Comparing the performance of a traditional classification algorithm and a deep learning classification algorithm based on the size of the training set

How does the volume of training data affect the classification accuracy and training time of a Support Vector Machine (traditional) compared to Convolutional Neural Network (deep learning)?

Subject: Computer Science

Word count: 3981

Table of Contents

1. Introduction

Image classification is a core function in computer vision, and a necessary feature in automated medical diagnosis, vehicles, and security systems (Golemanova, 2018).

For a computer to classify an image, visual/training data must be fed into a classifier algorithm. Traditional classification algorithms are based on statistical analysis: the process of collecting/interpreting data to find underlying patterns/trends (Brooks, 2020). These algorithms are often simple in structure and require pre-processed labelled data. Comparatively, deep learning classification is based on an Artificial Neural Network: a machine imitation of a real, biological neural network composed of artificial neurons/brain cells. Deep learning models require hardly any user intervention as parameters are set automatically and data is pre-processed within the network (IBM, 2020).

Both types of classification algorithms are used extensively in bioinformatics to categorize genes among other biological issues (Chauhaun, 2019). However, the amount of available training data varies depending on the grandeur of the institution involved, forcing researchers to undergo an iterative selection process to find the most suitable algorithm.

This raises the question “How does the volume of training data affect the classification accuracy and training time of a traditional classification algorithm, Support Vector Machine (SVM), compared to a deep learning classification algorithm, Convolutional Neural Network (CNN)?”. In this paper, the performance of each algorithm on different dataset sizes will be analyzed, making it easier to pinpoint a suitable algorithm for a given dataset and reduce the selection process.

This investigation will use two models, an SVM model imported from sklearn and the CNN model built using Keras. The dataset used is the Modified National Institute of Standards and Technology (MNIST)

dataset of handwritten digits. The training set size (ranging from 1000 - 60000 training images) is altered after each rerun, and the average accuracy and training time after ten trials are recorded. Patterns and logical-mathematical conclusions will be then discussed.

2. Research Question

How does the volume of training data affect the classification accuracy and training time of a Support Vector Machine (traditional) compared to Convolutional Neural Network (deep learning)?

3. Background Information

3.1 Machine Learning

Machine learning is the study of setting computers to carry out tasks without it being explicitly programmed to do so (IBM, 2020). These algorithms generally fall under two categories: supervised and unsupervised. Supervised learning is when algorithms learn from and output predictions based on pre-characterized/labelled data (Johnson, 2021). In other words, the algorithm has to be given pre-answered examples to “learn” to classify similar ones. Unsupervised learning is when algorithms try to learn the inherent structure of the data without being given explicit labels (Johnson, 2021). Image classification is a supervised learning problem: a list of target classes is defined, and a model is trained using labelled photos (Jain, 2019).

3.2 Traditional Classification

Traditional classification algorithms are based on simple mathematical equations that groups/separate data that look similar. SVM, which will be further explained below, generalizes a boundary line that separates

training data into two classes and based on this generated line, it categorizes new/different test data (Gandhi).

3.3 Deep Learning Classification

Deep learning classification is based on Artificial Neural Networks (ANNs), which attempts to mimic the way humans draw conclusions. The network is composed of neurons/nodes, which store values, and connections that store weights. Nodes are split into three categories: input, output, and hidden nodes responsible for interpreting information (IBM, 2021). CNN, the representative model, is composed of a convolution layer (image processing), a pooling layer (reducing resolution), and finally, a fully-connected layer which is the neural network (Das, 2021).

3.4 MNIST DataSet

The dataset used is the Modified National Institute of Standards and Technology (MNIST) dataset, composed of 70 000 digits extensively used in both training and testing machine learning visual classification (Yann, 1998). Each image is bounded by a 28*28 pixel grid (784 pixels in total), and approximately 250 writers contributed handwritten digits to ensure the machine learning algorithm can quantify a variety of handwritten digits.

3.5 Features

In machine learning, a feature is a characteristic/property of an object in an image that helps classify it (Brown, 2021). For example, a triangle has 3 corners and 3 edges - these properties are features of a triangle as it helps people to identify the shape. For MNIST, each number has multiple identifying features, although they may differ based on the handwriting the general features will stay the same, and thus can be classified.

3.6 Support Vector Machines (SVM)

Support Vector Machines (SVM) is a traditional classification algorithm for both non-linear and linear classification problems (Jain, 2019). The fundamental principle behind SVM is straightforward: find the most optimal boundary line/hyperplane that separates n-dimensional training data into two distinct classes (Gandhi, 2018). This line is then generalized on testing data for classification purposes.

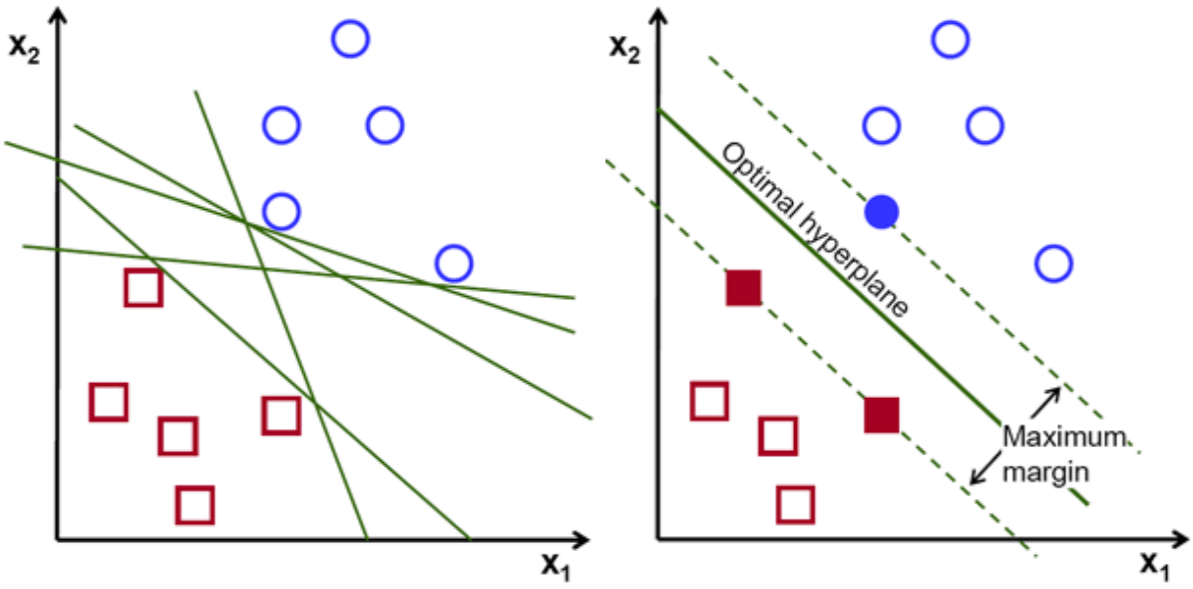


Figure 1: Most optimal hyperplane (Gandhi)

As shown in figure 1, there are a number of possible hyperplanes that can be drawn, the most optimal plane maximizes the distance/margin between the closest data points of both classes (Gandhi, 2018). These points are the support vectors, as they "support" or determine the placement of the decision boundary (Wilimitis, 2019). The clearer the separation, the easier it is to classify new data points as there is a distinct separation between the two classes. Therefore, to train an SVM classifier model, the *most optimal hyperplane* that *maximizes the margin* must be found to generalize new data points and make accurate predictions (Gandhi, 2018).

3.6.1 Multiclass Classification

For multiclass classification, the same principle applies: $\frac{n*(n-1)}{2}$ binary classifiers are used to compare each defined class to another (Marius, 2020). For the MNIST dataset, there are 10 classes of numbers ranging from 0-9, therefore $\frac{10*(10-1)}{2}$ or 45 binary classifiers will be used.

3.6.2 Non-Linear SVM

Although a linear SVM can be used for this experiment, for the purpose of increased accuracy a non-linear SVM will be used - a non-linear or wavy separator would categorize the MNIST data more accurately as a linear separator may incorrectly classify certain data points. To separate data that is not linearly separable, it must be transformed to a higher dimensional feature space where properties of the image are represented as singular points (Brown, 2021). To visualize this transformation, non-linearly separable 2-dimensional data will be shown below:

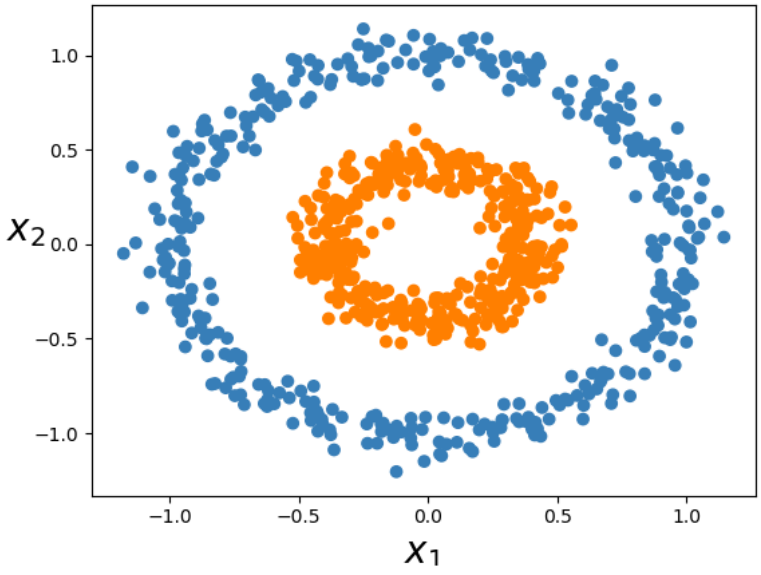


Figure 2: Non-linearly separable data (Wilimitis, 2019)

Evidently, the data is nonlinear and a line can not be drawn to accurately classify the data, thus the data points must be plotted in a 3-dimensional space.

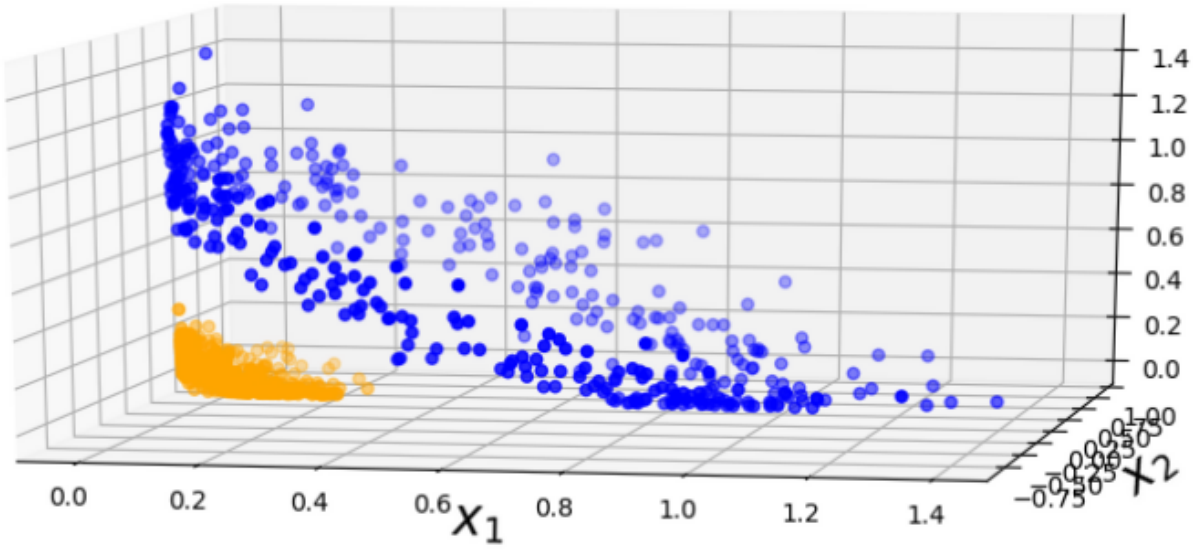


Figure 3: Transformed Data (Wilimitis, 2019)

The data is linearly separable, and a boundary line can now be drawn to divide and classify the data.

However, operations must now be performed in the transformed feature space. Realistically, there are a variety of features within the data, and applying new transformations involving polynomial combinations of these features would generate impractical and high computational costs (Bhattacharya, 2018). Therefore, for the sake of optimization, the Kernel Trick can be used.

3.6.3 Kernel Trick

Instead of representing the data with these higher dimensional coordinates and explicitly applying these transformations, the dataset X can now be represented using n*n kernel matrix where points (i,j) can now be denoted as kernel function k(xi, xj) (Wilimitis, 2019). In other words, adding an entire dimension to the data would be inefficient, so a reversible mathematical kernel function is used instead to easily store

this transformed data while keeping the original pattern. For SVM, linear, polynomial, and radial basis functions (RBF) are used. For this investigation, the RBF kernel will be used due to its popularity and high accuracy (Dobilas, 2021).

3.6.4 C & Gamma Parameters

The C parameter controls the balance between training accuracy and finding the maximum margin for the optimal hyperplane (Kumar, 2020). The larger the C value, the more training points are classified correctly and the more intricate the boundary line is. Conversely, the smaller the C value is, the larger the maximum margin and the smoother the boundary line (Kumar, 2020). A more intricate line does not mean higher accuracy as the line may fit exactly on training data and performs poorly on test data that it was not explicitly trained on.

The Gamma parameter controls the “influence” of a point or how much a single point matters depending on its distance to the boundary line (Kumar, 2020). The greater the value, the decision boundary will be dependent only on the points closest to it, ignoring the further points (more intricate boundary line). The lower the value, the points furthest away are weighted higher, therefore the smoother the boundary line (Kumar, 2020). Thus, finding the balance between the Gamma and C parameters is crucial when training an SVM model.

3.7 Convolutional Neural Networks (CNN)

Convolutional Neural Networks or ConvNet is a deep learning algorithm that takes raw images, assigns importance to features of the object, and uses them to differentiate the image without much preprocessing necessary (Saha, 2018). To achieve this, a CNN model is composed of 3 principal layers: convolution, pooling, and fully-connected. Each of the layers and their properties will be explained below.

3.7.1 Convolutional Layer

Computers see images as *tensors*, where coloured input images are seen as a stack of 3 RGB matrices - red, blue, and green, while grayscale input images are seen as a singular matrix (Special Issue, 2018). These coloured matrices are called *channels*, so a 4x4 RGB image would be represented with 3 channels of 4x4 matrices. For this experiment, grayscale images will be used, therefore only a singular matrix/channel is needed.

Although the CNN model could directly process this tensor representation, it would be computationally slow and impractical for larger datasets (Saha, 2018). Therefore, a Convolution layer is used. Convolution is the mathematical process of combining two signals into one, in this case, these signals are two matrices - the original tensor matrix combined with a filter matrix (Saha, 2018). These filters are called "kernels" and they are responsible for filtering images to decrease the resolution size while keeping the original pixel pattern of the image (Saha, 2018).

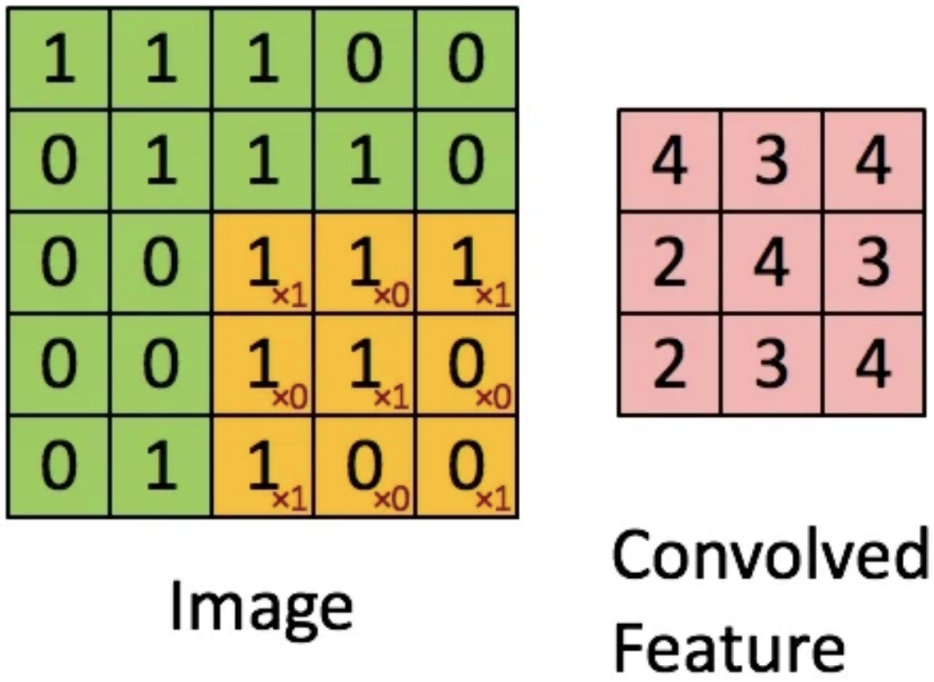


Figure 5.1: Convolution with a 3x3 Kernel Matrix (Das, 2021)

The input matrix at the left represents a single grayscale channel, the 3x3 overlaid yellow matrix represents the kernel filter and the red matrix on the right represents the final convolve matrix. The kernel filter traverses the input matrix rightward with a stride (pixel per step) value till it reaches the rightmost column of the matrix. Then, it moves downwards and starts from the leftmost column, and repeats the entire process till it traverses the entire image (Saha, 2018). Every time the filter rests on a specific section of the matrix, it performs "matrix multiplication" meaning it multiplies the numbers in the matrices by the numbers in the kernel channel. Then, the products are summed and an arbitrary parameter or a "bias" is added. The matrix is then stored within an activation map (Saha, 2018). An is example shown below:

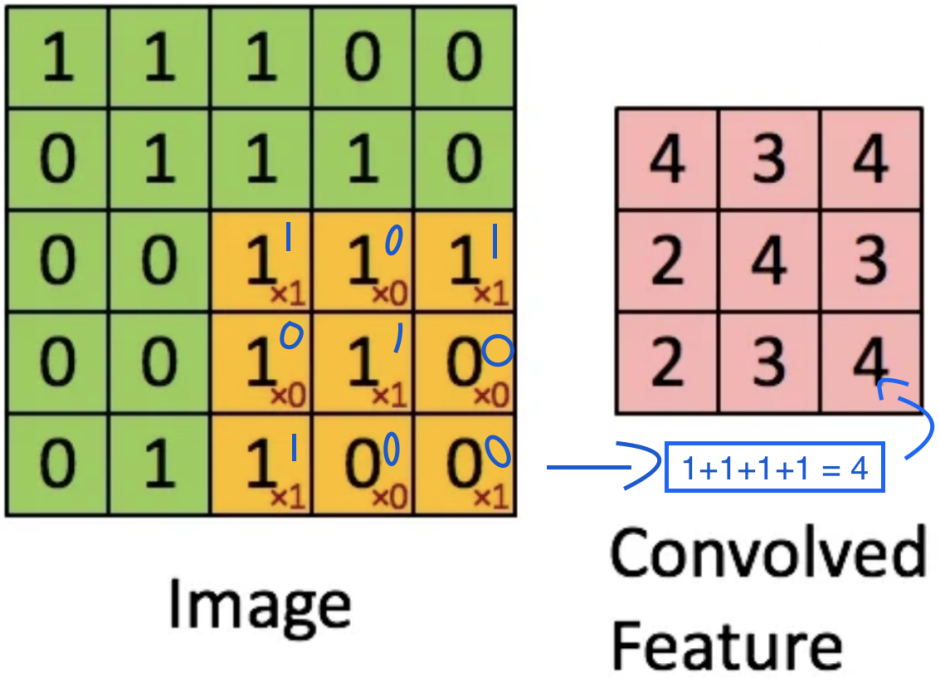


Figure 5.2: Matrix Multiplication (Das, 2021)

The convolved matrix is now stored in an activation map and is now the input for the rest of the model. For the MNIST dataset, a single convolution layer with 32 (3x3) parallel kernel filters will be used. A

modest 32 filters are used to allow the model to “see” the input data in multiple ways as each filter acts as a “feature detector” for specific aspects/features of handwritten digits.

3.7.2 Pooling Layer

A pooling layer is used for resolution reduction after the image is processed within the convolution layer (Brownlee, 2019). It is also used to prevent overfitting, which occurs when models fit exactly to the training data, and perform inaccurately for testing data that it was not explicitly trained on (IBM, 2021). Max Pooling which takes the max value of a certain part of an image under the kernel filter will be used because of its simplicity and popularity (Yalçın, 2018).

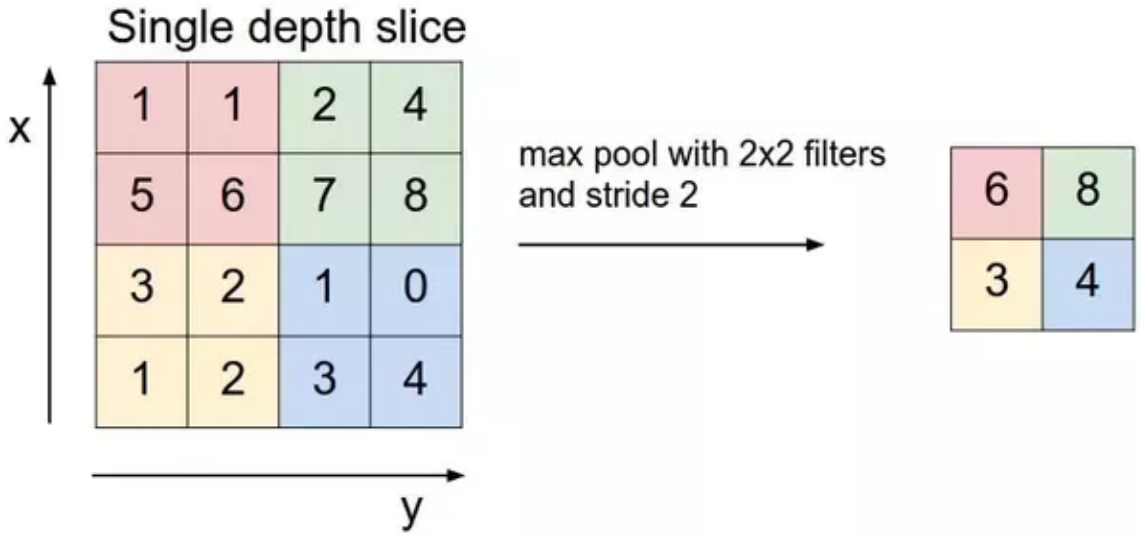


Figure 6: Max Pooling Diagram (Nielson, 2018)

After combining the convolutional layer and pooling layer, the CNN model can now recognize and understand features.

3.7.3 Fully Connected Layer/Neural Network

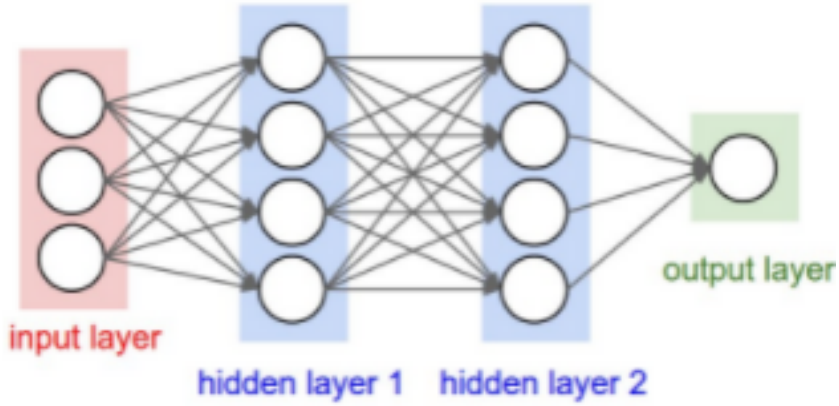


Figure 7: Fully Connected Layer (Karpathy, n.d.)

The final layer is the Fully Connected Layer, it receives the transformed input data (after pooling and convolution). This layer is composed of neurons that receive input, perform an operation and pass on the results to the next neuron layer (Karpathy, n.d.). Once the output finally reaches the terminal layer, the last few neurons output a probability of whether the chosen output is correct. By comparing how far the particular output is from the correct one, a total "cost" is generated (Knocklein, 2019). Gradually, as weights are adjusted, the cost function minimizes, and the network becomes more accurate in its predictions (Knocklein, 2019).

After the fully connected layer is added, a CNN model is then normally evaluated using a K-fold cross validation method where the training set is separated into k groups and used for both testing and training purposes. This evaluation method is popular because of its simplicity and removes some bias/optimism when predicting the model's capabilities. However, for the sake of consistency with the SVM model, the CNN model will be fit with the entire training set size, then evaluated with a consistent 10000 testing images.

4. Methodology

This investigation involves two models that were programmed and fed a specified number of images from the MNIST dataset, accuracy and training time were then recorded.

The SVM model was imported from Sklearn, set with the RBF kernel, and balance between C and Gamma parameters were tested (code in Appendix A, heavily adapted from Kothari). The average accuracy and training time of 10 trials were recorded (set with C & Gamma parameters that resulted in the highest training accuracy).

The CNN model was imported from Keras and the sequential model (a model built layer by layer) was used (code in Appendix B, heavily adapted from Brownlee). A convolutional layer composed of 32 (3x3) kernel filters was added first, then a max-pooling layer and a final fully connected/neural network layer. The accuracy and training time was recorded as the average of the 10 trial evaluations.

4.1 Dataset Used

MNIST dataset of grayscale handwritten digits is composed of 60000 training set examples and 10000 test set examples for 28x28 pixel numbers ranging from 0-9. For this investigation, all 10 classes of the MNIST dataset will be processed, and the training set size will be altered for each trial.

4.2 Dataset Processing for CNN

The dataset is in grayscale, thus only a single channel is necessary, therefore the data arrays can be reshaped to have a single colour channel. There are also 10 classes of unique integers being classified, thus the utility function to_categorical can be used to transform the integer into 10 element binary vectors (1 is the index of the current class value, 0 index of the other class values).

The pixel integers for the grayscale images range between 0- 255 and can be normalized/rescaled to the range of [0,1]. This would involve using float32 instead of int, and dividing the pixel values by 255.

4.3 Variables

4.3.1 Independant

The independent variable in this investigation is the training set size/batch size from the MNIST training set used to train the model. This was altered using mnist[:different sizes] to retrieve parts of the MNIST dataset instead of all 60000 images.

4.3.2 Dependant

Accuracy

Accuracy is quantified by the total number of correct outputs over the total predictions made. A prediction can be considered correct if the algorithm could correctly identify whether this object belongs to this class. The final accuracy recorded was the average of 10 trials performed for each training set size.

Training Time

The amount of time needed for the two models to train on the MNIST training set. This was recorded with the python time module, the start time and end time were recorded and the difference was used as the training time. The final training time was calculated as the average of all 10 trials performed for each training set size.

4.3.3 Constants

- MNIST dataset used
- Testing data used
- Size of the Testing data (10000 images)
- Visual Studio Code IDE

4.4 Procedure

4.4.1 SVM model

1. Imported SVM model from Sklearn and loaded MNIST dataset
2. Loaded all 10000 testing images and labels, and the number of training images and labels used for this trial
3. Created an SVM classifier model and set the kernel to RBF
4. Looped through all possible combinations of C and Gamma to train and test the model
5. Stored the C and Gamma values that give the maximum accuracy
6. Created a new SVM classifier using the recorded C and Gamma variables
7. Performed 10 trials and recorded the average accuracy and training time
8. Repeated steps 2-7 for the other training set sizes

4.4.2 CNN model

1. Imported CNN sequential model from Keras
2. Loaded the MNIST dataset from Keras, and all 10000 testing images and labels, then loaded the number of training images and labels used for this specific trial
3. Used to_categorical() function to transform integers into binary vectors, then rescaled the pixel values to [0,1] and converted the data type to float32 then divided each value by 255
4. Added all necessary layers to the model (Conv, pooling, fully connected)
5. Evaluated the accuracy and training time of the model 10 times, and recorded the average
6. Repeated steps 2-5 for the other training set sizes

5. Results

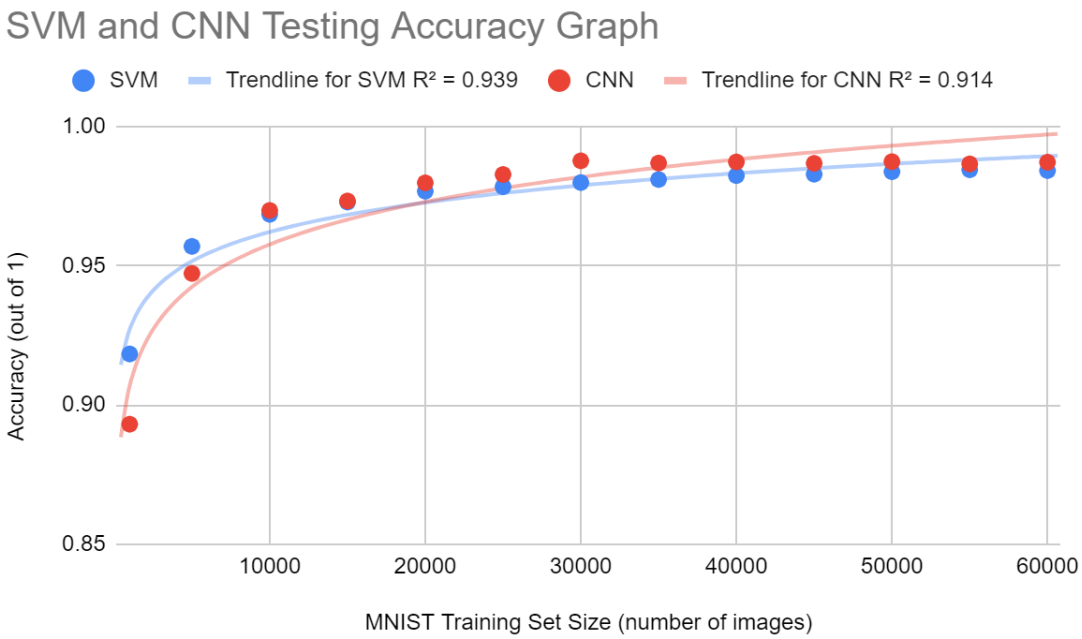
5.1 Tabulated Results

The tabulated results for both classifiers in relation to training set size can be found below. The results from every 10 trials were averaged and recorded.

MNIST Training Set Size	Training Time (SVM) in seconds	Testing Accuracy (SVM)	Training Time (CNN) in seconds	Testing Accuracy (CNN)
1000	1.5523	0.9183	12.578	0.8931
5000	11.324	0.9569	46.983	0.9472
10000	24.996	0.9684	79.932	0.9698
15000	68.632	0.9728	119.34	0.9732
20000	74.392	0.9767	134.52	0.9797
25000	101.93	0.9783	151.95	0.9827
30000	138.53	0.9798	206.15	0.9876
35000	283.95	0.9809	204.14	0.9868
40000	371.85	0.9823	214.79	0.9872
45000	382.13	0.9828	264.32	0.9867
50000	542.78	0.9837	276.97	0.9873
55000	569.76	0.9844	294.36	0.9865
60000	553.32	0.9841	354.45	0.9871

5.2 Graphical Representation of Data

Below is the graphical representation of the data for a clearer understanding of the trends present within the data - both graphs were made using Google Sheets. The blue and red points represent the average accuracy/training time for the SVM and CNN model respectively.

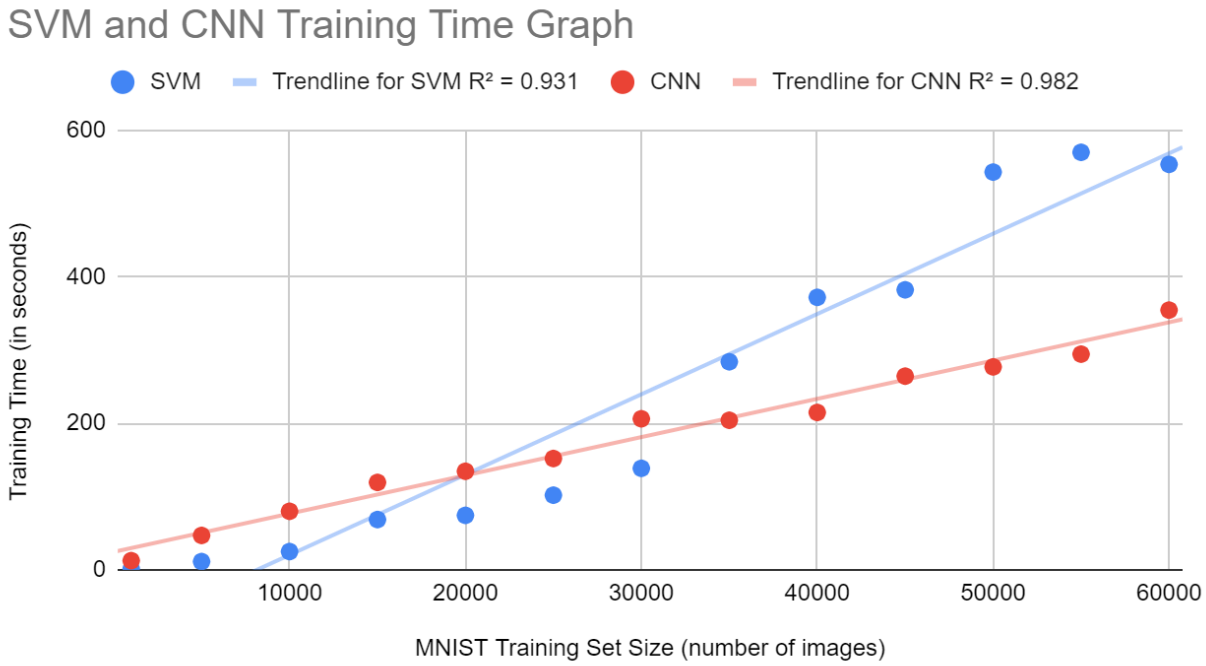


The trendline equations generated using Google Sheets. The accuracy trendlines for both SVM and CNN are respectively:

$y = 0.822 + 0.0152 \ln x$

$y = 0.755 + 0.022 \ln x$

With a R squared (proportion of variance) value of 0.939 and 0.914 respectively.



The trendline equations were generated with Google Sheets. The training time trendline for both SVM and CNN are accordingly:

$y = 0.011x + -89.2$

$y = 5.22E\text{-}03\ x + 24.5$

The R squared value was 0.931 and 0.981 respectively.

5.3 Observations

The accuracy of the SVM model was generally higher than the accuracy of the CNN model when the training set size was smaller (1000-10000 images), and the opposite was true for larger training set sizes (10000-60000 images). Since the trendline used is logarithmic, the rate of increase for accuracy decreases the more training images there are. If more training images were used for either model, the resultant accuracy would only increase by a negligible margin. For training time, SVM was faster when the dataset size was smaller (1000- 30000 images) while CNN was faster when the dataset size was larger

(30000-60000 images). The trend line for the training time was generally steeper for the SVM model compared to the CNN model, meaning CNN scales much better with the size of the dataset.

5.4 Explanation of Results

In terms of accuracy, CNN models generally perform better (higher accuracy) when there is more data available. It is prone to overfitting on smaller sized training sets as a result of the numerous layers, kernels and parameters necessary to build a complete model. Moreover, the more weights within the neural network (10 terminal neurons for MNIST), the higher probability an activation pattern will be fitted explicitly to the training set. Thus, larger datasets are necessary to distribute enough variations of data to avoid this issue. Conversely, SVM models avoid overfitting by finding the most optimal hyperplane, and a balance between C and Gamma parameters. This ensures that an intricate boundary line is not created to fit explicitly on training data, so the model can generalize well with the testing data and output accurate predictions. Thus, SVM models perform better on smaller-scaled data sets, as overfitting is not an issue. However, SVM models must store a n*n kernel matrix, n being the number of instances within the data. As a result, storing the n*n matrix for larger datasets requires a massive amount of memory space, making the model much more inefficient and inaccurate.

In terms of training time, the time complexity for SVM models generally scales superlinearly with the number of data points because of the memory needed to store the kernel matrix. This makes it infeasible for larger datasets. CNN models generally scale linearly with the amount of visual data given, working well with larger datasets.

5.5 Limitations of Methodology Used

1. Only a certain range of C and Gamma parameters were tested for the SVM model, meaning if a higher accuracy/lower training time was possible given different parameters, it was not recorded.

2. The default structure of the CNN model was used due to time constraints. Other models which could have resulted in higher accuracy/lower training time were not tested. These models could potentially have further validated the results and trendlines.
3. Only the MNIST grayscale dataset was used. Differently sized datasets such as the small Corel1000 set or larger image datasets could have been tested to fully validate the conclusion.
4. Both models were not made from scratch, the SVM model was imported from Sklearn, the CNN model was built using Keras pre-built sequential model, and convolution & pooling layers were simply added. If both models were made from scratch optimization techniques could have been applied.

5.6 Further Scope

As a possible extension to this paper, other traditional classification algorithms (K-NN, Logistic Regression, Random Tree Forest, etc.) or other deep learning classification algorithms (MLP) could have been explored to further validate the results. Another interesting extension would be exploring how models would classify RGB visual data with 3 channels, instead of grayscale single-channel MNIST images. The accuracy and training time of both models would be possibly lower due to the complexity of the image being classified - the image would be represented by an RGB (M*N*3) matrix instead of a grayscale (M*N) matrix. However, it is also possible that the accuracy would be higher considering an RGB image provides more features (ex. saturation) helping the algorithm differentiate objects more effectively.

6. Conclusion

Overall, traditional classification algorithms (SVM) performs better when the training set size is smaller while deep learning algorithms (CNN) performs better when the training set size is larger - answering the

research question: “How does the volume of training data affect the classification accuracy and training time of a Support Vector Machine (traditional) compared to Convolutional Neural Network (deep learning)?”.

SVM finds the most optimal boundary line to separate n-dimensional data into 2 distinct classes. The model outputs higher accuracies on smaller datasets as C & Gamma parameters prevent overfitting, allowing the boundary line to generalize well on new data. SVM is slower on larger datasets as the training time scales superlinearly.

CNN or Convolutional Neural Network is a layered model that assigns importance to features of the object, and uses them to differentiate the image - it is composed of 3 layers, convolution for filtering, max pooling for reducing resolution and a fully-connected network which “learns” how to differentiate these images. The model outputs higher accuracies on larger datasets as more variations of data is necessary for the weighted neural network to fit properly. CNN performs more consistently for all dataset sizes, as the training time scales linearly.

Ultimately, this paper compares the classification accuracy and training time of traditional and deep learning classification models - helping determine which model to use depending on the amount of available data.

7. Works Cited

Bhattacharyya, S. (2018, December 19). *Understanding support Vector Machine: Part 2: Kernel TRICK; Mercer's theorem*. Medium. https://towardsdatascience.com/understanding-support-vector-machine-part-2-kernel-trick-mercers-theorem-e1e6848c6c4d.

Brooks, C. (2020, October 19). *What is statistical analysis?* Business News Daily. https://www.businessnewsdaily.com/6000-statistical-analysis.html.

Brownlee, J. (2020, August 20). *A gentle introduction to the rectified linear unit (relu)*. Machine Learning Mastery. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

Brownlee, J. (2020, August 24). *How to develop a CNN for mnist handwritten Digit Classification*. Machine Learning Mastery. https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/.

Brownlee, J. (2019, June 16). *What is the difference between a parameter and A hyperparameter?* Machine Learning Mastery. https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/#:~:text=Parameters%20are%20key%20to%20machine%20learning%20algorithms.&text=In%20this%20case%2C%20a%20parameter,a%20prediction%20on%20new%20data.

Brown, R. (2021, July 19). *What are features in machine learning and why it is important?* Medium. https://cogitotech.medium.com/what-are-features-in-machine-learning-and-why-it-is-important-e72f9905b54d.

Chauhan, N. S. (2019, September). *Explore the world of bioinformatics with machine learning*. KDnuggets. https://www.kdnuggets.com/2019/09/explore-world-bioinformatics-machine-learning.html.

Das, A. (2021, January 11). *Convolution neural network for image processing-using Keras*. Medium. Retrieved November 15, 2021, from

https://towardsdatascience.com/convolution-neural-network-for-image-processing-using-keras-dc342905 6306.

Dobilas, S. (2021, January 19). *SVM classifier and RBF kernel - how to make better models in Python*. Medium. https://towardsdatascience.com/svm-classifier-and-rbf-kernel-how-to-make-better-models-in-python-73bb 4914af5b.

Gandhi, R. (2018, June 7). *Support vector machine - introduction to machine learning algorithms*. Medium. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934 a444fca47.

Golemanova, R. (2019, June 27). *7 image recognition uses of the future*. Imagga Blog. https://imagga.com/blog/7-image-recognition-uses-of-the-future/.

IBM Cloud Education. (2020, July 15). *What is machine learning?* IBM. https://www.ibm.com/cloud/learn/machine-learning.

IBM Cloud Education. (2021, March 3). *What is overfitting?* IBM. https://www.ibm.com/cloud/learn/overfitting.

IBM Cloud Education, I. B. M. (2020, May 1). *What is deep learning?* IBM. https://www.ibm.com/cloud/learn/deep-learning.

Jain, T. (2019, August 29). *Basics of machine learning image classification techniques*. OpenGenus IQ: Computing Expertise & Legacy. https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/.

Johnson, D. (2021, August 28). *Supervised vs unsupervised learning: Key differences*. Guru99. https://www.guru99.com/supervised-vs-unsupervised-learning.html.

Karpathy, A. (n.d.). *Convolutional Neural Networks (CNNs / ConvNets)*. CS231n convolutional neural networks for visual recognition. https://cs231n.github.io/convolutional-networks/.

Kothari, D. (n.d.). SVM with MNIST. https://dmkothari.github.io/Machine-Learning-Projects/SVM_with_MNIST.html.

Knocklein, O. (2019, June 15). *Classification using neural networks*. Medium. https://towardsdatascience.com/classification-using-neural-networks-b8e98f3a904f.

Kumar, A. (2020, July 18). *SVM RBF kernel parameters with code examples*. Data Analytics. https://vitalflux.com/svm-rbf-kernel-parameters-code-sample/.

Marius, H. (2020, June 9). *Multiclass classification with support vector Machines (SVM), kernel trick & kernel functions*. Medium. https://towardsdatascience.com/multiclass-classification-with-support-vector-machines-svm-kernel-trick-kernel-functions-f9d5377d6f02.

Nielsen, D. (2018, September 9). *Deep learning Cage Match: Max POOLING VS CONVOLUTIONS*. Medium. https://medium.com/@duanenielsen/deep-learning-cage-match-max-pooling-vs-convolutions-e42581387cb9.

Saha, S. (2018, December 15). *A comprehensive guide to convolutional neural networks - the eli5 way*. Medium. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

Special Issue. (2018, November 10). *Tensor image processing*. Guide 2 Research. https://www.guide2research.com/special-issue/tensor-image-processing.

Wilimitis, D. (2019, February 21). *The kernel trick*. Medium. https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f.

Yalçın, O. G. (2018, August 19). *Image classification in 10 minutes with MNIST Dataset*. Medium. https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d.

Yann, L., Corinna, C., & Chris, B. (1998). *The mnist database*. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. Retrieved November 15, 2021, from http://yann.lecun.com/exdb/mnist/.

8. Appendices

Appendix A

```
from mnist import MNIST
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import time
import pandas as pd

print("Loading dataset...")
#loading the dataset
mndata = MNIST("/Users/leonachen/Downloads/")
images, labels = mndata.load_training()
img, lbl = mndata.load_testing()

#get training size
train_x = images[:45000]
train_y = labels[:45000]
#get testing size
test_x = img[:10000]
expected = lbl[:10000].tolist()

#test all c parameters
C_2d_range = [1e1, 1e2, 1e3, 1e4, 1e5]
# G_2d_range = [1e-2, 1e-1, 1,1e1, 1e2, 1e3, 1e4, 1e5]

m_a=0
m_c=0
m_g=0

#find best c and g parameter
for C in C_2d_range:
   clf = SVC(kernel = 'rbf', C=C, gamma='scale')
   print("Train model")
   start = time.time()
   clf.fit(train_x, train_y)
   stop = time.time()
   print(f"Training Time: {stop - start}s")
```

```
   print("Compute predictions")
   predicted = clf.predict(test_x)

   print("Accuracy: ", accuracy_score(expected, predicted))
   if m_a<accuracy_score(expected, predicted):
       m_a=accuracy_score(expected, predicted)
       m_c=C

# for x in range(5):
#     clf = SVC(kernel = 'rbf', C=m_a, gamma='scale')


print("Accuracy: ", m_a)



# Test on the next 1000 images:
from mnist import MNIST
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import time
import pandas as pd

print("Loading dataset...")
#loading the dataset
mndata = MNIST("/Users/leonachen/Downloads/")
images, labels = mndata.load_training()
img, lbl = mndata.load_testing()

#get training size
train_x = images[:45000]
train_y = labels[:45000]
#get testing size
test_x = img[:10000]
expected = lbl[:10000].tolist()

#test all c parameters
C_2d_range = [1e1, 1e2, 1e3, 1e4, 1e5]
# G_2d_range = [1e-2, 1e-1, 1,1e1, 1e2, 1e3, 1e4, 1e5]

m_a=0
m_c=0
m_g=0
```

```
#find best c and g parameter
for C in C_2d_range:
   clf = SVC(kernel = 'rbf', C=C, gamma='scale')
   print("Train model")
   start = time.time()
   clf.fit(train_x, train_y)
   stop = time.time()
   print(f"Training Time: {stop - start}s")
   print("Compute predictions")
   predicted = clf.predict(test_x)

   print("Accuracy: ", accuracy_score(expected, predicted))
   if m_a<accuracy_score(expected, predicted):
       m_a=accuracy_score(expected, predicted)
       m_c=C

# for x in range(5):
#     clf = SVC(kernel = 'rbf', C=m_a, gamma='scale')


print("Accuracy: ", m_a)


# Test on the next 1000 images:
from mnist import MNIST
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import time
import pandas as pdfrom mnist import MNIST
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import time
import pandas as pd

print("Loading dataset...")
#loading the dataset
mndata = MNIST("/Users/leonachen/Downloads/")
images, labels = mndata.load_training()
img, lbl = mndata.load_testing()

#get training size
```

```
train_x = images[:45000]
train_y = labels[:45000]
#get testing size
test_x = img[:10000]
expected = lbl[:10000].tolist()

#test all c parameters
C_2d_range = [1e1, 1e2, 1e3, 1e4, 1e5]
# G_2d_range = [1e-2, 1e-1, 1,1e1, 1e2, 1e3, 1e4, 1e5]

m_a=0
m_c=0
m_g=0

#find best c and g parameter
for C in C_2d_range:
   clf = SVC(kernel = 'rbf', C=C, gamma='scale')
   print("Train model")
   start = time.time()
   clf.fit(train_x, train_y)
   stop = time.time()
   print(f"Training Time: {stop - start}s")
   print("Compute predictions")
   predicted = clf.predict(test_x)

   print("Accuracy: ", accuracy_score(expected, predicted))
   if m_a<accuracy_score(expected, predicted):
       m_a=accuracy_score(expected, predicted)
       m_c=C

# for x in range(5):
#     clf = SVC(kernel = 'rbf', C=m_a, gamma='scale')


print("Accuracy: ", m_a)


# Test on the next 1000 images:


print("Loading dataset...")
#loading the dataset
```

```
mndata = MNIST("/Users/leonachen/Downloads/")
images, labels = mndata.load_training()
img, lbl = mndata.load_testing()

#get training size
train_x = images[:45000]
train_y = labels[:45000]
#get testing size
test_x = img[:10000]
expected = lbl[:10000].tolist()

#test all c parameters
C_2d_range = [1e1, 1e2, 1e3, 1e4, 1e5]
# G_2d_range = [1e-2, 1e-1, 1,1e1, 1e2, 1e3, 1e4, 1e5]

m_a=0
m_c=0
m_g=0

#find best c and g parameter
for C in C_2d_range:
   clf = SVC(kernel = 'rbf', C=C, gamma='scale')
   print("Train model")
   start = time.time()
   clf.fit(train_x, train_y)
   stop = time.time()
   print(f"Training Time: {stop - start}s")
   print("Compute predictions")
   predicted = clf.predict(test_x)

   print("Accuracy: ", accuracy_score(expected, predicted))
   if m_a<accuracy_score(expected, predicted):
       m_a=accuracy_score(expected, predicted)
       m_c=C

# for x in range(5):
#     clf = SVC(kernel = 'rbf', C=m_a, gamma='scale')


print("Accuracy: ", m_a)
```

```
# Test on the next 1000 images:
```

Appendix B

```
# imports
from numpy import mean
from numpy import std
from matplotlib import pyplot
from sklearn.model_selection import KFold
from keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
import time
from tensorflow.keras.optimizers import SGD


def load_dataset():
# function for training and loading numbers
   (tx, ty), (tex, tey) = mnist.load_data()
       # getting number of images
   tnX=tx[:55000]
   tnY=ty[:55000]
   testX=tex[:10000]
   testY=tey[:10000]

# reshaping inputted data to get the single grayscale channel
   tnX = tnX.reshape((tnX.shape[0], 28, 28, 1))
   testX = testX.reshape((testX.shape[0], 28, 28, 1))
# transform integer into classes
   tnY = to_categorical(tnY)
   testY = to_categorical(testY)
   return tnX, tnY, testX, testY

# change the pixels or rescale them
def prep_pixels(train, test):
# change the data type
```

```
   tn_norm = train.astype('float32')
   test_norm = test.astype('float32')
# put it in a 0 to 1 range
   tn_norm /=255.0
   test_norm /= 255.0
# return the values
   return tn_norm, test_norm

#make the convolution model
def define_model():
# add sequential layer
   model = Sequential()
# add other layers
   model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',
input_shape=(28, 28, 1)))
# poolin
   model.add(MaxPooling2D((2, 2)))
#dense/activation layer necessary for Keras
   model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
   model.add(Dense(10, activation='softmax'))
       # compile the model
   model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
   return model
#understanding the models performance
def evaluate_model(dataX, dataY):
# scores and history
   sc, his = list(), list()
# loopppp
   for x in range(10):

       model = define_model()
       tnX, tnY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix],
dataY[test_ix]
       # start the training time
       start = time.time()
       history = model.fit(tnX, tnY, epochs=10, batch_size=32, validation_data=(testX,
testY), verbose=0)
# end it
       end = time.time()-start
# output training time and record
       print("training time: ",end)
#output the accuracy
```

```
        _, acc = model.evaluate(testX, testY, verbose=0)
# round it
        print('> %.4f' % (acc))
        # stores the info somewhere
        sc.append(acc)
        hs.append(history)
    return sc, his
# running the code
def run_test_harness():
    # get the dataset
    tnX, tnY, testX, testY = load_dataset()
    # and make the pixels the right size
    tnX, testX = prep_pixels(tnX, testX)
# check/evaluate the accuracy and training time
    sc, his = evaluate_model(tnX, tnY)

# finally run the code
run_test_harness()
```