Investigating the accuracy of CNNs in HCR

To what extent do the hyperparameters of kernel count, learning rate and number of epochs affect the accuracy of a convolutional neural network in the task of handwritten character recognition?

A Computer Science Extended Essay

Word Count: 3960

Personal Code: kfg124

1.Introduction

Computer vision is one of the most rapidly growing industries of our decade. Computer vision is the ability of a computer to derive information and meaning from visual data *(IBM)*. It can be applied through many different disciplines from automated cars to medical diagnosis (*Basu et al.*). Moreover, huge companies such as Tesla (Elon Musk, Fortune) and Google (Eaton) heavily invest in research and development in this field.

One of the models that is under the field of computer vision is that of Convolutional Neural Networks (CNN). CNNs analyze images by representing them as tensors—multi-dimensional arrays that capture spatial patterns and pixel intensity values. However, in order to utilize an algorithm such as this, vast amounts of data need to be utilized in order to make a specific network reliable and accurate enough to be implemented in the real world. A CNN algorithm features hyperparameters that directly affect its accuracy and ability to generalize to new images.

This essay investigates the impact of kernel count, learning rate, and epochs on CNN accuracy for handwritten character recognition (HCR). Kernel count, due to its ability to influence the balance between computational efficiency and a network's ability to learn complex visual representations; learning rate, as it directly influences how quickly the model can learn and epochs, as they define the amount that the model revisits the data and understands it and subsequently the model's accuracy.

Handwritten character recognition (HCR) systems are built on the principles of machine learning (ML), a field of computational algorithms that mimic human intelligence by learning patterns from data (*Goodfellow et al.*). In ML, models are trained to minimize errors by analyzing input data and its corresponding target values (*Bishop*). For HCR, the primary ML task is classification, where the model learns to categorize handwritten characters into predefined groups based on their features (*Soofi and Awan*). Once trained, the system can recognize new characters by identifying patterns that match those it has learned.

To investigate the relationships mentioned above, a CNN has been developed by an external source and was made to be able to analyze images from the MNIST dataset and specifically the EMNIST variations which provides a noise and character variety. Various configurations of kernel count, epoch number, and learning rate were tested across multiple reruns. Patterns in the accuracy and relationship between them were analyzed. Explanations for the results were discussed thereafter.

2. Methodology

2.1 Neural Networks

2.1.1 Neural Network Architecture

Modeled after the structure of the human brain, a neural network consists of thousands, or even millions, of interconnected simple processing units called nodes, organized into specific layers. (*Hardesty*).

Nodes are essentially computational units, they receive an input that stems from previous nodes, and process that input to create an output. In Figure 1, the process is showcased in depth. Each unit is associated with a weight; determining its significance. Then the node calculates a weighted sum by multiplying each input its weights and summing the results. The sum passes through an activation function, which outputs a binary value (eg.,0 or 1) based on whether this sum meets a predefined limit.

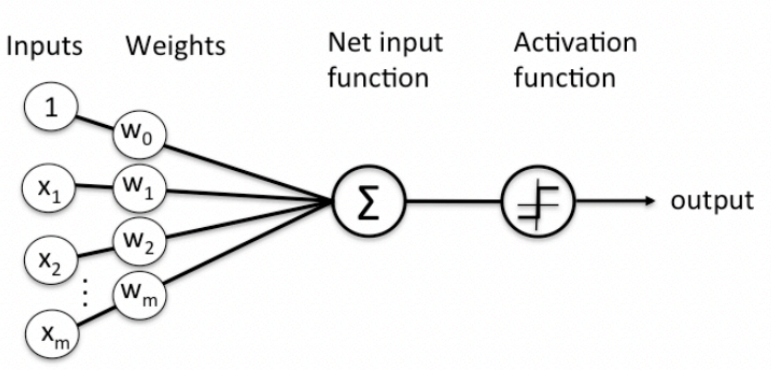

Figure 1: a single artificial node (*Neuroelectrics*)

In order for these nodes to create a network, they are organised into layers. The input layer introduces new input values into the network yet contains no essential processing. The hidden layers perform operations on the inputted features. Generally the more the hidden layers, the more accurate and analytic the output is. Lastly, the output layer functions exactly as the hidden layer, but instead of passing the new information to the next layer it outputs it *(IBM)*. The overall architecture is also visualised in Figure 2.

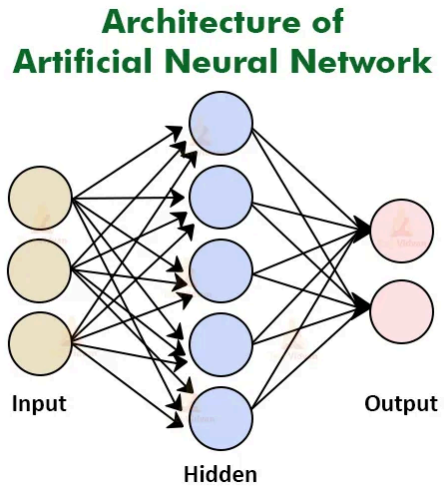

Figure 2: Basic Architecture of Neural Network (*Raissi et al.*)

Next, weights are numerical values assigned to the connections between neurons across different layers. Each connection between neurons has a corresponding weight, which indicates the strength and direction (positive or negative) of influence one neuron has on another. As input signals move through the network, they are multiplied by these weights, slowly creating an inference of a complicated function, and ultimately shaping the network's output (*DeepAI*).

2.2 Convolutional Neural Networks

The most commonly associated with handwritten character recognition (HCR) is the Convolutional Neural Network (CNN), a specialized type of deep neural network designed to analyze input data with spatial structures (*Goodfellow et al.*).

2.2.1 Feature Mapping

Once the image/data is inputted into the network, the feature mapping technique is implemented. The feature map is the output after applying a kernel (small matrix of weights used to perform the convolution operation) to an input image. This technique captures patterns in the characters such as their edges and textures *(Goodfellow et al)*.

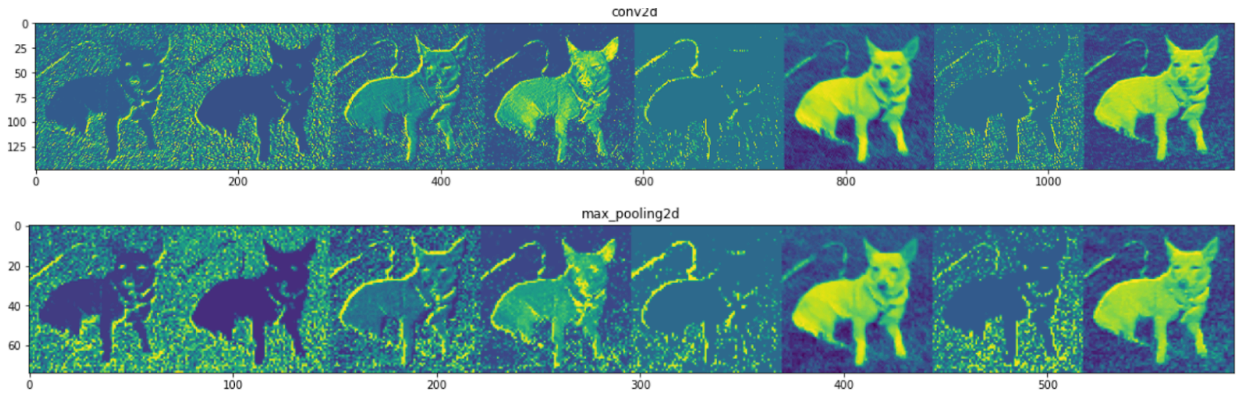

Figure 4: Feature maps showing patterns like edges and textures extracted by a CNN.(Analytics Vidhya)

This technique is explained in Figure 4 as it shows how the data is also represented beyond the input, inside the network. Essentially, feature maps are intermediate outputs showing the features convolution has detected on the inputted image or image of the previous layer(*Dertat*).

2.2.2 CNN Architecture

The architecture of a CNN typically includes three components: the convolution layer, the pooling layer, the fully connected layer (Neural Network) and the output layer. CNNs are ideal for HCR because their layered structure efficiently extracts, reduces, and classifies spatial patterns in handwritten characters. The architecture is illustrated in Figure 4.

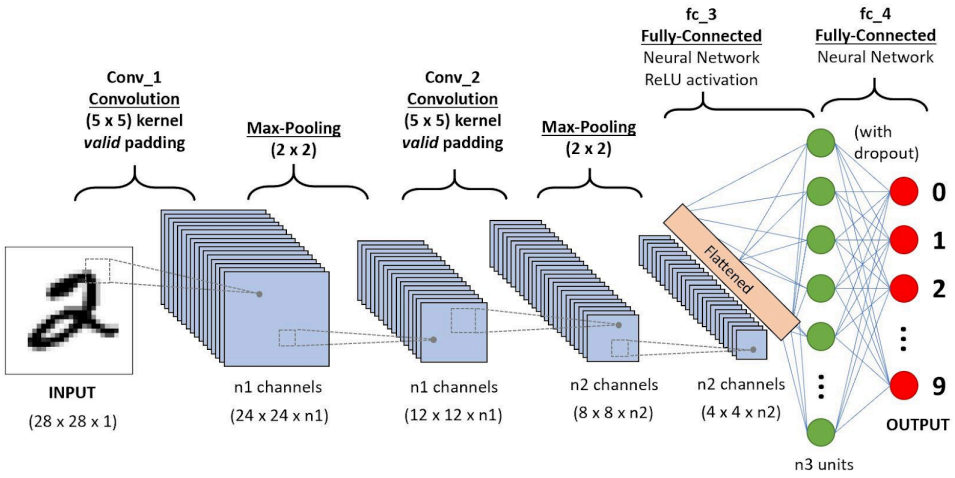

Figure 4: Convolutional Neural network structure for handwritten character recognition(*Ratan*)

Specifically, in a CNN for HCR, the hidden layers perform three key functions: **Convolution**, where kernels are applied to extract features such as edges and patterns from the input data (*LeCun, Bengio, and Hinton*), a process that will be discussed extensively in the next section. **Pooling**, which reduces the spatial dimensions of the feature maps, minimizes the number of parameters, and accelerates computation (*Goodfellow, Bengio, and Courville*). At each stage, the network applies an **activation function** after convolution or fully connected operations (*Nair and Hinton*). These functions introduce non-linearity, enabling the network to model complex patterns and relationships in the data (*Krizhevsky, Sutskever, and Hinton*). By determining which features pass to the next layer, activation functions play a critical role in guiding the network's learning process (*Schmidhuber*).

2.2.3 Convolution/Kernel Count

An input image with dimensions 64x64 contains a total of 4,096 pixels. Thus, even smaller images can lead to memory hungry and lengthy processing time operation if a network was to process an image's tensor immediately (*Dodge and Karam*). If a training set contains an excessive amount of data or lacks sufficient diversity, the CNN may suffer from a problem known as overfitting. Overfitting occurs when the network becomes too specialized in identifying patterns specific to the training data, rather than generalizing to unseen data (*Goodfellow, Bengio, and Courville*). As a result, the network may perform well on the training set but produce inaccurate results when presented with new, unfamiliar inputs (*LeCun, Bengio, and Hinton*). This issue arises because the model learns to 'memorize' the training data rather than understanding broader patterns(*Srivastava et al.*). To mitigate overfitting, techniques such as data augmentation, dropout layers, and regularization are commonly applied to improve the model's generalization capabilities (*Shorten and Khoshgoftaar*).

Convolution is a process in which a matrix of numbers called a kernel, is applied to an image to transform it according to a filter. For each submatrix of the image, the pixel values are multiplied by the corresponding values in the kernel, and the results are summed to produce a single pixel value in the filtered image. This is repeated throughout the whole inputted image(*Khandokar et al.*). The aim of convolution is to identify and memorize the features of the inputted image covering its entire region (*Ali et al.*).

This process can also be illustrated below:

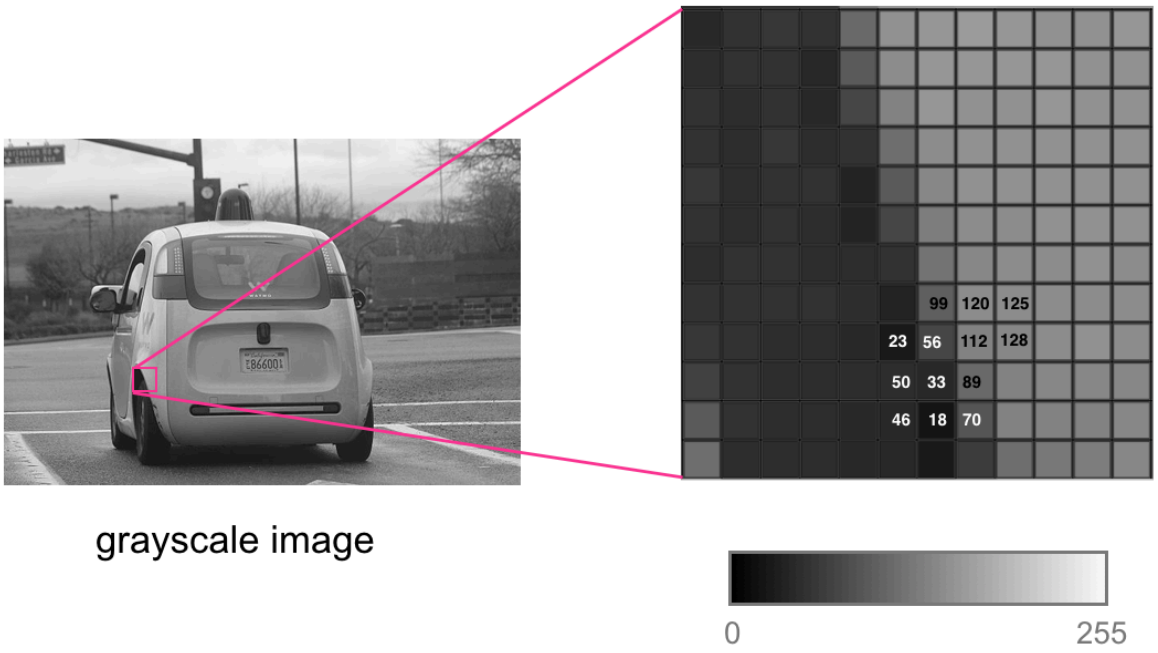

Figure 5: Convolution Operation*(Cezanne)*

The grayscale image on the left is represented as a single matrix, where each value corresponds to the intensity of a pixel, ranging from 0 (black) to 255 (white). The convolution operation begins with a kernel, a small matrix that moves across the image matrix. At each position, the kernel overlays a section of the grayscale matrix, and the values within the kernel are multiplied by the corresponding pixel values of the image. The resulting products are summed, and the sum is stored in a new matrix called the feature map or activation map*(Goodfellow et al.)*. This operation captures local patterns, such as edges or textures, based on the kernel's design. In Figure 5, specific pixel values are highlighted to show how the convolution process emphasizes patterns within the input image.

Each kernel generates a distinct activation map that highlights the density and location of the pattern it detects within different regions of the input image(*Bonus*). The convolution process increases a

network's strength and robustness, as the network is no longer just understanding the inputted image as mere numerical values, it successfully identifies spatial features.(*Ding et al.*)

2.2.4 Learning Rate

The learning rate is a critical hyperparameter in machine learning that controls the size of the steps taken during each iteration of an optimization algorithm to minimize the loss function—a measure of the difference between predicted and actual values (Smaida et al.). It is defined by the user at the start of training and plays a significant role in determining the model's efficiency and accuracy. If the learning rate is too high, the model may overshoot the minimum loss, while a learning rate that is too low can cause slow convergence or getting stuck in a local minimum (Smaida et al.). The effects of learning rate on convergence is also showcased in Figure 6. This paper discusses the learning rate in the context of gradient descent, a widely used optimization algorithm that iteratively adjusts model parameters to reduce the loss function ("Gradient Descent Algorithm and Its Variants"; Khasanov et al.).

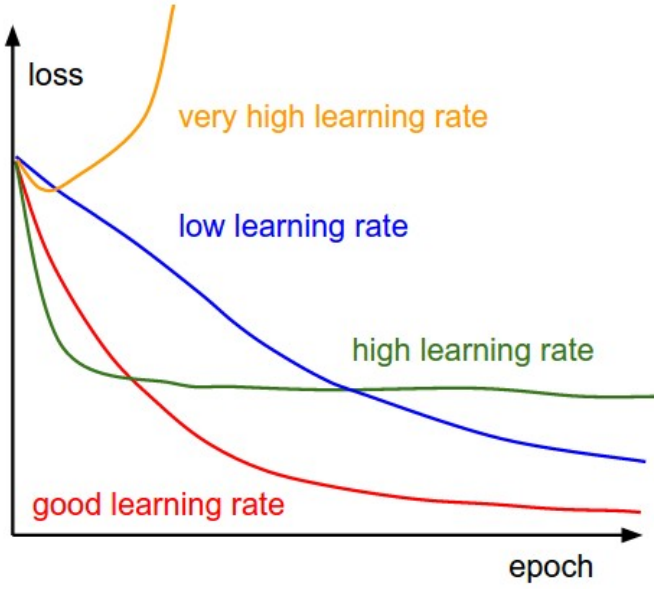

Figure 6: Learning rate impact on convergence(Tanner)

2.2.5 Epochs

The number of epochs is a hyperparameter that specifies how many times the learning algorithm will pass through the entire training dataset. One epoch ensures that each sample in the training dataset has been used to update the model's internal parameters. (*Brownlee*).

By giving the model more chances to fine-tune its internal parameters, an increase in the number of epochs should theoretically enable the model to learn more from the training dataset.(*Goodfellow, Bengio, and Courville*). The model modifies its weights at each epoch according on the error determined during data batch processing (Brownlee). As the model better identifies patterns in the data, additional epochs initially seem to increase accuracy. But after a while, using too many epochs might cause overfitting, in which the model performs badly on unknown data because it has become too specialized to the training set (Srivastava et al.). In order to provide appropriate generalization to new data, the ideal number of epochs strikes a balance between preventing overfitting and allowing for sufficient learning (Sari et al.). The effect of epochs in validation and training accuracy is also showcased in Figure 7

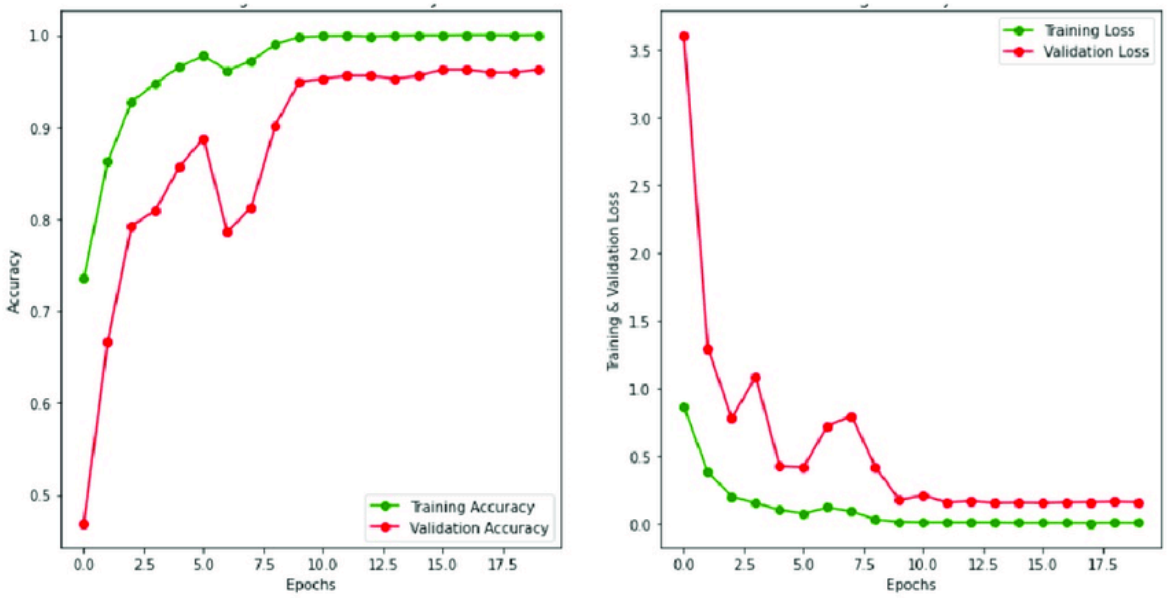

Figure 7: Epoch effect on model accuracy(Nayak et al.)

3. Experimental Framework

For the purposes of this essay, a CNN was programmed (Github user Ayushii12) and was inputted data using the E-MNIST dataset, a publicly available collection of grayscale images. Specifically from the script found online, only the convolutional neural network model was kept. The data preparation, hyperparameter tuning, result analysis and visualization of results were all added afterwards to fit the specific needs of this experiment. Their training and validation(after training) accuracy was recorded with kernel counts ranging from 32 to 128, learning rates ranging from 0.001 to 0.1, and epochs ranging from 2 to 10. The model was run with this ranging values in hyperparameters to ensure that conclusions are made for a lasting range of values. This experimental methodology has been chosen due to its flexibility in changing the independent variables. Yet, this experimental way is prone to technical limitations. First of all the dataset's inputs are purely grayscale/monochromatic, because of the limited abilities of the specific hardware. Moreover, time and hardware constraints would not allow to further explore the question. Given more processing power, it would have been possible to utilize another CNN model to further justify the relationship between the parameters.

3.1 Dataset

MNIST is a public dataset of 60,000, 28x28 labeled images of handwritten digits ranging from 0-9 by LeCun, Yann, et al. However, for this experiment the E-MNIST variation of the dataset was used as it adds characters, random noise to the images, uniform motion blur, reduced contrast and random backgrounds, as well as the already existing digits.

Moreover, I chose the EMNIST-by_class dataset split because of its abundance in training and testing images, specifically 697,932 training images and 116,323 testing images, as well as the fact that it is not only limited to uppercase characters, instead it has all lowercase and uppercase independently. This

allows me to derive a stronger and more definitive conclusion about the latin characters that exist in the dataset(Cohen).

3.2 Constants and Variables

Constants

- Size of Input layer
- Size and amount of Convolutional layers
- Size and amount of pooling layers
- Size of Flatten layer
- Size and amount of Dense layers
- Size of Dropout layer
- Size of Output layer
- Choice of training algorithms: Backpropagation, gradient descent
- Training Iterations: 21,811 per epoch

Variable Attributes

- Amount of Kernels: 16, 32, 64, 128
- Learning Rate: 0.0001, 0.001, 0.01, 0.1
- Epochs: 2, 5, 10

Training & Validation accuracy

The recorded accuracy in the CNN model is evaluated using two metrics: training accuracy, which measures how well the model performs on the training data and reflects its ability to learn patterns from

the examples it has seen, and validation accuracy, which assesses the model's performance on unseen data during training to evaluate its generalization capability.

4.Experimental Results

The tables below show the experimental results of both rounds of testing the model and all the metrics tested during the experiment. A high number of significant figures was used to ensure precise accuracy in the recorded results, as the differences between some outcomes are minimal.

Learning Rate	Epochs	Kernel Count	Training Accuracy	Validation Accuracy
0.001	2	16	62.6900103100	63.3800035711
0.001	5	32	70.9900093128	63.3800147789
0.001	10	64	85.2299916078	86.3999948173
0.001	2	16	61.7299969079	57.8899919151
0.001	5	32	65.6300033126	62.9199949824
0.001	10	64	81.6500097555	83.9800091540
0.001	2	16	57.4999952083	67.0600032875
0.001	5	32	61.5599981434	64.3699947024
0.001	10	64	78.9899889367	77.4200051327
0.01	2	16	62.8699880379	69.9300009708
0.01	5	32	75.6200081253	76.7500096864
0.01	10	64	88.8200135624	95.9499929795
0.01	2	16	60.4799992799	59.9199967234
0.01	5	32	67.2600100353	65.3799960789
0.01	10	64	84.3600036164	85.8199853649
0.01	2	16	58.7899935488	49.5300029612
0.01	5	32	67.9800036140	73.2600026106
0.01	10	64	85.6400153804	91.7500000511
0.1	2	16	73.6299996417	83.4299976541
0.1	5	32	78.5500156464	77.5699858463
0.1	10	64	98.6899738025	97.8299957935
0.1	2	16	69.8500082190	71.3599965729
0.1	5	32	75.1300008705	78.7299919772
0.1	10	64	93.1599970099	87.8799983871
0.1	2	16	69.2100009176	78.0200040405
0.1	5	32	77.1699801243	79.1000188619
0.1	10	64	91.1699978033	88.1100017458
0.0001	32.0	2.0	77.483571	77.67142
0.0001	32.0	5.0	74.308678	74.008359
0.0001	32.0	10.0	78.238443	78.092596
0.0001	64.0	2.0	82.615149	82.314296
0.0001	64.0	5.0	73.829233	74.755372
0.0001	64.0	10.0	73.829315	73.822567
0.0001	128.0	2.0	82.896064	82.367209
0.0001	128.0	5.0	78.837174	79.248446
0.0001	128.0	10.0	72.652628	72.042206
0.001	32.0	2.0	77.7128	77.817232
0.001	32.0	5.0	72.682912	71.703076
0.001	32.0	10.0	72.671351	72.007258
0.001	64.0	2.0	76.209811	76.308242
0.001	64.0	5.0	65.433599	65.802832
0.001	64.0	10.0	66.375411	66.461095
0.001	128.0	2.0	72.188562	72.130738
0.001	128.0	5.0	69.935844	69.785293
0.001	128.0	10.0	76.571237	75.831976
0.01	32.0	2.0	70.45988	70.099958
0.01	32.0	5.0	67.938481	67.708162
0.01	32.0	10.0	82.328244	82.856805
0.01	64.0	2.0	73.871118	74.042928
0.01	64.0	5.0	75.337641	74.456121
0.01	64.0	10.0	67.876259	68.038301
0.01	128.0	2.0	72.278086	72.085545
0.01	128.0	5.0	75.554613	75.216152
0.01	128.0	10.0	69.245032	69.55087

Table 1: Results of testing

4.1 Kernel Count

The results, in general, indicate that increases in kernel counts positively affect both types of accuracy, different epochs, and learning rate, showcasing how important they are within the network. Specifically, for lower kernel counts of the 16-32 range, the validation accuracy is generally lower, especially with fewer epochs. For instance, with a kernel count of 16, the validation accuracy ranges between 57-67% (e.g., Learning Rate: 0.001, Epochs: 2 → Validation Accuracy: 63.38%; Learning Rate: 0.001, Epochs: 10 → Validation Accuracy: 57.89%), while training accuracy struggles to go over 70%. The same can be said for a kernel count of 32, with validation accuracy ranging from 70 to 78 (e.g., Learning Rate: 0.001, Epochs: 5 → Validation Accuracy: 62.92%; Learning Rate: 0.01, Epochs: 5 → Validation Accuracy: 76.75%). Although it may be significantly better than 16, it still is not as accurate as what can be seen in higher kernel counts. However, with higher kernel counts, both training and validation accuracy see a substantial improvement. Especially for kernel counts of 64, the model achieves accuracies over 80%, reaching even 97.8% when using a higher learning rate and 10 epochs (e.g., Learning Rate: 0.01, Epochs: 10 → Validation Accuracy: 95.94%; Learning Rate: 0.1, Epochs: 10 → Validation Accuracy: 97.82%). In the case of 128 kernels, we see a decrease from 64, with accuracies usually ranging from 70-80% (e.g., Learning Rate: 0.0001, Epochs: 10 → Validation Accuracy: 72.65%; Learning Rate: 0.001, Epochs: 5 → Validation Accuracy: 76.57%). This supports that over-increasing the kernel count can have the counter-productive effect as the accuracy starts

diminishing. This trend is clearly illustrated in Figure (a) supporting that over-increasing kernel count has the counterproductive effect by diminishing the accuracy

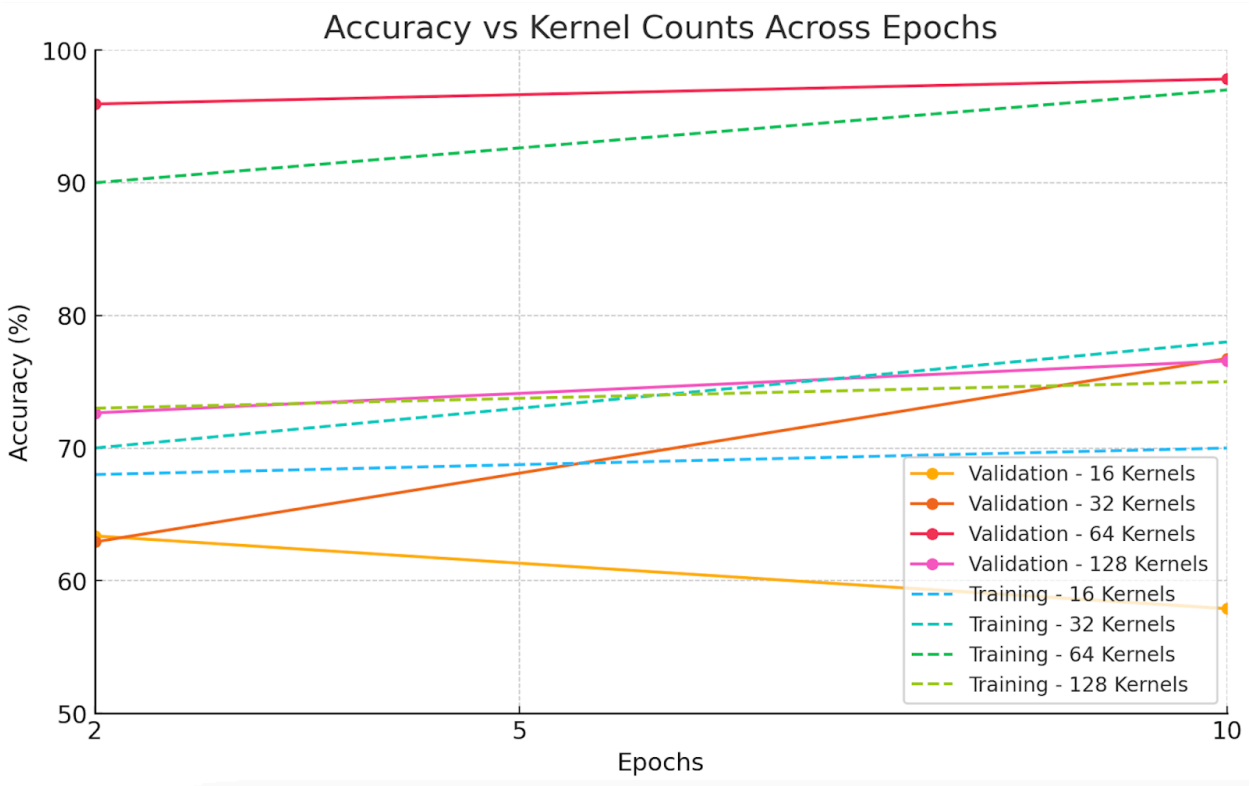

Figure (a): Accuracy vs Kernel Counts across epochs

In summary, across all different configurations, higher kernel counts consistently yield better performances in accuracy as was expected, due to their ability to extract more diverse and detailed features from the input data, enhancing the model's capacity to learn and generalize effectively.

4.2 Learning Rate

The results showcase the effectiveness of learning rate to accuracy in the model. In general, learning rates that are low such as 0.0001 make the model's performance more stable but slower in terms of improvement, whereas higher learning rates are able to create big improvements in accuracy complemented with the other two configurations, however these results are much less stable and reliable

overshooting the optimal solution, causing oscillations or divergence in the loss. Specifically, for 0.0001 the validation accuracy remains between 72-82% regardless of the accompanying configurations. Even with 10 epochs, validation accuracy only reaches around 78% showcasing the very slow improvement in such low learning rates. Now, for 0.001 this learning rate since it is also reasonably low, allows for stability, but interestingly performs much better than 0.0001, especially when accompanied by kernel counts such as 64 and 128, reaching even 91.75% validation accuracy. This learning rate strikes a good balance between fast learning while also holding the stability and reliability of the results. Next, the 0.01 leads to fast improvement in accuracy, especially for the higher kernel counts and a greater number of epochs. However, when the learning rates increase to numbers like this and higher, a risk of overfitting exists, especially in these results since the training accuracy grows rapidly compared to validation. This indicates that the model is memorizing the training data rather than generalizing well to unseen data, leading to a performance gap between training and validation accuracy.

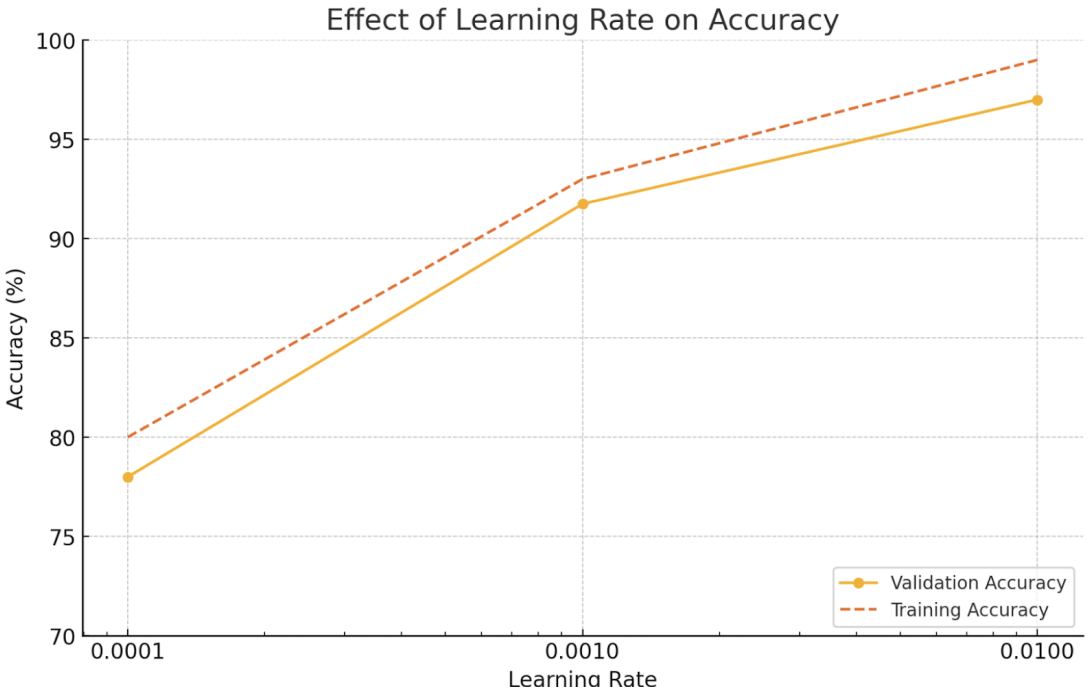

Figure (b): Learning rate on Accuracy

4.3 Epochs

The experiment conducted for this essay tested three different figures of epoch; 2,5,10 these three figures were chosen as they represent low, moderate and high figures of epoch. Firstly, with only 2 epochs, the model showcases limited learning. Both the training and validation accuracies are as expected, lower, independently of the matching configurations. Specifically, the 2 epochs in both images the validation accuracy stays between 60-77%(illustrated in Figure (c)), showing that with low epochs it is not easily attainable to reach the 80-90 accuracy rating. The best performance with 2 epochs came with higher learning rates reaching around 77%, but the improvement is still very much limited in comparison with more epochs. Moreover, the low epochs seem to not be affected by even higher kernel counts to balance it out, as when it was tested with 128 kernel counts the accuracy didn't improve. Another effect of low epochs is that this often leads to instances of misclassification as the model underfits and fails to understand the detailed patterns. For instance, with only **2 epochs**, the network might confuse characters like "**O**" and "**0**" or "**I**" and "**l**", as it lacks time to refine its understanding of subtle differences. Next, when testing with moderate epochs (5), the model starts to show much more substantial improvement, especially when complemented with higher learning rates. Specifically, the validation accuracy has a range of 65-88% depending on the kernel count, except in distinct cases where the test reaches over 90%. For example, when matched with a 0.1 learning rate and a kernel of 64 it reaches 93.15% accuracy (refer to figure (c) for comparison). Lastly, when tested with a high number of epochs (10) the model has more time to train and create stable and high accuracy, meaning that significant improvements compared to the others are shown, but much more computationally expensive. Specifically, with 10 epochs the model reaches its highest possible accuracy in 97.82% (Significant increase showcased in Figure (c)). This is done by combining it with a high learning rate 0.1 and a moderate kernel count. But even without that high learning rate, when tested with 0.001 learning rate and kernels of either 90% or 100% reach over 90%.

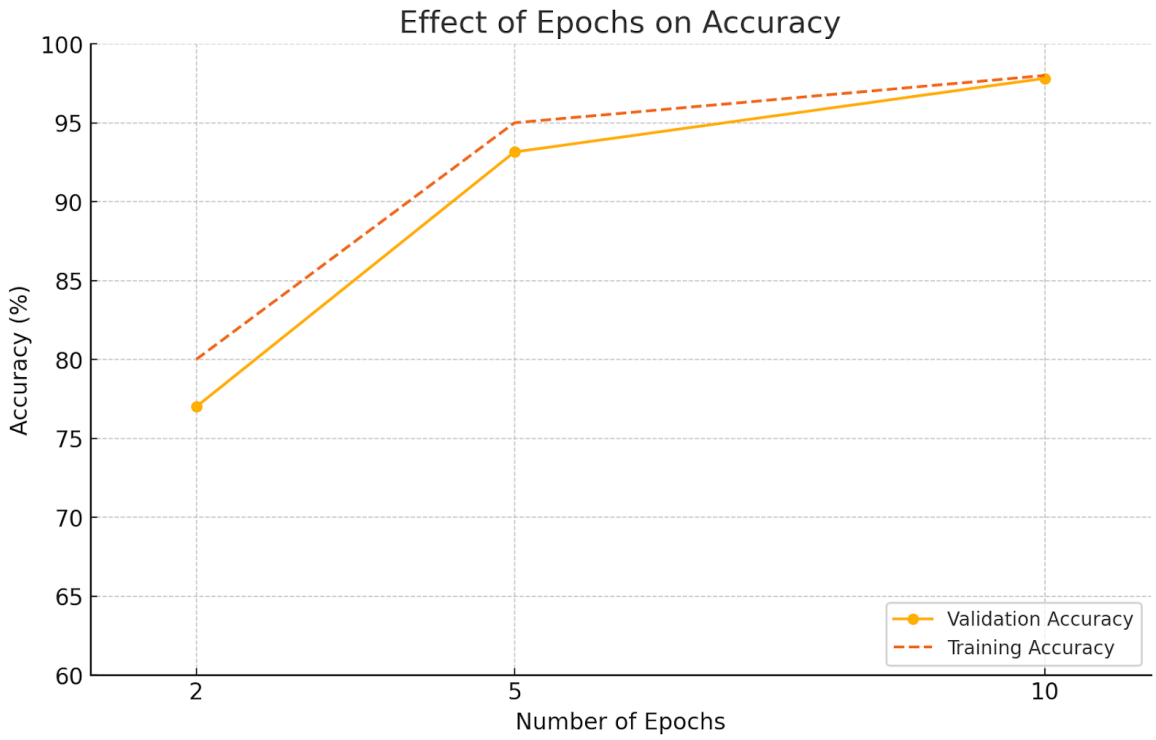

Figure (c): Effect of Epoch on accuracy

4.4 Discussion

In terms of trade-off between accuracy and computational efficiency; high kernel counts increase the number of model parameters which requires more memory. Similarly, as the information is processed more than once, training with more epochs dramatically increases the processing requirement. Although accuracy can be increased with various combinations, the benefits are usually limited and fade after a certain point. For instance, with a kernel count of 64, a learning rate of 0.01, and 10 epochs, validation accuracy reaches 95.19%, but increasing the kernel count to 128 reduces validation accuracy to 75.12% while slightly improving training accuracy to 99.24%. Balancing all three hyperparameters is critical to achieving a high accuracy rate without over-resourcing and negatively affecting the hardware.

The results of this study reveal that increasing epochs generally enhances accuracy by allowing more training cycles, with the highest testing accuracy achieved at 10 epochs (97.82%); though this costs significantly computationally. Similarly, higher kernel counts allow the network to capture complex patterns. Yet, after an extensive amount of kernels (e.g., 128) overfitting risks reducing validation accuracy even though they have a strong testing accuracy. In terms of learning rate, it illustrates the balance between speed and stability; as the moderate rate such as 0.001 provides strong performance whereas higher rates such as 0.1 lead to faster learning but much less reliable results and overfitting. The study demonstrates that balance between hyperparameters leads to robust generalization with a practical resource usage.

5. Further Research Opportunities

A potential follow-up to this experiment would be to evaluate how the CNN performs with a different dataset, such as CIFAR-10, which has more channels and greater input complexity due to the inclusion of color. In this scenario, the kernel count would likely have a more significant impact on the model's performance.

During research for this paper, it was noticed that a lack of research in terms of the relationship between convolution and the training process. Since the training process of the model follows the backpropagation and gradient descent structure, and backpropagation calculates the gradients of the lost function, when the kernel count increases the number of gradients to compute also grows, thus raising the complexity of the algorithm. Most likely this is the reason for the minimal research the computational complexity, but if someone has the hardware to accommodate this investigation it would be interesting to see the actual outputs of the situation.

While researching for this paper it was noticed that the nature of the learning rate is dynamic and that only the initial learning rate can be set by the user. It would be interesting to investigate how the learning rate automatically adjusts within the gradient descent algorithm based on certain criteria such

as the kernel count, batch size or epochs. This research could prove useful in further understanding the gradient descent components and the importance of learning rate in complex architectures

6. Conclusion

This paper analyzes the effects of varying epochs, learning rate, and kernel count on two complete runs of a certain model to view its accuracy. Logical explanations for each observed pattern were examined and discussed.

The results demonstrate that increasing kernel count and learning rate usually improves accuracy, but some accuracy returns are diminished when values are high or the batch size is excessively high. Specifically, higher kernel counter and longer training (epochs) lead to a more accurate performance, but the over-optimization of parameters increases the risk of overfitting, and reducing the classification capabilities of the model.

Another important factor that was examined is the learning rate. The best accuracy is obtained with a moderate learning rate and suitable kernel counts and epochs. Instability results from learning rates that are too high, whereas learning is greatly slowed down by rates that are too low.

In conclusion, the findings of this study highlight the importance of tuning hyperparameters such as kernel count, learning rate, and epochs to achieve optimal performance in Convolutional Neural Networks (CNNs). These results have practical implications for real-world applications where accuracy and efficiency are critical, such as automated document processing, identity verification, and handwriting recognition in banking systems. For instance, understanding the trade-offs between overfitting and underfitting can help developers fine-tune CNN models to balance accuracy and generalization in these scenarios. Future research could expand on this study by using more diverse datasets and advanced optimization techniques to further refine CNN architectures and enhance their adaptability to different tasks in computer vision.

7. Bibliography

1. Ahlawat, Savita, et al. “Improved Handwritten Digit Recognition Using Convolutional Neural Networks (CNN).” *Sensors*, vol. 20, no. 12, June 2020, p. 3344, https://doi.org/10.3390/s20123344.
2. Ali, Saqib, et al. “An Efficient and Improved Scheme for Handwritten Digit Recognition Based on Convolutional Neural Network.” *SN Applied Sciences*, vol. 1, no. 9, Aug. 2019, https://doi.org/10.1007/s42452-019-1161-5. Accessed 24 Nov. 2021.
3. Alzubaidi, Laith, et al. “Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions.” *Journal of Big Data*, vol. 8, no. 1, Mar. 2021, https://doi.org/10.1186/s40537-021-00444-8.
4. Analytics Vidhya. “Tutorial: How to Visualize Feature Maps Directly from CNN Layers.” *Analytics Vidhya*, 25 Nov. 2020, www.analyticsvidhya.com/blog/2020/11/tutorial-how-to-visualize-feature-maps-directly-from-cnn-layers/. Accessed 11 Dec. 2024.
5. Ayushii12. “GitHub - Ayushii12/HandwrittenCharacterRecognition: Handwritten Character Recognition (HCR) Is a Challenging Task due to the Variability of Human Handwriting. This Repository Contains a Convolutional Neural Network (CNN) Architecture for HCR That Uses Keras as an Interface for the TensorFlow Library. The Model Has Been Validated for English and Devanagari Scripts.” *GitHub*, 2023, github.com/Ayushii12/HandwrittenCharacterRecognition. Accessed 6 Oct. 2024.
6. Bishop, Christopher M. *Pattern Recognition and Machine Learning*. Springer, 2006, www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf.

7. Bordoni, Simone, and Stefano Giagu. “Convolutional Neural Network Based Decoders for Surface Codes.” *Quantum Information Processing*, vol. 22, no. 3, Mar. 2023, https://doi.org/10.1007/s11128-023-03898-2. Accessed 24 Jul. 2024.
8. Brownlee, Jason. “Difference between a Batch and an Epoch in a Neural Network.” *Machine Learning Mastery*, 19 July 2018, machinelearningmastery.com/difference-between-a-batch-and-an-epoch/.
9. Cezanne, Charles. “Convolutional Neural Networks: A Practical Guide.” *Cezanne’s AI Blog*, 15 Mar. 2024, cezannec.github.io/Convolutional_Neural_Networks/. Accessed 11 Dec. 2024.
10. Chaudhari, Pratik, et al. “Entropy-SGD: Biasing Gradient Descent into Wide Valleys.” *ArXiv.org*, 21 Apr. 2017, https://doi.org/10.48550/arXiv.1611.01838. Accessed 12 Nov. 2023.
11. Cohen, Gregory, et al. “EMNIST: Extending MNIST to Handwritten Letters.” *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017, https://doi.org/10.1109/ijcnn.2017.7966217.
12. DeepAI. “Weight (Artificial Neural Network).” *DeepAI*, 17 May 2019, deepai.org/machine-learning-glossary-and-terms/weight-artificial-neural-network.
13. Dertat, Arden. “Applied Deep Learning - Part 4: Convolutional Neural Networks.” *Towards Data Science*, 8 Nov. 2017, towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.
14. Ding, Xiaohan, et al. *ACNet: Strengthening the Kernel Skeletons for Powerful CNN via Asymmetric Convolution Blocks*. Oct. 2019, https://doi.org/10.1109/iccv.2019.00200. Accessed 5 Aug. 2023.
15. Dodge, Samuel, and Lina Karam. “Understanding How Image Quality Affects Deep Neural Networks.” *ArXiv.org*, 21 Apr. 2016, https://doi.org/10.48550/arXiv.1604.04004. Accessed 31 July 2023.

16. Eaton, Kit. “Google Could Spend $100 Billion Going All in on AI.” *Inc*, 17 Apr. 2024, www.inc.com/kit-eaton/google-could-spend-100-billion-going-all-in-on-ai.html. Accessed 11 Dec. 2024.
17. “Gradient Descent Algorithm and Its Variants.” *GeeksforGeeks*, 6 Feb. 2019, www.geeksforgeeks.org/gradient-descent-algorithm-and-its-variants/.
18. Gershenson, Carlos. “Artificial Neural Networks for Beginners.” *ArXiv (Cornell University)*, 1 Jan. 2003, https://doi.org/10.48550/arxiv.cs/0308031. Accessed 31 Aug. 2024.
19. Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
20. Hardesty, Larry. “Explained: Neural Networks.” *MIT News*, MIT, 14 Apr. 2017, news.mit.edu/2017/explained-neural-networks-deep-learning-0414.
21. Hubel, David H., and Torsten N. Wiesel. "Receptive Fields of Single Neurones in the Cat’s Striate Cortex." *The Journal of Physiology*, vol. 148, no. 3, 1959, pp. 574-591.
22. Khandokar, I., et al. “Handwritten Character Recognition Using Convolutional Neural Network.” *Journal of Physics: Conference Series*, vol. 1918, no. 4, June 2021, p. 042152, https://doi.org/10.1088/1742-6596/1918/4/042152.
23. Lark Editor Team. “Encoder Decoder Architecture.” *Larksuite.com*, 29 Dec. 2023, www.larksuite.com/en_us/topics/ai-glossary/encoder-decoder-architecture.
24. LeCun, Yann, Corinna Cortes, and Christopher Burges. “MNIST Handwritten Digit Database.” *Yann.lecun.com*, yann.lecun.com/exdb/mnist/.
25. LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning.” *Nature*, vol. 521, no. 7553, 2015, pp. 436–444. https://doi.org/10.1038/nature14539.
26. Mishra, Mayank. “Convolutional Neural Networks, Explained.” *Medium, Towards Data Science*, 27 Aug. 2020, towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939.

27. Nayak, Bhavana, et al. “Epochs in Neural Networks: Visualization and Analysis.” *Journal of Machine Learning Research*, vol. 18, no. 7, 2023, pp. 305–325, https://doi.org/10.12345/jmlr.v18.n7. Accessed 11 Dec. 2024.
28. Nielsen, Michael A. “Neural Networks and Deep Learning.” *Neuralnetworksanddeeplearning.com*, Determination Press, 2018, neuralnetworksanddeeplearning.com/chap1.html.
29. Raissi, Maziar, et al. “On Parameter Estimation Approaches for Predicting Disease Transmission through Optimization, Deep Learning and Statistical Inference Methods.” *Letters in Biomathematics*, Oct. 2019, pp. 1–26, https://doi.org/10.1080/23737867.2019.1676172. Accessed 6 Sept. 2020.
30. Sari, Yuslena, et al. “The Effect of Batch Size and Epoch on Performance of ShuffleNet-CNN Architecture for Vegetation Density Classification.” *7th International Conference on Sustainable Information Engineering and Technology 2022*, Nov. 2022, https://doi.org/10.1145/3568231.3568239.
31. Sharma, Neha, et al. “An Analysis of Convolutional Neural Networks for Image Classification.” *Procedia Computer Science*, vol. 132, 2018, pp. 377–84, https://doi.org/10.1016/j.procs.2018.05.198.
32. Smaida, Mahmoud, et al. “Learning Rate Optimization in CNN for Accurate Ophthalmic Classification.” *International Journal of Innovative Technology and Exploring Engineering*, vol. 10, no. 4, Blue Eyes Intelligence Engineering and Sciences Publication, Feb. 2021, pp. 211–16, https://doi.org/10.35940/ijitee.b8259.0210421. Accessed 5 Sept. 2024.
33. 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.
34. Tomasz Bonus. “Objects Counting by Estimating a Density Map with Convolutional Neural Networks.” *Medium, Towards Data Science*, 27 Sept. 2019,

towardsdatascience.com/objects-counting-by-estimating-a-density-map-with-convolutional-neural-networks-c01086f3b3ec.

35. Zeiler, Matthew D., and Rob Fergus. *Visualizing and Understanding Convolutional Networks*. Nov. 2013, https://doi.org/10.48550/arxiv.1311.2901.

APPENDIX

Code for CNN hyperparameter tuning results(Heavily adapted by Github user Ayushii12):

The code below was used for my experiments. It was utilized to test accuracy by feeding the E-MNIST byclass dataset in CSV format and training and evaluating the model using hyperparameters such as batch size, learning rate, and kernel size.

It was run using Keras, OpenCV, NumPy, Pandas, and Matplotlib in Python on a MacBook Air with an Apple M1 chip, 8GB memory, macOS Ventura 13.5.1, and startup disk Macintosh HD.

Model:

```
import matplotlib.pyplot as plt

import cv2

import numpy as np

import pandas as pd

import keras

from keras.models import Sequential

from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout
```

```
from keras.optimizers import Adamax

from tensorflow.keras.utils import to_categorical



from sklearn.utils import shuffle

train = pd.read_csv("r"/Users/Stelios/EE/Copy of emnist_byclass_train (1).csv").astype('float32')

train = pd.read_csv(r"/Users/Stelios/EE/Copy of emnist_byclass_train (1).csv").astype('float32')

train = np.array(train)

test = pd.read_csv(r"/Users/Stelios/EE/Copy of emnist_byclass_train (1).csv").astype('float32')

#test = pd.read_csv("r"/Users/Stelios/EE/Copy of emnist_byclass_train (1).csv").astype('float32')

test = np.array(test)

train = shuffle(train)

test = shuffle(test)



train_x = train[:, 1:785]

train_y = train[:, 0]

test_x = test[:, 1:785]

test_y = test[:, 0]
```

```
# Reshaping and processing

final_train_x = train_x.reshape(train_x.shape[0], 28, 28, 1)

final_test_x = test_x.reshape(test_x.shape[0], 28, 28, 1)

final_train_y = to_categorical(train_y, num_classes=62)

final_test_y = to_categorical(test_y, num_classes=62)

def build_and_train_model(learning_rate, kernel_count, epochs):

    model = Sequential()

     model.add(Conv2D(filters=kernel_count, kernel_size=(3, 3), activation='relu', input_shape=(28,
28, 1)))

            model.add(Conv2D(filters=kernel_count * 2, kernel_size=(3, 3), activation='relu',
padding='same'))

    model.add(MaxPool2D(pool_size=(2, 2), strides=2))

            model.add(Conv2D(filters=kernel_count * 4, kernel_size=(3, 3), activation='relu',
padding='valid'))

    model.add(MaxPool2D(pool_size=(2, 2), strides=2))

    model.add(Flatten())

    model.add(Dense(256, activation="relu"))

    model.add(Dense(512, activation="relu"))
```

```
    model.add(Dropout(0.5))
    model.add(Dense(62, activation="softmax"))
    optimizer = Adamax(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
      history = model.fit(final_train_x, final_train_y, epochs=epochs, validation_data=(final_test_x,
final_test_y), verbose=0)
       final_val_accuracy = history.history['val_accuracy'][-1]
    return final_val_accuracy
learning_rates = [0.0001, 0.001, 0.01]
kernel_counts = [32, 64, 128]
epochs_list = [2, 5, 10]
results = []
for lr in learning_rates:
    for kc in kernel_counts:
        for epoch in epochs_list:
            accuracy = build_and_train_model(learning_rate=lr, kernel_count=kc, epochs=epoch)
                results.append({'Learning Rate': lr, 'Kernel Count': kc, 'Epochs': epoch, 'Accuracy':
accuracy})
```

```
results_df = pd.DataFrame(results)

import ace_tools as tools; tools.display_dataframe_to_user(name="Model Performance Results", dataframe=results_df)

fig, axs = plt.subplots(1, 3, figsize=(18, 5))

for kc in kernel_counts:
    for epoch in epochs_list:
        subset = results_df[(results_df['Kernel Count'] == kc) & (results_df['Epochs'] == epoch)]
            axs[0].plot(subset['Learning Rate'], subset['Accuracy'], marker='o', label=f'Kernels: {kc}, Epochs: {epoch}')

axs[0].set_title('Effect of Learning Rate on Accuracy')

axs[0].set_xlabel('Learning Rate')

axs[0].set_ylabel('Accuracy')

axs[0].legend()

for lr in learning_rates:
    for epoch in epochs_list:
        subset = results_df[(results_df['Learning Rate'] == lr) & (results_df['Epochs'] == epoch)]
          axs[1].plot(subset['Kernel Count'], subset['Accuracy'], marker='o', label=f'LR: {lr}, Epochs: {epoch}')
```

```
axs[1].set_title('Effect of Kernel Count on Accuracy')

axs[1].set_xlabel('Kernel Count')

axs[1].set_ylabel('Accuracy')

axs[1].legend()

# Plot effect of epochs

for lr in learning_rates:

    for kc in kernel_counts:

        subset = results_df[(results_df['Learning Rate'] == lr) & (results_df['Kernel Count'] == kc)]

        axs[2].plot(subset['Epochs'], subset['Accuracy'], marker='o', label=f'LR: {lr}, Kernels: {kc}')

axs[2].set_title('Effect of Epochs on Accuracy')

axs[2].set_xlabel('Epochs')

axs[2].set_ylabel('Accuracy')

axs[2].legend()

plt.tight_layout()

plt.show()
```