A Machine Learning Model for Classifying Hand Signs

Sebok, April and Rahimi, Mohammad
Section #13302 12/04/2023

***Abstract*—The objective of this project was to classify American Sign Language (ASL) characters. A transfer learning approach with convolutional neural network (CNN) was used to train the model on a total of 8443 pictures of nine different hand signs. The model performed in training and validation sets with 99.9% and 100% accuracy respectively. The model is capable of handling mislabeled and uncategorized data as well. This capability was achieved through defining a threshold function to deal with uncertainty in the data. All in all with a combination of data preprocessing and fine tuning the parameters of the problem the aforementioned accuracies was attained proving the robustness of the CNN approach.**

***Index Terms*—feedback, PID control, RMS error, Ziegler-Nichols tuning**

I. Introduction

THE objective of this project is to train a neural network model capable of classifying American Sign Language Characters in the form of 9 different sign pictures. The pictures were captured from hands of the students participating in the class encompassing various genders and ethnicities. The dataset comprises 8443 images exhibiting a wide array of backgrounds, orientations, proportions (relating the hand size to the background), as well as different angles and alignments. These diverse characteristics present challenges that traditional classification methods such as SVM struggle to effectively address.

In this scenario that we are dealing with such complexities in conjunction with limited dataset that might not be sufficient for which to train the model adequately, we decided to implement a transfer learning paradigm. Transfer learning is creating a high performance learner for a specific domain from a related source domain[1] in other words, we take the features that are learned from one related problem and leverage them on the current problem. In this light, we implemented a pretrained deep learning neural network with depthwise separable convolutions named as Xception or "ExtremeInception"[2]. This model have modules conceptually similar to conventional convolutional feature extractors, but capable of learning richer representations with less parameters which consequently yields better performance[2]. Hence it was our final choice to trained our model with.

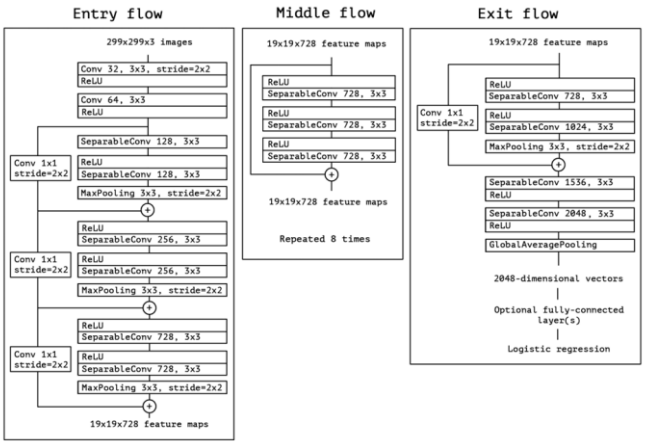


Fig. 1. The architecture of Xception is composed of three flows in which all convolution and separable convolutions are followed by batch normalization.[2]

TABLE 1
NUMBER OF PARAMETERS

Total params	Trainable params	Non-trainable params
21,484,559	10,101,423	11,383,136

II. Implementation

Model structure

The architecture of this model is demonstrated in figure 1. In this model the data goes through 3 general flows with the middle flow repeated 8 times. All these flows have 14 modules with 36 convolutional layers structured in them in total. All of the modules except for the first and the last one have linear residual connections around them. In order to tailor this model for our purposes we first added an input layer of shape 220×220×3 to feed the images with the same size. Then we attached a flatten layer followed by two dense layers to gradually decrease the number of nodes from 2048 to 9 to represent the 9 different hand signs. Rectified linear unit activation function were used those layers except for the last

layer that we used SoftMax function as the activation function. We utilized dropouts for added dense layers to avoid overfitting. Additionally to avoid overfitting and reduce the computational expenses we froze the first 100 layers of the model and made the rest trainable. Note that we looked at the changes of training and validation performance as the main determinant of overfitting. If the performance of training increased without increasing that of validation. We stopped and changed the parameters. We froze the first 100 layers because they were mostly responsible for extracting features such as edges that were common in both problems. despite being such a complex model less than half of the parameters sum to 10101423 were needed to be trained which is one of the advantages of transfer learning. The details of parameters is illustrated in table 1. Finally we transferred the features extracted by Xception model from ImageNet dataset [3] to efficiently tackle the current problem. ImageNet is a dataset composed of 1000-class single-label pictures which makes it appropriate for our purpose.

Data structure

The input data, initially flattened to (27000, 8443), underwent a reshaping process to revert to its original shape of (8443, 300, 300, 3). This transformation was crucial to prepare it for resizing into a more manageable size, accommodating the constraints of accessible computational resources. Employing Keras' resizing function yielded a smaller data size compared to OpenCV's resizing function. Downsizing achieved through the use of bilinear interpolation. Then we turned the label data into categorical format which we found out yields higher confidence for assigning a class to a image compared to the initial format. To streamline data handling, TensorFlow's tf.data.Dataset was employed to create a source dataset comprising the reshaped data and labels. This streamlined various data operations, including shuffling and splitting, simplifying the overall process. Subsequently, the data underwent shuffling and were batch-fed into the model, both of which impacted the model's performance positively. It's noteworthy that all data augmentation processes were handled separately, aiming to distribute computational loads across different tasks, ensuring lighter computational demands for each specific operation. We saved this data for training.

Training and Evaluation

In the training code, we loaded the previously resized data and subsequently performed a shuffle them along with the labels. Following this, we partitioned the data into an 80% training set and a 20% validation set. We trained the model with data batches as this allow the model to update the weight based on a range of different data which in turn increased the generalizability of the model. For optimization, we employed the Nadam solver with small adjustment on the parameters and utilized categorical cross-entropy as the chosen cost function. Our evaluation of performance relied on loss as the primary metric. The model performed with 98.2, 97.4 accuracy on the training, validation set respectively.

Hard test set implementation

To address uncategorized data, we examined the output of the softmax function in the final layer. If the model's confidence surpassed a predefined threshold, we accepted the predicted label. However, if the certainty fell below this threshold, we reassigned the label to -1, indicating that the data fell into the category of uncategorized data. This approach allowed us to effectively handle instances where the model's confidence in its prediction was insufficient.

III. Experiments

Model performance

Initially, we began our experimentation with the ResNet50 model but upon comparison with Xception, we observed that Xception demonstrated superior performance aligned with our specific objectives. As a result, we proceeded with further experiments utilizing the Xception model due to its better performance in meeting our defined criteria.

Effect of Data

Managing the data intricacies—ranging from sample quantity, size, format, to batch configurations—proved to be the pivotal factor impacting performance. Initially, using the 300×300 resolution with less than half of the available data and batch sizes of 1 yielded approximately 50% accuracy. To leverage the entire dataset, we attempted data resizing. However, using OpenCV's interpolation function, the highest resolution sustainable without system limitations was 120×120, achieving around 70% performance. Shuffling the data and loading batches with a size of 8 notably enhanced accuracy. Shuffling played a crucial role, diversifying the data from sample to sample. Batch loading further enriched training by exposing the model to groups of varied signs, preventing overfitting to specific sequences of similar signs. Adjusting batch sizes presented a trade-off with image resolution. To accommodate larger batches while maintaining a resolution suitable for processing, we segregated data preprocessing into a separate file. This allowed us to handle a resolution of 140×140 with a batch size of 8, boosting performance to 95%. Though a notable increase, this resolution was still below half of the original image size. To explore higher resolutions within memory constraints, we adopted TensorFlow's Keras resizing function, enabling the model to process images at 220×220 resolution with batch sizes of 73. This led to a remarkable performance leap to 97%. The rationale behind the batch size choice was twofold: to augment the model's overall adaptability while utilizing the highest resolution feasible. Setting the batch size to close to multiples of 9 was strategic, due to a higher chance of encompassing all classes in one batch while minimizing overemphasis on specific classes. We chose the batch size that maximized the validation performance while minimized the difference between validation and training performance.

Effect of the model structure

We leveraged transfer learning by adopting a pre-trained model, albeit initially designed for a different dataset. To tailor it to our specific needs, we made crucial modifications. Initially, we reduced the node count from 2048, which was originally optimized for a dataset encompassing 1000 classes, down to 9 nodes, representing 9 classes in our case. This reduction was achieved by introducing a single dense layer of 27 nodes. Further enhancing this adaptation, we introduced an additional transition layer, first reducing the node count from 2048 to 300 and subsequently down to 27. This adjustment yielded a slight improvement in performance. As we had already achieved a commendable 93% performance level, we maintained this configuration, finding it to be optimal for our objectives without further alterations. Further, we improved the performance on validation set by introducing dropouts to two additional dense layers added to the model and adjusting them so the difference between train and validation is minimized.

Another factor in the model was selecting the number of trainable layers it was a tradeoff more trainable layers increase the performance as well as the chance of overfitting. We found the best performance is achieved by not training the first 100 layers of the model.

Effect of NADAM optimizer parameters:

We started with an initial step size of 0.005. When we changed the solver to Nadam and kept all the other variables constant we had a 1% increase in accuracy. Finally, we attempt to increase the performance as much as possible with fine tuning Nadam optimizer parameters. The final values are as follows: the learning rate of 0.0001, beta 1 =0.96 and beta_2=0.998. We also assessed the effect of the learning rate scheduler on our performance we didn't find any specific improvement.

Final stage:

When we were sure of the model parameters and performance in terms of accuracy and generalizability, we kept the same configuration and trained the model again with 90-10 percent ratio for training and validation to expose the model with more data. This the confidence of assigning each sign to each class which consequently improved the performance for the hard test set. Further we found the use of categorical labels and consequently categorical cross entropy as the cost function remarkably improve the model confidence in predictions compared to not categorical labels and sparse categorical cross entropy as the cost function. The reason for this is the fact that the structure of categorical cross entropy whereas sparse categorical cross entropy yields a smaller loss value and higher max value of SoftMax function for each sample and consequently higher confidence. The accuracy of the final model is illustrated in figure 2. As shown with measures implemented during the training the training and validation performance were kept close to each other throughout the training, ensuring the generalizability of the model while maximizing the performance. This also proves the stability robustness of the model. Figure 3 shows the performance of the model for the validation set. As can be seen the signs for letter "D" were most confused by "I" 3 times and "H" for "G" 3 times as well.

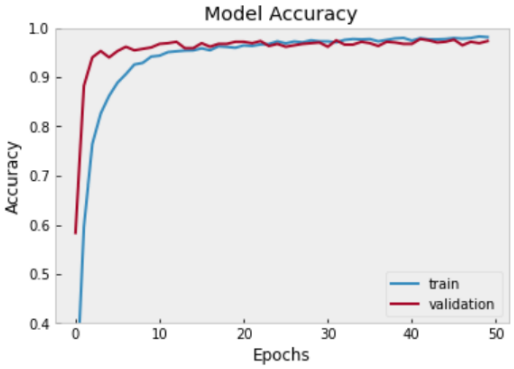
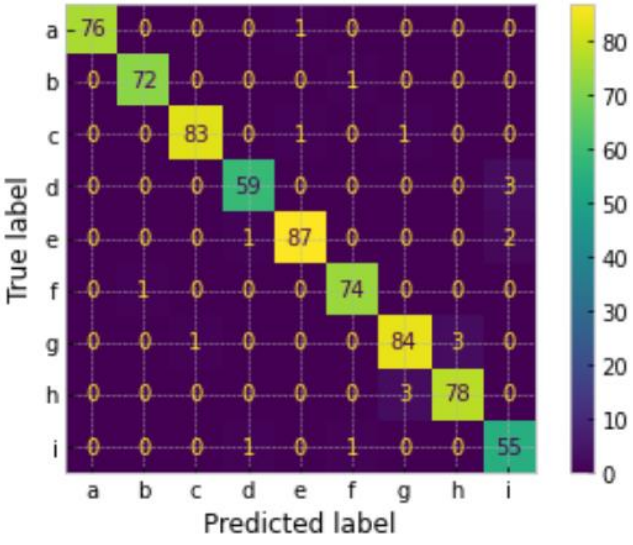


Fig. 2. The accuracy of the model during the training.

Class

	precision	recall	f1-score	support
0	1.00	0.99	0.99	77
1	0.99	0.99	0.99	73
2	0.99	0.98	0.98	85
3	0.97	0.95	0.96	62
4	0.98	0.97	0.97	90
5	0.97	0.99	0.98	75
6	0.95	0.95	0.95	88
7	0.96	0.96	0.96	81
8	0.92	0.96	0.94	57
accuracy			0.97	688
macro avg	0.97	0.97	0.97	688
weighted avg	0.97	0.97	0.97	688



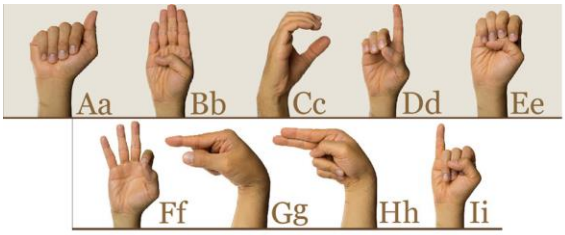


Fig. 3. Classification report and confusion matrix of the validation set.

Conclusion

The best performance achieved by the model was 98.2% on the training set and 97.4% on the validation set. Factoring out prediction below 80% confidence only drops the validation performance by less than a percentage point which was 96.8%. This shows the capabilities of transfer learning approaches. This project shows even with limited computational resources and data a good performance can be achieved if the right strategies based on the needs and limitations are implemented. We found the first and foremost contributing factors is data augmentation methods how to handle your data, so it needs less storage and ram while keeping most of the necessary details. How to add diversity to your training by shuffling and batch feeding your data to the model the choose of right batch sizes related to your data.

Second we found choosing the right model for the data have a crucial role especially when it comes to transfer learning the knowledge of on what data this model was trained previously and how can I leverage the pretrained parameters to my need. For example, we found that. The first layers of Xception model were responsible for finding low level features such as edges so we didn't train them.

Third, was the importance of avoiding overfitting, this can be a disadvantage of implementing transfer learning your model might be overly qualified for the purpose of your project. Having that knowledge and how to tackle it by adding dropouts limiting the number of training layers, increasing batch size, etc. is highly beneficial.

Finally, after you are sure you have a generalizable model that doesn't overfit it is time to explore the effects of different cost functions and optimizer and their parameters to make the model as optimized as possible, after this it might be the time to increase the training to validation ratio and train your model all over again.

References

[1] Weiss, Karl, Taghi M. Khoshgoftaar, and DingDing Wang. "A survey of transfer learning." Journal of Big data 3.1 (2016): 1-40.

[2] Chollet, François. "Xception: Deep learning with depthwise separable convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[3] O.Russakovsky,J.Deng,H.Su,J.Krause,S.Satheesh,S.Ma, Z.Huang,A.Karpathy,A.Khosla,M.Bernstein,etal. Imagenetlargescalevisualrecognitionchallenge.2014.