CPSC 585 - Spring 2023 - Group Project 1 - Review

Introduction

This review compares and discusses our team's project with two other teams' work. The tasks assigned to each group were to create convolutional neural network models to accurately predict handwritten letters presented in the Extended MNIST or EMNIST dataset. Furthermore, we were tasked to apply transfer learning to predict the results of the Binary AlphaDigits dataset, then construct a new network only on the AlphaDigits dataset for accuracy comparison. We will look at the other team's implementation of these networks and compare their methods and results to our own.

Comparing Methods

Group 12

EMNIST CNN

Method

First, we will look at this group's implementation of a CNN (Convolutional Neural Network) that can identify handwritten letters. The group used the EMNIST Dataset, which is already split into training, validation, and testing sets. This group decided on a single architecture for their network but applied two optimization approaches: Adam (Adaptive Moment Estimation) and SGD (Stochastic Gradient Descent). Their architecture consisted of a 28x28x1 input layer, which feeds into the first two-dimensional convolutional layer, which consisted of 32 filters, a 3x3 kernel size, a kernel and bias regularizer with an L2 regularization penalty of 0.000001, a default stride of 1 on the height and width, batch normalization, and uses the activation function ReLU (Rectified Linear Unit), with max pooling with size and stride of 2 height and width of 2. The next layer was a two-dimensional convolutional layer with 64 filters and the same parameters, except without pooling. The next layer was the same as the previous, but with 96 filters, transforming the data with flattening to be fed to a dense layer with 64 neurons with the same regularizers, activation function, and normalization, but also with a 0.2 dropout. Then, it was densely connected to a softmax activation function layer with 27 neurons to get a value for each digit used for prediction. When training the network, the group used the Adam optimizer with a learning rate of 0.001, then the SGD with a learning rate of 0.0005 and momentum of 0.9. Both were run over 20 epochs, used early stopping, and used sparse categorical cross entropy as its loss function, looking for the most accurate model training off of the training set and validating with the validation set. They then used the test set to display the testing loss and accuracy between each network training.

Similarities

Comparing their method to that of the method we used for our group, there were some similarities. We also used three layers of two-dimensional convolutional layers, with a densely connected network, after using the ReLU activation function, which feeds into the softmax layer. We both felt it was appropriate to use max pooling on the first convolutional network layer, with our reasoning that we would be okay losing some less significant pixels when training the first filters. We also used the same kernel size on our layers of 3 overall, as it is an odd number that avoids having hard-to-pad results, and from running the model, seemed to have diminishing accuracy returns as we increased it higher, along with the fact that training time took longer with a larger kernel, and hardware in Google Collab became more of a limiting factor. We also both used the ReLU activation function for our densely connected layers, as when testing different activation functions, most notably Leaky ReLU and some others, we did not see an increase in accuracy. We both also used early stopping as hardware was a limitation, so when it was likely the model was no longer learning, we could stop it and retry or run with other hyperparameters to achieve a better result.

Differences

The main differences between our architectures were some changes to the number of filters in the convolutional layers, even though both agreed to increase the number of filters as it fed to the next layer led to some marginal improvements. Another main difference was that we tested different pooling methods, resulting in higher validation accuracy when using average pooling for the other convolutional layers. We believed that max-pooling could cut out some of the noise, as when max pooling is applied, it takes the highest values out of the pooling size, then the average pooling would preserve what was left, as we would not be losing any more important information from the data, and therefore could learn better from it. This team also took advantage of l2 regularizers, but as we will see in the results section, the benefits of this are inconclusive from the current project.

Pre-Trained and Tuned AlphaDigits

Methods

The team loaded the images and labels from the Binary AlphaDigits dataset, which contains letters, but of a different size than what was used to train the previous model. The team loaded the images and labels and split them into a training and testing split of 20% testing data and 80% training. Using TensorFlow, they resized the data into 28x28x1 for use with the model trained off of the EMNIST dataset. For predicting the character from the Binary Alpha Digits dataset, this team utilized their EMNIST network by freezing its layers, making their weights unchangeable, removing the last two dense and softmax layers, then adding on two ReLu layers, a dense 128 neuron dense layer with l2 regularizers using the same hyperparameters as earlier, a 64 neuron dense layer with the same l2 regularizer, then lastly the 27 neuron softmax layer for the classification. They trained the new network using the Adam optimizer, with the loss function as the previous model, and for ten epochs.

Similarities

When comparing our tuned model to predict the new data, we also decided to freeze our network and add an additional dense layer using the same ReLU activation function.

Differences

Our group trained on 150 epochs compared to their 10, likely due to us having less of a hardware limitation from our own group's access to their own graphics card. We also only added one new dense layer and did not cut off any of our pre-trained layers except for the last softmax layer.

Group 13

EMNIST CNN

Method

Now coming to the group 13 implementation of a CNN (Convolutional Neural Network) that can identify handwritten letters. The group used the EMNIST Dataset, which is already split into training, validation, and testing sets. This group's architecture consisted of a 28x28x1 input layer, which feeds into the first two-dimensional convolutional layer, composed of 32 filters, a 3x3 kernel size, padding, a He uniform variance scaling initializer to draw samples from a uniform distribution, followed by batch normalization, a max pooling layer with pool size 2x2 and activation function ReLU (Rectified Linear Unit). The second layer was a two-dimensional convolutional layer with 64 filters and the same parameters. The third layer was the same as the previous, but with 128 filters, transforming the data with flattening and is fed to a dense layer with 256 neurons with the same kernel initializer, activation function, and normalization but with a dropout of 0.5. Then, it was densely connected to a softmax activation function layer with 27 neurons to get a value for each of the digits used for prediction. While training the network, the group used the Adam optimizer. They implemented 15 epochs, used early stopping with a patience of 3, and used categorical cross entropy as their loss function. They received a training accuracy of 97% and a validation accuracy of 94%. Furthermore, the test accuracy of this model was 93.99%.

Similarities

There are multiple similarities between the convolutional network architecture used by group 13 and the architecture used by our group. First, both architectures use 2D convolutional methods with a square kernel of size 3 to extract essential features from images. Both models use kernel initialization techniques (kernel_initializer) such as 'he_uniform' to initialize the weights of the convolutional and dense layers to facilitate convergence and improve model performance. Next, both architectures perform batch normalization after each convolution layer to improve the stability of the learning process and normalize the inputs to the next layer. Following batch normalization, the ReLU activation function is applied, which aids the model in learning complex patterns in the training dataset. Both the convolution network architecture use max-pooling before flattening to minimize the spatial dimensions of the feature maps and capture the essential features. Both architectures use the dropout after the flattened layer, which aids in preventing the overfitting of the model.

Differences

One of the key distinctions between our architecture and Group 13's architecture is that we have four Conv2D layers, whereas Group 13 only has three. The second difference in the architecture is that for the second and the third layer, we apply average pooling compared to max pooling used throughout three layers by group 13.

Pre-Trained and Tuned AlphaDigits

Methods

The team started by loading the BinaryAlphaDigs dataset and resizing the images using the resize_with_pad method. For the purpose of transfer learning, they fetched the model weights from an already saved file called ModelofGroup13.h5. They froze all the layers, trained the model on the new dataset, and received an accuracy of 90.19%. They added a new layer to this model, trained it again, and received an accuracy of 97%, which is way better than the existing model. The team added a Convolutional layer to further experiment, followed by BatchNormalization, and MaxPooling Layer with a dropout of 0.5. They applied LaekyRelu with a learning rate of 0.3 to this layer. Two dense layers with BatchNormalisation were added, and reLu activation was chosen before the output layer. Their model gave a 97.04% accuracy compared to the model in step (3), with an accuracy of 77.83%.

Similarities

Similarities can be observed in the new model creation to the training model on the Binary Alphadigits dataset. Both groups use three layers, starting with Conv2D, followed by two dense layers.

Differences

Several differences can be observed between Group 13's approach and our group's. Group 13, to improve the performance on the Binary Alphadigits dataset, used the same layers from the previous model that they trained on the EMNIST dataset. Compared to that, our team uses the last layer of the prior model and then adds a dense layer, followed by batch normalization and then followed by a softmax layer to predict the labels. For creating a new model to train on the Binary Alphadigits dataset, Group 13 used Leaky ReLU for the activation function. Moreover, after the dense layer, they perform batch normalization followed by the ReLU activation function and then a dropout, followed by the final dense layer. In contrast, our team uses two dense layers with no additional batch normalization or dropout in between. For training the model, our team uses 150 epochs with a batch size of 300 compared to only 15 epochs with a batch size of 30 run by Group 13.

Comparing Results

Group 12

EMNIST CNN

From a contemporary look at the group's Google Collab, there are no accurate results from running their model. The network using the Adam optimizer was run but ended early, which gives us inconclusive results. Comparing their networks and methods to ours and from our own testing, they likely would have gotten higher accuracy over the baseline as many methods for increasing prediction accuracy were applied, but overall likely half to over a percentage point or more below our model, as were able to utilize our team member's additional hardware and run many similar architectures.

Pre-Trained and Tuned AlphaDigits

The pre-trained and tuned model this group created to predict the newest digits reached around 89.7% accuracy. Compared to our group, we reached an accuracy of around 91.1%. We attribute this accuracy mainly to the difference in our earlier pre-trained models. We let the model train with many more epochs, allowing us to reach an extra percentage in accuracy. However, even in the early epochs of our model, we were reaching accuracies of over 90%.

Group 13

EMNIST CNN

The EMNIST CNN models created by both groups achieved high accuracy, with our group achieving around 94.27% and the other group achieving around 93.99%. Both models utilized convolutional layers with the 'relu' activation function and 'he_uniform' kernel initializer, which may have contributed to their similar performance. However, a critical difference between the two models is the inclusion of data augmentation techniques in the other group's model. Overall, both models demonstrated similar performance, but the other group's model utilized data augmentation techniques, potentially leading to its slightly lower accuracy than ours.

Pre-Trained and Tuned AlphaDigits

The Pre-Trained and Tuned AlphaDigits model created by the other team achieved high accuracy, 97.04 %. In comparison, our model achieved around 88.12 % test accuracy for the same model. Upon close inspection of the other team model, it seems they have used the same data ('x_test_new' and 'y_test_new') for both validation training and evaluation, which could have potentially led to the inflated test accuracy results. Thus, if we pick the validation results for the last epoch, which is 91.18 % validation accuracy and which is then again compared with our results, we could see that both models achieved very similar results.

Test Accuracy

Groups	**Group 12**	**Group 13**	**Group 15 (us)**
CNN model	interrupted	93.9%	94.27%
Transferred model	Did not calculate the accuracy	90.19%	21.78%
Tuning the model	89.15%	97.05%	91.18%