Table of Contents

Acknowledgments

First and foremost, I would like to thank my parents for teaching me how to code. I thank them for always being there for me, even when my code wasn't working, and also for teaching me resilience, grit, and the importance of never giving up.

Next, I would like to thank my sponsor and teacher, Mr. Albert, for guiding me through this process and always being there for me.

I would also like to thank Mr. Bakula for helping guide me through the paperwork and answering questions about the CPS STEM fair.

Additionally, I thank Mr. Mangione for helping me test my project on a different operating system to ensure its replicability.

I would like to thank GitHub for allowing me to track code changes, helping me organize my files, and hosting my repository for easy replicability. (See Appendix A)

I would also like to thank ChatGPT for helping me debug my code by guiding me to the solutions and encouraging me to actually understand why the bug happened and why the solution fixes it, instead of just giving me the answers. This really helped me understand why and how the bug happened and how the solution fixes it instead of just knowing that the solution works.

Finally, I would like to thank all my past, present, and future teachers, because without them, I wouldn't be who I am right now, or who I will be in the future.

Identification of the Independent and Dependent Variables

In this experiment, the independent variable is the size of the convolutional kernel. The experimental group for this experiment consists of a kernel size of 2x2 pixels (px), 4x4 px, and 5x5 px. The control for this experiment is a kernel size of 3x3 px. The dependent variable for this experiment is the model's accuracy in recognizing handwritten characters from the EMNIST Dataset.

Purpose

How does the convolutional kernel size affect handwritten character recognition accuracy?

Hypothesis

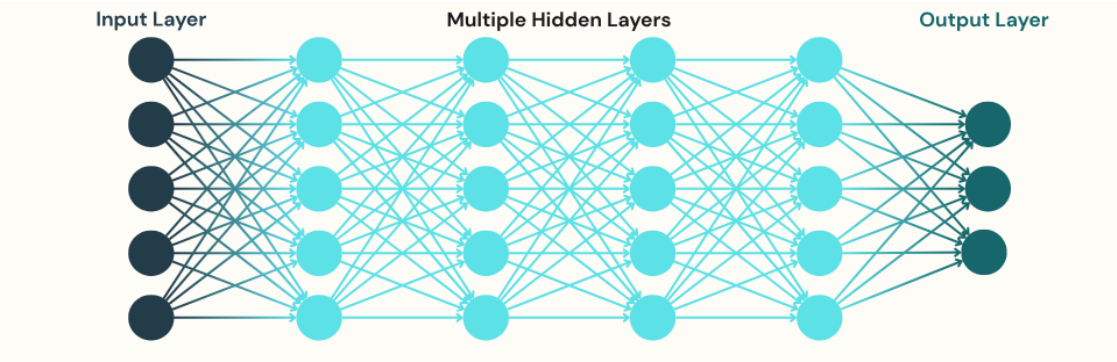
If the kernel size of a convolutional neural network is 4x4 px, then it will achieve the highest accuracy because smaller kernels capture very local features but may miss larger patterns, while larger kernels can blur finer details.

Background Research

For image recognition tasks using a convolutional neural network (CNN), different features, including kernel size, are inherent to how the CNN identifies, processes, recognizes, and predicts different handwritten characters. To understand this experiment, it is important to understand what a neural network is. According to Google Cloud (2024), "A neural network is a type of machine learning algorithm inspired by the human brain. It's a powerful tool that excels at solving complex problems more difficult for traditional computer algorithms to handle, such as image recognition and natural language processing."

A neural network is made up of artificial neurons. A single artificial neuron is a mathematical function that takes in information from previous artificial neurons, applies a calculation to the information, and then uses an activation function to see whether or not it should pass on its output to the next artificial neuron. See Fig. 1 for a visual example of a neural network with artificial neurons.

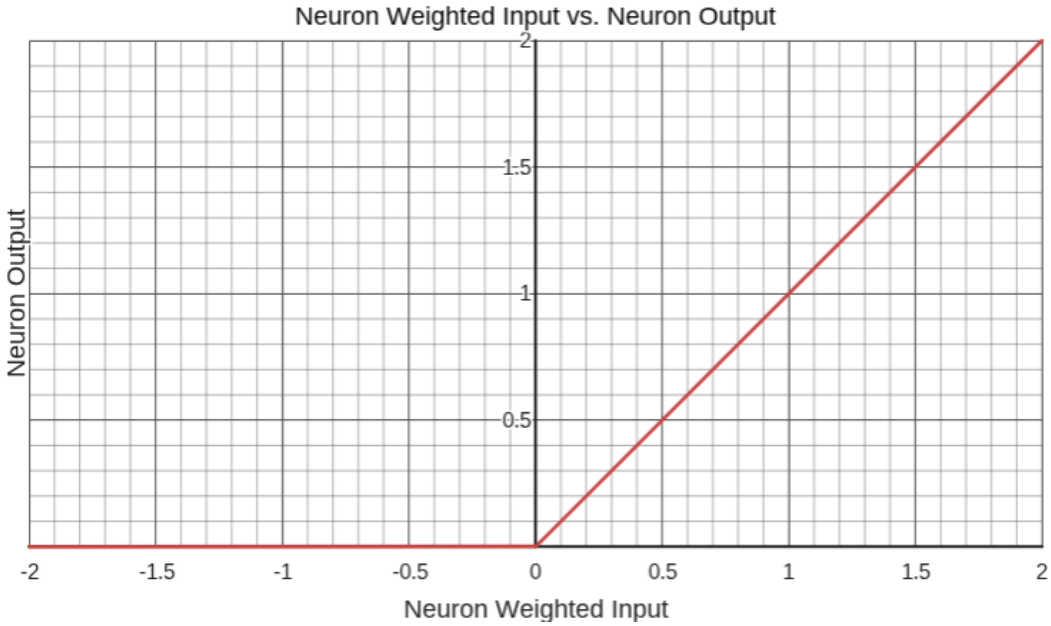
Fig. 1



A visual example of a neural network. Visualization created by the researcher with Canva.

An essential part of a neuron is the activation function. An activation function is a nonlinear function that is important to the neural network since it allows the neural network to identify more complex nonlinear shapes, such as curves or edges in handwritten characters (Ronaghan, 2018). This project uses 2 activation functions: the ReLU and SoftMax activation functions. The ReLU activation function is defined as a nonlinear function that passes the weighted input value of the neuron if it is above zero, and if it is zero, then it passes zero. See Fig. 2 for a graph of the ReLU activation function. The other activation function is the SoftMax activation function, which converts the raw scores from a neural network into a probability distribution, where each output value is between 0 and 1, and all values sum to 1. This is used for the final prediction by the neural network.

Fig. 2



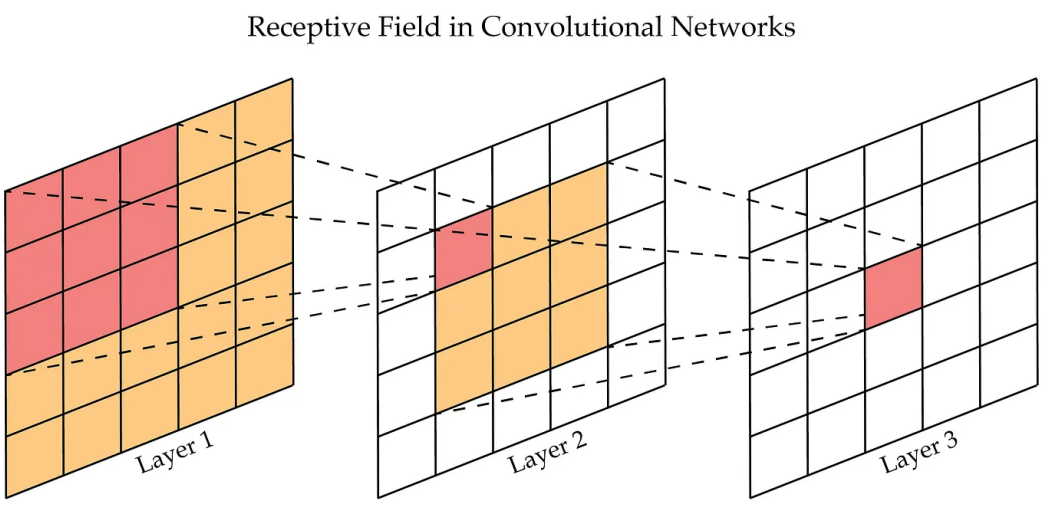
Neuron Weighted Input vs. Neuron Output. Visualization made by the researcher with Desmos.

The type of neural network used in this project is a convolutional neural network (CNN). According to IBM (2021a), “Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs.” CNNs work by using a convolutional kernel, which is a small matrix that slides over the inputs. This helps in highlighting specific features or patterns in the input, making it especially useful for image recognition (Jain, 2024). Because kernel size controls how much of the image is processed at once, it follows that kernel size is an important factor in determining the accuracy of a CNN.

This experiment also contains numerous examples of randomness within the code. This is an unavoidable part of training neural networks, as without randomization, the CNN doesn’t “learn”; it memorizes and inherently biases the CNN, making it unable to correctly predict characters with new data.

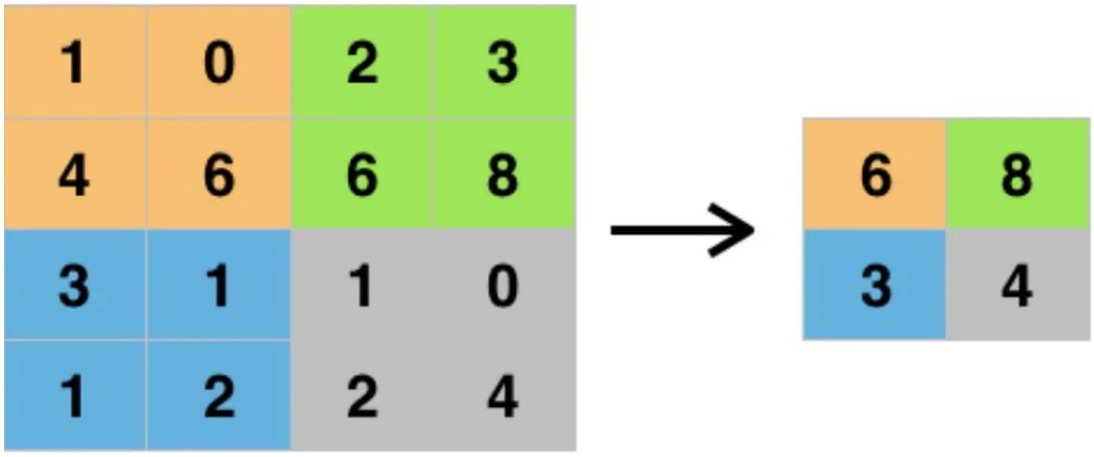
A CNN contains 2 parts: a convolutional feature extraction part and a fully connected neural network part. The convolutional feature extraction part of the CNNs used in this experiment contains 2 types of filters: A convolution filter and a max-pooling filter. The convolution is a matrix whose values get adjusted during training. The matrix slides across the image and, using matrix multiplication, passes on those values to the next filter. See Fig. 3 for an example of a convolution filter. MaxPooling is a filter that takes the largest number out of a matrix. This is useful because it helps resist small changes and because it helps take the most important features out of an image. See Fig. 4 for an example of a MaxPooling Filter.

Fig. 3



An example of a convolution filter. Source: Kalantar 2023

Fig. 4

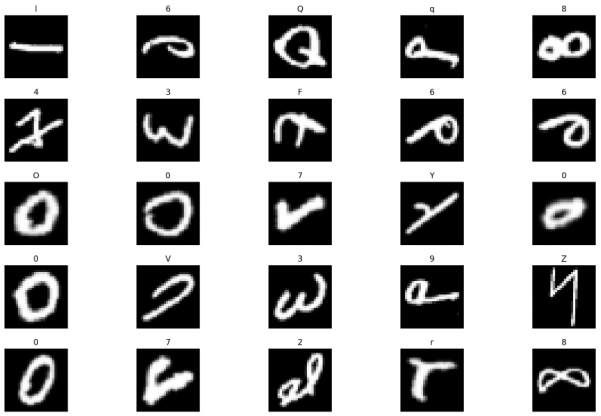


An example of a MaxPooling filter. Source: Kılıç 2023.

For training CNNs, this project uses the EMNIST dataset, which contains 28x28 pixel images of 62 different characters, 10 digits: 0–9, and 26 majuscules (uppercase letters): A–Z, 26 minuscules (lowercase letters): a–z. The EMNIST datasets' labels for each character need to be converted to ASCII, which can be converted to digits and letters for the label.

Some images in the EMNIST dataset are flipped and/or rotated. Sample images from the EMNIST dataset are shown in Fig. 5.

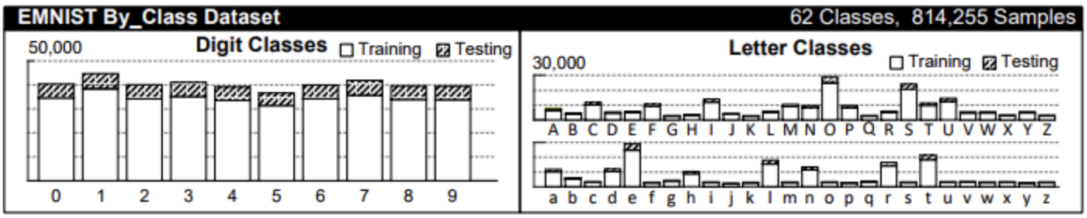
Fig. 5



25 random sample images of the EMNIST dataset. Visualization created by the researcher with Matplotlib.

The EMNIST dataset comes with different splits. This experiment uses the ByClass split of the EMNIST dataset. The ByClass split is modeled around the frequency distribution of characters found in real life, i.e, “e” is the most frequent letter in the English language; therefore, there will be more images of the letter “e”. It is necessary to oversample so that the model can see each character enough to meaningfully learn the different characteristics of that character. See Fig. 6 for a frequency distribution of the characters in the EMNIST dataset.

Fig. 6



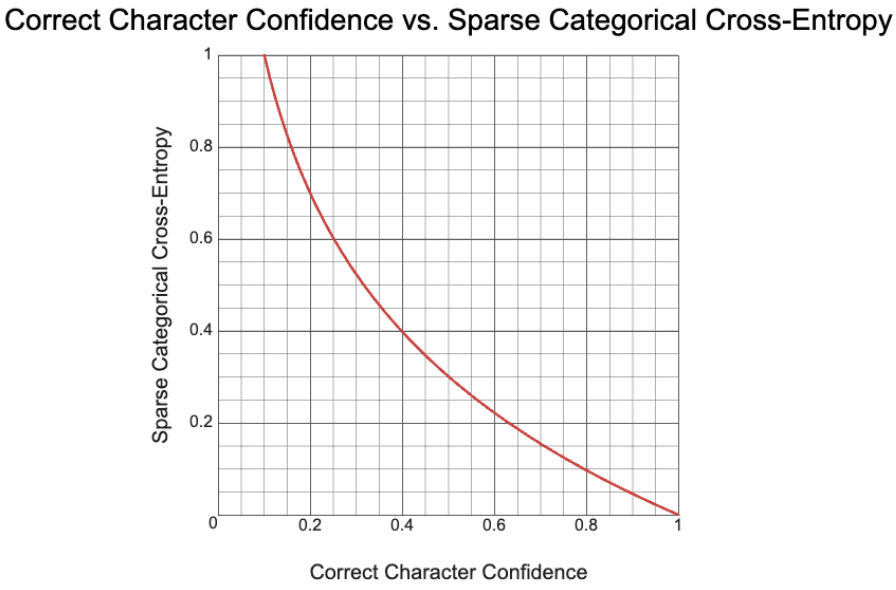
The frequency distribution of all the characters in the EMNIST ByClass split. Source: Crawford, 2017

The images in the dataset used by this project are 28 px by 28 px, so the range of kernel sizes needs to be small enough to avoid losing data. According to Sahoo (2018), “3x3 convolution filters work in general, and is often the popular choice!” Because it represents the standard, a kernel size of 3x3 px is the control in this experiment. Sahoo (2018) also mentions in his article that smaller kernel sizes are preferable because helpful features can be in more than one part of the image; therefore, it makes sense to have a smaller kernel size. Smaller kernels can miss some important features because the feature is bigger than the kernel, while larger kernels can miss some detail, which is why, in this experiment, the hypothesis predicts a kernel size of 4x4 to be the most accurate.

This experiment uses CNN’s that predict by giving a list of numbers correlating to their confidence (0–1) that a character is the correct character. i.e $[0.2,\ 0.3,\ 0.15,\ 0.4,\ 0.9..]$, where the index of the confidence number correlates to a unique character. In the example above, the model would’ve predicted the 5th element in the set for the image.

For the loss calculation for training CNN’s, this experiment uses sparse categorical cross-entropy (SCCE). SCCE is a formula that helps the CNN learn by giving it a low score if it predicts the right character confidently, whereas if it predicts incorrectly but confidently, it gives it a high score. This “punishes” and “rewards” the model for making correct predictions. The formula for SCCE is: $L_{oss}(y, \hat{y}) = -\sum_{i=1}^{k}(y_i \cdot log(\hat{y}_i))$ where y is the correct character in an array, i.e., $[0,\ 0,\ 0,\ 0,\ 1,\ 0...]$ where 1 represents the correct index of the character, and that gets compared to the model's output: $\hat{y}$. k Is the total number of characters, so 62 for this experiment. See Fig. 7 for a graph of CSSE.

Fig. 7



Correct Character Confidence vs. SCCE. Visualization made by the researcher with Desmos.

In training CNNs, this experiment introduces randomness. With CNNs, a model can just adapt to the training data and not be able to predict correctly with other data. Randomness prevents this overfitting by exposing the model to unpredictable inputs, encouraging the model to learn identifiable features of different characters to help it predict correctly instead of just memorizing training data.

This experiment uses Python, which, according to GeeksForGeeks (2021), "Python language is widely used in Machine Learning because it provides libraries like NumPy, Pandas, Scikit-learn, TensorFlow, and Keras. These libraries offer tools and functions essential for data manipulation, analysis, and building machine learning models." All of the reasons aforementioned are part of why this experiment uses Python.

Materials

Hardware

Quantity	Item	Specification
1	Computer	MacBook Air, 2022, M2, 16 GB RAM, >4.5 GB Memory Remaining

*Development Tools**

Item	Version
Github	https://github.com 4/11/26†
Visual Studio Code	1.106.2

*Programming Language**

Item	Version
Python	3.11.5

*Packages**

Item	Version
Matplotlib	3.10.7
NumPy	2.2.6
OpenCV Python	4.12.0.88
Rich	14.2.0
scikit-learn	1.7.2
TensorFlow	2.20.0
TensorFlow Datasets (TFDS)	4.9.9

**The item(s) in this list are digital resources; therefore, their numerical quantity is not applicable*
†This item does not have a specific version number, so a link to the website and the date of last use in this project was given

*Dataset**

Item	Version
EMNIST Dataset	https://arxiv.org/abs/1702.05373v1 11/27/25†

All custom code created for this project is available in the public repository and included in Appendix A. All custom code created for this project is licensed under the researcher's software license, which is available in Appendix B.

**The item(s) in this list are digital resources; therefore, their numerical quantity is not applicable*
†This item does not have a specific version number, so a link to the website and the date of last use in this project was given

Methods

System and Software Setup Procedure

For this experiment to be conducted, it is necessary to have a computer/laptop with at least 3.5 GB of storage available and 16 GB of RAM. It is highly recommended to download a code editor, such as Visual Studio Code, but it is not required. It is necessary to download and install Python, as well as the packages mentioned in the materials list.

Preparation of the Dataset Procedure

To start the experiment, it is necessary to prepare the EMNIST dataset for training purposes. Use TFDS to import the ByClass split of the EMNIST dataset. Using this, oversample each character to a random number based on the character type. For digits, majuscules, and minuscules, oversample to 20,000–25,000 characters, 15,000–20,000 characters, and 18,000–22,000 characters, respectively.

Construction of a CNN Procedure

To build a CNN, use the oversampled dataset as described above. Normalize the pixel values in the dataset for each image to scale between 0 and 1. Reshape each image into a 4-dimensional tensor that contains the number of images, height, width, and channels. For this experiment, there is only one channel due to the EMNIST dataset being grayscale. Define the CNN's structure by adding 3 convolution layers with the desired kernel size, with 2 maxpooling layers. The arrangement of the layers should be 1 convolution layer, 1 max pooling layer, 2 convolution layers, 1 max pooling layer. Additionally, add 1, 1-layer, 128-neuron neural network activation function as a layer to the CNN. Finally, add a 62 neuron output layer, using the softmax activation function, with each neuron corresponding

to one possible character in the EMNIST dataset. Each layer, except for the final output layer, which uses the SoftMax activation function, should use the ReLu activation function. Compile the model using the Adam optimizer with the loss calculation being SCCE and the metric being accuracy. Repeat this process 12 times: 3 for each kernel size.

Training of a CNN Procedure

To train a CNN, use the oversampled dataset. Using the train_test_split function of scikit-learn, split the oversampled dataset into a training category and a validation category and a validation category with a test size of 0.115, with a random positive integer for the random state. Then, augment the training split of the oversampled dataset with Gaussian noise with a noise factor of 0.05. Concatenate the original and noisy images, making sure to duplicate the labels for noisy images. Add an early stopping callback to prevent overfitting by monitoring the loss function, making sure to restore the best weights. Use this to train the model with 20 epochs and a batch size of 64. Save the model with a unique name and repeat this process 12 times for each model.

Testing of a CNN

To test a CNN, convert the testing section of the EMNIST ByClass split to a NumPy array. Normalize the pixel values of that array to a scale from 0 to 1. Reshape each image in that array into a 4-dimensional tensor: number of images, height, width, and 1 channel. Apply Gaussian noise to a subset of the testing images to help simulate real-world conditions. Compute the high-frequency content of images using a Laplacian filter and scale the Gaussian noise accordingly. Calculate accuracy as the percentage of images that the model correctly classified. Save the results. Repeat this for each model.

Analysis Procedure

To analyze a CNN, to support or reject the hypothesis of this experiment, plot the accuracy of all of the CNNs against each other, and compare the results.

Raw Data Table

Kernel Size (px)	Model Instance	Trial 1 Accuracy (%)*	Trial 2 Accuracy (%)*	Trial 3 Accuracy (%)*
2x2	1	80.162	80.174	80.382
2x2	2	80.525	80.539	80.435
2x2	3	80.473	80.541	80.490
3x3	1	81.655	81.623	81.671
3x3	2	81.460	81.477	81.484
3x3	3	81.515	81.501	81.536
4x4	1	81.896	81.863	81.803
4x4	2	80.500	80.508	80.519
4x4	3	82.199	82.138	82.133
5x5	1	81.900	81.881	81.948
5x5	2	81.600	81.643	81.684
5x5	3	82.494	82.465	82.480

Table 1. Raw Results

**All results have been rounded to three decimal places.*

Per-Model Mean Accuracy Data Table*

Kernel Size	Model Instance	Per-Model Mean Accuracy (%)†
2x2	1	80.240
2x2	2	80.499
2x2	3	80.501
3x3	1	81.649
3x3	2	81.474
3x3	3	81.517
4x4	1	81.854
4x4	2	80.509
4x4	3	82.156
5x5	1	81.910
5x5	2	81.642
5x5	3	82.480

Table 2. Per-Model Averaged Results

**Means were calculated with the raw, non-rounded values*
†All results have been rounded to three decimal places

Mean Accuracy Data Table*

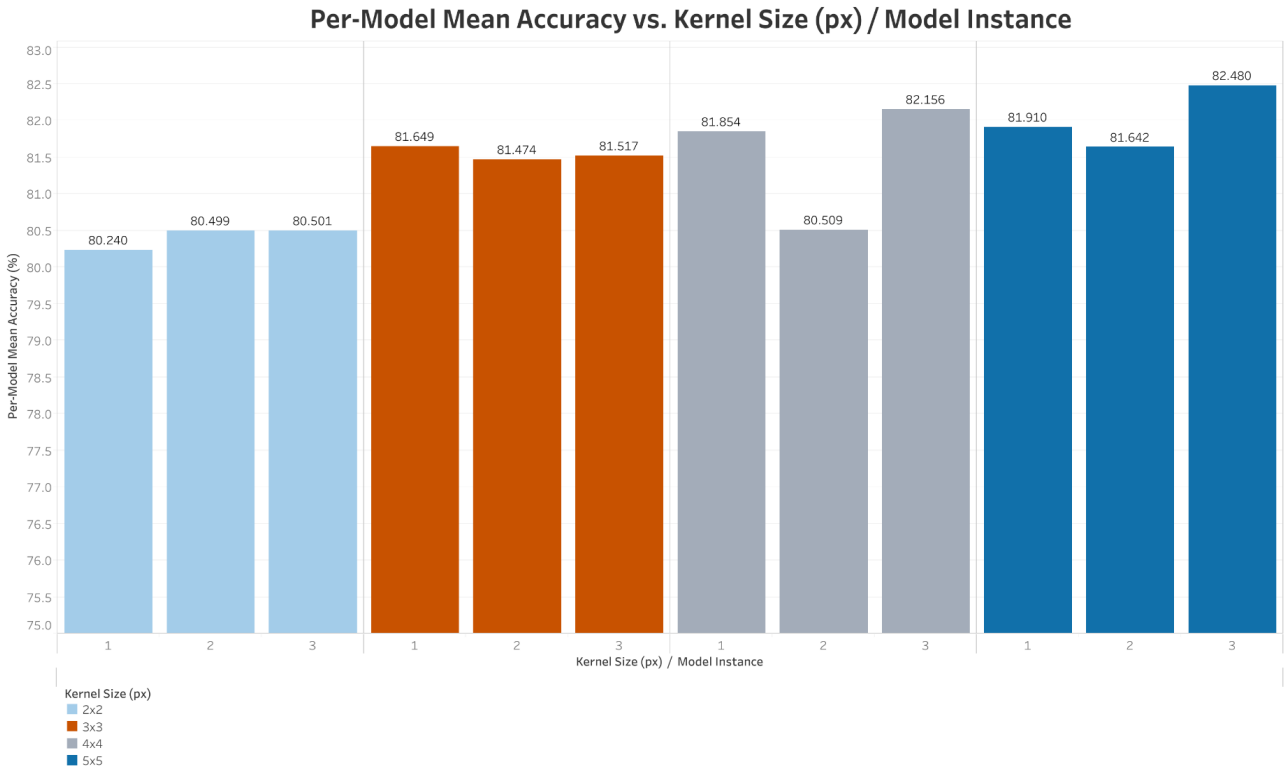
Kernel Size	Mean Accuracy (%)†
2x2	80.413
3x3	81.547
4x4	81.506
5x5	82.010

Table 3. Mean Results

*Means were calculated with the raw, non-rounded values
†All results have been rounded to three decimal places

Per Model Mean Data Graph

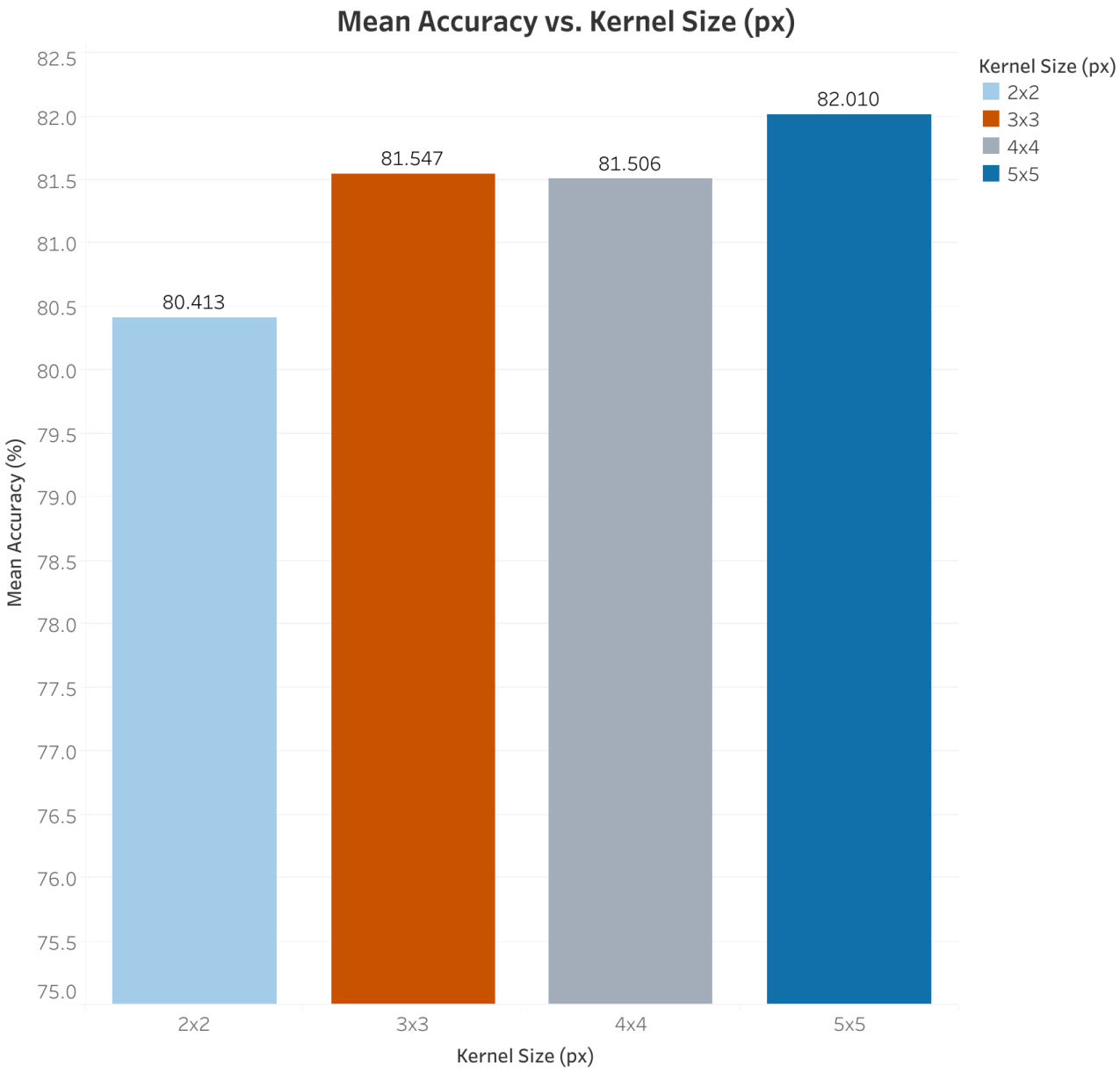
Fig. 8



Per-Model Mean Data Graph. Visualization created by the researcher with Tableau Public

Mean Data Graph

Fig. 9



Mean Data Graph. Visualization created by the researcher with Tableau Public

Observations

Some observations made during this experiment were that the first draft of the code required debugging due to slight mistakes made during the writing process. Another observation was that instead of using direct file paths, the code had to use OS file paths to ensure reproducibility. An additional observation was that the total training time for the 12 CNN models was approximately 30 hours. Additionally, the total folder size for the project took up around 3.5 GB of storage space.

Conclusions

Artificial Intelligence and neural networks are increasingly drawing attention from students, teachers, law enforcement, companies, the postal service, and consumers alike. This experiment made multiple convolutional neural networks that recognized handwritten characters from the EMNIST dataset. More importantly, this experiment's results can be applied to the real world for multiple purposes, like business records, handwritten mail addresses, and historical records, among others.

In this experiment, the independent variable was the size of the convolutional kernel. The experimental group for this experiment consisted of a kernel size of 2x2 px, 4x4 px, and 5x5 px. The control for this experiment was a kernel size of 3x3 px. The dependent variable for this experiment was the model's accuracy in recognizing handwritten characters from the EMNIST Dataset.

There are some limitations to this experiment, though. Experimental error could have affected the accuracy and reproducibility of this experiment. One possible experimental error is that, before training a neural network, including CNNs, the computer

generates random weights for all of the neurons so it can learn. Slight variances in the start could've slightly favored a certain model.

Another possible source of error is that the EMNIST dataset is relatively small, as some characters may be harder to distinguish, leading to small errors in classification that are inherent to the dataset.

Randomness inside of code may also pose experimental error, which is why this experiment trained 3 separate CNN's, tested them 3 different times, and then averaged their results for the conclusion to help normalize results per CNN, and ensure that repeated runs of the experiment's code will produce similar results.

The results of this experiment rejected the hypothesis: If the kernel size of a convolutional neural network is 4x4 px, then it will achieve the highest accuracy because smaller kernels capture very local features but may miss larger patterns, while larger kernels can blur finer details.

The best kernel size for handwritten recognition was a kernel size (px) of 5x5, followed by 3x3, 4x4, and 2x2, with a mean accuracy of 82.010%, 81.547%, 81.506%, and 80.413%, respectively.

One likely explanation for 5x5 px having the highest accuracy is that even-sized layers don't have a true center pixel, which can introduce misalignment across convolutional layers. Another likely explanation is that the 5x5 px kernel had the highest accuracy because it balanced local detail and contextual awareness for character recognition.

Next Steps

For me, this project mainly helped me learn how CNNs work and how to build one which was an amazing learning experience for me to understand what to do in the future. A future project that I will do based on the learning and experience that I got from this project is detecting cancerous tumors in mammograms or determining whether a tumor is malignant or benign.

References

Chollet, F. (2020, April 12). *The Sequential model*. Keras.io; Keras. https://keras.io/guides/sequential_model/

Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. *ArXiv:1702.05373 [Cs]*. https://arxiv.org/abs/1702.05373

Crawford, C. (2017). *EMNIST (Extended MNIST)*. Kaggle.com. https://www.kaggle.com/datasets/crawford/emnist?resource=download

GeeksforGeeks. (2021, December 29). *Machine Learning with Python Tutorial*. GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning/machine-learning-with-python/

GeeksforGeeks. (2024, June 26). *Kernels (Filters) in convolutional neural network*. GeeksforGeeks. https://www.geeksforgeeks.org/deep-learning/kernels-filters-in-convolutional-neural-network/

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. Deeplearningbook.org; MIT Press. https://www.deeplearningbook.org/

Google Cloud. (2024). *What is a Neural Network & How Does It Work?* Google Cloud; Google. https://cloud.google.com/discover/what-is-a-neural-network

IBM. (2021a, October 6). *Convolutional Neural Networks*. Ibm.com; IBM. https://www.ibm.com/think/topics/convolutional-neural-networks

IBM. (2021b, October 6). *What is a Neural Network?* IBM. https://www.ibm.com/think/topics/neural-networks

Jain, A. (2024, February 12). *All about convolutions, kernels, features in CNN - Abhishek Jain - Medium*. Medium; Medium.

https://medium.com/@abhishekjainindore24/all-about-convolutions-kernels-features-in-cnn-c656616390a1

Kalantar, R. (2023, January 2). *Receptive Field in Deep Convolutional Networks*. Medium; Medium. https://medium.com/%40rekalantar/receptive-fields-in-deep-convolutional-networks-43871d2ef2e9

Kılıç, İ. (2023, September 3). *What is Max Pooling and Why Do We Need Max Pooling?* Medium; Medium. https://medium.com/@ilyurek/what-is-max-pooling-and-why-do-we-need-max-pooling-57247a3fbca9

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1988). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Matplotlib. (2024). *Pyplot tutorial — Matplotlib 3.8.0 documentation*. Matplotlib.org. https://matplotlib.org/stable/tutorials/pyplot.html

NumPy Developers. (2025, November 16). *NumPy quickstart — NumPy v1.20 Manual*. Numpy.org. https://numpy.org/doc/stable/user/quickstart.html

Python Software Foundation. (2025). *What Is Python? Executive Summary*. Python. https://www.python.org/doc/essays/blurb/

PyTorch. (n.d.). *Neural Networks — PyTorch Tutorials 1.6.0 documentation*. Pytorch.org; The Linux Foundation. https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html

PyTorch. (2024). *EMNIST — Torchvision main documentation*. Pytorch.org; The Linux Foundation. https://docs.pytorch.org/vision/main/generated/torchvision.datasets.EMNIST.html

Ronaghan, S. (2018, July 26). *Deep Learning: Overview of Neurons and Activation Functions*. Medium; Medium. https://srnghn.medium.com/deep-learning-overview-of-neurons-and-activation-functions-1d98286cf1e4

Sahoo, S. (2018, August 19). *Deciding optimal kernel size for CNN*. Medium; TDS Archive. https://medium.com/data-science/deciding-optimal-filter-size-for-cnns-d6f7b56f9363

TensorFlow. (2019). *Convolutional Neural Network (CNN) | TensorFlow Core*. TensorFlow; Google. https://www.tensorflow.org/tutorials/images/cnn

Thor, W.-M. (2016). *Randomized Algorithms for Robustness*. Apxml.com; ApX Machine Learning. https://apxml.com/courses/data-structures-algorithms-ml/chapter-6-algorithmic-strategies-ml/randomized-algorithms-ml

Appendix A
Source Code Repository

This repository contains all of the project files, scripts, and programs for this project, except for the neural networks, due to their large size. The project files, scripts, and programs are available at: https://github.com/om-kanabar/ks-cnn.

Appendix B
README

The README that provides detailed documentation regarding the repository structure and setup instructions is available at:
https://github.com/om-kanabar/ks-cnn/blob/main/README.md

Appendix C
License

The following license applies exclusively to the custom code developed by the researcher for this project.
https://github.com/om-kanabar/ks-cnn/blob/main/LICENSE