CERTIFICATE

This is to certify that the dissertation/project report (CEN-792) entitles **"Sign Language Recognizer"** , is an authentic work carried out by **Sahil Malik**, **Mohd. Intekhab** and **Sharik Khan**.

The work is submitetd in complete fulfilment of the requirement for the award of the degree of Batchelor of Technology in Computer Engineering under my guidance. The matter emboidied in this project work has not been submitted earlier for the award of any degree to the best of my knowledge and belief.

Place: New Delhi

Date: 22th December, 2018

Dr. Tanvir Ahmad

Professor & Head,

Dept. Computer Engineering

Faculty Of Engineering & Technology

Jamia Millia Islamia

Mr. Danish Rizvi

Assistant Professor,

Dept. Computer Engineering

Faculty Of Engineering & Technology

Jamia Millia Islamia

ACKNOWLEDGMENT

We would like to thank our mentor Mr. Danish Rizvi (Assistant Professor, Dept. of Computer Engineering) for giving us the opportunity to take up this project. We thank him for his invaluable guidance and overwhelming moral support. We are also very grateful to our HOD, Prof. Tanvir Ahmad for his valuable support throughout the project. We also thank our entire faculty for their teaching, guidance and support.

We are also thankful to our friends for their suggestions and support when required. We regret any inadvertent omissions.

Sahil Malik   Mohd. Intekhab   Sharik Khan

Students :
Semester 7, Bachelor of Technology
Department of Computer Engineering
Faculty of Engineering and Technology
Jamia Millia Islamia
New Delhi

Table Of Content:

Abstract

Human Computer Interaction moves forward in the field of sign language interpretation. Sign Language Interpretation system is a good way to help the hearing impaired people to interact with normal people with the help of computer. As compared to other sign languages, ISL interpretation has got less attention by the researcher. Vision based hand gesture recognition system have been discussed as hand plays vital communication mode. Vision based system have challenges over traditional hardware based approach; by efficient use of computer vision and pattern recognition, it is possible to work on such system which will be natural and accepted, in general.

In this project we have used to classify the signs into readable english text/alphabets.

Chapter 1: Introduction

1.1 About The Project

The intent of this project is to provide a digital autonomous platform to the hearing impaired and the english speakers.

1.2 About Sign Language

Sign Languages are a set of languages that use predefined actions and movements to convey a message. These languages are primarily developed to aid deaf and other verbally challenged people. They use a simultaneous and precise combination of movement of hands, orientation of hands, hand shapes etc. We focus on Sign language in this project. It is the primary language of many people who are deaf and is one of several communication options used by people who are deaf or hard-of-hearing. So as to link a bridge between people who can't talk in Sign Language can this described system to communicate with a deaf person.

1.3 Need For Sign Language Recognizition

While sign language is very important to deaf-mute people, to communicate both with normal people and with themselves, is still getting little attention from the normal people. We as the normal people, tend to ignore the importance of sign language, unless there are loved ones who are deaf-mute. One of the solution to communicate with the deaf-mute people is by using the services of sign language interpreter. But the usage of sign language interpreter can be costly. Cheap solution is required so that the deaf-mute and normal people can communicate normally.

1.4 Challenges

One of the major obstacle that we faced was the availability of multiple sign langauges present(eg. ASL, ISL, BSL). Finding a suitable and large enough data set to train our model to obtain an acceptable result was also a hurdle we had to face.

Training the given data set was computationally challenging as well.

1.5 Tools Used

Tensorflow: Tensorflow is a computational framework for building machine learning models. TensorFlow provides a variety of different toolkits that allow you to construct models at your preferred level of abstraction. You can use lower-level APIs to build models by defining a series of mathematical operations.

OpenCV: OpenCV is an open source C++ library for image processing and computer vision, originally developed by Intel and now supported by Willow Garage. Therefore it is not mandatory for your OpenCV applications to be open or free. It is a library of many inbuilt functions mainly aimed at real time image processing.

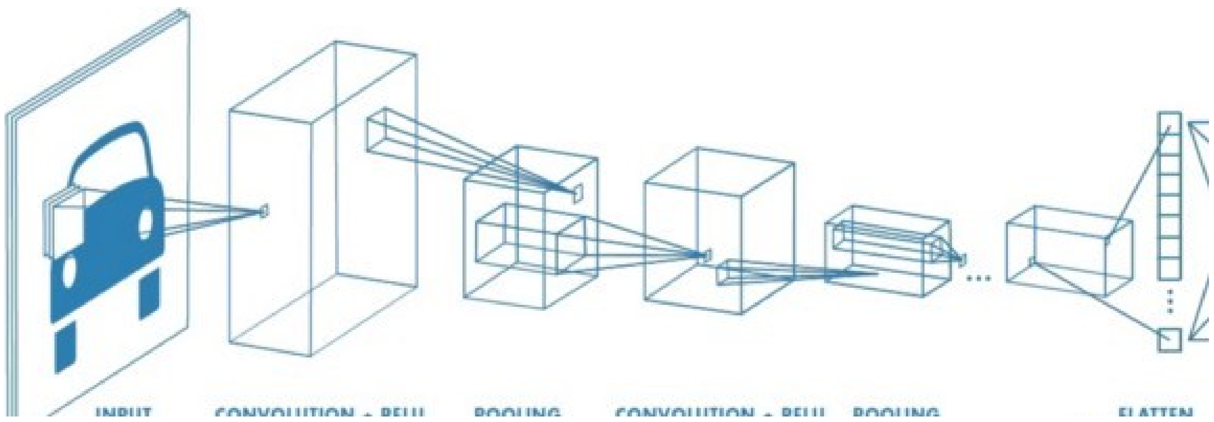
CNN (Inception): Convolutional neural network (ConvNets or CNNs) is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.

CNN image classifications takes an input image, process it and classify it under certain categories

1.6 Convolutional Neural Network

In neural networks, Convolutional neural network (ConvNets or CNNs) is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.

Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernals), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.



There are mainly six layers in any CNN model, namely:

Input Layer: Layer that deals with taking input and supplying the same to the input layer.

ConVolutional Layer: Convolution is the layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernal

Non Linearity(ReLU): **RELU** is just a non linearity which is applied similar to neural networks.

igmoid: takes a real-valued input and squashes it to range between 0 and 1 $\sigma(x) = 1 / (1 + \exp(-x))$

tanh: takes a real-valued input and squashes it to the range [-1, 1] $\tanh(x) = 2\sigma(2x) - 1$

Pooling Layer: Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or downsampling which reduces the dimensionality of each map but retains the important information.

Fully Connected layer: The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like neural network.

1.7. Need for Neural Network:

ANN is nonlinear model that is easy to use and understand compared to statistical methods. ANN is non-paramateric model while most of statistical methods are parametric model that need higher background of statistic. ANN with Back propagation (BP) learning algorithm is widely used in solving various classification and forecasting problems. Even though BP convergence is slow but it is guranteed. However, ANN is black box learning approach, cannot inteprete relationship between input and output and cannot deal with uncertainties. To overcome this several approches have been combined with ANN such as feature selection and etc.

Chapter 2: Literary survey:

A) One other method we found to process the same was by using computer vision,

Below is a short description of the system process represented by the algorithm.

- **Capture the video frames**
- **Blur the image:**The second step is to blur the image in order to make it smooth and to reduce the noise. This is done because the interest is not on the details of the image but the shape of the hand which is to be tracked. In this program Gaussian Blurring was used on the original image.
- **Thresholding:** The third step is to use thresholding for image segmentation, to create binary images from grayscale images. In thresholding a low pass filter is applied by allowing only particular color ranges to be highlighted as white while the other color are suppressed by showing them as black. In this program Otsu's Binarization method was applied.
- **Draw contours:** The fourth step is to draw contours along the boundaries of the black and white image. The region is then enclosed with an envelope that is referred to as the convex hull.
- **Find convex hull and convexity defects:** The fifth step is to find the convex points and convexity defects. The convex points are the tips of the finger while the convexity defects are the deepest points of deviation on the contour.

Tasks like finding contours and finding hull points are computationally expensive so this idea too idea was droped as this method would not have worked on realtime video.

B)Another research approach is a sign language recognition system using a data glove user need to wear glove consist of flex sensor and motion tracker. Data are directly obtained from each sensor depends upon finger flexures and computer analysis sensor data with static data to produce sentences. It's using neural network to improve the performance of the system. The main advantage of this approach less computational time and fast response in real time applications. Its portable device and cost of the device also low. Another approach using a portable Accelerometer (ACC) and Surface Electro Myogram (sEMG) sensors used to measure the hand gesture.

ACC used to capture movement information of hand and Arms. EMG sensor placed, it generates different sign gesture. Sensor output signals are fed to the computer process to recognize the hand gesture and produce speech/text. But none of the above methods provide users with natural interaction.

But due to the fact that this method uses extra hardware, it is highly immpractical; as you need these gloves at all point of time.

C) Convolutional Neural Networks have been extremely successful in image recognition and classification
problems, and have been successfully implemented for human gesture recognition in recent years. In particular,
there has been work done in the realm of sign language recognition using deep CNNs, with input-recognition that is sensitive to more than just pixels of the images. With the use of cameras that sense depth and contour, the process is made much easier via developing characteristic depth and motion profiles for each sign language gesture.

Until recently, however, methods of automatic sign language recognition weren't able to make use of the
depth-sensing technology that is as widely available today. Previous works made use of very basic camera technology to generate datasets of simply images, with no depth or contour information available, just the pixels present. Attempts at using CNNs to handle the task of classifying images of ASL letter gestures have had some success but using a pre-trained GoogLeNet architecture.

Our Sign laguage recognizor is inspired by this papers and Classification method used is CNN.

Chapter 3: Implementation

3.1 Introduction

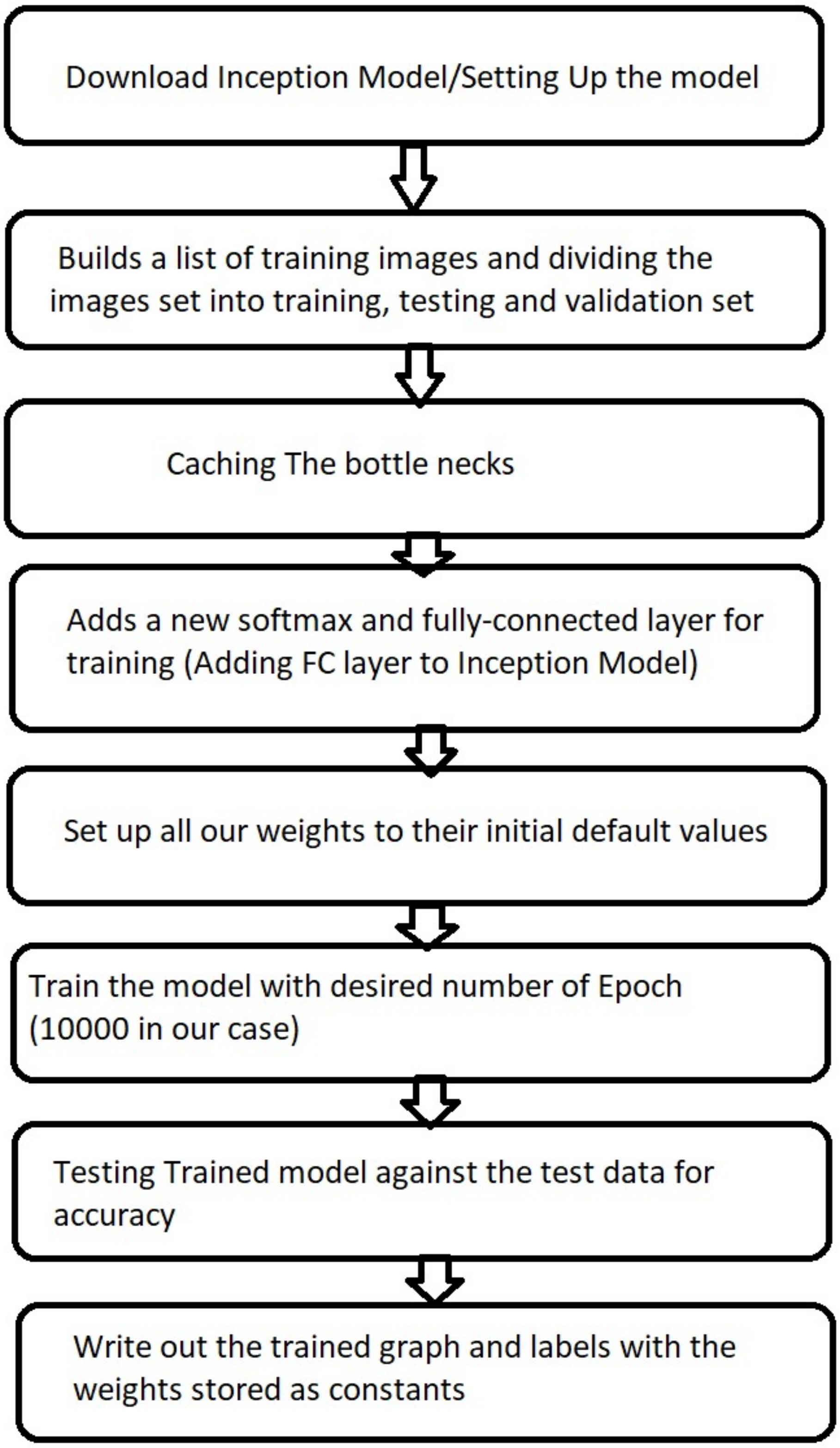
The program applies Transfer Learning to this existing model and re-trains it to classify a new set of images.

This program shows how to take a Inception v3 architecture model trained on ImageNet images, and train a new top layer that can recognize other classes of images. You can replace the image_dir argumenta with any folder containing subfolders of images. The label for each image is taken from the name of the subfolder it's in.

Modern image recognition models have millions of parameters. Training them from scratch requires a lot of labeled training data and a lot of computing power (hundreds of GPU-hours or more). Transfer learning is a technique that shortcuts much of this by taking a piece of a model that has already been trained on a related task and reusing it in a new model. I

Though it's not as good as training the full model, this is surprisingly effective for many applications, works with moderate amounts of training data (thousands, not millions of labeled images), and can be run in as little as thirty minutes on a laptop without a GPU. This tutorial will show you how to run the example script on your own images, and will explain some of the options you have to help control the training process.

3.2 Basic Algorithm


Download Inception Model/Setting Up the model
Builds a list of training images and dividing the images set into training, testing and validation set
Caching The bottle necks
Adds a new softmax and fully-connected layer for training (Adding FC layer to Inception Model)
Set up all our weights to their initial default values
Train the model with desired number of Epoch (10000 in our case)
Testing Trained model against the test data for accuracy
Write out the trained graph and labels with the weights stored as constants

Fig(1)

Transfer Learning

Modern image recognition models have millions of parameters. Training them from scratch requires a lot of labeled training data and a lot of computing power (hundreds of GPU-hours or more). Transfer learning is a technique that shortcuts much of this by taking a piece of a model that has already been trained on a related task and reusing it in a new model.

Bottlenecks

The script can take thirty minutes or more to complete, depending on the speed of your machine. The first phase analyzes all the images on disk and calculates and caches the bottleneck values for each of them. 'Bottleneck' is an informal term we often use for the layer just before the final output layer that actually does the classification. (TensorFlow Hub calls this an "image feature vector".) This penultimate layer has been trained to output a set of values that's good enough for the classifier to use to distinguish between all the classes it's been asked to recognize. That means it has to be a meaningful and compact summary of the images, since it has to contain enough information for the classifier to make a good choice in a very small set of values. The reason our final layer retraining can work on new classes is that it turns out the kind of information needed to distinguish between all the 1,000 classes in ImageNet is often also useful to distinguish between new kinds of objects.

Training

Once the bottlenecks are complete, the actual training of the top layer of the network begins. You'll see a series of step outputs, each one showing training accuracy, validation accuracy, and the cross entropy. The training accuracy shows what percent of the images used in the current training batch were labeled with the correct class. The validation accuracy is the precision on a randomly-selected group of images from a different set. The key difference is that the training accuracy is based on images that the network has been able to learn from so the network can overfit to the noise in the training data. A true measure of the performance of the network is to measure its performance on a data set not contained in the training data -- this is measured by the validation accuracy. If the train accuracy is high but the validation accuracy remains low, that means the network is overfitting and memorizing particular features in the training images that aren't helpful more generally. Cross entropy is a loss function which gives a glimpse into how well the learning process is progressing. The training's objective is to make the loss as small as possible, so you can tell if the learning is working by keeping an eye on whether the loss keeps trending downwards, ignoring the short-term noise.

Using the Retrained Model

The script will write out the new model trained on your categories to /tmp/output_graph.pb, and a text file containing the labels to /tmp/output_labels.txt. The new model contains both the TF-Hub module inlined into it, and the new classificiation layer.

Training on Your Own Categories

If you've managed to get the script working on the sign gesture example images, you can start looking at teaching it to recognize categories you care about instead. In theory all you'll need to do is point it at a set of sub-folders, each named after one of your categories and containing only images from that category. If you do that and pass the root folder of the subdirectories as the argument to --image_dir, the script should train just like it did for the sign gesture.

3.3 Coding (Software Design Document)

- **def create_image_lists(image_dir, testing_percentage, validation_percentage)**

Brief: Builds a list of training images from the file system. Analyzes the sub folders in the image directory, splits them into stable training, testing, and validation sets, and returns a data structure describing the lists of images for each label and their paths.

Args:

image_dir: String path to a folder containing subfolders of images.

testing_percentage: Integer percentage of the images to reserve for tests.

validation_percentage: Integer percentage of images reserved for validation.

Returns:A dictionary containing an entry for each label subfolder, with images split into training, testing, and validation sets within each label

- **def get_image_path(image_lists, label_name, index, image_dir, category)**

Brief: Returns a path to an image for a label at the given index.

Args:

image_lists: Dictionary of training images for each label.

label_name: Label string we want to get an image for.

index: Int offset of the image we want. This will be moduloed by the available number of images for the label, so it can be arbitrarily large.

image_dir: Root folder string of the subfolders containing the training images.

category: Name string of set to pull images from - training, testing, or validation.

- **def get_bottleneck_path(image_lists, label_name, index, bottleneck_dir, category)**

Brief: Returns a path to a bottleneck file for a label at the given index.

Args:image_lists: Dictionary of training images for each label.

label_name: Label string we want to get an image for.

index: Integer offset of the image we want. This will be moduloed by the available number of images for the label, so it can be arbitrarily large.

bottleneck_dir: Folder string holding cached files of bottleneck values.

category: Name string of set to pull images from - training, testing, or validation.

Returns: File system path string to an image that meets the requested parameters.

- **def create_inception_graph()**

Brief: Creates a graph from saved GraphDef file and returns a Graph object.

Returns: Graph holding the trained Inception network, and various tensors we'll be manipulating.

def run_bottleneck_on_image(sess, image_data, image_data_tensor, bottleneck_tensor):

Brief: Runs inference on an image to extract the 'bottleneck' summary layer.

Args:

sess: Current active TensorFlow Session.

image_data: String of raw JPEG data.

image_data_tensor: Input data layer in the graph.

bottleneck_tensor: Layer before the final softmax.

Returns: Numpy array of bottleneck values.

- **def maybe_download_and_extract():**

Brief: Download and extract model tar file. Downloads it from the TensorFlow.org website and unpacks it into a directory.

- **def ensure_dir_exists(dir_name):**

Brief: Makes sure the folder exists on disk.

Args:

dir_name: Path string to the folder we want to create.

- **def write_list_of_floats_to_file(list_of_floats, file_path):**

Brief: Writes a given list of floats to a binary file.

Args:

list_of_floats: List of floats we want to write to a file.

file_path: Path to a file where list of floats will be stored.

- **def read_list_of_floats_from_file(file_path)**

Brief: Reads list of floats from a given file.

Args: **file_path:** Path to a file where list of floats was stored.

Returns: Array of bottleneck values (list of floats).

- **def create_bottleneck_file(bottleneck_path, image_lists, label_name, index, image_dir, category, sess, jpeg_data_tensor, bottleneck_tensor)**

Create a single bottleneck file.

- **def get_or_create_bottleneck(sess, image_lists, label_name, index, image_dir, category, bottleneck_dir, jpeg_data_tensor, bottleneck_tensor):**

Brief: Retrieves or calculates bottleneck values for an image.

If a cached version of the bottleneck data exists on-disk, return that, otherwise calculate the data and save it to disk for future use.

Args:**sess:** The current active TensorFlow Session.

image_lists: Dictionary of training images for each label.

label_name: Label string we want to get an image for.

index: Integer offset of the image we want. This will be modulo-ed by the available number of images for the label, so it can be arbitrarily large.

image_dir: Root folder string of the subfolders containing the training images.

category: Name string of which set to pull images from - training, testing, or validation.

bottleneck_dir: Folder string holding cached files of bottleneck values.

jpeg_data_tensor: The tensor to feed loaded jpeg data into.

bottleneck_tensor: The output tensor for the bottleneck values.

Returns: Numpy array of values produced by the bottleneck layer for the image.

- **def cache_bottlenecks(sess, image_lists, image_dir, bottleneck_dir, jpeg_data_tensor, bottleneck_tensor)**

Brief: Ensures all the training, testing, and validation bottlenecks are cached. Because we're likely to read the same image multiple times it can speed things up a lot if we calculate the bottleneck layer values once for each image during preprocessing, and then just read those cached values repeatedly during training. Here we go through all the images we've found, calculate those values, and save them off.

Args:

sess: The current active TensorFlow Session.

image_lists: Dictionary of training images for each label.

image_dir: Root folder string of the subfolders containing the training images.

bottleneck_dir: Folder string holding cached files of bottleneck values.

jpeg_data_tensor: Input tensor for jpeg data from file.

bottleneck_tensor: The penultimate output layer of the graph.

- **def get_random_cached_bottlenecks(sess, image_lists, how_many, category, bottleneck_dir, image_dir, jpeg_data_tensor, bottleneck_tensor):**

Brief: Retrieves bottleneck values for cached images.

If no distortions are being applied, this function can retrieve the cached bottleneck values directly from disk for images. It picks a random set ofimages from the specified category.

Args:

sess: Current TensorFlow Session.

image_lists: Dictionary of training images for each label.

how_many: If positive, a random sample of this size will be chosen. If negative, all bottlenecks will be retrieved.

category: Name string of which set to pull from - training, testing, or validation.

bottleneck_dir: Folder string holding cached files of bottleneck values.

image_dir: Root folder string of the subfolders containing the training images.

jpeg_data_tensor: The layer to feed jpeg image data into.

bottleneck_tensor: The bottleneck output layer of the CNN graph.

Returns: List of bottleneck arrays, their corresponding ground truths, and the relevant filenames.

- **def get_random_distorted_bottlenecks(sess, image_lists, how_many, category, image_dir, input_jpeg_tensor, distorted_image, resized_input_tensor, bottleneck_tensor)**

Brief: Retrieves bottleneck values for training images, after distortions. If we're training with distortions like crops, scales, or flips, we have to recalculate the full model for every image, and so we can't use cached bottleneck values. Instead we find random images for the requested category, run them through the distortion graph, and then the full graph to get the bottleneck results for each.

Args: **sess:** Current TensorFlow Session.

image_lists: Dictionary of training images for each label.

how_many: The integer number of bottleneck values to return.

category: Name string of which set of images to fetch - training, testing,or validation.

image_dir: Root folder string of the subfolders containing the training images.

input_jpeg_tensor: The input layer we feed the image data

distorted_image: The output node of the distortion graph.

resized_input_tensor: The input node of the recognition graph.

bottleneck_tensor: The bottleneck output layer of the CNN graph.

Returns: List of bottleneck arrays and their corresponding ground truths.

- **def variable_summaries(var):**

Attach a lot of summaries to a Tensor (for TensorBoard visualization)."""

- **def add_final_training_ops(class_count, final_tensor_name, bottleneck_tensor):**

Brief: Adds a new softmax and fully-connected layer for training. We need to retrain the top layer to identify our new classes, so this function adds the right operations to the graph, along with some variables to hold the weights, and then sets up all the gradients for the backward pass. The set up for the softmax and fully-connected layers is based on: https://tensorflow.org/versions/master/tutorials/mnist/beginners/index.html

Args:**class_count:** Integer of how many categories of things we're trying to recognize.

final_tensor_name: Name string for the new final node that produces results.

bottleneck_tensor: The output of the main CNN graph.

Returns: The tensors for the training and cross entropy results, and tensors for the bottleneck input and ground truth input.

- **def add_evaluation_step(result_tensor, ground_truth_tensor)**

Brief: Inserts the operations we need to evaluate the accuracy of our results.

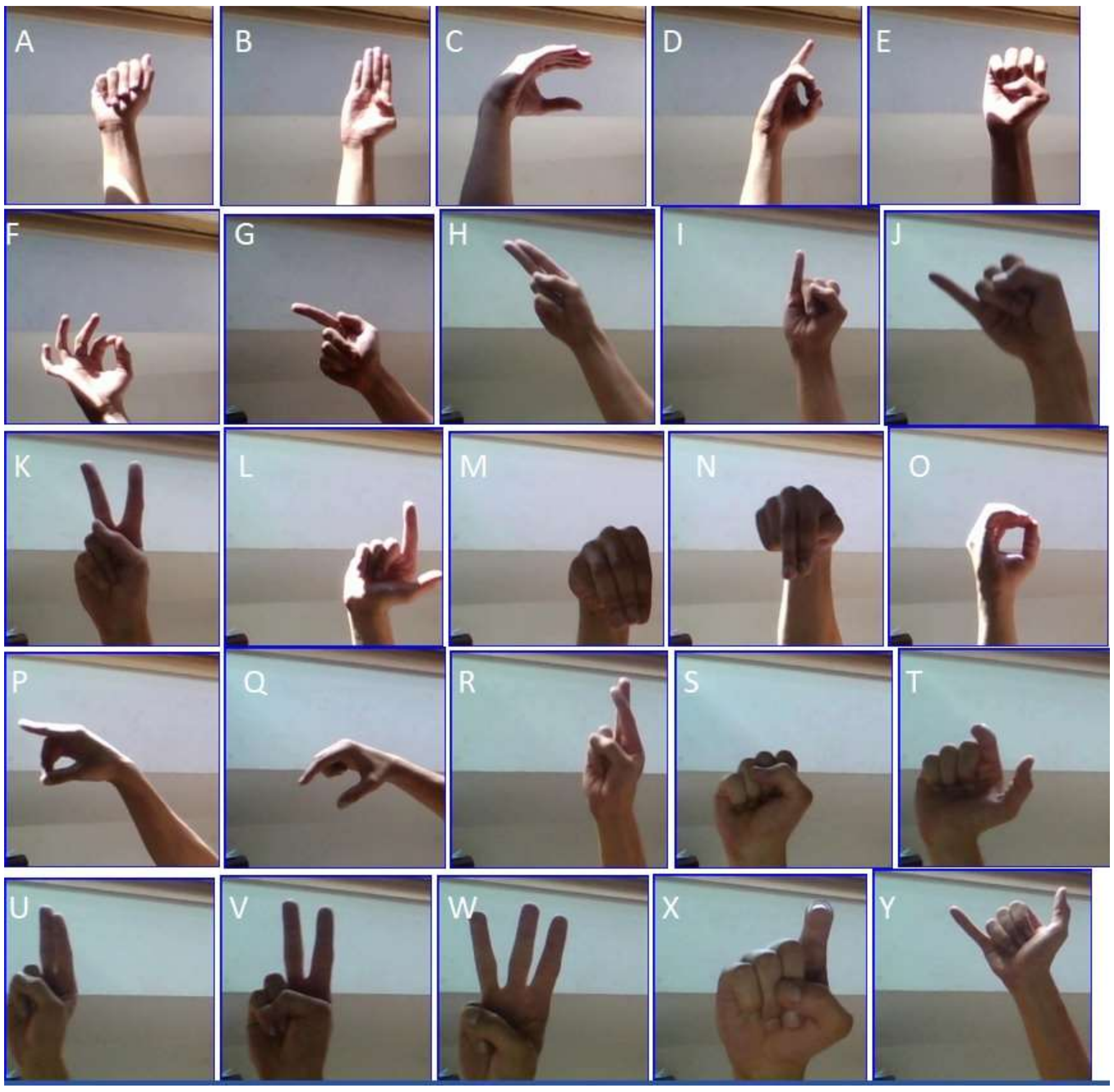
Args:**result_tensor:** The new final node that produces results.

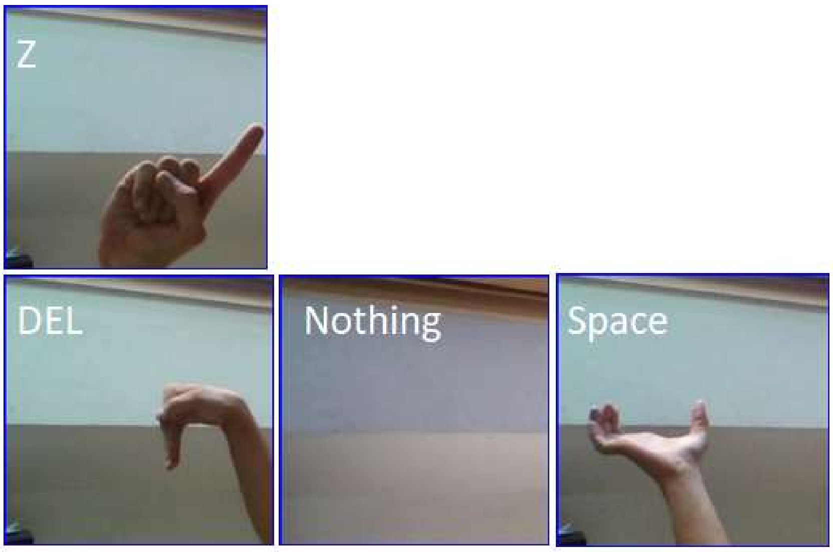
ground_truth_tensor: The node we feed ground truth data into.

Returns: Tuple of (evaluation step, prediction).

Chapter 4: Experimental Results

4.1 DataSet:

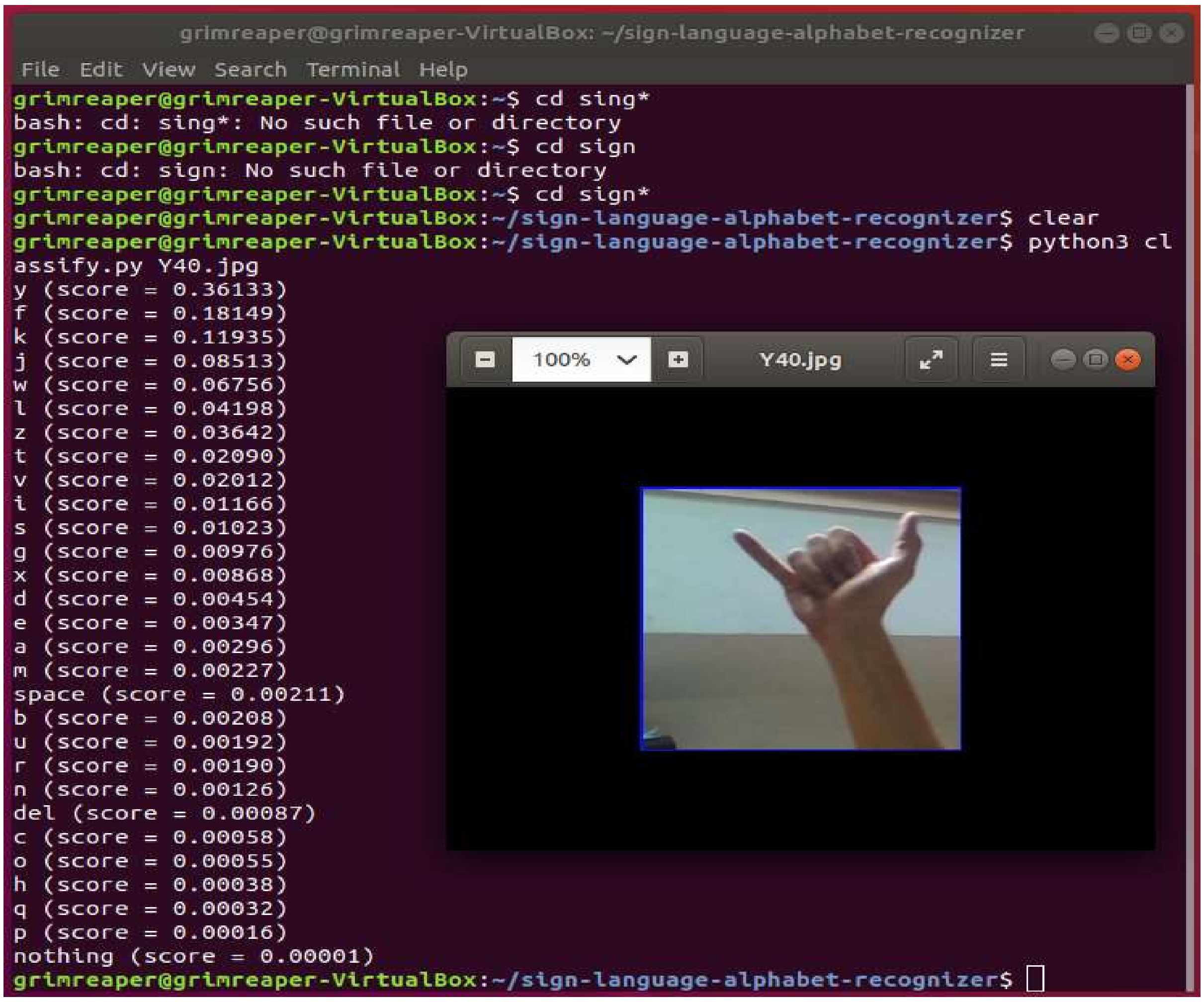




Custom Symbols

4.2 Results for Character 'Y' Recognition

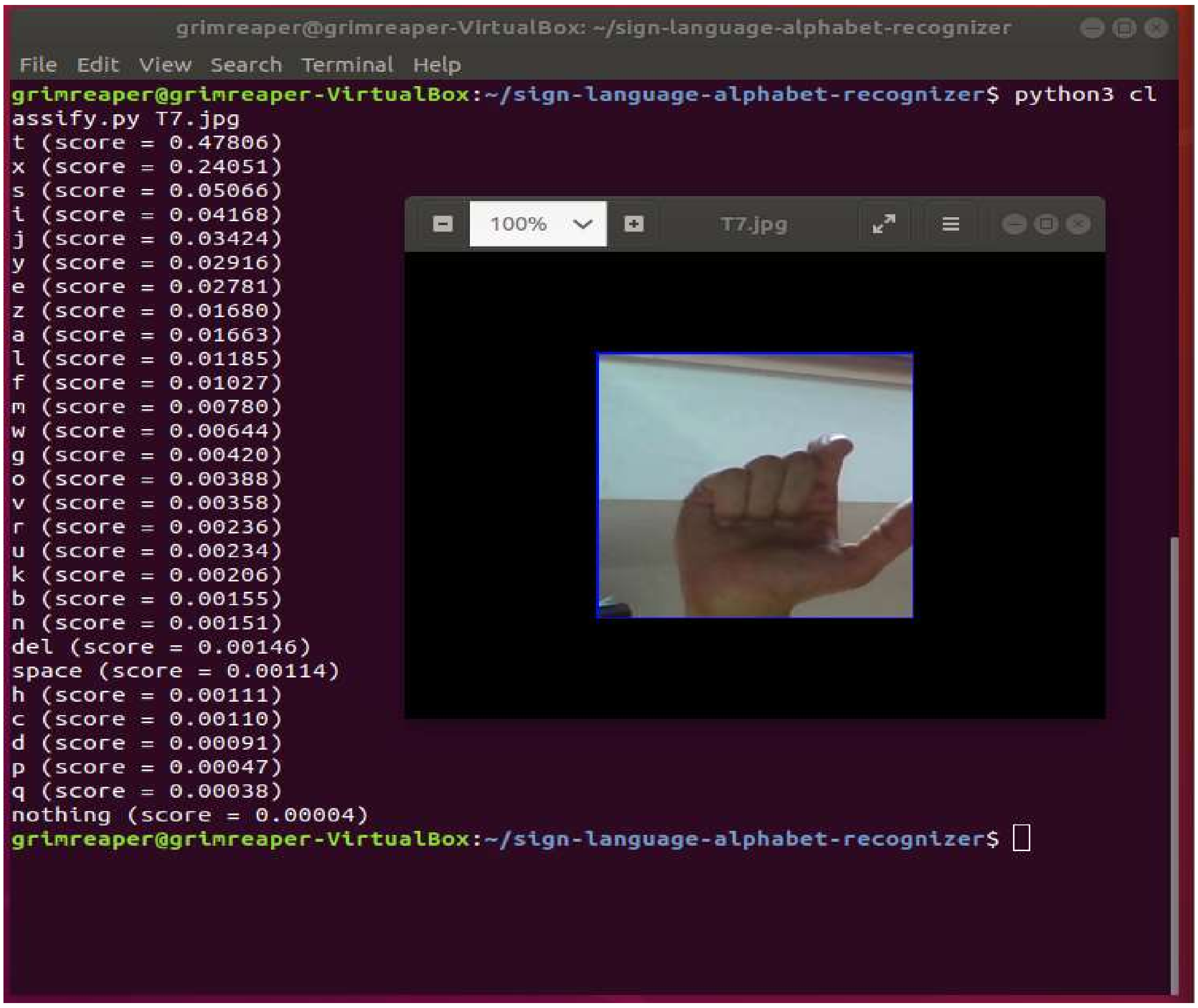
(Using image as an input)


grimreaper@grimreaper-VirtualBox: ~/sign-language-alphabet-recognizer
File Edit View Search Terminal Help
100%
Y40.jpg

```
grimreaper@grimreaper-VirtualBox:~$ cd sing*
bash: cd: sing*: No such file or directory
grimreaper@grimreaper-VirtualBox:~$ cd sign
bash: cd: sign: No such file or directory
grimreaper@grimreaper-VirtualBox:~$ cd sign*
grimreaper@grimreaper-VirtualBox:~/sign-language-alphabet-recognizer$ clear
grimreaper@grimreaper-VirtualBox:~/sign-language-alphabet-recognizer$ python3 cl
assify.py Y40.jpg
y (score = 0.36133)
f (score = 0.18149)
k (score = 0.11935)
j (score = 0.08513)
w (score = 0.06756)
l (score = 0.04198)
z (score = 0.03642)
t (score = 0.02090)
v (score = 0.02012)
i (score = 0.01166)
s (score = 0.01023)
g (score = 0.00976)
x (score = 0.00868)
d (score = 0.00454)
e (score = 0.00347)
a (score = 0.00296)
m (score = 0.00227)
space (score = 0.00211)
b (score = 0.00208)
u (score = 0.00192)
r (score = 0.00190)
n (score = 0.00126)
del (score = 0.00087)
c (score = 0.00058)
o (score = 0.00055)
h (score = 0.00038)
q (score = 0.00032)
p (score = 0.00016)
nothing (score = 0.00001)
grimreaper@grimreaper-VirtualBox:~/sign-language-alphabet-recognizer$
```

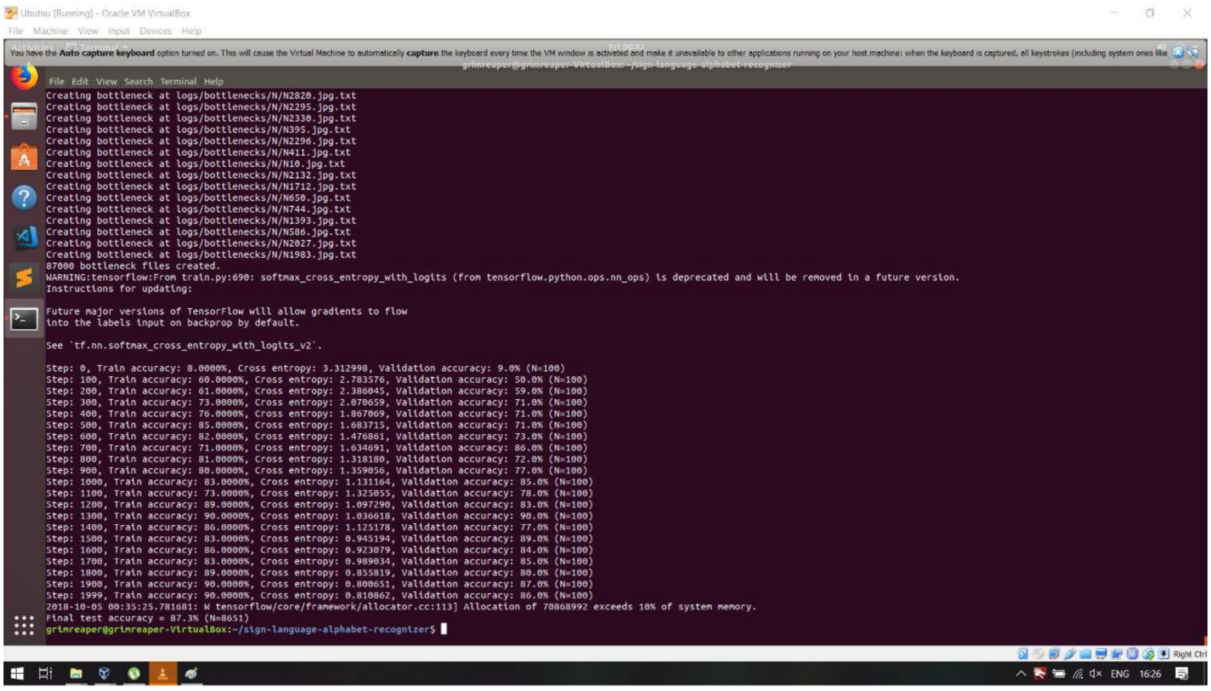
Results for Character 'Y' Recognition

(Using image as an input)



grimreaper@grimreaper-VirtualBox: ~/sign-language-alphabet-recognizer
File Edit View Search Terminal Help
100%
T7.jpg

```
grimreaper@grimreaper-VirtualBox:~/sign-language-alphabet-recognizer$ python3 cl
assify.py T7.jpg
t (score = 0.47806)
x (score = 0.24051)
s (score = 0.05066)
i (score = 0.04168)
j (score = 0.03424)
y (score = 0.02916)
e (score = 0.02781)
z (score = 0.01680)
a (score = 0.01663)
l (score = 0.01185)
f (score = 0.01027)
m (score = 0.00780)
w (score = 0.00644)
g (score = 0.00420)
o (score = 0.00388)
v (score = 0.00358)
r (score = 0.00236)
u (score = 0.00234)
k (score = 0.00206)
b (score = 0.00155)
n (score = 0.00151)
del (score = 0.00146)
space (score = 0.00114)
h (score = 0.00111)
c (score = 0.00110)
d (score = 0.00091)
p (score = 0.00047)
q (score = 0.00038)
nothing (score = 0.00004)
grimreaper@grimreaper-VirtualBox:~/sign-language-alphabet-recognizer$
```


Ubutnu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
File Edit View Search Terminal Help
Right Ctrl
ENG 16:26

You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to automatically **capture** the keyboard every time the VM window is activated and make it unavailable to other applications running on your host machine: when the keyboard is captured, all keystrokes (including system ones like

grimreaper@grimreaper-VirtualBox: ~/sign-language-alphabet-recognizer

```
Creating bottleneck at logs/bottlenecks/N/N2820.jpg.txt
Creating bottleneck at logs/bottlenecks/N/N2295.jpg.txt
Creating bottleneck at logs/bottlenecks/N/N2330.jpg.txt
Creating bottleneck at logs/bottlenecks/N/N395.jpg.txt
Creating bottleneck at logs/bottlenecks/N/N2296.jpg.txt
Creating bottleneck at logs/bottlenecks/N/N411.jpg.txt
Creating bottleneck at logs/bottlenecks/N/N10.jpg.txt
Creating bottleneck at logs/bottlenecks/N/N2132.jpg.txt
Creating bottleneck at logs/bottlenecks/N/N1712.jpg.txt
Creating bottleneck at logs/bottlenecks/N/N650.jpg.txt
Creating bottleneck at logs/bottlenecks/N/N744.jpg.txt
Creating bottleneck at logs/bottlenecks/N/N1393.jpg.txt
Creating bottleneck at logs/bottlenecks/N/N586.jpg.txt
Creating bottleneck at logs/bottlenecks/N/N2027.jpg.txt
Creating bottleneck at logs/bottlenecks/N/N1983.jpg.txt
87000 bottleneck files created.
WARNING:tensorflow:From train.py:690: softmax_cross_entropy_with_logits (from tensorflow.python.ops.nn_ops) is deprecated and will be removed in a future version.
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See `tf.nn.softmax_cross_entropy_with_logits_v2`.

Step: 0, Train accuracy: 8.0000%, Cross entropy: 3.312998, Validation accuracy: 9.0% (N=100)
Step: 100, Train accuracy: 60.0000%, Cross entropy: 2.783576, Validation accuracy: 50.0% (N=100)
Step: 200, Train accuracy: 61.0000%, Cross entropy: 2.386045, Validation accuracy: 59.0% (N=100)
Step: 300, Train accuracy: 73.0000%, Cross entropy: 2.070659, Validation accuracy: 71.0% (N=100)
Step: 400, Train accuracy: 76.0000%, Cross entropy: 1.867069, Validation accuracy: 71.0% (N=100)
Step: 500, Train accuracy: 85.0000%, Cross entropy: 1.683715, Validation accuracy: 71.0% (N=100)
Step: 600, Train accuracy: 82.0000%, Cross entropy: 1.476861, Validation accuracy: 73.0% (N=100)
Step: 700, Train accuracy: 71.0000%, Cross entropy: 1.634691, Validation accuracy: 86.0% (N=100)
Step: 800, Train accuracy: 81.0000%, Cross entropy: 1.318180, Validation accuracy: 72.0% (N=100)
Step: 900, Train accuracy: 80.0000%, Cross entropy: 1.359056, Validation accuracy: 77.0% (N=100)
Step: 1000, Train accuracy: 83.0000%, Cross entropy: 1.131164, Validation accuracy: 85.0% (N=100)
Step: 1100, Train accuracy: 73.0000%, Cross entropy: 1.325055, Validation accuracy: 78.0% (N=100)
Step: 1200, Train accuracy: 89.0000%, Cross entropy: 1.097290, Validation accuracy: 83.0% (N=100)
Step: 1300, Train accuracy: 90.0000%, Cross entropy: 1.036618, Validation accuracy: 90.0% (N=100)
Step: 1400, Train accuracy: 86.0000%, Cross entropy: 1.125178, Validation accuracy: 77.0% (N=100)
Step: 1500, Train accuracy: 83.0000%, Cross entropy: 0.945194, Validation accuracy: 89.0% (N=100)
Step: 1600, Train accuracy: 86.0000%, Cross entropy: 0.923079, Validation accuracy: 84.0% (N=100)
Step: 1700, Train accuracy: 83.0000%, Cross entropy: 0.989034, Validation accuracy: 85.0% (N=100)
Step: 1800, Train accuracy: 89.0000%, Cross entropy: 0.855819, Validation accuracy: 80.0% (N=100)
Step: 1900, Train accuracy: 90.0000%, Cross entropy: 0.800651, Validation accuracy: 87.0% (N=100)
Step: 1999, Train accuracy: 90.0000%, Cross entropy: 0.810862, Validation accuracy: 86.0% (N=100)
2018-10-05 00:35:25.781681: W tensorflow/core/framework/allocator.cc:113] Allocation of 70868992 exceeds 10% of system memory.
Final test accuracy = 87.3% (N=8651)
grimreaper@grimreaper-VirtualBox:~/sign-language-alphabet-recognizer$
```

Chapter 5: Conclusion and Future Work

5.1 Conclusion

The software is designed in such a way that future modification can be done easily. The following conclusions can be deduced from the development of the project:

- It provides an abstract platform between the user.
- CNN has improved the efficiency as discussed before.
- This application will avoid the manual work.

5.2 Future Works

- Incorporation of other sign languages.
- A mobile application that can do the same.
- And this work can be extended to the applications of Controlling of robot, video gaming etc.
- As we know when no image is supplied or an illegal image is provide then too the CNN would classify the false image into a class. This is the major drwback in using CNN. Reducing false positive would also be a priority work.

Reference:

[1] Pratibha Pandey, Vinay Jain, “Hand Gesture Recognition for Sign Language Recognition: A Review”, International Journal of Science, Engineering and Technology Research (IJSETR), Volume 4, Issue 3, March 2015 .

[2] Neelam K. Gilorkar, Manisha M. Ingle, “Real Time Detection And Recognition Of Indian And American Sign Language Using Sift”, International Journal of Electronics and Communication Engineering & Technology (IJECET), Volume 5, Issue 5, pp. 11-18 , May 2014

[3] https://tensorflow.org/versions/master/tutorials/mnist/beginners/index.html

[4] Wario, Ruth & Nyaga, Casam. (2018). Sign Language Gesture Recognition through Computer Vision.

[5] Vivek Bheda & Dianna Radpour Using Deep Convolutional Networks for Gesture Recognition in American Sign Language (arXiv:1710.06836)

[6] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens & Zbigniew Wojna - Rethinking the Inception Architecture for Computer Vision (arXiv:1512.00567)