Natural Image Classification

Lior Haroosh, *Student, UF,* Doron Lisiansky, *Student, UF,* Minato Myers, *Student, UF,* and Piotr Suder, *Student, UF*

Abstract—Our team developed a natural image classifier using convolutional neural networks to categorize images into 10 different categories. We created the model using the Keras framework from TensorFlow library and trained it using a training set gathered by students in the class. By using techniques such as data augmentation and hyperparameter tuning, we were able to achieve a maximum accuracy of approximately 78% on the validation data.

1 Introduction

In the early stages of experimentation, we considered several classification methods. Our initial proposition was to reduce the dimensionality of the data by extracting the histogram of colors and applying edge detectors to every image. We were planning to then build Gaussian naive Bayes classifiers with these features and combine them into an ensemble learning algorithm. However, upon noticing poor classification results, we searched for a more robust method.

Recently, convolutional neural networks have become more popular within the field of computer vision. These networks learn from the data by assigning weights to different parts of images and extract features from them. After several trial runs, we noticed issues in overfitting, so we introduced data augmentation on the training data. An article by Fonseka and Chrysoulas reveals how applying random transformations, like rotation, zooming, and flipping, can increase the size of the training data set while reducing overfitting [1]. When running the model with the augmented training data, the validation accuracy increased to about 78%.

Other methods to reduce overfitting included fine-tuning parameters, such as dropout regularization to the network. Dropout randomly omits some hidden units within each layer during training, so that the network's outputs are not overly dependent on certain units. An article by Thanapol et al. suggests better overall performance when applying data augmentation together with dropout [2]. We ran the neural network model under varying architectures and parameters, adjusting these along the way while keeping track of the changes in accuracy. The general pattern in our architectures began with smaller filters to collect as much local information as possible, and then slowly increase this filter width to reduce the feature space and represent high-level information [3]. Between the convolutions, downsampling, or pooling, reduces the spatial dimensions of the image. The batch size and dropout parameters were modified as seen fit, resulting in higher validation accuracy.

Manuscript received December 1, 2021; revised December 5, 2021.

2 Implementation

2.1 Gaussian Naive Bayes classifier

The first models we implemented were based on the Gaussian naive Bayes classifier. We attempted to reduce the dimensionality of the data by extracting features from the images with help of the histogram of colors and edge detectors.

2.2 Convolutional Neural Networks

We tested several architectures of convolutional neural networks. All the architectures used the ReLu activation for all of their layers. In each architecture we had 16 neurons for the last hidden layer, since we used one-hot encoding which meant that the last hidden layer could not have fewer than 10 neurons, i.e. fewer than the number of classes. Then we were doubling the number of neurons in the hidden layers before that, with each layer being a 2D convolution layer, until we arrived at the first hidden layer which would be a dense layer. Finally, for each architecture we would perform the dropout of 0.2 for the first 2D convolution layer.

2.3 First architecture

1st Hidden Layer (Dense): 128 neurons
2nd Hidden Layer (2D Convolution): 64 neurons
3rd Hidden Layer (2D Convolution): 32 neurons
4th Hidden Layer (2D Convolution): 16 neurons

2.4 Second architecture

1st Hidden Layer (Dense): 256 neurons
2nd Hidden Layer (2D Convolution): 128 neurons
3rd Hidden Layer (2D Convolution): 64 neurons
4th Hidden Layer (2D Convolution): 32 neurons
5th Hidden Layer (2D Convolution): 16 neurons

2.5 Third architecture

1st Hidden Layer (Dense): 512 neurons
2nd Hidden Layer (2D Convolution): 256 neurons
3rd Hidden Layer (2D Convolution): 128 neurons
4th Hidden Layer (2D Convolution): 64 neurons
5th Hidden Layer (2D Convolution): 32 neurons
6th Hidden Layer (2D Convolution): 16 neurons

2.6 Fourth architecture

1st Hidden Layer (Dense): 1024 neurons
2nd Hidden Layer (2D Convolution): 512 neurons
3rd Hidden Layer (2D Convolution): 256 neurons
4th Hidden Layer (2D Convolution): 128 neurons
5th Hidden Layer (2D Convolution): 64 neurons
6th Hidden Layer (2D Convolution): 32 neurons
7th Hidden Layer (2D Convolution): 16 neurons

2.7 Ensemble learning

We attempted to combine some of the models we mentioned above by the means of ensemble learning. In the experimental phase we would take all the models included in the particular ensemble learning scheme and have them participate in the majority vote by taking the weighted average of the predicted probabilities for the sample belonging to each class. Each of the votes in this average vote was rescaled proportionally to the individual accuracy obtained for that model in the earlier experiments with it.

The first ensemble model we created combined the two models which used the Gaussian naive Bayes classifier with histogram of colors and edge detectors as features.

The second ensemble model we created combined the two models from the first ensemble with the first convolutional neural network architecture.

The third ensemble model combined the convolutional neural network with the model using histogram of colors as the features.

3 Experiments

3.1 Splitting the dataset

In order to experiment with the architectures we split the data for the first time into training data (90%) and testing data (10%). Then we further split the training data into training set (90%) and validation set (10%).

We first trained the models based on Gaussian naive Bayes classifier. We obtained the validation accuracy of 0.34 for the model using edge detectors and 0.26 for the one using histogram of colors as features. Then we trained the first CNN architecture on the training set. We recorded the highest accuracy on the validation set after around 30 epochs after seeing that the validation accuracy started decreasing due to overfitting. The peak validation accuracy we recorded was 0.55. we then used these accuracies to construct the voting scheme for the ensemble learning methods.

The accuracies recorded on validation datasets are presented in the table below.

Model	Accuracy
GNB-Hist	0.26
GNB-Edges	0.34
CNN	0.55
Ensemble I (GNB-Hist + GNB-Edges)	0.31
Ensemble II (CNN + GNB-Hist)	0.56
Ensemble III (CNN + GNB-Hist + GNB-Edges)	0.53

Table I. Accuracies for the first group of models

We did not get any significant improvement with using ensemble learning, in fact the accuracy decreased to 0.53 when using all three models. With using only CNN and GNB-Hist the accuracy was at 0.56, which we deemed an insignificant improvement to justify using it.

The reasons for this approach not producing good results were revealed when we computed the correlation matrix for the errors made by all the three algorithms. That is, for each algorithm we created a vector of indicators of errors for the algorithm, where the i-th entry of the vector is set to 1 if the label corresponding to the i-th sample from the test set predicted by the algorithm was incorrect, and 0 if it was correct. Then we computed the correlation matrix for these vectors for all the algorithms. If we want the voting scheme to improve the accuracy of prediction we would want to have negative correlations between the error vectors for the algorithms we are combining. Then if one of the algorithms makes a mistake it can be outweighed by the votes of the other algorithms which would likely not make a mistake in this instance, because of the negative correlation between the errors being made by the algorithms.

Below we present the correlation matrix obtained for the convolutional neural network (CNN) and the two Gaussian naive Bayes classifiers (GNB-Hist and GNB-Edges).

	CNN	GNB-Hist	GNB-Edges
CNN	1	0.248	0.195
GNB-Hist	0.248	1	0.111
GNB-Edges	0.195	0.111	1

Table II. Correlation matrix for CNN and GNB classifiers

As we can see, there is a significant positive correlation between any pair of models here which explains why using any combination of votes from these three models does not improve the accuracy of prediction. We hence decided to exclude the models based on Gaussian naive Bayes classifier from further consideration since they did not produce accurate results themselves and did not support the CNN architecture in making decisions. We decided to focus on CNNs, particularly on the larger architectures in hope of extracting more complex features from the images.

3.2 Augmenting the data

During the training of the first CNN architecture we noticed that the validation accuracy started significantly decreasing at certain point during the training which was indicative that there was not enough data and model was staring to overfit. In order to get more data for the training and

improve the accuracy we performed data augmentation on the training set. We performed the data augmentation by using three random transformations: flipping, zooming, and rotation. We produced 25 augmented images for every sample image. These were executed using the Keras preprocessing layers for data augmentation. Then we proceeded to training larger CNN architectures on this augmented dataset. As we experimented with different architectures, we kept track of the difference in accuracy between the training and validation datasets which can show signs of overfitting. However, with the augmented data we did not see any significant drop in the validation accuracy for later epochs, as it was orinigally the case.

3.3 Results for larger architectures

We trained each architecture using batches of size 64 with the learning rate of 0.001. We recorded the highest accuracy we obtained on the validation set for each of them.

Architecture	Accuracy
256 neurons in 1st hidden layer	0.73
512 neurons in 1st hidden layer	0.76
1024 neurons in 1st hidden layer	0.78

Table III. Highest accuracies for larger architectures

Since the largest architecture produced the best results we decided to stick with it.

3.4 Selecting batch size and learning rate

We considered different batch sizes and learning rates. Below we present the learning curves we obtained for them on the architecture with 1024 neurons in the first hidden layer.

3.4.1 *Learning rate*

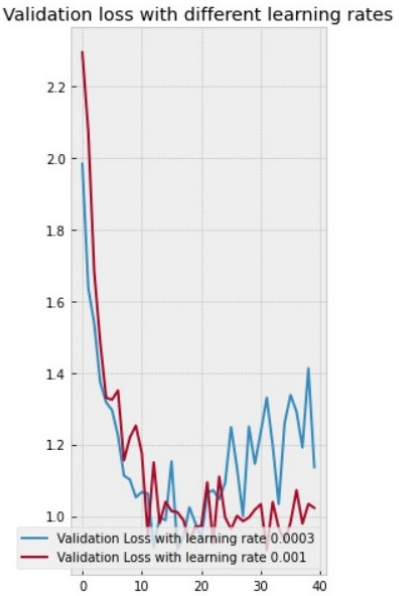

Fig.1. Validation loss function for different learning rates

As seen in Fig.1 above, using the learning rate of 0.001 yielded better results than using the rate of 0.0003. What is surprising is that the loss for the larger learning rate seems to be decreasing at a more stable rate. Since 0.001 yielded better results we chose this value as the learning rate.

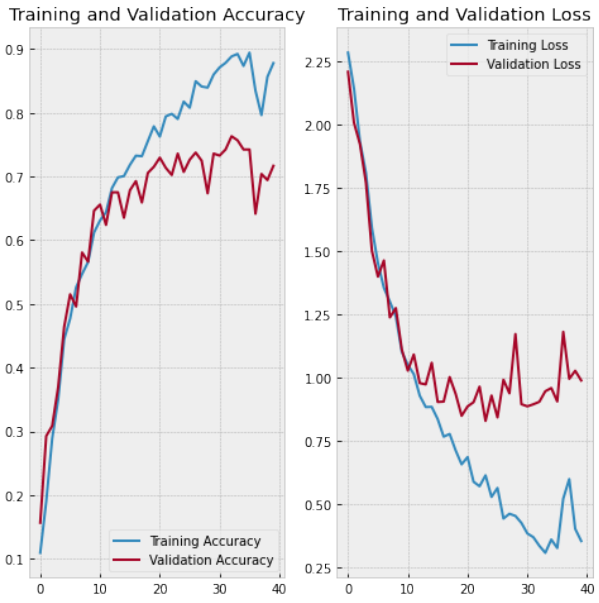

Fig.2. Loss function and accuracy for batch size of 64

3.4.2 *Batch sizes*

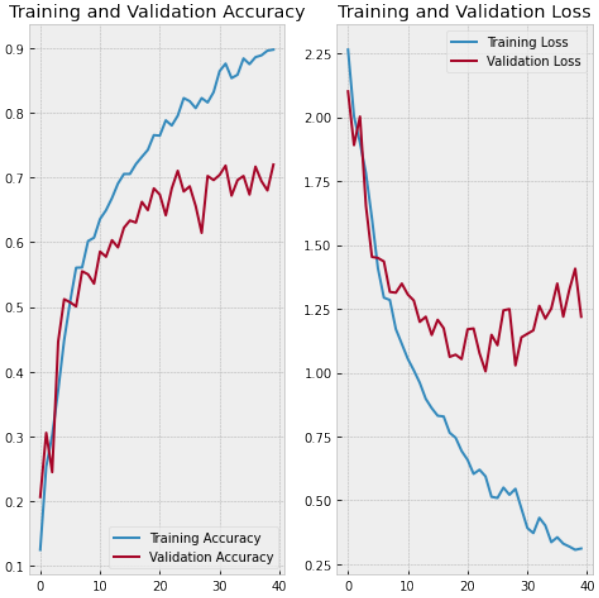

Fig.3. Loss function and accuracy for batch size of 128

We can see in both Fig.2 and Fig.3 that the loss function both for training and validation datasets decreases at a very similar rate for both batch sizes. For batch size of 64 we do not observe significantly more variability and instability in the learning curves than for batch size of 128. Moreover, we can see that for batch size of 64 we reach a higher peak value for the validation accuracy. So we decided to stick with the batch size of 64.

3.5 Number of epochs

In most of the experiments we run from 40 to 60 epochs, since we would not see the validation accuracy increasing beyond these epochs. However, for the final training of the model we run 80 epochs just to make sure that the accuracy stabilized between the epochs at the highest possible level.

4 Conclusion

After finishing this experiment, we have concluded several methods to improve on our base neural network model, resulting in a maximum accuracy of approximately 78% on the validation data:

(1) Increase number of hidden layers. Our best model's first hidden layer was comprised of 1024 neurons. This was the largest model we considered so it is possible that using an even larger model would yeild better results.

(2) Introduce dropout layers. These prevent overfitting on the training data by nullifying some hidden neurons.

(3) Subject training data to random transformations. The additional augmented data could help expose the model to more aspects of the data and generalize better to unseen data.

We also saw that ensemble learning does not always yield better results, especially if the classifiers we are adding to the ensemble have problems with similar groups of samples. Furthermore, this project shows that the training of neural networks requires a very large number of epochs to run before the validation accuracy stabilizes at the highest level, unless we do not have enough training data, in which case early stopping can prevent overfitting. Finally, this project demonstrates that in a situation with insufficient amounts of data to train the model, using data augmentation can significantly improve the accuracy of the model.

5 Future Work

Regarding future experimentation, we can compare different combinations of data augmentation techniques and parameter changes, such as the number of epochs, layers, neurons, etc. Given enough computational resources, we can consider expanding the training dataset further by using other image transformations like color augmentations, random cropping, and noise injection [1]. A different clustering technique we considered was the K-means clustering algorithm for image compression. By reducing the number of colors that appear in an image to only those that are most common, a 24-bit color representation which contains thousands of colors can be reduced to a 4-bit color representation, containing only 16 colors. Also, principal component analysis (PCA) can be applied to further reduce the dimensionality of the data. PCA can improve the overall model performance as well as reveal important features that stand out more.

References

[1] D. Fonseka and C. Chrysoulas, "Data Augmentation to Improve the Performance of a Convolutional Neural Network on Image Classification," 2020 International Conference on Decision Aid Sciences and Application (DASA), 2020, pp. 515-518, doi: 10.1109/DASA51403.2020.9317249.

[2] P. Thanapol, K. Lavangnananda, P. Bouvry, F. Pinel and F. Leprévost, "Reducing Overfitting and Improving Generalization in Training Convolutional Neural Network (CNN) under Limited Sample Sizes in Image Recognition," 2020 - 5th International Conference on Information Technology (InCIT), 2020, pp. 300-305, doi: 10.1109/InCIT50588.2020.9310787.

[3] C. S. Silva. (2021). Lecture 39 - CNNs continued; Dimensionality Reduction with PCA [PDF file]. Available: https://github.com/Fundamentals-of-Machine-Learning-F21/Lectures/blob/main/Lecture%2039%20-%20CNNs%20continued%3B%20Dimensionality%20Reduction%20with%20PCA/Lecture%2039-in-class%20edits.pdf