COVENTRY

UNIVERSITY

COURSE: M.Sc. COMPUTER SCIENCE

MODULE CODE:7088CEM

STUDENT ID:13143956

SUBMITTED BY

: RAJINDER KAUR

SUBMITTED TO:

Prof. ANUP PANDEY

7088CEM

ARTIFICIAL NEURAL

NETWORKS

**FACULTY OF ENGINEERING, ENVIRONMENT AND COMPUTING**

**7088CEM ARTIFICIAL NEURAL NETWORKS**

**Table of Contents**

[Section 1. Introduction 3](#_Toc127375847)

[Section 2. Related work 4](#_Toc127375848)

[Section 3. Dataset 5](#_Toc127375849)

[Section 4. Method(s) 6](#_Toc127375850)

[Data Analysis Method 7](#_Toc127375851)

[Section 5. Experimental results 8](#_Toc127375852)

[Section 6. Discussion and future work 18](#_Toc127375853)

[Conclusion 19](#_Toc127375854)

[References 20](#_Toc127375855)

[Appendix 1 22](#_Toc127375856)

[Appendix 2 33](#_Toc127375857)

# Section 1. Introduction

Handwritten digit recognition is the process of recognizing the human handwritten digits from several resources such as handwritten papers, screening and recognizing those digits with ten predefined digits 0-9. This image recognition is the most bounding research aspect using the deep learning method. This kind of digit identity is the most valuable process, to recognize the number plate, postal code numbers, check processing of the banking system etc. While this research analysis of many problems occur, different handwriting of different humans. In this research, several methods and algorithms are differentiated comprehensively among several machine learning algorithms while handwritten digits reorganization. For this image recognition, the use of the deep learning model such as Artificial Neural Networks, Convolution Neural Networks and Adam Optimizers going to use. By utilizing those algorithms the training, testing, accuracy execution and errors and several kinds of performance like visualization and plotting have effective use.

This research performance gives detailed information and understating of the different kinds of algorithms like the ANN, CNN and Adam Optimizer while the performance of the handwritten identifying performance. Additionally, the performance of the image digit reorganization provides information on the different kinds of algorisms of the uses algorithms. Furthermore, in this research paper, the related works, data collection methods, detailed operation of the data analysis method and many more are provided.

The MNIST handwritten digit images are a real-world problem that has become increasingly important for machine learning and artificial intelligence systems. This has been the real-world problem that has been chosen for this assignment. A few identified issues are:

1. Classifying images of digits correctly: Since handwritten digits can vary greatly in terms of their shape, size and orientation, it can be difficult to accurately classify them.

2. Separating the foreground from the background: Many images of handwritten digits contain a lot of background noise, making it difficult to accurately distinguish between the foreground and the background (Pashine, Dixit, & Kushwah, 2021).

3. Robustness against variations in handwriting: It can be difficult to accurately classify images of handwritten digits that are written in different styles.

The significance of the MNIST dataset is that it provides an important testing ground for machine learning algorithms, helping to identify successful algorithms and areas of improvement. It is also a great dataset for teaching machine learning as it provides a simple, yet challenging problem that can be used to demonstrate the power of machine learning algorithms (Ahlawat et al., 2021). Moreover, it has been used to develop new algorithms that can recognize patterns in image data, as well as provide insight into how humans perceive and classify patterns in images.

# Section 2. Related work

After the invention of the process of the Artificial Intelligence, there have been several kinds of analysis and invention along with the aspects of Machine Learning and deep learning algorithms (Ahmed et al., 2023). Within the right time, using a large amount of data, the machines get sophisticated, by calculating the mathematical implementation to give execution of real-life problems. Likely, human digit reorganization by using papers and several things, the use of the machine learning algorithms are used to analyze and recognize and classify the images. Several kinds of machine learning algorithms like CNN, ANN and Adam Optimizer have been used to calculate and analysis of handwritten digits.

According to Ahlawat, & Choudhary, (2020), MNIST handwritten digit images problem is a well-known and widely used benchmarking problem in the field of computer vision and pattern recognition. It involves recognizing the hand-written digits from a collection of images. The dataset taken by the author consists of 60,000 images for training and 10,000 for image testing of size 28x28 pixels. Each image contains a single handwritten digit between 0 to 9. The challenge is to accurately classify the digits in the test images using the training set. The goal of this research paper is to develop an efficient hybrid CNN-SVM classifier that can accurately recognize handwritten digits from the MNIST dataset.

According to Prabhu, (2019), the MNIST handwritten digit images problem is a well-known problem in machine learning and computer vision. It involves recognizing handwritten digits from scanned images. The task is to correctly identify the digit in the image. This problem has been widely used in machine learning research and has been successfully used in many applications such as automatic checkout systems, handwriting recognition, and character recognition. The problem is well-studied and has been explored extensively. The dataset used in the research paper is the Kannada-MNIST dataset which is a new dataset of handwritten digits in the Kannada language. The dataset contains 60,000 images of handwritten digits in the Kannada language, which is an Indian language spoken by nearly 44 million people in the southern part of India. The dataset is used to evaluate and compare the performance of various machine learning algorithms on the Kannada-MNIST dataset.

The problem that this paper addresses is the recognition of handwritten digits using deep convolution self-organizing map networks. Handwritten digit recognition has been a challenging task due to the variability of handwriting styles, and existing methods have struggled to achieve high accuracy (Aly, & Almotairi, 2020). The authors of this paper proposed a deep convolution self-organizing map network (DCSOM) to address this problem.

The findings of this paper indicate that the proposed DCSOM was able to achieve high accuracy in recognizing handwritten digits. The authors evaluated the DCSOM on a benchmark dataset of handwritten digits and found that it was able to achieve an accuracy of 99.9%. This was significantly higher than the accuracy of existing methods, which was only around 97%.

The discussion in this paper focuses on the effectiveness of the proposed DCSOM in recognizing handwritten digits. The authors discussed how the DCSOM was able to achieve high accuracy in recognizing handwritten digits due to its ability to learn features from the data in an unsupervised manner. They also discussed the potential of the DCSOM to be used in other recognition tasks, such as recognizing handwritten characters and handwriting styles.

# Section 3. Dataset

This dataset consists of images of handwritten digits, ranging from 0 to 9, collected from various sources. The dataset has been used to train and test a variety of machine learning algorithms and has been widely adopted by the research community as a benchmark for evaluating the effectiveness of machine learning algorithms.

The link to the dataset is:

<https://www.kaggle.com/code/prashant111/mnist-deep-neural-network-with-keras/>

The dataset consists of the main dataset, test dataset, and training dataset. The training dataset was helpful for training the model while the prediction was made using the test dataset. The test and training dataset consists of attributes of Level and various pixels for the development and prediction through the neural network.

All the datasets were cleaned before providing input to the model. These processes were followed to pre-process the data.

1. Remove Duplicates: The first step in cleaning and formalizing a data set is to remove any duplicate entries (Plantin, 2019). This can be done by identifying any rows with the same values and deleting them.

2. Formatting: Format the data set by ensuring that all data is in the correct format, such as ensuring dates are in the correct format and that text is formatted correctly.

3. Type Conversion: Convert data into the correct type, such as converting a text column into a numeric column if it is needed for analysis.

4. Missing Values: Handle any missing values, such as by imputing the median or mean value for a missing value in a numerical column, or by using a mode value for a missing value in a categorical column (Sanjar et al., 2020).

# Section 4. Method(s)

**Data Collection Methods**

The quantitative and secondary data were collected from the provided link. Qualitative data was also collected for this research. The relevant journals aligned to the problem were collected and the literature review was performed. As discussed the dataset was divided into training and test dataset.

# Data Analysis Method

**ANN**

Artificial Intelligence is the most valuable giant technology in this generation, which has several characteristics, like machine learning deep learning and all. Here in this present assessment, the deep learning method of an Artificial Neural Network has been used to classify the MNIST digits characteristics (Albattah and Albahli, 2022). Machine learning is the subpart of AI which allows machines to develop a model of self-learning without explicit programming. The ML-embedded programmers can learn, change and apply according to the dataset. Taking help from these technology computers can analyze and execute valuable pieces of information. The deep learning method is the class of ML algorithms that executes higher levels of features from the dataset progressively taking multiple layers. MNIST raw dataset is used here to execute the features and classify the images.

ANN has been used for developing the software model. ANNs (Artificial Neural Networks) are a powerful technique for image recognition, and they are especially well-suited to the MNIST handwritten digit images dataset. This is because ANNs are designed to mimic the human brain, which is well-equipped to recognize patterns and shapes in images. ANNs can be trained to recognize patterns in the MNIST dataset, and they can learn to respond to different inputs with different outputs. Additionally, ANNs are able to make use of a large amount of data, which is essential for image recognition tasks. The MNIST dataset is a large collection of images, and ANNs can be used to train the network to recognize the patterns in each image. Finally, ANNs are able to generalize the patterns they have learned, which allows them to accurately recognize patterns in images that they have not seen before. This is especially important for the MNIST dataset, which contains a wide variety of handwritten digits.

**CNN**

The convolution neural network is a character of network architecture for deep learning methods and algorithms, which is specially used for image detection and those tasks that process pixel data. There are several kinds of neural network features of deep learning but in identifying the image and image recognition or object identifying several objects, the convolution neural network is the better choice among them (Xin and Wang, 2019). It is a type of supervised machine learning method, which means that the machines are well-trained to utilize the label of the data, training data and execute the output by predicting the data. The labeled data is those data which are already existing with some output data. In supervised machine learning the models learns from the labeled data, here the model learns from every type of data. After the learning and training process, the model is tested using the tested data, and after the data is predicted.

CNN model was also used as part of solving the problem. CNNs are well suited for MNIST handwritten digit images because they are able to capture the spatial relationships between the pixels that make up the images. CNNs have been used extensively for image recognition tasks such as MNIST because they are able to extract the features from the images that are important for the classification task. They are also able to learn the features that are important for distinguishing between different classes of digits. This makes them an ideal choice for classifying MNIST images as they can learn the features that are important for identifying the different digits in an image.

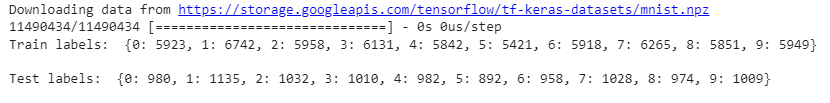
**Adam Optimization Method**

Adaptive moment optimization is a deep learning for optimizing processes for gradient descent. Big problem using a big dataset and using a lot of data parameters this Adam optimizer is very effective. Adam optimizer is the combination of RMSP (root mean square propagation) and gradient descent momentum (Ngu and Lee, 2022). Less power and less memory are required by Adam Optimizer. This Adam optimizer has been used in this present MNIST data analysis.

Adam Optimization Method is an effective optimization algorithm that is used to train deep learning models. It is useful for MNIST handwritten digit images because it can accurately capture the complex patterns and relationships between the pixels to identify and classify the digits. Adam is a variant of stochastic gradient descent, which is known to be efficient and reliable for large datasets. It uses adaptive learning rates to make sure the model converges more quickly and efficiently. Adam also helps to reduce the amount of training time required by allowing the model to update its parameters more effectively. In addition, Adam can help prevent over fitting by using a combination of momentum and adaptive learning rates to reduce the variance of the gradients. All these features make Adam a great choice for training MNIST handwritten digit images.

# Section 5. Experimental results

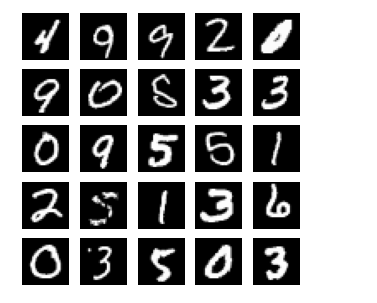
Artificial neural networks and classification models are used here in this particular data analysis performance of the image processing analysis technique. The details of the classification and the regression analysis techniques are illustrated. Step-by-step execution results of the current analysis are illustrated further. *MNIST* *dataset* has been used here for the image processing analysis method. *MNIST* dataset is actually the handwritten digits, those are used here as a dataset and analysed by the image processing technique. The train labels and the test labels are executed in an array format. The executed result of the train and test labels is provided below in figure 1.

****

**Figure 1: Train and Test Tables**

(*Source:* Self-Created)

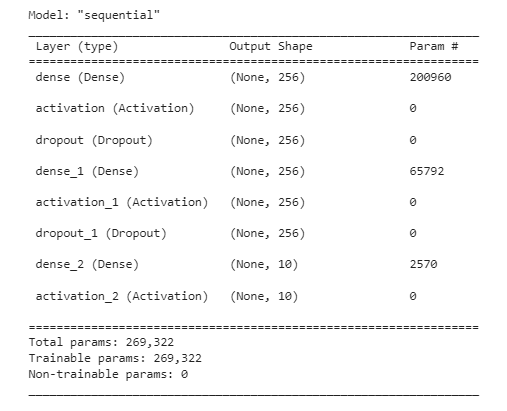
The execution png image is executed depending on the x-train and y\_train labels. Using the *pandas* library of the Python programming language the execution is performed. As an output, the *MNIST* data image has been executed. *Plt.show()* function has been used to execute the image file as an output. Figure 2 is provided below.

****

**Figure 2: MNIST png image**

(*Source:* Self-Created)

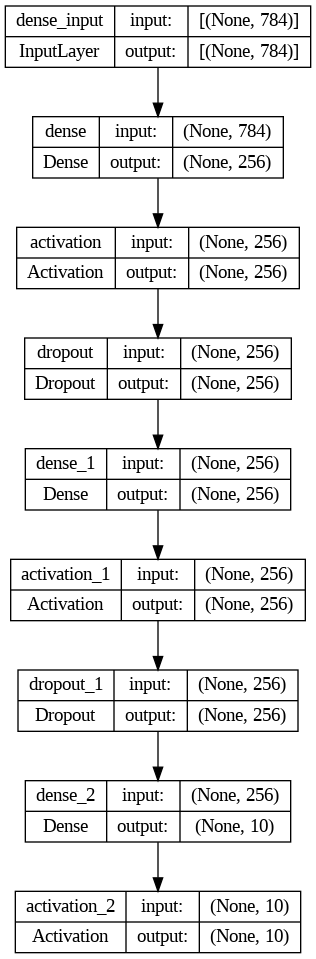
In figure 3, the model summary of the *Artificial Nural Network* has shown. The total parameters are *269322*, the trainable parameters are *269322* and the non-parameters values are *0* in this case.

****

**Figure 3: Model summary**

(Source: Self-Created in Google Colab )

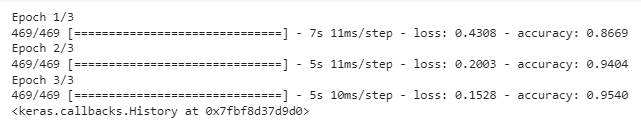
The plot model has been executed as *mlp-mnist.png*. The input and the input layers are executed by plotting the image. The image plot has been provided in figure 4.

****

**Figure 4: Model plot**

(Source: Self-Created in Google Colab )

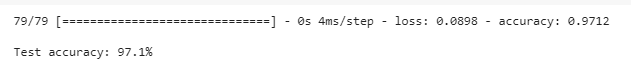
Next, the model epochs are executed of the used ANN model. Up to three epochs are executed here in this analysis performance. In the first epoch, the accuracy is executed as 0.86, in the second epoch the accuracy is executed as 0.94 and in the third epoch is executed as 0.95. Below, in figure 5, the model epochs are shown.

****

**Figure 5: Model epochs**

(Source: Self-Created in Google Colab )

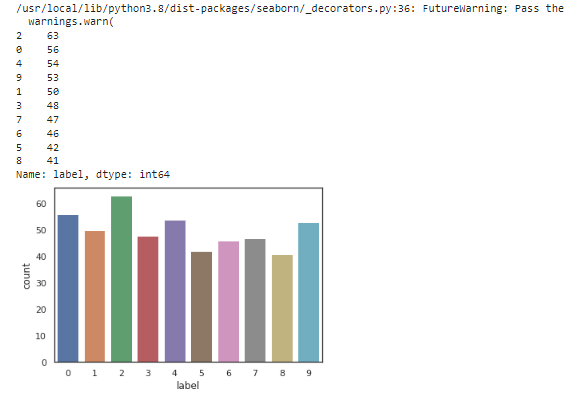
Next, the test accuracy has been shown in figure 6. The test accuracy has been executed as 97.1%, which is a very good accuracy score.

****

**Figure 6: Test accuracy**

(Source: Self-Created in Google Colab )

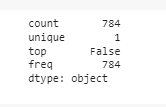
The count plot of the test accuracy is shown below in figure 7. The x-axis of the label represents the label and on the other side, the y-axis represented the count. The seaborn library helped to represent the count plot.

****

**Figure 7: Test accuracy count plot**

(Source: Self-Created in Google Colab )

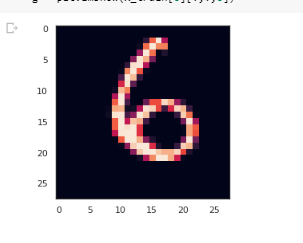
Tested the data by using the describe() function. The output of the test data is given below in the figure 8. The maximum value of count is 784 and the frequency of the data is 784, along with the data type, was an object.

****

**Figure 8: Test the data**

(Source: Self-Created in Google Colab )

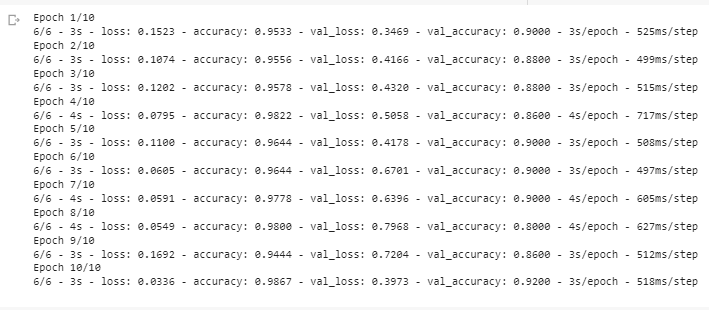
The image of the test data has been executed and provided in figure 9. The x\_train data has been used for the image showing. The x-axis of the image frequency and the y-axis represent the count's values.

****

**Figure 9: Image show**

(Source: Self-Created in Google Colab )

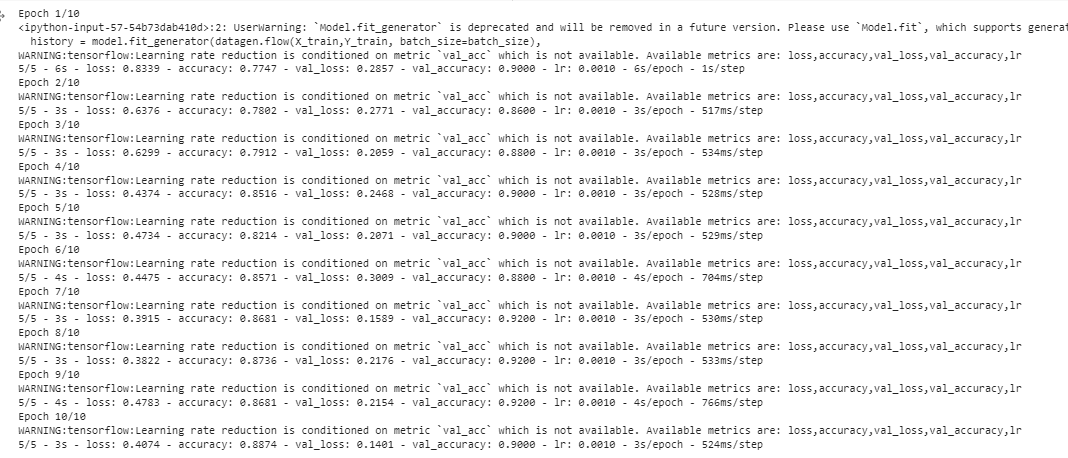
The epochs are generated in this part and the execution model has been shown in figure 10. 10 epochs are executed to classify the accuracy of the used *CNN* model. 10 accuracies are 0.95, 0.95, 0.95, 0.98, 0.96, 0.96, 0.97, 0.98, 0.94 and 0.98 respectively.

****

**Figure 10: Model history epoch**

(Source: Self-Created in Google Colab )

The history epoch model has been executed here in this part of the analysis. The execution image has been provided below in figure 11.

****

**Figure 11: Fit the model**

(Source: Self-Created in Google Colab )

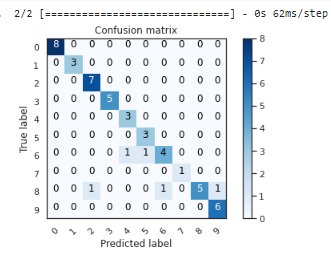
The loss and accuracy plot has been provided in below figure 12. Two several plots are executed respectively. The first one is training loss vs validation loss and another image is training accuracy vs validation accuracy.

****

**Figure 12: Loss and accuracy plot**

(Source: Self-Created in Google Colab )

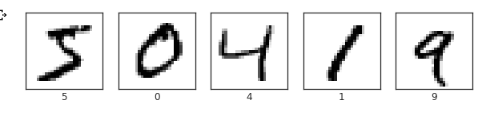
The confusion matrix of the *CNN* model has been provided below in figure 13. The x-axis represents the predicted label and the y-axis represented the true label.

****

**Figure 13: Confusion Matrix**

(Source: Self-Created in Google Colab )

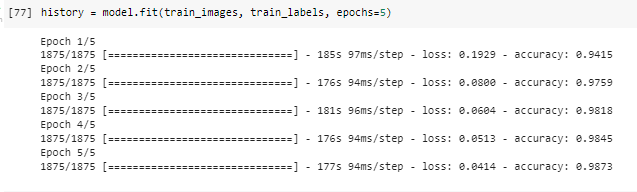
In figure 14, the trained images have been provided below.

****

**Figure 14: Trained images**

(Source: Self-Created in Google Colab )

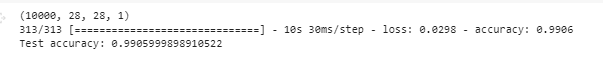
The epoch of the classification model has been provided below in figure 16. The accuracy is respectively 0.94, 0.97, 0.98, 0.98 and 0.98.

****

**Figure 15: Epoch of the classification model**

(Source: Self-Created in Google Colab )

In figure 16, the accuracy of the classification model has been provided. The final accuracy of the model is 0.99.

****

**Figure 16: Accuracy of the classification model**

(Source: Self-Created in Google Colab )

# Section 6. Discussion and future work

Artificial Intelligence (AI) is developed by the functionalities of the human brain, and this technology has Neural Networks like the human brain waves. Machine learning is the sub-part of AI and deep learning is part of machine learning. The algorisms like CNN, ANN, Logistic regression, SVM and many more are popular algorithms of machine learning and deep learning (Ahlawat and Choudhary, 2020). Machine learning algorithms are developed by the implementation of mathematics. Predicting the data using the algorithms is the mathematical implementation to analyse and execute the data output predictions. The difference between machine learning and deep learning algorithms is that the machine learning algorithms are the AI, which can automatically adapt to human interference. On the other hand, deep learning is part of machine learning, where it uses the human pattern brain waves neural network. To parse data machine learning utilizes algorithms, it trains by the prior data and makes decisions from the data, Where deep learning algorithms create layers to make sophisticated decisions on their own. This deep learning is the subset of machine learning.

Image processing using the machine learning algorithms like the ANN, CNN and the classification model. The Python programing language is used for image classification analysis. The machine learning models consist the mathematical implementation. Several kinds of mathematical implementations like regression models and classification models have been used. MNIST dataset has been used for image classification analysis. Several kinds of Python libraries like *Numpy, pandas, seaborn* and many more are used for the specific model integration and analysis of the *MNIST* dataset.

The future work of image processing analysis is provided in this section of the research paper. The development of AI and ML is booming in this era of digital implementation. The image processing method is used for intelligent and smart human life. The mathematical implementation of machine learning and artificial networks uses the mathematical formula. Several kinds of mathematical implementations are implementing in this generation to improve the level of AI and ML. All the implementations and the formulations are developed by the scientist, and the scientist is developing the mechanism in future.

# Conclusion

The traditional MNIST image identification has been performed in this research paper successfully depending on the convolutional neural network, artificial neural network and adam optimizer algorithms of deep learning. Three deep learning functionalities algorithms have been successfully used and tested. The MNIST data set consists of several kinds of human handwritten digits and using deep learning purposes and methods the digits are recognized successfully. The advantages and the disadvantages of the uses of the deep learning process have been discussed successfully prior in this current analysis report. The use of deep learning helped a lot to apply the applications and execute the values of the dataset.

Machine learning algorithms are used for further image processing analysis. The MNSIT data has been used for image processing analysis. The artificial neural network and the convolution neural network have been used for further analysis. Mathematical implementations like the regression and classification analysis have been done here. The use of the Python libraries helped here to demonstrate the executions.

# References

Ahlawat, S., & Choudhary, A. (2020). Hybrid CNN-SVM classifier for handwritten digit recognition. *Procedia Computer Science*, *167*, 2554-2560. <https://www.sciencedirect.com/science/article/pii/S1877050920307754/pdf?md5=73bd1a6161202e2c412c28c92e2b4e4e&pid=1-s2.0-S1877050920307754-main.pdf>

Ahlawat, S., & Choudhary, A. (2020). Hybrid CNN-SVM classifier for handwritten digit recognition. *Procedia Computer Science*, *167*, 2554-2560.

Ahlawat, S., Choudhary, A., Nayyar, A., Singh, S., & Yoon, B. (2020). Improved handwritten digit recognition using convolutional neural networks (CNN). *Sensors*, *20*(12), 3344. <https://www.mdpi.com/1424-8220/20/12/3344/pdf>

Ahmed, S. S., Mehmood, Z., Awan, I. A., & Yousaf, R. M. (2023). A Novel Technique for Handwritten Digit Recognition Using Deep Learning. *Journal of Sensors*, *2023*.

Albattah, W., & Albahli, S. (2022). Intelligent arabic handwriting recognition using different standalone and hybrid CNN architectures. *Applied Sciences*, *12*(19), 10155.

Ali, S., Li, J., Pei, Y., Aslam, M. S., Shaukat, Z., & Azeem, M. (2020). An effective and improved CNN-ELM classifier for handwritten digits recognition and classification. *Symmetry*, *12*(10), 1742. <https://www.mdpi.com/2073-8994/12/10/1742/pdf>

Aly, S., & Almotairi, S. (2020). Deep convolutional self-organizing map network for robust handwritten digit recognition. *IEEE Access*, *8*, 107035-107045. <https://ieeexplore.ieee.org/iel7/6287639/8948470/09110900.pdf>

Krzysztof, K., Mateusz, S., Marek, S., & Rafał, R. (2021). Applying a quantum annealing based restricted Boltzmann machine for mnist handwritten digit classification. *CMST*, *27*(3), 99-107. <https://cmst.eu/wp-content/uploads/files/10.12921_cmst.2021.0000011_KUROWSKI.pdf>

Krzysztof, K., Mateusz, S., Marek, S., & Rafał, R. (2021). Applying a quantum annealing based restricted Boltzmann machine for mnist handwritten digit classification. *CMST*, *27*(3), 99-107. <https://www.researchgate.net/profile/Murat-Ucar-5/publication/339659155_Applying_Capsule_Network_on_Kannada-MNIST_Handwritten_Digit_Dataset/links/5e5e751392851cefa1d73062/Applying-Capsule-Network-on-Kannada-MNIST-Handwritten-Digit-Dataset.pdf>

Ngu, H. C. V., & Lee, K. M. (2022). Effective conversion of a convolutional neural network into a spiking neural network for image recognition tasks. *Applied Sciences*, *12*(11), 5749.

Pashine, S., Dixit, R., & Kushwah, R. (2021). Handwritten digit recognition using machine and deep learning algorithms. *arXiv preprint arXiv:2106.12614*. <https://arxiv.org/pdf/2106.12614>

Plantin, J.C., 2019. Data cleaners for pristine datasets: Visibility and invisibility of data processors in social science. *Science, Technology, & Human Values*, *44*(1), pp.52-73. <https://www.mdpi.com/2220-9964/9/4/227/pdf>

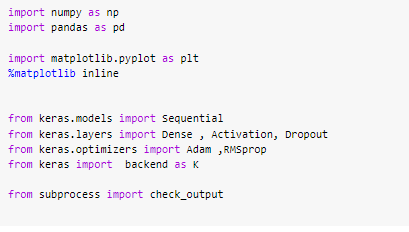
Prabhu, V. U. (2019). Kannada-MNIST: A new handwritten digits dataset for the Kannada language. *arXiv preprint arXiv:1908.01242*. <https://arxiv.org/pdf/1908.01242>

Sanjar, K., Bekhzod, O., Kim, J., Paul, A., & Kim, J. (2020). Missing data imputation for geolocation-based price prediction using KNN–MCF method. *ISPRS International Journal of Geo-Information*, *9*(4), 227. https://www.mdpi.com/2220-9964/9/4/227/pdf

Seng, L. M., Chiang, B. B. C., Salam, Z. A. A., Tan, G. Y., & Chai, H. T. (2021). MNIST Handwritten Digit Recognition with Different CNN Architectures. *Journal of Applied Technology and Innovation (e-ISSN: 2600-7304)*, *5*(1), 7. <https://jati.sites.apiit.edu.my/files/2021/01/MNIST-Handwritten-Digit-Recognition-with-Different-CNN-Architectures.pdf>

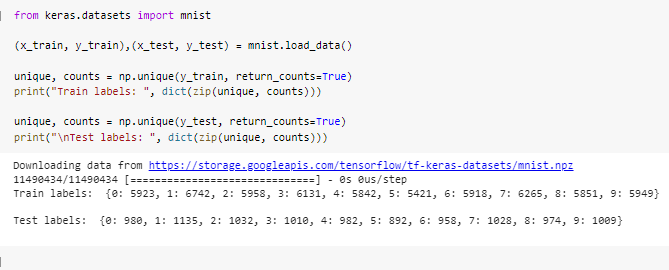
Xin, M., & Wang, Y. (2019). Research on image classification model based on deep convolution neural network. *EURASIP Journal on Image and Video Processing*, *2019*, 1-11.

# Appendix 1

****

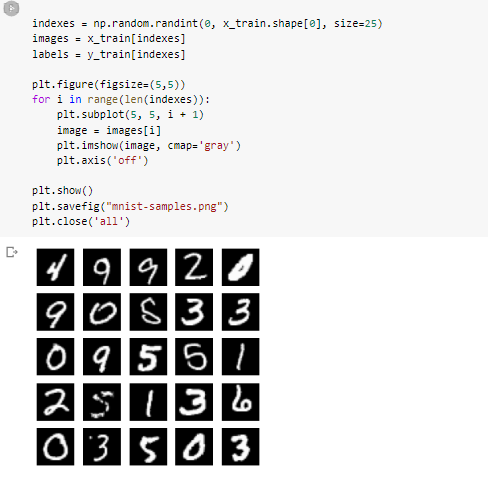
**Figure 17: Importing the Python libraries**

(*Source:* Self-Created)

****

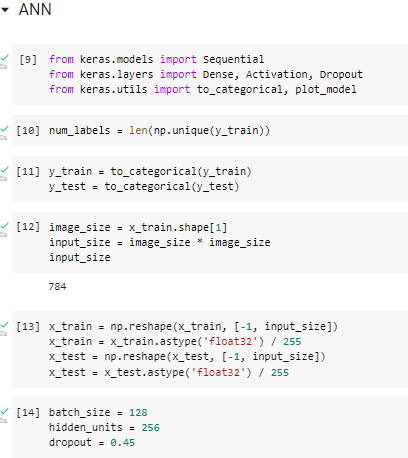
**Figure 18: Train and test**

(*Source:* Self-Created)

****

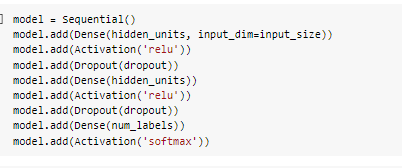
**Figure 19: Train and test**

(*Source:* Self-Created)



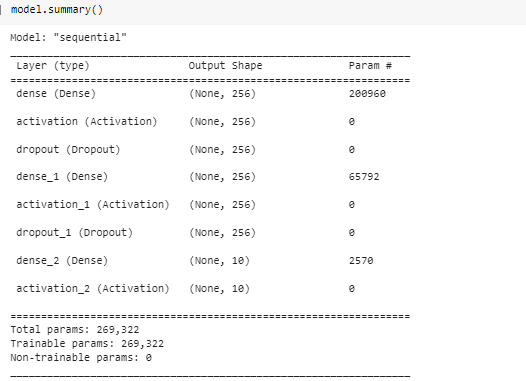
**Figure 20: ANN implementations**

(*Source:* Self-Created)



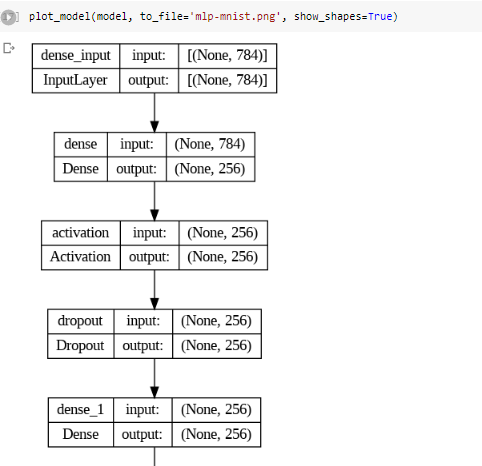
**Figure 21: Model creations**

(*Source:* Self-Created)



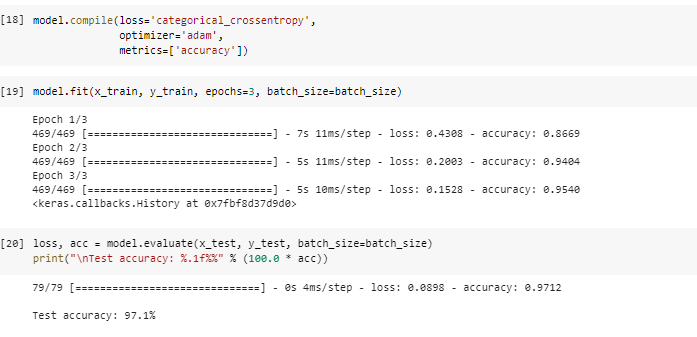
**Figure 22: Model summary**

(*Source:* Self-Created)



**Figure 23: Plot model**

(*Source:* Self-Created)



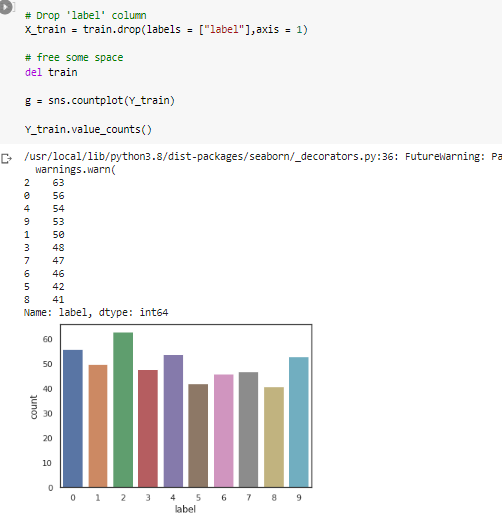
**Figure 24: Accuracy test**

(*Source:* Self-Created)



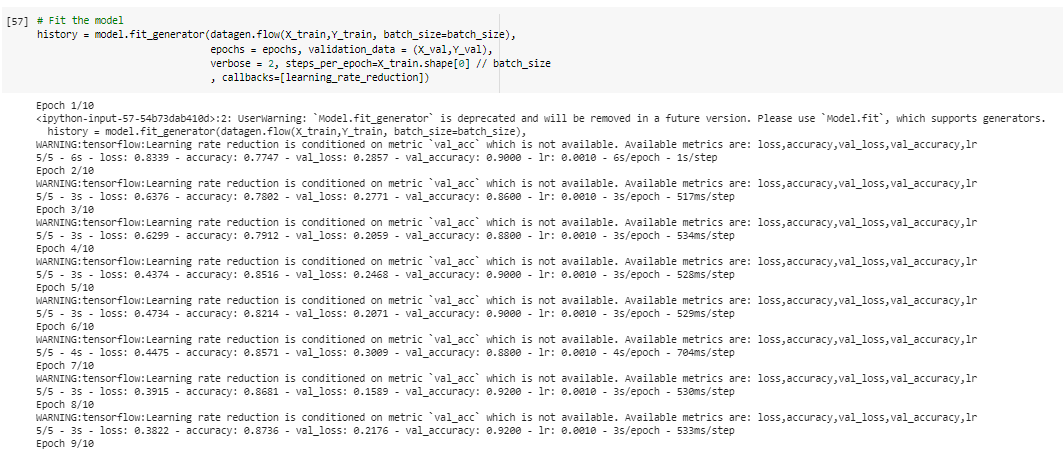
**Figure 25: CNN model**

(*Source:* Self-Created)



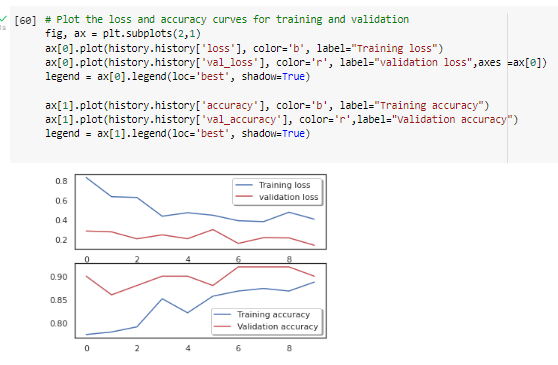
**Figure 26: Label model**

(*Source:* Self-Created)



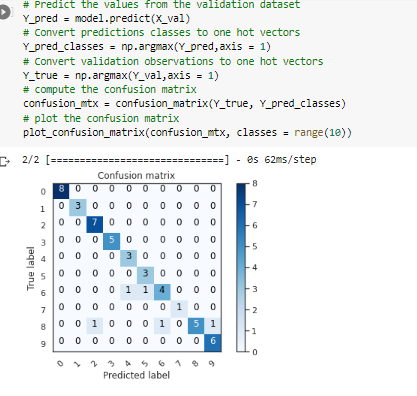
**Figure 27: Epoch test**

(*Source:* Self-Created)



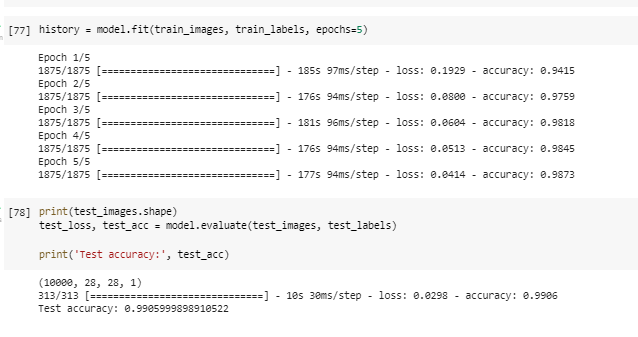
**Figure 28: Loss and accuracy plot**

(*Source:* Self-Created)



**Figure 29: Confusion matrix**

(*Source:* Self-Created)



**Figure 30: Accuracy test**

(*Source:* Self-Created)

# Appendix 2

| **import numpy as np**  **import pandas as pd**  **import matplotlib.pyplot as plt**  **%matplotlib inline**  **from keras.models import Sequential**  **from keras.layers import Dense , Activation, Dropout**  **from keras.optimizers import Adam ,RMSprop**  **from keras import backend as K**  **from subprocess import check\_output**  **from keras.datasets import mnist**  **(x\_train, y\_train),(x\_test, y\_test) = mnist.load\_data()**  **unique, counts = np.unique(y\_train, return\_counts=True)**  **print("Train labels: ", dict(zip(unique, counts)))**  **unique, counts = np.unique(y\_test, return\_counts=True)**  **print("\nTest labels: ", dict(zip(unique, counts)))**  **indexes = np.random.randint(0, x\_train.shape[0], size=25)**  **images = x\_train[indexes]**  **labels = y\_train[indexes]**  **plt.figure(figsize=(5,5))**  **for i in range(len(indexes)):**  **plt.subplot(5, 5, i + 1)**  **image = images[i]**  **plt.imshow(image, cmap='gray')**  **plt.axis('off')**    **plt.show()**  **plt.savefig("mnist-samples.png")**  **plt.close('all')**  **from keras.models import Sequential**  **from keras.layers import Dense, Activation, Dropout**  **from keras.utils import to\_categorical, plot\_model**  **num\_labels = len(np.unique(y\_train))**  **y\_train = to\_categorical(y\_train)**  **y\_test = to\_categorical(y\_test)**  **image\_size = x\_train.shape[1]**  **input\_size = image\_size \* image\_size**  **input\_size**  **x\_train = np.reshape(x\_train, [-1, input\_size])**  **x\_train = x\_train.astype('float32') / 255**  **x\_test = np.reshape(x\_test, [-1, input\_size])**  **x\_test = x\_test.astype('float32') / 255**  **batch\_size = 128**  **hidden\_units = 256**  **dropout = 0.45**  **model = Sequential()**  **model.add(Dense(hidden\_units, input\_dim=input\_size))**  **model.add(Activation('relu'))**  **model.add(Dropout(dropout))**  **model.add(Dense(hidden\_units))**  **model.add(Activation('relu'))**  **model.add(Dropout(dropout))**  **model.add(Dense(num\_labels))**  **model.add(Activation('softmax'))**  **model.summary()**  **plot\_model(model, to\_file='mlp-mnist.png', show\_shapes=True)**  **model.compile(loss='categorical\_crossentropy',**  **optimizer='adam',**  **metrics=['accuracy'])**  **model.fit(x\_train, y\_train, epochs=3, batch\_size=batch\_size)**  **loss, acc = model.evaluate(x\_test, y\_test, batch\_size=batch\_size)**  **print("\nTest accuracy: %.1f%%" % (100.0 \* acc))**  **from keras.regularizers import l2**  **model.add(Dense(hidden\_units,**  **kernel\_regularizer=l2(0.001),**  **input\_dim=input\_size))**  **import pandas as pd**  **import numpy as np**  **import matplotlib.pyplot as plt**  **import matplotlib.image as mpimg**  **import seaborn as sns**  **%matplotlib inline**  **np.random.seed(2)**  **from sklearn.model\_selection import train\_test\_split**  **from sklearn.metrics import confusion\_matrix**  **import itertools**  **from keras.utils.np\_utils import to\_categorical # convert to one-hot-encoding**  **from keras.models import Sequential**  **from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D**  **from keras.optimizers import RMSprop**  **from keras.preprocessing.image import ImageDataGenerator**  **from keras.callbacks import ReduceLROnPlateau**  **sns.set(style='white', context='notebook', palette='deep')**  **# Load the data**  **train = pd.read\_csv("train.csv")**  **test = pd.read\_csv("test.csv")**  **Y\_train = train["label"]**  **# Drop 'label' column**  **X\_train = train.drop(labels = ["label"],axis = 1)**  **# free some space**  **del train**  **g = sns.countplot(Y\_train)**  **Y\_train.value\_counts()**  **# Check the data**  **X\_train.isnull().any().describe()**  **# Normalize the data**  **X\_train = X\_train / 255.0**  **test = test / 255.0**  **# Reshape image in 3 dimensions (height = 28px, width = 28px , canal = 1)**  **X\_train = X\_train.values.reshape(-1,28,28,1)**  **test = test.values.reshape(-1,28,28,1)**  **# Encode labels to one hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0])**  **Y\_train = to\_categorical(Y\_train, num\_classes = 10)**  **# Set the random seed**  **random\_seed = 2**  **# Split the train and the validation set for the fitting**  **X\_train, X\_val, Y\_train, Y\_val = train\_test\_split(X\_train, Y\_train, test\_size = 0.1, random\_state=random\_seed)**  **# Some examples**  **g = plt.imshow(X\_train[0][:,:,0])**  **# Set the CNN model**  **# my CNN architechture is In -> [[Conv2D->relu]\*2 -> MaxPool2D -> Dropout]\*2 -> Flatten -> Dense -> Dropout -> Out**  **model = Sequential()**  **model.add(Conv2D(filters = 32, kernel\_size = (5,5),padding = 'Same',**  **activation ='relu', input\_shape = (28,28,1)))**  **model.add(Conv2D(filters = 32, kernel\_size = (5,5),padding = 'Same',**  **activation ='relu'))**  **model.add(MaxPool2D(pool\_size=(2,2)))**  **model.add(Dropout(0.25))**  **model.add(Conv2D(filters = 64, kernel\_size = (3,3),padding = 'Same',**  **activation ='relu'))**  **model.add(Conv2D(filters = 64, kernel\_size = (3,3),padding = 'Same',**  **activation ='relu'))**  **model.add(MaxPool2D(pool\_size=(2,2), strides=(2,2)))**  **model.add(Dropout(0.25))**  **model.add(Flatten())**  **model.add(Dense(256, activation = "relu"))**  **model.add(Dropout(0.5))**  **model.add(Dense(10, activation = "softmax"))**  **# Define the optimizer**  **optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)**  **# Compile the model**  **model.compile(optimizer = optimizer , loss = "categorical\_crossentropy", metrics=["accuracy"])**  **# Set a learning rate annealer**  **learning\_rate\_reduction = ReduceLROnPlateau(monitor='val\_acc',**  **patience=3,**  **verbose=1,**  **factor=0.5,**  **min\_lr=0.00001)**  **epochs = 10 # Turn epochs to 30 to get 0.9967 accuracy**  **batch\_size = 86**  **# Without data augmentation i obtained an accuracy of 0.98114**  **history = model.fit(X\_train, Y\_train, batch\_size = batch\_size, epochs = epochs,**  **validation\_data = (X\_val, Y\_val), verbose = 2)**  **# With data augmentation to prevent overfitting (accuracy 0.99286)**  **datagen = ImageDataGenerator(**  **featurewise\_center=False, # set input mean to 0 over the dataset**  **samplewise\_center=False, # set each sample mean to 0**  **featurewise\_std\_normalization=False, # divide inputs by std of the dataset**  **samplewise\_std\_normalization=False, # divide each input by its std**  **zca\_whitening=False, # apply ZCA whitening**  **rotation\_range=10, # randomly rotate images in the range (degrees, 0 to 180)**  **zoom\_range = 0.1, # Randomly zoom image**  **width\_shift\_range=0.1, # randomly shift images horizontally (fraction of total width)**  **height\_shift\_range=0.1, # randomly shift images vertically (fraction of total height)**  **horizontal\_flip=False, # randomly flip images**  **vertical\_flip=False) # randomly flip images**  **datagen.fit(X\_train)**  **# Fit the model**  **history = model.fit\_generator(datagen.flow(X\_train,Y\_train, batch\_size=batch\_size),**  **epochs = epochs, validation\_data = (X\_val,Y\_val),**  **verbose = 2, steps\_per\_epoch=X\_train.shape[0] // batch\_size**  **, callbacks=[learning\_rate\_reduction])**  **# Plot the loss and accuracy curves for training and validation**  **fig, ax = plt.subplots(2,1)**  **ax[0].plot(history.history['loss'], color='b', label="Training loss")**  **ax[0].plot(history.history['val\_loss'], color='r', label="validation loss",axes =ax[0])**  **legend = ax[0].legend(loc='best', shadow=True)**  **ax[1].plot(history.history['accuracy'], color='b', label="Training accuracy")**  **ax[1].plot(history.history['val\_accuracy'], color='r',label="Validation accuracy")**  **legend = ax[1].legend(loc='best', shadow=True)**  **# Look at confusion matrix**  **def plot\_confusion\_matrix(cm, classes,**  **normalize=False,**  **title='Confusion matrix',**  **cmap=plt.cm.Blues):**  **"""**  **This function prints and plots the confusion matrix.**  **Normalization can be applied by setting `normalize=True`.**  **"""**  **plt.imshow(cm, interpolation='nearest', cmap=cmap)**  **plt.title(title)**  **plt.colorbar()**  **tick\_marks = np.arange(len(classes))**  **plt.xticks(tick\_marks, classes, rotation=45)**  **plt.yticks(tick\_marks, classes)**  **if normalize:**  **cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]**  **thresh = cm.max() / 2.**  **for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):**  **plt.text(j, i, cm[i, j],**  **horizontalalignment="center",**  **color="white" if cm[i, j] > thresh else "black")**  **plt.tight\_layout()**  **plt.ylabel('True label')**  **plt.xlabel('Predicted label')**  **# Predict the values from the validation dataset**  **Y\_pred = model.predict(X\_val)**  **# Convert predictions classes to one hot vectors**  **Y\_pred\_classes = np.argmax(Y\_pred,axis = 1)**  **# Convert validation observations to one hot vectors**  **Y\_true = np.argmax(Y\_val,axis = 1)**  **# compute the confusion matrix**  **confusion\_mtx = confusion\_matrix(Y\_true, Y\_pred\_classes)**  **# plot the confusion matrix**  **plot\_confusion\_matrix(confusion\_mtx, classes = range(10))**  **# TensorFlow and tf.keras**  **import tensorflow as tf**  **from tensorflow import keras**  **from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense**  **# Commonly used modules**  **import numpy as np**  **import os**  **import sys**  **# Images, plots, display, and visualization**  **import matplotlib.pyplot as plt**  **import pandas as pd**  **import seaborn as sns**  **import cv2**  **import IPython**  **from six.moves import urllib**  **print(tf.\_\_version\_\_)**  **# Set common constants**  **this\_repo\_url = 'https://github.com/arunkumarramanan/mit-deep-learning/raw/master/'**  **this\_tutorial\_url = this\_repo\_url + 'tutorial\_deep\_learning\_basics'**  **(train\_images, train\_labels), (test\_images, test\_labels) = keras.datasets.mnist.load\_data()**  **# reshape images to specify that it's a single channel**  **train\_images = train\_images.reshape(train\_images.shape[0], 28, 28, 1)**  **test\_images = test\_images.reshape(test\_images.shape[0], 28, 28, 1)**  **def preprocess\_images(imgs): # should work for both a single image and multiple images**  **sample\_img = imgs if len(imgs.shape) == 2 else imgs[0]**  **assert sample\_img.shape in [(28, 28, 1), (28, 28)], sample\_img.shape # make sure images are 28x28 and single-channel (grayscale)**  **return imgs / 255.0**  **train\_images = preprocess\_images(train\_images)**  **test\_images = preprocess\_images(test\_images)**  **plt.figure(figsize=(10,2))**  **for i in range(5):**  **plt.subplot(1,5,i+1)**  **plt.xticks([])**  **plt.yticks([])a**  **plt.grid(False)**  **plt.imshow(train\_images[i].reshape(28, 28), cmap=plt.cm.binary)**  **plt.xlabel(train\_labels[i])**  **model = keras.Sequential()**  **# 32 convolution filters used each of size 3x3**  **model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1)))**  **# 64 convolution filters used each of size 3x3**  **model.add(Conv2D(64, (3, 3), activation='relu'))**  **# choose the best features via pooling**  **model.add(MaxPooling2D(pool\_size=(2, 2)))**  **# randomly turn neurons on and off to improve convergence**  **model.add(Dropout(0.25))**  **# flatten since too many dimensions, we only want a classification output**  **model.add(Flatten())**  **# fully connected to get all relevant data**  **model.add(Dense(128, activation='relu'))**  **# one more dropout**  **model.add(Dropout(0.5))**  **# output a softmax to squash the matrix into output probabilities**  **model.add(Dense(10, activation='softmax'))**  **model.compile(optimizer=tf.optimizers.Adam(),**  **loss='sparse\_categorical\_crossentropy',**  **metrics=['accuracy'])**  **print(test\_images.shape)**  **test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)**  **print('Test accuracy:', test\_acc)** |
| --- |