Detecting American Sign Language from Hand Gestures
Report for Project by Team 35

Sai Abhishek Guraja Georgios Panayiotou
Shreyas Shivakumara

June 1, 2020

Abstract

American Sign Language (ASL) is the most common language used by the hard of hearing communities, with over half a million people speaking it. However, not a lot of people are able to converse in ASL without the presence of a translator, but various technological approaches aim to support this difficulty in communicating. In this work, a Deep Learning network model is proposed to identify ASL alphabet characters from static images and a possible explanation for which regions of the image are considered important to the network when recognizing some letters is presented.

1. Introduction

Sign languages are forms of communication, consisting of hand signs, facial expressions and body language, helping people with speech and hearing disabilities carry on with their daily tasks. Albeit recognized as natural languages, sign languages usually are not known by the general public, making communication without a translator fairly difficult. With society advancing technologically, programming-based approaches to this problem come to aid.

Created in 1817 by the American School for the Deaf, American Sign Language (ASL) is the most known sign language between hard of hearing people, spoken mainly in USA, Canada and Mexico, as well as some parts of Asia and Africa. Similarly to oral languages, it has its own variety of dialects, Its alphabet, like English, is made of 26 gestures, each representing a letter from 'A' to 'Z', that can create words when used in order, although gestures for words commonly used also exist.

The aim of this work is to provide a method to translate ASL alphabet gestures from images using neural networks, as well as to give an explanation to the how these networks recognize the gestures provided.

2. Related Work

Various implementations tackling the problem of ASL alphabet recognition have been published. One of the first methods published for the problem used Hidden Markov Models [18]. Since then, methods using CNNs have been proposed to solve the problem with very high accuracy, even in real time, although the letters 'j' and 'z' cannot be easily recognized since they involve movement that cannot be extracted from a single image. Deep Learning has been proposed as a potential method, using the Squeezenet model [10]. Tuning existing pre-trained CNNs has resulted in the letters being recognized with good accuracy [3], [12]. Some other interesting methods used were generating a diagram of the hand's landmarks when given a depth image input from Microsoft Kinect's glove and feeding it to an Radial-Basis Function classifier [2], or performing Multiview Augmentation and Inherence Fusion methods [19], both resulting in a high letter recognition accuracy. For our work, we combined Deep Learning networks, along with some data augmentation methods.

3. Approach

Recognizing the ASL alphabet using Deep Neural Networks has been done before, with great success [10]. For this project, we propose a solution for the problem using Deep Learning methods and compare how accurate our solution is when performing various augmentation methods on the images in our dataset. This section provides the description of the dataset and the CNN configuration we used.

3.1. Design

For this work, we are using Deep Learning algorithms, because they have shown a high accuracy rate processing data in a short time. Among all the different deep learning algorithms, we are particularly interested in Convolutional Neural Network (CNN), the neural network class most commonly applied to analyzing visual input, since it compares the images piece by piece. Also, downsampling the image using adjacent pixel information can easily extract high level features from the images.
A possible method we initially considered was to use a pre-trained network for our problem, which has also shown success in recognizing ASL gestures [3],[12], but we decided to instead build our network from scratch.

3.2. Dataset

Creating our own dataset for the purposes of the project would take an enormous amount of time, thus we decided to use a publicly available one found on Kaggle, following the Modified National Institute of Standards and Technology (MNIST) convention. The pixel values of the 28x28 images are represented by values in grayscale colouring and the letter they represent is given as a label 1. Using grayscale colour values may present an advantage, since it contains one color channel, and reduces the effect of illumination's differences [5]. An example of how these images look when converted using Matplotlib [7] is shown in Figure 2.
The dataset has approximately the same number of sequences for each target, as can be seen in 3, which prevents an imbalance in the set [9]. However, the dataset does not contain images for letters J and Z, since they require a motion, nor images that do not represent a letter in the alphabet but look similar.

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14	pixel15	pixel16	pixel17	pixel18
0	3	107	118	127	134	139	143	146	150	153	156	158	160	163	165	159	166	168	170
1	6	155	157	156	156	156	157	156	158	158	157	158	156	154	154	153	152	151	149
2	2	187	188	188	187	187	186	187	188	187	186	185	185	185	184	184	184	181	181
3	2	211	211	212	212	211	210	211	210	210	211	209	207	208	207	206	203	202	201
4	13	164	167	170	172	176	179	180	184	185	186	188	189	189	190	191	189	190	190

5 rows × 785 columns

Figure 1: American Sign Language csv dataset

The dataset is split into train and test with 27,455 images in training data and 7172 images for testing. The data is stored as a 28x28 pixel image. An image pipeline was used based on ImageMagick [20] and included cropping to hands-only, gray-scaling, resizing, and then creating at least 50+ variations to enlarge the quantity[11].

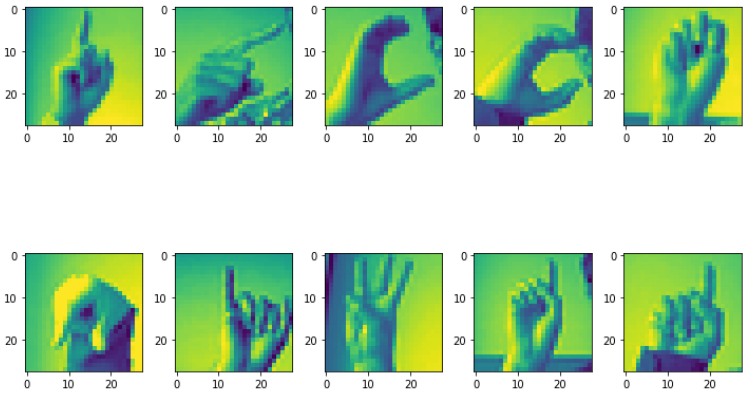


Figure 2: American Sign Language Dataset

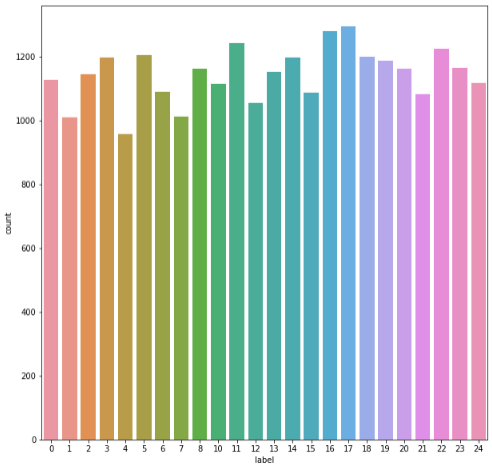


Figure 3: Result of visualizing the training labels

3.3. Data Augmentation

Data augmentation methods including zoom, shear, rotation, flip of the image can increase the dataset's size [15], resulting in better accuracy. With this in mind, we performed various methods on the set. More specifically, we performed zooming, shifting to both left and right, rotating by ten degrees to both left and right, and flipping only horizontally to assimilate ways that this gesture can be captured in an image. Flipping vertically or rotating by a bigger angle would not have made sense, since they can become gestures that represent different letters.

3.4. CNN Configuration

The CNN that has been modeled for this project to recognise the hand gestures consists of three convolutional layers, three max pooling layers, one fully connected layer and output layer. There is one dropout in this network to prevent it from overfitting [17].

Initially, our image input, resized to be three-dimensional, is fed to the first convolution layer, consisting of 75 filters and a 3*3 kernel. Rectified Function (ReLU) is used as the activation function in this layer, since it performs better than other optimizers, like tanh or sigmoid and is introduced for the non-linearity [4]. The feature maps produced are then sent to a max-pooling layer to reduce the spatial representation of the images, so as for the network to only consider the essential features of the image. This procedure is repeated once again, with two similar convolution layer of 50 and 25, respectively, filters and two max-pooling layers, in alternating order.

The flattening layer then receives the output from the last max pooling layer and converts the two-dimensional matrix to a vector, which is subsequently passed onto the 512-node connected layer, activated by ReLU. This layer is followed by the dropout layer which removes 20% of neurons randomly to prevent overfitting[6]. Finally, the output layer has 24 nodes correspondingly to the 24 letter classes of the dataset. It has a softmax function[8] that outputs a probabilistic value for each of the classes.

The model is compiled using the ADAM optimizer with a learning rate of 0.01. Categorical Cross Loss used to calculate the loss. In the end, loss and accuracy are two metrics introduced used to keep track of the evaluation process.

Summary of the CNN Configuration is seen in Table 1

3.5. System Implementation

Python[21] is the programming language used to implement the system and Google Collab was used to write and execute the code. Numpy[13] is used for the arithmetic operations. Matplotlib[7] is used to visualize the datasets. Keras[1] library is used to implement the CNN Classifier. Scikit-learn [14] is used to split the data into test dataset and training dataset and to calculate the confusion matrix.

Model Content	Details
Epochs	20
First Convolution Layer	75 filters of size 3x3, ReLU, input size 28x28
First Max Pooling Layer	Pooling Size 2x2
Second Convolution Layer	50 filters of size 3x3, ReLU
Second Max Pooling Layer	Pooling size 2x2
Third Convolutional Layer	25 filters of size 3x3, ReLU
Third Max Pooling Layer	Pooling size 2x2
First Fully Connected Layer	512 nodes, ReLU
Dropout	Excludes 20% neurons randomly
Output Layer	24 nodes for 24 classes, SoftMax
Optimisation Function	ADAM Optimizer
Learning rate	0.01
Metrics	Accuracy,Loss

Table 1: CNN Configuration

4. Results

In this section, we show the accuracy of our system using the CNN configuration shown in table 1 on the MNIST ASL dataset. Furthermore, we present the saliency map retrieved from our trained CNN using the Randomized Input Sampling for Explanation (RISE) method [16] for two letters of the ASL alphabet.

4.1. Evaluation of CNN

In order to evaluate our system, we compared the network's accuracy by training for 20 epochs, first on the raw dataset, then on the augmented one. The results show us that the model trained with a data augmented dataset achieved an accuracy of 98.65%, slightly higher compared to the model trained without the data augmented dataset. The summary of the results in shown in Table 2.

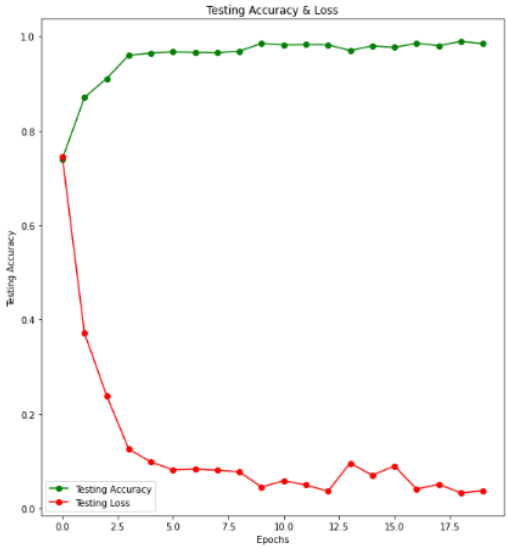
Method	Data Size	Accuracy
Training with Data Augmentation	21524720	98.65%
Training without Data Augmentation	27455	92.45%

Table 2: Results of the analysis

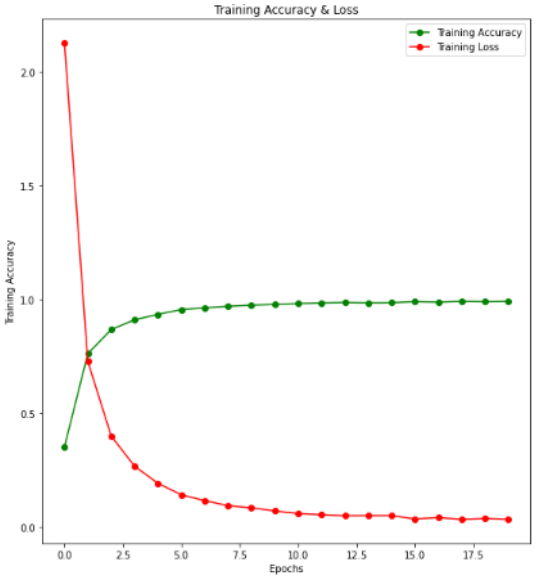
The accuracy and loss graphs for CNN model training with augmentation data is shown in 4.The accuracy and loss graphs for CNN model training without augmentation data is shown in 5. The accuracy of augmented models was higher than the non-augmented model in each epoch. the loss of the augmented model was also less compared to the non-augmented model. The summary of the classification results is shown in table 3.

Performance	CNN without Augmentation	CNN with Augmentation
Precision	0.9250	0.9835
Recall	0.9245	0.9822
F-Measure	0.9247	0.9273
Accuracy	0.9245	0.9865

Table 3: Classification results

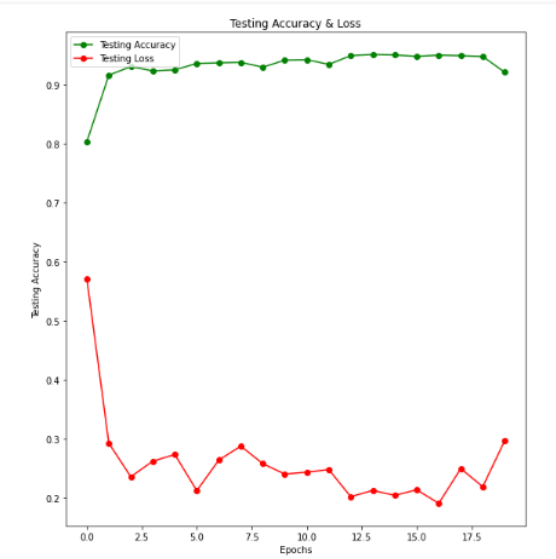


(a) Result of testing accuracy and loss

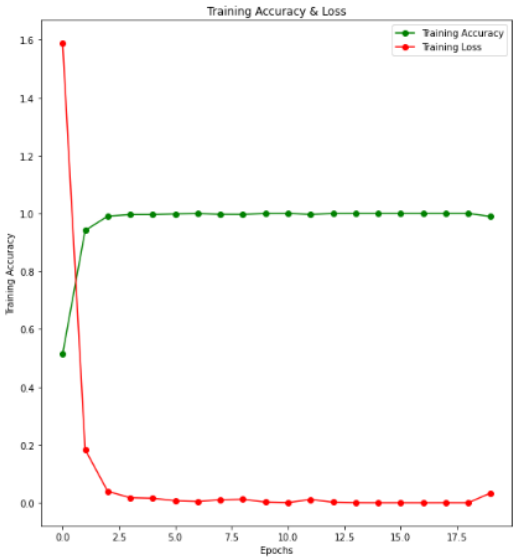


(b) Result of training accuracy and loss

Figure 4: Analysis of model with data augmenting the dataset



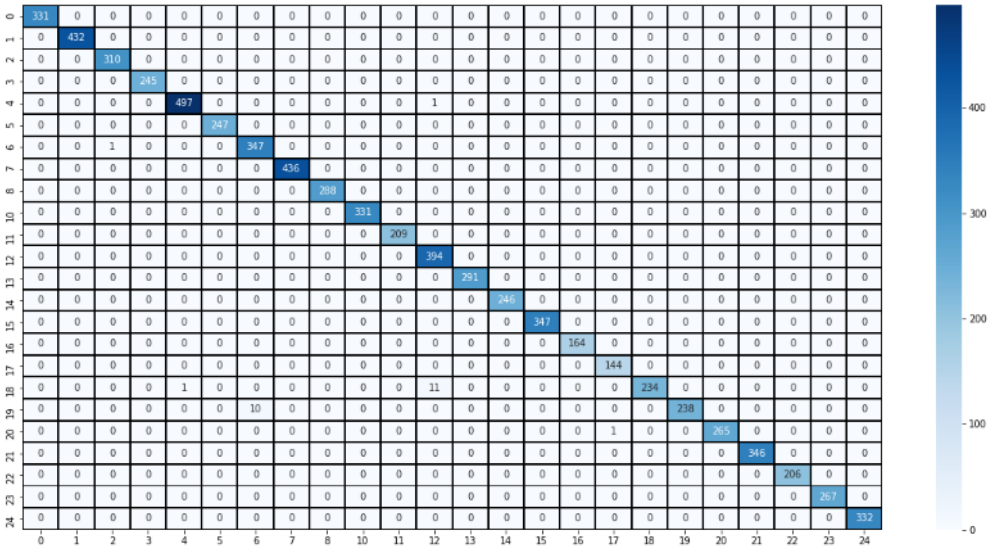
(a) Result of testing accuracy and loss



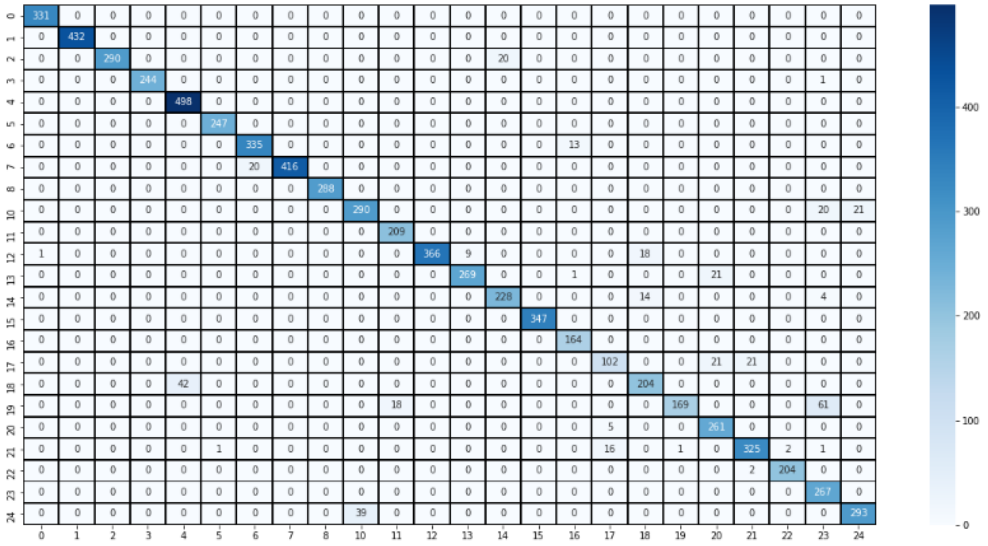
(b) Result of training accuracy and loss

Figure 5: Analysis of model without data augmenting the dataset

The confusion matrices of both the models are seen in 6a and 6b. The diagonal values show the correctly classified classes for each of the classes. The non-diagonal values show the misclassified results in each of the classes. The results show that the augmented model shows better results for class 10, 17 and 19 with 330, 144 and 238 correctly classified while the non-augmented model classified only 290, 107 and 169 correctly classified. There is a difference of around 35 between each class, which points towards a better performance of the data augmented model.



(a) Confusion Matrix with Data Augmentation

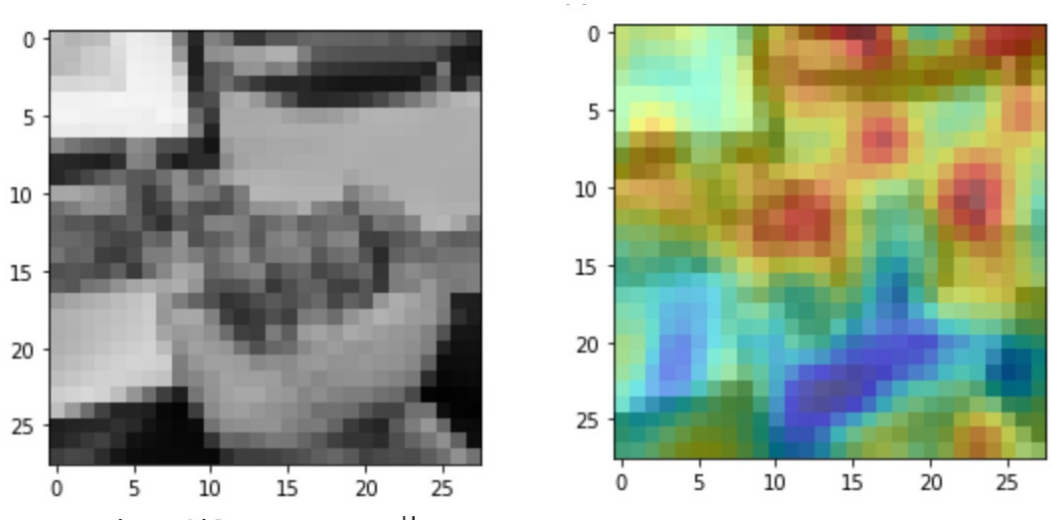


(b) Confusion Matrix without Data Augmentation

Figure 6: Results of Confusion Matrix

4.2. Applying the RISE algorithm

RISE is an algorithm that measures importance maps for image regions processed through neural networks [16]. This is done by generating multiple randomized masks over an image and feeding them to a trained network, treated as a black box. By calculating a weighted sum over prediction values for the masked image, one can gain a saliency map, with higher values in the map indicating a higher importance of that particular region in the network's decision.
For the purposes of this project, we applied the RISE method over two pictures captured by a mobile phone, resized and converted to grayscale, that represent

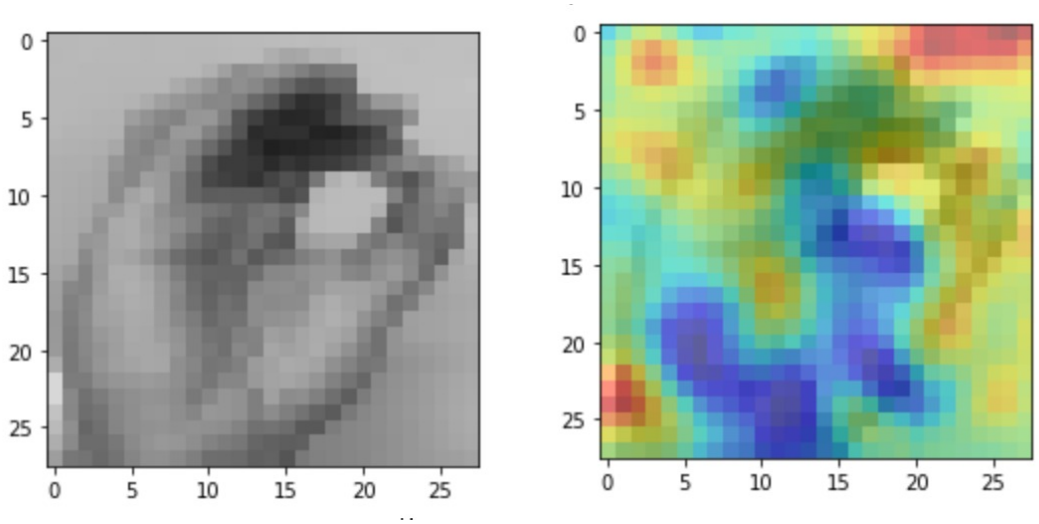


(a) ASL letter 'I' (b) Saliency map for 'I'

Figure 7: Letter I

the gestures for the ASL letters 'I' and 'O'. To generate more accurate results, the probability of the original image keeping its colour was set to $p = 0.1$, and $N = 5000$ masks were generated for each image. The results are shown in Figures 7 and 8, where higher importance values for pixels have a red colour, while pixels with lower values are blue.

As can be seen in both of the figures, pixels on the palm tend to have a lower importance value on the map, where one would expect the palm region to be important when recognizing a hand gesture. Another important observation is that regions where the grayscale values suddenly change by a significant amount tend to have more important. That is prevalent on the map for the letter I (Figure 7), where the much darker upper right region has received very high saliency values.



(a) ASL letter 'O' (b) Saliency map for 'O'

Figure 8: Letter O

5. Discussion

The results of our evaluation suggest a better recognition accuracy when recognizing ASL alphabet letters when the dataset is augmented using the methods described, further agreeing with [19]. However, a possible reason for such a high accuracy, could be the lack of implementation of a class for an invalid gesture. Some letters are quite similar in ASL, and minor gesture changes could result in different or invalid characters, which is not accounted for in our model. Also, while it may be easier to recognize gestures from images converted to grayscale, this may result in extra computational time when trying to identify characters in real-time.

Our model, along with many of the proposed ones, are unable to identify the motion-involving characters J and Z. It would be interesting to see whether this model can be extended to include them, along with other moving gestures that represent common words in ASL.

Finally, the RISE algorithm on our model has provided us with a possible explanation on how neural networks go about identifying gestures in ASL. However, the effect of the method on colourized and larger pictures representing gestures should also be observed, since different and smoother importance vectors will probably be generated. Furthermore, saliency maps for more letters and gestures in sign languages should be generated so that a greater understanding can be gained.

In general, our proceedings further suggest using Deep Learning Networks for the

problem of recognizing letters in ASL from static images and augmenting them to gain a larger dataset, could prove to be successful, while also giving a limited explanation on which parts of the image could be important to the network.

6. Future Works

Albeit a project with a limited scope, the present work can provide a guideline for future research. The model can be extended to additionally detect moving gestures, or perform real-time recognition from a video feed. The approach of using a pre-trained model for the same dataset could also be attempted, in order to compare the effectiveness of transfer learning. Finally, this model can be used to gain importance maps for more of the letters in the ASL alphabet.

References

[1] François Chollet et al. *Keras*. https://keras.io. 2015.

[2] Cao Dong, Ming Leu, and Zhaozheng Yin. "American Sign Language alphabet recognition using Microsoft Kinect". In: June 2015, pp. 44–52. DOI: 10.1109/CVPRW.2015.7301347.

[3] Brandon Garcia and Sigberto Alarcon Viesca. "Real-time American Sign Language Recognition with Convolutional Neural Networks". In: *Convolutional Neural Networks for Visual Recognition* 2 (2016).

[4] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dud$\sqrt{}\neq k$. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323. URL: http://proceedings.mlr.press/v15/glorot11a.html.

[5] Mark Grundland and Neil Dodgson. "Decolorize: Fast, contrast enhancing, color to grayscale conversion". In: *Pattern Recognition* 40 (Nov. 2007), pp. 2891–2896. DOI: 10.1016/j.patcog.2006.11.003.

[6] Geoffrey E. Hinton et al. *Improving neural networks by preventing co-adaptation of feature detectors*. 2012. arXiv: 1207.0580 [cs.NE].

[7] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[8] Changhun Hyun and Hyeyoung Park. "Recognition of Facial Attributes Using Multi-Task Learning of Deep Networks". In: *Proceedings of the 9th International Conference on Machine Learning and Computing*. ICMLC 2017. Singapore, Singapore: Association for Computing Machinery, 2017, 284,Äì288. ISBN: 9781450348171. DOI: 10.1145/3055635.3056618. URL: https://doi.org/10.1145/3055635.3056618.

[9] Khoshgoftaar Johnson Justin M. and Taghi M. "Survey on deep learning with class imbalance". In: *Journal of Big Data* 6 (Mar. 2019). DOI: 10.1186/s40537-019-0192-5.

[10] Nikhil Kasukurthi et al. "American Sign Language Alphabet Recognition using Deep Learning". In: *CoRR* abs/1905.05487 (2019). arXiv: 1905.05487. URL: http://arxiv.org/abs/1905.05487.

[11] Madz2000. *Sign Language MNIST Data Description*. 2017. URL: https://www.kaggle.com/datamunge/sign-language-mnist/data (visited on 05/26/2020).

[12] Manuel Eugenio Morocho-Cayamcela and Wansu Lim. "Fine-tuning a pre-trained Convolutional Neural Network Model to translate American Sign Language in Real-time". In: Jan. 2019, pp. 100–104. DOI: 10.1109/ICCNC.2019.8685536.

[13] Travis Oliphant. *NumPy: A guide to NumPy*. USA: Trelgol Publishing. [Online; accessed <today>]. 2006–. URL: http://www.numpy.org/.

[14] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[15] Luis Perez and Jason Wang. *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*. 2017. arXiv: 1712.04621 [cs.CV].

[16] Vitali Petsiuk, Abir Das, and Kate Saenko. “RISE: Randomized Input Sampling for Explanation of Black-box Models”. In: *CoRR* abs/1806.07421 (2018). arXiv: 1806.07421. URL: http://arxiv.org/abs/1806.07421.

[17] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[18] Thad Starner and Massachusetts Group. “Visual Recognition of American Sign Language Using Hidden Markov Models”. In: (May 1995).

[19] Wenjin Tao, Ming C. Leu, and Zhaozheng Yin. “American Sign Language alphabet recognition using Convolutional Neural Networks with multiview augmentation and inference fusion”. In: *Engineering Applications of Artificial Intelligence* 76 (2018), pp. 202–213. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2018.09.006. URL: http://www.sciencedirect.com/science/article/pii/S0952197618301921.

[20] The ImageMagick Development Team. *ImageMagick*. Version 7.0.10. June 12, 2020. URL: https://imagemagick.org.

[21] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.