American Sign Language Image Classification Using A Convolutional Neural Network (April 2019)

Michael Barnard, Nikita Buslov, Rosemond Fabien and Trung Tran, *Member, TRM*

***Abstract*—American Sign Language (ASL) image classification suffers from noise and overtraining due to limited labeled data and people not following instructions. In this report, we examine how different features can be extracted from an image to assist in classification of ASL images. To this end, multiple classification models were explored and used to determine the best model to accurately label the ASL images. From k-nearest neighbors (KNN), random forest, and convolutional neural networks (CNN). Through trial and error with all of the aforementioned classifiers, the CNN gave the most accurate results. Using a pretrained CNN from the deep learning "Fast.ai" library called a deep residual network (ResNet), specifically "ResNet34," the resulting accuracy was approximately 88% with the entire class' given dataset.**

***Index Terms*—American Sign Language (ASL), random forest, hue, saturation, and value (HSV), k-nearest neighbors (KNN) convolutional neural network (CNN), Exchangeable Image File (EXIF), deep residual network (ResNet), deep learning library (Fast.ai)**

I. Introduction

AN ongoing challenge within the field of machine learning and artificial intelligence is what classifiers to use and what parameters to set them. This paper will discuss and show how to implement a convolutional neural network (CNN) classification to achieve the highest accuracy possible and maybe shed some insight as to why other classifier may not be optimal for ASL image classification.

Despite noise, random shoulders, and other items besides the hand itself appearing in the images. The preprocessing was able to crop out the shoulders using a box blur and an HSV threshold. Outlined in this letter is the implementation of the preprocessing, feature extractions, and random forest classifier. Furthermore, an explanation on why a CNN was decided in lieu of a random forest will be provided alongside their results. The result of the CNN model was approximately 88% accurate.

This paper is formatted as follows. Section II will discuss how this project was implemented. Section III will discuss the experiments that were done in order to achieve our final results. Section IV is the conclusion based on the experiments that were performed.

II. Implementation

The following section is separated into 4 parts: Preprocessing, Feature Extractions, Random Forests, and CNN.

A. Preprocessing

The dataset comprised of 100x100 pixeled images. Each image is expected to have only one ASL letter signed in the center with a white opaque background. However, upon further inspection of the dataset, people did not follow instructions when captured their images. There were several issues in the images that includes but are not limited to: 75x100 pixeled images, items appearing in the image other than the hand itself such as a shoulder, non-white backgrounds, reflective backgrounds, noisy backgrounds (i.e. brick walls, pillow case), dark shadows in the images, hand not centered, and some of them were rotated by 90 degrees when they were opened.

All the images in the dataset were resized to 100x100 pixels and the orientation of the images were fixed by examining their Exchangeable Image File (EXIF) data.

Another issue was how to account for different skin tones in the images. Red, green, and blue (RGB) images would have not been viable anymore because the training may not account for enough variation of skin tones. Thus, gray scaling the images was the next step to try to mitigate this issue. However, with the darker images, the gray scaling only made the hand more unrecognizable since we applied a threshold mask to the image in order to extract the hand only. Inspecting the properties of the image revealed that on the hue, saturation, and value (HSV) spectrum, the hand always had a significantly higher saturation than the background even on a background where the color closely matched the hand's skin tone.

At first, a blob detection was used to try and crop out the hand from the background, however, there was too much noise that was picked up by the detector that the blob cropped in pixels that were not the hand, sometimes even the shoulder which rendered the blob detection useless.

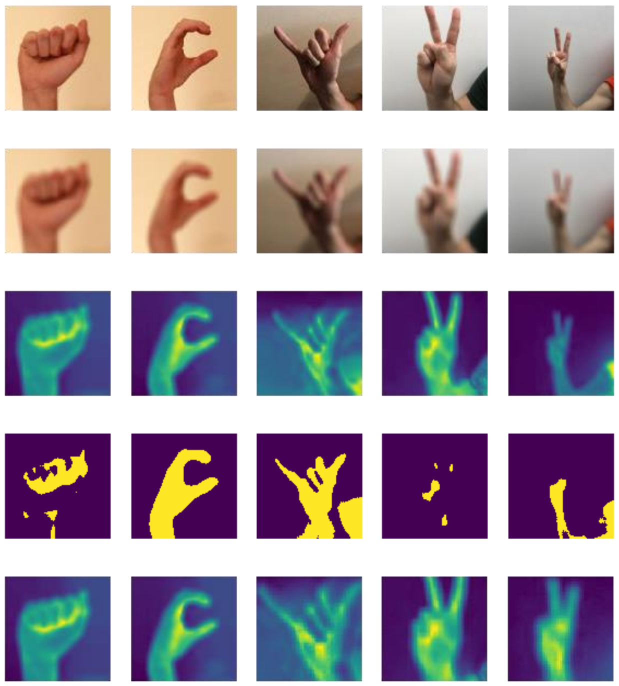

Fig. 1. Each row of images is a different step in the preprocess starting with the second row. From top to bottom:

(1) The original RGB images.
(2) Apply a box blur of 2.
(3) Get saturation of images.
(4) Threshold saturated images.
(5) Crop images.

Through some research and reading what previous people have tried on the internet, the following steps outlined in Fig. 1 were made for preprocessing the dataset. In order to crop out the shoulder from images, a box blur was first applied to the original RGB image to get rid of as much noise on the image as possible. Get only the saturation of the image. Take a 10x10 pixel window in the middle of the 100x100 pixeled image. Under the assumption that the hand is in the center on the image, the 10x10 window will be comprised of mostly the hand. By finding the mean of the window and using that as the threshold value, a rough outline of the hand can be found. However, this does not work for all the images, as seen in the 4th and 5th images. There is almost no discernable hand shape in the resulting Boolean image. From the 10x10 pixel window, slowly increase the size of the window until the number of pixels on the edges of the window meet a ratio of false and true pixels, 2.5, to signify that the three other edges of the image are the wall and the fourth edge will have the arm in it so it would have to be true values there.

The resulting images are cropped with only the hand in the center and then resized to a 100x100 pixels ready to be trained or tested on. This worked very well in most cases, as shown in Fig. 1. The shoulders are cropped out and the hand makes up most of the image now. This preprocessing step depends on the hand being centered in the image to work correctly, otherwise; the cropping method will give poor results.

B. *Feature Extractions*

An edge histogram was used as a feature. This was a recommendation from the instructor. Using a Gaussian blur to reduce more of the noise in the images. Sobel edge detection was performed in the x and y dimensions to calculate the type (horizontal, diagonal, and vertical) and location of the edges. The resulting image is partitioned into 9 equal sections where the x and y edges are averaged and then placed into an 2x9 array.

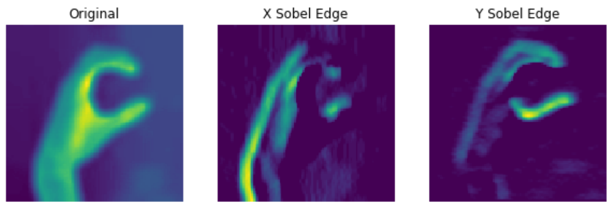

Fig. 2. The three images above illustrate how the edge histogram feature extraction worked. From left to right, that is the original image after preprocessing with an added Gaussian blur to reduce more noise. The following two images are the Sobel edge detection on the x and y dimensions respectively.

C. *Random Forests*

A random forest was decided during the initial development because the team had more experience with it. From the preliminary testing, it was decided that a random forest would be a backup plan if the CNN did not give higher accuracy results.

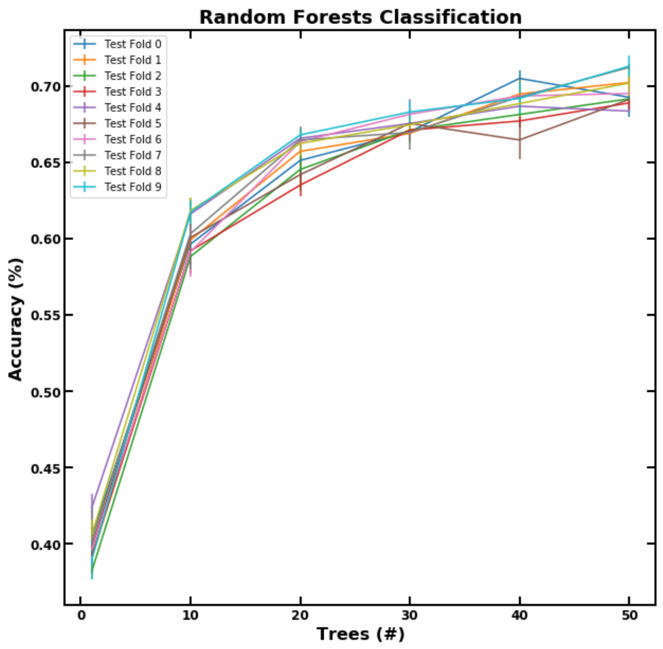

Fig. 3. A random forest classification of 50 trees resulted in approximately 70% accuracy when tested on the class' dataset. 10 testing folds were performed and increasing the number of trees above 50 did not result in a better score so it was maxed out at 50 trees. "entropy" was used for the criterion; the number of features was set to 50.

D. Convolutional Neural Network

Before implementation, the following commands must be ran in the Anaconda Prompt terminal:

1) *conda install -c pytorch -c fastai fastai*
2) *pip install fastai*

Separated all the data into a file structure consisting of test, training, and validation data. To load in the datasets, an image data bunch function from the “fastai” library was utilized. In which we passed in the file path to the training, testing, and validation directories respectively. The function separates all the data that was passed in into a single object that can be parsed and used by CNN.

To train the CNN, the data was passed in, defined the deep residual network (ResNet) model, and choose to stay with the default error metric which was “error rate”. The ResNet model that was decided upon was “resNet34.” Since the CNN is already pretrained, it took very few iterations to adapt its functionality to learning how to identify ASL images. The CNN was trained for 3 epochs while frozen. After that, the CNN was unfrozen, allowing the parameters (weights) of the networks to adapt to our specific training data for 2 epochs where the learning rate ranged from 0.001 to 0.0001.

The new model is exported to a save file to be used later for testing. The testing is based on the guidelines laid out by the instructor. With the current model, an accuracy on the entire alphabet, excluding “j” and “z”, resulted in a 88% accuracy while the “a” and “f” testing set gave a 98% accuracy.

III. Experiments

The following section is separated into 4 parts: Preliminary Experiments, Building Neural Net from Scratch, Transfer Learning CNN, and Avoiding Overfitting.

A. Preliminary Experiments

Before a CNN classifier was decided or even considered. A k-nearest neighbors (KNN) and random forest classifier was used on the small dataset that the team contributed. The labeled data was shuffled and split into a training and testing set. Different combinations were tried where the training comprised of everyone except for one person because that last person’s dataset would be used for testing.

For the KNN, all that was changed was the K-value, leaving the other parameters to their default states and values, as the K-value increased from one, the accuracy decreased significantly. The maximum K-value that was tested was 10 and it gave the lowest results consistently. The random forest on the other hand, showed promised to be the best classifier for the ASL image labeling. As seen in Fig. 3., the random forest reached approximately 70% accuracy when tested on the entire class’ dataset. As mentioned before, the criterion was “entropy”, 50 trees, and a limit on the number of features was 50 to prune the thousands of features that the random forest algorithm deemed important to the classification. The goal of this limit to avoid overfitting and hopefully all the Sobel edge histogram feature to shine. The accuracy seemed to plateau around 50 trees.

B. Building Neural Net from Scratch

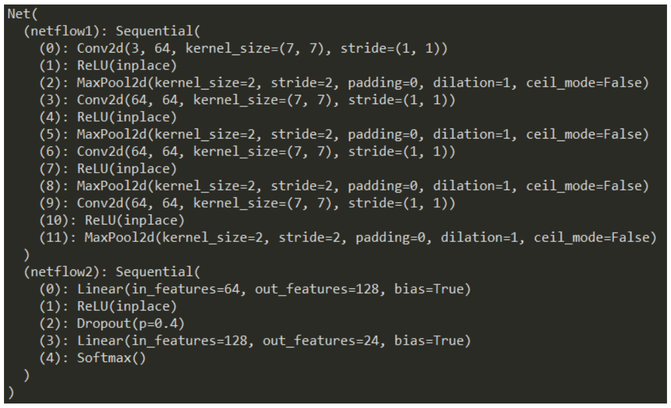
```
Net(
  (netflow1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(1, 1))
    (1): ReLU(inplace)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 64, kernel_size=(7, 7), stride=(1, 1))
    (4): ReLU(inplace)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(64, 64, kernel_size=(7, 7), stride=(1, 1))
    (7): ReLU(inplace)
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (9): Conv2d(64, 64, kernel_size=(7, 7), stride=(1, 1))
    (10): ReLU(inplace)
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (netflow2): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): ReLU(inplace)
    (2): Dropout(p=0.4)
    (3): Linear(in_features=128, out_features=24, bias=True)
    (4): Softmax()
  )
)
```

Fig. 4. This is the neural net definition that was used.

Beginning with a CNN structure as dictated by a similar problem from a similar project someone has done [1]. This original net does not have weights provided but reached 85.5% accuracy on testing data from the Sign Language MNIST dataset. There are several key differences between this dataset and our dataset, namely number of datapoints, size of the images, and the MNIST dataset is grayscale only. The MNIST dataset has about five times more data, that is four times smaller than our current dataset, additionally, our current dataset has three times the data channels. Iterating through several architecture changes, we found the best results in a slightly deeper net, with larger convolutions at every layer. The goal with these modifications was to better take into account the increased information per image and ignore the increased background noise.

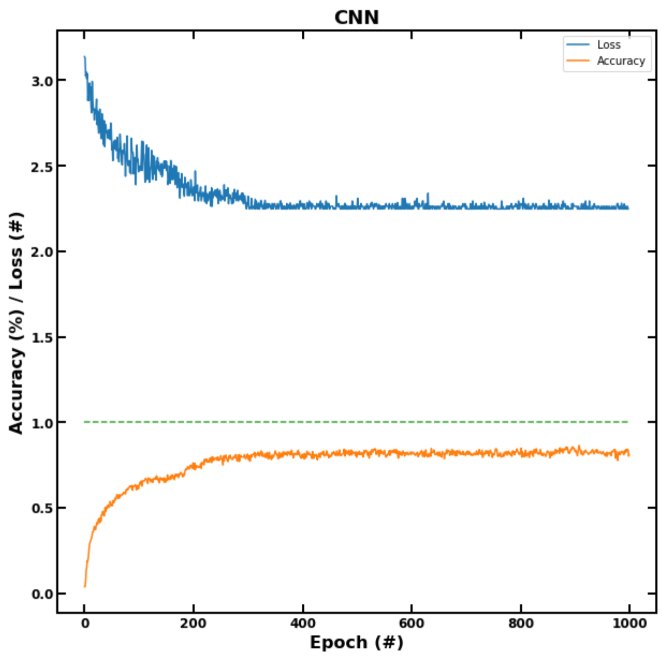

Fig. 5. Using a learning rate of 0.0001, a mini-batch size of 64, and 1000 epochs resulted in an accuracy of 85.5% plateau near the 250 epochs. Running it at 1000 epochs did not change the accuracy.

The hyperparameters of learning rate and batch size were also considered with a wide variety tested to find the optimal midpoint between speed and functionality. Learning rates from 0.01 to 0.00001 were tested and 0.0001 was found to be the highest functional value. Batch size was tested between 500 and 64. Smaller batch sizes seem consistently better but going any further would cause the network to train at an unacceptably slow rate. Tests were stopped at 64, which is what is currently used.

The final consideration was for data normalization. Several standard methods were employed, including per pixel mean normalization, min-max normalization, and L2 normalization. All standard methods tested had a strong negative effect on the learning ability of the network and were thus excluded from the final product.

The goal in running the neural network for 1000 epochs was to hopefully discover the global optima [3]. However, it was a futile effort. Reflecting on the graph of Fig. 5., not only does the epoch play a role, but varying the learning rate may have increased the chances of finding the global optima.

C. Transfer Learning CNN

After implementing the CNN organically and testing various architectures, we found it difficult to pinpoint exact correlations between network architecture and performance. Though some trends between network depth and classifier accuracy were noticed, the micro parameters of the network such as number of convolutions channels at each layer along with corresponding activation functions were problematical to identify. To alleviate these issues and improve overall accuracy of our training algorithm, we decided integrate transfer learning into our machine learning pipeline.

The deep learning architecture of which our design is transferred learned is the ResNet CNN. In general, ResNet address the issue of vanishing gradient which states that as networks become too deep, the model weights in the early layers cannot update correctly due gradient error converging towards zero. However, without a deep network, there may not be enough tunable parameters to tailor a network to a specific problem. With ResNet this no longer becomes an issue.

In the experiments of our hand designed neural network we noticed that smaller learning rates around 1e-4 were more effective at training the network. Therefore, we applied the same rate to our ResNet based system. After five epochs of utilizing the widely accepted one cycle learning policy, the network was completely trained with adequate results.

We tested our CNN against an easier dataset of A and F images in which it yielded an accuracy of 98% and a harder dataset of all the ASL letters in which it yielded an accuracy of 88%.

D. Avoiding Overfitting

To avoid overtraining on the data, a chunk of the given dataset was separated and never given to the model to train on. To further combat this problem, the model was train on data that was swapped in and out respectively from the training and testing pool. Each batch of images that were swapped in each run were hand-picked to ensure that there was no overlap between the images in the training and testing datasets. Training mages were also augmented with random combinations of rotation, zoom, and translation to help the model see an increased variety of images to prevent overfitting [5].

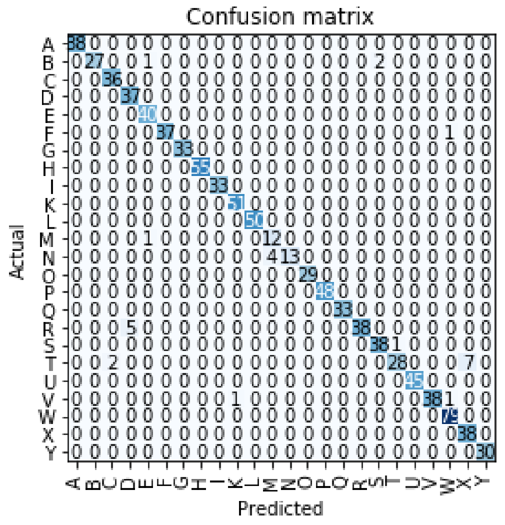

Fig. 6. Confusion matrix of resulting outputs.

As a final attempt, the team created and pulled in other images from the internet to test on ensuring that the model did not overfit and still gave a consistent score.

As seen in Fig. 6., the outputs of the model when tested against the class' dataset are very accurate. This confusion matrix was generated to visualize what letters the model was not accurately labeling and forces more examination and inspection to determine the reasons why that may have been the case. This matrix is another indicator for overfitting or underfitting because it shows exactly which letters are getting

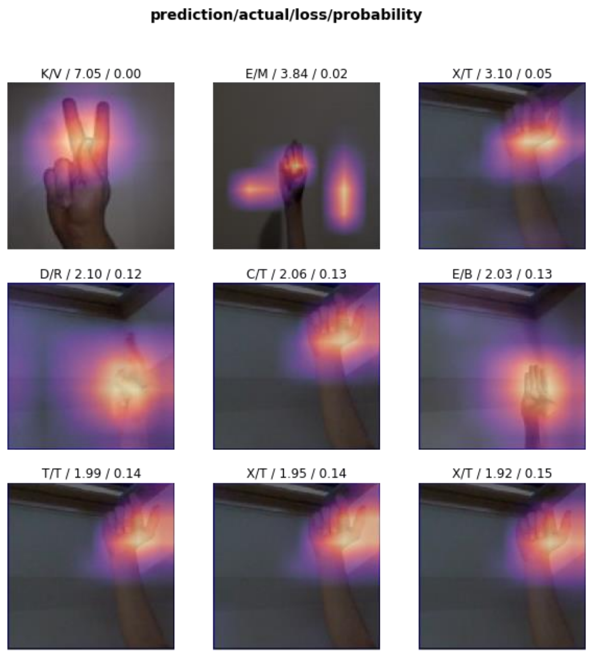

Fig. 7. The top 9 losses with heat maps according to the model.

mislabeled and what they are being labeled as.

Fig. 7. served as another visual guide to ensure no overfitting or underfitting. The model was able to recognize the area in which the hand was located, and from there the predicted letter. The loss value and prediction percentage associated with our most missed letters showed that the model had a low confidence when predicting the most missed images.

IV. Conclusion

To deal the complexity of identifying and correctly labeling ASL images, a deep neural network is required, and more varying data needs to be used to train on and test with. In Fig. 3., the random forest preformed with only a maximum accuracy of approximately 70% and even tweaking the parameters did not increase this accuracy. Either more useful features needed to be extracted or a different model needed to be used. This is where the CNN comes in. As seen in Fig. 5., the CNN that was created from scratch barely reached 85.5% accuracy and was only trained and tested on the limited provided dataset. This was the next bottleneck and after trying out different epochs, batch size, and learning rate, the accuracy did not increase and always plateaued to a maximum score of 85.5%. Thus, another solution had to be used. When using the "Fast.ai" and the ResNet to use transfer learning in order to solve this problem, the accuracy and time to train the model took almost no time and performed with a consistent 88% accuracy. This is the highest that the accuracy has ever been, illustrating the power of transfer learning in deep neural networks and its far-reaching applications to recognize more than just ASL images.

Acknowledgment

To the group of people who have dedicated themselves to creating the "Fast.ai" deep learning library. We, TRM, thank you for all that you have done.

Thank you, Dr. Zare and Connor, for a wonderful semester. See you in the Fall for the graduate version.

References

[1] Jain, R. (2019). *Deep learning using sign langugage | Kaggle*. [online] kaggle.com. Available at: https://www.kaggle.com/ranjeetjain3/deep-learning-using-sign-langugage [Accessed 20 Apr. 2019].

[2] P. R. Ruiz, "Understanding and visualizing ResNets," *Towards Data Science*, 08-Oct-2018. [Online]. Available: https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8. [Accessed: 20-Apr-2019].

[3] S. S. 1395550283894582, "Epoch vs Batch Size vs Iterations," *Towards Data Science*, 23-Sep-2017. [Online]. Available: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9. [Accessed: 21-Apr-2019].

[4] Xiaojie Jin, Yunpeng Chen, Jian Dong, Jiashi Feng: "Collaborative Layer-wise Discriminative Learning in Deep Neural Networks", 2016; arXiv:1607.05440

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren: "Deep Residual Learning for Image Recognition", 2015; arXiv:1512.03385