Comparing and Contrasting a Simple Convolutional Neural Network against LeNet-5 Using Two Different Datasets

Hayden Labrie, hcl18001, hcl180001@utdallas.edu
Shane Ray, smr180013, smr180013@utdallas.edu

The University of Texas at Dallas
Computer Science Department

Abstract **(Hayden: 70%, Shane: 30%)**

Machine learning is an extremely fast-growing market for computer scientists to solve various problems. A Forbes article states that "The global machine learning market was valued at $1.58B in 2017 and is expected to reach $20.83B in 2024" [1]. There is an overall bullish market for the use of machine learning in many different areas of the workforce.

Machine learning is powerful at solving problems such as predicting what object is in an image; thus, this is what the paper will be focusing on, image detection using CNNs. This paper will provide a solution on image detection for both the CIFAR10 dataset and the MNIST dataset. Provided is the background needed to understand CNNs with the introduction of artificial neural networks, how it ties into CNNs, and then how CNNs work. Later the paper dives into how our CNN gets constructed step by step in a way that beginners of machine learning can still grasp the concepts and implement these techniques into their own work. The last main part discussed is data analysis to see how well our machine learned to detect the objects we were trying to teach it using our CNN and compare it to a popular CNN called LetNet.

1. Introduction **(Hayden: 50%, Shane: 50%)**

Image prediction is one of the subsets of machine learning that can be used as a powerful tool in many applications. The system used to implement image predictions is called a convolutional neural network.

A convolutional neural network comes from artificial neural networks. An artificial neural network is made from 3 main parts: data that we want to learn about, 1 or more hidden layers, and an output layer.

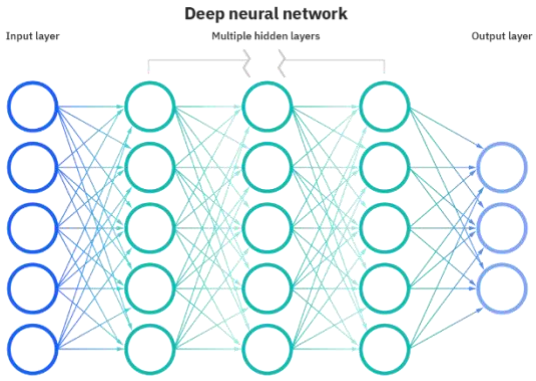


Fig 1. A high-level example of a deep neural network structure

The example [fig 1] shows an artificial neural network with a data layer, 3 hidden layers, and an output layer. One can think of an example such as an artificial neural network that can learn how to predict the price of a car. Important data that effects a cars price are brand, new or used, miles driven, engine type, color, etc. When looking at the model above you can see that the input nodes, which are all the far-left circles, are pointing to a variety of the first layer of hidden nodes. What the model is depicting here is a non-linear function of some of the input

nodes that will be produced at each given hidden layer 1 node. For better understanding I believe an example will help clarify what is trying to be expressed in this model. Let each input node be expressed as a value by x1 through xn where n is the total number of input nodes, thus we can say node one has a numerical value of x1. Also, let all hidden layer 1 nodes be expressed as a value by y1 through ym where m is the total hidden layer 1 nodes. A possible nonlinear function for y1 can be y1 = 2x1^3 + 4x3 – x4. Each layer 1 node gets some possible function from the combinations of the input nodes. Layer 2 of the hidden nodes gets the combination from layer 1 of the hidden nodes. This process keeps happening until we hit the output nodes which will give the results; We can analyze the results and pick which output node is the most accurate. This is the fundamental idea behind artificial neural networks.

One may ask, well what is the point of a CNN (Convolutional Neural Network)? Thus, this is where we get to the issues that artificial neural networks face. The elephant in the room for artificial neural networks is the method of analyzing images and coming out with output predictions from it. The inevitable question comes, what should one then use for input to do image analysis? Well, what brilliant people have found out for us is to do image convolution and then use that as the input. Image convolution is a process that uses matrix manipulation to get important features out of an image such as vertical and horizontal lines. The features are expressed in a new image called a feature map where the image is also stored in a matrix. The process of getting these features is having an image represented by a matrix, then having a convolution kernel run across the whole image to generate the feature map. The feature map is a function of the kernel convolution manipulation on the input image fig[2].

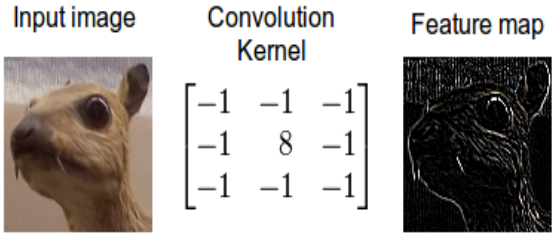


Fig 2. Image demonstrating the process of getting a feature map by the convolution kernel

Below is a formal mathematical definition of convolution.

$$y(t) = f \otimes x = \int_{-\infty}^{\infty} f(k) \cdot x(t-k)\, dk$$

Fig 3. Convolution formula

Another question comes upon us, how do we know what features are important to use as input? Well to put it bluntly, we don't know what features are the most important for image detection. So, what shall we do; we must make many different feature maps and let the machine learn what features are the best. Let's say we are trying to figure out what animal is in the picture; well, what we can do is run many different convolution kernels on the image to get different features out of the picture and have that as the input nodes. Then on these inputs we can run more convolution kernels for each input and get multiple feature maps from the input feature map; this is basically adding more and more abstraction. Thus, what is happening is we get multiple feature maps from the original image, then on the second layer we get multiple feature maps for each original feature map, and then we can continue these layers until there is one pixel left. At the very end there is a neuron that will give the output for what animal is in the picture or whatever is wanted. We can train this CNN by back propagation so that the correct combination of convolution kernels are done on the image to produce the highest chance of being correct An example of the transformation process can be seen below [fig 4].

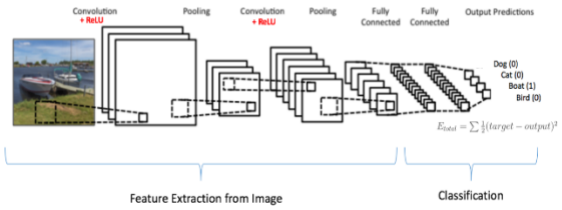


Fig 4. A CNN model going through the different layers.

Now with the fundamental knowledge of how a CNN works I will present the problem being presented. The task at hand is to create a CNN which learns what object is in an image for the CIFAR10 and the

digits in the MNIST dataset. We construct a CNN that both of these datasets will use so that the machine can learn, and then provide the accuracy of predicting the objects and digits in the datasets.

2. Related Work – (Hayden: 40%, Shane: 60%)

Image classification has been one of the main areas of machine learning research dating back to the 1980's[2]. The problem of classifying and recognizing the images of the CIFAR-10 and MNIST data sets has been tested and optimized by many researchers and university students over the years [5, 6]. Originally the LeNet-5 network was developed by Yann LeCun in the 1980's and was one of the first ever convolutional neural networks used to classify handwritten digits[5]. This would be akin to the MNIST dataset used in our analysis. Hence, the work of our nature has been going on for many years. Naturally, the comparison of different convolutional neural networks on the same dataset is a product of finding the best solutions. It is in the nature of optimization and image classification to find the best solution for each time of classification.

A recent study dating to November of 2018 entitled "*Assessing four neural networks on handwritten digit recognition dataset (MNIST)*" out of Cornell University, with researchers Feiyang Chen, Nan Chen, Hanyang Mao, and Hanlin Hu, compared four different neural networks on the handwritten digit recognition dataset (MNIST)[3]. The four different neural networks they used are a Convolutional Neural Network (CNN)[7], Deep Residual Network (ResNet)[2], Dense Convolutional Network (DenseNet)[3], and an improved upon CNN baseline through introducing Capsule Network [3]. A general overview of the CapsNet architecture is shown below fig[5].

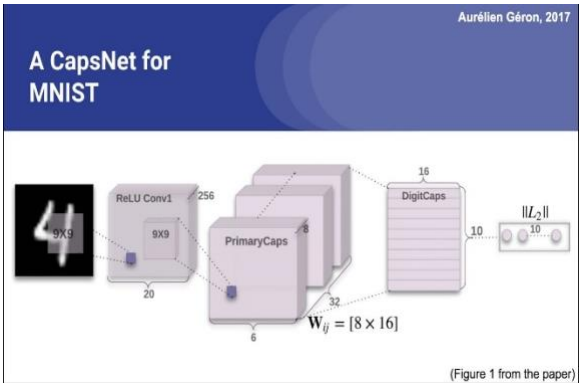


Fig 5. A high level view of the CapsNet architecture

Their goal of the study was to improve upon their baseline CNN to see if their model could outperform the other three models of the study. To the researchers' success they were able to achieve a 99.78% accuracy rate claiming that it has been the best publish accuracy rate up until that date in November of 2018 [3]. The researchers intended to go on and improve their CNN to further classify and recognize other images from other datasets

A study entitled "*Batch Normalization in Convolutional Neural Networks — A comparative study with CIFAR-10 data' published* " [4] published by Fifth International Conference on Emerging Applications of Information Technology (EAIT) in 2018 did a comparative study of four different convolutional neural networks on the CIFAR-10. The researchers were particularly trying to find out the effects of data normalization and how it affects the accuracy of predicting the classification of the images. The convolution neural networks used were DenseNet, VGG, Inception (v3) Network, and Residual Network. Each network ran for a total of 200 epochs. The researchers were able to achieve accuracies of 82.68%, 88.79%, 81.01%, and 84.92%, respectively[4]. Below is a representation graph of the author's findings fig[6].

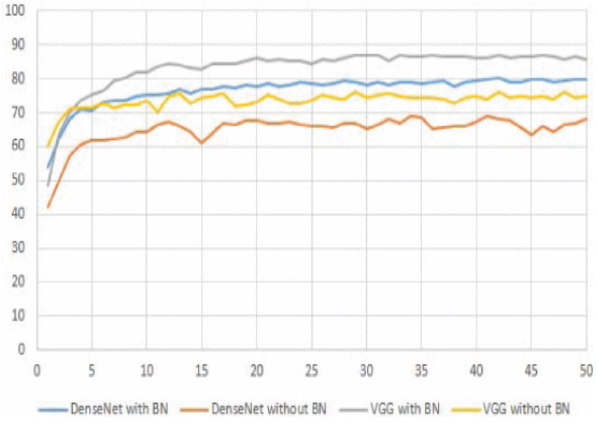


Fig 6. Results of comparing 4 different convolutional neural networks on the CIFAR10 dataset

3. Our Approach - **(Hayden: 60%, Shane: 40%)**

Our solution to the problem starts with getting the training data loaded into our project and splitting it into 2 subgroups; 1 subgroup being 90% of the images and the other subgroup being 10% of the images. We make 2 subgroups so that we can have a set to train the simple CNN and have a set to test the simple CNN for its accuracy; of course the 90% will be the training set and the 10% will be the testing set. We do this by calling random_() function with values being 45,000 and 5,000 which corresponds to 90% of 50,000 equal to 45,000 and 10% equal to 5,000. Then after we get the random 9/10 and 1/10 sets we define the objects that we are trying to identify from the CIFAR10 image set. From the CIFAR10 dataset we use 10,000 test samples. We define an array of objects that map to an individual image where an object is a class: classes = ('plane', 'car', 'bird', 'cat','deer', 'dog', 'frog', 'horse', 'ship', 'truck'). Thus, as one can see we are trying to detect a plane, car, bird, cat, deer, dog, frog, horse, ship, and truck [fig 7].

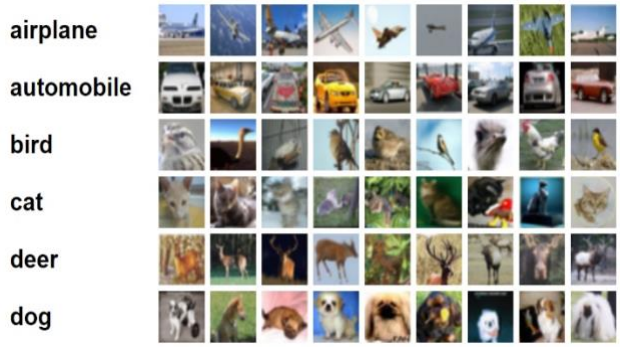


Fig 7: Example of the CIFAR10 images

We also use the same simple CNN on the MNIST dataset which is a bunch of images of digits that students wrote. The MNIST has a collection of 70,000 samples. 60,000 training samples and 10,000 testing samples. We will split the training samples to a ratio of 90/10, where 90% of the training set will be used for training and 10% of the samples will be used for validation. This comes to 54,000 training samples and 6,000 validation samples. Our goal is to correctly identify the objects in the CIFAR10 dataset and the digits in the MNIST dataset.

After we have correctly gotten the input in our project and the random 9/10 and 1/10 sets we implemented a CNN. Before I dive further into the simple CNN creation first I would like to provide a good definition of what a tensor is. University of Cambridge states “Tensors are simply mathematical objects that can be used to describe physical properties, just like scalars and vectors. In fact tensors are merely a generalization of scalars and vectors; a scalar is a zero-rank tensor, and a vector is a first rank tensor.”[7]. To put more clearly, we can think of a Tensor as the matrix that is representing the picture as a mathematical object. Each pixel is represented by some numerical value and all pixels of the picture get mapped to some numerical value.

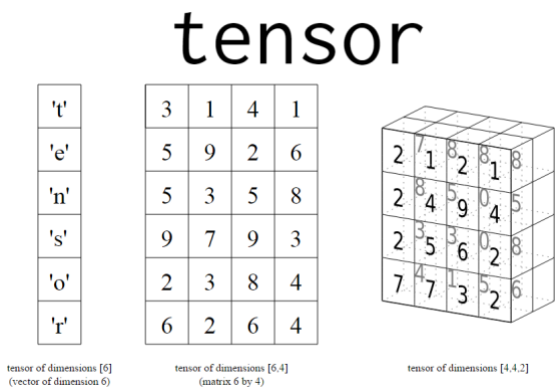


Fig 8. Example of 1d, 2d, and 3d tensors

Thus, what we are working with is ultimately a 3d tensor. Why do I say 3d when the picture is obviously only composed of 2 dimensions? Well we have to keep in mind that the image is colored by using 3 main colors, being red, green, and blue (rgb). Thus we can actually think of the tensor object having depth as well as length and width.

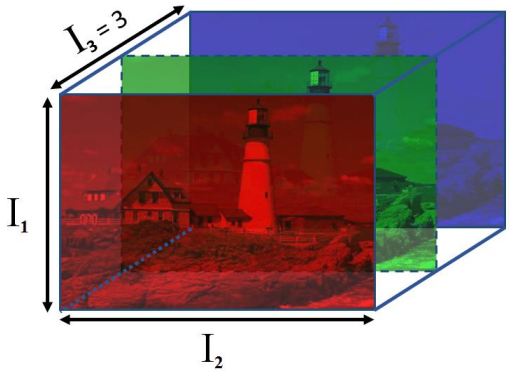


Fig 9. Depiction of a rgb image as 3 dimensions

The first part of our CNN starts with the conv2d() function. When calling this function we give it 3 parameters. The first parameter is 3 which indicates there are 3 channels. We are saying there are 3 channels because the images we are analyzing are rgb, thus it has 3 layers of matrices that we need to feed into the CNN. The next parameter is 32 which indicates how many filters we want to call on the matrix. Pytorch will provide 32 random filters to run on these tensors. The parameter is the kernel size which we set to 3. The kernel is the convolutional kernel matrix that was discussed in the introduction on how convolutions work. Also it is important to realize that each channel gets their own tensor that gets translated to a feature map by the conv2d() function.

The second function we call for our CNN is maxpool2d(). Before diving into this pytorch function it is important to understand what max pooling is. "Max Pooling is a pooling operation that calculates the maximum value for patches of a feature map, and uses it to create a downsampled (pooled) feature map. It is usually used after a convolutional layer. It adds a small amount of translation invariance - meaning translating the image by a small amount does not significantly affect the values of most pooled outputs"[8].

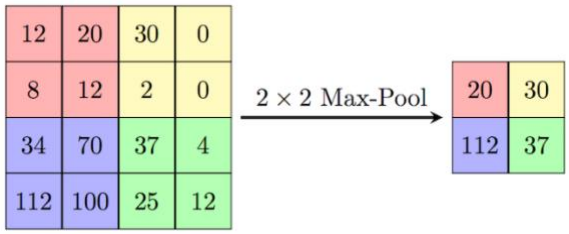


Fig 10. Max pooling of a matrix

Max pooling basically extracts the most important features for each given area of a feature map and then puts those features into a more compact matrix. It is often called downscaling when referring to taking the matrix and then reducing it to a smaller size, though the matrix still has the most important information.

We called a maxpool2d(2) function in our CNN. The parameter entered, being 2, represents how big of a kernel we want to create the max pooled matrix. The value is 2, thus this means we have a matrix of 2x2 go across each section of the feature map matrix to create the new reduced matrix. One can think of it as grouping each 4 cells in the feature map matrix, mapping each group of cells to one cell in the new matrix, and having the biggest number in the group be mapped.

The next thing in building our CNN we do linear transformation on the matrix. The official linear transformation definition for pytorch is "y=xAT+b" [9]. Let's discuss this function of A → y in more detail. First let's look at A^T. A represents some matrix, and A^T represents the transpose of matrix A. I believe an example will give a clearer picture to the readers of what is happening.

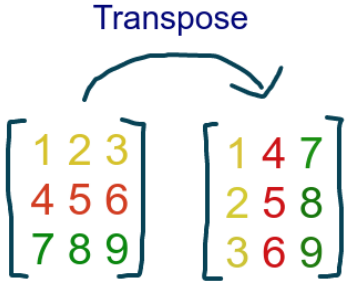


Fig 11. Transpose function mapping A to A^T

As seen in the image above you can see how the transpose function maps the original matrix A to A^T. The rows in A become the columns in A^T. The next part in our linear transformation function is x. All x is doing is acting as a scalar on A^T. Scaling a vector is pretty simple after some practice, thus I won't dive in too much detail, though I will provide an example below so that readers can understand the fundamentals of scaling a matrix.

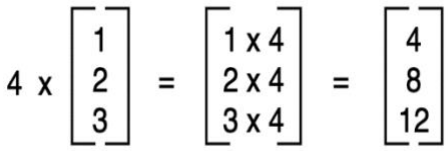


Fig 12. Example of scaling a matrix

The last part of the linear transformation to be discussed is +b. This part is simply matrix addition.

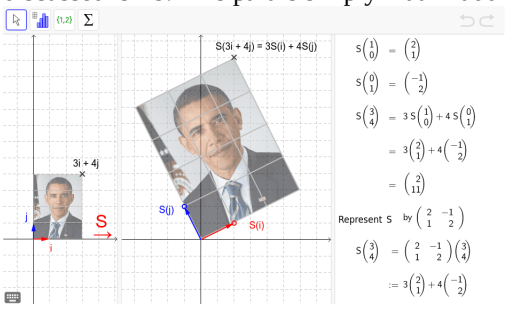


Fig 12. Example of linear transformation

So ultimately what we are getting is some type of image that is tilted and stretched by the linear transformation function. After all these manipulations are done on the original input we send them to 10 output nodes, with each output representing 1 class like “cat” or “car” for CIFAR10 and “0” or “8” for MNIST. Then, we let our CNN learn and see how accurate our results are.

We will use the LeNet-5 architecture in its LeNet-5 form to compare and contrast with our simple CNN. Below is an overview of its architecture fig[14]

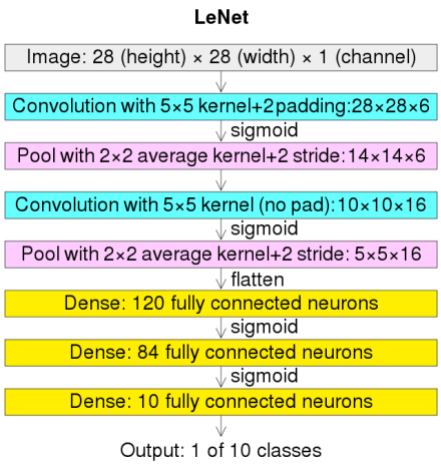


Fig 14. High level view of the LeNet5 architecture

We will use this layering technique for the LeNet-5. Convolutional layer 1 will have an input of 3 channels(rgb), an output of 6 channels, a kernel size of 5 and a stride size of 1. Convolutional layer 2 will have an input of 6 channels, an output of 6 channels, a kernel size of 5 and a stride size of 1. Both of the pooling layers will be MaxPools() and will be 2x2. Finally, the three fully connected layers will follow the LeNet-5 structure from 120, 84, 10 fully connected layers.

Both the simple CNN and the LeNet-5 network will be updated with the same optimization and loss functions. The loss for each epoch will be calculated with pytorches cross entropy loss function fig[14] .

$$\ell(x, y) = \begin{cases} \frac{\sum_{n=1}^{N} l_n}{N}, & \text{if reduction} = \text{‘mean’;} \\ \sum_{n=1}^{N} l_n, & \text{if reduction} = \text{‘sum’.} \end{cases}$$

Fig 14. Pytorch formula for cross-entropy loss

Cross-entropy can be defined as “ a measure from the field of information theory, building upon entropy and generally calculating the difference between two probability distributions. ….. Cross-entropy is a measure of the difference between two probability”distributions for a given random variable or set of events.” [10].

We will be using stochastic gradient descent (SGD) as an optimizer to out training model. SGD “is an optimization algorithm often used in machine learning applications to find the model parameters that correspond to the best fit between predicted and actual outputs. It’s an inexact but powerful technique”[11]. Each CNN model will run for a total of 25 epochs in an attempt to keep the learning time as close to the same for all models. The learning rate of each model will vary between .01 and .00001. These learning rates were adjusted according to each model by how much overfitting was happening and how fast or slow the model’s accuracy rates incremented per epoch. Batch size for simple CNN is 32 and for LeNet-5 is 64. This decision was made after trial and error concerning validation loss.

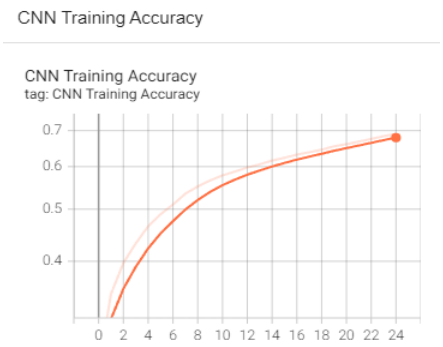
We will compare the two CNN’s in terms of training accuracy/loss, validation accuracy/loss, and most importantly, in our work, compare prediction accuracy with the test sets from each dataset. We will visualize the predictions using a heat map of confusion-matrix. A confusion-matrix gives insight into what classes were predicted correctly and what classes were predicted incorrectly.

4. Analysis **(Hayden: 50%, Shane: 50%)**

In this section I will discuss how our simple CNN performed on the MNIST dataset as well as the CIFAR10 dataset and compare it to a popular CNN called LeNet-5.

Before diving into the analysis I would like to define a few things that should make digesting the data easier for the readers. The first dichotomy we must understand is training accuracy and validation accuracy. Recall that we train our machine by sending predetermined data and tell it what type of class it should map to for the output. Well we can think of this set as the training data set; thus, when we refer to training accuracy what we are referring to here is how accurate the system is at detecting the exact same images that we trained the system on. The validation accuracy is referring to the image set that

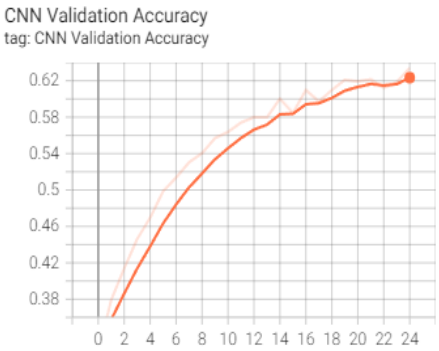
we have not used to train the system with. We can think of the training accuracy as the accuracy on the 90% set that has been trained on and the validation accuracy as the 10% set that the system has not been trained on.



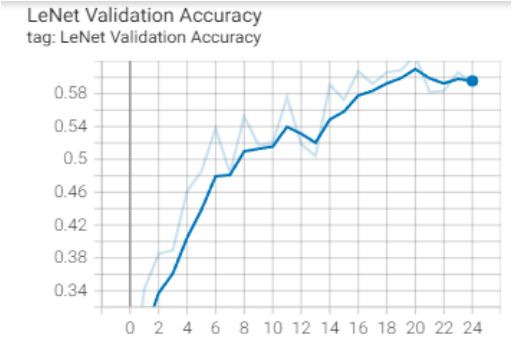
Above is the simple CNN: function that maps epochs to training accuracy for CIFAR10. The x-axis represents the number of epochs (time) and the y-axis represent the training accuracy.

above is the LeNet-5 CNN: function that maps epochs to training accuracy for CIFAR10. The x-axis represents the number of epochs (time) and the y-axis represent the training accuracy after each epoch.

The images of the simple CNN and LeNet-5 network show the training accuracy for the CNN we created vs the LeNet-5 CNN. As you can see the LeNet-5 performed better than our CNN on the training dataset, though this is of lesser importance for the training vs validation sets. The validation set accuracy is of more importance because it is testing the CNNs on images that it has never seen, thus performing its ultimate purpose.

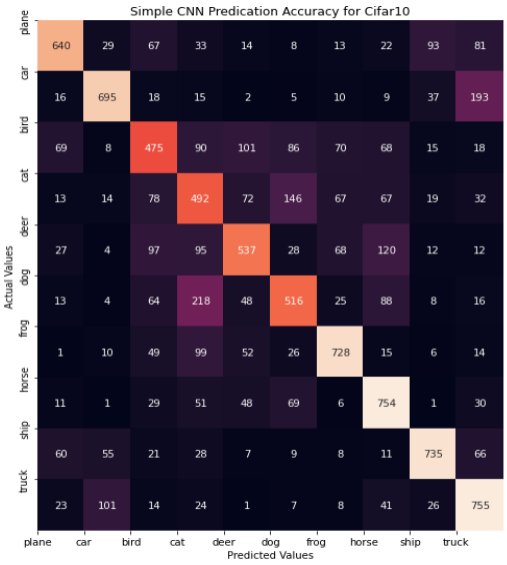


Above is the simple: function that maps epochs to validation accuracy for CIFAR10. The x-axis represents the number of epochs and the y-axis represent the validation accuracy.



Above is the LeNet-5 CNN: function that maps epochs to validation accuracy for CIFAR10.The x-axis represents the number of epochs (time) and the y-axis represent the validation accuracy after each epoch iteration.

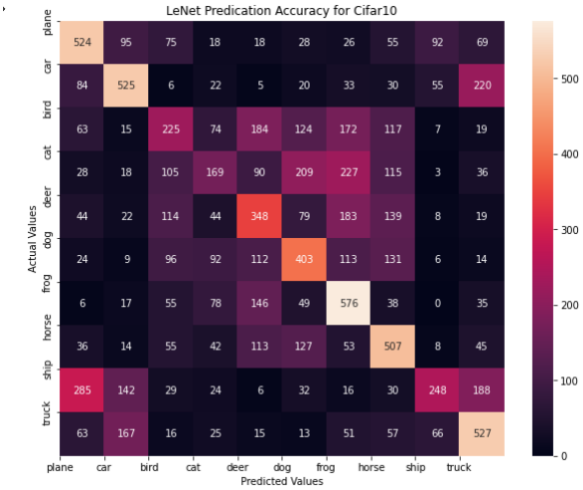
Now when comparing the validation accuracy between the 2 CNNs we can see that the CNN we developed performed better than the LeNet CNN. This empirical evidence shows our CNN to be better at detecting new images in the dataset compared to the LeNet-5 CNN. Validation accuracy and loss visualization and calculation for our simple CNN and for the LeNet-5 on the CIFAR 10 dataset was done by Hayden Labrie.



Our simple CNN: Prediction accuracy for each object in the CIFAR10 dataset

The first confusion-matrix (bottom left) shows how accurate our CNN was at predicting each image in the CIFAR10 test dataset. The y axis of the matrix represents the actual image, while the x axis represents the predicted image. The correct predictions are along the diagonal of the matrix. Each type of object has a total of 1000, thus a perfect prediction on the dataset would be all 1000s on the diagonal of the matrix. When looking at the matrix we can see that position [0,0] is 640; thus, we can say it predicted 640 out of 1000 images of planes correctly, or 640/1000 = 0.64 => 64% accuracy. If we look at the element to the right of the first entrance, which can be denoted as [0,1] it has a value of 29. This value represents that of the 1000 plane images being tested on the CNN 29 were predicted as being a car rather than the plane that the image is. If we add all correct predictions on a row plus all incorrect predictions, we will get 1000. For example, the second row we see there are 695 correct predictions at [1,1]. If we add [1,0] through [1,n-1], where n represents the amount of different objects, we will get 16 + 695 + 18 + 15 + 2 + 5 + 10 + 9 + 37 + 193 = (total car images worked on) = 1000. This holds true on all rows.

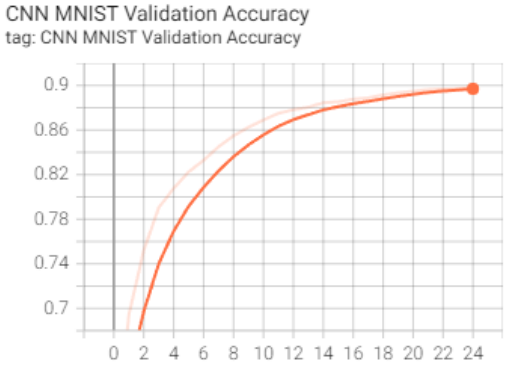
Some interesting characteristics to note is that our simple CNN performed best at truck detection, and barely worse on horse detection. Our simple CNN performed the worst on bird detection at only 47.5% accuracy. The highest incorrect object relation was cat and dogs. When presented with a dog 21.8% of the images were thought of as being a cat. We still see a strong relation of this incorrect prediction when showing an image of a cat but getting a prediction of a dog, though at a lesser extent of 14.6% being thought of as dogs. I can see why this would happen; cats and dogs have fairly similar characteristics, especially if it's a smaller dog with medium hair and short ears.



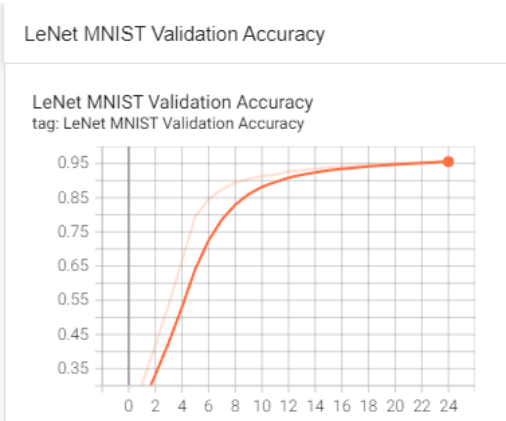
CM2. LeNet-5 CNN: Prediction accuracy for each object in the CIFAR10 dataset

The LeNet-5 CNN seen above has definitely had a tougher time getting accurate results compared to our CNN. We can see it struggled a lot with cat detection with only 16.9% accuracy. Surprisingly, dogs were not the most mistaken object, but frogs were the highest at 22.7% of all cats being mistaken as frogs while 20.9% were mistaken as dogs. It performed the best with frog detection at 57.6% of frog images being predicted correctly, though less than our CNN where 72.8% of frog images were detected.

Now let's discuss the other dataset we applied the 2 CNN to, being the MNIST.

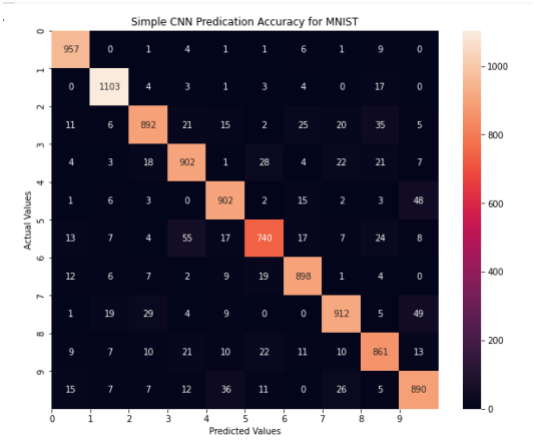


Our CNN: function that maps epochs to validation accuracy for MNIST



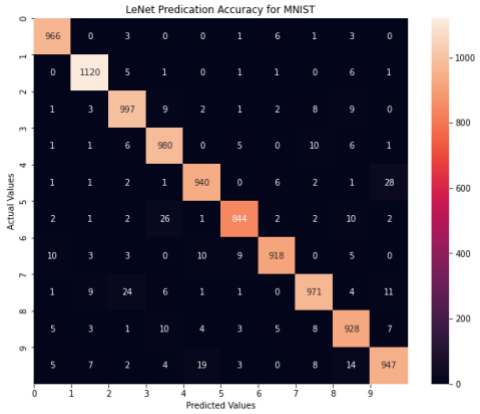
LeNet-5 CNN: function that maps epochs to validation accuracy for MNIST

As you can see from the diagrams the MNIST did perform better on the LeNet-5 CNN rather than our CNN. When we compare validation accuracy analysis the LeNet achieved above 95% while the simple CNN only achieved 90%.



Our simple CNN: Prediction accuracy for each object in the MNIST dataset

As you can see from the matrix above, the highest correctly predicted image is the number 1 for our CNN. Our CNNs lowest accuracy was on identifying number 8. Overall prediction son the test set were great.



The LetNet-5 CNN performed overall better at predicting the correct numbers and having the best prediction being for 1 and worst prediction being for 5. Overall prediction son the test set were great.

5. Conclusion (Hayden: 60%, Shane: 40%)

From the analysis results we can see that the CNN we created did better on the CIFAR10 dataset, though it performed worse on the MNIST dataset. The LetNet skyrocketed in accuracy on the MNIST data set and seems to be designed to perform detection on symbols rather than more complicated objects like cars and cats in the CIFAR10 dataset. A possible theory is that LeNet-5 does not do well with a lot of noise caused by stuff in the background that digits in the MNIST dataset does not have.

Overall I believe the reader can have a good understanding of how a CNN works and see techniques on how to compare CNNs using different approaches to analysis. CNNs are very powerful at teaching a machine image detection while only requiring just a little bit of linear algebra to understand what is going on. Machine learning is not going away and will only continue to impact our lives more and more through all avenues. It would be in many computer scientist's best interest to understand at least the fundamentals of machine learning, because we will probably see companies looking for these approaches to solve problems more and more as time goes.

6. Coding Distributions

Shane Ray

- Coded the sections of bringing in and formatting the datasets for all of the Neural Networks
- Coded the LeNet-5 CNN,
- Coded training parameters for LeNet, training and validation for LeNet on both datasets,
- Coded the tensorboard code for all tensorboard visualizations,
- Coded the confusion matrix calculations and visualization

Hayden Labrie

- Coding for the simple CNN,
- Coded the dataset visualization for both the Cifar-10, and MNSIT,
- Set up device gpu usage,
- Coded training parameters for simple CNN,
- Coded training and validation calculations for simple CNN on both datasets
- Coded the testing for simple CNN on both data sets

7. References **(Hayden: 40%, Shane: 60%)**

[1] .L. Columbus, “Roundup of machine learning forecasts and market estimates, 2020,” *Forbes*, 23-Jan-2020. [Online].Available:https://www.forbes.com/sites/louiscolumbus/2020/01/19/roundup-of-machine-learning-forecasts-and-market-estimates-2020/?sh=92286515c020. [Accessed: 12-May-2022].

[2]. B. D. -, By, B. Dickson, -, Ben DicksonBen (CNN)?,” *TechTalks*, 06-Jan-2020. [Online]. Available: https://bdtechtalks.com/2020/01/06/convolutional-neural-networks-cnn-convnets/. [Accessed: 12-May-2022].

[3]. F. Chen, N. Chen, H. Mao, and H. Hu, “Assessing four neural networks on handwritten digit recognition dataset (MNIST),” *arXiv.org*, 20-Jul-2019. [Online]. Available: https://arxiv.org/abs/1811.08278. [Accessed: 12-May-2022].

[4]. V. Thakkar, S. Tewary and C. Chakraborty, "Batch Normalization in Convolutional Neural Networks — A comparative study with CIFAR-10 data," *2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT)*, 2018, pp. 1-5, doi: 10.1109/EAIT.2018.8470438.: 13-May-2022].

[5]. “The mnist database,” *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges*. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 13-May-2022].

[6]. *CIFAR-10 and CIFAR-100 datasets*. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html. [Accessed: 13-May-2022].

[7]. Department of Materials Science and Metallurgy - University of Cambridge, *What is a tensor?* [Online]. Available: https://www.doitpoms.ac.uk/tlplib/tensors/what_is_tensor.php. [Accessed: 13-May-2022].

[8]. “Papers with code - max pooling explained,” *Explained | Papers With Code*. [Online]. Available: https://paperswithcode.com/method/max-pooling. [Accessed: 13-May-2022].

[9]. “Linear,” *Linear - PyTorch 1.11.0 documentation*. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Linear.html. [Accessed

[10]. J. Brownlee, “A gentle introduction to cross-entropy for Machine Learning,” *Machine Learning Mastery*, 22-Dec-2020. [Online]. Available: https://machinelearningmastery.com/cross-entropy-for-machine-learning/. [Accessed: 13-May-2022].

[11]. Real Python, “Stochastic gradient descent algorithm with python and NumPy,” *Real Python*, 19-Jan-2021. [Online]. Available: https://realpython.com/gradient-descent-algorithm-python/#:~:text=Stochastic%20gradient%20descent%20is%20an,used%20in%20machine%20learning%20applications. [Accessed: 13-May-2022].

8. Image References **(Hayden: 40%, Shane: 60%)**

[1]: shauryafulfagar1. “AI-BLOG/INDEX.HTML at Main · Shauryafulfagar1/AI-Blog.” *GitHub*, https://github.com/shauryafulfagar1/ai-blog/blob/main/index.html.

[2]: “Convolution.” *NVIDIA Developer*, 24 Sept. 2018, https://developer.nvidia.com/discover/convolution.

[3]: “Convolution.” *NVIDIA Developer*, 24 Sept. 2018, https://developer.nvidia.com/discover/convolution.

[4] Pavansanagapati. “A Simple CNN Model Beginner Guide !!!!!!” *Kaggle*, Kaggle, 8 June 2020,https://www.kaggle.com/pavansanagapati/a-simple-cnn-model-beginner-guide.

[5]*Https://I.ytimg.com/Vi/WTE_Ys4iwM/Maxresdefault.jpg*.https://www.youtube.com/watch?v=ELQ20u19yXw.

[6]V. Thakkar, S. Tewary and C. Chakraborty, "Batch Normalization in Convolutional Neural Networks — A comparative study with CIFAR-10 data," *2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT)*, 2018, pp. 1-5, doi: 10.1109/EAIT.2018.8470438.

[7]. Ardi, Muhammad. “CIFAR-10 Image Classification.” *Medium*, Becoming Human: Artificial Intelligence Magazine, 6 Apr. 2021, https://becominghuman.ai/cifar-10-image-classification-fd2ace47c5e8.

[8]. Uniqtech. “Understand Tensors and Matrices.” *Medium*, Data Science Bootcamp, 3 Jan. 2022, https://medium.com/data-science-bootcamp/understand-tensors-and-matrices-2ea361e303b8.

[9] *Quantifying Blur in Color Images Using Higher Order Singular Values*. https://www.researchgate.net/publication/307091456_Quantifying_Blur_in_Color_Images_using_Higher_Order_Singular_Values.

[10] “Papers with Code - Max Pooling Explained.” *Explained | Papers With Code*, https://paperswithcode.com/method/max-pooling.

[11] “Numpy How to Transpose a Matrix.” *Codingem*, 7 Feb. 2022, https://www.codingem.com/numpy-transpose-matrix/.

[12]GreekDataGuy. “A Complete Beginners Guide to Matrix Multiplication for Data Science with Python Numpy.” *Medium*, Towards Data Science, 9 Apr. 2020, https://towardsdatascience.com/a-complete-beginners-guide-to-matrix-multiplication-for-data-science-with-python-numpy-9274ecfc1dc6.

[13] /u/mon+Prof+De+Maths+Est+Genial. “Copie De Matrix and Linear Transformation (HTML5 Version).” *GeoGebra*, 2 Apr. 2020, https://www.geogebra.org/m/zwvdfqke.

[14] “Crossentropyloss¶.” *CrossEntropyLoss - PyTorch 1.11.0 Documentation*, https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html.