Hand sign interpretation using ML

Sebastian Bengtsson
DVAMI20H
Karlskrona, Sverige
sebe20@student.bth.se

Marius Stokkedal
DVAMI20H
Karlskrona, Sverige
matk20@student.bth.se

I. Introduction

This report handles hand sign interpreting using machine learning and a computer's webcam. It allows individuals who are deaf or hard of hearing to communicate with individuals who are able to hear. It involves the use of a computer's webcam to capture hand gestures made by an individual using sign language, and a CNN (Convolutional Neural Network), SVM (Support vector machine) and RF (Random Forrest) model to translate these gestures into written letters. This technology has the potential to greatly improve the ability of deaf or hard of hearing individuals to communicate with the wider world and to participate fully in society. It is an example of the ways in which machine learning and artificial intelligence can be used to improve the lives of people with disabilities.

II. Method

At first a fitting data set had to be found and for this the website Kaggle was used. [1] When using images for training a model one must choose between a small usage of memory and good quality. The data set that was chosen was so because it includes over 70 000 images with label and every image 28x28 pixels, i.e., images with lower quality to have a relatively small data set in regard to memory usage.

Since the team member's experience with image recognition is limited three different algorithms were chosen, both to test their pros and cons, and to get wider experience working with different algorithms. The tree algorithms which were chosen are: CNN, SVM and RF. SVM had been used by the team previously, RF was a built-upon of the familiar Decision Tree algorithm and CNN is a unfamiliar algorithm for both team members but it is a famous image classification algorithm.

A. Convolutional Neural Network

Convolutional Neural Network is a type of neural network that works by processing data through multiple layers, each of which extracts increasingly complex features from the input data. The layers include the input layer, the convolutional layers which analyze the input data and detects patterns or features within it, the pooling layers which reduce the spatial dimension of the

data to help reduce the computational cost of the network and prevent overfitting, and the fully connected layers, which are used to make predictions based on the features from the previous layers.

To implement the CNN, we used the available functionality of the TensorFlow library, in conjunction with Keras.

To improve the functionality of the CNN and reduce overfitting we tried to implement data augmentation, meaning that we extend the training data by manipulating the exciting data in different ways. For example, with one photo we could rotate it, zoom in, shift the image horizontally and vertically, change the shear range and do random channel shifts. This could turn what originally was only one training image into multiple ones, while at the same time reducing the chance of overfitting.

B. Support vector machine

Support Vector Machine (SVM) tries to find the best separating line (or hyperplane) between the datapoints of different classes. It does this by finding the highest margin by the datapoints and the separator i.e., the support vectors. If it can't find a good separator it can add another dimension so the data can be linearly separable. It can then classify new datapoints by seeing on which side the new datapoint ends up. [2]

There are a few parameters that were used, first is the C-value which was set to 0.07 after some trial. The C-value is a sort of penalty of the error, i.e., the cost of misclassification. Higher C value means potentially higher accuracy but also higher risk of overfitting. The kernel was set to linear which means that as mentioned the data can be transformed into a higher dimension until it can be linearly separable.

To implement the SVM algorithm we used the available functionality of the scikit-learn library.

C. Random Forest

Random Forest builds upon the Decision Tree algorithm. It creates a multitude of trees and then select the most frequent output of the trees as its prediction. This can be advantageous since when training the decision tree there is a chance it has a low accuracy due to the randomness when splitting the data in to training and test while creating a tree. A single tree is also prone to overfitting. By averaging the predictions of many trees, random forests reduce the variance of the predictions and improve the overall accuracy of the model. The randomization used in the process of building a random forest also helps in reducing overfitting as it makes the model less sensitive to the noise in the data.

For our parameters we used 100 trees, we tried both less and more, trying to balance a good accuracy without overfitting due to too many trees.

To implement the Radom Forest algorithm, we yet again used the available functionality of the scikit-learn library.

D. User interface and program

As the final program was intended to be able be run in a live-action scenario, we needed for the algorithms to make predictions on the fly, and from the live user input.

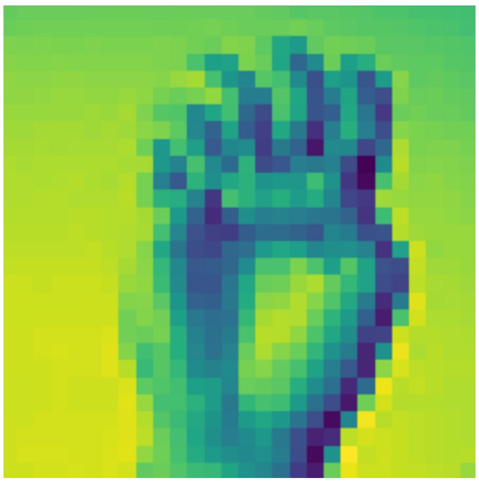
The program uses the open-cv library to access the camera of the user. The interface shows the live camera, the predicted alphabet guesses formed as a text, and a box where the user can place their hand to make the hand signs. With a click of a button, specifically the ‘c’ key, the code will take a photo of the current sign and do preprocessing of the image to convert it in to the needed 28x28 pixel format with the correct color scaling. It will then send the processed image to the different models and retrieve the result that was the most common among the models and display the letter on the interface. The user can then add more letters with the same process or clear the text for a new string to be made. The program can be closed by pressing ‘q’.

III. Results and Analysis

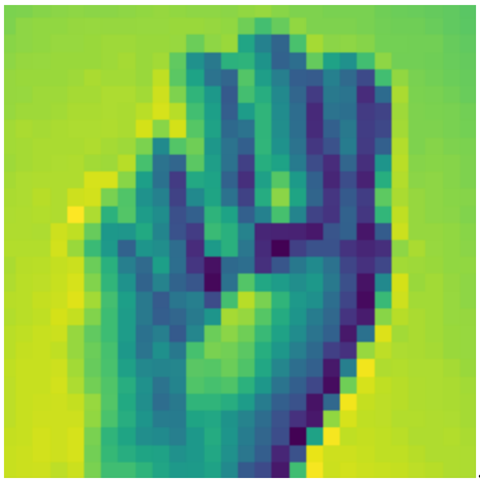
When the models were trained and then later tested, they all performed very well on the test data; all the algorithms got over 90% accuracy and the CNN even got 100% accuracy. When the algorithms then were used on new pictures taken by us the accuracy went down significantly. Tests were carried out in different lightings, with different backgrounds, with different hands, etc. but the result was still far from the same as with the test data. The models were in other words very overfitted to the test and training data. To avoid overfitting data augmentation was used which gave a bit better result, but still nothing close to the test data.

The bad result was not a big surprise since the pictures used had such low quality and some letters have a very close resemblance to each other.

For example: this is the letter *E*



And this is the letter *M*

.

The difference between these letters is that *E* has the thumb in front of the fingers and *M* has the thumb behind the index, long and ring finger. The combination of many letters looking alike and the low quality of the images it is hard to train a model with a good result. The different models often give different results for the same image and the same model often predicts a different letter if the user signs the same letter but changes the orientation of the hand a little bit.

The data set which was used to make the models is 83.3 MB of training data and 21.8 MB of test data. If we had opted for better quality of the images and thereby also the models both the data size and the size of the models would increase significantly.

IV. Conclusions

The idea of a computer being able to determine which letter a person is signing can be great opportunity for a person who is deaf. It can help with their communication because sign language is not widely understood outside of the deaf community. For them to spell words is a start to them being able to sign whole words which will be translated without the need for a human translator.

The result would probably have benefitted from using a data set of higher quality, but, as said earlier, this would have resulted in more memory usage and a longer training time of the models.

More or better data augmentation would probably have helped somewhat with the implementation but judging by the result of the earlier tries with this made the team think that it would be a waste of time compared to the gained result.

If this project would be redone a better data set should be used for the training and the work that has been done can be used as a ground floor to start from so we do not have to start from scratch.

V. Contribution

A. Marius

Marius implemented the CNN and RF models, as well as contributed to the report by writing the sections on results and analysis, and the introduction to the method.

The CNN model required a lot of time to complete, both because it took about 20 minutes to train each time and it was a whole new concept. The model is saved when trained to save time for the user, but it does obviously need to be retrained if some metrics are changed. Before the work could start with the CNN the python version had to be downgraded since the modules needed did not support python 3.11.

The CNN model also had problems with overfitting of the model. Trying to avoid this data augmentation was tested with a bit better result. This was a very time-consuming moment since, as mentioned earlier, the model took about 20 minutes to retrain each time.

RF was still an unfamiliar concept but easier to grasp. We have used a discission tree earlier on in the course which made the work on random forest much easier, since a random forest is, as the name suggests, multiple discission trees. Playing around with the parameters to find the best result required the most time, especially since this model required around 70 seconds to train.

B. Sebastian

Sebastian implemented the SVM model, wrote the code for the camera and interface implementation and contributed to the report by writing the introduction, the section on the algorithms used and the user interface and program.

Getting a somewhat natural way of capturing the user's hand was an issue where various solutions were tried, but the most easy-to-understand and fairly easy to implement way was the final product with the displayed box where users can do their signing in.

Cross-platform support for the program was also important as the team used different programs so a requirement was to be able to use either Mac, Linux or Windows.

As work had been done with the SVM model earlier in the course, it was a natural choice and a model that felt familiar to work with. The core implementation went smoothly, and more focus could be put in to fine-tuning the parameters.

C. Together

However, despite the splitting up works, we discussed a lot and often worked together on problem solving and details, read each other work and gave feedback so all in all it was an equal amount of work put in and done. The data set was discovered when both of us searched Kaggle.com for a fitting one and a lot of work was put down to determining if the data was of any use. When a data set of good quality was found it was downloaded and we started to look for any kind of problems or oddities in the data, both by using pandas and Excel but no problems were discovered. After this the coding started and the team did split the work, but still had a good communication throughout the whole project to ensure both parties liked the new features and understood what happened so everyone would gain new and important experiences working with these algorithms, especially CNN.

VI. References

[1] A. Singh, "Hand Sign Images," [Online]. Available: https://www.kaggle.com/datasets/ash2703/handsignimages.

[2] S. Bengtsson and M. Stokkedal, "DV2599: Assignment 2," Karlskrona, 2022.