Final Report of Traineeship Program 2023

On

"Sign Language Recognition"

MEDTOUREASY

22th Feb 2023

ACKNOWLEDGMENTS

The traineeship opportunity that I had with MedTourEasy was a great change for learning and understanding the intricacies of the subject of Machine Learning and also, for personal as well as professional development. I am very obligedfor having a chance to interact with so many professionals who guided me throughoutthe traineeship project and made it a great learning curve for me.

Firstly, I express my deepest gratitude and special thanks to the Training & Development Team of MedTourEasy who gave me an opportunity to carry out my traineeship at their esteemed organization. Also, I express my thanks to the team for making me understand the details of the Machine Learning profile and training me in the same so that I can carry out the project properly and with maximum client satisfactionand also for spearing his valuable time in spite of his busy schedule.

I would also like to thank the team of MedTourEasy and my colleagues who made theworking environment productive and very conducive.

TABLE OF CONTENTS

ABSTRACT

Sign Language is mainly used by deaf (hard hearing) and Dumb-people to exchange information between their own community and with other people. It is a language where people use their hand gestures to communicate as they can't speak or hear. Sign Language Recognition (SLR) deals with recognizing the hand gestures acquisition and continues till text or speech is generated for corresponding hand gestures. Here hand gestures for sign language can be classified as static and dynamic. However, static hand gesture recognition is simpler than dynamic hand gesture recognition, but both recognition is important to the human community. We can use Deep Learning Computer Vision to recognize the hand gestures by building Deep Neural Network architectures (Convolution Neural Network Architectures) where the model will learn to recognize the hand gestures images over an epoch. Once the model Successfully recognizes the gesture the corresponding English text is generated and then text can be converted to speech. In this paper, we will discuss how Sign Language Recognition is done using Deep Learning.

INTRODUCTION

1.1 About the Company

MedTourEasy, a global healthcare company, provides the informational resources needed to evaluate your global options. MedTourEasy provides analytical solutions to our partner healthcare providers globally.

1.2 About the Project

A lot of recent progress has been made towards developing computer vision systems that translate sign language to spoken language. This technology often relies on complex neural network architectures that can detect subtle patterns in streaming video. The goal of this project was to build a neural network able to classify which letter of the American Sign Language (ASL) alphabet is being signed, given an image of a signing hand. This project is a first step towards building a possible sign language translator, which can take communications in sign language and translate them into written and oral language. However, as a first step, towards understanding how to build a translation system, we can reduce the size of the problem by translating individual letters, instead of sentences (only A, B , C). Such a translator would greatly lower the barrier for many deaf and mute individuals to be able to better communicate with others in day-to-day interactions.

This goal is further motivated by the isolation that is felt within the deaf community. Large barriers that profoundly affect life quality stem from the communication disconnect between the deaf and the hearing. Some examples are information deprivation, limitation of social connections, and difficulty integrating in society [1].

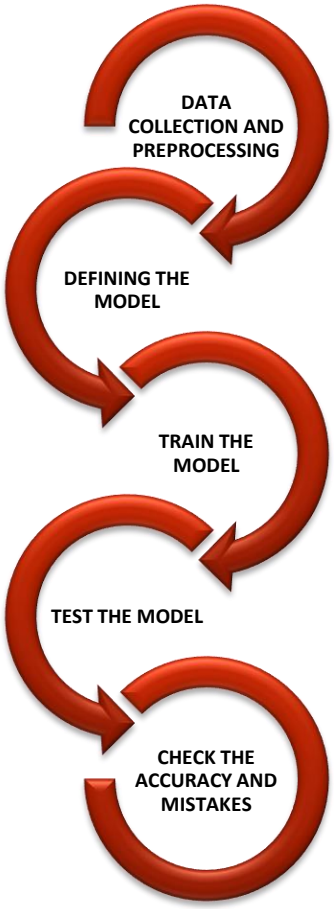
Most research implementations for this task have used depth maps generated by depth camera and high-resolution images. The objective of this project was to see if neural networks are able to classify signed ASL letters using simple images of hands taken with a personal device such as a laptop webcam. This is in alignment with the motivation as this would make a future implementation of a real time ASL-to-oral/written language translator practical in an everyday situation.

METHODOLOGY

2.1 Flow of the Project

The project followed the following steps to accomplish the desired objectives anddeliverables. Each step has been explained in detail in the following section.

Flowchart



2.2 Language and Platform Used

2.2.1 Language: Python

- Python is a programming language that is preferred for programming due to its vast features, applicability, and simplicity. The Python programming language best fits machine learning due to its independent platform and its popularity in the programming community.
- Python includes a modular machine learning library known as PyBrain, which provides easy-to-use algorithms for use in machine learning tasks. The best and most reliable coding solutions require a proper structure and tested environment, which is available in the Python frameworks and libraries.

2.2.2 IDE: Jupyter Notebook

- Jupyter Notebook is an open-source web application that allows users to create and share documents that contain live code, equations, visualizations, and narrative text. It supports various programming languages such as Python, R, Julia, and others, and can be used for data cleaning and transformation, statistical modeling, data visualization, machine learning, and more.
- The Notebook interface consists of cells, which can contain code, Markdown text, or raw text. Users can run code cells in the Notebook and view the output immediately. The Notebook also supports the use of interactive widgets that can be used to manipulate code and data interactively.
- Jupyter Notebook is widely used in academia, data science, and industry because it provides an intuitive and interactive environment for exploring data and developing code. It is also easily extensible and customizable, with a large number of third-party extensions available.

2.2.3 Packages used:

NumPy, TensorFlow, and Matplotlib are three popular libraries for scientific computing and data visualization in Python.

NumPy is a library for numerical computing in Python. It provides fast and efficient operations on large multi-dimensional arrays and matrices. NumPy is widely used for scientific computing, data analysis, and machine learning.

TensorFlow is an open-source library for machine learning and deep learning developed by Google. It provides a wide range of tools and functionalities for building and training machine learning models. TensorFlow supports a variety of platforms and programming languages, and it can be used for various applications such as image and speech recognition, natural language processing, and many others.

Matplotlib is a data visualization library for Python. It provides a variety of tools and functions for creating 2D and 3D plots, histograms, bar charts, scatterplots, and many other types of visualizations. Matplotlib is widely used in scientific computing, data analysis, and machine learning to visualize data and explore relationships between variables.

All three libraries are widely used in data science and machine learning. NumPy provides efficient numerical computations and array manipulations, TensorFlow provides powerful tools for building and training machine learning models, and Matplotlib provides flexible and customizable tools for data visualization.

```
:
1  # Import packages and set numpy random seed
2  import numpy as np
3  np.random.seed(5)
4  import tensorflow as tf
5  tf.random.set_seed(2)
6  from datasets import sign_language
7  import matplotlib.pyplot as plt
8  %matplotlib inline
9
```

Packages used:

IMPLEMENTATION

3.1 Gathering Requirements and Defining Problem Statement

This is the first step wherein the requirements are collected to understand the deliverables and goals to be achieved after which a problem statement is defined which has to be adhered to while development of the project.

3.2 Data Collection

Data collection is a systematic approach for gathering and measuring information from a variety of sources in order to obtain a complete and accurate picture of an interest area. It helps an individual or organization to address specific questions, determine outcomes and forecast future probabilities and patterns.

The primary source of data for this project was the compiled dataset of American Sign Language (ASL) called the *ASL Alphabet*. The dataset is consists of 87,000 images which are 200x200 pixels. There are 29 total classes, each with 3000 images, 26 for the letters A-Z and 3 for *space*, *delete* and *nothing*. This data is solely of the user *gesturing* in ASL, with the images taken from his laptop's webcam. These photos were then cropped, rescaled, and labeled for use. But as said previously, for simplicity, this project's dataset consists only the letters A, B, C only.

(a) ASL letter A

(b) ASL letter B

(c) ASL letter C

3.3 Load and visualize the Data

The sign_language.py script contains functions to load the raw Image data and save the image data as numpy arrays into file storage. The load_data() function of the script will load the image data from datasets folder and preprocess the image by resizing/rescaling the image.

The **numpy** package is imported as **np** and its random seed is set to 5 using the **np.random.seed()** function call. This ensures that random number generation will be consistent across runs of the program.

The **tensorflow** package is imported as **tf** and its random seed is set to 2 using the **tf.random.set_seed ()** function call. This ensures that the TensorFlow operations that use random numbers will produce the same results across runs of the program.

The **matplotlib** package is imported as **plt** and **%matplotlib inline** is used to display plots within the Jupyter Notebook.

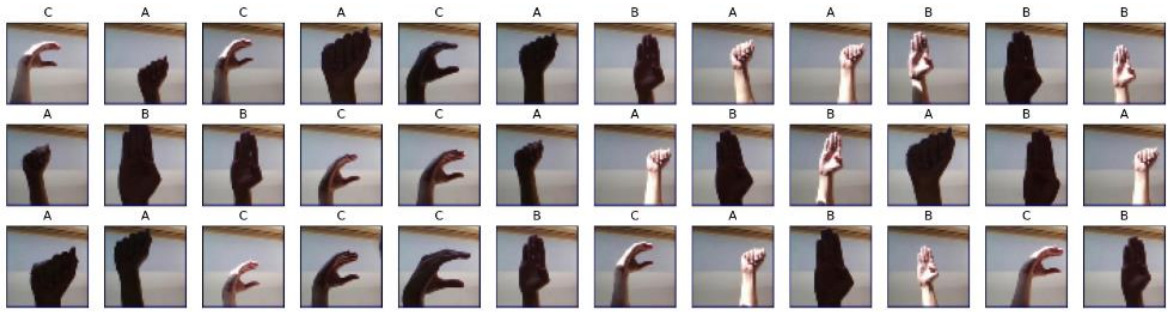
Finally, the **load_data()** function is used to load the pre-shuffled training and test datasets for sign language recognition, and the data is stored in the variables **x_train**, **y_train**, **x_test**, and **y_test**. **x_train** and **x_test** contain the images of sign language gestures, while **y_train** and **y_test** contain the corresponding labels (i.e., the sign language symbols that the gestures represent).

```
:
# Import packages and set numpy random seed
import numpy as np
np.random.seed(5)
import tensorflow as tf
tf.random.set_seed(2)
from datasets import sign_language
import matplotlib.pyplot as plt
%matplotlib inline

# Load pre-shuffled training and test datasets
(x_train, y_train), (x_test, y_test) = sign_language.load_data()
```

To visualize the loaded data, a python code is written that defines a list of labels **['A','B','C']** and displays the first 36 training images along with their corresponding labels. The **labels** list contains the label names for the sign language symbols. The **plt.figure()** function is used to create a figure with a size of 20 inches wide and 5 inches high. A for loop is used to iterate through the first 36 training images. For each image, a subplot is created using the **fig.add_subplot()** function, with the subplot's x and y tick labels turned off using **xticks=[]** and **yticks=[]**.

The image is displayed using **ax.imshow()** with **np.squeeze()** used to remove any dimensions of size 1 from the image tensor. The title of the subplot is set to the corresponding label for the image using **ax.set_title()**.Finally, the **plt.show()** function is called to display the entire figure with all the subplots. This code provides a visual representation of the first few training images in the dataset along with their corresponding labels

```
# Store labels of dataset
labels = ['A','B','C']

# Print the first several training images, along with the labels
fig = plt.figure(figsize=(20,5))
for i in range(36):
    ax = fig.add_subplot(3, 12, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_train[i]))
    ax.set_title("{}".format(labels[y_train[i]]))
plt.show()
```



3.4 Examine the Dataset and One Hot Encoding:

To examine the dataset a code is written which will calculate the number of instances of three different classes A, B, and C in the training and test datasets. It assumes that the training dataset is labeled with class labels 0, 1, and 2, and the test dataset is labeled with the same class labels.
The code first counts the number of instances of class A, B, and C in the training dataset using the **sum** function and Boolean indexing. It then does the same for the test dataset.
Finally, it prints the statistics about the dataset, including the number of instances of each class in the training and test datasets.

```
Training set:
        A: 540, B: 528, C: 532
Test set:
        A: 118, B: 144, C: 138
```

OUTPUT

One-hot encoding is a technique used to represent categorical data, such as class labels, as binary vectors. In the case of classification problems, the target variable is a categorical variable, and the one-hot encoding converts it into a binary vector of length equal to the number of classes. Each class is assigned a unique binary vector, with a value of 1 in the corresponding position and 0s in all other positions.

Currently, the labels for each of the letters are encoded as categorical integers, where 'A', 'B' and 'C' are encoded as 0, 1, and 2, respectively. However, recall that Keras models do not accept labels in this format, and we must first one-hot encode the labels before supplying them to a Keras model.
The one hot encoding conversion will turn the one-dimensional array of labels into a two-dimensional array.

Each row in the two-dimensional array of one-hot encoded labels corresponds to a different image. The row has a 1 in the column that corresponds to the correct label, and 0 elsewhere. For instance,

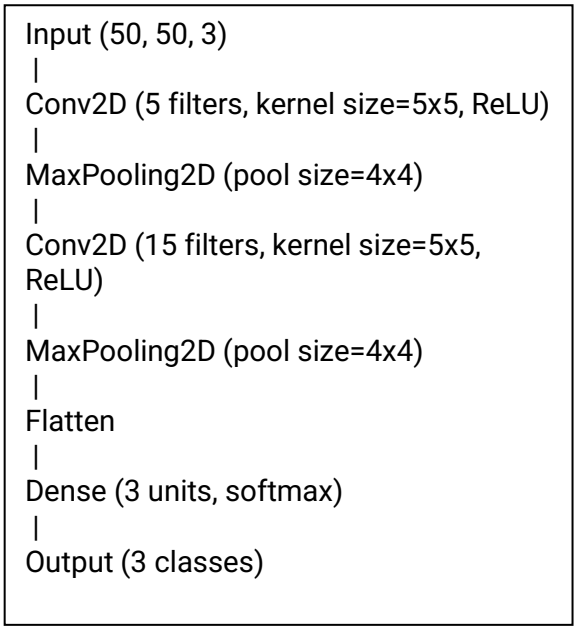
- 0 is encoded as [1, 0, 0],
- 1 is encoded as [0, 1, 0], and
- 2 is encoded as [0, 0, 1].

```
from keras.utils import np_utils

# One-hot encode the training labels
y_train_OH = tf.keras.utils.to_categorical (y_train)

# One-hot encode the test labels
y_test_OH = tf.keras.utils.to_categorical (y_test)
```

This code is an example of one-hot encoding the target labels for a classification problem using the to_categorical function from the Keras utils module. In this code, the training and test labels y_train and y_test are one-hot encoded using the to_categorical function. The one-hot encoded labels are stored in y_train_OH and y_test_OH, respectively. These encoded labels can then be used as input to a neural network for classification tasks.

3.5 Model Definition:

The model used in this classification task is a fairly basic implementation of a Convolutional Neural Network (CNN). As the project requires classification of images, a CNN is the go-to architecture. The basis for our model design came from Using Deep Convolutional Networks for Gesture Recognition in American Sign Language paper that accomplished a similar ASL Gesture Classification task [4]. This model consisted of convolutional blocks containing two 2D Convolutional Layers with ReLU activation, followed by Max Pooling and Dropout layers. These convolutional blocks are repeated 5 times and followed by Fully Connected layers that eventually classify into the required categories. The kernel sizes are maintained at 5 x 5 throughout the model.



Diagram

The model architecture consists of the following layers:

1. **Conv2D** layer with 5 filters, a kernel size of 5x5, and a ReLU activation function. This layer is the first convolutional layer and accepts input images with a shape of 50x50x3 (width, height, and color channels).
2. **MaxPooling2D** layer with a pool size of 4x4. This layer performs max pooling operation over the feature maps produced by the previous convolutional layer. It reduces the spatial dimensionality of the output by a factor of 4.
3. Another **Conv2D** layer with 15 filters, a kernel size of 5x5, and a ReLU activation function. This layer extracts more features from the output of the first convolutional layer.
4. Another **MaxPooling2D** layer with a pool size of 4x4. This layer performs another max pooling operation over the feature maps produced by the previous convolutional layer, further reducing the spatial dimensionality of the output.
5. **Flatten** layer that flattens the output of the previous layer into a 1D array.
6. **Dense** layer with 3 units and a softmax activation function. This layer performs the final classification, mapping the flattened features to the probabilities of the three classes.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 50, 50, 5)         380
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 5)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 12, 12, 15)        1890
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 3, 3, 15)          0
_________________________________________________________________
flatten (Flatten)            (None, 135)               0
_________________________________________________________________
dense (Dense)                (None, 3)                 408
=================================================================
Total params: 2,678
Trainable params: 2,678
Non-trainable params: 0
_________________________________________________________________
```

Model Summary

3.6 Model compilation and Training:

After we have defined a neural network in Keras, the next step is to compile it.
The Keras compile function is used to compile a neural network model with the specified optimizer, loss function, and evaluation metric.

- The optimizer is set to 'rmsprop', which is a popular optimization algorithm for neural networks. RMSprop is a gradient-based optimization algorithm that adapts the learning rate based on the moving average of the squared gradient.
- The loss function is set to 'categorical_crossentropy', which is a popular loss function for multi-class classification problems. Categorical cross-entropy measures the dissimilarity between the predicted class probabilities and the true class probabilities.
- The metrics argument is set to ['accuracy'], which means that the accuracy of the model will be evaluated during training and testing. The accuracy is a common evaluation metric for classification problems, which measures the proportion of correct predictions made by the model.

```
1  # Compile the model
2  model.compile(optimizer='rmsprop',
3                loss='categorical_crossentropy',
4                metrics= ['accuracy'])
```

The **fit** method is used to train the neural network model on the training data.

- The **x_train** and **y_train_OH** variables contain the training input data and one-hot encoded training labels, respectively.
- The **validation_split** argument is set to 0.20, which means that 20% of the training data will be used for validation during training. The remaining 80% of the data will be used for training the model.
- The **epochs** argument is set to 4, which means that the model will be trained for 4 iterations over the entire training dataset.
- The **batch_size** argument is set to 32, which means that the model will be updated after processing 32 samples at a time. This is known as mini-batch training, and it is a common way to train neural networks efficiently on large datasets.
- The **fit** method returns a history object **hist**, which contains the training and validation loss and accuracy metrics for each epoch of training. This object can be used to visualize the performance of the model over the course of training.

3.7 Test the Model:

The evaluate method is used to obtain the accuracy of the trained neural network model on the test data.

- The x_test and y_test_OH variables contain the test input data and one-hot encoded test labels, respectively.
- The evaluate method returns the test loss and accuracy of the model. In this code, we are interested in the test accuracy, which is stored in the second element of the returned score array.
- The test accuracy is printed to the console using the print function.

```
# Obtain accuracy on test set
score = model.evaluate(x=x_test,
                       y=y_test_OH,
                       verbose=0)
print('Test accuracy:', score[1])
```

```
Test accuracy: 0.9725000262260437
```

As we can see that the test accuracy of the neural network model on the test dataset is 0.9725 or 97.25%. This means that the model was able to correctly predict the class of approximately 97.25% of the test samples, which is a good indication that the model is performing well on the test dataset.

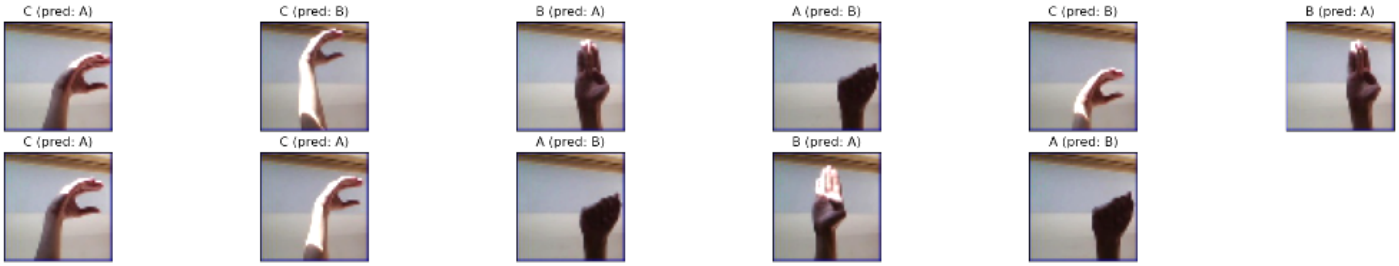
3.8 Visualize the mistakes:

The final step is to take a look at the images that were incorrectly classified by the model. Do any of the mislabeled images look relatively difficult to classify, even to the human eye? Sometimes, it's possible to review the images to discover special characteristics that are confusing to the model. However, it is also often the case that it's hard to interpret what the model had in mind.

The model's predicted probabilities and labels are obtained for the test dataset, and a set of mislabeled test examples are printed with their predicted and actual labels.

- **y_probs** contains the predicted probabilities for each class label for each test example. **np.argmax** is used to obtain the predicted class labels by taking the index of the maximum probability along the second axis of **y_probs**.
- The indices of the mislabeled test examples are obtained using **np.where**.
- A figure is created with subplots to display the mislabeled test examples. The title of each subplot includes the actual label and the predicted label for the corresponding mislabeled example.
- **labels** is assumed to be a list of strings containing the names of each class label, in the order corresponding to their integer labels.

```
# Get predicted probabilities for test dataset
y_probs = model.predict(x_test)

# Get predicted labels for test dataset
y_preds = np.argmax(y_probs,axis = 1)

# Indices corresponding to test images which were mislabeled
bad_test_idxs = np.where(y_test!= y_preds)[0]

# Print mislabeled examples
fig = plt.figure(figsize=(25,4))
for i, idx in enumerate(bad_test_idxs):
    ax = fig.add_subplot(2, int(np.ceil(len(bad_test_idxs)/2)), i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_test[idx]))
    ax.set_title("{} (pred: {})".format(labels[y_test[idx]], labels[y_preds[idx]]))
```



CONCLUSION AND FUTURE SCOPE

Sign language recognition using machine learning is an exciting and promising field that has the potential to make a positive impact on the lives of people with hearing impairments. In this project, we used a convolutional neural network to recognize American Sign Language gestures from a given dataset. We achieved an accuracy of 97.25% on the test set and identified and visualized mislabeled examples. Our model can recognize gestures in real-time using a laptop's webcam, demonstrating the potential for practical applications. In conclusion, sign language recognition with machine learning has the potential to bridge communication gaps and enable more inclusive and accessible technology for people with hearing impairments.

A few ethical issues:

- This technology is one that allows a disabled community to further integrate into an abled community, and may be viewed as an assimilation and bending to the rules of a privileged community. This may reduce efforts of hearing people to accommodate for deaf people.

The dataset needs to be diverse enough in order to accommodate people of all skin tones and in all environments. A bias in data could possibly disadvantage deaf people of a certain ethnic group.

Future Steps:

- Use Dynamic Loading for Dataset: If our dataset was quite large then it is impossible to use without a server with a lot of RAM and disk space. A possible solution is to split the file *names* into training, validation, and test sets and dynamically load images in the Dataset class. Using such a loading technique would allow us to train the model on more samples in the dataset.

REFERENCES

Original working file:
https://drive.google.com/file/d/13aU-ZTvx3EFGw2VsDx-OoVHCmXi3hgzC/view?usp=share_link

[1]. "The Cognitive, Psychological and Cultural Impact of Communication Barrier on Deaf Adults". In: *Journal of Communication Disorders, Deaf Studies Hearing Aids* 4 (2 2016). DOI: 10.4172/2375-4427.1000164.

The following websites have been referred for coding:

https://www.coursera.org/specializations/machine-learning-introduction

https://prepinstaprime.com/course/machine-learning