JOURNAL OF
EMERGING INVESTIGATORS

Article

The effect of activation function choice on the performance of convolutional neural networks

Shin-Hao Wang[1], Eric Sakk[2]
[1]Kang Chiao International School, Taipei City, Taiwan
[2]Department of Computer Science, Morgan State University, Baltimore, Maryland

SUMMARY

With the advance of technology, artificial intelligence (AI) is now applied widely in society. For example, natural language processing, speech recognition, and autopilot are all famous examples of how AI is changing our world. In the study of AI, machine learning (ML) is a subfield in which a machine learns to be better at performing certain tasks through experience. This work focuses on the convolutional neural network (CNN), a framework of ML, applied to an image classification task. Specifically, we analyzed the performance of the CNN as the type of neural activation function changes. Choosing the right neural activation function is crucial to prevent the loss of important trends and increase the efficiency of training time. Among all the different widely used activation functions, we hypothesized that a rectified linear unit (ReLU) would be the most efficient in training time and attain the highest accuracy on the image recognition task because ReLU has the advantage of avoiding the vanishing gradient problem and only requires light mathematical calculations. Having high accuracy and efficiency in the task of image classification is beneficial as this technique can be employed in many different real-world applications, such as diagnosing in healthcare, identifying potential threats in security, or developing an autonomous vehicle. Our results indicate that when the number of hidden layers is small, networks employing the ReLU performed similarly to networks using hyperbolic tangent, and both networks with ReLU and hyperbolic tangent outperformed networks using the sigmoid function.

INTRODUCTION

Artificial intelligence (AI) is a popular and powerful branch of computer science. Machine learning (ML) is a specific type of AI that focuses on constructing a system that improves automatically through experience (1). Recently, with advances in the available online datasets and newly developed algorithms, ML has progressed significantly in multiple different fields, including but not limited to health care, education, finance, and policing (1). In this paper, we aim to discuss a specific application of ML algorithms: image recognition. Image recognition is already in use within our society. Facial recognition, handwriting recognition, and medical image recognition are a few examples of their applications (2). ML models are used for 2D image analysis, and convolutional neural networks (CNNs) outperform other architectures, such as deep belief networks and support vector machines (3).

In CNNs, there are artificial neurons that simulate the behaviors of the networks in human brains (4). The network comprises mainly three sections of layers, the convolution layers, the pooling layers, and the fully connected layer (3). In the convolution layers, the network uses a kernel, a grid of weights that serves to extract features from the input (5). The weights can be adjusted after training over multiple epochs. In the convolution layers, the input is processed by element-wise multiplication with the kernel grid. Thus, the feature can be extracted from the original input (5). Next, the pooling layers serve to down-sample the data into smaller dimensions without losing the general features (5). This process can be done using techniques such as max-pooling or average-pooling, where the values are calculated into smaller sizes (5). The neurons are weighted and interconnected, and the information of the neurons is processed to the next layer. Lastly, the information will be passed on to the fully connected layer as a flattened vector that classifies the results (5). The network includes a loss function on this last layer that calculates the prediction error. For instance, soft-max, cross-entropy, or Euclidean error are examples of loss functions in ML (3).

Additionally, in CNNs, a problem of overfitting may occur (6). Overfitting is a problem in which the machine adapts to the training data too well (6). The units co-adapt with each other to the extent that they cannot generalize on test data other than the training set (6). Therefore, when the CNN is used with unseen data, the performance of the model decreases even though the performance with seen data can be decent. To tackle this problem, the method of including dropout layers is often introduced to the network (6). Dropping out each neuron with a certain probability creates possible combinations of the network (6). Every reduced network is called a thinned network (6). Therefore, training the network becomes training multiple different thinned networks, with weight sharing as the weights are the same, reducing the problem of overfitting (6).

Finally, an essential part of CNNs is the activation function. Before the neurons pass information to the next layer, the output is modified by an activation function, which often adds nonlinearity to the network (4). With complex distributions of the data, choosing a linear function as the activation would only introduce the network with linear complexity (4). The network would not be able to deal with images with complex data systems or data distribution (4). So, the network needs activation functions to increase the machine's ability to process complex information. In reality, the distribution of more complex data, such as image recognition, can be sparse, noisy, and skewed (4). Therefore, choosing the wrong activation function can cause the loss of important trends or information on the data, which can drastically affect the final accuracy of the ML model. Each point of the dataset can be valuable and provides much information.

Therefore, in our work, we aim to compare the performance of the CNN using different activation functions, such as Sigmoid, Hyperbolic Tangent (Tanh), and Rectified Linear Unit (ReLU), and benchmark the results in terms of training time, validation accuracy, testing accuracy, precision, recall, and F1-Score. We would build on the foundation of CNNs to determine which activation among ReLU, Tanh, and Sigmoid was the most suitable for the field of image recognition. We would thoroughly review what functions to use in image recognition and other fields related to image recognition, such as voice recognition.

While most of the existing studies emphasize networks with multiple layers and parameters (7–9), in this research, we focus mainly on a CNN with a small number of layers, aiming to discuss the results of different activation functions with a relatively small network. For general networks, ReLU is the most widely used activation function (4). For our experiment setup, we hypothesized that ReLU would have the best performance on our image recognition dataset because it avoids the problem of vanishing gradients and does not require heavy calculations. The vanishing gradient problem occurs when the gradient of the loss function is very small, close to zero, drastically slowing down the learning process (4). Sigmoid and Tanh have very small derivatives close to zero when the input is too large or small, while ReLU avoids this problem. Therefore, we hypothesized that ReLU would outperform Sigmoid and Tanh (4).

We ran a CNN using Python and TensorFlow, changing the activation function used in an image recognition task with three trials. We concluded that ReLU and Tanh performed similarly on our small CNN model. Even though the training time taken was slightly less, the advantage of ReLU in accuracy was not statistically significant. Nonetheless, ReLU and Tanh performed significantly better than Sigmoid in terms of training time, testing and validation accuracy, precision, recall, and F1-score.

The results could also be inferred or extrapolated into other neural networks requiring activation functions. We would also analyze processes and verify the existing studies comparing existing functions' performance. Our research would serve as helpful data and conclusions for researchers in the field of lightweight CNNs. Specifically, our research could benefit researchers that aim to attain high accuracy with limited hardware and time, such as for image detection in auto driving that requires high accuracy within a short amount of time (10).

RESULTS

We first built the CNN with Python code and changed the activation functions accordingly to test the results. We ran the CNN on the Modified National Institute of Standards and Technology database (MNIST) dataset, which consists of 60,000 28 by 28 pixels images in grayscale. The activation functions included in this study include ReLU, defined as f(x)=*max*(0,x), the Sigmoid function, expressed as $f(x)=\frac{1}{1+e^{-x}}$, and Tanh, represented by $f(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$. We split the dataset into training and testing sets. First, we compared the training time of the three activation functions to evaluate the different activations. Sigmoid and Tanh functions have similar training times of about 49 to 50 seconds for each epoch (**Figure 1A,**

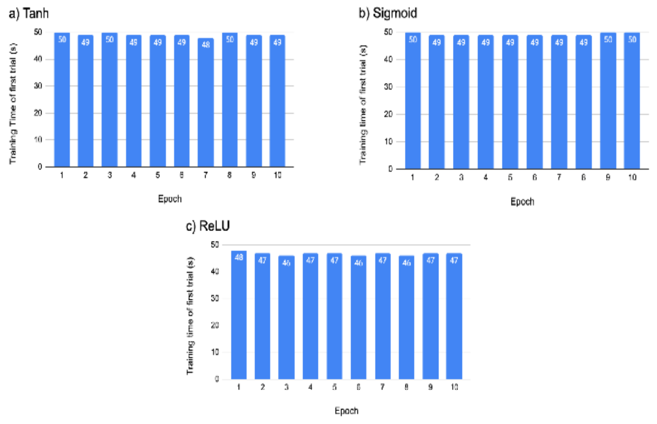

Figure 1: Training time minimized using ReLU function compared to Tanh and Sigmoid. The CNN is run with Tanh, Sigmoid, and ReLU on the training set taken from the MNIST dataset with the Keras TensorFlow module. Training time is obtained with Keras TensorFlow's function. Bar graph showing the training time of the CNN with the **A)** Tanh function, **B)** Sigmoid function, and **C)** ReLU function for each epoch. Data shown from first trial only.

B). On the other hand, ReLU had a slightly lower training time, taking about 46 to 48 seconds for each epoch (**Figure 1C**). As the results show, ReLU had better efficiency in training the model.

Next, we compared the validation accuracy that was evaluated when training. Even though Sigmoid and Tanh had significantly lower validation accuracies at the beginning of the training epochs, each of them had around 98% to 99% training accuracy in the last epoch (**Figure 2A,B**). On the other hand, ReLU had a high validation accuracy throughout the 10 epochs (**Figure 2C**). Additionally, Tanh and ReLU had slightly better accuracies in the last epoch, with about 99% compared to Sigmoid's 98% (**Figure 2**).

Last, we compared the testing accuracy, precision,

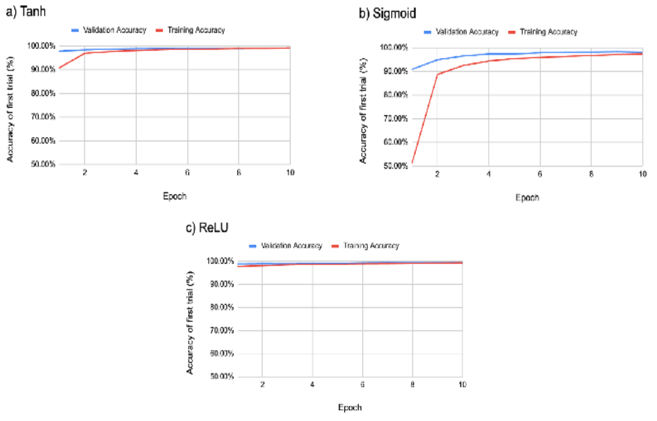

Figure 2: All three functions' training and validation accuracy increase with successive epochs; Sigmoid started with about 90% of validation accuracy while Tanh and ReLU both started at about 98% validation accuracy in the first epoch. The CNN is run with Tanh, Sigmoid, and ReLU on the training set taken from the MNIST dataset with the Keras TensorFlow module. Training and validation time is obtained with Keras TensorFlow's function. Line graph showing the training and validation accuracy with A) Tanh function, B) Sigmoid function, and C) ReLu function for all epochs. Data shown from first trial only.

Function	Accuracy	Precision	Recall	F1-score
ReLU	99.17%	99.18%	99.17%	99.17%
Sigmoid	98.20%	98.20%	98.20%	98.20%
Tanh	99.10%	99.10%	99.09%	99.09%

Table 1: Performance of the three activation functions with the test set. The table shows the results of the test accuracy, precision, recall, and F1-score of the three activation functions. The data was obtained from the trained network using the Keras Tensorflow module.

recall, and F1-score of the three functions (**Table 1**). Testing accuracy served as the final evaluation with the test set after the training was completed. ReLU and Tanh had the best accuracies above 99%, with ReLU being slightly better by a margin of 0.07%. Sigmoid performed the worst on the final accuracy, with only about 98%. For precision, recall, and F1-score, all three functions have similar values with their test accuracy; sigmoid had 98% of all the metrics, while ReLU and Tanh were slightly better with 99%.

To calculate and verify the error of the experiment, we performed the experiment three times and calculated the standard deviation of the accuracies of the trials. Tanh and ReLU had a standard deviation of about 0.00033, and Sigmoid had a standard deviation of 0.00017.

Additionally, with the data from the three trials of the experiment, we also performed ANOVA tests on the mean accuracies of the trials. Since precision, recall, and F1-score are all similar to the accuracies, we used testing accuracy as the major metric. We found a significant difference in accuracy of the three groups (p=6.3 * 10-8, f=750). Therefore, we performed Tukey's test as the post-hoc test to do pairwise comparisons among the groups. We found no significant difference in testing accuracy between the ReLU and Tanh functions (p=0.11) (11). On the other hand, we found a significant difference in accuracy between ReLU and Sigmoid (p=0.0) and Tanh and Sigmoid (p=0.0) (11).

DISCUSSION

Our results showed that ReLU had similar testing accuracies with Tanh in our model, and both ReLU and Tanh performed significantly better than Sigmoid in terms of testing accuracies. There is a trend of ReLU also being better in terms of training time, validation accuracy, and F1-score from our raw data. The Sigmoid function fell behind ReLU slightly, with a testing accuracy of 98% compared to ReLU's 99%. Therefore, we concluded that for the MNIST dataset, the ReLU function is the most efficient function by a narrow margin. Our results also showed that the MNIST dataset is a balanced dataset, as the precision, recall, and F1-score are all consistent with its accuracy. This further highlighted that the functions would not have the problem of too many false positive or false negative results (12).

One of the reasons ReLU was the most efficient is that it avoided the problem of vanishing gradients. This problem happens when the input is very large or small, and the gradients of the activation function become infinitely small, which causes the learning process of utilizing the derivative to decrease drastically (13). Therefore, non-saturated nonlinearities generally perform better than saturated ones (14). The saturated nonlinearities almost behave like a horizontal line with an extremely small gradient, slowing the gradient descent process. Sigmoid and Tanh functions are saturated nonlinearities, while ReLU is not. Therefore, ReLU avoids this problem as its derivative does not become small when the input is very large or small. Moreover, Sigmoid's derivative ranges from 0 to 0.25, and Tanh's derivative ranges from 0 to 1. After multiple layers in the network, the output might result in infinitely small gradients, also causing the convergence performance to decrease drastically.

Next, ReLU is computationally lighter than other exponential-related functions such as Sigmoid and Tanh. ReLU does not involve any exponential or logistic operations but simply takes the max of zero and input, as the function of ReLU is given by the formula f(x)=max(0,x) (4). In the MNIST dataset, ReLU took 1 to 2 fewer seconds to train per epoch compared to the exponential functions Sigmoid and Tanh. In short, ReLU is exponentially lighter than the other two functions because it doesn't involve computational-heavy exponential calculations, and it has bigger gradients when the input is large. Moreover, since ReLU's F1 score was consistent with its accuracy, we verified ReLU didn't have the problem of having false positive or false negative results (12). For the above reasons, we concluded that ReLU was the best activation function for the MNIST dataset.

However, in our model, ReLU was significantly better than Tanh. ReLU was only slightly better than Tanh in terms of accuracy by a margin of 0.07% on the mean of the three trials (**Figure 3**). On the other hand, Tanh and ReLU both performed significantly better in terms of accuracy on the MNIST dataset than Sigmoid.

The reason why ReLU only had a slim advantage in the MNIST dataset was likely because our CNN only used a small number of layers. Even though the ReLU function avoided vanishing gradient problems and was computationally lighter than the Tanh function, in our dataset with 28*28-pixel pictures, we only needed two hidden layers to compute our results. Therefore, the advantages of the ReLU function did not optimize the training processes as much as they would in networks with more than 100 layers. The problem of vanishing gradients was not as significant in our network with 2 convolution and pooling layers as it would be in a network

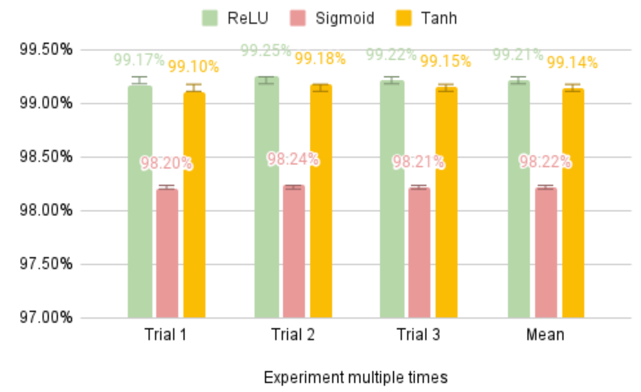

Figure 3: ReLU and Tanh and similarly high testing accuracies throughout the three trials and mean, while Sigmoid remained lower than the ReLU and Tanh. The graph shows the accuracies of the three activation functions with three trials. The rightmost column shows the mean of the three trials. ReLU and Tanh performed significantly better than Sigmoid (p<0.05), while ReLU doesn't perform significantly better than Tanh (p > 0.05).

with 100 convolution and pooling layers.

We also observed that Tanh had a better performance than the Sigmoid function. The Tanh function most likely performed better than the Sigmoid function because it had a larger range, from -1 to 1, compared to the Sigmoid ranging from 0 to 1. If the machine wanted to obtain negative outputs from the input, it would need extra transformations on the Sigmoid function to approximate a negative value, which required extra numbers of neurons and layers. On the other hand, the Tanh function could already create negative outputs, so the Tanh function had better accuracy and training time in our case. With a larger range but maintaining a similar s-shape, the Tanh function had steeper derivatives ranging from 0 to 1 compared to Sigmoid's derivatives ranging from 0 to 0.25, which made gradient descent more efficient when converging. Moreover, convergence during the backpropagation is usually more efficient if the average of the input variable is close to zero (15). Tanh's input took negative and positive values, so its average value was more likely to be closer to zero compared to Sigmoid's input, which only took positive values.

In conclusion, in the MNIST dataset using CNN, ReLU performed better than the other two functions due to its advantage in the steepness of the slope and computationally light operations. However, ReLU did not have a significant superiority in networks with only a small number of hidden layers. Additionally, the ReLU and Tanh functions performed better than the Sigmoid function as it had steeper derivatives and a wider range of output that optimized the gradient descent process.

Further research on a similar topic could be done on datasets that required more hidden layers in the neural network. For example, medical image classification is a topic that is more complex in nature, requiring more hidden layers in CNN. In a 2021 study, Helen and colleagues showed that to diagnose diabetes using CNN, more than 2 layers were needed for the network to attain better accuracy on diagnosing diabetes (16). In experiments with more hidden layers required, the results could differ from the outcome of this experiment as the network included more layers to process the data. ReLU's advantages in the gradient descent algorithm could have a bigger impact on accuracy and efficiency. However, since grayscale handwritten digits in the MNIST dataset and real-world images are very different in nature, the outcome of this experiment might not be generalizable to medical images or other complex datasets. Nevertheless, our research on small CNNs can serve as a benchmark of ReLU, Tanh, and Sigmoid's performance for further research. Specifically, it can benefit researchers that aim to attain high accuracy with limited hardware and time, such as for image detection in auto driving that requires high accuracy within a short amount of time.

MATERIALS AND METHODS

The MNIST dataset used for this experiment consists of 60,000 28 by 28 pixels of black and white images of handwritten digits (17). It was obtained from the TensorFlow dataset to train and test the model (17). The dataset was split into a training set and a testing set with a ratio of 8 to 2. The training set was used to train the network. The testing set was used to evaluate the result of each function based on its final accuracy and final loss function.

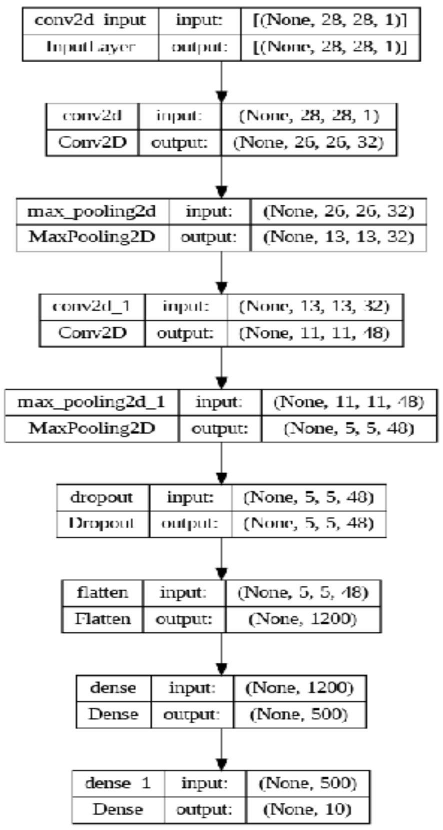
conv2d_input | input: | [(None, 28, 28, 1)]
InputLayer | output: | [(None, 28, 28, 1)]
conv2d | input: | (None, 28, 28, 1)
Conv2D | output: | (None, 26, 26, 32)
max_pooling2d | input: | (None, 26, 26, 32)
MaxPooling2D | output: | (None, 13, 13, 32)
conv2d_1 | input: | (None, 13, 13, 32)
Conv2D | output: | (None, 11, 11, 48)
max_pooling2d_1 | input: | (None, 11, 11, 48)
MaxPooling2D | output: | (None, 5, 5, 48)
dropout | input: | (None, 5, 5, 48)
Dropout | output: | (None, 5, 5, 48)
flatten | input: | (None, 5, 5, 48)
Flatten | output: | (None, 1200)
dense | input: | (None, 1200)
Dense | output: | (None, 500)
dense_1 | input: | (None, 500)
Dense | output: | (None, 10)

Figure 4: Flow chart of the CNN structure. The schematic shows the layers created in the CNN with input and output sizes. The Keras Tensorflow module was used to construct the flow chart. From top to bottom, the CNN will first receive input in the input layer. The information will be processed in the first Conv2D layer. Next, the Maxpooling2D layer will downsample the data. After downsampling, the information will be passed on to the second Conv2D and Maxpooling2D layer. The CNN then has a dropout layer to prevent overfitting. Lastly, the information is passed into the flattened layer and fully connected layer to eventually get the final classification results.

The TensorFlow package and Python codes were used to build a CNN and test the performance (18). We built a CNN with two hidden layers and downsampling layers using the maximum pooling (**Figure 4**). The final loss function of the network was the cross-entropy function. With the fully connected layer, we dropped neurons by the chance of 50% to address the problems of overfitting (6). The code used to build the neural network and load the dataset is provided in section one of the appendix.

The same neural network was fed with three different activation functions: ReLU, Sigmoid, and Tanh. We measured the performance of the activation functions in three aspects: training time, validation accuracy, and testing metrics (accuracy, precision, recall, and F1-score). Each activation function was evaluated using the following approach: the CNN was run three times, and the mean of the testing metrics of the three trials was taken. For the first trial, other than the accuracy, we also recorded the training time, training accuracy, and validation accuracy.

After compiling the results, we performed three trials to calculate the standard deviation of the accuracies and calculated the mean of the accuracies. The standard deviation was calculated with the Python NumPy module (19). We computed the one-way ANOVA test of the groups ReLU, Sigmoid, and Tanh. Next, we used Tukey's HSD test as the post-hoc test to perform pairwise comparisons. We used 0.05 as alpha for the ANOVA test and Tukey's test. Both tests were performed using the Python statsmodel module (20).

Received: March 18, 2022
Accepted: May 1, 2023
Published: September 15, 2023

REFERENCES

1. Jordan, Michael I., and Tom M. Mitchell. "Machine learning: Trends, perspectives, and prospects." *Science*, vol. 349, no. 6245, Jul. 2015, doi:10.1126/science.aaa8415.
2. Ker, Justin, et al. "Deep learning applications in medical image analysis." *IEEE Access*, vol. 6, 29 Dec. 2017, pp. 9375-9389. doi:10.1109/ACCESS.2017.2788044
3. LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*, vol. 86, no. 11, Nov. 1998, pp. 9375-9389. doi:10.1109/5.726791
4. Sharma, Siddartha, et al. "Activation functions in neural networks." *International Journal of Engineering Applied Sciences and Technology*, vol. 4, no. 12, 2020, pp. 310-316.
5. Gu, Jiuxiang, et al. "Recent Advances in Convolutional Neural Networks." *Pattern Recognition*, vol. 77, 2018, pp. 354-377. doi.org10.1016/j.patcog.2017.10.013.
6. Alzubaidi, Laith, et al. "Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions." *Journal of Big Data*, vol. 8, no. 1, 2021, doi:10.1186/s40537-021-00444-8.
7. Dryden, Nikoli, et al. "Channel and Filter Parallelism for Large-Scale CNN Training." *Proceedings of the International Conference for High Performance Computing*, Networking, Storage and Analysis, 2019, doi:10.1145/3295500.3356207.
8. S. Hershey et al., "CNN architectures for large-scale audio classification," *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp.131-135. doi: 10.1109/ICASSP.2017.7952132.
9. Li, Liu, et al. "Attention Based Glaucoma Detection: A Large-Scale Database and CNN Model." *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, doi:10.1109/cvpr.2019.01082.
10. Ma, Ningning, et al. "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design." *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 122-138. doi:10.1007/978-3-030-01264-9_8.
11. Cuesta-Albertos, J. A., and M. Febrero-Bande. "A Simple Multiway ANOVA for Functional Data." *TEST*, vol. 19, no. 3, 2010, pp. 537-557. doi:10.1007/s11749-010-0185-3.
12. Goutte, Cyril, and Eric Gaussier. "A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation." *Lecture Notes in Computer Science*, vol. 3408, 2005, pp. 345-359. doi:10.1007/978-3-540-31865-1_25.
13. Hochreiter, Sepp. "The Vanishing Gradient Problem during Learning Recurrent Neural Nets&nbsp; and Problem Solutions." *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, 1998, pp. 107-116. doi:10.1142/s0218488598000094.
14. Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, 2014, pp.1929-1958.
15. LeCun, Yann, et al. "Efficient Backprop." *Lecture Notes in Computer Science*, vol. 1524, 1998, pp. 9-50. doi:10.1007/3-540-49430-8_2.
16. Helen Josephine, V L, et al. "Impact of Hidden Dense Layers in Convolutional Neural Network to Enhance Performance of Classification Model." *IOP Conference Series: Materials Science and Engineering*, vol. 1131, no. 1, 2021, doi:10.1088/1757-899x/1131/1/012007.
17. LeCun, Yann, et al. 'MNIST Handwritten Digit Database.' *ATT Labs [Online]*. Yann.Lecun.Com/Exdb/Mnist, Accessed 21 Jan. 2023.
18. Abadi, Martín, et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems." *ArXiv*, 2016, arXiv:1603.04467.
19. Harris, Charles R., et al. 'Array Programming with NumPy.' *Nature*, vol. 585, no. 7825, Springer Science and Business Media LLC, Sept. 2020, pp. 357-362, doi.org10.1038/s41586-020-2649-2.
20. Seabold, Skipper, et al. "statsmodels: Econometric and statistical modeling with python." *9th Python in Science Conference*. 2010.

Appendix

The first section of the code uses tensorflow's keras module to build the CNN and run the network. The second section uses the results obtained from section 1 to calculate standard deviation and perform ANOVA test and Tukey's test with the statsmodel module.

```
#Section 1: CNN and testing
from numpy import unique, argmax
from tensorflow.keras.datasets.mnist import load_data
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPool2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout

from tensorflow.keras.utils import plot_model
from matplotlib import pyplot
import matplotlib.pyplot as plt
import numpy as np

# Loading MNIST Dataset
(x_train, y_train), (x_test, y_test) = load_data()

x_train = x_train.reshape((x_train.shape[0], x_train.shape[1],x_train.shape[2], 1))
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1], x_test.shape[2], 1))

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

#Building the neural network
n_filters = 32
filter_size = (3,3)
model = Sequential()
# The activation part is varied to test different activation functions
```

```
model.add(Conv2D(n_filters, filter_size, activation='sigmoid', input_shape=inp_shape))
model.add(MaxPool2D((2, 2)))
# The activation part is varied to test different activation functions
n_filters = 48
filter_size = (3,3)
model.add(Conv2D(n_filters, filter_size, activation='sigmoid'))
model.add(MaxPool2D((2, 2)))
model.add(Dropout(0.5))  #sometimes may want to drop out a percentage of the neurons

model.add(Flatten())
model.add(Dense(500, activation='sigmoid'))
model.add(Dense(10, activation='softmax'))

model.summary()
plot_model(model, 'model.png', show_shapes=True)

#define hyperparameters
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# fit model
history = model.fit(x_train, y_train, epochs=10, batch_size=128, verbose=2, validation_split=0.1)

#evaluate model
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f'Accuracy: {accuracy*100}')
print(f'Loss: {loss}')

from sklearn import classification_report
y_pred = model.predict(x_test, batch_size=64, verbose=1)
y_pred_bool = np.argmax(y_pred, axis=1)
print(classification_report(y_test, y_pred_bool))

#Section 2: Perform ANOVA test and Tukey's test

import statsmodels.api as sm
import numpy as np
from statsmodels.formula.api import ols
import pandas as pd
from statsmodels.stats.multicomp import pairwise_tukeyhsd

data = {
    "ActivationFunction": ["ReLU", "ReLU", "ReLU", "Sigmoid", "Sigmoid", "Sigmoid", "Tanh", "Ta
nh", "Tanh"],
    "Accuracy": [0.9917, 0.9922, 0.9925, 0.9620, 0.9624, 0.9621, 0.9910, 0.9918, 0.9916]
}

#Calculate standard deviation
df = pd.DataFrame(data)
relu_accuracies = df[df["ActivationFunction"] == "ReLU"]["Accuracy"]
sigmoid_accuracies = df[df["ActivationFunction"] == "Sigmoid"]["Accuracy"]
tanh_accuracies = df[df["ActivationFunction"] == "Tanh"]["Accuracy"]

tanh_std = np.std(tanh_accuracies)
sigmoid_std = np.std(sigmoid_accuracies)
relu_std = np.std(relu_accuracies)
print('Standard Deviation of Tanh', tanh_std)
print('Standard Deviation of Sigmoid', sigmoid_std)
print('Standard Deviation of ReLU', relu_std)
# ANOVA test
model = ols('Accuracy ~ C(ActivationFunction)', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

print(anova_table)

# Tukey's test
tukey = pairwise_tukeyhsd(endog=df['Accuracy'],     # Data
                          groups=df['ActivationFunction'],  # Groups
                          alpha=0.05,)          # Significance level
```

```
tukey.summary()
```