Investigating CNNs Performance on the CIFAR-10 Dataset through Hyperparameter Tuning

Claire Hyeju Lee

Received September 23, 2024
Accepted January 31, 2025
Electronic access February 28, 2025

In the typical everyday lives of individuals, images are present: whether that be of the scenery or the animals in our world, the cars on the street, or the birds living on the trees. Photos in the world are not entirely unique: their subject matters inevitably intersect. But what differentiates every photograph is the angle that they are taken. Though the same deer may be photographed, one photo may be taken from below, and the other from above. It is quite plain, regardless of angle though, that the subject matter is what it is (such as in this case: a deer). It is not as straightforward as computers, which work off of code and formulas. So, how can computers identify these images? With neural networks such as convolutional neural networks, and training. Of course, these structures are not perfect when initially faced with the photo; they need to be changed and modified to improve their performance. To do so, hyperparameters of these networks must be tuned. These are part of what this paper examines: the success of CNNs in identifying such photos and the extent to which their performance can be improved with hyperparameter tuning. This paper fills in the research gap of further exploration of CNN performance on images of real life objects from multiple angles and towards finding out even more improvements of the performance of CNNs (with regards to hyperparameter tuning, in particular).
This topic is important to consider since the work involved in delving into it contributes towards finding out the true efficiency/accuracy of convolutional neural networks at identification of images, and the worth of convolutional neural networks in identifying photos taken at multiple camera angles.
This paper's methods involved using a pre-existing CNN framework[1] to then tune the hyperparameters of. The particular hyperparameters changed in this paper were batch size, learning rate, number of convolutional layers, size and number of linear layers, and kernel sizes. To ensure the most amount of testing, various combinations of hyperparameters were tested (such as differing numbers of convolutional layers and linear layers in tangency). To see the results and success of these hyperparameter modifications, an individual training and validation dataset was created from the CIFAR-10 dataset with the random_split method, with each training being run with an NVIDIA GPU. Ultimately, it was found that increasing the number of convolutional layers resulted in lowering accuracy and longer times, keeping a linear layer size of 2 led to the highest accuracy regardless of the number of convolutional layers, increasing the size of linear layers upped accuracy, and kernel sizes had the highest accuracy when the layers had the equivalent kernel size of 5. The final tuning of these hyperparameters resulted in an accuracy of 70.07% for this CNN model.

Introduction

Research Question & Importance Of Research

Our ultimate research question is: how well do convolutional neural networks perform at identifying real world objects and animals at any camera angle, and how can the performance of this identification be improved through hyperparameter tuning? This paper hypothesizes that convolutional neural networks are relatively accurate at identifying images from multiple camera angles.

The hope is to infer the true efficiency/accuracy of convolutional neural networks at identification of images, and to thus add to the goal of determining the true worth of convolutional neural networks in identification. Many papers have already examined the performance of image identification: but have they specifically targeted images from multiple angles? If CNNs cannot bring about a desirable performance, then this implies the potential necessity of needing to explore other neural networks besides CNNs for these types of images. Additionally, CNNs continue to have room to grow. Research such as this that focuses on improving performance through some way contributes towards discovering generalizations of modifications that can be made to improve performances of CNNs. Lastly, future papers can utilize the information gained from this research paper to then determine the most efficient method of image identification, and further explore ways to improve the performance of CNN models on identifying images.

Convolutional Neural Networks

This paper worked with Convolutional Neural Networks (CNNs). CNNs are neural networks involving convolutional and fully connected layers. Convolutional layers involve convolutions, which use filters[2].[3], and also contain neurons.

Fully connected layers are essentially linear layers, which are layers with neurons. Neurons take in data and run them through a formula.

For each run or training of the model, the weights and biases of the neurons are modified to improve the model's accuracy[2]. Using non-linear activation functions, outputs of a layer of neurons are put into the next layer[2]. Backpropagation, at the end, changes the neurons' weights and biases to then increase accuracy[4]. Each training iteration improves upon the previous training's weights and biases, resulting in a final model that performs admirably better than it initially did.

Hyperparameter Tuning

This paper tunes hyperparameters, which is essentially the changing or modification of the parameters of the model to improve its performance. Some examples of hyperparameters are the sizes of layers, the size of kernels, and batch size.

Literature Review

Reference[5] compares gradient based learning techniques used on various types of neural networks[5], including convolutional neural networks. The paper's goal was to look at multiple ways for networks to identify "handwritten characters, and compare[d] them on a standard handwritten digit recognition task"[5], discovering that CNNs were the most effective out of all of them in identification. Graph transformer networks, otherwise known as GTNs, were used to make it so that systems like CNNs could be "trained globally using gradient-based methods so as to minimize an overall performance measure"[5]. While using CNNs with global training, their findings were that CNNs did well identifying handwritten digits. This implies the benefits of continuing to use CNNs due to their effectiveness in identifying images, and shows the effectiveness of CNNs in identification of images.[5]

Reference[6] discusses methods of training multi-layer generative models to concentrate on only certain parts of data[6]. The focus of the paper was to show how to train models with several layers to focus on important parts of images. The paper also created two datasets: CIFAR-10 and CIFAR-100. This paper discusses the flawed dataset created by MIT and NYU. The database's constitution made it hard for models to gain the images' filters, for the database had patterns between pixels of images (or two-way correlations). To solve this particular issue, the paper used a whitening matrix. Then, RBMs, otherwise known as Restricted Boltzmann Machines, were trained and used in tangency with Deep Belief Networks (which can increase the RBMs' layers). The now changed weights from the RBM's training were then put into a neural network with back propagation. This neural network was then trained. It was found that using parts rather than the whole image allowed RBMs to train under harder situations. A novel parallelization algorithm was used to have multiple machines work together to finish the learning process. The author also create labeled classes with the CIFAR-10 and CIFAR-100 datasets, proving, ultimately, how identification is better when "pre-training a layer of features on a large set of unlabeled tiny images"[6]. This paper shows the benefits of using CIFAR datasets.[6]

Reference[7] discusses usage of residual functions with layer inputs, and found it to be a method that could still improve its performance[7]. It compares Dense Neural Networks and Convolutional Neural Networks in how well they approximate a building's energy usage given the building's model. The paper used the Rhino-Grasshopper software to form their dataset's building model. EnergyPlus was used, with factors like the floors, area, WWR, width to length ratio, orientation, construction, energy system, and operating schedule being controlled using TMY3 weather data, ASHRAE standard 55 and EnergyPlus schedule library. The DNNs and CNNs were then trained on this database with an 80% and 20% training & testing split. The paper used the Keras library to make their DNN and CNN. The DNN was inputted four vector lists while the CNN was inputted the floorplan photos (with set sizes 48x30). The CNN had 32 convolutional layers and the DNN had 64 dense layers, with both having the Adam optimizer. The paper found that more layers did not lead to CNN loss getting better by much. Using the mean squared error for the loss function, the paper found that the MSE of the prediction for the DNN was in the range of 0.019 and was in the range of 0.430 for the CNN model's prediction. Larger datasets led to better results for both models, and also caused cross-validation to help both (the paper found that cross-validation did not help CNNs when datasets were small). The CNNs generally had more parameters than DNNs, and took longer. Using 5x5 and 4x4 filters lowered the CNN's total number of parameters and only changed the CNN's final results slightly. The paper ultimately found that, though DNN's were better than CNNs, it remains necessary to try to keep making CNNs better.[7]

Lastly, reference[8] discusses sparse and dense neural networks trained with the CIFAR-10 dataset[8]. This paper examined sparse networks, looking at pruning-based and RadiX-Nets topologies. Another paper's pruning-method was used in this paper (which pruned each 200 steps). Lenet-5 and Lenet-100-300 were used, with the MNIST and CIFAR-10 datasets being used as well for their training. The layers of the models were made to be sparse, and the model's structure was RadiX-Nets. For pruning-based topologies, the sparsities 50%, 75%, 90%, 95%, and

99% were tested for both Lenet models by pruning the models once. This process takes out some of the connections that had weights that were not as large as others. For RadiX-Nets, either the connection amounts are kept the same and the neuron amounts change or vice versa while changing sparsity. The paper discovered that 95% sparsity Lenet-300-100 caused better performance while 50%, 75%, and 90% sparsity Lenet-300-100 made it perform normally. Lenet-5 did better than Lenet-300-100 on MNIST, and was the one which had convolutional layers. When completely dense, Lenet-300-100 on MNIST had its highest accuracy. Generally, more sparsity equated to worse results. With sparse networks, larger learning rates caused better results, and sparse networks with RadiX-Net and 50% sparsity were able to do about as well as similar dense networks. This paper adds to the success of CNNs at image recognition.[8]

Methodology

Dataset Used

This research paper used the CIFAR-10 dataset[6]. The CIFAR-10 dataset is made up "of 60,000 32x32 images color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images."[6]. The class names are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. CIFAR-10 has five training and one testing batch. Both have 10,000 images. The test batch has 1000 random images from every class, while the other images are randomly ordered in the training batches. The training batches have 5000 images from every class between them. Each class is "mutually exclusive. There is no overlap between automobiles and trucks"[6].

Validation Of Dataset

CIFAR-10 is a gold standard, and a dataset that is used by many in machine learning to train and evaluate models in identifying models.

Consideration For Future Researchers & Limitations Of Using Cifar-10 For This Paper A limitation of using the CIFAR-10 dataset is that it does not naturally contain images from multiple camera angles. Future researchers should consider using the more recent CIFAR-100 instead.

Network Structure

A simple neural network with fully connected-layers (2 linear layers) was created, through having two linear layers taking in the initial input as a one dimensional tensor, and outputting 10 neurons. The ReLU activation function was used between layers in this neural network as well as in the convolutional neural network. This was compared to the convolutional neural network through adjusting its number of batches to 16, 32, and 64, and then comparing the results between the two neural networks. 15 epochs were used.

Convolutional Neural Network: Detailed Explanation

As noted earlier, Convolutional Neural Networks (CNNs) were used. CNNs are neural networks with convolutional and fully connected layers, with their convolutional layers having convolutions. These convolutions use filters, which are a set of particular values that are passed over an image's pixels. The values of the pixels are multiplied by the values of the filter that are in the same locations as those pixels. The sum of these values yields the final value of the pixel in the middle. Doing this repeatedly over the original image for each pixel makes a new image. For convolutional neural networks, individual neurons all perform convolutions, then have these outputs passed into non-linear activation functions[2,3]

Neurons are within the fully connected layers, and take in data to run them through a y = ax+b formula. The "b" part is the bias, and the "a" part is the weight that the input, "x," is multiplied by.[9].

As for filters, these are a set of particular values that are passed over an image's pixels. The values of the pixels are multiplied by the values of the filter that are in the same locations as those pixels. The sum of these values yields the final value of the pixel in the middle. Doing this repeatedly over the original image for each pixel makes a new image. For convolutional neural networks, individual neurons all perform convolutions, then have these outputs passed into non-linear activation functions[2,3]

Finally, each time the neural network is trained, the neurons' weights ("a") and biases ("b") are changed in order to improve the model's accuracy[2]. The current layer's outputs of its neurons are inputted into non-linear activation functions, which become the inputs of the next layer's neurons[2]. Having non-linear activation functions allow the next layer's neurons to be able to properly interpret the previous layer's neuron's outputs and thus have values relevant to the input after the next layer's neurons perform their own convolutions on their inputted values[2]. Then, the network uses backpropagation: going backwards from the final to the initial layer to change the parts of the neurons in the layers in order to increase the network's accuracy the most.[4].

Hyperparameter Tuning

Parameters can consist of parameters such as linear layer size, kernel size, filter size, and more. Changing or "tuning" such kinds of parameters (hyperparameter tuning) can result in either increases or decreases in the model's total number of parameters and in the model's accuracy or performance.

Paper Title	Authors	Methods	Findings	Implications
Gradient-Based Learning Applied to Document Recognition	Y. LeCun, L. Bottou, Y. Bengio, & P. Haffner	-Used CNNs with global training -Compared multiple different methods for neural networks	-Presented CNNs accuracy of analyzing things like handwritten numbers -CNNs are the best methods for identification out of multiple other methods	-Showcases CNNs usefulness & success in analyzing datasets with things like handwritten digits -Adds to the motivation of continuing to use CNNs in future research endeavors because of how well it does compared to other methods for networks.
Learning Multiple Layers of Features from Tiny Images	Alex Krizhevsky	-RBMs get trained first; then, the weights that it adjusted after training, were inputted into a neural network with back propagation -Novel parallelization to run across multiple machines -Whitening matrix to get rid of two-way patterns for the images' pixels	-Effectiveness of datasets like CIFAR-10 and CIFAR-100, which are properly labeled and minimize similarities between pixels	-Showcases benefits of CIFAR datasets in training models due to their labeling, and in their uniqueness between one another
Convolutional versus Dense Neural Networks: Comparing the Two Neural Networks' Performance in Predicting Building Operational Energy Use Based on the Building Shape	Farnaz Nazari & Wei Yan	-DNNs and CNNs were both trained on a dataset of a building model -Used MSE to determine final accuracy	-DNNs had a better accuracy than CNNs; yet, CNNs have great potential -Expresses the need for further exploration and tuning of CNNs to see if they could be improved more.	-Necessity of continuing to improve CNNs, as there is more to explore with them.
Training Behavior of Sparse Neural Network Topologies	Simon Alford, Ryan Robinett, Lauren Milechin, & Jeremy Kepner	-Sparse neural network topologies (pruning-based and RadiX-Nets) using MNIST and CIFAR	-Sparse neural networks are generally still worse than denser neural networks -CNNs are successful with image recognition	-Benefits of continuing to use denser neural networks such as CNN in the future due to its success

Dataset Split

The CIFAR-10 dataset was randomly split with an 83:17 train/test split. In other words, the training dataset was separated into a size of 50,000 (around 83%) and the testing dataset was split into a size of 10,000 (around 17%).

Hyperparameter Impact

To see the impact of different hyperparameters on the performance of neural networks, the network was trained with different hyperparameters, and trained using GPU. The specific parameters changed were learning rate, batch size, the number of convolutional and linear layers, and the size of the kernels and linear layers.

Changes To The Original Network

The convolutional neural network tutorial from Rafael Castro[1] was used as a basis to evaluate the performance of the CNN on the CIFAR-10 dataset. The DataLoader was modified to load the CIFAR-10 dataset, and had its batch_size be 16 and num_workers be 4.

Batch Size

The in_channels and out_channels of the convolutional layers were modified to be 10 (except for the in_channels for the 1st layer, which was set as 3). The MaxPool2D of each layer was left as is. The neural network was trained on varying batch sizes of 16, 32, and 64 for 15 epochs.

Learning Rate

For learning rates, comparisons of 0.001, 0.0005, 0.00001, and 0.0007 were made. For each of these instances, 700 epochs were used. Then, the validation and training losses of the neural networks were plotted, along with calculating the accuracy from the accuracy calculating loop in the given code[1]. The code for accuracy looped through the inputs and labels in the DataLoader for the testing images, and then evaluated the trained neural network's outputs after receiving those inputs from the testing dataset.

Number Of Convolutional Layers & Linear Layers

To compare between the best number of convolutional and linear layers, the convolutional layers had kernel sizes of 5, but, for every number of convolutional layers tested that was greater than one layer (example: two layers), the previous convolutional layers' MaxPool2D was commented out. For the linear layers, the size of the first layer was (x.size(0), 100) and all following layers were sizes (100, 100). The ReLU activation function and the dropout method of 0.2 was used for every linear layer that was not the final output layer. 70 epochs were used. Learning rate was kept at 0.0005.

Size Of Linear Layers For CNN

For the size of linear layers, a single convolutional layer (with kernel size of 5) and 2 linear layers were used. After applying the convolutions to the input, the input was resized to be (x.size(0), 30*3*3). The first linear layer had its number of output neurons the size that was being tested, and the number of inputs be x.size(0). The second linear layer had the same input and output size (same number of neurons inputted and outputted), and both values were made to be the size that was being tested. After any layer that was not the final output layer, the ReLU activation function was applied, and then a dropout method of 0.2 was applied to prevent overfitting. Learning rate was kept at 0.0005.

Kernel Sizes

The kernel sizes were modified while the in_channels and out_channels of the model were kept as 10 (except for the first convolutional layer, whose in_channels were set as 3). Kernel sizes were modified for 1 and 2 convolutional layers. For 2 convolutional layers, the kernel sizes for the first layer were 5 and 3 and 9 and 5, and were tested for 70 epochs. For 1 convolutional layer, the kernel sizes examined were 3, 5 and 7. The size of the linear layers were (100,100) (except for the first linear layer, which had a size of (input.size(0), 100). The MaxPool2D was commented out as necessary if, after the convolutions, the image grew to be too small. Learning rate was kept at 0.0005.

Fully Connected Vs Convolutional Layer

For this experiment, the sizes of the 2 linear layers for the fully connected neural network were (x.size(0), 6) and (6, 10), respectively. For the convolutional layer(s), the linear layers had sizes of (x.size(0),100) and (100, 100). The neural networks were trained for 30 epochs on a batch size of 16. Learning rate was kept at 0.0005.

Final Performance

The CNN had a single convolutional layer of kernel size 5 and 2 linear layers of sizes (input.size(0), 1000) and (1000, 1000). The model was trained under a learning rate of 0.0005, with 700 epochs.

Results

See Table 1 where, as batch size increased, the time the CNN took to learn decreased. However, as batch size increased, the CNNs accuracy decreased.

Table 1: Batch Size, Time and accuracy for CNN

Number of Batches	Accuracy	Number of Epochs	Number of Parameters	Time Taken
16	32.02%	15	17470	9 min 11s
32	28.44%	15	17470	8 min 14s
64	21.29%	15	17470	7 min 32s

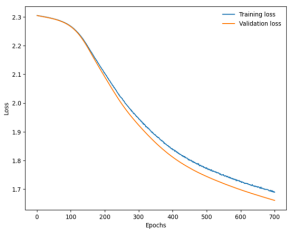

Figure 1: Training loss and validation loss for learning rate 0.00001

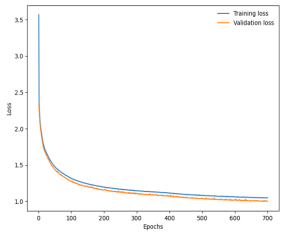

Figure 2: Training loss and validation loss for learning rate 0.0005

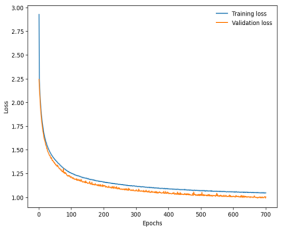

Figure 3: Training loss and validation loss for learning rate 0.0007

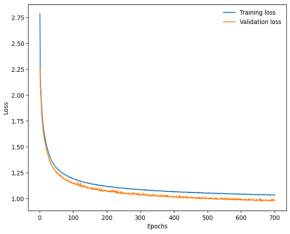

Figure 4: Training loss and validation loss for learning rate 0.001

Figure 5: Training and Validation loss for varying learning rates

See Figure 1, where a learning rate of 0.00001 led to overlapping in the start, and separation between training and validation loss towards the end. See Figure 2, where a learning rate of 0.0005 leads to the model's training and validation loss being the closest. See Figures 3 and 4, where a learning rate of 0.0007 and 0.001, respectively, leads to a slightly larger gap than Figure 2. See Figure 5 to see how a learning rate of 0.0005 is the best.

Table 2: Accuracy for 1-3 Convolutional Layers

# of Convolutional Layers	Accuracy	Time Taken	# of Epochs	# of Linear Layers	# of Parameters
1	63.57%	41 min 57s	70	2	154960
2	62.32%	43 min 40s	70	2	113470
3	59.25%	45 min 51s	70	2	79980

See Tables 2,3,4, and 5: as the number of linear layers and number of parameters increased, the CNNs accuracy decreased. In all cases, if the number of linear layers was less than 2, the accuracy decreased. As the number of linear layers increased, the number of parameters increased. As the number of convolutional layers increased, the number of parameters decreased. As the number of linear layers decreased, the number of parameters decreased. See Table 3 to see that the CNN with the best overall performance was one convolutional layer and two linear layers (63.57%).

Table 3: Accuracy, time taken and # of Params for 1-3 linear layers for 1 convolutional layers

# of Linear Layers	# of Epochs	Time Taken	Accuracy	# of Parameters
1	70	39 min 51s	61.32%	144860
2	70	41 min 57s	63.57%	154960
3	70	42 min 57s	60.04%	165060

Table 4: Accuracy, time taken and # of Params for 1-3 linear layers for 2 convolutional layers

# of Linear Layers	# of Epochs	Time Taken	Accuracy	# of Parameters
1	70	43 min 4s	60.55%	103370
2	70	43 min 40 s	62.32%	113470
3	70	45 min 47s	59.54%	123570

Table 5: Accuracy, time taken and # of Params for 1-3 linear layers for 3 convolutional layers

# of Linear Layers	# of Epochs	Time Taken	Accuracy	# of Parameters
1	70	44 min 28 s	59.17%	69880
2	70	45 min 51 s	59.25%	79980
3	70	45 min 42s	53.52%	90080

Table 6: Accuracy, time taken and # of Params for varying linear layer sizes, for 1 CNN layer and 2 linear layers

Linear Layer Size	Accuracy	# of Epochs	# of Parameters	Time Taken
100	56.91%	30	154960	17 min 20s
500	58.89%	30	971760	17 min 50s
1000	60.27%	30	2442760	17 min 16s
5000	61.77%	30	32210760	26 min 23s

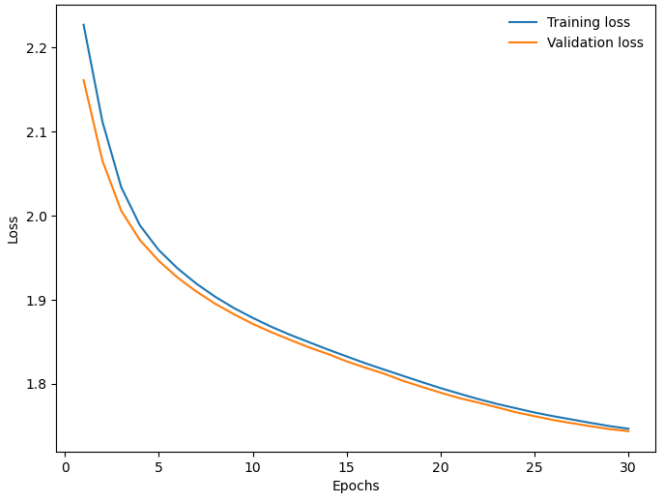

Figure 6: Training loss and validation loss for neural network with fully connected layers

See Table 6, for a CNN with one convolutional layer and 2 linear layers (whose sizes are modified). As the linear layer size increased, the accuracy and number of parameters increased.

Table 7: Accuracy, time taken and # of Params for varying kernal sizes for a CNN with 2 convolutional layers and 2 linear layers

# of Kernels in Layer 1	# of Kernels in Layer 2	Accuracy	# of Epochs	# of Parameters	Time Taken
5	3	52.94%	70	27870	43 min 24s
5	5	62.32%	70	113470	43 min 40s
9	5	57.83%	70	79150	43 min 32s

See Table 7 for a CNN with two convolutional and linear layers. The accuracy worsened as the kernel sizes went from a larger size to a smaller size. The number of parameters decreased if kernel sizes were changed from larger to smaller.

Table 8: Accuracy, time taken and # of Params for varying kernal sizes for a CNN with 2 convolutional layers and 2 linear layers

Kernel Size	Accuracy	# of Epochs	# of Parameters	Time Taken
3	57.62%	30	2962280	17 min 23s
5	59.64%	30	2442760	18 min 3s
7	56.84%	30	2003480	17 min 21s

See Table 8 for a CNN with 2 convolutional layers and 2 linear layers. As kernel size increased, the number of parameters decreased. As kernel size decreased the number of parameters increased. The best accuracy was from a kernel size of 5.

See Figures 5 and 6 for a fully connected neural network and a CNN. The fully connected layers' losses for validation and training began to converge, while they remained separate for the CNN.

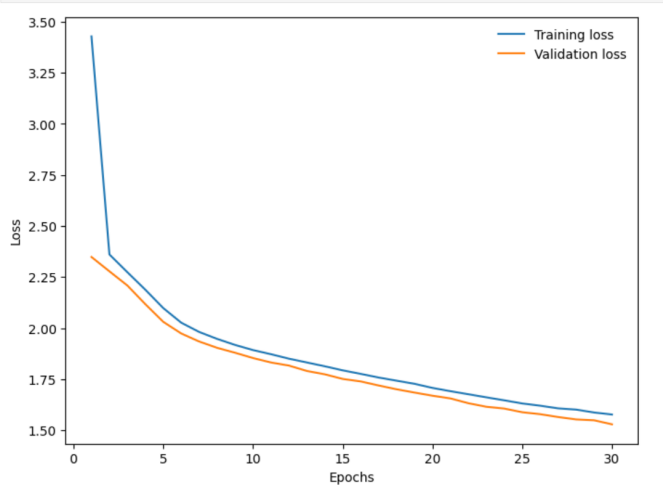

Figure 7: Training loss and validation loss for CNN

Table 9: Accuracy, time taken and # of Params for varying kernal sizes for a CNN with 2 convolutional layers and 2 linear layers

	# of Batches	Accuracy	# of Epochs	# of Parameters	Time Taken
Linear Layer: 2	16	36.74%	30	18508	17 min 26s
Convolutional Layer: 2	16	43.15%	30	17470	18 min 52s

See Table 9 for a fully connected neural network and a CNN. The accuracy for a CNN was much better than that of a fully connected neural network (even though the CNN had fewer parameters to train).

Table 10: Accuracy, time taken and # of params

# of Linear Layers	# of Convolutional Layers	Accuracy	# of Epochs	# of Parameters	Time Taken
2	1	70.07%	700	2442760	6h 49 min 5s

See Figure 8 for a CNN with 1 convolutional layer and 2 linear layers of sizes (1000, 1000) and learning rate of 0.0005, and the subsequent convergence of the training and validation loss over longer epochs.

Discussion

These findings support the conclusions of other articles that convolutional neural networks perform well in identifying images and in comparison to fully connected networks. The learning rate and CNNs accuracy are also inversely related: increasing learning rate lowers accuracy. The CNN does not overfit, as is seen from the fact that the validation loss is never higher than the training loss. Though lower learning rates could technically outperform higher learning rates, the number of epochs necessary greatly outweigh the number of epochs utilized

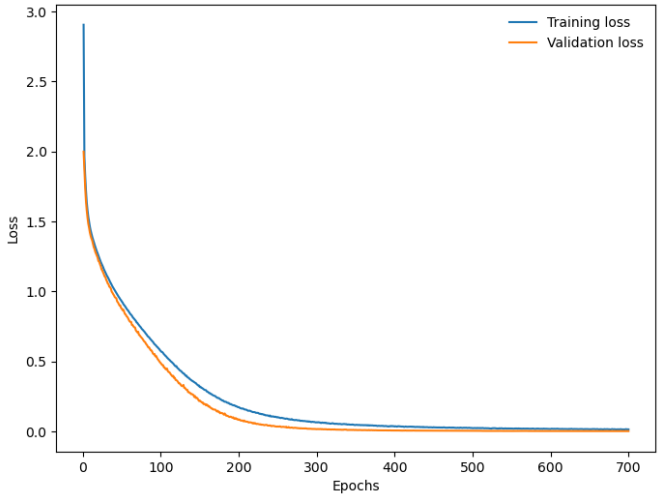

Figure 8: Accuracy, time taken and # of params

(700). A learning rate of 0.00001 could outdo a learning rate of 0.0005 but would require significantly more time and memory. Higher learning rates caused greater oscillations, indicating the CNNs struggles with generalization as learning rate increases too much. Increasing batch size caused the CNNs performance to become worse. Lowering the number of parameters caused the CNN to perform worse for higher epochs, and increasing the number of parameters, while it, again, may allow for the CNN to perform better than under lower parameter amounts, requires too much memory and time. For the changing linear layer numbers for the CNN, accuracy and time had minimal differences across varying numbers of linear layers. However, 2 linear layers is the best, as, though having the 2nd highest time to complete its running, it also had the highest accuracy. For the sizes of the linear layers, the number of parameters increased as the size of the linear layer increased. The accuracy went up as linear layer sizes increased. However, upon having too many parameters, the time taken increased significantly. The best choice was linear layer sizes of 1000, since it had the best accuracy while not taking an extreme amount of time. It is also at a spot where the accuracy for sizes greater than it increases minimally ($\sim$1%).

Training a model with fully connected layers for 30 epochs had its training and validation loss begin to converge. Comparatively, training a CNN with a similar number of parameters for 30 epochs had both of the training and validation loss be different. In addition, despite having similar numbers of parameters, the fully connected neural network performance was worse than that of the CNN. Changing the kernel size from being larger to smaller (one layer having a larger one, then the second layer having a smaller kernel size) decreased the number of parameters, and led to overall worse performance for a CNN with 2 convolutional layers. Maintaining both as having kernel sizes of 5, with the first MaxPool2D being commented out, led to the best accuracy. For a CNN with one convolutional layer, the number of parameters increased as kernel size decreased, and decreased as kernel size increased. The best performance was from a kernel size of 5. The final performance was an accuracy of 70.07%. In short, convolutional neural networks are efficient at image identification, and hyperparameter tuning can increase the model's subsequent performance. It is worth exploring CNNs further to determine the absolute best CNN performance that can be extracted from databases based on the general results from tuning hyperparameters from this paper. This paper suffers from not having the resources at hand to train CNNs for a day or more, and that many parameters may be hypertuned even further (to specific values rather than whole numbers).

Conclusion

Convolutional neural networks are relatively efficient at identifying images, and their accuracy can be increased through hyperparameter tuning (such as through modifying learning rate, batch size, size of layers, etc.). Batch size increases cause the CNN to perform worse, and learning rate increases cause the CNN to perform worse. Increasing the number of convolutional layers decreased the model's final accuracy, and numbers of linear layers less than or greater than 2 caused worse accuracies. Fewer parameters lowered accuracy. Fully connected neural networks underperform compared to convolutional neural networks. Kernel sizes of 5 were the best. The discussion implies that the hypertuning of neural networks cannot be overstated, and that convolutional neural networks work well when attempting to identify data. These results are significant in demonstrating the extent to which convolutional neural network performance can be stretched under hyperparameter tuning, showing the relevancy in continuing to see the efficiency of CNNs for different kinds of image identifications, and its relevance in future studies which examine how to increase performance with neural networks. To further advance the state of this field, convolutional neural network performance should be further explored through fine tuning even more parameters (such as the number of layers, channels, kernels), and by applying it to other datasets.

References

1 R. Castro, *pytorch-hands-on*, GitHub, 2019, `https://github.com/rafaela00castro/pytorch-hands-on/blob/master/mnist_cnn.ipynb`.

2 3Blue1Brown, *But what is a neural network? | Chapter 1, Deep learning [Video]*, Youtube, `https://www.youtube.com/watch?v=aircAruvnKk`.

3 C. Course, *Computer Vision: Crash Course Computer Science #35 [Video]*, Youtube, 2017, `https://www.youtube.com/watch?v=-4E2-0sxVUM`.

4 3Blue1Brown, *What is backpropagation really doing? | Chapter 3, Deep learning [Video]*, Youtube, https://www.youtube.com/watch?v=Ilg3gGewQ5U&t=636s.

5 Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, *Proceedings of the IEEE*, 2002, **86**, 2278–2324.

6 A. Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*, 2009, https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

7 F. Nazari and W. Yan, *Convolutional versus Dense Neural Networks: Comparing the Two Neural Networks' Performance in Predicting Building Operational Energy Use Based on the Building Shape*, 2021, https://arxiv.org/pdf/2108.12929#:~:text=A%20Dense%20Neural%20Network%20.

8 S. Alford, R. Robinett, L. Milechin and J. Kepner, 2019 IEEE High Performance Extreme Computing Conference (HPEC), 2019, pp. 1–6.

9 L. Cao and A. Choe, *Intro to AI for High Schoolers*, 2022, https://www.amazon.com/dp/B0BGFRTB4M.