Elective 4 | Deep Learning

DS4A

American Sign Language Image Classification using Convolutional Neural Network

Borromeo, Timothy Jonah

Candel, Sheena

Daug, Nicole Mikka

Llamis, Charlyn Jean

Ramas, Juamalida

Senepete, Jayve

Villanueva, Cherithy Claire

Chapter 1

Introduction

Background of the study

According to the National Institute on Deafness and Other Communication Disorders, the 200-year-old American Sign Language is a comprehensive, sophisticated language (of which letter gestures constitute only a minor part), but it remains the primary language for many deaf North Americans (NIDCD). Building a system that recognizes sign language will allow the deaf and hard of hearing to communicate more effectively with modern technologies.

According (M. Mohandes, et al) More than 5% of the world's population is affected by hearing impairment. To overcome the challenges faced by these individuals, various sign languages have been developed as an easy and efficient means of communication. Sign language depends on signs and gestures which give meaning to something during communication.

Image classification has a great help to society, especially to people with disabilities. With the various assistants a technology can offer, one of which is being a medium to communicate with the mute and deaf people. The American sign language is a sign language used by many people. The family of the mute and deaf people may know how to communicate with them hence some where accompanied when they go outside. Classifying the sign language allows them to easily communicate with other people other than their family and to some people who know sign language. In this study, the researchers aim to develop a model that classifies American sign language with different images. Using Convolution Neural Network, a dataset of different images will be fed to the model for training and testing. Evaluation metrics will be used to determine the performance of the model.

There are two types of sign language recognition methods namely sensor-based and image-based. The first method is dependent on localized sensors or wearing specific gloves. The main benefit of this method is its ability to provide accurate information about signs or gestures such as the movement, rotation, orientation as well as positioning of the hands. The second method uses different types of cameras. It is based on image processing which does not require equipment such as sensors. This approach only relies on the application of various image processing techniques and pattern recognition.

Sign language differs among countries. In addition, various sign languages generally contain non-manual signs such as body gestures and facial expressions. They often require two hands or sequential movements for performing these signs. These issues increase the complexity in design of sign language recognition systems. To overcome this, researchers showed a specific interest in the sign language recognition systems

In recent years, researchers have employed deep learning for sign language recognition systems. This involves the use of various methodologies and datasets which aim to improve the accuracy of the system. Various datasets are created because of many factors such as regional differences, type of images (RGB or Depth) and so on. Sign language differs from one region to another just like spoken languages and includes American sign language, Indian sign language, Arabic sign language, etc. Moreover, the type of images used for recognition systems depends on the camera that generates or depth images. Furthermore, the methodologies used by different researchers that underlie the core of gesture recognition systems, vary from one system to another. Each study works to create and improve a different system to enhance its accuracy. Currently, there is no system which can deal with all conditions with high accuracy. Researchers have previously focused on CNNs with different parameters for sign language recognition systems due to their high performance in image classification. Also, some studies use CNNs along with other methods to obtain more accurate results. However, others have used methods such as SVM and PCANET. Comparisons of CNNs with other methods have proved the superiority and ability of CNNs.

Objectives

- To develop a model using Convolutional Neural Network that will classify what letter in the English Alphabet is associated with the respective Sign Language.
- To contribute knowledge for the development of Sign language recognition systems.
- To discuss the results obtained using (ResNet) Residual Neural Network.
- To provide an efficient and accurate way to convert sign language into text or voice has aids for the hearing impaired.

Scope and Limitations

In order to communicate with a deaf individual, you will either require a translator or sign language proficiency. However, not everyone has access to this knowledge, particularly deaf people who typically attend their own school with qualified teachers. With assistance for the hearing challenged, a model that categorizes American sign language using various visuals in text or voice is being developed.

In writing our code for our project, we face a lot of challenges in our datasets and models. One is that we have to run out of epochs, which is time-consuming since we don't have a graphics processing unit or GPUs; one epoch takes 1–2 hours to run, and we think it will take a month to run all the epochs if we continue to do so. We want to use Google Colab since it has a free GPU, but the problem is that the model will only be saved in the Google Colab system, not in our file. That's why we used Jupyter Notebook instead.

Chapter 2

Related Works/Related Projects

Anantha Rao et al. (2018) proposes a CNN for the recognition of the gestures of the Indian sign language. They used a continuous capture method from smartphone cameras in selfie mode and built a mobile application. As a result, they obtained 92.88% accuracy in comparison to other classifier models reported on the same dataset.

R. Daroya et al. (2018) reported a method which is CNN inspired called Densely Connected Convolutional Neural Networks (DenseNet) to classify RGB (Red_Green_Blue) images of static letter hand gestures in Sign Language. A proposed deep network is useful for sign language classification tasks and has an accuracy of 90.3%. In addition, they use both depth images and RGB images.

V. Jain et al. created a system to recognize ASL by using both Support Vector Machine (SVM) and Convolutional Neural Network (CNN). V. Jain et al. (2021) created a system to recognize ASL by using both Support Vector Machine (SVM) and Convolutional Neural Network (CNN). They found that the single layer CNN yielded 97.344% accuracy and 98.581% accuracy was reported for a two-layer CNN.

P. Kurhekar et al. (2019) present a system that can learn signs from videos by processing the video frames under a minimum disturbed background. After that, the obtained sign is converted to readable text. This system utilizes a CNN which depends on a ResNet-34 CNN classifier, Fast.ai and OpenCV libraries for webcam inputs and displaying the predicted signs. They obtained a model with an accuracy of 78.5% on the presented testing set.

K. Bantupalli and Y. Xie (2019) implemented a vision-based application that provides sign language translation into text. They aimed to aid people with hearing disabilities to communicate with hearing individuals. In this system, the utilized dataset is the American Sign Language dataset, and the accuracy is 91% for 150 signs. Researchers report a sign language fingerspelling alphabet identification system. HOG and LBP features of each gesture are extracted from training images. After that, they train these extracted data by multi-class SVMs. Their results showed that the improved model had 82% precision and 80% recall.

S. Alyl. et al. (2017) designed a system for alphabetic Arabic sign language recognition by using intensity and depth images, which were obtained from a SOFTKINECTM sensor. They utilized a PCANet method to learn local features from intensity and depth images and a linear support vector machine classifier to recognize the extracted features. The performance of the suggested system was assessed on a dataset of real images captured from multi-users. They implemented the tests in two ways; the first one utilized intensity and depth images separately, the second one utilized a combination of intensity and depth images. The acquired results revealed that the second test produced an accuracy of 99.5%.

Authors have developed several theories and carried out various original studies with various objectives. In Anantha Rao et al. (2018), the author suggests a CNN for the recognition of the motions of the Indian sign language. where they created a mobile application and used a continuous recording technique from selfie-mode smartphone cameras. In this study, they used both Support Vector Machine (SVM) and Convolutional Neural Network (CNN), which was proposed by V. Jain et al., to achieve 97.344% accuracy, compared to other classifier models such as ASL, which achieved 92.88% accuracy.

In several studies, for example, done in a similar way to this one, passing accuracy was attained in each model by using CNN and other models. Y. and Bantupalli. A vision-based application that converts sign language into text was put into place by Xie (2019). In this study, which aimed to help those who have hearing impairments converse with hearing persons, the author received 91% for 150 signals, while S. Alyl. et al. (2017) used intensity and depth pictures from a SOFTKINECTM sensor to create a system for alphabetic Arabic sign language recognition.

Chapter 3
Methodology

This chapter presents the methodology used for performing and analyzing the study. It explains the Dataset, Deep learning model CNN and ResNet. Additionally, it provides more details on a model for classifying the sign languages' alphabets.

Dataset

The dataset used by the researchers was obtained from Kaggle. It contains two folders namely asl_alphabet_test and asl_alphabet_train. Each folder contains images labeled by its respective english letter. It also includes images for “Space”, “del”, and “noting”. Inside the asl_alphabet_train folder contains 3000 images of each letter, including the three labels. Inside the asl_alphabet_test have 28 images. The researchers then created a function to randomly delete images per subfolder/letter to lessen the dataset. Giving the researchers 11,165 datapoints in total for training.

The dataset format is designed to be similar to the classic MNIST. Each training and test case represents a label (0-25) as a one-to-one map for each alphabetic letter A-Z (However, due to gesture movements, there are no cases for 9=J or 25=Z.) The training data (78000 cases) and test data (11165 cases) *since we use a function that randomly deleted the image per folder hence the datasets become 11,165*. Have a header row of label, pixel1, pixel2,...pixel784 that represents a single 28x28 pixel picture with grayscale values ranging from 0-255 but are nearly half as big as the normal MNIST. Multiple people doing the same move against various backgrounds were depicted in the original image data for hand gestures. The limited (1704) number of color photos used in the study that weren't cropped around the hand region of focus provided the basis for the Sign Language MNIST data.

Deep Learning Model

Computer Vision has many interesting applications ranging from industrial applications to social applications. It has also been applied in support for physically challenged people. For deaf-mute people, computer vision can generate English alphabets based on the sign language symbols. It can recognize the hand symbols and predict the correct corresponding alphabet through sign language classification.

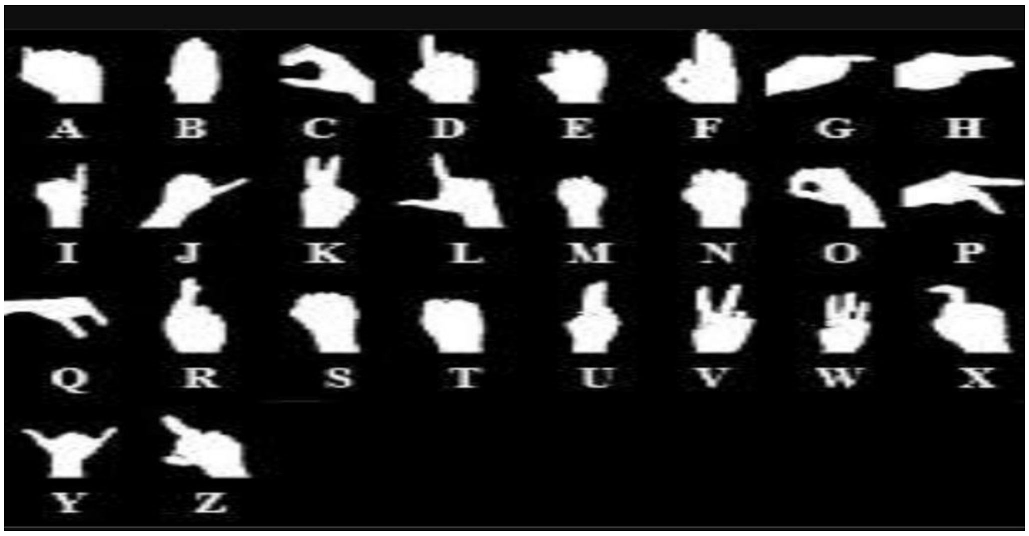


Figure 1. Threshold hand poses of ASLA.

The researchers will classify the sign language symbols using the Convolutional Neural Network (CNN). After successful training of the CNN model, the corresponding alphabet of a sign language symbol will be predicted. We will evaluate the classification performance of our model using the non-normalized and normalized confusion matrices. Finally, we will obtain the classification accuracy score of the CNN model.

CNN

Convolutional Neural Networks, also known as CNNs, are a subset of artificial neural networks used in deep learning and are frequently employed for object and picture recognition and categorization. Thus, Deep Learning uses a CNN to identify items in a picture. CNNs are being used extensively in a variety of tasks and activities, including speech recognition in natural language processing, video analysis, and difficulties with image processing, computer vision, and self-driving car obstacle detection. Due to their major contribution to these rapidly developing and expanding fields, CNNs are widely used in deep learning.

ResNet

A ResNet is a type of convolutional neural network, it was able to achieve very high accuracy in image classification tasks, such as the ImageNet dataset, by addressing the problem of vanishing gradients in very deep networks. ResNet achieves this by introducing skip connections, or shortcuts, between the layers of the network, which allow the gradients to flow more easily through the network during training. This breakthrough in deep learning has led to the development of even deeper architectures such as DenseNet, and many ResNet variants are still widely used in computer vision tasks.

ResNet50

A ResNet50 is a deeper version of the original ResNet architecture, containing 50 layers, and was able to achieve even higher accuracy in image classification tasks, such as the ImageNet dataset. Additionally, ResNet50 is widely used as a feature extractor in computer vision tasks. It is trained on a large dataset and the learned features can be used to train smaller datasets. This is called Transfer Learning, in which pre-trained models are fine-tuned on specific dataset. ResNet50 is a popular choice for transfer learning due to its high accuracy and the fact that it is relatively lightweight, making it easy to use in a variety of applications.

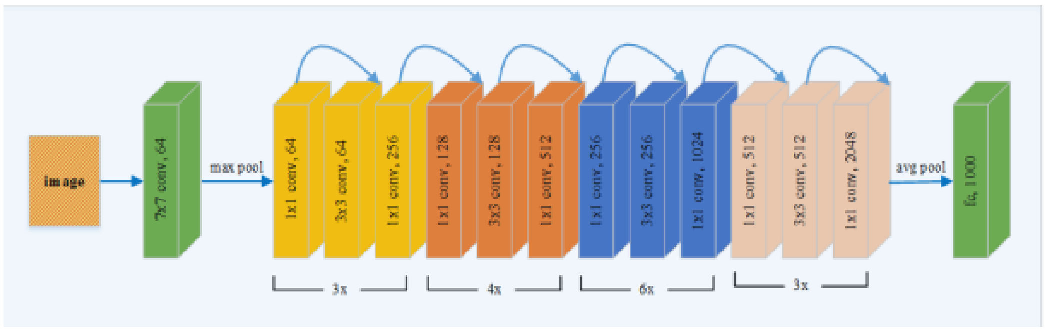


Figure 2. A block diagram representation of pre-trained Resnet-50 architecture.

The researchers will be using the American Sign Language (ASL) dataset that is publicly available at Kaggle. This dataset contains 78000 training images and 11,165 test images all with a shape of 28 x 28 pixels. These images belong to the 25 classes of English alphabet starting from A to Y (No class labels for Z because of gesture motions). The dataset on Kaggle is available in the CSV format where training data has 27455 rows and 785 columns. The first column of the dataset represents the class label of the image and the remaining 784 columns represent the 28 x 28 pixels. The same paradigm is followed by the test data set.

The title or topic of the project that the researchers choose is not really popular with the programming students. The topic is really different from our course, but we think rare topics for projects are challenging enough for us to learn new things. The researchers think that making a model to classify the alphabets in sign languages will make a huge contribution to easier communication with special children or people.

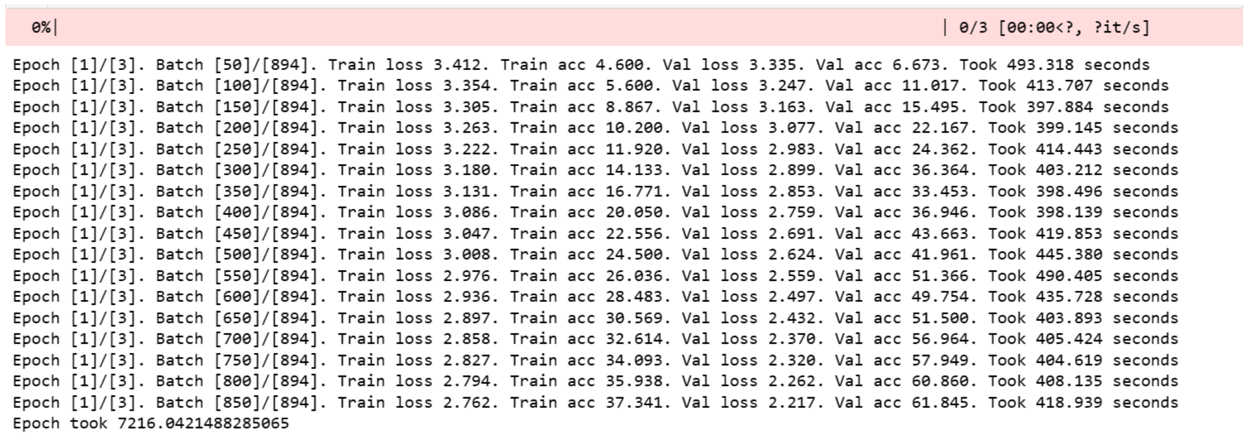
Chapter 4

Results and Discussion

This chapter discusses the findings, analysis and interpretation of the model’s performance. The deep learning model used in the study is ResNet50 which is widely used in computer vision tasks such as image classification, object detection, and semantic segmentation.

The Residual Network (ResNet) is considered the best in deep learning networks, for the following reasons; residual solved the overfitting problem and the vanishing gradient problem. The basic idea of the residual network is to add something called an “identity shortcut connection” that bypasses one or more layers, making it easier to train. The number of images used in data modeling is big enough and took awhile to see the results. Using the test size of 0.2, the number of data for training is 7572 while the validation data is 1893. Using a batch size of 10 and epochs of 3 the results are as follows.

First Epoch

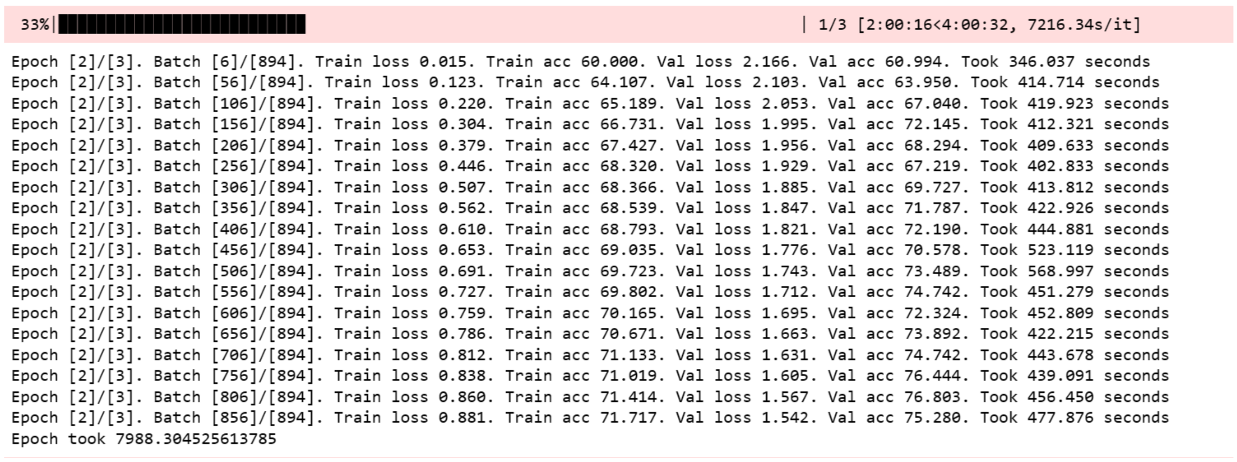


```
  0%|                                                                              | 0/3 [00:00<?, ?it/s]
Epoch [1]/[3]. Batch [50]/[894]. Train loss 3.412. Train acc 4.600. Val loss 3.335. Val acc 6.673. Took 493.318 seconds
Epoch [1]/[3]. Batch [100]/[894]. Train loss 3.354. Train acc 5.600. Val loss 3.247. Val acc 11.017. Took 413.707 seconds
Epoch [1]/[3]. Batch [150]/[894]. Train loss 3.305. Train acc 8.867. Val loss 3.163. Val acc 15.495. Took 397.884 seconds
Epoch [1]/[3]. Batch [200]/[894]. Train loss 3.263. Train acc 10.200. Val loss 3.077. Val acc 22.167. Took 399.145 seconds
Epoch [1]/[3]. Batch [250]/[894]. Train loss 3.222. Train acc 11.920. Val loss 2.983. Val acc 24.362. Took 414.443 seconds
Epoch [1]/[3]. Batch [300]/[894]. Train loss 3.180. Train acc 14.133. Val loss 2.899. Val acc 36.364. Took 403.212 seconds
Epoch [1]/[3]. Batch [350]/[894]. Train loss 3.131. Train acc 16.771. Val loss 2.853. Val acc 33.453. Took 398.496 seconds
Epoch [1]/[3]. Batch [400]/[894]. Train loss 3.086. Train acc 20.050. Val loss 2.759. Val acc 36.946. Took 398.139 seconds
Epoch [1]/[3]. Batch [450]/[894]. Train loss 3.047. Train acc 22.556. Val loss 2.691. Val acc 43.663. Took 419.853 seconds
Epoch [1]/[3]. Batch [500]/[894]. Train loss 3.008. Train acc 24.500. Val loss 2.624. Val acc 41.961. Took 445.380 seconds
Epoch [1]/[3]. Batch [550]/[894]. Train loss 2.976. Train acc 26.036. Val loss 2.559. Val acc 51.366. Took 490.405 seconds
Epoch [1]/[3]. Batch [600]/[894]. Train loss 2.936. Train acc 28.483. Val loss 2.497. Val acc 49.754. Took 435.728 seconds
Epoch [1]/[3]. Batch [650]/[894]. Train loss 2.897. Train acc 30.569. Val loss 2.432. Val acc 51.500. Took 403.893 seconds
Epoch [1]/[3]. Batch [700]/[894]. Train loss 2.858. Train acc 32.614. Val loss 2.370. Val acc 56.964. Took 405.424 seconds
Epoch [1]/[3]. Batch [750]/[894]. Train loss 2.827. Train acc 34.093. Val loss 2.320. Val acc 57.949. Took 404.619 seconds
Epoch [1]/[3]. Batch [800]/[894]. Train loss 2.794. Train acc 35.938. Val loss 2.262. Val acc 60.860. Took 408.135 seconds
Epoch [1]/[3]. Batch [850]/[894]. Train loss 2.762. Train acc 37.341. Val loss 2.217. Val acc 61.845. Took 418.939 seconds
Epoch took 7216.0421488285065
```

Figure 3. First Epoch

The figure above shows the number of train loss and validation loss during the first epoch in training data. In the first batch of the data the train loss is 3.412 which was decreased as the last batch was trained. The train accuracy was increasing in each batch as well as the validation accuracy. Validation loss was decreasing as well as until the last batch on the first epoch. It took about 2 hours to finish the first epoch.

Second Epoch

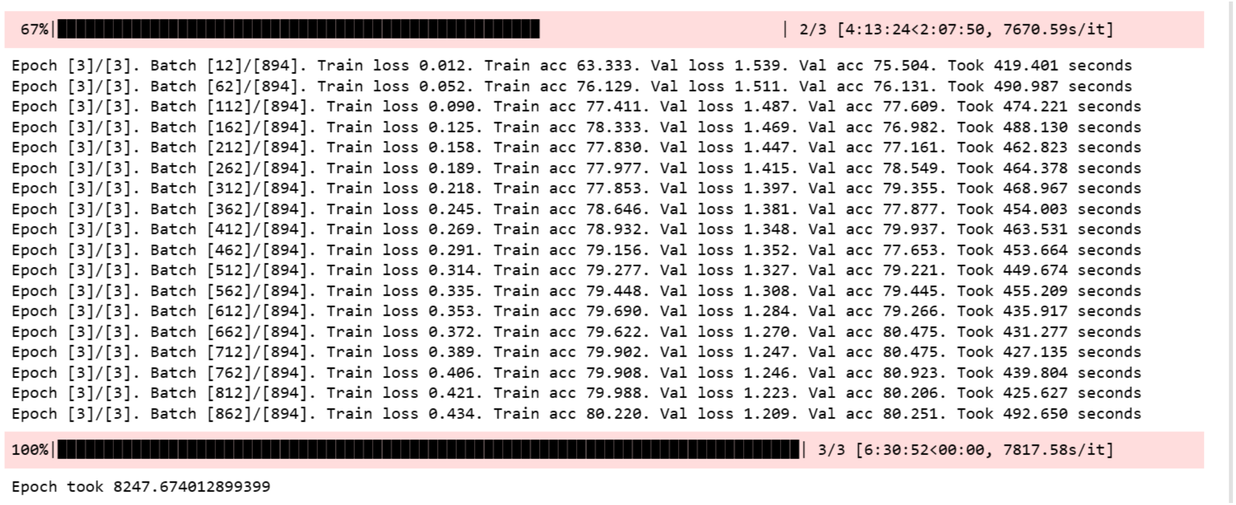


```
33%|█████████████████████████                                             | 1/3 [2:00:16<4:00:32, 7216.34s/it]
Epoch [2]/[3]. Batch [6]/[894]. Train loss 0.015. Train acc 60.000. Val loss 2.166. Val acc 60.994. Took 346.037 seconds
Epoch [2]/[3]. Batch [56]/[894]. Train loss 0.123. Train acc 64.107. Val loss 2.103. Val acc 63.950. Took 414.714 seconds
Epoch [2]/[3]. Batch [106]/[894]. Train loss 0.220. Train acc 65.189. Val loss 2.053. Val acc 67.040. Took 419.923 seconds
Epoch [2]/[3]. Batch [156]/[894]. Train loss 0.304. Train acc 66.731. Val loss 1.995. Val acc 72.145. Took 412.321 seconds
Epoch [2]/[3]. Batch [206]/[894]. Train loss 0.379. Train acc 67.427. Val loss 1.956. Val acc 68.294. Took 409.633 seconds
Epoch [2]/[3]. Batch [256]/[894]. Train loss 0.446. Train acc 68.320. Val loss 1.929. Val acc 67.219. Took 402.833 seconds
Epoch [2]/[3]. Batch [306]/[894]. Train loss 0.507. Train acc 68.366. Val loss 1.885. Val acc 69.727. Took 413.812 seconds
Epoch [2]/[3]. Batch [356]/[894]. Train loss 0.562. Train acc 68.539. Val loss 1.847. Val acc 71.787. Took 422.926 seconds
Epoch [2]/[3]. Batch [406]/[894]. Train loss 0.610. Train acc 68.793. Val loss 1.821. Val acc 72.190. Took 444.881 seconds
Epoch [2]/[3]. Batch [456]/[894]. Train loss 0.653. Train acc 69.035. Val loss 1.776. Val acc 70.578. Took 523.119 seconds
Epoch [2]/[3]. Batch [506]/[894]. Train loss 0.691. Train acc 69.723. Val loss 1.743. Val acc 73.489. Took 568.997 seconds
Epoch [2]/[3]. Batch [556]/[894]. Train loss 0.727. Train acc 69.802. Val loss 1.712. Val acc 74.742. Took 451.279 seconds
Epoch [2]/[3]. Batch [606]/[894]. Train loss 0.759. Train acc 70.165. Val loss 1.695. Val acc 72.324. Took 452.809 seconds
Epoch [2]/[3]. Batch [656]/[894]. Train loss 0.786. Train acc 70.671. Val loss 1.663. Val acc 73.892. Took 422.215 seconds
Epoch [2]/[3]. Batch [706]/[894]. Train loss 0.812. Train acc 71.133. Val loss 1.631. Val acc 74.742. Took 443.678 seconds
Epoch [2]/[3]. Batch [756]/[894]. Train loss 0.838. Train acc 71.019. Val loss 1.605. Val acc 76.444. Took 439.091 seconds
Epoch [2]/[3]. Batch [806]/[894]. Train loss 0.860. Train acc 71.414. Val loss 1.567. Val acc 76.803. Took 456.450 seconds
Epoch [2]/[3]. Batch [856]/[894]. Train loss 0.881. Train acc 71.717. Val loss 1.542. Val acc 75.280. Took 477.876 seconds
Epoch took 7988.304525613785
```

Figure 4. Second epoch

The figure above shows the second epoch of training the data. The train loss started incredibly low and eventually increased on the last batch of the data. With the train accuracy of 60 and validation accuracy 0f 60.99 on the first batch, it is high enough accuracy compared to the first epoch. As the validation loss increases, the training accuracy increases while the validation accuracy changes in every batch . For the second epoch, the validation accuracy of 75.28 is good enough. The epochs also took over 2 hours of run time.

Third Epoch

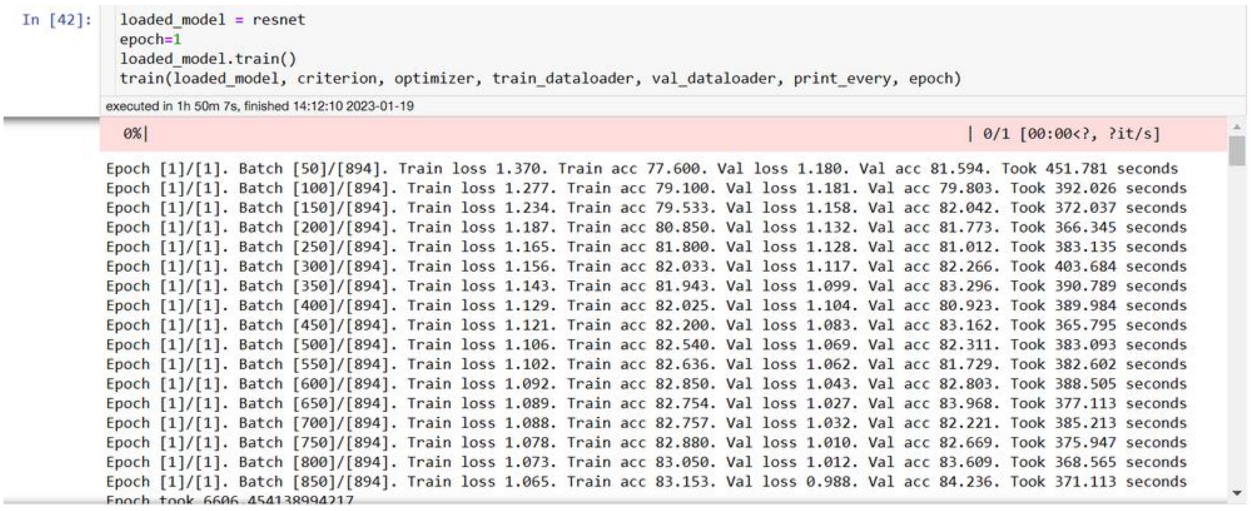


```
 67%|██████████████████████████████████████████████████                    | 2/3 [4:13:24<2:07:50, 7670.59s/it]
Epoch [3]/[3]. Batch [12]/[894]. Train loss 0.012. Train acc 63.333. Val loss 1.539. Val acc 75.504. Took 419.401 seconds
Epoch [3]/[3]. Batch [62]/[894]. Train loss 0.052. Train acc 76.129. Val loss 1.511. Val acc 76.131. Took 490.987 seconds
Epoch [3]/[3]. Batch [112]/[894]. Train loss 0.090. Train acc 77.411. Val loss 1.487. Val acc 77.609. Took 474.221 seconds
Epoch [3]/[3]. Batch [162]/[894]. Train loss 0.125. Train acc 78.333. Val loss 1.469. Val acc 76.982. Took 488.130 seconds
Epoch [3]/[3]. Batch [212]/[894]. Train loss 0.158. Train acc 77.830. Val loss 1.447. Val acc 77.161. Took 462.823 seconds
Epoch [3]/[3]. Batch [262]/[894]. Train loss 0.189. Train acc 77.977. Val loss 1.415. Val acc 78.549. Took 464.378 seconds
Epoch [3]/[3]. Batch [312]/[894]. Train loss 0.218. Train acc 77.853. Val loss 1.397. Val acc 79.355. Took 468.967 seconds
Epoch [3]/[3]. Batch [362]/[894]. Train loss 0.245. Train acc 78.646. Val loss 1.381. Val acc 77.877. Took 454.003 seconds
Epoch [3]/[3]. Batch [412]/[894]. Train loss 0.269. Train acc 78.932. Val loss 1.348. Val acc 79.937. Took 463.531 seconds
Epoch [3]/[3]. Batch [462]/[894]. Train loss 0.291. Train acc 79.156. Val loss 1.352. Val acc 77.653. Took 453.664 seconds
Epoch [3]/[3]. Batch [512]/[894]. Train loss 0.314. Train acc 79.277. Val loss 1.327. Val acc 79.221. Took 449.674 seconds
Epoch [3]/[3]. Batch [562]/[894]. Train loss 0.335. Train acc 79.448. Val loss 1.308. Val acc 79.445. Took 455.209 seconds
Epoch [3]/[3]. Batch [612]/[894]. Train loss 0.353. Train acc 79.690. Val loss 1.284. Val acc 79.266. Took 435.917 seconds
Epoch [3]/[3]. Batch [662]/[894]. Train loss 0.372. Train acc 79.622. Val loss 1.270. Val acc 80.475. Took 431.277 seconds
Epoch [3]/[3]. Batch [712]/[894]. Train loss 0.389. Train acc 79.902. Val loss 1.247. Val acc 80.475. Took 427.135 seconds
Epoch [3]/[3]. Batch [762]/[894]. Train loss 0.406. Train acc 79.908. Val loss 1.246. Val acc 80.923. Took 439.804 seconds
Epoch [3]/[3]. Batch [812]/[894]. Train loss 0.421. Train acc 79.988. Val loss 1.223. Val acc 80.206. Took 425.627 seconds
Epoch [3]/[3]. Batch [862]/[894]. Train loss 0.434. Train acc 80.220. Val loss 1.209. Val acc 80.251. Took 492.650 seconds
100%|████████████████████████████████████████████████████████████████████████████| 3/3 [6:30:52<00:00, 7817.58s/it]
Epoch took 8247.674012899399
```

Figure 5. Third epoch

The image above shows the third epoch data training on the model. The training loss started 0.012 on the first batch of the data and finished 0.434 train loss which resulted in 80.22 training accuracy. The validity loss is decreasing as the validation accuracy changes on every batch of the epochs which gives a result of 80.25 on the last batch of the third epoch.

Fourth Epoch



```
In [42]:  loaded_model = resnet
          epoch=1
          loaded_model.train()
          train(loaded_model, criterion, optimizer, train_dataloader, val_dataloader, print_every, epoch)
```

executed in 1h 50m 7s, finished 14:12:10 2023-01-19

```
  0%|                                                                                | 0/1 [00:00<?, ?it/s]

Epoch [1]/[1]. Batch [50]/[894]. Train loss 1.370. Train acc 77.600. Val loss 1.180. Val acc 81.594. Took 451.781 seconds
Epoch [1]/[1]. Batch [100]/[894]. Train loss 1.277. Train acc 79.100. Val loss 1.181. Val acc 79.803. Took 392.026 seconds
Epoch [1]/[1]. Batch [150]/[894]. Train loss 1.234. Train acc 79.533. Val loss 1.158. Val acc 82.042. Took 372.037 seconds
Epoch [1]/[1]. Batch [200]/[894]. Train loss 1.187. Train acc 80.850. Val loss 1.132. Val acc 81.773. Took 366.345 seconds
Epoch [1]/[1]. Batch [250]/[894]. Train loss 1.165. Train acc 81.800. Val loss 1.128. Val acc 81.012. Took 383.135 seconds
Epoch [1]/[1]. Batch [300]/[894]. Train loss 1.156. Train acc 82.033. Val loss 1.117. Val acc 82.266. Took 403.684 seconds
Epoch [1]/[1]. Batch [350]/[894]. Train loss 1.143. Train acc 81.943. Val loss 1.099. Val acc 83.296. Took 390.789 seconds
Epoch [1]/[1]. Batch [400]/[894]. Train loss 1.129. Train acc 82.025. Val loss 1.104. Val acc 80.923. Took 389.984 seconds
Epoch [1]/[1]. Batch [450]/[894]. Train loss 1.121. Train acc 82.200. Val loss 1.083. Val acc 83.162. Took 365.795 seconds
Epoch [1]/[1]. Batch [500]/[894]. Train loss 1.106. Train acc 82.540. Val loss 1.069. Val acc 82.311. Took 383.093 seconds
Epoch [1]/[1]. Batch [550]/[894]. Train loss 1.102. Train acc 82.636. Val loss 1.062. Val acc 81.729. Took 382.602 seconds
Epoch [1]/[1]. Batch [600]/[894]. Train loss 1.092. Train acc 82.850. Val loss 1.043. Val acc 82.803. Took 388.505 seconds
Epoch [1]/[1]. Batch [650]/[894]. Train loss 1.089. Train acc 82.754. Val loss 1.027. Val acc 83.968. Took 377.113 seconds
Epoch [1]/[1]. Batch [700]/[894]. Train loss 1.088. Train acc 82.757. Val loss 1.032. Val acc 82.221. Took 385.213 seconds
Epoch [1]/[1]. Batch [750]/[894]. Train loss 1.078. Train acc 82.880. Val loss 1.010. Val acc 82.669. Took 375.947 seconds
Epoch [1]/[1]. Batch [800]/[894]. Train loss 1.073. Train acc 83.050. Val loss 1.012. Val acc 83.609. Took 368.565 seconds
Epoch [1]/[1]. Batch [850]/[894]. Train loss 1.065. Train acc 83.153. Val loss 0.988. Val acc 84.236. Took 371.113 seconds
Epoch took 6606.454138994217
```

Figure 6. Fourth epoch

The figure above shows the fourth epoch which was separated from the previous epochs. The number shows only one but the training continued from the last epoch. The train loss started bigger than the previous epochs as well as the validation loss except on batch 400 which suddenly went higher than the previous batch . The training accuracy started 77.66 on the first batch and finished 83.153 on the last batch. The validation accuracy started high enough but isn't consistent on each batch which means that the model is learning. The last batch had a validation score of 84.24 which was the highest to all validation accuracy.

Train and Validation Losses with 3 epochs

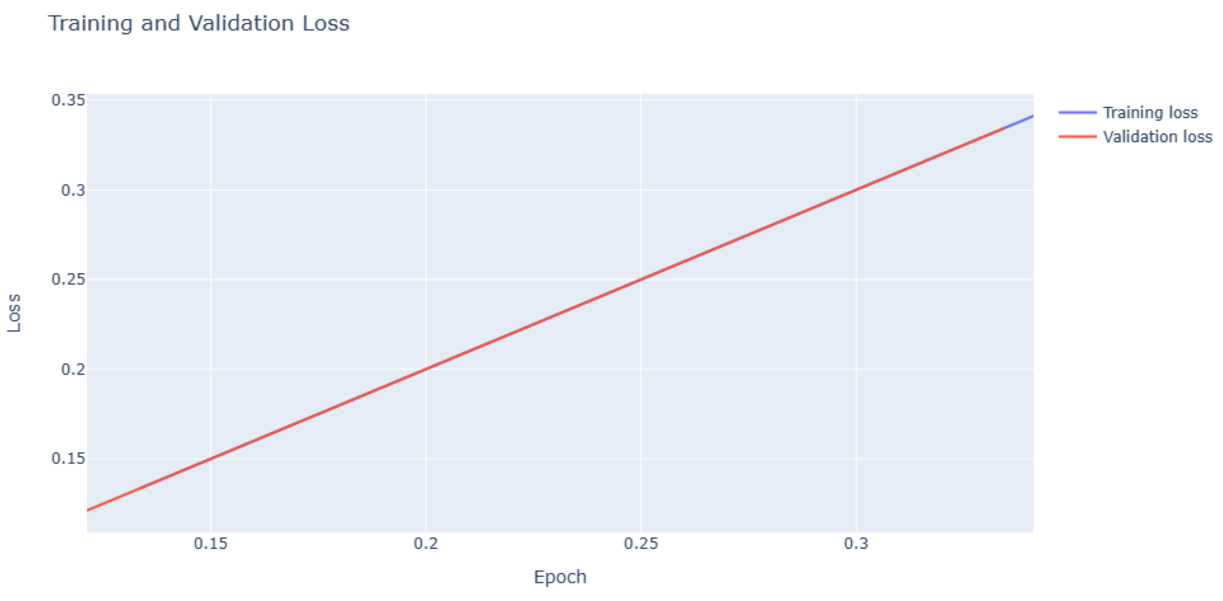


Figure 7. Train losses and Validation losses in 3 epochs.

Based on the figure above, the training loss is decreasing while the validation loss is increasing.

Train and Validation Losses for the Fourth epoch

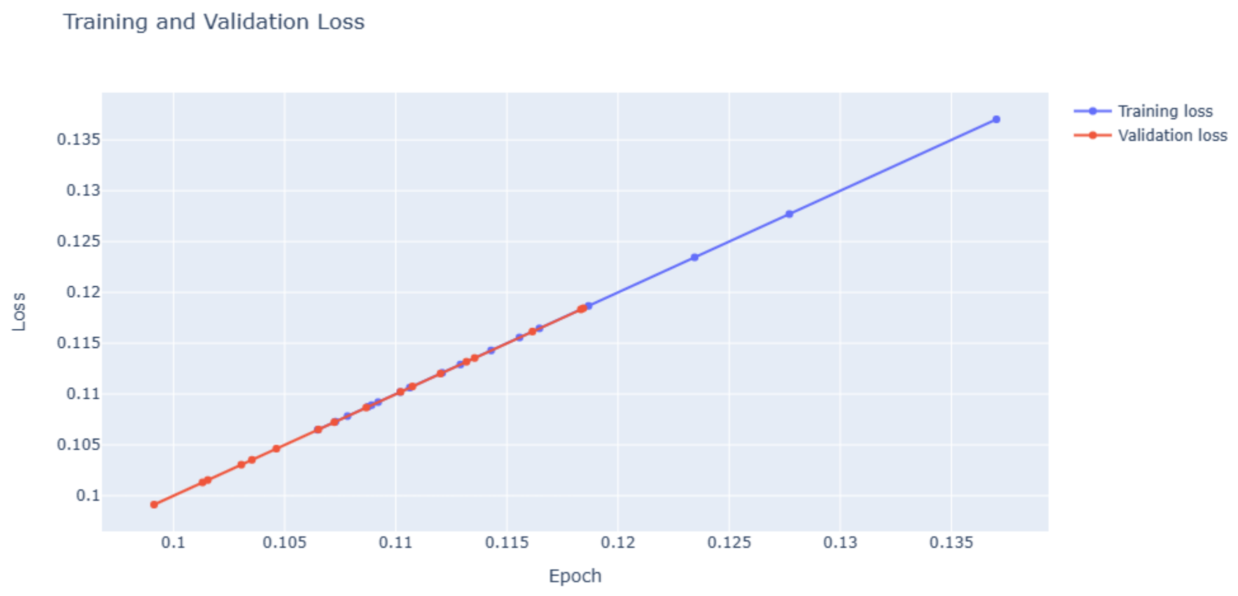


Figure 8. Fourth Epoch

The figure above shows the fourth and last epoch data presentation. The training loss is almost half the value of validation loss. Both training loss and validation loss decreases as validation accuracy changes every batch size which means its learning and it’s a good model.

Chapter 5

Conclusion

The proposed project could be regarded as a promising solution in medical applications that employ deep learning with high accuracy. Furthermore, our dataset was created under variable conditions, increasing the number of contributions, comparisons, results, and conclusions in the field of SLR and potentially improving such systems.

Individuals with hearing impairments frequently use sign language to communicate. As a result, advancements in sign language recognition detection have a significant impact on these people. As a result, considerable effort has been expended in developing a device capable of automatically recognizing the physical gestures of sign language. Many systems have been developed to interpret signs from images, but many problems remain for researchers in the field. Such difficulties include variations in the hand's size, position, shape, and background, lighting, and distance from the camera. Many studies have focused on developing sign language recognition systems by combining feature extraction and classification methods to identify hand movements.

This study can be improved by including more images in the dataset for more letters and words. Additionally, more images can be added to improve accuracy and decrease loss. The proposed system could be improved to predict a complete word by adding new words and terms. These predicted words can also be converted into speech using a text-to-speech engine. Other future work could be done to convert ASL to commands that robots or machines can understand and execute. In this case, anyone can stand in front of the robot and communicate with it using ASL.

The performance of the ResNet-50 network in the predicted data was high enough reaching 89.25% indicating the ability of the network to classify American sign language alphabet images well. The training of the model is GPU limited so the results probably can get higher than it is. The more epochs used, the more accurate the model is since it will have more time to learn from the data, This further required a higher GPU to have better model accuracy.

REFERENCES

Ahmed, et al(2022) DeepASLR: A CNN based human computer interface for American Sign Language recognition for hearing-impaired individuals. https://www.sciencedirect.com/science/article/pii/S2666990021000471#:~:text=Jain%20et%20al.,for%20a%20two%2Dlayer%20CNN.

Anantha Rao et al. (2018) proposes a CNN for the recognition of the gestures of the Indian sign language.https://www.researchgate.net/publication/323863011_Deep_convolutional_neural_networks_for_sign_language_recognition

G.A.Syamala, et al. Deep convolutional neural networks for sign language recognition in 2018 Conference on Signal Processing And Commun. Eng. Syst. SPACES (2018) 2018, vol.

https://www.sciencedirect.com/science/article/pii/S2666990021000471

M. Mohandes, J. Liu, M. Deriche A survey of image-based Arabic sign language recognition

2014 IEEE 11th International Multi-Conference on Systems, Signals & Devices (SSD14) (2014), pp. 1-4,

P. Kurhekar et al. (2019) present a system that can learn signs from videos by processing the video frames under a minimum disturbed background.https://www.sciencedirect.com/science/article/pii/S2666990021000471

S. Alyl. et al. (2017) designed a system for alphabetic Arabic sign language recognition by using intensity and depth images, which were obtained from a SOFTKINECTM sensor.https://www.sciencedirect.com/science/article/pii/S2666990021000471

Y. and Bantupalli. A vision-based application that converts sign language into text was put into place by Xie (2019).https://www.sciencedirect.com/science/article/pii/S2666990021000471#:~:text=Bantupalli%20and%20Y.,spatial%20features%20from%20video%20sequences.