50.038 Computational Data Science

American Sign Language Recognition using Computer Vision

Final Report

Completed by: Team SGD

1004677 Hoo Yong Wei, Gerald

1004116 Goh Shao Cong Shawn

1004299 Darren Loh Zhao Ying

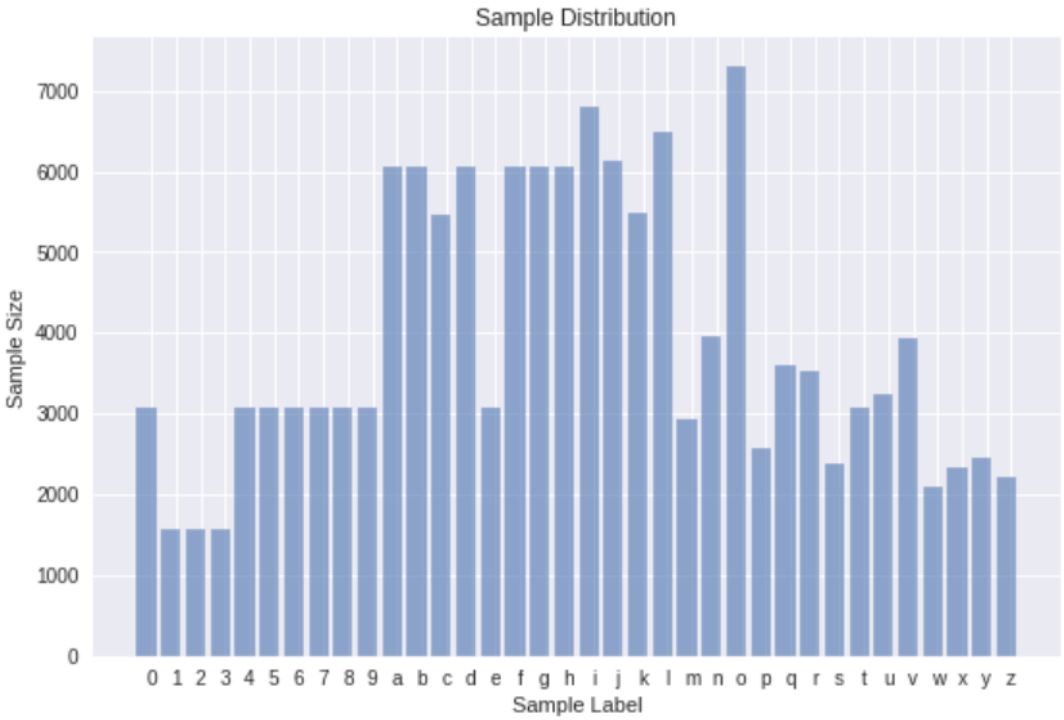
September 2021 - December 2021

Source Code: https://github.com/geraldhyw/CDSProject

Contents

1 Dataset and Collection

Our dataset[1] is obtained from the "American Sign Language Dataset" by Prathum Arikeri from Kaggle. This dataset is comprehensive and consists of over 140,000 images from 36 different categories representing numbers (0-9) and alphabets (a-z).



Despite the vast amounts of images provided, it poses 3 main problems as mentioned below:

(1) uneven distribution across the categories
(2) various images looked similar due to poor augmentation
(3) we do not have sufficient computing power to use all the images

In order to combat these problems, we have chosen to reduce the number of images in each category to 100 images, which at the same time ensures equal distribution across the classes, as well as allow us to have sufficient computing power to process our data. In addition, before training our model, we also included various data augmentation techniques such as resizing the images to a fixed size, rotation, translation, shearing and horizontal flip. These would be further discussed in our next section.

Figure 1: Sample images of categories A, B, C, D

[1]Source: https://www.kaggle.com/prathumarikeri/american-sign-language-09az

2 Data Pre-processing

2.1 Data Filtering/Reduction

In order address two of the above mentioned problems: (1) uneven distribution across the categories and (3) insufficient computing power, we cleaned the dataset by randomly selecting 100 images from each category and creating a new dataset. The randomness ensures that we pick the images with no bias to ensure the robustness of our training and test data. By reducing the images to 100 per category, we are able to have even distribution across categories, sufficient computing power to train our model and also able to achieve reasonable accuracy given our computing limits.

2.2 Folder Structuring

We made use of PyTorch's built-in class, ImageFolder, for ease of loading our dataset into PyTorch's DataLoader which will be used to process our data. ImageFolder uses the folder names as the labels for the data sets and hence a directory with a specific structure is required. In the case of our project, we have a directory that consists of a "train" and "test" folder to store the training data set and test data set respectively. Each of the folder would then contain a total of 36 folders representing the categories of 0 to 9 and a to z which stores the images of the respective categories. Shown below is an example of how our directory structure would look like.

```
dataset/
|---train/
|     |
|     |---0/
|     |   |---0train-001.jpg
|     |   |---0train-002.jpg
|     |     ...
|     |---1/
|     |   |---1train-001.jpg
|     |   |---1train-002.jpg
|     |     ...
|     | ...
|     |---z/
|         |---ztrain-001.jpg
|         |---ztrain-002.jpg
|
|---test/
      |
      |---0/
      |   |---0test-001.jpg
      |   |---0test-002.jpg
      |     ...
      |---1/
      |   |---1test-001.jpg
      |   |---1test-002.jpg
      |     ...
      | ...
      |---z/
          |---ztest-001.jpg
          |---ztest-002.jpg
```

Figure 2: Directory structure example

To populate our directory, every category folder in the raw data set would first be shuffled. Afterwards, every category folder in our directory would be populated by a predetermined number of images from the shuffled raw data set with the ratio between the training data set and the testing data set to 8:2. This method imitates a stratified sampler where every category would have the same number of randomised data, preventing the over-fitting of our model due to skewed data inputs given during the training of the model.

2.3 Image Augmentation

To improve the effectiveness of our model training, our group made use of data augmentation to include a wider ranger of variety in our training inputs. To do this, we made use of the Transforms method in PyTorch that chains together a bunch of image transformations to be applied to the images in our data set.

Listed below are the transformations that were used and applied in our dataset:

1. Resizing the images to a size of (150,150).
2. Rotating the images by a range of -5 to 5 degrees.
3. Translating the images in the range of -image-width * 0.1 to image-width * 0.1 horizontally and vertically.
4. Shearing the images in the range of -7 to 7 degrees.
5. Randomly flipping the images horizontally.
6. Normalizing the tensor images with a mean of [0.485, 0.456, 0.406] and a standard deviation of [0.229, 0.224, 0.225]

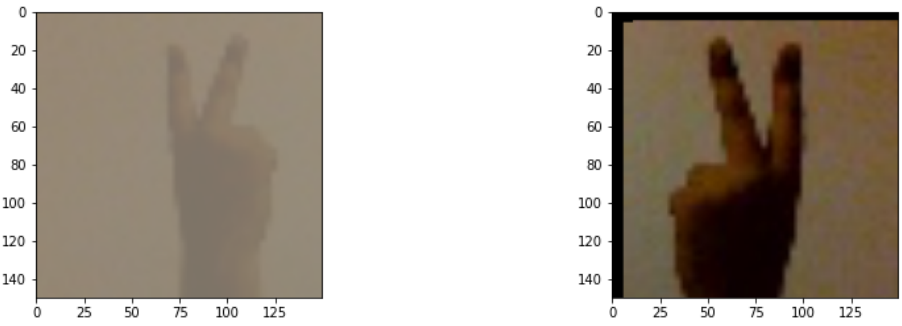


Figure 3: Before(left) and After(Right) Augmentations

3 Problem and Algorithm/Model

3.1 Problem Statement

Our project aims to explore, understand and implement human-computer interactions using American Sign language to enable the translation and instruction of computer actions using hand signs. In essence, we would like to predict an alphabetical letter given a valid hand sign image input with reference to the American sign language.

With computers learning how to identify speech or recognising voice, it is essential for us to incorporate technologies that are friendly towards disabilities such as deafness. By allowing computers to understand sign language, we can become more inclusive towards the deaf. We can foresee a more inclusive future where technologies can recognize speech and also sign language to communicate to both deaf and non-deaf individuals. This is important as we do not wish to leave pockets of individuals behind while advancing our technology.

3.2 Algorithm/Model

The model that we are using is the ResNet18 model. ResNet18 is a convolutional neural network that has a total of 18 layers. A pretrained version of the model can also be loaded that was trained using the ImageNet database. In our project, we have made use of the pretrained version of the model to improve the accuracy of the model and reduce the number of incorrect classifications. The use of a pretrained model also reduces the amount of time needed to train the model.

In general the ResNet model offers an advantage in the form of a solution to the degradation problem, which refers to a negative effect on performance when more layers are added to the model. ResNet solves this by introducing residual blocks within the intermediate layers of a block that learns a residual function with reference to the block input. There also exists an identity function, that could also be coined as a "skip connection", that passes along the input from the beginning of the block to the end of the block and adds it to the original output of the block, hence preserving information as the model progresses through the layers.

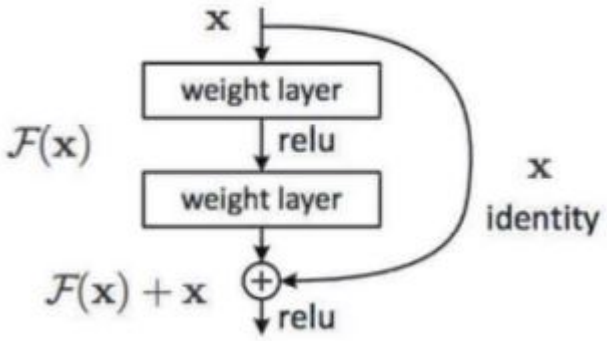


Figure 4: Example of inner ResNet function

3.3 Related Works

3.3.1 Urdu Handwritten Text Recognition [1]

The project proposed by Muhammad Kashif and his team seeks to use machine learning to develop a handwriting character recognition system for the Urdu language, which is also the national language of Pakistan. The project focuses on handwritten texts which is known to be difficult due to the multitude of different writing styles that every individual has. This is especially the case for the Urdu language due to the cursive nature of the characters. Using a ResNet18 model, the project aims to uses symbol and pattern recognition to understand the Urdu characters and numerals, hence recognising a handwritten text pattern and converting the characters or words into a format that the computer would understand. The use of the ResNet18 model is significant due to its 18 convolution layers, allowing the model to better recognise the Urdu text with its cursive style, hence resulting in a higher accuracy as compared to previous models used.The ResNet18 model used would also be paired with the Urdu Nastaliq Handwritten data set that contains 3,12000 words written by 500 different candidates, making the variety of the data set larger for more accurate text recognition. Shown below is an example of the Urdu characters to illustrate its cursive nature.

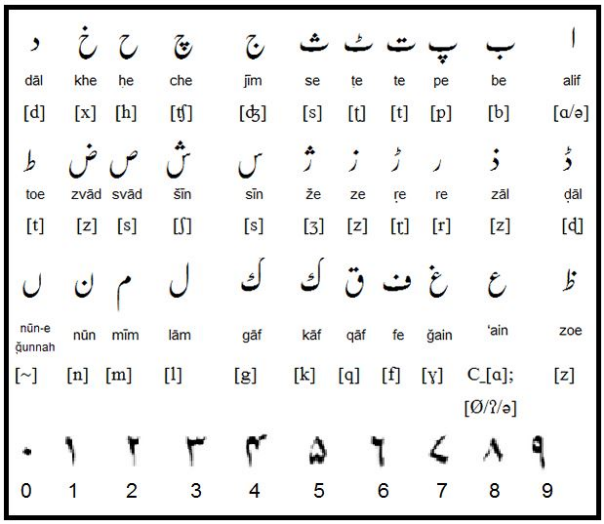


Figure 5: Example of Urdu characters

Other than the common Resnet18 model that both projects use, the project done by Muhammad Kashif and his team shares many similarities with our project due to their common goals of deciphering and translating the given inputs into recognisable characters from a language. The project also shares similarities due to their emphasis on pattern recognition which is important due to the multitude of varieties present in both handwritten texts and hand signs where texts and signs from different individuals would often have minute differences.

4 Evaluation Methodology

4.1 Loss Function

In order to evaluate our model, we have chosen our loss function to be the cross entropy loss function. As the cross entropy loss is a multi-class function, it is useful in our case compared to the binary cross entropy loss function because we have a total of 36 categories in our application.

Cross entropy loss measures the performance of a classification model where the output would be a probability value between 0 and 1, where the cross entropy loss increases the more the output diverges from its actual label. For example, having a low probability of 0.1 when the true label is 1 would be bad and results in a high loss value. In multi-class classification, which is also the one used in our project, a separate loss would be calculated for each class label per an observation and the final result would be summed.

Shown below is the formula for categorical cross entropy function

$$H(y, p) = -\sum_{i} y_i log(p_i) \tag{1}$$

4.2 Accuracy

For our model, we have trained it to reach an accuracy of around 82% when predicting on unseen dataset. The formula for accuracy is as follows

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

where TP: True Positive, FP: False Positive, TN: True Negative, FN: False Negative. Since we have the same number of samples in each category, using accuracy as a means of evaluating our model is reasonable.

```
Validation-epoch 45. Avg-Val.Loss: 1.1931, Val. Accuracy: 0.7833
| epoch  46 |    50/  288 batches | train_loss    1.452 | accuracy    0.784 | elapsed time   5.5 seconds
| epoch  46 |   100/  288 batches | train_loss    1.463 | accuracy    0.774 | elapsed time   5.3 seconds
| epoch  46 |   150/  288 batches | train_loss    1.470 | accuracy    0.754 | elapsed time   5.3 seconds
| epoch  46 |   200/  288 batches | train_loss    1.414 | accuracy    0.804 | elapsed time   5.3 seconds
| epoch  46 |   250/  288 batches | train_loss    1.377 | accuracy    0.814 | elapsed time   5.3 seconds
Validation-epoch 46. Avg-Val.Loss: 1.1370, Val. Accuracy: 0.7792
| epoch  47 |    50/  288 batches | train_loss    1.439 | accuracy    0.769 | elapsed time   5.5 seconds
| epoch  47 |   100/  288 batches | train_loss    1.459 | accuracy    0.762 | elapsed time   5.3 seconds
| epoch  47 |   150/  288 batches | train_loss    1.350 | accuracy    0.796 | elapsed time   5.3 seconds
| epoch  47 |   200/  288 batches | train_loss    1.377 | accuracy    0.788 | elapsed time   5.3 seconds
| epoch  47 |   250/  288 batches | train_loss    1.438 | accuracy    0.782 | elapsed time   5.3 seconds
Validation-epoch 47. Avg-Val.Loss: 1.0867, Val. Accuracy: 0.8125
| epoch  48 |    50/  288 batches | train_loss    1.376 | accuracy    0.792 | elapsed time   5.6 seconds
| epoch  48 |   100/  288 batches | train_loss    1.414 | accuracy    0.762 | elapsed time   5.3 seconds
| epoch  48 |   150/  288 batches | train_loss    1.370 | accuracy    0.822 | elapsed time   5.3 seconds
| epoch  48 |   200/  288 batches | train_loss    1.415 | accuracy    0.748 | elapsed time   5.3 seconds
| epoch  48 |   250/  288 batches | train_loss    1.296 | accuracy    0.808 | elapsed time   5.3 seconds
Validation-epoch 48. Avg-Val.Loss: 1.0779, Val. Accuracy: 0.8111
| epoch  49 |    50/  288 batches | train_loss    1.413 | accuracy    0.763 | elapsed time   5.5 seconds
| epoch  49 |   100/  288 batches | train_loss    1.320 | accuracy    0.798 | elapsed time   5.3 seconds
| epoch  49 |   150/  288 batches | train_loss    1.337 | accuracy    0.792 | elapsed time   5.3 seconds
| epoch  49 |   200/  288 batches | train_loss    1.303 | accuracy    0.796 | elapsed time   5.3 seconds
| epoch  49 |   250/  288 batches | train_loss    1.240 | accuracy    0.820 | elapsed time   5.3 seconds
Validation-epoch 49. Avg-Val.Loss: 1.0380, Val. Accuracy: 0.8194
```

Figure 6: Epoch Accuracy

4.3 Confusion Matrix

In addition to the accuracy, we have also obtained our confusion matrix which is able to give us a visualisation of which categories were predicted correctly or which categories we predicted when the prediction was wrong. In the confusion matrix below, a darker shade of blue represents a higher count while a lighter shade of blue represents a lower count. The x-axis shows the predicted categories while the y-axis shows the actual categories. The diagonal from the top left to bottom right corresponds to our true positive counts. Meaning to say, these are the counts where our model predicted the category that is actually the true category. Ideally, we would like to have more counts on this diagonal as this means that our model is predicting well.

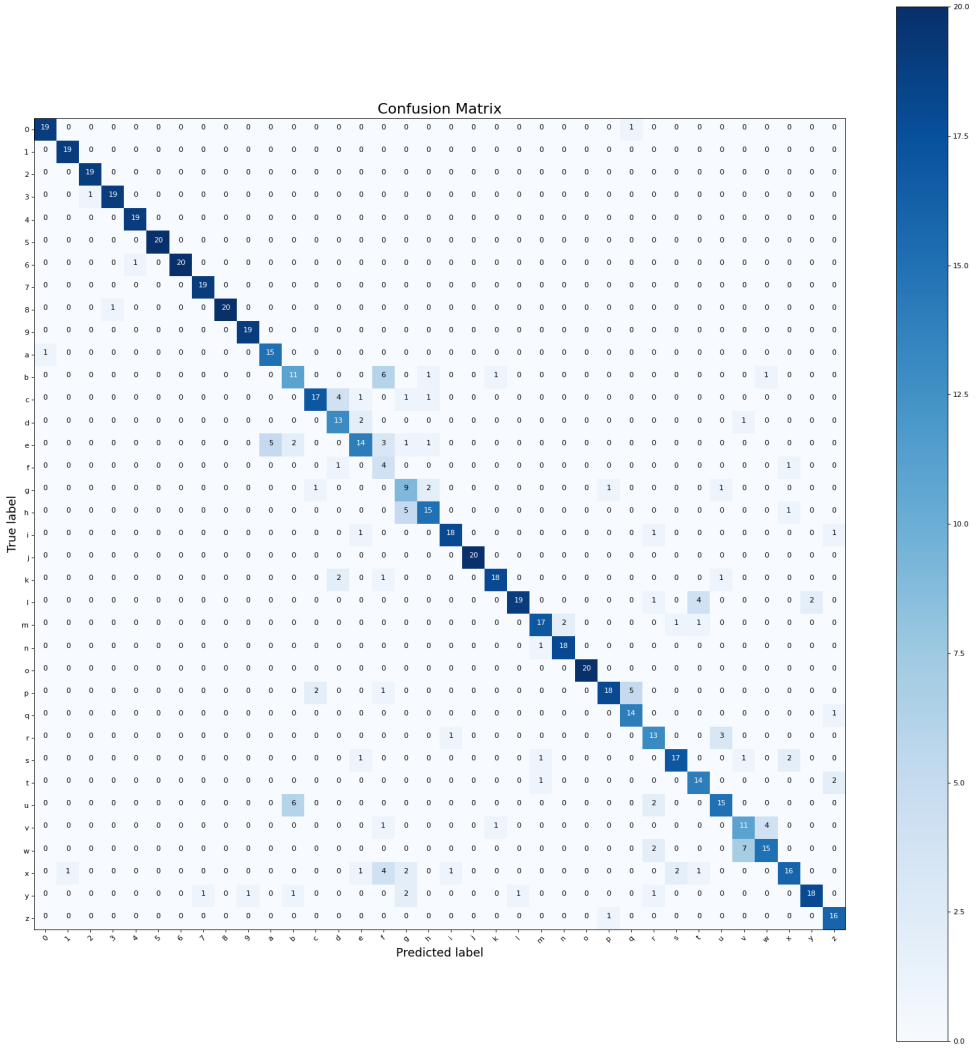


Figure 7: Confusion matrix

As can be seen from the confusion matrix, our model has a very high amount of true positives which means our model predicts well on unseen images, backing up our accuracy. For context, we are calculating our confusion matrix using 20 images from each category, hence the highest count in each box will be 20.

5 Results and Discussion

All in all, our model was able to predict unseen images from our test set quite reliably, with an accuracy of about 82% and also a confusion matrix with high number of true positives. This is all achieved using only 100 images from each category (given our computing constraints), which may be considered quite a small dataset compared to state of the art computer vision technologies. Given no constraints we would definitely work on improving two main areas (1) improving the quality of the images and (2) increasing the number of images for training. Training images could be improved by having more variances in background, lighting and in terms of the camera resolution. By having more pixels (or more information), we should be able to attain a better model that predicts with an even higher accuracy. In addition, by allowing our model to train on a larger dataset, our model is able to learn more on how a sign language may look like, increasing the effectiveness of our model.

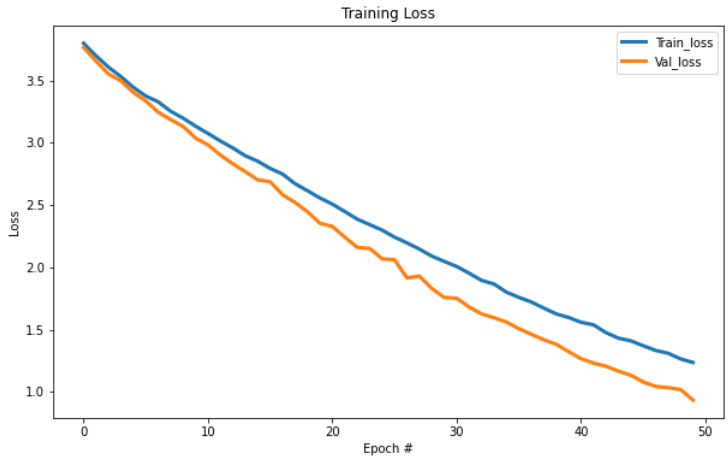


Figure 8: Loss Graph

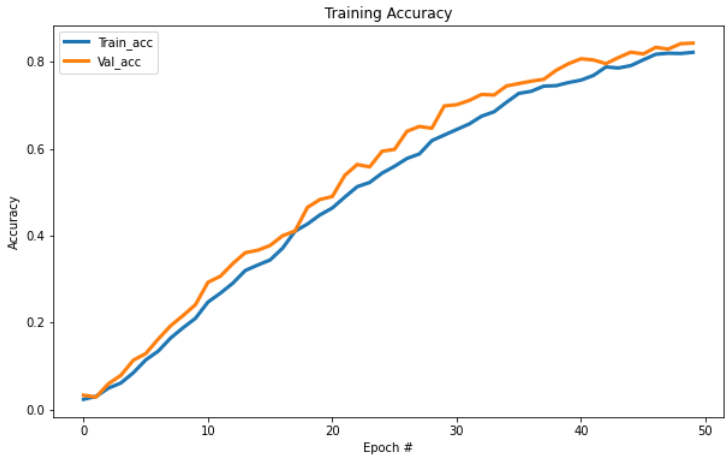


Figure 9: Accuracy Graph

References

[1] Muhammad Kashif. Urdu handwritten text recognition using resnet18. *arXiv preprint arXiv:2103.05105*, 2021.