a long gaze an event is triggered that turns on your devices such as television, lights, smart mirror and many more.

Sight detection for mobile devices can be a great future application area where users control their devices with eyes and does operations such as scrolling up or down, or clicking an object with a pre-defined set of gestures. We have come from an era where keypads were used and touch screen is the new technology but with sigh detection this can be a touch free operation technology.

Sight detection can be very useful for websites and businesses that need ads to sustain business. By detecting where the reader or the person browsing through the webpage is mostly looking at the webpage; the area of interest which has an article or a blog can be detected and featured for individual recommendations. This could help in bringing for traffic on websites and providing more content that is relevant to user preference without asking for it.

Sight detection can be used in automation industry where the operator's eyes can be detecting to find if he is looking at the area on automation led screen which requires his attention for job to be finished with accuracy. If he misses a step the sight detection can provide alerts which can help avoid loss in manufacturing defective products.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, 2012.

[2] OpenSourceComputerVision. Face detection using haar cascades. URL http://docs.opencv.org/master/d7/d8b/ tutorial_py_face_detection.html.

[3] TFlearn. Tflearn: Deep learning library featuring a higher-level api for tensorflow. URL http://tflearn.org

[4] Dynamics of facial expression: Recognition of facial actions and their temporal Segments from face profile image sequences; M. Pantic and I. Patras,IEEE Trans. Syst., Man, Cybern. B, vol. 36,no. 2, pp. 433 449,2006.

[5] ”3-D facial expression recognition based on primitive surface feature Distribution”,J. Wang, L.Yin, X. Wei, and Y. Sun, Proc. IEEE Conf. Comput. Vis. Pattern Recognition., June 2006, pp.1399-1406.

[6] The TensorFlowAuthors (2017) TensorFlowObject Detection API(readme). URI: https://github.com/TensorFlow/models/tree/master/research/object_detection. Cited November 23, 2017.

[7] Rathod V & Wu N (2017) Object Detection Demo. URI: https://github.com/TensorFlow/models/blob/master/research/object_detection/object_detection_tutorial.ipynb. Cited November 23, 2017.

[8] The TensorFlowAuthors (2017) Supported object detection evaluation protocols. URI: 30https://github.com/TensorFlow/models/blob/master/research/object_detection/g3doc/evaluation_protocols.md. Cited December 12, 2017.

[9] Tran D (2017) How to train your own Object Detector with TensorFlow’s Object DetectorAPI. URI: https://medium.com/towards-data-science/how-to-train-your-own-object-detector-with-TensorFlows-object-detector-api-bec72ecfe1d9. Cited November 5, 2017.

Appendix

The completed demo of project is hosted on GitHub pages. An interactive web tutorial will also guide you through for using sight detection application.

Demo Link: https://shubhamkj5.github.io/Object-Recognition-Using-CNN/

For source code and contribution send an email on: shubhamkj2405@gmail.com

QR Code for website: