Final Report

1st Cole Rottenberg
Electrical and Computer Engineering
University of Florida
Gainesville, USA
colerottenberg@ufl.edu

***Abstract*—A summary description of the contents of the report and your findings.**

I. Introduction

Research into the use of fundamental machine learning techniques in order to accurately classify images of American Sign Language (ASL) letters has been a topic of interest for many years. I began working with this limited dataset in order to explore the potential of using machine learning to classify images of ASL letters. The dataset consists of 9 classes, each corresponding to a different letter in the ASL alphabet. The dataset is relatively small, with only 1440 images in total with an 80/20 split for training and testing. The images are 100x100 pixels in size and have 3 RGB channels. The goal of this project is to explore the potential of using machine learning to classify these images and to experiment with different techniques to improve the accuracy of the model. The first approach came with exploring the use of basic convulation neural networks (CNNs) to classify the images. Built onto this, I experimented with handcrafted sobel filters to preprocess the images before feeding them into the CNN. The second direction taken was exploring the generational leap of using deep learning techniques from AlexNet, VGG16, and DenseNet121. Research by has shown that these models have been successful in classifying images in the ImageNet dataset [1]. After exploring the early research of using these models, I experimented with using transfer learning to classify the ASL images. The first approach with transfer learning involved using the Xception model and removing the last layer to add a new dense layer with 9 neurons to classify the images. The approach had a validation accuracy of only 10%. Dishartened by the results, I decided to experiment with using the other networks, landing on the VGG19, MobileNet, and DenseNet121. Eventually I read about the use of dense networks and feed-forward and decided to experiment with the DenseNet121 architecture [2].

II. Implementation

The implementation of the model was carried out using the Tensorflow and Keras libraries.

A. Data Preprocessing

The initial data comes to use as 1,440 images of 30,000 pixels. The images are 100x100 pixels in size and have 3 RGB channels. The first step in preprocessing the data was to reshape the images to 100x100x3, channels last. The next step was to seperate the images into training and testing sets. The training set consisted of 1,152 images and the testing set consisted of 288 images. All normalization was done internally in the model with batch normalization. The final step in preprocessing the data was to one-hot encode the labels.

B. Convolutional Neural Network

The model was entirely using the Tensorflow and Keras libraries. The model was built using the functional API of Keras allowing for more flexibility in the model architecture. The final model consisted of three blocks. The initial layer input was 100x100x3. The first block consisted of data augmentation layes within Keras: RandomFlip, RandomRotation, RandomZoom, and RandomContrast. The second block consisted of the DenseNet121 architecture with random weights. Normally, this decision would lead to overfitting due to the low sample size and overkill nature of large models. However, the data augmentation layers helped to prevent this. The final block consisted of a pooling layer, a dense layer with 512 neurons, and a final dense layer with 9 neurons. The final layer used a softmax activation function to classify the images.

C. Training

The implementation of the model took place through the use of the University of Florida's HiPerGator supercomputer. The model was trained using 2 NVIDIA Tesla A100 GPUs. The model was trained using the Adam optimizer with a learning rate of $1e-4$. The model was trained for 500 epochs with a batch size of 32. The model was trained using the sparse categorical crossentropy loss function. The model was trained using early stopping with a patience of 10 epochs. The model was trained using a validation split of 20%. The model was trained using a callback to save the best model weights.

III. Expirements

A. Sobel Filters

Before learning about how CNNs use kernel filters to learn features, I experimented with using handcrafted sobel filters to preprocess the images. I began by applying the sobel filters to the images with heightened contrast. The results were not as expected. The model was unable to learn any features from the images and the accuracy was 10%. It became evident that the model was unable to learn any features from the images and the sobel filters were not useful in this context.

B. Data Augmentation

The cornerstone of the model was the use of data augmentation layers within the model. Due to the lack of data, the model was prone to overfitting. The data augmentation layers helped to prevent this by creating new images from the existing images. The data augmentation layers consisted of RandomFlip, RandomRotation, RandomZoom, and RandomContrast. The results were significantly better than the model without data augmentation. The accuracy was 95% on validation data and over 99.9% on the training data.

C. Transfer Learning

I explored the use of transfer learning with ImageNet weights which initially proved to be unsuccessful. The first approach was to use the Xception model and remove the last layer to add a pooling and a dense layer of 9 neurons. The model was unable to learn any features from the images and the accuracy was no better than random guesses. I then explored other pretrained models such as VGG19, MobileNet, and ResNet121. The results were significantly better, however did not reach the desired accuracy. The final approach was to use the ResNet121 architecture with random weights. The model was able to learn features from the images and the accuracy was 95% on validation data and over 99.9% on the training data.

D. Hyperparameter Tuning

The use of the Adam optimizer with a learning rate of $1e-4$ was the best hyperparameter for the model. Within training, I observed that the model was able to accurately learn parameters up until 90% accuracy on validation data with ease. However, the model struggled to learn the final 5% of the data. The use of early stopping with a patience of 10 epochs helped to prevent overfitting. The model was trained for 500 epochs with a batch size of 32.

E. Model Architecture

The model architecture was the most important aspect of the project. The use of the DenseNet121 architecture with random weights was the best model for the dataset. In order to avoid the shortcomings of using large CNNs, such as the vanishing gradient problem, researchers used a technique to connect all layers directly with each other. This technique is called Dense Convolutional Network (DenseNet) [2]. Instead of distorting the feature maps through summation, the feature maps are concatenated. Every layer recieves the feature maps of all preceding layers as input. "Hence, the *l*th layer has l inputs, consisting of the feature-maps of all preceding convolutional blocks. Its own feature-maps are passed on to all $\mathrm{L}-l$ subsequent layers." [2]. In addition to the information preservation, the DenseNet architecture has an internal mechanism for regularization. Researcher concludes this benefit orginates from the fact each layer has access to the gradients of the loss function with respect to the model parameters. DenseNets were able to achieve state-of-the-art results on CIFAR-10, CIFAR-100, SVHN, and ImageNet datasets with relatively fewer parameters. The DenseNet proposes a different approach to connectivity between all layers. The model is constructed of multiple dense blocks, each consisting of multiple convolutional layers. Within a block the input is concatenated with the output of the previous layer but only the output of the final layer in a block is passed onto the next block. The researchers describe the architecture as such: "Consequently, the *l*th layer receives the feature-maps of all preceding layers, $x_0, x_1, ..., x_{l-1}$, as input: $x_l = H_l([x_0, x_1, ..., x_{l-1}])$." [2].

IV. Conclusion

From this project, I learned the importance of model architecture in the success of a model. The use of the DenseNet121 architecture was the most successful model for the dataset. The use of data augmentation layers within the model was also a key factor in the success of the model. The use of transfer learning was not as successful as I had hoped. The use of the DenseNet121 architecture with random weights was the best model for the dataset. The model was able to learn features from the images and the accuracy was 95% on validation data and over 99.9% on the training data. The success of the model was due the models ability to learn and pass feature-maps to each additional learning allowing for the model to combat the vanishing gradient problem.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012.

[2] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993