CLASSIFICATION OF AMERICAN HAND SIGNS USING CONVOLUTIONAL NEURAL NETWORKS

(CSE4019)

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology
in
Computer Science and Engineering

BY

18BCE0818 JONATHAN A. PATTA
18BCE2203 RAJIV GUPTA

SLOT G2

Under the guidance of

Prof. Anisha M Lal

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

JUNE 2020

DECLARATION

I hereby declare that the Project entitled

"**CLASSIFICATION OF AMERICAN HAND SIGNS USING CONVOLUTIONAL NEURAL NETWORKS**" submitted by us, for the award of the degree of *Bachelor of Technology in CSE* to VIT is a record of bonafide work carried out by me under the supervision of Anisha M. Lal.

I further declare that the work reported in this Project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 07-06-20

JOHNATHAN RAJIV

CERTIFICATE

This is to certify that the project entitled "**CLASSIFICATION OF AMERICAN HAND SIGNS USING CONVOLUTIONAL NEURAL NETWORKS**" submitted by **JOHNATHAN 18BCE0818 AND RAJIV 18BCE2203**, SCHOOL OF COMPUTER SCIENCE AND ENGINEERING, VIT, for the award of the degree of *Bachelor of Technology in CS* is a record of bonafide work carried out by them.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 07-06-2020

CONTENTS

ABSTRACT

Sign Language Recognition is one of the most growing fields of research area. Many new techniques have been developed recently in this area. The Sign Language is mainly used for communication of deaf-dumb people. Sign language recognition (SLR) aims to interpret sign languages automatically by a computer in order to help the deaf communicate with hearing society conveniently. Our aim is to design a system to help the person who trained the hearing impaired to communicate with the rest of the world using sign language or hand gesture recognition techniques. The aim of this project is to develop an application which will translate sign language to English in the form of text, thus aiding communication with sign language. The application acquires image data using the webcam of the computer, then it is preprocessed using skin adaptation algorithm and recognition is done using convolution neural networks.

Sign Language is the primary means of communication in the deaf and dumb community. As like any other Language it has also got grammar and vocabulary but uses visual modality for exchanging information. The problem arises when dumb or deaf people try to express themselves to other people with the help of these sign language grammars. This is because normal people are usually unaware of these grammars. As a result it has been seen that communication of a dumb person are only limited within his/her family or the deaf community. Sign Language is the most natural and expressive way for the hearing impaired people. People, who are not deaf, never try to learn the sign language for interacting with the deaf people. This leads to isolation of the deaf people. But if the computer can be programmed in such a way that it can translate sign language to text format, the difference between the normal people and the deaf community can be minimized. Indian sign language (ISL) uses both hands to represent each alphabet and gesture. The important research problem in computer recognition is the sign language for enabling communication with hearing impaired people. This system introduces efficient and fast techniques for identification of the hand gesture representing an alphabet of the Sign Language. Currently, more interest is created to do research in the field of sign language recognition system. A computerized interpreter could be a reliable and cheaper alternative. A system that can translate sign language into plain text or audio can help in real-time communication. It can also be used to provide interactive learning of sign language.

This paper proposes a gesture recognition method using convolutional neural networks. The procedure involves the application of morphological filters and segmentation during preprocessing, in which they contribute to a better feature extraction. Training and testing are performed with different convolutional neural networks, compared with architectures known in the literature and with other known methodologies.

KEYWORDS

CONVOLUTIONAL NEURAL NETWORK (CNN), SEGEMENTATION, MORPHOLOGICAL PROCESSING

BACKGROUND

American Sign Language (ASL) substantially facilitates communication in the deaf community. However, there are only ~250,000-500,000 speakers which significantly limit the number of people that they can easily communicate with [1]. The alternative of written communication is cumbersome, impersonal and even impractical when an emergency occurs. In order to diminish this obstacle and to enable dynamic communication, we present an ASL recognition system that uses Convolutional Neural Networks (CNN) in real time to translate a video of a user's ASL signs into text. Our problem consists of three tasks to be done in real time:

1. Obtaining video of the user signing (input)
2. Classifying each frame in the video to a letter
3. Reconstructing and displaying the most likely word from classification scores (output).

From a computer vision perspective, this problem represents a significant challenge due to a number of considerations, including:

- Environmental concerns (e.g. lighting sensitivity, background, and camera position)
- Occlusion (e.g. some or all fingers, or an entire hand can be out of the field of view)
- Sign boundary detection (when a sign ends and the next begins)
- Co-articulation (when a sign is affected by the preceding or succeeding sign)

While Neural Networks have been applied to ASL letter recognition (Appendix A) in the past with accuracies that are consistently over 90% , many of them require a 3-D capture element with motion-tracking gloves or a Microsoft Kinect, and only one of them provides real-time classifications. The constraints imposed by the extra requirements reduce the scalability and feasibility of these solutions.

OVERVIEW/ ALGORITHM

PROPOSED WORK

The techniques reviewed are suitably categorized into different stages: data acquisition, pre-processing, segmentation, feature extraction and classification, where the various algorithms at each stage are elaborated and their merits compared. Further, we also discuss the challenges and limitations faced by gesture recognition research in general, as well as those exclusive to sign language recognition. Overall, it is hoped that the study may provide readers with a comprehensive introduction into the field of automated gesture and sign language recognition, and further facilitate future research efforts in this area.

We aim to create a working model to classify images of hand gestures . We propose a convolutional neural network to classify such images. As the number of classification labels are pretty high, we have scaled the network, evenly and have come to the conclusion that 3 layers was the optimum amount. We also propose a new algorithm for extracting skin color from a picture.

SOFTWARE REQUIREMENTS

We have used the following software and libraries for our project:

1. Python 3.7
2. Anaconda - Spyder
3. Convolutional Neural Networks Library
4. OpenCV library
5. Numpy library
6. Pytorch library

LITERATURE REVIEW

Several types of researches have been done in translating Indian Sign language using deep learning. Some of them used instrument-based approach and some have used a video based approach. In Ref. [1] Pham The Hai uses Microsoft Kinect to translate Vietnamese Sign Language. In the proposed system, the person has to place himself with Kinect's field of view and then perform sign language gestures. It can recognize both static and dynamic gestures using multiclass Support Vector Machine. During recognition, the gesture features are extracted, normalized and filtered out based on Euclidean distance. Purva Badhe [2] uses Fourier Descriptor for feature extraction. The system translates Indian Sign language gestures to English language. To represent the boundary points, the Fourier series were calculated using Fast Fourier Transform (FFT) algorithm. The extracted data being too large is compressed using vector quantization. This data is then stored into a codebook. For testing purpose, the code vector generated from gestures is compared with existing codebook and gesture is recognized. P. Gajalakshmi [3] uses Support Vector Machine and Error Correcting Output Codes for American Sign Language recognition. Muhammad Aminur Rahaman [4] uses a novel method for Bengali hand gesture recognition. The system uses cascaded classifiers to detect the hand in each frame. It captures hand gestures based on Hue and Saturation value from HIS color model. Otiniano-Rodr´ıguez et al. [5] present a methodology in which the features of gesture images are extracted through Hu and Zernike moments, while an SVM is used for classification. Another method is the use of neural networks to classify the data extracted from the images, as presented by the authors Tolba et al. [6], in which a special type of neural network is used, called learning vector quantization. The project from Nguyen et al. [7] was defined by principal component analysis (PCA) to select the best attributes and a neural network for classification. Oyedotun and Khashman [8] presented a methodology that uses several image processing operations to extract the shape of the hand and use it as input to compare two methods of classification: convolutional neural networks and stacked de-noising auto encoder. Chevtchenko et al. [9] propose a method that uses Gabor features, Zernike moments, Hu moments, and contour-based descriptors to improve the features fed to the CNN, which is defined by feature fusion-based convolutional neural network (FFCNN).

DESIGN/ ARCHITECTURE

THREE COMPONENTS/ MODULES:

1. Segmentation by color

2. Morphological Operations

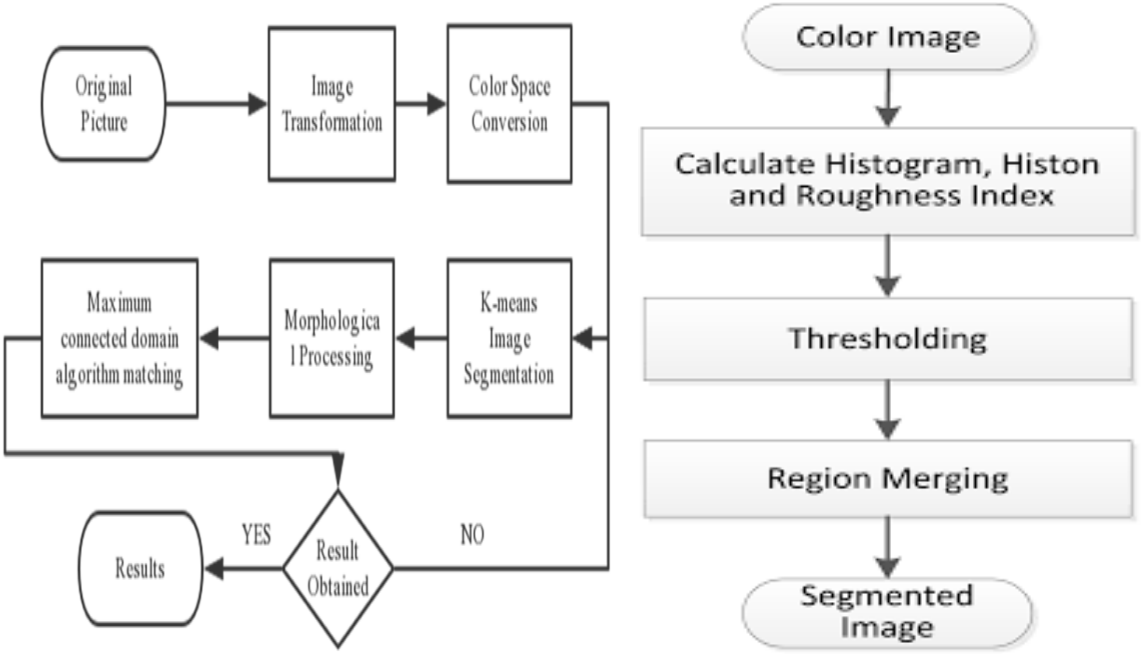
3. Convolutional Neural Networks

SEGMENTATION BY COLOR

Segmentation subdivides an image into regions, so that it is possible to highlight regions that contain characteristics of interest. Therefore, segmentation algorithms can be implemented to separate colors, textures, points, lines, discontinuities, borders, among others. The segmentation process varies according to the problem. In the case of gesture recognition, the entire background region of the image is not of interest, so only the set of pixels with the presence of the human hand must be maintained.

One method for this segmentation is the implementation of background removal, where image samples are collected from the environment and then objects are added to the scene. In this way, the pixels of the new images are compared with the images of the scenario. The regions that show large amount of pixel differences, possibly containing the hand and gesture, are considered foreground. Although it is a good method, this type of segmentation is quite susceptible to variations in lighting.

An alternative is the color-segmentation technique, where it is possible to divide the images into regions that have previously defined color tones. In the case of the presented problem, the color tones to be segmented are similar to human skin tones. To solve this, we can train an MLP network that learns the skin color tones and then classifies which pixels in the image belong to the skin color sets. This method is more robust because it only depends on the correct learning of color sets.



MORPHOLOGICAL OPERATIONS

The use of morphological filters is common as a tool for extracting image components so that they are useful in the representation and description of forms. These filters are applied during image processing to remove or highlight features from segmentation by closing holes and/or reducing noise. They are defined by two elementary operations: erosion and dilation; other important operations are opening and closing. These operations act with structuring elements that have the most diverse forms, thus altering the final result even when the same filter is used. Some examples of structuring elements may be a vector or a square matrix. We use of these elements in an erosion operation will remove lines and square regions from the image, respectively. The closing operation, however, will remove holes contained in the pixel sets of the image.

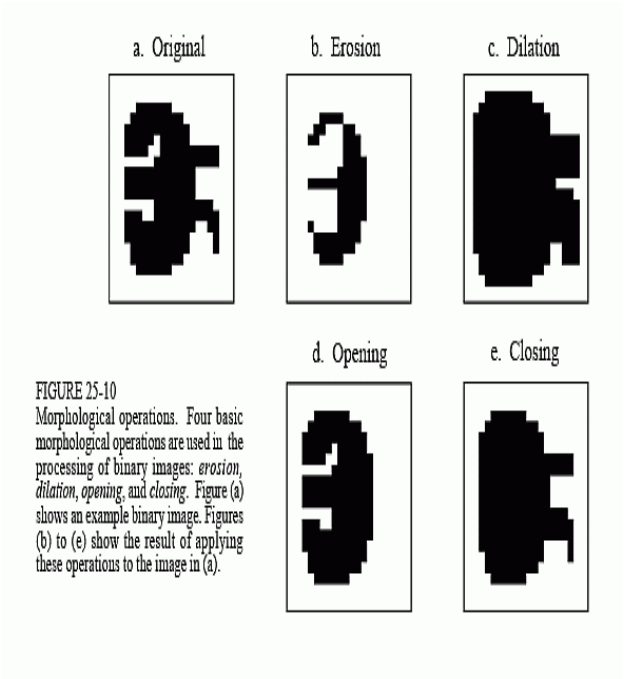
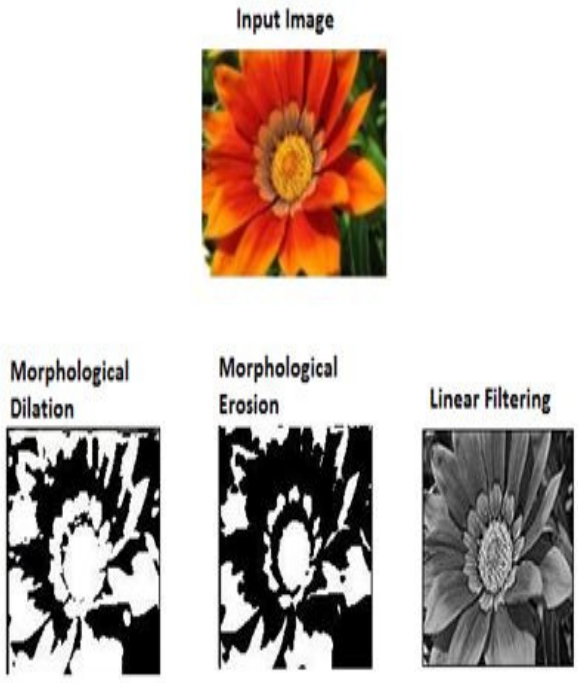


FIGURE 25-10
Morphological operations. Four basic morphological operations are used in the processing of binary images: *erosion*, *dilation*, *opening*, and *closing*. Figure (a) shows an example binary image. Figures (b) to (e) show the result of applying these operations to the image in (a).

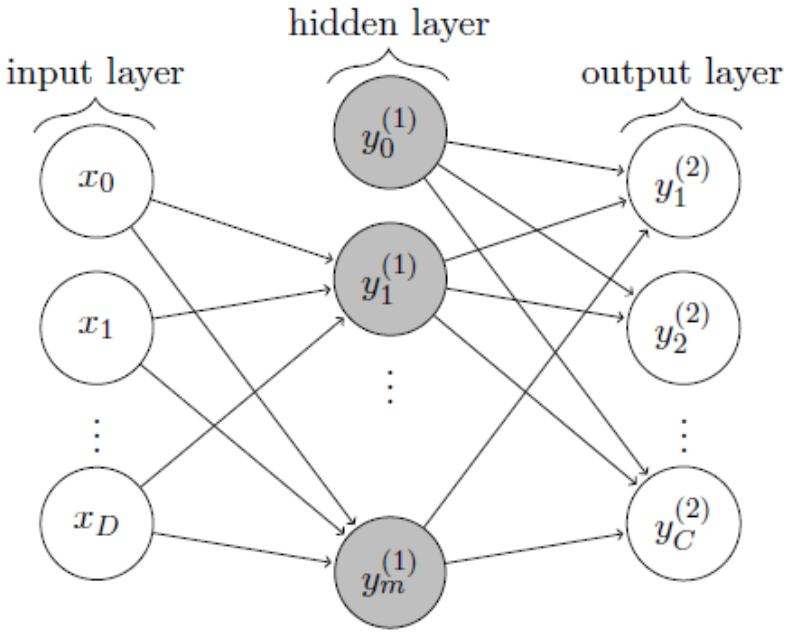


CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks, or CNNs, are widely used for image classification, object recognition, and detection. Three types of layers can summarize its structure: convolution, pooling, and classification. The CNN architecture must be defined according to the application and is usually defined by the number of alternate convolution and pooling players, number of neurons in each layer, and choice of activation function. In the context of image classification, the input for a CNN is an image represented by an arbitrary color model. At the convolution layer, every neuron is associated with a kernel window that is convolved with the input image during CNN training and classification. This convolution kernel is composed of the weights of each associated neuron. The output of this convolution step is a set of N images, one for each of the N neurons. Because of convolution, these new images can contain negative values. In order to avoid this issue, a rectified linear unit (ReLU) is used to replace negative values by zero. The outputs of this layer are called feature maps.

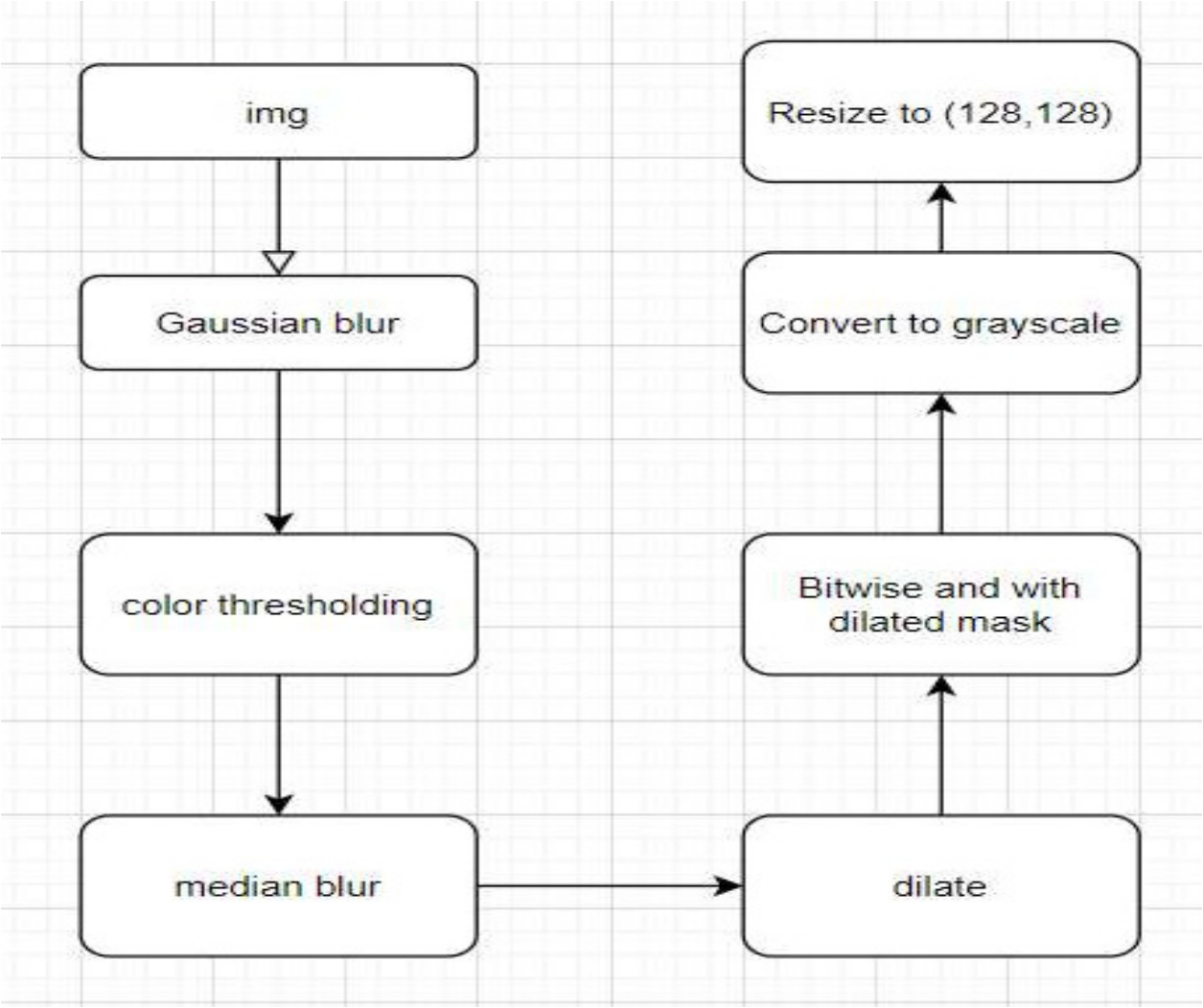
After a convolution layer, it is common to apply a pooling layer. This is important because pooling reduces the dimensionality of feature maps, which subsequently reduces the network training time. Some architecture alternates between convolution and pooling, for example, Google Net has five convolution layers followed by one pooling layer. At the end of the convolution and pooling architectures, there is a multilayer perceptron neural network that performs classification based on the feature maps computed by the previous layers. Because of its large number of layers and successful applications, CNNs are one of the preferred techniques for deep learning. Its architecture allows automatic extraction of diverse image features, like edges, circles, lines, and texture. The extracted features

are increasingly optimized in further layers. It is important to emphasize that the values of the kernel filters applied in the convolution layers are the result of back propagation during CNN training.



FLOWCHART

1. **MORPHOLOGICAL OPERATIONS PERFORMED:**



METHODOLOGY

Following are the methods or algorithms used in our project:

For Training:

- Gaussian blur
- Color threshold keying
- Dilation with morph ellipse kernel
- Convert to grayscale
- Resize to (128,128)
- Pass through CNN model = (2* Convolutional layers + 1*Linear layer)

For Prediction:

- Capture frame from camera
- Gaussian blur
- Color threshold keying
- Dilation with morph ellipse kernel
- Convert to grayscale
- Resize to (128,128)
- Display predicted Label

APPLICATIONS

Hand gesture is one of the methods used in sign language for non-verbal communication. Hand Gesture recognition system provides us an innovative, natural, user friendly way of communication with the computer which is more familiar to the human beings. By considering in mind the similarities of human hand shape with four fingers and one thumb, the software aims to present a real time system for recognition of hand gesture on basis of detection of some shape based features like orientation, Centre of mass centroid, fingers status, and thumb in positions of raised or folded fingers of hand.

When you use the hand signs for letters to spell out a word, you are finger spelling. Finger spelling is useful to convey names or to ask someone the sign for a particular concept. ASL uses one-handed signals for each letter of the alphabet (some other sign languages use both hands for some letters). Many people find finger spelling the most challenging hurdle when learning to sign, as accomplished speakers are very fast finger spellers.

It is most commonly used by deaf & dumb people who have hearing or speech problems to communicate among themselves or with normal people. Various sign language systems has been developed by many makers around the world but they are neither flexible nor cost-effective for the end users

SYSTEM IMPLEMENTATION

GITHUB LINK FOR OUR CODE

https://github.com/Jonathanpatta/asl_image_classifier

Few modules from our Code

1. Data Module

```
import torch
import torchvision
from torchvision.datasets import ImageFolder
from torchvision import transforms
root_folder = r"D:\data\asl\significant-asl-sign-language-alphabet-dataset\Training Set"
IMSIZE = 128
transformations = transforms.Compose([
                                    #transforms.RandomRotation(degrees=20),
                                    transforms.Grayscale(),
                                    transforms.Resize((IMSIZE,IMSIZE)),
                                    #transforms.RandomPerspective(distortion_scale=0.2, p=0.5,
interpolation=3),
                                    transforms.ToTensor(),
                                ]
                                )
images = ImageFolder(root_folder,transform=transformations)
no_of_imgs = len(images.imgs)
image_train,image_test =
torch.utils.data.random_split(images,[int(0.9*no_of_imgs),int(0.1*no_of_imgs)])
'''import matplotlib.pyplot as plt
import cv2'''
Train_loader = torch.utils.data.DataLoader(image_train,
                                        batch_size=100,
                                        shuffle=True,
                                        pin_memory=True,
                                        num_workers=10)
Test_loader = torch.utils.data.DataLoader(image_test,
                                        batch_size=100,
                                        shuffle=True,
                                        num_workers=0)
print("loaded")
```

2. MainModel Module

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from tqdm import tqdm
class MyModel(nn.Module):
    def __init__(self, img_size=64, no_of_classes=2, ksize=5, dim=32):
        self.no_of_classes = no_of_classes
        self.img_size = img_size
        self.ksize = ksize
        self.dim = dim

        super().__init__()
        self.conv1 = nn.Conv2d(1,dim,ksize)
        self.conv2 = nn.Conv2d(dim,dim*2,ksize)
        self.conv3 = nn.Conv2d(dim*2,dim*4,ksize)
        self.linput_shape = None
        #self.forward(torch.randn(self.img_size,self.img_size).view(-
1,1,self.img_size,self.img_size))
        self.convpass(torch.randn(self.img_size,self.img_size).view(-
1,1,self.img_size,self.img_size))
        self.linear1 = nn.Linear(self.linput_shape,512)
        self.linear2 = nn.Linear(512,self.no_of_classes)

    def convpass(self,x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv3(x)), (2, 2))
        if self.linput_shape is None:
            self.linput_shape = x[0].shape[0]*x[0].shape[1]*x[0].shape[2]
        return x
    def forward(self,x):
        x = self.convpass(x)
        x = x.view(-1, self.linput_shape)
        x = F.relu(self.linear1(x))
        x = self.linear2(x)

        return F.softmax(x, dim=1)
```

3. Model Functions Module

```
from mainmodel import MyModel
from data import Train_loader,Test_loader,image_test
from tqdm import tqdm
import torch
import torch.optim as optim
import torch.nn as nn
import torch.nn.functional as F
from time import time
def one_hot(x, class_count):
    return torch.eye(class_count)[x,:]
def train():
    device = torch.device('cuda')
    MODEL_PATH = "asl_classifier.pt"
    BATCH_SIZE = 100
    EPOCHS = 3
    img_size = 128
    net = MyModel(img_size=128, no_of_classes=27).to(device)
    net.load_state_dict(torch.load(MODEL_PATH, map_location=device))

    optimizer = optim.Adam(net.parameters(), lr=0.001)
    loss_function = nn.MSELoss()
    for epoch in range(EPOCHS):
        imgno=0
        #net.zero_grad()
        for imgs,labels in Train_loader:
            labels = one_hot(labels,27)
            imgs,labels = imgs.to(device),labels.to(device)
            optimizer.zero_grad()
            outputs = net(imgs)
            acc = 0
            for i,output in enumerate(outputs):
                if output.argmax()==labels[i].argmax():
                    acc+=1
            loss = loss_function(outputs,labels)
            loss.backward()
            optimizer.step()
            imgno += 100
            print(f"Epoch:{epoch}  img:{imgno}  loss:{loss} acc:{acc}")
        torch.save(net.state_dict(), MODEL_PATH)
def predict():
    with torch.no_grad():
        device=torch.device('cpu')
        net = MyModel(img_size=128, no_of_classes=27).to(device)
        net.load_state_dict(torch.load(r"asl_classifier.pt", map_location=device))
        for j,(imgs,labels) in enumerate(Test_loader):
            labels = one_hot(labels,27)
```

```
            imgs,labels = imgs.to(device),labels.to(device)
            outputs = net(imgs)
            #print([output.argmax() for output in outputs])
            #print([label.argmax() for label in labels])
            acc = 0
            for i,output in enumerate(outputs):
                if output.argmax()==labels[i].argmax():
                    acc+=1
            print(f"acc:{acc}")
            if j%5==4:
                break
import cv2
def predict_img(img):

    #img = cv2.imread(r"D:\data\asl\significant-asl-sign-language-alphabet-dataset\Training
Set\F\color_5_0002 (4).png",0)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.resize(img,(128,128))
    with torch.no_grad():
        device=torch.device('cpu')
        net = MyModel(img_size=128, no_of_classes=27).to(device)
        net.load_state_dict(torch.load(r"asl_classifier.pt", map_location=device))
        return int(net(torch.Tensor(img).view(1,1,128,128)).argmax())
if __name__=='__main__':

    train()
```

4. Video read

```
import numpy as np
import cv2
import torch
import torch.nn.functional as F
import torch.nn as nn
from mainmodel import MyModel
from model_funcs import predict_img
cap = cv2.VideoCapture(0)
'''
class Torchconvnet(nn.Module):
    def __init__(self):
        self.img_size = 128
        self.no_of_classes = 12
        super().__init__() # just run the init of parent class (nn.Module)
        self.conv1 = nn.Conv2d(1, 32, 5) # input is 1 image, 32 output channels, 5x5 kernel /
window
        self.conv2 = nn.Conv2d(32, 64, 5) # input is 32, bc the first layer output 32. Then we say
the output will be 64 channels, 5x5 kernel / window
        self.conv3 = nn.Conv2d(64, 128, 5)
        x = torch.randn(self.img_size,self.img_size).view(-1,1,self.img_size,self.img_size)
        self._to_linear = None
        self.convs(x)
        self.fc1 = nn.Linear(self._to_linear, 512) #flattening.
        self.fc2 = nn.Linear(512, self.no_of_classes) # 512 in, 2 out bc we're doing 2 classes (dog
vs cat).
    def convs(self, x):
        # max pooling over 2x2
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv3(x)), (2, 2))
        if self._to_linear is None:
            self._to_linear = x[0].shape[0]*x[0].shape[1]*x[0].shape[2]
        return x
    def forward(self, x):
        x = self.convs(x)
        x = x.view(-1, self._to_linear)  # .view is reshape ... this flattens X before
        x = F.relu(self.fc1(x))
        x = self.fc2(x) # bc this is our output layer. No activation here.
        return F.softmax(x, dim=1)'''
fingers = r"C:\Users\Jonathan\Documents\pytorch\image_proc_project\fingers\data.pt"
device=torch.device('cpu')
net = MyModel(img_size=128, no_of_classes=27).to(device)
net.load_state_dict(torch.load(r"C:\Users\Jonathan\Documents\pytorch\image_proc_project\asl\asl_cla
ssifier.pt", map_location=device))
def predict_imgs(net,img_path):
    import cv2
```

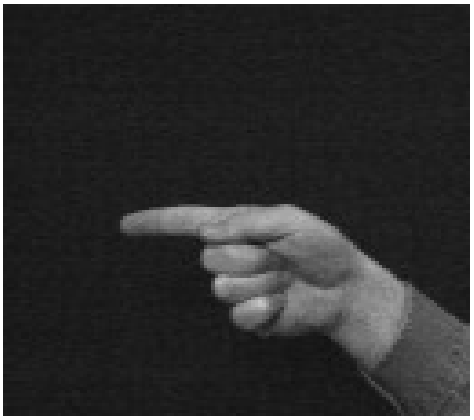
```
    #img_path = r"C:\Users\Jonathan\Downloads\banana.jpg"
    with torch.no_grad():
        #img = cv2.imread(img_path,cv2.IMREAD_GRAYSCALE)
        img = img_path
        img = cv2.resize(img, (128,128))
        imgtensor = torch.Tensor(img).view(-1,128,128)
        imgtensor /= 255
        #imgtensor = test_X[23]
        net_out = net(imgtensor.view(-1, 1, 128, 128))[0]  # returns a list,
        predicted_class = torch.argmax(net_out).tolist()
    #print(predicted_class,net_out[predicted_class],net_out.tolist())
    print(net_out[predicted_class])
#predict_img(net,r"C:\Users\Jonathan\Downloads\IMG_2032.jpg")
import time
import cv2
alphabet = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','
','t','u','v','w','x','y','z']
t = time.time()

while(True):
    ret, frame = cap.read()
    h,w,_ = frame.shape
    frame = frame[0:int(h), 0:int(w/2)]
    frame = cv2.flip(frame,1)
    cv2.imshow('frame',frame)

    cv2.imshow('frame',frame)
    if cv2.waitKey(1) & 0xFF == ord('t'):
        print(alphabet[predict_img(frame)])
        #print(1/(time.time()-t))
    t = time.time()
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

RESULTS AND DISCUSSIONS

Dataset for the project was collected from Google images for the purpose of training the algorithms.

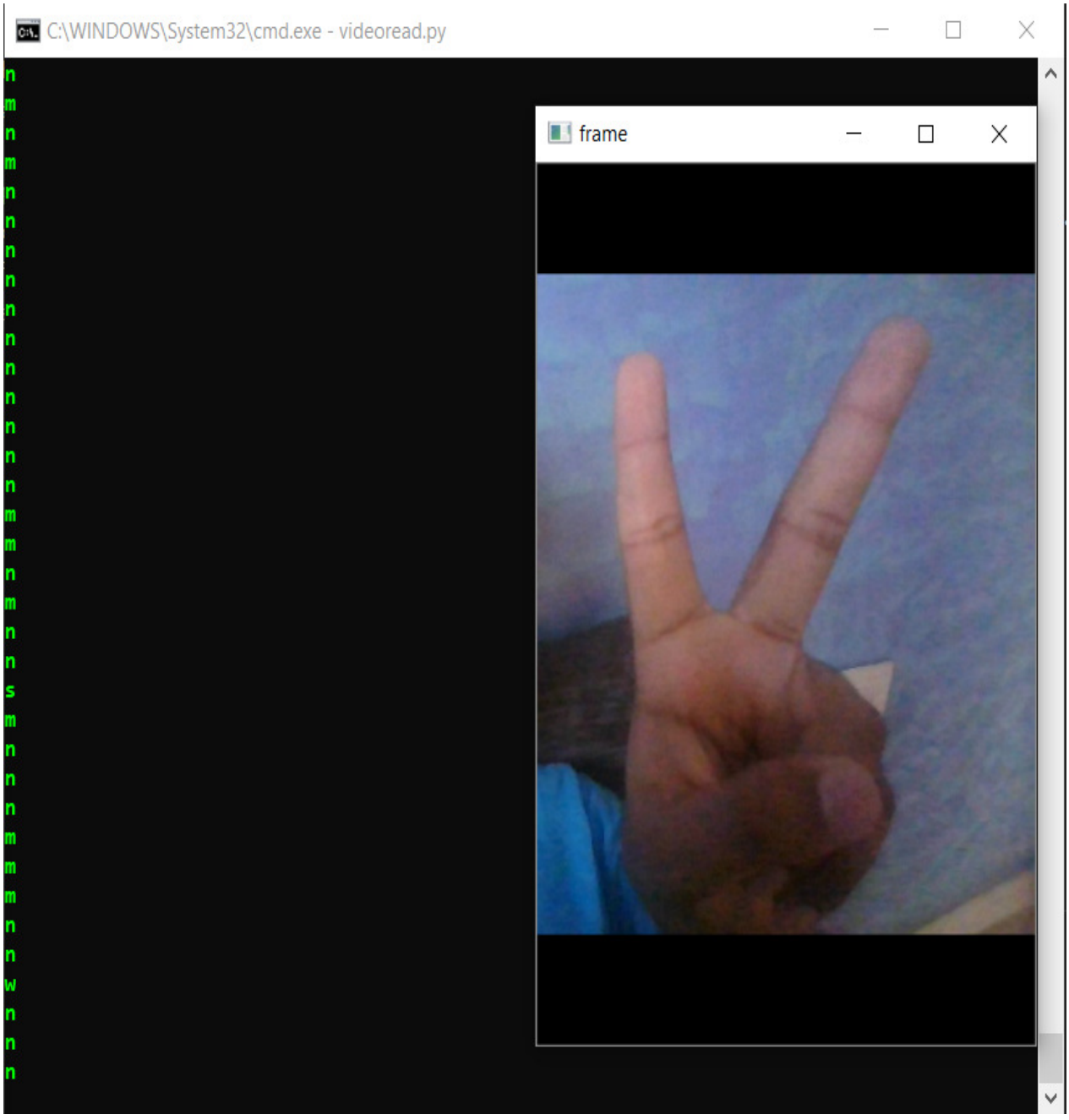
Sample:

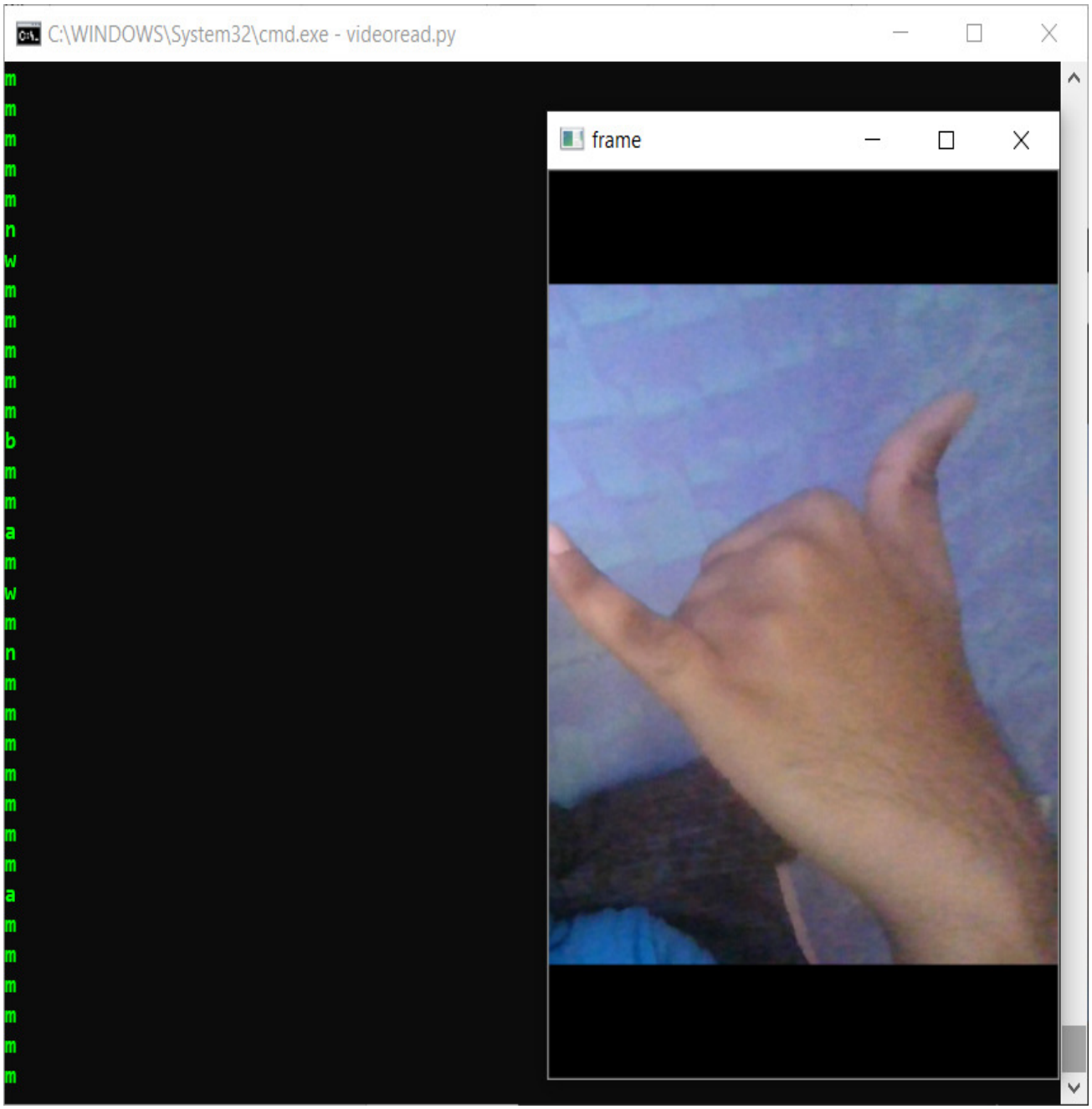
Hand Sign Dataset

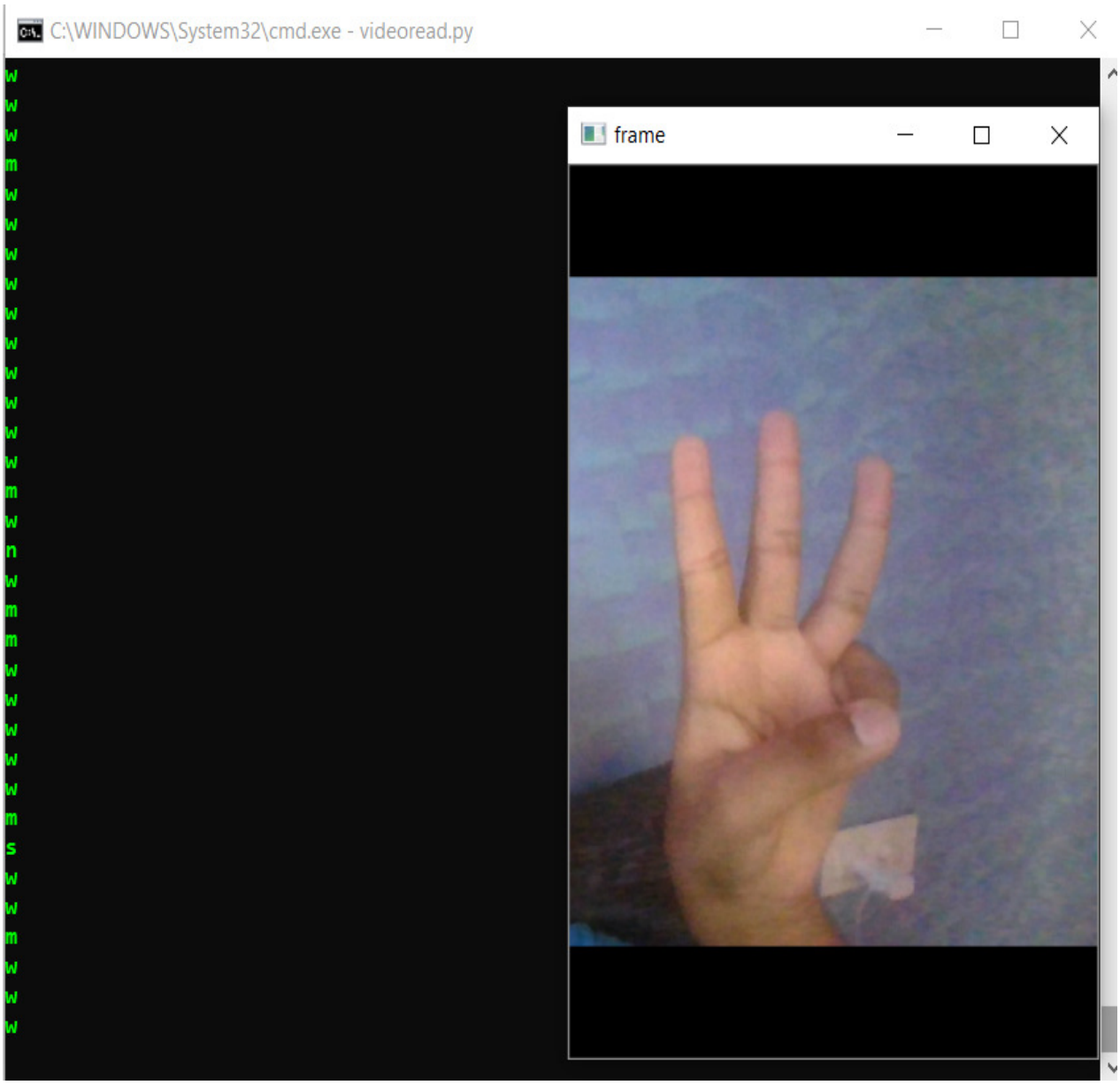


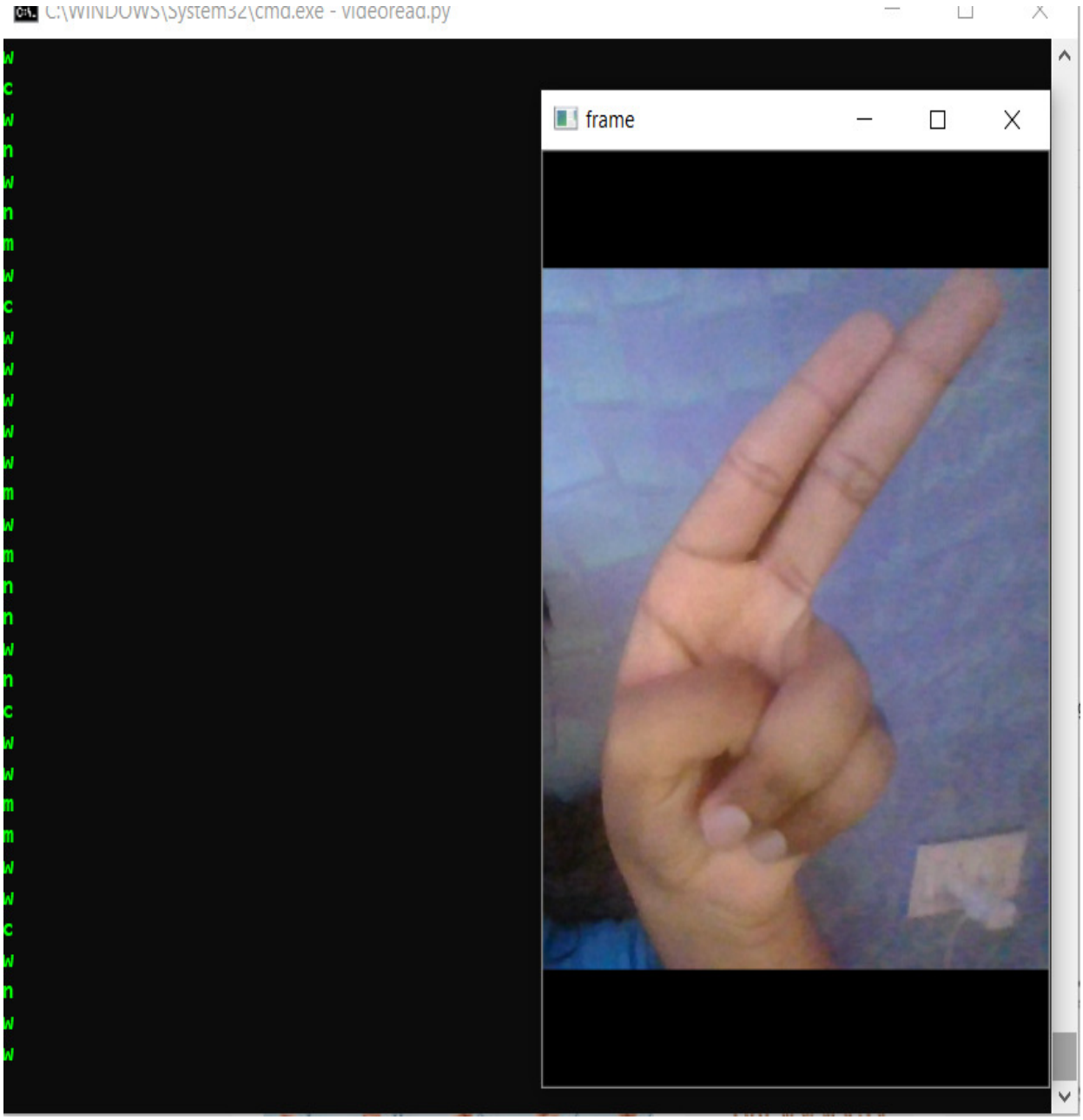
After using labeled images (over thousands of images) to train the different classifiers we use non labeled images as the test set and the classifier correctly predicts the images into their respective classes, although the correctness would be around 94-97% if we had used more number of images.

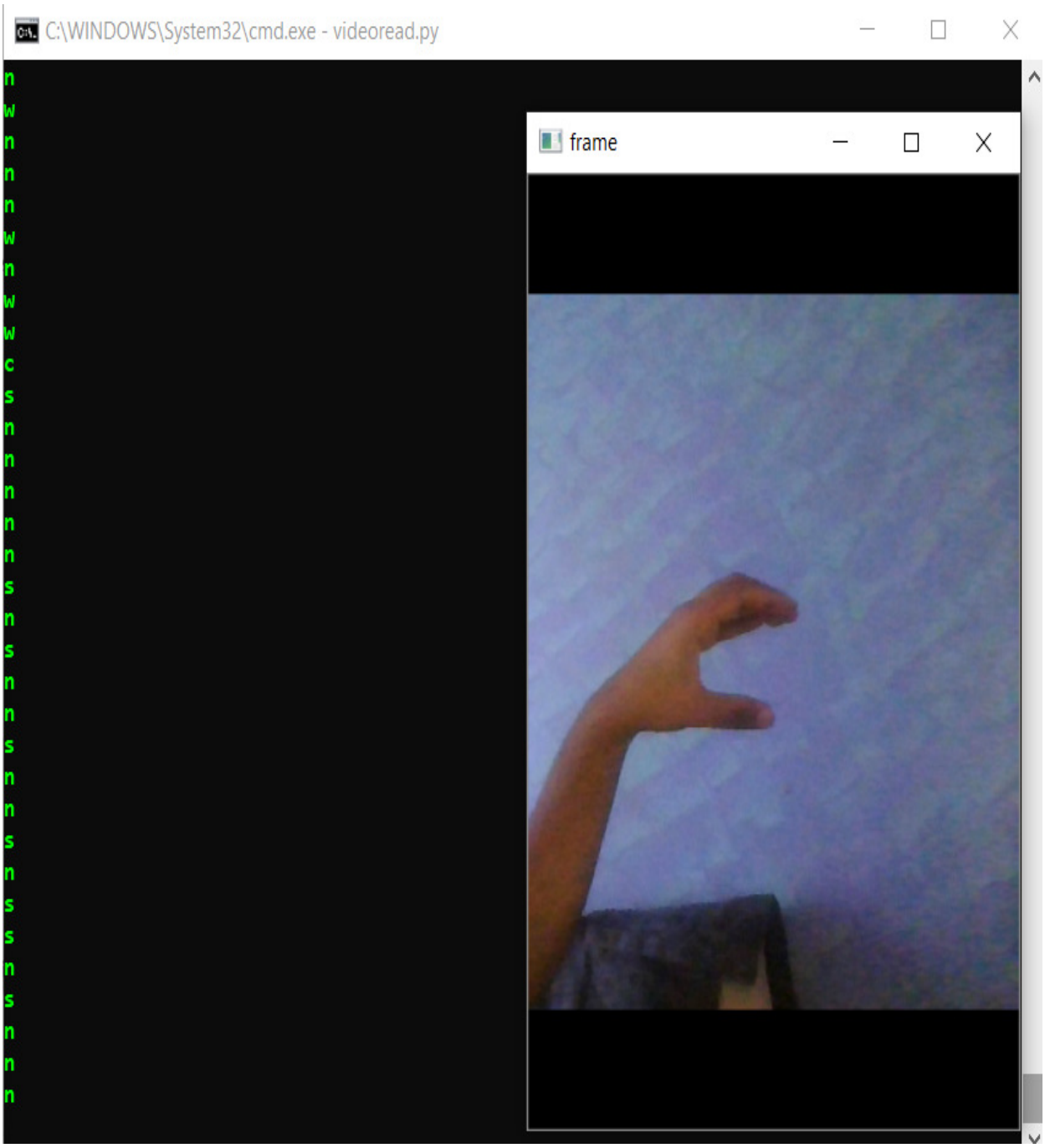
OUTPUT IMAGES




C:\WINDOWS\System32\cmd.exe - videoread.py
frame


C:\WINDOWS\System32\cmd.exe - videoread.py
frame
w
w
w
m
w
w
w
w
w
w
w
w
w
m
w
n
w
m
m
w
w
w
w
w
m
s
w
w
m
w
w
w


C:\WINDOWS\system32\cmd.exe - videoread.py
frame


C:\WINDOWS\System32\cmd.exe - videoread.py
n
w
n
n
n
w
n
w
w
c
s
n
n
n
n
n
s
n
s
n
n
s
n
n
s
n
s
s
n
s
n
n
n
frame

CONCLUSION

One of the problems in gesture recognition is dealing with the image background and the noise often present in the regions of interest, such as the hand region. The use of neural networks for color segmentation, followed by morphological operations presented excellent results as a way to separate the hand region from the background and to remove noise. This step is important because it removes image objects that are not relevant to the classification method, allowing the convolutional neural network to extract the most relevant gesture features through their convolution and pooling layers and, therefore, to increase network accuracy.

The proposed methodology and CNN architecture open the door to a future implementation of gesture recognition in embedded devices with hardware limitations. The proposed methodology approaches only cases of gestures present in static images, without hand detection and tracking or cases of hand occlusion. In the future, we intend to work on these particular cases in a new data preprocessing methodology, investigating other techniques of color segmentation and deep learning architectures

Data Availability

In this article, two image databases were used, they were used together. One of them is known in the literature and has been referenced. The other was produced by the team members, and it cannot be made publicly available at this time.

FUTURE WORK

We believe that the classification task could be made much simpler if there is very heavy preprocessing done on the images. This would include contrast adjustment, background subtraction and potentially cropping. A more robust approach would be to use another CNN to localize and crop the hand.

REFERENCES

[1] Hải, P.T., Thinh, H.C., Phuc, B.V., & Kha, H.H. (2018). Automatic feature extraction for Vietnamese sign language recognition using support vector machine. 2018 2nd International Conference on Recent Advances in Signal Processing, Telecommunications & Computing (SigTelCom), 146-151.

https://doi.org/10.1109/sigtelcom.2018.8325780

[2] P. C. Badhe and V. Kulkarni, "Indian sign language translator using gesture recognition algorithm," 2015 IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS), Bhubaneswar, 2015, pp. 195-200.

https://doi.org/10.1109/CGVIS.2015.7449921

[3] P.Gajalaxmi, T, Sree Sharmila, "Sign Language Recognition for Invariant features based on multiclass Support Vector Machine with BeamECOC Optimization", IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI-2017).

https://issuu.com/irjet/docs/irjet-v6i4413

[4] M. A. Rahaman, M. Jasim, M. H. Ali and M. Hasanuzzaman, "Real-time computer vision-based Bengali Sign Language recognition," 2014 17th International Conference on Computer and Information Technology (ICCIT), Dhaka, 2014, pp. 192-197.

https://doi.org/10.1109/ICCITechn.2014.7073150.

[5] K. C. Otiniano-Rodr´ıguez, G. Camara-Ch´avez, and D. Menotti, "Hu and zernike moments for sign language recognition," in Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV), November 2012.

https://doi.org/10.1007/s11042-018-6102-6

[6] Elsoud, A. S. T. M. A., & Elnaser, O. A. (2009). LVQ for hand gesture recognition based on DCT and projection features. Journal of Electrical Engineering, 60(4), 204-208.

https://pdfs.semanticscholar.org/f024/2d1f6215060c0b94c2c45295b818f449f385.pdf

[7] T.-N. Nguyen, H.-H. Huynh, and J. Meunier, "Static hand gesture recognition using principal component analysis combined with artificial neural network," Journal of Automation and Control Engineering, vol. 3, no. 1, 2015.

https://doi.org/10.12720/joace.3.1.40-45

[8] O. K. Oyedotun and A. Khashman, "Deep learning in visionbased static hand gesture recognition," Neural Computing and Applications, vol. 28, no. 12, pp. 3941–3951, 2017.

https://dl.acm.org/doi/10.1007/s00521-016-2294-8

[9] Chevtchenko, S. F., Vale, R. F., Macario, V., & Cordeiro, F. R. (2018). A convolutional neural network with feature fusion for real-time hand posture recognition. Applied Soft Computing, 73, 748-766.

https://doi.org/10.1016/j.asoc.2018.09.010

[10] Pinto, Raimundo & Braga Borges, Carlos & Almeida, Antonio & Paula Jr, Ialis. (2019). Static Hand Gesture Recognition Based on Convolutional Neural Networks. Journal of Electrical and Computer Engineering. 2019. 1-12.

https://doi.org/10.1155/2019/4167890

[11] Cheok, M.J., Omar, Z. & Jaward, M.H. A review of hand gesture and sign language recognition techniques. Int. J. Mach. Learn. & Cyber. **10,** 131–153 (2019).

https://doi.org/10.1007/s13042-017-0705-5

[12] Pisharady, P. K., & Saerbeck, M. (2015). Recent methods and databases in vision-based hand gesture recognition: A review. Computer Vision and Image Understanding, 141, 152-165.

https://doi.org/10.1016/j.cviu.2015.08.004

[13] Nikam, A. S., & Ambekar, A. G. (2016, November). Sign language recognition using image based hand gesture recognition techniques. In 2016 Online International Conference on Green Engineering and Technologies (IC-GET) (pp. 1-5). IEEE.

https://sci-hub.tw/10.1109/GET.2016.7916786