Detection of Sign Language Through Gesture Recognition

A Project Report

Submitted by,

Ch. Aaron John - 18BCE0204
R Ravi Maithrey - 18BCE0459
Ippili Akarsh - 18BCE2523

CSE3013
Artificial Intelligence

Under the guidance of,

Dr. Anto S.
Associate Professor, SCOPE VIT, Vellore

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
June, 2021

Contents

Abstract

Deaf people use sign languages to communicate with other people in the community. Although the sign language is known to hearing-impaired people due to its widespread use among them, it is not known much by other people. In this article, we have developed a real-time sign language recognition system for people who do not know sign language to communicate easily with hearing-impaired people. The sign language used in this paper is American sign language. In this study, the convolutional neural network was trained by using dataset from IEEE Dataport, and 100% test accuracy was obtained. Following this another neural network based on the MobileNet Architecture was also trained to try for a comparable performance with the advantages given by MobileNet.

Keywords: Sign Language Detection, Deep Learning, Classification, Convolutional Neural Networks, MobileNet

1 Introduction

Communication is of great importance to us as a species, and personally, we do it by talking and listening to people talk. But for the deaf and mute among us, the only way they are able to communicate is through sign language This sign language is in the form of hand gesture, with each gesture referring to either a letter, a word, or an action. This forms a great barrier between people who can and can't understand the sign language being used. So, there is a need for the sign language to be translated Sign language recognition is a problem that has been addressed in research for years. However, we are still far from finding a complete solution available in our society.

Among the works developed to address this problem, the majority of them have been based on basically two approaches: contact-based systems, such as sensor gloves; or vision-based systems, using only cameras. The latter is way cheaper and the boom of deep learning makes it more appealing.

Vision is a key factor in sign language, and every sign language is intended to be understood by one person located in front of the other, from this perspective, a gesture can be completely observable. Viewing a gesture from another perspective makes it difficult or almost impossible to be understood since every finger position and movement will not be observable.

For this reason, any possible solution to the problem of sign language recognition needs to take into consideration the appearance of different signs and how they can be represented in a static image. It is this static image which can be classified and shown into different classes of the alphabets it represents.

2 Literature Survey

2.1 Deep convolution neural network for image recognition

This paper proposes a new methodology to map images in a medical scenario. They propose a methodology based on efficient Convolution Neural Network (CNN) architecture in order to classify epidemic pathogen with five deep learning phases: (1) Training dataset of provided images (2) CNN Training (3) Testing data preparation (4) CNN generated model on testing data and finally (5) Evaluation of images classified.

Although this document addresses the classification of epidemic pathogen images using a CNN model, the underlying principles apply to the other fields of science and technology, because of its performance and its capability to handle more layers than the previous traditional neural networks.

2.2 Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning

This paper talks about three major techniques that successfully employ CNNs to medical image classification: training the CNN from scratch, using off-the-shelf pre-trained CNN features, and conducting unsupervised CNN pre-training with supervised fine-tuning. Another effective method is transfer learning, i.e., fine-tuning CNN models pre-trained from natural image dataset to medical image tasks. In this paper, we can see how the authors exploit three important, but previously understudied factors of employing deep convolutional neural networks to computer-aided detection problems.

The authors first explore and evaluate different CNN architectures. The studied models contain 5 thousand to 160 million parameters, and vary in numbers of layers. Then they evaluate the influence of dataset scale and spatial image context on performance. Finally, they examine when and why transfer learning from pre-trained ImageNet (via fine-tuning) can be useful.

2.3 Sign language recognition: State of the art

Computer recognition of sign language deals from sign gesture acquisition and continues till text/speech generation. Sign gestures can be classified as static and dynamic. However static gesture recognition is simpler than dynamic gesture recognition but both recognition systems are important to the human community. The sign language recognition steps are described in this survey. The data acquisition, data preprocessing and transformation, feature extraction, classification and results obtained are examined. Some future directions for research in this area also suggested.

2.4 Very Deep Convolutional Networks for Large-Scale Image Recognition

In this work the authors investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Their main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3x3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16-19 weight layers.

The authors also show that these representations generalise well to other datasets, where they achieve state-of-the-art results.

2.5 MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

The authors present a class of efficient models called MobileNets for mobile and embedded vision applications. MobileNets are based on a streamlined architecture that uses depthwise separable convolutions to build light weight deep neural networks. They introduce two simple global hyper-parameters that efficiently trade off between latency and accuracy. These hyper-parameters allow the model builder to choose the right sized model for their application based on the constraints of the problem.

They also present extensive experiments on resource and accuracy tradeoffs and show strong performance compared to other popular models on ImageNet classification. Then the authors demonstrate the effectiveness of MobileNets across a wide range of applications and use cases including object detection, finegrain classification, face attributes and large scale geo-localization.

2.6 Sign Language Recognition Using Convolutional Neural Networks

This paper considers a recognition system using the Microsoft Kinect, convolutional neural networks (CNNs) and GPU acceleration. Instead of constructing complex handcrafted features, CNNs are able to automate the process of feature construction. We are able to recognize 20 Italian gestures with high accuracy. The predictive model is able to generalize on users and surroundings not occurring during training with a cross-validation accuracy of 91.7%.

2.7 You Only Look Once: Unified, Real-Time Object Detection

This paper presents YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, this paper frames object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

Compared to state-of-the-art detection systems, YOLO makes more localization errors but is far less likely to predict false detections where nothing exists. Finally, YOLO learns very general representations of objects. It outperforms all other detection methods, including DPM and R-CNN, by a wide margin when generalizing from natural images to artwork on both the Picasso Dataset and the People-Art Dataset.

2.8 Deep Neural Networks for Object Detection

Deep Neural Networks (DNNs) have recently shown outstanding performance on image classification tasks [14]. In this paper the authors go one step further and address the problem of object detection using DNNs, that is not only classifying but also precisely localizing objects of various classes. They present a simple and yet powerful formulation of object detection as a regression problem to object bounding box masks. They define a multi-scale inference procedure which is able to produce high-resolution object detections at a low cost by a few network applications

2.9 Object Detection with Deep Learning: A Review

In this paper, the authors provide a review on deep learning based object detection frameworks. This review begins with a brief introduction on the history of deep learning and its representative tool, namely Convolutional Neural Network(CNN). Then they focus on typical generic object detection architectures along with some modifications and useful tricks to improve detection performance further.

As distinct specific detection tasks exhibit different characteristics, the authors also briefly survey several specific tasks, including salient object detection,face detection and pedestrian detection. Experimental analyses are also provided to compare various methods and draw some meaningful conclusions. Finally, several promising directions and tasks are provided to serve as guidelines for future work in both object detection and relevant neural network based learning systems

2.10 Problem Definition

The problems to be tackled in this project are to develop a Convolutional Neural Network for the purpose of sign language recognition. Further, this developed model will be compared against the existing MobileNet Architecture to see the comparison in performance.

3 Overview of the Work

3.1 Objectives of the Project

In this project, we aim to come up with a model for hand gesture recognition using Convolutional Neural Networks and measure its performance. Following this, a model based on the MobileNet architecture is also developed and trained. Their performance is compared and shown.

3.2 Hardware and Software Requirements

The code for the project was executed on a google colab notebook with the code being written entirely in python. From the google colab notebook, access to a GPU was obtained for the purposes of training the neural network in an efficient manner.

Parameters	Google Colab Specification
Disk Space	25GB
CPU Model	Intel Xeon
CPU Freq.	2.30 GHz
CPU Cores	2
Available RAM	25GB

4 System Design

The user must be able to capture images of the hand gesture using web camera and the system shall predict and display the name of the captured image. We use the HSV colour algorithm to detect the hand gesture and set the background to black. The images undergo a series of processing steps which include various Computer vision techniques such as the conversion to grayscale, dilation and mask operation.

And the region of interest which, in our case is the hand gesture is segmented. The features extracted are the binary pixels of the images. We make use of Convolutional Neural Network (CNN) for training and to classify the images.

Identification of sign gesture is performed with either of the two methods. First is a glove-based method whereby the signer wears a pair of data gloves during the capture of hand movements. Second is a vision-based method, further classified into static and dynamic recognition. Static deals with the 2D representation of gestures while dynamic is a real time live capture of the gestures.

And despite having an accuracy of over 90%, wearing of gloves are uncomfortable and cannot be utilised in rainy weather. They are not easily carried around since their use require computer as well. In this case, we have decided to go with the static recognition of hand gestures because it increases accuracy as compared to when including dynamic hand gestures

4.1 Convolutional Neural Networks - CNNs

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction.

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

The basic CNN architecture includes Convolution layer, Pooling layer, ReLU layer and a fully connected layer.

Convolution Layer: the real power of deep learning, especially for image recognition, comes from convolutional layers. It is the first and the most important layer. In this layer, a CNN uses different filters to convolve the whole image as well as the intermediate feature maps, generating various feature maps. Feature map consists of a mapping from input layers to hidden layers.

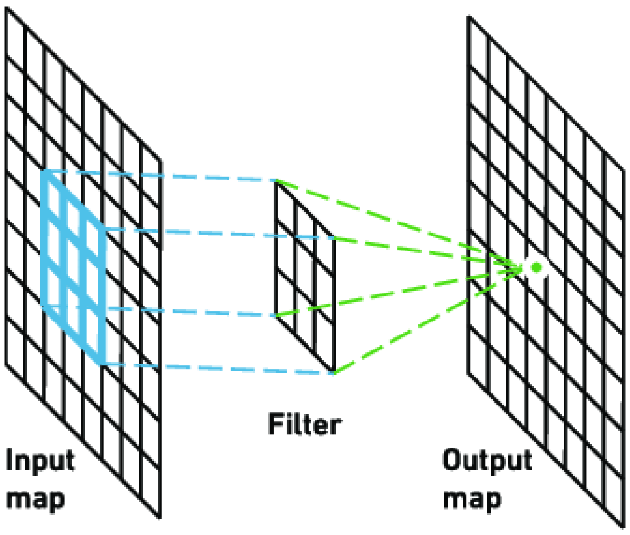


Figure 1: Typical Function of the Convolutional Layer

Pooling Layer: Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.

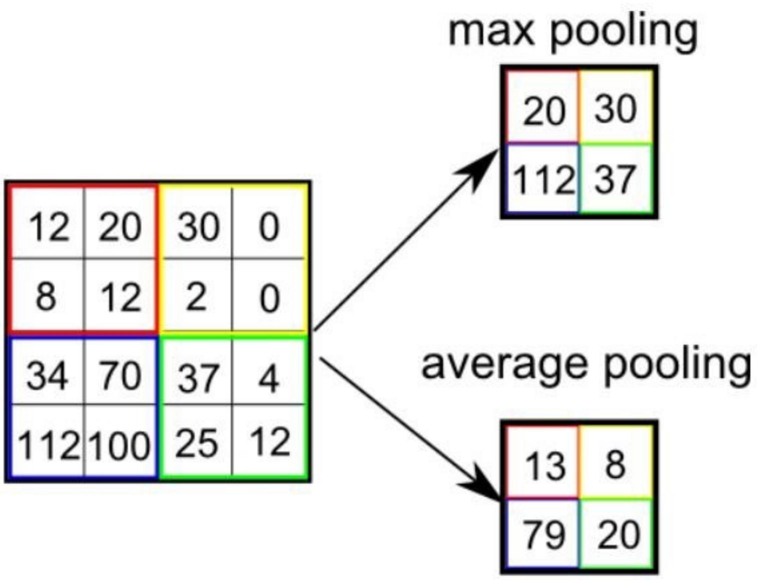


Figure 2: The types of pooling done in pooling layer

ReLU Layer: it is the Rectified Linear Units Layer. This is a layer of neurons that applies the non-saturating nonlinearity function or loss function:

$$f(x) = max(0, x)$$

It yields the nonlinear properties of the decision function and the overall network without affecting the receptive fields of the convolution layers. Pooling layer: its task consists to simplify or reduce the spatial dimensions of the information derived from the feature maps.

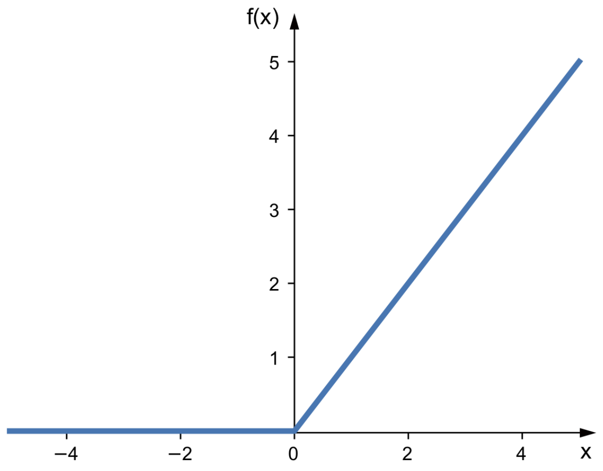


Figure 3: Typical Function of the Convolutional Layer

Fully Connected Layer:Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space. It helps us in gaining an output that is more efficient than without using this fully connected layer.

With the addition of this layer as well, the final CNN structure is as follows.

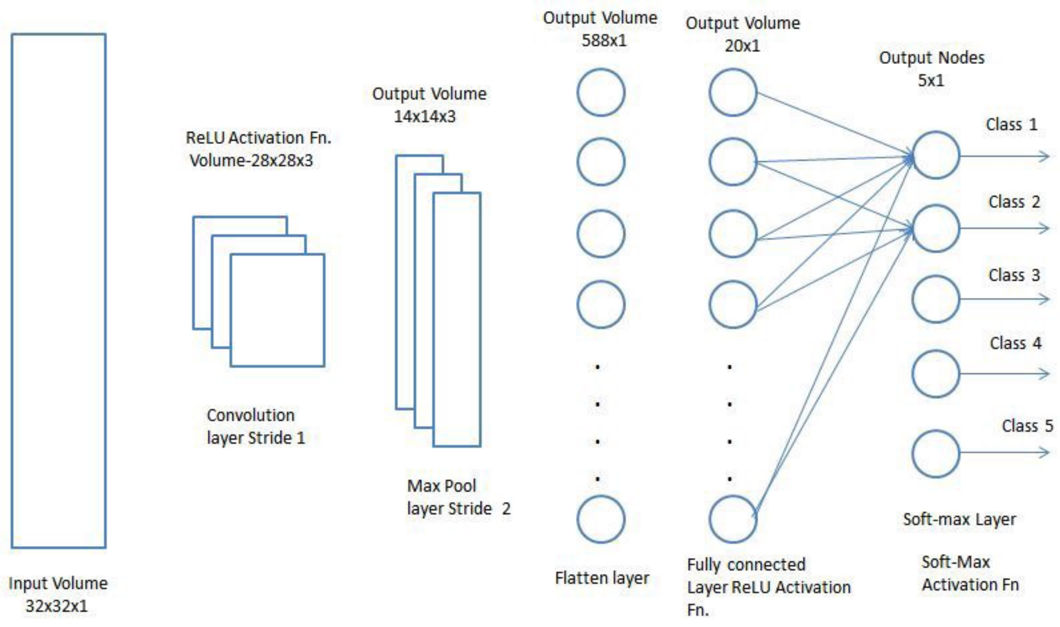


Figure 4: The full structure of the CNN

4.2 MobileNet Architecture

Mobilenet is a neural network architecture which is primarily used for feature extraction and for supporting classification. The idea behind mobilenet is to develop a neural network architecture which is both deep enough to allow for significant learning but also lightweight enough that it can work on any small device. Mobilenet was chosen in order to create this model which works on all sorts of devices to ensure this hand gesture recogniser is accessible to all.

Depthwise Separable Convolution is used to reduce the model size and complexity. It is particularly useful for mobile and embedded vision applications because -

- **Smaller model size:** Fewer number of parameters
- **Smaller complexity:** Fewer Multiplications and Additions (Multi-Adds)

The main ideas in MobileNet Architecture are

4.2.1 Depthwise Separable Convolution

Depthwise separable convolution is a depthwise convolution followed by a pointwise convolution as follows:

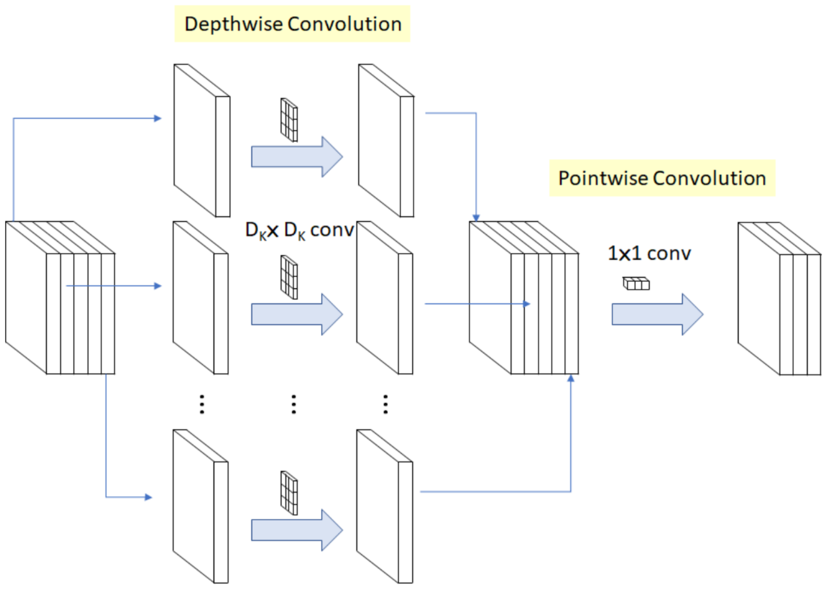


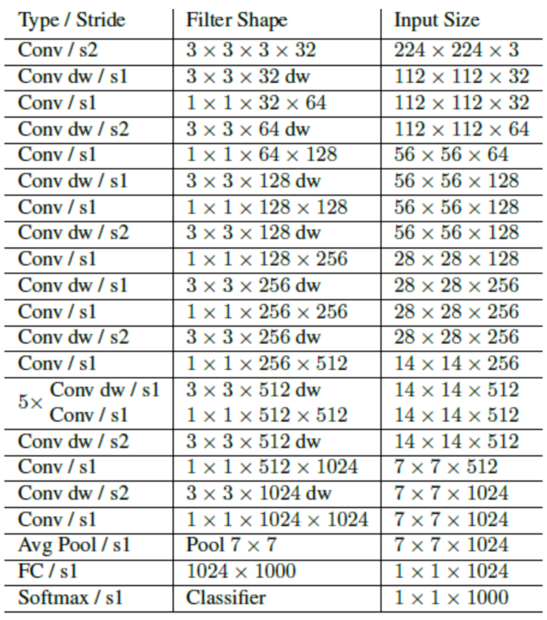
Figure 5: Depthwise Separable Convolution

- Depthwise convolution is the channel-wise DK×DK spatial convolution. Suppose in the figure above, we have 5 channels, then we will have 5 DK×DK spatial convolution.
- Pointwise convolution actually is the 1×1 convolution to change the dimension.

When DK×DK is 3×3, 8 to 9 times less computation can be achieved, but with only small reduction in accuracy.

4.2.2 MobileNet Architecture

The architecture of the MobileNet is described in the table below



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

Figure 6: Different layers in the MobileNet

In this architecture, batch normalization and ReLU are applied after every convolution layer. Adding this to the Depthwise Seperable Convolution layers gives us a huge reduction in the number of multiplication and additions steps to be performed in each layer and its calculation.

5 Implementation

5.1 CNN model

This model was developed as a 12 layer model, with the 12th layer being a softmax activation function in order to classify the inputs into the different categories. Of these 12 layers 6 are convolutional layers to aid in image recognition. The number of layers was chosen to be this number to both have a deep enough network to aid in recognition and yet be light enough to be trained easily

The optimizer used is an ADAM optimizer and the loss function used is categorical cross-entropy loss function.

5.2 Mobilenet Model

The MobileNet model is created as a function from the tensorflow library in python. It has 28 layers including ReLU, convolutional and padding layers. This model was also trained with an ADAM optimizer and a categorical cross-entropy loss function.

5.3 Source Code

5.3.1 CNN Model

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, Flatten,
   BatchNormalization, Conv2D, MaxPool2D,Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix
import itertools
import os
import shutil
import random
import glob
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
%matplotlib inline

path = "./asl/asl_alphabet_train/asl_alphabet_train/"

target_size = (200, 200)
target_dims = (200, 200, 3) # add channel for RGB
n_classes = 29
val_frac = 0.1
batch_size = 64

data_augmentor = ImageDataGenerator(samplewise_center=True,
                                    samplewise_std_normalization=True,
                                    validation_split=val_frac)

train_generator = data_augmentor.flow_from_directory(
path,target_size=target_size, batch_size=batch_size,
                      shuffle=True,subset="training")
val_generator = data_augmentor.flow_from_directory(
path,target_size=target_size, batch_size=batch_size,
                                   subset="validation")

import cv2
def plot_three_samples(letter):
    print("Samples images for letter " + letter)
    base_path = path
    img_path = base_path + letter + '/**'
    #print(img_path)
    path_contents = glob.glob(img_path)

    plt.figure(figsize=(16,16))
    imgs = random.sample(path_contents, 3)
    plt.subplot(131)
    plt.imshow(cv2.cvtColor(cv2.imread(imgs[0]),cv2.COLOR_BGR2RGB))
    plt.subplot(132)
    plt.imshow(cv2.cvtColor(cv2.imread(imgs[1]),cv2.COLOR_BGR2RGB))
    plt.subplot(133)
    plt.imshow(cv2.cvtColor(cv2.imread(imgs[2]),cv2.COLOR_BGR2RGB))
    return

plot_three_samples('M')

my_model = Sequential()
my_model.add(Conv2D(64, kernel_size=4, strides=1, activation='relu',
   input_shape=target_dims))
my_model.add(Conv2D(64, kernel_size=4, strides=2, activation='relu'))
my_model.add(Dropout(0.5))
my_model.add(Conv2D(128, kernel_size=4, strides=1, activation='relu'))
my_model.add(Conv2D(128, kernel_size=4, strides=2, activation='relu'))
my_model.add(Dropout(0.5))
my_model.add(Conv2D(256, kernel_size=4, strides=1, activation='relu'))
my_model.add(Conv2D(256, kernel_size=4, strides=2, activation='relu'))
my_model.add(Flatten())
my_model.add(Dropout(0.5))
my_model.add(Dense(512, activation='relu'))
my_model.add(Dense(n_classes, activation='softmax'))

my_model.compile(optimizer='adam', loss='categorical_crossentropy',
   metrics=["accuracy"])

my_model.fit_generator(train_generator, epochs=5, validation_data=
   val_generator)

my_model.evaluate(val_generator)
my_model.save('saved_model/my_model')
new_model = tf.keras.models.load_model('saved_model/my_model')
new_model.summary()
```

5.3.2 MoblieNet Model

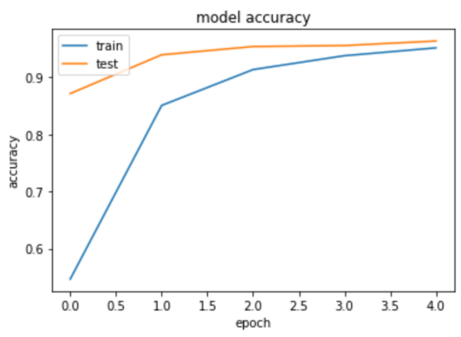
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential,Model
from tensorflow.keras.layers import Activation, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy

os.chdir("/content/asl_alphabet")
if os.path.isdir('train') is False:
    os.mkdir('train')
    os.mkdir('valid')
    os.mkdir('test')
    for i in all:
        shutil.move(f'{i}', 'train')
        os.mkdir(f'valid/{i}')
        os.mkdir(f'test/{i}')

        valid_samples = random.sample(os.listdir(f'train/{i}'), 100)
        for j in valid_samples:
            shutil.move(f'train/{i}/{j}', f'valid/{i}')

        test_samples = random.sample(os.listdir(f'train/{i}'), 50)
        for k in test_samples:
            shutil.move(f'train/{i}/{k}', f'test/{i}')
os.chdir('../..')

train_path = "/content/asl_alphabet/train"
valid_path = "/content/asl_alphabet/valid"
test_path = "/content/asl_alphabet/test"

train_batches = ImageDataGenerator(preprocessing_function=tf.keras.
    applications.mobilenet.preprocess_input).flow_from_directory(
    directory=train_path, target_size=(224,224), batch_size=256)

valid_batches = ImageDataGenerator(preprocessing_function=tf.keras.
    applications.mobilenet.preprocess_input).flow_from_directory(
    directory=valid_path, target_size=(224,224), batch_size=64)

test_batches = ImageDataGenerator(preprocessing_function=tf.keras.
    applications.mobilenet.preprocess_input).flow_from_directory(
    directory=test_path, target_size=(224,224), batch_size=64, shuffle
        =False)

mobile = tf.keras.applications.mobilenet.MobileNet()

model.compile(optimizer=Adam(lr=0.0001), loss='
    categorical_crossentropy', metrics=['accuracy'])

model.fit(x=train_batches,
            steps_per_epoch=len(train_batches),
            validation_data=valid_batches,
            validation_steps=len(valid_batches),
            epochs=10,
            verbose=1
)

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
```

```
Normalization can be applied by setting ‘normalize=True‘.
"""
plt.imshow(cm, interpolation='nearest', cmap=cmap)
plt.title(title)
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    print("Normalized confusion matrix")
else:
    print('Confusion matrix, without normalization')

print(cm)

thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape
    [1])):
    plt.text(j, i, cm[i, j],
        horizontalalignment="center",
        color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

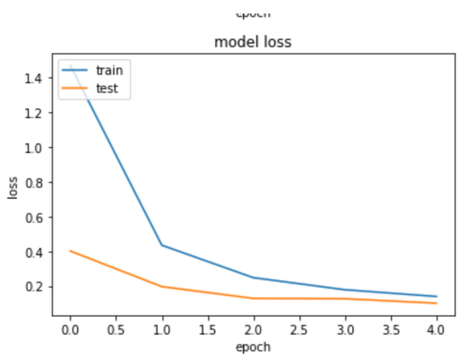
6 Output

6.1 Results

The loss and accuracy during the training of the custom CNN is shown below



(a) Model Accuracy



(b) Model Loss

Figure 7: Result of custom CNN in classification

We then move on to plot the confusion matrix over the test data to see the classification metrics

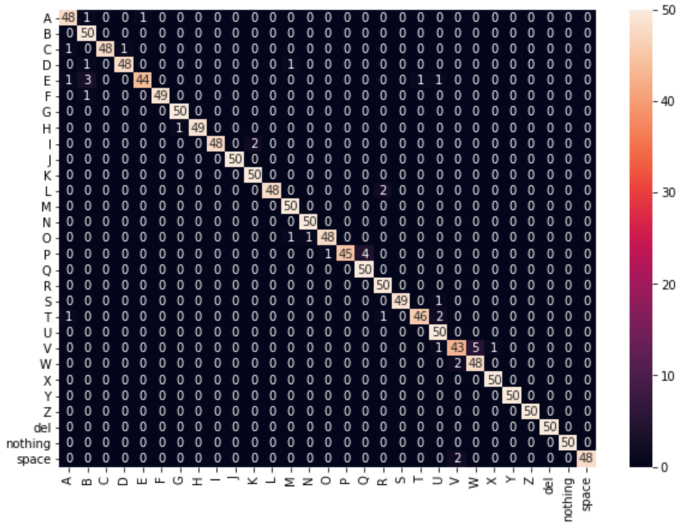
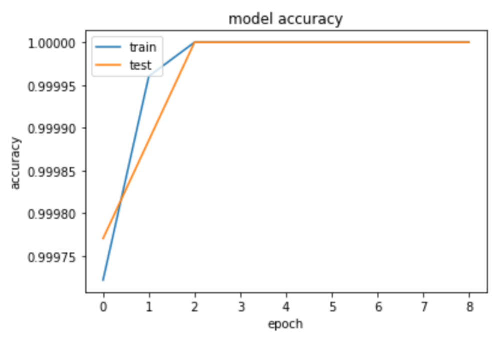
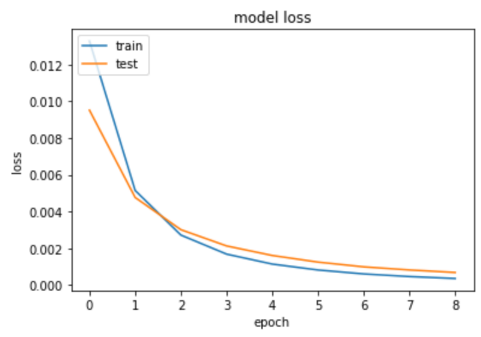


Figure 8: Confusion Matrix for the custom CNN

The same metrics for the MobileNet when trained on the same dataset are shown below



(a) Model Accuracy



(b) Model Loss

Figure 9: Result of MobileNet in classification

For this MobileNet we get the following Confusion Matrix

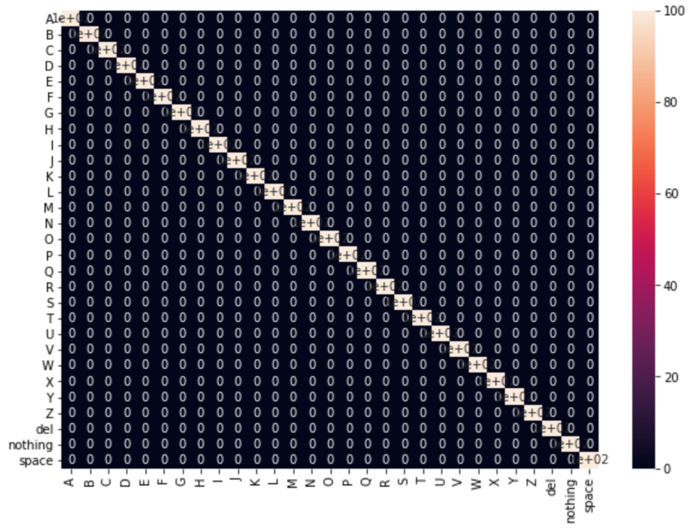


Figure 10: Confusion Matrix for MobileNet

6.2 Interpretation

From the confusion matrices shown above and from the was we can see the change in the model loss and accuracy, the CNN we developed for this specific purpose gives results on par with the MobileNet architecture. Some part of this might be due to our network being developed specifically for this task, while MobileNet is a general framework, but we believe that this level of accuracy presents a significant step forward.

The system however is still overfitting a little and can be improved by fine-tuning the hyperparameters to make the model generalise better on the test data.

7 Conclusion and Future Scope

In this paper we presented a method to classify sign language hand gestures. This method was trained on the data set and a model obtained. Then, a model based on the MobileNet architecture was also trained on the same architecture and the performance of our model was compared to the same. The results show us how we might improve on our own system.

In the future, a more robust system can be developed using the method described here on a system which has sufficient computing power to enable more hidden layers and more training data. This would help that system to generalise on any given input data and help in solving this open problem of sign language recognition.

8 References

[1] B. B. Traore, B. Kamsu-Foguem, and F. Tangara, “Deep convolution neural network for image recognition,” *Ecological Informatics*, vol. 48, p. 257–268, 2018.

[2] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, “Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning,” *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, p. 1285–1298, 2016.

[3] A. Sahoo, G. Mishra, and K. Ravulakollu, “Sign language recognition: State of the art,” *ARPN Journal of Engineering and Applied Sciences*, vol. 9, pp. 116–134, 02 2014.

[4] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2015.

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” 2017.

[6] L. Pigou, S. Dieleman, P.-J. Kindermans, and B. Schrauwen, “Sign language recognition using convolutional neural networks,” *Computer Vision - ECCV 2014 Workshops Lecture Notes in Computer Science*, p. 572–578, 2015.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” 2016.

[8] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, “Scalable object detection using deep neural networks,” *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[9] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, “Object detection with deep learning: A review,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, p. 3212–3232, 2019.