BILKENT UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

CS 464 Introduction to Machine Learning

TERM PROJECT

Final Report

SIGN LANGUAGE DETECTION

Group 26:

Bertan Köfön

Emre Tarakçı

Osman Serhat Yılmaz

Ayberk Güven

Section: 2

28.12.2022

1. Introduction

Sign language is a type of manual communication that is mostly used by people with hearing impairment [1]. Sign language is very useful since it enables hearing impaired people to communicate with other people. However, there is no universal sign language that is being used all around the world. Different sign languages are being used in different countries. The project will mainly focus on American Sign Language (ASL) which is a complete natural language that has the same linguistic properties as the spoken languages [2].

Apart from the lack of a universal sign language, the number of hearing people that use sign language is very limited. Thus, it is quite challenging for hearing impaired people to find someone who can help them in difficult situations. Considering this real-life problem, detection of sign language can be useful in providing communication between a person who knows sign language and another person who does not. It can also be used as a tool to enable people who know sign language to be able to communicate with a computer using it. There are various machine learning methods currently developed and used that aim to improve the communication between hearing impaired people and hearing people.

This term project aims to implement machine learning algorithms that can classify the letters of American Sign Language (ASL). As the first step of the project the “Sign Language (ENG Alphabet)” dataset on Kaggle has been chosen and examined. Feature extraction is performed on the images in the dataset for dimensionality reduction to decrease the sizes of images in order to achieve ease of computation. After preprocessing the images they were converted to arrays that consist their grayscale pixel information. Having preprocessed the dataset, machine learning classification models such as k-Nearest Neighbor (k-NN), and Support Vector Machine (SVM) were used for supervised learning. k-NN is also known as “lazy learning” and it is based on the principle that the instances with similar properties will be in close proximity with each other in a dataset [3]. SVM is a supervised learning algorithm and it is a popular pattern recognition learning technique for classification regression analysis [3]. A deep learning algorithm which is Convolutional Neural Networks (CNN) is planned to be used in the following stage of the project.

2. Background Information

2.1 k-Nearest Neighbor (kNN)

k-NN is a supervised learning classifier which uses proximity with the training data for classifying new data [4]. There is an underlying assumption that the data points that are near each other belong to the same class.

In the case of image recognition, the csv data that is produced as a result of the pre-processing is used as the training data for the kNN algorithm. In kNN, there is no real learning, since there are no parameters that are updated with each training data row, but when data is classified, the algorithm finds the label that produces the minimum distance to the new data. Distance is used in a sense of visual representation in this case, but since our images have 576 dimensions each, the term distance is used in an abstract manner in this case. A proximity map of a kNN algorithm can be seen in fig. 1 below.

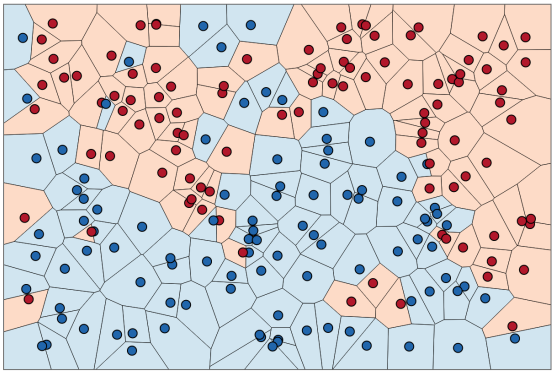

Fig.1: Visualization of Nearest Neighborhoods [5]

2.2 Support Vector Machine (SVM)

SVM is a supervised learning algorithm that can be used for classification and regression problems. [6] SVM works by choosing extreme points that help the algorithm create a hyperplane that can divide n-dimensional space into classes. In our case, each point will be the result of the extracted features with a row of pixels. SVM will create a separation between the classes using the edge cases in our sign language data. How SVM operates can be seen in the fig. 2 below.

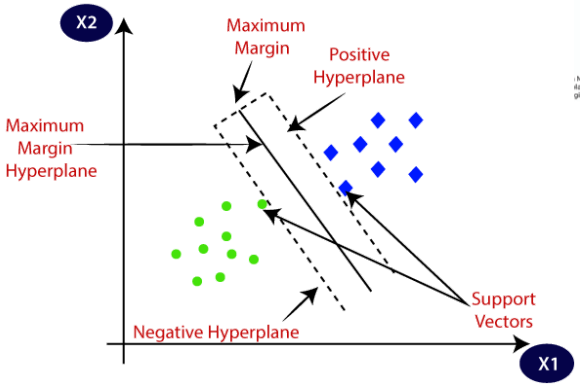

Fig.2: Two different categories that are classified using a decision boundary or hyperplane [6]

2.3 Convolutional Neural Network (CNN)

A convolutional neural network is a deep learning algorithm. It is composed of an input layer, an output layer and hidden layers between the input and output layers. In the hidden layers, operations are performed that change the data with the goal of learning from the data. The common operations on the layers are convolution, ReLU and pooling. A CNN has the same weight and biases in a given layer, which means the neurons in the layer are looking for the same feature. In our case, the input layers will be consisted of the extracted pixel values of the images. However, since the CNN will be learning the features itself, compressing the images and making them greyscale may reduce the information that it can learn from. For that reason, we will first try to implement CNN without the compression and grayscaling operation.

3. Implementation Progress

We divided the project into 4 parts, the first being preprocessing the images and others being using training models for this dataset (kNN, SVM and CNN). We decided to use Python as our language to implement this project. Our aim is to compare the accuracy of training models on the same dataset. As we stated in our proposal, we selected the dataset from Roboflow, it is titled “American Sign Language Letters Dataset”. Dataset consists of 1728 JPG images, the dataset is divided into three sets. Training set consists of 1368 images. Validation set has 144 images and test set has 68 images. Total size of the dataset is 20.1 MB.

Fig.3: Example images from the dataset with different backgrounds

3.1 Data Preprocessing

The dataset that we have used initially contained 456 images. In order to obtain a bigger dataset, we applied augmentation to our original images. Augmentation processes included rotation, cropping, and changing the color palette.

Fig. 4: Different augmentation of a sample image from the dataset

For the kNN and SVM models, we decided to shrink our image sizes and convert them to grayscale for these basic models to interpret the results more accurately. We converted all our images from size 416x416 to 24x24 to shrink the images, for this we used the PIL library of Python. Then using the pandas library, we converted 24x24 grayscale images to data frames, then we wrote the pixel values in these data frames to a csv file as one line representing an image. Here is our ProcessImage method that we used to shrink images and put them in a data frame named all_data:

```
img = Image.open('image.png') #an imported png file
all_data = pd.DataFrame()

def ProcessImage(img, label):
   # dimensionality reduction
   new_img = img.resize((24, 24))

   # make it black and white
   new_img = new_img.convert('L')
   img_array = np.array(new_img)
   df = pd.DataFrame(img_array)

   ndf = df.unstack().to_frame().T
   ndf['label'] = label

   global all_data
   ndf.columns = ndf.columns.map('{0[0]}_{0[1]}'.format)

   all_data = pd.concat([all_data, ndf])

   return ndf
```

Here are the original image from dataset on the left and the converted image to use it on our kNN and SVM models on the right:

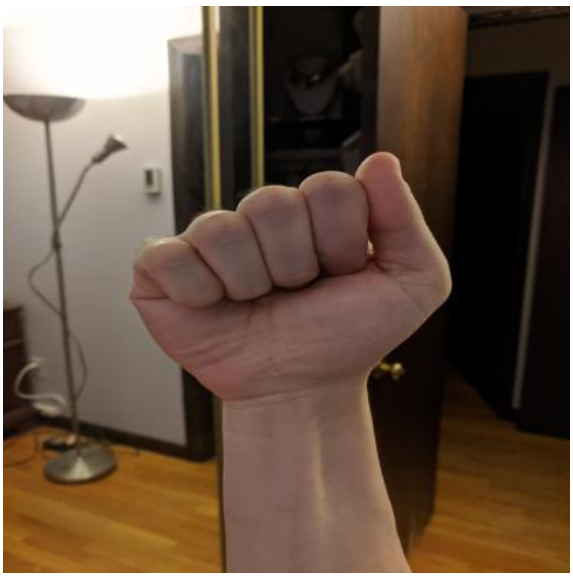

Fig. 5: Example image before preprocessing

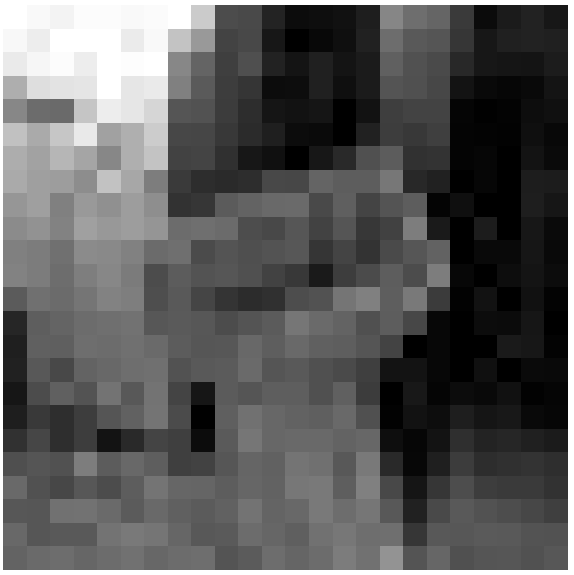

Fig. 6: Example image after preprocessing

3.2 k-NN

For the kNN model we used the KNeighboursClassifier class from the sklearn.neighbors library. It takes the k value (neighbor count) as input as it is shown below:

```
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier(n_neighbors=165)
classifier = KNN.fit(x_train,y_train)
```

x_train is our data frame which is converted from a csv file that we identified as our training set, and y_train includes the labels of images in this training set. After we tried for different k values, we obtained these results:

For k = 100 => Accuracy = %5.2
For k = 75 => Accuracy = %3.8
For k = 50 => Accuracy = %5.2
For k = 5 => Accuracy = %15.54
For k = 4 => Accuracy = %12.5

We obtained the maximum accuracy rate of the k-NN model with k=5 value.

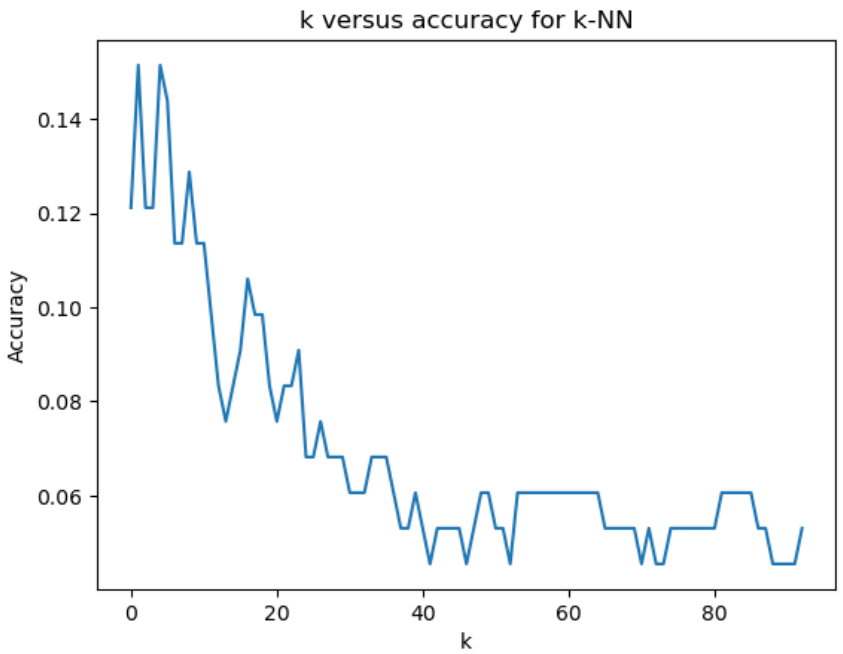

Fig. 7: Graph of Accuracy vs. k values of k-NN model

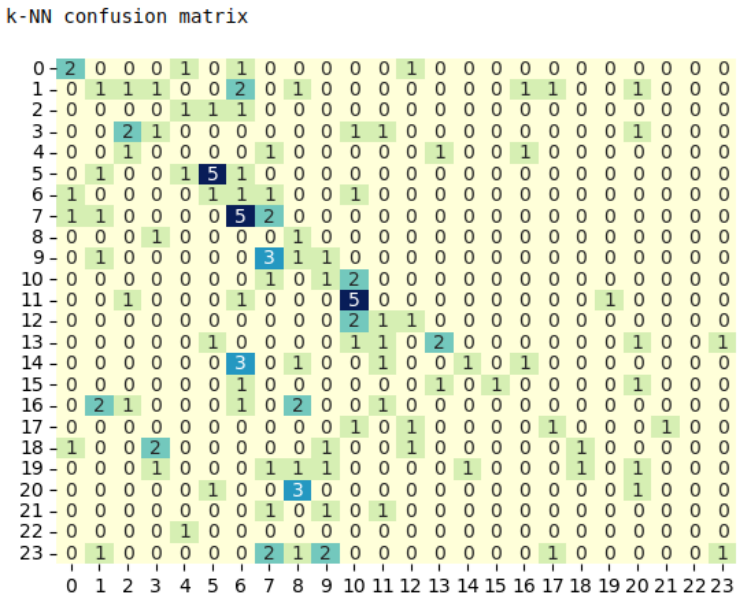

Fig. 8: Confusion Matrix of k-NN model

These results showed us that our dataset was very complicated for the k-NN model which is considered to be a lazy learner. The dataset consisted of different images with varying backgrounds. Since the locations of the objects of interest were indefinite, the algorithm could not perform well with the given 24x24 grayscale images. It was observed that, the more we reduced the neighbor count to train the kNN model, the more accuracy we got. This situation is not ideal since the optimal value for neighbors in the k-NN model should not be too small, in our case it is. Due to these reasons, obtaining the above low accuracy rates was expected.

3.3 SVM

For the SVM model we used the SVC class from the sklearn.svm library. Here is how we used this library to train our dataset on SVM model:

```
from sklearn.svm import SVC
classifier = SVC(decision_function_shape='ovr')
classifier.fit(x_train, y_train)
```

x_train is our data frame which is converted from a csv file that we identified as our training set, and y_train includes the labels of images in this training set.

The results of the training and test is as follows:

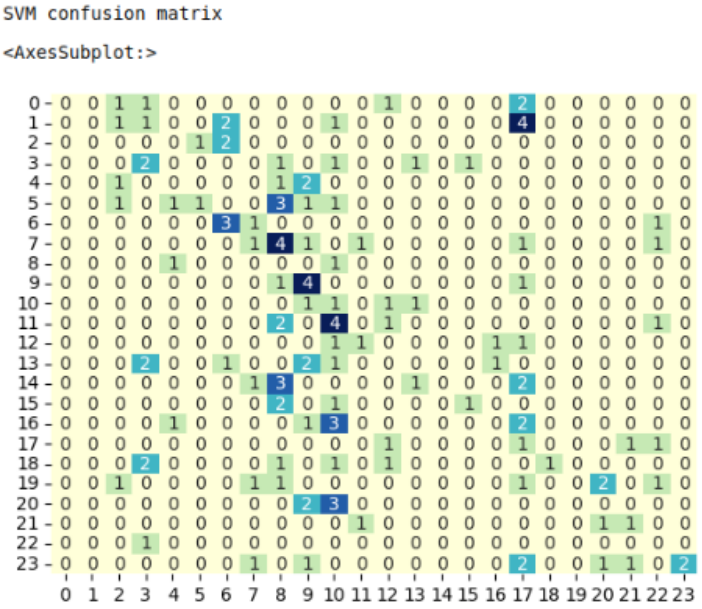

Fig. 9: Confusion Matrix of SVM model

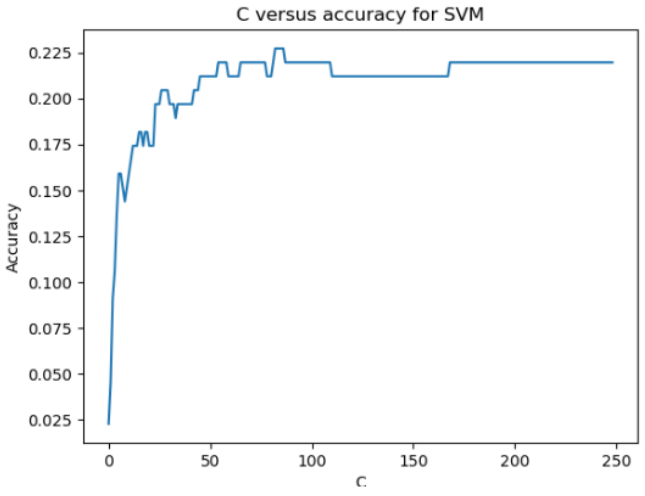

Fig. 10: C versus accuracy graph for SVM model

The C value determines the penalty parameter for the error term. For high values of C, the model will be fitting better for the training data, and for relatively lower values of it, it will be more tolerant to fault, so more generalization with simpler weights can be achieved. In our case, the C value of 74 yielded the best accuracy with the test data, which shows that making the C values greater does not correspond to better generalizations after a certain point.

The maximum accuracy achieved with SVM is 22.54%. SVM has performed better than k-NN in the case of our dataset.

3.4 CNN

To build our convolutional neural network model, we used the nn library of pytorch. We used following parameters to build our model:

- Convolutional layers: 3
- Activation functions:
    - Hidden Layers: 3 ReLu functions
- Pooling layer: MaxPooling2D((2,2))
- Loss function: CrossEntropyLoss function

Accuracy of the model has been calculated as 31.06%.

Here is the accuracy of the model according to training set vs epochs:

Fig 11: Training with CNN

As it can be seen from the graph, after the 15th epoch it is unnecessary to train the model further since it reaches its max potential.

And here is the confusion matrix of the model with the test set:

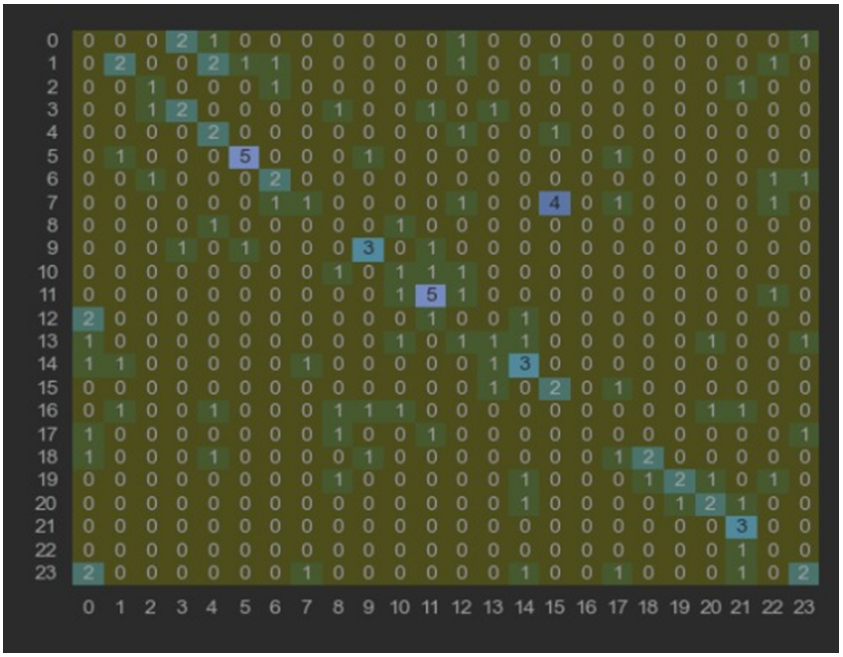

Fig 12: Confusion matrix of the classification with CNN model

This gives us 31.06 % accuracy on a 24 label classification system.

4. Results

The purpose of this project was to investigate the effectiveness of using machine learning algorithms for the task of detecting and classifying American Sign Language (ASL) gestures. To develop the ASL detection model, we first collected a dataset of ASL gestures for the 24 letters of the alphabet, excluding J and Z. This dataset consisted of images of individuals signing each letter in ASL. We then used this dataset to train and test three machine learning algorithms: convolutional neural networks (CNN), k-nearest neighbors (K-NN), and support vector machines (SVM).

After performing Data Preprocessing, we trained CNN, SVM and K-NN models with our dataset and we compared their performance. The Accuracy and Loss Function metrics helped us compare. We determined the best model according to maximum accuracy and working performance.

4.1 Best Model

The results of our experiments showed that the CNN model had the highest accuracy rate which is 31.06%. One reason why CNNs may be the best-performing algorithm for the task of detecting and classifying ASL gestures is due to their ability to automatically extract features from images. CNNs are able to learn and identify important features in the input data, such as edges, shapes, and patterns, without the need for manual feature engineering. This can make them more effective at learning complex relationships in the data and achieving high accuracy rates.

Another reason why CNNs may be the best-performing algorithm is due to their ability to handle large amounts of data and complex models. CNNs are able to process large volumes of data quickly and efficiently, making them well-suited for tasks such as image classification. They are also able to handle complex models with many layers and a large number of parameters, allowing them to capture more subtle patterns and relationships in the data.

Overall, the combination of these characteristics makes CNNs a powerful and effective tool for image recognition tasks such as detecting and classifying ASL gestures.

4.2 Worst Model

The results of our experiments showed that the K-NN model had the lowest accuracy rate which is 15.54%. One reason why K-NN may be the worst-performing algorithm for the task of detecting and classifying ASL gestures is due to its reliance on the entire training set for making predictions. In order to classify a new data point, the K-NN algorithm must compare it to all of the points in the training set, which can be computationally expensive and slow. This can make K-NN less efficient than other algorithms that are able to learn from the training data and make predictions more quickly.

Another reason why K-NN may be the worst-performing algorithm is due to its sensitivity to the choice of K. The value of K can greatly impact the accuracy of the model, and finding the optimal value can be challenging. If K is too small, the model may be too sensitive to noise and outliers in the data, leading to poor performance. If K is too large, the model may be too smooth and not capture enough of the underlying structure of the data.

Overall, while K-NN is a simple and intuitive algorithm, it may not be well-suited for tasks such as detecting and classifying ASL gestures that require high accuracy and efficiency. Other algorithms, such as CNNs or support vector machines (SVMs), may be more effective for this task.

In conclusion, our machine learning models showed promise for the task of detecting and classifying ASL gestures for the 24 letters of the alphabet. The CNN model performed the best, followed by the SVM model and then the K-NN model. Further research and development is needed to improve the accuracy of these models and make them more widely available to the deaf community. This may involve collecting a larger and more diverse dataset, or exploring different machine learning algorithms and approaches.

5. Contributions

Emre Tarakçı: Data preprocessing, report writing, literature review, dataset research, participation in coding of k-NN, SVM and CNN models, training, plot creation using seaborn.

Osman Serhat Yılmaz: Data preprocessing, report writing, coding of SVM model, literature review, coding of CNN model and choosing the suitable CNN model. Training and testing the model and taking results and plotting them.

Ayberk Güven: Coding of k-NN, SVM and CNN models. Took part in the report-writing process, did a part of the literature review, training, coding of SVM and CNN models.

Bertan Köfön:. Dataset research, checking the coding model, took part in the report-writing process and contributed to the literature review, presentation preparation, training of the CNN model.

6. References

[1] “What is sign language?,” *What is sign language?* | *AccessComputing*. [Online]. Available: https://www.washington.edu/accesscomputing/what-sign-language. [Accessed: 01-Dec-2022].

[2] “American sign language,” *National Institute of Deafness and Other Communication Disorders*. [Online]. Available: https://www.nidcd.nih.gov/health/american-sign-language. [Accessed: 01-Dec-2022].

[3] I. A. Adeyanju, O. O. Bello, and M. A. Adegboye, “Machine learning methods for sign language recognition: A critical review and analysis,” *Intelligent Systems with Applications*, vol. 12, p. 200056, 2021.

[4] “What is the K-nearest neighbors algorithm?,” *IBM*. [Online]. Available: https://www.ibm.com/topics/knn. [Accessed: 01-Dec-2022].

[5] “A complete guide to K-nearest-neighbors with applications in python and R,” *Kevin Zakka's Blog*, 13-Jul-2016. [Online]. Available: https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/. [Accessed: 02-Dec-2022].

[6] “Support Vector Machine (SVM) algorithm - javatpoint,” *www.javatpoint.com*. [Online]. Available: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm. [Accessed: 03-Dec-2022].