Improving CNN Test Accuracy and Generalization with Targeted Optimization Training: Data augmentation, Dropout, L2 Weight Decay, and Learning Rate Scheduling

Jiseok Ham
Saint Johnsbury Academy Jeju

ABSTRACT

This study evaluates whether targeted optimization (early data augmentation, dropout, L2 kernel regularization, and exponential learning-rate decay) improves the generalization and accuracy of a convolutional neural network (CNN) model on a three-class image dataset (450 training images consisting of backpack, eraser, and pen images and 15 test images). The optimized CNN reached 100% test accuracy (15/15), outperforming the unoptimized CNN along with other traditional models tested in the same conditions (Random Forest 46.67%, Gradient Boosting 53.33%). Whilst training, the unoptimized model showed a validation loss trend of bottoming early and inconsistently rising afterwards, a clear sign of overfitting. On the other hand, the optimized model maintained a stable validation curve along with a significantly smaller train-validation gap. The added optimizations moderately increased the runtime of the CNN (optimized CNN = 399 seconds, unoptimized CNN = 355 seconds, Random Forest = 10 seconds, Gradient Boosting = 1946 seconds). All models failed a simple unknown-class detection relying on confidence thresholds, mainly due to the small dataset size. Such results note that targeted optimization can notably improve CNN test accuracy and generalization, although having minor drawbacks.

KEYWORDS

Optimization, Image Classification, Convolutional Neural Networks

INTRODUCTION

Convolutional neural networks (CNNs) are reliable models for image classification as they incorporate efficient feature extraction and learning directly using the pixels from the datasets. However, unoptimized CNNs are often prone to overfitting (the model failing to generalize when learning and instead focusing on unnecessary elements), especially when the datasets used to train the models are small and show signs such as training loss continuing to fall while validation loss rises [1]. This study evaluates the performance of a CNN model and compares it with other renowned models after applying practical optimization techniques: image data augmentation to expand sample diversity, dropout to reduce the likelihood of the model focusing on unwanted elements, L2 weight decay to penalize overly large weights, and learning rate decay to allow structured training [2]. Prior works note that augmentation shows a notable improvement in vision task performance, dropout reduces overfitting in deep nets, L2 regularization is one of the most used regularizers in deep learning, and that step-size scheduling is crucial for effective optimization [3][4]. Other classical methods, such as Random Forest and Gradient Boosting, are also compared to the optimized deep learning model using identical datasets and training conditions. The study tests whether the optimization techniques show notable improvements in the accuracy, validation-loss trend, and generalization ability of a standard CNN model. An unoptimized CNN model and two untuned tree models are used for comparison.

Hypothesis

Applying targeted optimization, such as data augmentation, dropout, L2 weight decay, and learning-rate scheduling, to a base CNN will improve the generalization (higher test accuracy and lower validation loss), increasing accuracy and performance of the model compared to the identical unoptimized CNN and traditional models (Random Forest, Gradient Boosting) on the same dataset.

MATERIALS & METHODOLOGY

Dataset

We established three object classes: pen, eraser, and backpack. 150 images for each class were collected from the public datasets Pixabay and Images.CV, totalling 450 training images (Table 1) [5][6]. Selection prioritized visual consistency in background and lighting; more images had white backgrounds, with 10-20 per class featuring darker backgrounds for variance. Shadows were generally absent. Images were downloaded in their original size and resolution without modification. For testing, we captured five original photographs per class (15 total) in a controlled indoor environment (Table 2). All images were resized within the models to 180x180 pixels.

Table 1. Example images from the 150-image backpack training set from Images.CV [5] used in the training of the four models

Table 2. Example image from the self-recorded 5-image test set used in the evaluation of the four models

Models

Codes of each model were created using existing public online tutorials. The base deep learning code was from the TensorFlow tutorial website; the Gradient Boosting and Random Forest codes were written based on the Basecamp tutorial website [7][8][9]. The optimized deep learning model was based on the TensorFlow deep learning model with the addition of multiple optimization alterations mentioned in 1.3.3.

Model Architectures

Cross-Model Setup

For all four models, after loading the 450 labeled images (150 per class), the data is split into training and validation sets with a validation split of 0.2, using 80% of the data to train the model and the remaining 20% to evaluate the model's ability to generalize during training.

Model 1: Deep Learning Model (Unoptimized)

A Sequential model is a simple way to build a neural network by stacking layers one after another in order [10]. It's used because it's best when the model flows in one direction from the first layer to the last without branching or merging layers [10]. This model is sequential, using three 2D convolutional (Conv2D) layers (16, 32, and 64 filters) with ReLU and 3x3 kernels. This model uses the standard TensorFlow CNN tutorial model without modifications in its architecture and training strategy, excluding a change in its epoch from 15 to 50 during its training session [7]. Convolutional (Conv2D) layers apply filters across image pixels to automatically detect features [11]. The number of filters (16, 32, 64) is how many separate sets of pattern detectors the layer has. Each filter detects a distinct feature. The 3x3 kernel size is the size of the scanning window in the training process. To progressively extract features and patterns, each is followed by a 2x2 max pooling layer (looks at each 2x2 block of pixels in the feature map and keeps only the largest value from the block) [12]. This speeds up training and reduces the chance of overfitting. These are implemented with functions from TensorFlow Keras [13][14]. With a batch size of 32, the model trains for 50 epochs, meaning it goes over the entire dataset 50 times, updating its weights after every 32-image batch. This allows the minimization of loss and the growth of accuracy as the code is run. The model is compiled with the Adam (Adaptive Moment Estimation) optimizer. The model includes no optimization techniques, adjustments made to minimize error and increase the efficiency and accuracy of the code, including dropout, L2 regularization, learning rate scheduling, or data augmentation. Thus, the model is vulnerable to overfitting, meaning it would struggle to generalize as any regularization is absent. Without optimization that tackles overfitting, the model learns on training data too precisely, including unneeded noise, and fails to perform well when classifying new data.

Model 2: Deep Learning Model (Optimized)

We decided to apply optimization techniques on the deep learning model as it is one of the most suitable approaches for image classification [15]. This is because the model uses Convolutional Neural Networks (CNN) which utilize convolutional layers that learn both low-level features (ex. textures) and high-level features (ex. patterns) as the layers progress and thus improve the ability to generalize, which is essential in image classification where large datasets are used [16]. This model uses the standard TensorFlow CNN tutorial model with added modifications that focus on the optimization of the code [7]. We focused on tackling overfitting as it is extremely common in CNNs that are trained on moderate-sized datasets [17]. The goal of the optimization was to resolve overfitting and reduce the mean train-validation accuracy gap without losing validation accuracy, a critical sign of increased accuracy. Essential changes made are:

1. **Additional early data augmentation layer**: Before rescaling, we placed a custom sequential augmentation block that includes random flip, random rotation, random zoom, random contrast, and random translation with values of 0.1. This was placed to allow the CNN to train from variances (rotated, shifted, brighter, and darker variants) instead of learning from exact pixel layouts, an essential step in increasing generalization ability.
2. **Dropout Layers:** Dropout is a core method to mitigate overfitting as it randomly shuts down activations (neurons) during training, which prevents the program from overfitting and training from unneeded features such as noise while also allowing diverse representations [2]. A dropout layer with a probability of 0.25 is added after every convolutional layer, while a dropout layer with a probability of 0.5 is added before the dense layer. A lighter rate of dropout was added to the convolutional blocks to prevent disrupting the image scanning process. A higher rate was added to the dense block to improve decision-making.
3. **L2 Regularization:** In addition to dropout layers, we added an L2 regularization coefficient of 0.001 on every convolutional layer and dense layer. Due to L2 regularization, a penalty is given to high weights. Preventing weights from growing large allows the model to avoid focusing on meaningless details when training and allows for a smoother result.
4. **Exponential Decay Learning Rate Scheduler:** The Adam optimizer from the original code decides the size of the step to take when weight updating, but was not adaptive enough [18]. The step should show balance as if it grows too big, the model might skip good solutions while if it becomes too small, the model might focus on unneeded areas [19]. We kept the existing Adam optimizer but modified its learning rate. We decreased the learning rate by 4% every 100 steps to find the balance, allowing cleaner decision making. This allows the model to go over the set in big steps in its initial stages to escape unfavorable areas but zooms into smaller areas as the model progresses to keep the details.
5. **Reduced later augmentation:** We kept the existing augmentation block from the original code (flip, rotation 0.1, zoom 0.1) but decreased the rates to 0.05 each. This was done to ensure balance in the code, preventing the added optimization changes from causing underfitting.

These changes were implemented with functions from TensorFlow Keras [13][14].

Model 3: Random Forest Model

This model uses the standard Random Forest model based on Datacamp's tutorial without modifications in its architecture or training strategy [8]. We decided to compare the deep learning models with a Random Forest because it is one of the most proven traditional models that has clear differences with the deep learning model (e.g. lack of convolutional features) that would show a clear contrast in the results. After validation split, images are converted into 97,200-dimensional vectors (180x180x3). A Random Forest is a traditional model that consists of "decision trees" that are independently trained using bootstrapping and random sampling [20]. Random subsets of features are considered at every split, lowering variance and thus increasing accuracy [20]. During inference, a fraction of trees vote for certain classes, and class prediction is done by counting these votes (i.e. majority voting) [21]. This model does not contain methods of invariance (e.g. translation, rotation, light varying), making it potentially weaker in the area of generalization when compared to the optimized deep learning model.

Model 4: Gradient Boost Model

This model uses the standard Gradient Boost model based on Datacamp's tutorial without modifications in its architecture and training strategy [9]. Gradient Boosting operates by building and training decision trees in a sequential manner, one after the other [22]. This allows the newly trained tree to predict and correct the errors from the previous trees [22]. A critical difference between Random Forest models and Gradient Boost models is that instead of trees being trained independently and combining their votes afterwards, the model optimizes its training by utilizing iterative refinement to decrease the loss function [23]. Thus, as the iteration increases, accuracy increases as well. However, such additional iterations potentially cause longer training time and a chance

of overfitting. During inference, the combined predictions from all trees by a weighted sum forms the final class decision [20]. Similar to the Random Forest model, this model does not contain methods of invariance (e.g. translation, rotation, light varying), making it potentially weaker in the area of generalization when compared to the optimized deep learning model.

Experimental Controls

Same dataset, train-test split, and input image dimensions were used to keep fairness in comparison. For deep learning models, loss function, optimizer (Adam), learning rate settings, and training epochs were controlled (Table 3). Hyperparameter tuning was not applied in the Random Forest and Gradient Boosting, keeping them unoptimized (Table 4). Results including accuracy, confidence, validation accuracy/loss and runtime were recorded under identical computing conditions.

Training and Validation Loss Definition and Evaluation

Loss Concept Overview

For the two CNNs (unoptimized and optimized deep learning models), we recorded the training and validation loss trends during training sessions across epochs. Training loss is the amount of errors the model experiences during its training on a given batch (In this case, 32) [24]. The amount of errors reported after a batch is averaged, and the model attempts to alter its settings (weight) to decrease the number [24]. After an epoch (one round full of learning using the training set), the error numbers for each batch are averaged, which becomes the training loss of the corresponding epoch [25]. Validation loss uses an identical calculation to training loss but uses it on the validation set (20% of the whole dataset) that the model itself does not use to train [26][27]. This helps the model generalize on new images [28]. We only apply this test to the two CNNs as they either lack epoch-level optimization or show minimal results.

Trend Concept Overview

In brief terms, training and validation loss are the averaged amounts of errors reported after an epoch. Thus, a trend where both values are constantly decreased with a stable gap proves that the performance of the model is adequate. On the other hand, a trend where the losses show an increasing trend or is far apart from each other indicates a defect in the training process. When validation loss reaches a low point then rises while the training loss keeps on decreasing, it is an indicator of an overfitting. When both losses stay flat, it indicates underfitting.

Table 3. Core Hyperparameters of the two CNNs

Core Hyperparameters - CNNs						
Model	Input	Epochs	Batch	Val Split	Optimizer /Loss	Regularization
CNN (Unoptimized)	180x180 x3	50	32	0.2	Adam / h	N/A
CNN (Optimized)	180x180 x3	50	32	0.2	Adam / h	Aug, Dropout=0.2, 0.25, 0.5, L2=0.001, LRsched=schedule

Table 4. Core Hyperparameters of the Random Forest and Gradient Boosting models

Core Hyperparameters - Trees						
Model	Features	Training Budget	Max Depth	Learning Rate	Subsample	Random State
Random Forest	97,200 (180×180×3 flatten)	n_estimators=100	None	N/A	N/A	42
Gradient Boosting	97,200 (180×180×3 flatten)	N/A	N/A	N/A	N/A	42

TEST RESULTS

Accuracy and Confidence

For this section, accuracy is calculated by dividing the amount of instances when the model formed a correct guess by the total number of test guesses (e.g. 13/15 = 86.66% accuracy). For the CNNs, confidence is the highest class probability output by the softmax layer. It refers to the model's likelihood of its prediction being correct [29]. For the Random Forest model, confidence is calculated by averaging class probabilities from each tree (each tree's probability is the class frequency among its leaf node samples) [30]. For the Gradient Boosting model, confidence is determined by choosing the highest value from predict_proba(x), which are class probabilities formed by adding all trees' scores and converting them to probabilities [30]. A low confidence (less than 50%) may imply that the image showcases an unknown object that was not included in the training set.

Among 15 test images, the unoptimized deep learning model scored 13 correct, resulting in a 86.67% accuracy (Table 5a). Excluding two instances where the results showed a confidence of 49.55% and 49.19% each where the model misclassified eraser images as pen, the confidence levels were between 97% and 100% (Table 5b). The random forest model scored 7 out of 15, resulting in a 46.67% accuracy with an average confidence level of 73.1% (Table 5a). The model struggled in identifying images of erasers and pens, getting all eraser samples incorrect and getting three out of five pen samples incorrect as well (Table 5b). The gradient boost model scored 8 out of 15, resulting in a 53.33% accuracy (Table 5a). The model also showed difficulty identifying erasers and pens, miscategorizing every eraser sample as pen while getting two pen samples incorrect as well. Except for two instances, the confidence levels were between 99% and 100% (Table 5b). The optimized deep learning model scored 15 out of 15, resulting in a 100% accuracy (Table 5a). The confidence levels of the decisions were all within the range of 91 to 100%, excluding two eraser samples where the confidence levels were 70.23% and 62.92% each and a pen sample where the confidence level was 78.45% (Table 5b).

Deep learning models have a deeper architecture than the tree models, therefore being specialized for image classification tasks. The overall trend of unoptimized non-deep learning models struggling to identify erasers and pens could be due to a dataset limitation where the objects have relatively fewer comparable pixels than backpacks as they have less visual features and because of weak coloring when compared to the background for instances of erasers, illustrated by the nine sample training images and three test images shown in (Table 6). Increasing image resolution, object zoom, or color contrast for pens and erasers could partially relieve these limitations by enhancing distinctive pixel-level features. However, such adjustments alone would unlikely fully resolve the

misclassification for unoptimized non-deep learning models as those models lack the ability to learn spatial patterns or layered visual features in images. Deep learning models learn from a lot of features and place their networks deeper, allowing them to extract features even from harder objects. The optimized deep learning model excels as not only it is specialized for the job but also integrates targeted optimization techniques such as augmentation, learning rate scheduler, and dropout to tackle overfitting.

Table 5a. Accuracy and confidence tables of the four models

model	accuracy	average confidence
CNN	13/15 (86.67%)	92.44%
CNN (Op)	15/15 (100%)	92.32%
R.F	7/15 (46.67%)	74.13%
G.B	8/15 (53.33%)	96.73%

Table 5b. Confidence table of the four models (In percentage, green = correct, red = incorrect, invalid prediction noted)

Accuracy and Confidence															
Model	backpack 1	backpack 2	backpack 3	backpack 4	backpack 5	eraser 1	eraser 2	eraser 3	eraser 4	eraser 5	pen 1	pen 2	pen 3	pen 4	pen 5
CNN	100	99.98	98.62	99.98	100	pen (49.55)	pen (49.19	99.95	99.86	99.92	94.38	99.98	99.93	97.85	98.04
CNN (Op)	100	99.97	99.99	99.90	100	70.23	62.92	98.58	98.71	99.13	91.50	93.15	93.38	78.45	98.93
R.F	81	81	80	86	86	pen (80)	b.p (56)	pen (78)	pen (81)	pen (84)	b.p (37)	eraser (57)	71	b.p (89)	75
G.B	100	100	100	99	100	pen (100)	pen (100)	pen (100)	pen (100)	pen (99)	b.p (71)	eraser (99)	99	99	85

Table 6. Sample train & test images for the four models (outlined in black = used for training, outlined in green = used for testing)

Training and Validation Loss - Result Analysis

The unoptimized model shows a training loss trend of falling smoothly from 1.4806 (epoch 1) to 0.0544 (epoch 50), approaching zero (Table 7a). This indicates that the model is memorizing the training set. The validation loss drops to a minimal point of 0.3049 at epoch 8 (Table 7a). Then, the validation loss value increases up to 1.1247 (epoch 44) with noisy spikes, indicating a strong overfitting. The gap between the training and validation loss widens as epoch increases. These features indicate an issue of overfitting, which causes the model to degrade after the epoch of 8.

The optimized model shows a training loss trend where it declines from 2.4242 (epoch 1) to 0.1271 (epoch 50), but does not reach zero (Table 7b). This indicates healthy learning where regularization is performing well. Validation loss falls from 1.0531 (epoch 1) to 0.3195 (epoch 7) and reaches its lowest value of 0.2675 at epoch 22 (Table 7b). From there, it stays relatively flat with an oscillation in the range from 0.30 to 0.53 with no continuous upward movement, ending at 0.3630 (epoch 50). The final gap between the training and validation loss is 0.236 (0.3630-0.1271), showing signs of better generalization compared to the unoptimized model as the value is lower. Overall, the optimized model shows a significantly increased performance compared to the unoptimized model. Issues shown in the trend of the unoptimized model (e.g. overfitting) do not appear in the trend of the optimized model.

Some additional changes to better the graphs are utilizing early stopping so that the model stops training once it reaches its best performance level (The optimized model would have performed better if it stopped at epoch 22). An increase in the number of data used may also mitigate minor fluctuations shown in the graphs.

Table 7a. Training and Validation Loss Graph for the unoptimized deep learning model during training

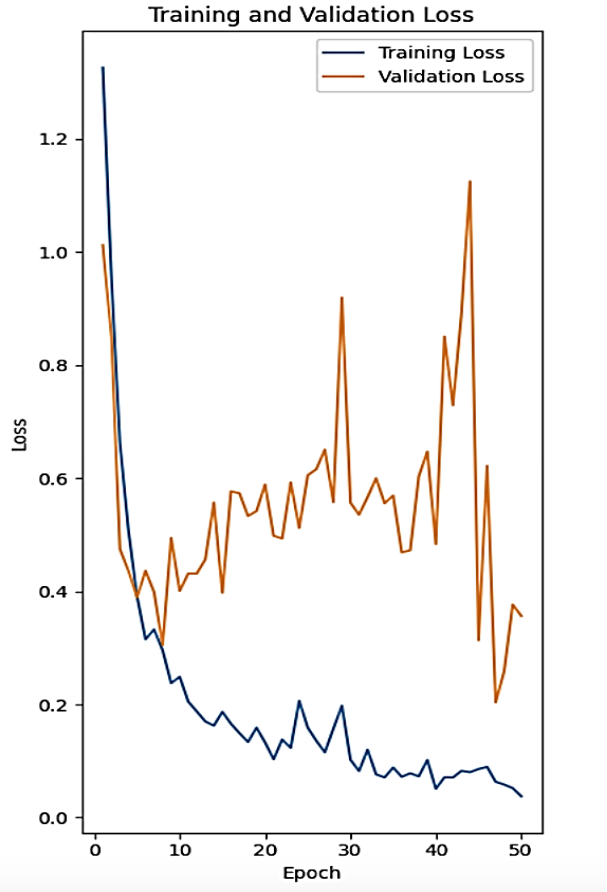
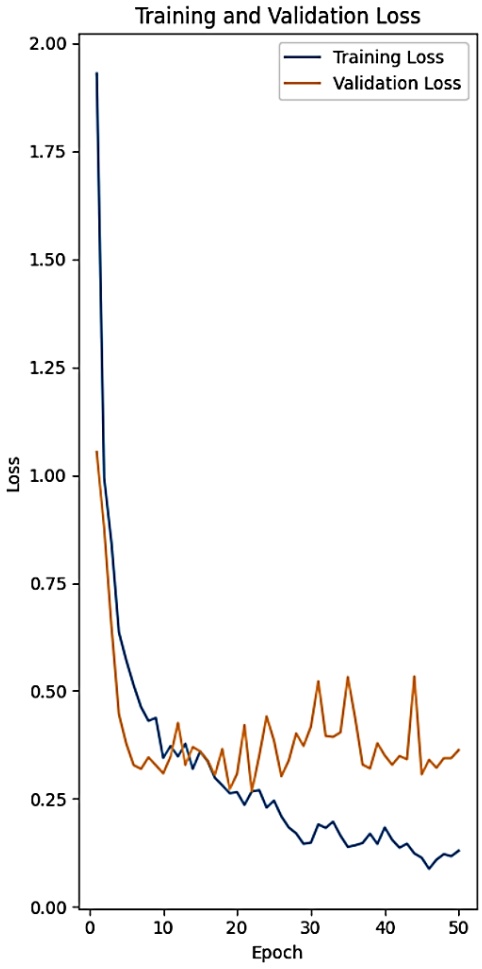


Table 7b. Training and Validation Loss Graph for the optimized deep learning model during training



Runtime

Running time for the unoptimized deep learning model was 355 seconds, followed by 399 seconds for the optimized model. The random forest model had a running time of 10 seconds while the gradient boost model recorded 1946 seconds. The Random Forest model finished fastest since the model allows trees to train independently. This features parallel construction across central processing unit (CPU) cores, significantly reducing elapsed time [20]. The Gradient Boosting model recorded the highest running time as the model sequentially adds trees. Each tree adds on to the previous tree, making parallelized stages impossible [31]. The optimized CNN took slightly longer to run compared to the unoptimized CNN as the added data augmentation features such as additional image transforms added more computational requirements per batch, increasing training time by a noticeable amount (Regularizers were also computed during training). Still, CNNs are more preferred for image detection due to its strength in specialization in classification and its ability to scale better with vision tasks that include larger image datasets despite being slower than models such as Random Forest.

Unknown Dataset Detection

In addition to the 15 test images, 5 images were added. The images showed an image of an apple, an unknown class that the four models were never trained on. A code was written so that models reported an instance of unknown detection every time they scored less than 40% confidence on a certain image. However, all four models failed to report any instances of unknown data, assuming that the dataset consisted of trained images with an

unstable confidence level (50~90%). Such results may have been made because the models were trained in a closed-set, meaning that the models were taught to assume that every test image belongs to a known class [32]. Softmax, a critical function determining the confidence levels, may also be facing inaccuracies and overconfidence. High confidence may be assigned to unrecognizable images, making the threshold of 40% unreliable [33]. This issue could be handled either by adding OOD (out of distribution) detectors such as ODIN to reject images far from the training set or calibrating with temperature scaling and setting the threshold on calibrated scores [34][35].

DISCUSSION

Overview

Key findings of the experiments were that optimized CNN achieved an accuracy of 100%, surpassing that of unoptimized CNN by 13.33% and the two tree models by a significant amount. Training and validation loss showed a trend that illustrates relatively less overfitting or fluctuations in the optimized CNN model compared to the unoptimized original. Some drawbacks existed such as the runtime of the optimized CNN model being 399 seconds, notably higher compared to the random forest model and the unoptimized CNN model and all models failing unknown detection.

Optimization clearly improved the accuracy of the CNN model as added optimization features such as augmentation (random flips, rotations, zoom/contrast/translation), dropout, L2, and learning rate decay showed significant reduction in overfitting based on the training and validation loss trend as the optimized CNN model did not show major signs of overfitting unlike the unoptimized model where validation loss rose with notable fluctuations after clicking an early minimal point.

Random Forest and Gradient Boosting models used flattened inputs without tuning or variance-encouraging optimization. CNNs are specialized for image classification, with training done in a hierarchical manner along with variances that allow accurate extraction of features in the pixel level. With optimization added as well, the CNN model significantly outperformed the rest.

For the unoptimized CNN, errors were clustered among visually similar items that showed weaker signs of uniqueness in features such as monochromatic erasers due to overfitting, decreasing the model's ability to generalize instead of focusing on specific cues that could be unnecessary. Although not definite due to the small sample size of 15 test images, the optimized model's perfect score implies that the optimization features caused notable change.

Limitations

A key limitation is that the dataset size is too small for both the training set and the testing set. An increase in dataset size could improve validation and training loss trends along with a clearer discrepancy between the four models. In addition, the selection of data during validation split is randomized, resulting in an unstable outcome that could differ greatly between runs, especially in this case where data is minimal. The detection method for the unknown class was weak as it relied solely on thresholds instead of utilizing specialized techniques or calibrated probabilities. Methods such as calibration with temperature scaling, increasing the threshold, and utilizing OOD detectors would allow the mitigation of this issue.

CONCLUSION

This study shows that targeted optimizations (early data augmentation, dropout, L2 weight decay, and exponential learning rate decay) are capable of converting an unoptimized CNN into a strong data classifier. The optimized model reached 100% test accuracy (15/15), surpassing the unoptimized CNN and the two tree models by a notable

gap. The model also exhibits a cleaner validation loss trend with a decreased train-validation gap although runtime was increased by 44 seconds. These results show that targeted optimization (regularization) is the key to generalization when training on a small dataset. On the other hand, clear limitations exist such as the dataset being small, potentially limiting accuracy and the performance of the models and the unknown dataset detection being underperforming. For next steps, we will significantly increase the size of the dataset and utilize calibrated methods and OOD-specialized techniques such as temperature scaling and ODIN to encourage an increase in accurate feature extraction and precise unknown detection.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning," *Deeplearningbook.org*, 2016. https://www.deeplearningbook.org/contents/regularization.html (accessed Aug. 08, 2025).

[2] S. Y. Ali and H. Maseeh, "Dropout: An Effective Approach to Prevent Neural Networks from Overfitting," *Asian Journal of Research in Computer Science*, vol. 18, no. 2, pp. 163–185, Feb. 2025, doi: https://doi.org/10.9734/ajrcos/2025/v18i2569 (accessed Aug. 08, 2025) .

[3] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *Journal of Big Data*, vol. 6, no. 1, Jul. 2019, doi: https://doi.org/10.1186/s40537-019-0197-0 (accessed Aug. 08, 2025).

[4] N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014, Accessed: Aug. 10, 2025. [Online]. Available: https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

[5] "Pixabay," *Pixabay.com*, 2024. https://pixabay.com/ko/ (accessed Aug. 10, 2025).

[6] "Labeled image datasets for computer vision," *Images.cv*, 2024. https://images.cv/ (accessed Aug. 10, 2025).

[7] TensorFlow, "Image classification | TensorFlow Core," *TensorFlow*, Apr. 03, 2024. https://www.tensorflow.org/tutorials/images/classification (accessed Aug. 10, 2025).

[8] A. Shafi, "Random Forest Classification with Scikit-Learn," *Datacamp.com*, Oct. 01, 2024. https://www.datacamp.com/tutorial/random-forests-classifier-python?dc_referrer=https%3A%2F%2Fwww.google.com%2F (accessed Aug. 10, 2025).

[9] Bex Tuychiev, "A Guide to The Gradient Boosting Algorithm," *Datacamp.com*, Dec. 27, 2023. https://www.datacamp.com/tutorial/guide-to-the-gradient-boosting-algorithm (accessed Aug. 10, 2025).

[10] TensorFlow, "tf.keras.Sequential | TensorFlow Core v2.3.0," *TensorFlow*, Jun. 07, 2024. https://www.tensorflow.org/api_docs/python/tf/keras/Sequential (accessed Aug. 11, 2025).

[11] Keras, "Keras documentation: Conv2D layer," *keras.io*. https://keras.io/api/layers/convolution_layers/convolution2d/ (accessed Aug. 11, 2025).

[12] TensorFlow, "tf.keras.layers.MaxPool2D | TensorFlow Core v2.4.1," *TensorFlow*, Jun. 07, 2024. https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D (accessed Aug. 11, 2025).

[13] Keras, "Home - Keras Documentation," *Keras.io*, 2019. https://keras.io/ (accessed Aug. 11, 2025).

[14] TensorFlow, "TensorFlow White Papers | TensorFlow," *TensorFlow*, 2015. https://www.tensorflow.org/about/bib (accessed Aug. 13, 2025).

[15] Z. Wang, C. Tang, X. Sima, and L. Zhang, "Research on Application of Deep Learning Algorithms in Image Classification," *IEEE Xplore*, Apr. 01, 2021. https://ieeexplore.ieee.org/document/9421185?denied= (accessed Aug. 13, 2025).

[16] H. Li, "Research on image feature extraction and classification based on deep learning," *Academic Journal of Computing & Information Science*, vol. 8, no. 1, 2025, doi: https://doi.org/10.25236/ajcis.2025.080101 (accessed Aug. 13, 2025).

[17] Tensorflow, "Explore overfitting and underfitting | TensorFlow Core | TensorFlow," *TensorFlow*, 2017. https://www.tensorflow.org/tutorials/keras/overfit_and_underfit (accessed Aug. 13, 2025).

[18] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv*, Dec. 22, 2014. https://arxiv.org/abs/1412.6980 (accessed Aug. 14, 2025).

[19] S. Ruder, "An overview of gradient descent optimization algorithms *," Jun. 2017. Accessed: Aug. 14, 2025. [Online]. Available: https://arxiv.org/pdf/1609.04747

[20] scikit-learn, "1.11. Ensemble methods — scikit-learn 0.22.1 documentation," *Scikit-learn.org*, 2012. https://scikit-learn.org/stable/modules/ensemble.html (accessed Aug. 15, 2025).

[21] M. Olson and A. Wyner, "Making Sense of Random Forest Probabilities: a Kernel Perspective." Accessed: Aug. 15, 2025. [Online]. Available: https://ryansbrill.com/pdf/statistics_in_sports_papers/making_sense_of_random_forest_probabilities.pdf

[22] A. I. Adler and A. Painsky, "Feature Importance in Gradient Boosting Trees with Cross-Validation Feature Selection," *Entropy*, vol. 24, no. 5, p. 687, May 2022, doi: https://doi.org/10.3390/e24050687 (accessed Aug. 15, 2025).

[23] Alexandros Agapitos, A. Brabazon, and M. O'Neill, "Regularised gradient boosting for financial time-series modelling," *Computational Management Science*, vol. 14, no. 3, pp. 367–391, May 2017, doi: https://doi.org/10.1007/s10287-017-0280-y (accessed Aug. 15, 2025).

[24] Keras Team, "Keras documentation: Losses," *keras.io*. https://keras.io/api/losses/ (accessed Aug. 18, 2025).

[25] K. Team, "Keras documentation: Keras FAQ," *keras.io*. https://keras.io/getting_started/faq/ (accessed Aug. 16, 2025).

[26] Keras Team, "Keras documentation: Training & evaluation with the built-in methods," *keras.io*, Mar. 01, 2019. https://keras.io/guides/training_with_built_in_methods/ (accessed Aug. 16, 2025).

[27] K. Team, "Keras documentation: Model training APIs," *keras.io*. https://keras.io/api/models/model_training_apis/ (accessed Aug. 17, 2025).

[28] Tan, Steinbach, Karpatne, and Kumar, "Introduction to Data Mining, 2nd Edition," *Model Overfitting*, Mar. 02, 2021. https://www-users.cse.umn.edu/~kumar001/dmbook/slides/chap3_overfitting.pdf (accessed Aug. 17, 2025).

[29] D. Hendrycks and K. Gimpel, "A BASELINE FOR DETECTING MISCLASSIFIED AND OUT-OF-DISTRIBUTION EXAMPLES IN NEURAL NETWORKS." Accessed: Aug. 15, 2025. [Online]. Available: https://arxiv.org/pdf/1610.02136

[30] Scikit-Learn, “sklearn.ensemble.RandomForestClassifier — scikit-learn 0.20.3 Documentation,” *Scikit-learn.org*, 2025. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (accessed Aug. 16, 2025).

[31] A. Natekin and A. Knoll, “Gradient boosting machines, a tutorial,” *Frontiers in Neurorobotics*, vol. 7, no. 21, 2013, doi: https://doi.org/10.3389/fnbot.2013.00021 (accessed Aug. 17, 2025).

[32] W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, and T. E. Boult, “Toward Open Set Recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 7, pp. 1757–1772, Jul. 2013, doi: https://doi.org/10.1109/tpami.2012.256 (accessed Aug. 18, 2025).

[33] A. Nguyen, J. Yosinski, and J. Clune, “Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images,” 2015. Accessed: Aug. 18, 2025. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Nguyen_Deep_Neural_Networks_2015_CVPR_paper.pdf

[34] C. Guo, G. Pleiss, Y. Sun, and K. Weinberger, “On Calibration of Modern Neural Networks.” Accessed: Aug. 18, 2025. [Online]. Available: https://proceedings.mlr.press/v70/guo17a/guo17a.pdf

[35] S. Liang, Y. Li, and R. Srikant, “ENHANCING THE RELIABILITY OF OUT-OF-DISTRIBUTION IMAGE DETECTION IN NEURAL NETWORKS.” Accessed: Aug. 18, 2025. [Online]. Available: https://arxiv.org/pdf/1706.02690