PROJECT REPORT

Shubam Sachdeva
June 12, 2017

I. DEFINITION

Project Overview :

Indian Sign Language(ISL) forms the most widely accepted communication method used by the deaf and mute communities of India. The vocabulary of ISL is formed by various hand gestures, facial expressions and certain body movements. The fact being that India is home to more than 12 million deaf people, a way to lessen the communication gap between the deaf community and ordinary people is essential.

That said, we need a model that could overcome many difficulties posed by the recognition process. But technically speaking, no public dataset, little distortion from signer to signer, size of hands, color of skin, noise in background everything poses a difficulty while training a model. Furthermore, to make an effective model, data that encapsulates all these changes must be formed from scratch.

Although there are several problems those hinder the project, the solution to the problem has been tried by many researchers, both in the field of computer vision and machine learning. For instance, in [1], Washef Ahmed used Dynamic Time Warping methodology to resolve the issue. Purva C Badhe in [2], kept Template Matching as basis of her research. There are a number of researchers present for the topic but most of them use computer vision algorithms at their core.

Problem Statement :

I propose to develop a model that is capable of capturing images in real time from a computer webcam and return us the prediction of the static sign. This won't exactly solve the problem of less communication between the deaf and other people, because most of the signs are dynamic, but at least this can be a starting point. So, I propose a model capable of predicting static signs of ISL in real time.

A static sign is one that involves only spatial features, no temporal features or motion in other words is noticed for these features. The number of static signs in ISL are a lot. I have specifically chosen only 15 signs from the lot.

To address these problem, I propose to use a 2 step model that first extracts the regions of the image those are essential to the recognition part and feed the output binary image to a 2D CNN. Using a 2 step binary image model can have some extreme advantages over feeding a color image directly to the CNN. Doing it this way, doesn't need any gloves or depth images. A simple webcam mounted on a regular desktop PC is enough for the model. To add to it, feeding a binary image to the CNN requires us to use at least 4 times lesser data for training model in turn making the training step a lot faster and computationally inexpensive (by the curse of dimensionality, 2^1=2 for one channel data, 2^3=8 for image with RGB channel.). Unlike traditional models those use a 3 channel RGB image as direct input to the CNN, we preprocess the image. A skin detection decision tree classifier is run on the colored image.Taking this step before CNN allows us to a lot lesser weights and biases units for training and recognition purpose, making the model capable of prediction in real time. For the decision tree, the 3 channel value for RGB is fed to the classifier. In other words, the decision tree classifier will have 3 features here. Value of Red, Green and Blue component of the pixels each ranging from 0-255 are input to the classifier here. The classifier will then return us 1 if the particular pixel is skin and 0 otherwise. This way, we will obtain a binary image. This binary image is fed to CNN for further prediction of the static sign.

Metrics :

Imagine a situation you spell out your name R-I-C-K but the model being less accurate mis-spells the “R” with “D” or a thumbs up sign being misinterpret as thumbs down one. No one would want to use such a model. For accepting a model, it’s very necessary that the model be accurate and precise. The evaluation metrics that will be used in the project is accuracy and F1 score.

One can define accuracy as the number of correct predictions over number of total predictions. Since accuracy is directly proportional to the happiness index of the person using the model, it shall be a good way to measure the performance and correctness of the model.

F1 score in terms of true positives, false positives and false negatives is formulated as:

$$F_\beta = \frac{(1+\beta^2)\cdot \text{true positive}}{(1+\beta^2)\cdot \text{true positive} + \beta^2 \cdot \text{false negative} + \text{false positive}}$$

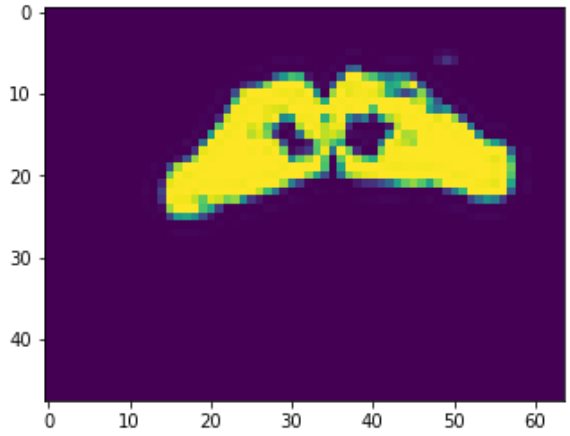
II. ANALYSIS

Data Exploration

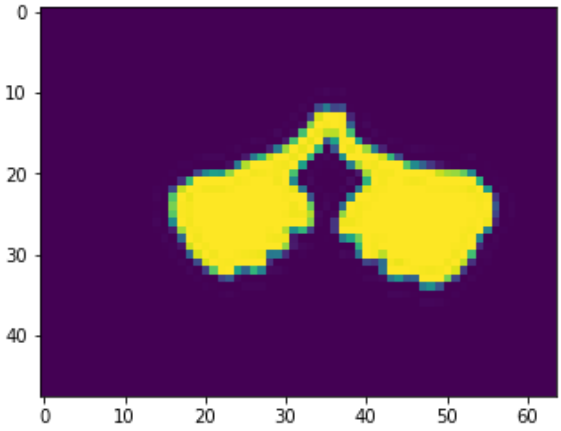
Since there is no publically available dataset, I have made dataset on my own. I and a few friends have captured around 9750 images. It's quite a tiring job but CNN requires a lot of images to train. For this, I have written code in "making data.ipynb". The process of data making is simple. Each signer stands infornt of a camera. The camera is on all the time but takes a picture only once every 2 seconds. By this, we allow enough time to each signer to distort their sign a little while not wasting too much of time on data collection part. These images are of 15 signs in ISL. The signs are a, b, c, d, e, 1, 2, 3, 4, 5, okay, fine, help, study, and promise. We have taken special care that the background in the images do not contain any skin color. The signers themselves vary a little in skin color. It is important to note that since all the signers taken are from India and that the sign language is Indian, it skin detection works fairly well on Indian skin in bright light but fails to impress white people sometimes. A few times, skin detection doesn't capture skin of white people. But, this should not be a problem since the audience for the project are Indians.

Exploratory Visualization

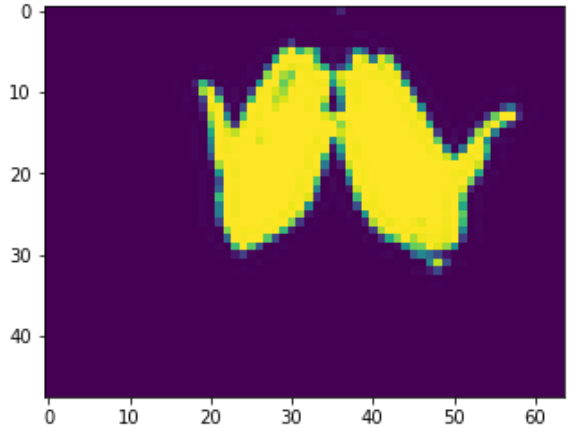
In the "making data.ipynb" file, one can make the data themselve. I have provided all the code for that. Further, in "training CNN.ipynb", you can visualize any data point that you want to visualize. For reference purpose only, I have pasted some images below. Feel free to check any images that you want.



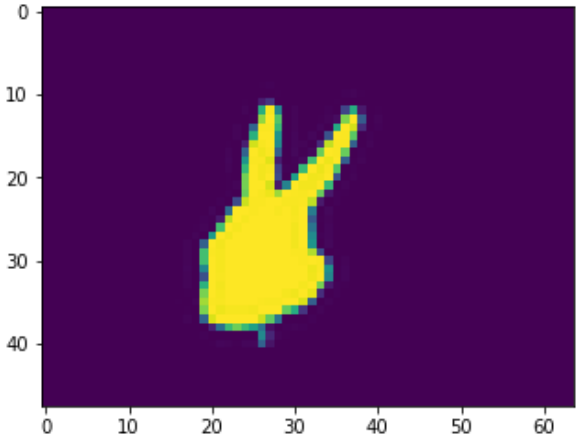
Alphabet “B”



Alphabet “A”

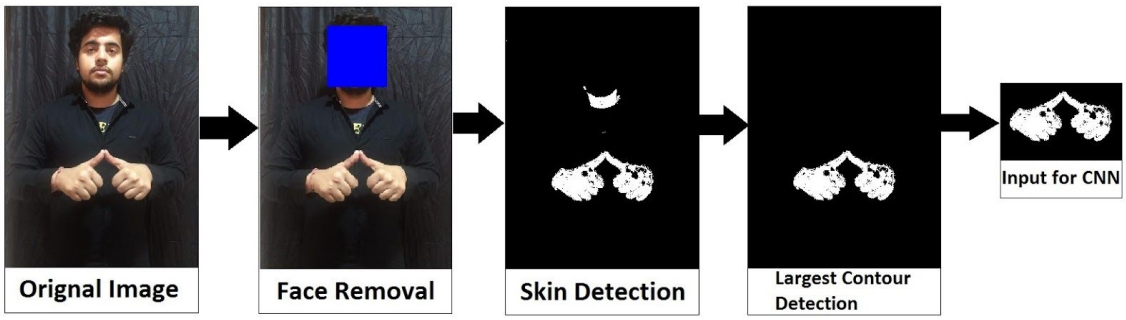


Sign “STUDY”



Digit “2”

As I have shown in “making data.ipynb”, the overall process of data making will look like this :



You can see the process yourself in the code.

Algorithms and Techniques

In the project I have used multiple algorithms and techniques from both computer vision and machine learning. Following are the algorithms used from machine learning :

1. ***Decision Trees*** :

All decision tree algorithms have the same structure. Basically, it's a greedy divide and conquer algorithm.

1. Take the entire data set as input.
2. Search for a split that maximizes the "separation" of the classes. A split is any test that divides the data in two (e.g. if attribute5 < 10).
3. Apply the split to the input data (the "divide" step) into two parts.
4. Re-apply steps 1 and 2 to each side of the split (the recursive "conquer" step).
5. Stop when you meet some stopping criteria.
6. (Optional) Clean up the tree in case you went too far doing splits (called "pruning")

Where the algorithms differ are:

Finding a split: Methods here vary from greedy search (e.g. C4.5) to randomly selecting attributes and split points (e.g. Random forests).
Purity measure: Measures here include: Information Gain, gain ratio, Gini coefficient, minimum description length and Chi-squared values.
Stopping criteria: Methods here vary from a minimum size, to a particular confidence in prediction, to certain purity criteria.
Pruning method: Methods here include no pruning, reduced-error pruning, and in ensemble cases such as bagging, out-of-bag error pruning.

2. *Convolutional Neural Networks*

There are four main steps in CNN: convolution, subsampling, activation and full connectedness.

The most popular implementation of the CNN is the LeNet, after Yann LeCun. The 4 key layers of a CNN are Convolution, Subsampling, Activation and Fully Connected.

Step 1: Convolution

The first layers that receive an input signal are called convolution filters. Convolution is a process where the network tries to label the input signal by referring to what it has learned in the past. If the input signal looks like previous cat images it has seen before, the "cat" reference signal will be mixed into, or convolved with, the input signal. The resulting output signal is then passed on to the next layer.

Convolving Wally with a circle filter. The circle filter responds strongly to the eyes.

Convolution has the nice property of being translational invariant. Intuitively, this means that each convolution filter represents a feature of interest (e.g whiskers, fur), and the CNN algorithm learns which features comprise the resulting reference (i.e. cat). The output signal strength is not dependent on where the features are located, but simply whether the features are present. Hence, a cat could be sitting in different positions, and the CNN algorithm would still be able to recognize it.

Step 2: Subsampling

Inputs from the convolution layer can be "smoothened" to reduce the sensitivity of the filters to noise and variations. This smoothing process is called subsampling, and can be achieved by taking averages or taking the maximum over a sample of the signal. Examples of subsampling methods (for image signals) include reducing

the size of the image, or reducing the color contrast across red, green, blue (RGB) channels.

Sub sampling Wally by 10 times. This creates a lower resolution image.

Step 3: Activation

The activation layer controls how the signal flows from one layer to the next, emulating how neurons are fired in our brain. Output signals which are strongly associated with past references would activate more neurons, enabling signals to be propagated more efficiently for identification.

CNN is compatible with a wide variety of complex activation functions to model signal propagation, the most common function being the Rectified Linear Unit (ReLU), which is favored for its faster training speed.

Step 4: Fully Connected

The last layers in the network are fully connected, meaning that neurons of preceding layers are connected to every neuron in subsequent layers. This mimics high level reasoning where all possible pathways from the input to output are considered.

From computer vision, I have used :

1. Haar Cascades

Haar-cascade is an object detection algorithm used to locate faces, pedestrians, objects and facial expressions in an image (Kumar R. & Bindu A.; 2006), and mainly used for face detection. In Haar-cascade, the system is provided with several numbers of positive images (like faces of different persons at different backgrounds) and negative images (images that are not faces but can be anything

else like chair, table, wall, etc.), and the feature selection is done along with the classifier training using Adaboost and Integral images.

In general, three kinds of features are used in which the value of a two rectangular features is the difference sum of the pixels within two rectangular regions. These regions have same shape and size and are horizontally or vertically adjacent. (Viola P. & Jones M.; 2001).

Classification learning process requires a set of positive and negative images for training and a set of features are selected using Ada-boost for training the classifier. To increase the learning performance of the algorithm (which is sometime called as weak learner), the Ada-boost algorithm is used. Ad-boost provides guarantees in several procedures.

2. Contour detection

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition. Inbuilt opencv functions for both contour detection and Haar Cascades is used in the process. Little changes if any are done with these computer vision algorithms because this is a machine learning project not a opencv project.

BENCHMARK MODEL

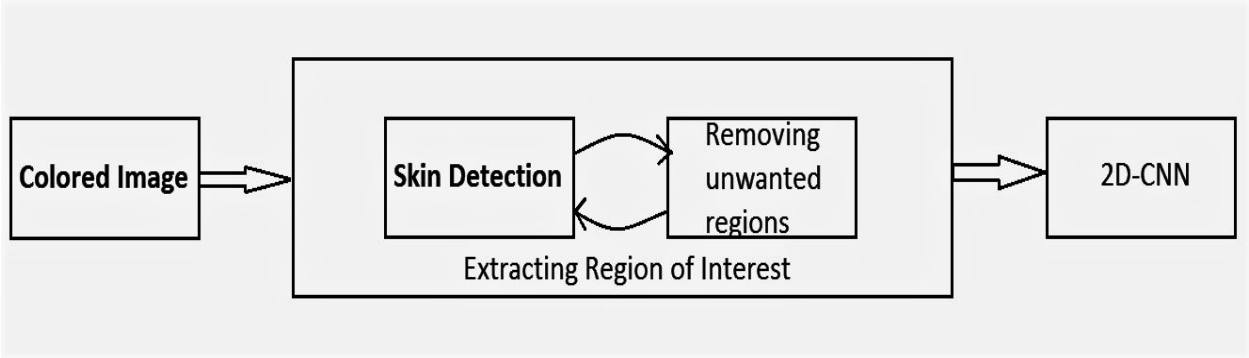
For the benchmark model, I would want to try out decision tree classifier. They are very good classifiers in the sense that we can reason why the particular image returned the particular output, something which isn't possible with a CNN, CNN being a black-box model. Both Decision Tree and CNN have been discussed earlier. A part of the explanation of how these two are used and which one is better of the two is given in the results section. To give an idea, because the data is so

highly processed according to our need, even decision tree classifier gives 99.3% accuracy. But again, this is mainly due to the overall data making process. This accuracy reflects the overall process' effectiveness and not the classifier alone.

III. METHODOLOGY

ARCHITECTURE

Main goal of the project is recognition of signs while maintaining an high accuracy. To achieve this goal, multiple steps layered one on another have been used. Each of these layers coupled together increase the efficiency of overall model reducing noise and data at each step. This design describes the top level architecture of the recognizer.



The initial input to the model, a colored image maybe an image just of hands, a full body image or just a half body image. If the image involves face of the signer, the model shall mess up. To rectify this, a face detection haar cascade is also used.

STEP 1: Image Acquisition

This part has been explained above in the section concerning datasets.

STEP 2: Data Preprocessing (Extracting the region of interest)

The next step is extracting only that portion of the image which determines the sign, the hands. Although CNN is an obvious choice when it comes to classify images but using CNN twice makes the whole process computationally very expensive. This poses difficulties for devices with lower RAM. It is a top priority to rescale the image to a size as small as possible without losing any information in the process. Many haar cascades can be deployed to extract hands but most of them only work well only when the palm is fully closed or fully open. The signs in ISL vary in orientation so these "haar cascades" fail to determine hands in most of the cases. To build a robust algorithm that could recognize hands in any orientation, the following steps are crucial :

2.1 : Skin Detection using Decision Tree Classifier

An entirely different dataset is made for skin detection. All the images for skin detection are taken in bright light. Each image collected has 3 color channels: Red, Green and Blue. The values for these vary from 0 to 255. Portion of the images displaying skin are manually extracted from all these images. Each pixel that belongs to skin portion is labeled 1 and the remaining pixels are labeled 0. A decision tree classifier effectively trains on the dataset of RGB values of the pixels and determines the skin portion. The trained Decision tree is then run on the dataset of images collected for sign recognition. Contours are made for each region of the image predicted as skin. The remaining non skin area is assigned value of 0 for RGB channel, giving it black color. This leaves us with region from hands and face only. Some noise that doesn't belong to skin region is obtained.

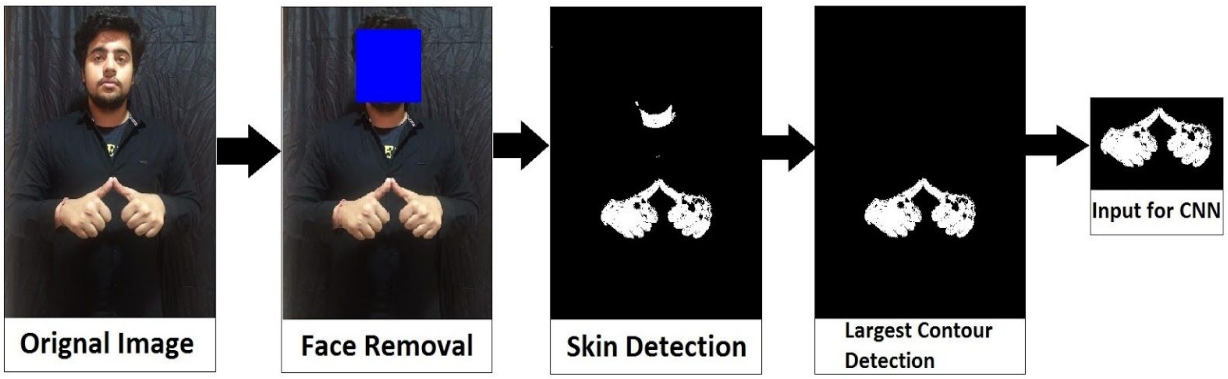
2.2 : Removing unwanted Skin Region in the image

As established earlier, only hands are of use to the model. Some more preprocessing is required to eliminate face from the image. This can be done before skin detection too. Extracting faces from haar cascade is easy and computationally inexpensive. Haar Cascade is run on all the images obtained after skin detection.

The area detected as face is removed from the image. This leaves us with just the hands portion and some random noise in background detected as skin.

2.3 : Removing background noise and rescaling the image

Now that face is removed from the image, only hands along-with some random background noise detected as skin remains. To remove this noise, we bring contours into play again. It's observed that of all the contours in the image, hands contour is always the largest in terms of area. This largest area contour is kept as such while the rest of the contours are again assigned a value of 0 for each of the 3 RGB channels. Thus, what we have now is only hands portion. A rectangle is drawn around the contour. Everything except the area in the rectangle is thrown away. The area of each image is different now. We resize these images down to 64 pixels * 48 pixels.

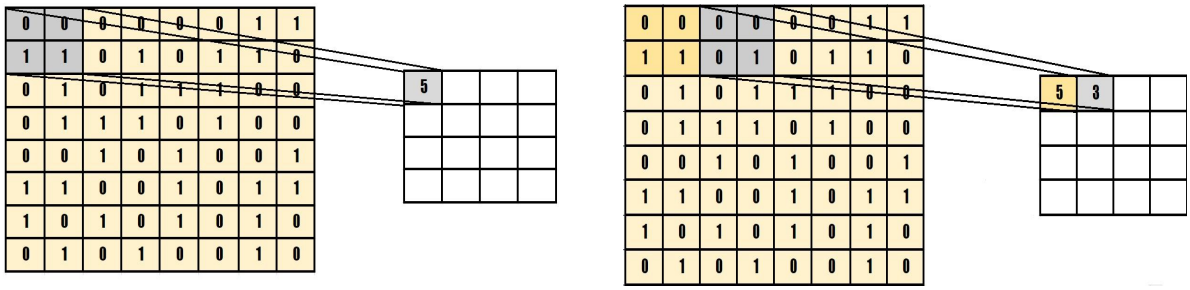


STEP 3: IMPLEMENTATION (2D Convolutional Neural Network)

The third and final step in the process of recognition is 2D Convolutional Neural Networks. A traditional CNN consists of a number of convolutional and subsampling layers optionally followed by fully connected layers. For colored images, a CNN takes in an input m*n*r image where m is the width, n the height and r the color channel of the image. Typical images have a RGB channel. We have an advantage here. Since our input image is binary, the input to our 2D CNN is m*n. This reduces the curse of dimensionality. We need three times less data for

training a CNN than we would for a RGB channel image.This reduces the computational cost and we also need a lot fewer parameters to train our model.

A figure representing the functionality of CNN is given below. A window called local receptive field slides over the input image(binary in this case). Each pixel has some weight associated with it. These weights are multiplied with the pixel value and the result is obtained by max pooling of the window pixels. At next time step, the window slides a predetermined number of pixels and the process of pooling continues.



Above figure represents 2D-CNN layer at time step t (left) and t+1 (right)

STEP 4: Refinement

This step is fairly important for most of the classifiers because tuning a classifier can increase the accuracy. But for this project, refinement is not required for CNN because it is already performing very well. All the refinement that was required was in data making. That refinement has already been discussed in the data making part.

IV. RESULTS

Model Evaluation and Validation:

The model performs well. It is tough to beat the decision tree benchmark accuracy of 99.3% but our CNN model outperforms the former, returning an accuracy of 99.7%. For the purpose of robustness and generalizing the model, as stated in the data making process, 15 signers signed in different backgrounds. There is a downfall though. The model fails to perform when the signer has parts of skin those are bigger than hands in the image. For eg, if the person isn't wearing any shirt. In this case, the largest area of skin will be the stomach and chest area rather than the hands area. Face, though is removed prior to the skin detection and doesn't affect the model. The background too, if contains skin color, it can mess up the model. If these two things are kept in mind, the model performs wonderfully. One can see the processes taking place in real time in "making data.ipynb" file. An example is also done for illustration of the same.

The confusion metrics from "training CNN.ipynb" sheds lights how our model outperforms the decision tree.

For the purpose of accuracy,
Decision Tree : 99.3%
CNN : 99.7%

About Robustness, the model is fairly robust because the overall image doesn't matter. All that matters is the hands. Since the face and other parts are cut off, nothing matters. Consider 2 images of same hand shape. One features a girl, little darkish and other a boy, a big fat white-toned boy. Their face may differ largely, body structure may differ largely and everything. Consider yet another image where only hands are captured. For all the images, it boils down to same thing, a contour of hands. This eliminates everything else. So, robustness of model is guaranteed.

And since the model performs well for 15 people of different ages,shape,gender and skin tone, it's safe to say that this model generalizes well and so yes, the model's predictions can be trusted

The model in its final shape with all the preprocessing seems to do a fairly good job, especially when it comes it precision of the model. And since the model is tested on real people and that too on so many images, the model is highly generalized. Therefore, the model is reasonable to use.

To add to the reason of expectations, the model has exceeded my expectations. For CNNs that I have personally come across, accuracy of 98% for the CIFAR dataset was my previous best. But the point here is that the data in CIFAR and this dataset differs a lot when it comes to preprocessing. I believe that the preprocessing steps are what makes the model more accurate, fast and reliable.

Justification

As the results clearly suggest, our model performs better. But to compare the performance based on just these 2 classifiers would not be a good choice. This is because we didn't use raw images in our classifiers. Instead there was a lot of preprocessing involved. Because of this preprocessing, the results are good. If we used raw images, such high accuracy was difficult to achieve, especially for decision tree.

V. CONCLUSION

Free-Form Visualization

For this part, high performing GPU's are required. The model has been saved and attached in the submission file as well. A combination of code from both the files is used. As my computer doesn't support much processing due to RAM issues, in real time, the model was tested elsewhere. On a computer with 16GB intel core i7 6th gen RAM, the model works in real-time and flawlessly. This images below was correctly classified.



Correct :	***A***	***B***	***C***
Prediction :	***A***	***B***	***C***

What's beautiful about these images is that it was clicked from a mobile camera connected to a computer. These images are very high resolution so that skin area can be covered effectively. The lightning conditions were bright favouring the process. The resolution of the original image has to do nothing with the process. Since the image fed to CNN is 64*48, any resolution doesn't make a difference.

Reflection

At the start of the project, I was worried about the real time implementation of the CNN. As I expected, the training purpose took more than 6 hours on my computer whereas the decision tree classifier took just about a minute. The difference is very very large for a slight increase in accuracy. But we must keep in mind that there were only 15 actions involved. Id a model is made to classify all the words in ISL, we would see large difference in the performance. Initially, I thought that CNN would be the toughest part but the overall process is fairly simple. Opposite to my expectations, tensorflow made it very very easy to use CNN. The most difficult part was to make data to feed CNN. This was the most time consuming part. To come up with idea to detect skin, to detect face and then use a binary image only containing only hands took the most time. I tried many ways to detect skin but at last came back to machine learning and used a decision tree classifier and the results prove the effectiveness of these simple algorithms.

I don't know how to deploy the model in real time on the server or to make an android application for the same. If and when I learn any of those, I would surely consider deploying the model and open sourcing my method if its even a bit of a help to the deaf community.

Improvements

There is always a room for improvement. For this project, I improved the model according to the limited knowledge and skills that I have. Maybe a better computer vision algorithm could have been used for skin detection. Maybe there could be remove unwanted parts of skin. Something that could eliminate stomach area or skin color from background and only capture hands could prove to be more effective for the process. Overall, I am quite happy with the results and this nanodegree has given me enough knowledge to develop something that could be useful for the society.