American Sign Language Classification using Transfer Learning

Adam Horton
Department of Electrical and Computer Engineering
University of Florida
Gainesville, FL
adam.horton@ufl.edu

Justin Beauchesne
Department of Computer and information Science and Engineering
University of Florida
Gainesville, FL
beauchesenej@ufl.edu

Michael Tung
Department of Electrical and Computer Engineering
University of Florida
Gainesville, FL
michael.tung@ufl.edu

***Abstract*—The purpose of this project is to collect and classify the first nine American Sign Language (ASL) letters. This training is a 2-step process that starts with transfer learning and ends with fine tuning the model. Through transfer learning, we have reduced the computational overhead typically associated with training deep learning models from scratch, while maintaining high accuracy. This approach not only optimizes training time but also improves the model's ability to generalize across diverse datasets. The following fine-tuning stage helps reduce overfitting and improve overall accuracy by adapting the pretrained model to our dataset.**

Keywords—Transfer Learning, Deep Neural Networks, Convolutional Neural Networks

I. INTRODUCTION

American Sign Language (ASL) serves as a primary mode of communication for many in the deaf and hard-of-hearing community. However, its recognition via 2D images has historically posed a challenge due to the complex variations of hand gestures [1]. Recent advancements in machine learning, particularly in the realm of transfer learning and deep neural networks, have opened new routes for creating more accurate and efficient ASL recognition systems than ever before. Previous efforts using Convolutional Neural Networks (CNNs) have yielded accuracy scores of 80% [2] and 98.98% [1] on the entire alphabet. Transfer learning, along with data augmentation, has been identified as an area where accuracy improvements could be made [2], and will guide our approach. While it is still challenging to understand complex gestures that require movement, it is possible to classify static images of ASL, such as the alphabet.

The goal of this project is to classify the first nine ASL letters in the alphabet. The process included gathering data, data processing, researching suitable training methods, and implementing a deep learning model architecture for training.

II. IMPLEMENTATION

Model Architecture

Of the many forms of deep learning and neural networks, we focused on Convolutional Neural Networks (CNN), as these model types specialize in spatial data, such as images. The group decision leaned towards a transfer learning approach of using a model pre-trained on ImageNet dataset [3]. The chosen architectures VGG [4], Xception [5], and EfficientNet [6] are known for their accuracy in image classification. These models were selected based on performance, model parameters, and in some cases based on how modern they were, all from the Tensorflow Keras application library [7]. As such we used VGG as a model that was small and quick, Xception that was in the middle as not too fast and not too large, and the most complex models coming from EfficientNet. Regarding EfficientNet, they offer many variations, we opted to try B7 as a more complex model and V2B1 as a more modern, but still fairly complex, model. The model architecture for each model mentioned above is different but the overall model architecture used for transfer learning and fine tuning is the same for each. The first step was to import the base model, the pretrained model, without the output layer. It's also important to freeze the base model so that we don't change the pretrained data. We then attached our own input as 100 by 100 with 3 color channels for the images, with a preprocessing layer to convert the data to a desirable shape for the model. After the base model, we attach our own output layers, this consisted of a pooling layer to average the output, a dense layer with 'leaky-relu' [8] and L2 regularizer to reduce the connections, a dropout layer to reduce overfitting, and a final dense layer with 'softmax' and L2 regularizer to return our desired output of 9 classes.

The model architecture for each model mentioned above is different but the overall model architecture used for transfer learning and fine tuning is the same for each. The first step was to import the base model, the pretrained model, without the output layer. It's also important to freeze the base model so that we don't change the pretrained data. We then attached our own input as 100 by 100 with 3 color channels for the images, with a preprocessing layer to convert the data to a desirable shape for

the model. After the base model, we attach our own output layers, this consisted of a pooling layer to average the output, a dense layer with ‘leaky-relu’ [8] and L2 regularizer to reduce the connections, a dropout layer to reduce overfitting, and a final dense layer with ‘softmax’ and L2 regularizer to return our desired output of 9 classes.

Training Process

As mentioned, the training process is a 2-step process of transfer learning then fine tuning [3]. For transfer learning we don't want to change the pretrained weights, just the output layers we added after the base model, which is why the base model gets frozen. We compiled the model with a Nadam [9] optimizer (the learning rate varies based on the base model used but it is about 0.001 - 0.0001). With the compiled model we train with augmented data, which helps generate additional (noisy) data for our small dataset to reduce overfitting. We also attach a model checkpoint callback to keep the model with the smallest ‘val_loss’. This training has run for 100 epochs.

The fine-tuning stage is run using the best model returned from the transfer learning callback. This best model is first unfrozen, then compiled with Nadam [9] optimizer and, typically, a smaller learning rate. This fine-tuning process updates the model weights as whole to improve total performance, thus fine tuning the model to our data specifically. This model is set up to return a final model that is determined based on another similar callback but focuses on ‘val_accuracy’ instead.

Categorical cross-entropy was the loss function for our model. It provides a measure of the difference between the predicted classes and the actual class labels in our multi-class ASL classification task. This loss function is well-suited for problems where each class is mutually exclusive, ensuring that the loss is minimized when the prediction is close to the actual label.

Our computational resources included 2 A100 GPUs, which allowed us to comprehensively experiment with transfer learning and fine-tuning approaches. The A100's capabilities enabled us to train our models efficiently because the parallel processing power can drastically reduce training time.

III. Experiments

In our empirical investigations, we systematically evaluated the performance of the different transfer learning models by experimenting with different hyperparameters and augmenting our data set with external sources and image rotations. During hyperparameter tuning, we modified the base models used for the transfer learning, the strength of regularization through dropout layers and L2 regularizes, and the learning rate for the Nadam optimizer. After tuning the models to achieve the best performance, we computed various performance metrics and analyzed the results.

Data Augmentation

All of the models we chose for transfer learning are deep convolutional neural networks. These models require lots of data in order to fully learn the patterns in the training images. To ensure that our models had plenty of data to learn from, we generated additional images from the given training set by using Tensorflow Keras’s ImageDataGenerator tool. This enabled us to rotate and flip the images, which has been shown to improve performance in convolutional neural networks used for image classification [10]. All experiments were conducted using the entire augmented dataset, split into training, test, and validation.

Comparative Analysis and Refinement

In the comparative analysis, we compared the performance of the VGG, Xception, and EfficientNet architectures. Initial performance results indicate that EfficientNet demonstrates superior performance in terms of accuracy and computational efficiency. Its compound scaling technique allows for a higher degree of model complexity, which seems to be beneficial in capturing the nuances of ASL gestures. VGG is less accurate than the other two models but still able to maintain around 90% accuracy in the test set. Xception's middle-ground approach yields competitive accuracy with reduced parameters, making it a strong candidate for deployment in resource-constrained environments. EfficientNet requires more computational resources for training and fine-tuning but performs the best among all three models and will be further pursued to create our final model.

During the subsequent iterations of training the EfficientNet model, we adopted L2 regularization to mitigate overfitting by penalizing large weights. This ensured that our model maintained a balance between accuracy and broader generalization. In addition, we employed the Nadam optimizer [9], which significantly improved the convergence rate over traditional methods. We experimented with different L2 regularizer values to determine the one that yielded the best performance in the validation set.

During the transfer learning phase, we froze all the layers to keep the pre-trained weights, added a new output layer, and trained them with our augmented ASL dataset. These models were saved and transitioned to the fine-tuning step. In fine-tuning, we unfroze all layers, so that we can make a more subtle adjustment of the deeper network weights, further refining our model to acutely capture the nuances of ASL gestures. The training process was recorded across all four models and is shown in Fig 1.

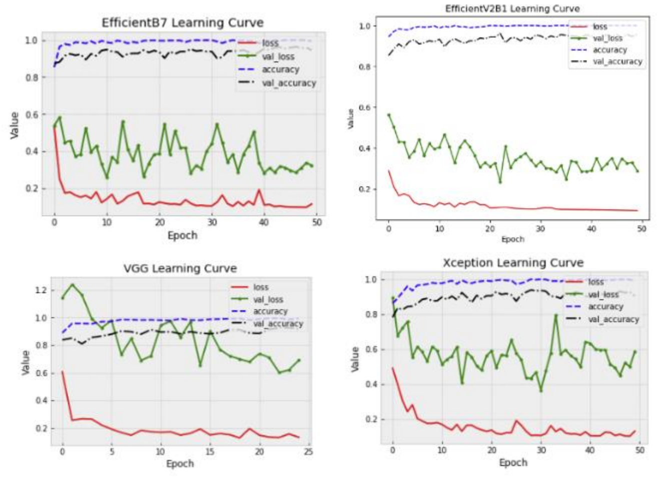

Fig. 1: Individual model learning curves during fine-tuning step.

Performance Metrics

Once the fine-tuned models were achieving their highest performance in the validation set, we evaluated their performance in the test set and the external test set consisting of 4,500 images (500 of each letter) from an online database [11]. Each model's accuracy scores are reported in Table 1.

TABLE I. MODEL ACCURACY SCORES

Model	**Train Accuracy**	**Validation Accuracy**	**Test Accuracy**	**External Accuracy**[1]
EfficientNetV2 B1	1.00	0.95	0.95	0.49
EfficientNetB7	1.00	0.95	0.96	0.53
VGG19	1.00	0.92	0.93	0.44
Xception	1.00	0.94	0.93	0.51

\

[1].External ASL images gathered from ASL Alphabet dataset [11]

Some notable conclusions can be drawn from the final accuracy scores. Firstly, all four underlying image classifiers perform nearly identically with the two EfficientNet models slightly outperforming VGG and Xception in test. Secondly, none of the models performed well in the external dataset, indicating that none of our models were as generalizable as we had hoped. The online dataset consists of images that are primarily backlit, and generally of lower quality than the images sourced from students in the class. Our models struggled to classify letters using these silhouette images, demonstrating that further training is necessary for a classifier to be utilized in a less-controlled environment. To achieve better results, we would need to retrain the model with a dataset that is more broadly representative of the types of ASL images it may encounter, including features such as varying backgrounds and skin tones, different levels of lighting, and different orientations.

IV. CONCLUSION

Through research and experimentation, our project approached ASL classification with a 2-step transfer learning and fine-tuning methodology. This was achieved through transforming the pre-trained, deep neural networks to work well on specific tasks, such as ASL recognition. With thorough experimentation out machine learning model has proven to be both successful and effective as classifying ASL characters 'A' through '. The implications of this work extend to real-time ASL translation applications and improved accessibility options for ASL users. Further research could explore incorporating dynamic gestures and using sequential models like RNNs or LSTMs.

V. REFERENCES

[1] R. K. Pathan, M. Biswas, S. Yasmin, M. U. Khandaker, M. Salman and A. A. F. Youssef, "Sign language recognition using the fusion of image and hand landmarks through multi-headed convolutional neural network," *Scientific Reports*, vol. 13, 09 October 2023.

[2] S. Ameen and S. Vadera, "A convolutional neural network to classify American Sign Language fingerspelling from depth and colour images," *Expert Systems*, vol. 34, p. e12197, June 2017.

[3] TensorFlow Core, "Transfer learning and fine-tuning," [Online]. Available: https://www.tensorflow.org/tutorials/images/transfer_learning. [Accessed April 2024].

[4] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556,* 2015.

[5] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," *arXiv preprint arXiv:1610.02357,* 2017.

[6] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," *arXiv preprint arXiv:1905.11946,* 2020.

[7] Keras, "Keras Applications," [Online]. Available: https://keras.io/api/applications/. [Accessed April 2024].

[8] B. Xu, N. Wang, T. Chen and M. Li, "Empirical Evaluation of Rectified Activations in Convolution Network," *arXiv preprint arXiv:1505.00853,* 2015.

[9] T. Dozat, "Incorporating Nesterov Momentum into Adam," in *Proceedings of the 4th International Conference on Learning Representations*, 2016.

[10] S. C. Wong, A. Gatt, V. M. Stamatescu and M. D. McDonnell, "Understanding data augmentation for classification: when to warp?," *CoRR*, vol. abs/1609.08764, 2016.

[11] A. Nagaraj, *ASL Alphabet,* 2018.