**1. MNIST DATASET USING CONVOLUTIONAL NEURAL NETWORK**

**1.1- INTRODUCTION**

**Artificial Intelligence**

Artificial Intelligence is an approach to make a computer, a robot, or a product to think how smart human think. AI is a study of how human brain think, learn, decide and work, when it tries to solve problems. And finally this study outputs intelligent software systems. The aim of AI is to improve computer functions which are related to human knowledge, for example, reasoning, learning, and problem-solving.The intelligence is intangible. It is composed of

* Reasoning
* Learning
* Problem Solving
* Perception
* Linguistic Intelligence

The objectives of AI research are reasoning, knowledge representation, planning, learning, natural language processing, realization, and ability to move and manipulate objects. There are long-term goals in the general intelligence sector. Approaches include statistical methods, computational intelligence, and traditional coding AI.

During the AI research related to search and mathematical optimization, artificial neural networks and methods based on statistics, probability, and economics, we use many tools. Computer science attracts AI in the field of science, mathematics, psychology, linguistics, philosophy and so on.

**Python Programming**

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

**1.2 OBJECTIVES OF RESEARCH**

Train a computer system to first understand what are handwritten images of digits then test the system to predict new handwritten images correctly. We will feed our system with MNIST dataset and test it with random images to check the accuracy of its predictions.

**1.3 PROBLEM STATEMENT**

For humans, identifying numbers in a picture is extremely simple, but how do you train the machine to recognize digits in images ? Convolutional Neural Networks (CNN) can solve this problem. In this a convolutional neural network has been trained by identifying digits in MNIST handwritten digital database to predict exactly what the numbers in the picture are. Obviously, human beings can perceive that there is a hierarchy or conceptual structure in the image, but the machine does not, for example the trained neural network is inconvenient to deal with special changes in a position of numbers in digital pictures. Exactly put, no matter what the environment of the image (image background) is, it is unchallenging for human beings to judge whether there is such a figure in the image and it is unnecessary to repeat the learning training.

**2. Review of literature**

In recent years, the development of neural networks has been extremely rapid in the field of pattern recognition system. We use a common pattern recognition technique convolution to improve this technique. Then we will introduce the entire project from the data sets used, the neural network we built, and the CNN improvement methods used. The neural network used before cannot recognize images without repeated training. It reckons that the ‘number’ appears at different places in a picture is not the same number. The neural network cannot understand the concept of whether an object appears in a different position in the picture is different objects or the same objects. That means the neural network must re-learn to identify various objects in every possible position. How to let neural network understand the concept of ‘translation invariance’, that means ‘0’ is ‘0’, no matter where it appears in the images. In this report, I used a convoluted solution to achieve the goal.

**3. Data Collection**

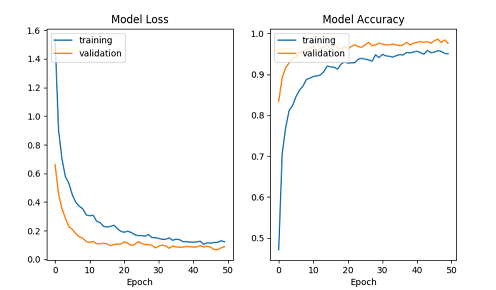
The MNIST dataset is imported from the keras.datasets library . The MNIST database of handwritten digits contains 60,000 training examples and 10,000 testing examples, which are 28 \* 28 images. All of digits have already been size normalized and preprocessed and formatted. The four files provided on the website are used in the training and testing for neural networks. In the process of loading the data set can be directly called from the MNIST database, but due to the requirements of the assignment, I downloaded these image files from the website that provides the data set. Because downloading browsers may unzip these image collection files without your attention, this operation may cause the downloaded files to be larger than previously mentioned. Thus, if you need to see some problems with the original image set or data set, you can view the original site of the data set via the link provided in the reference section of the paper. Due to the use of Python's own data set, simplifying the section on data preprocessing in the code. The images are all centered in 28 \* 28 field.



**4. Methodology**

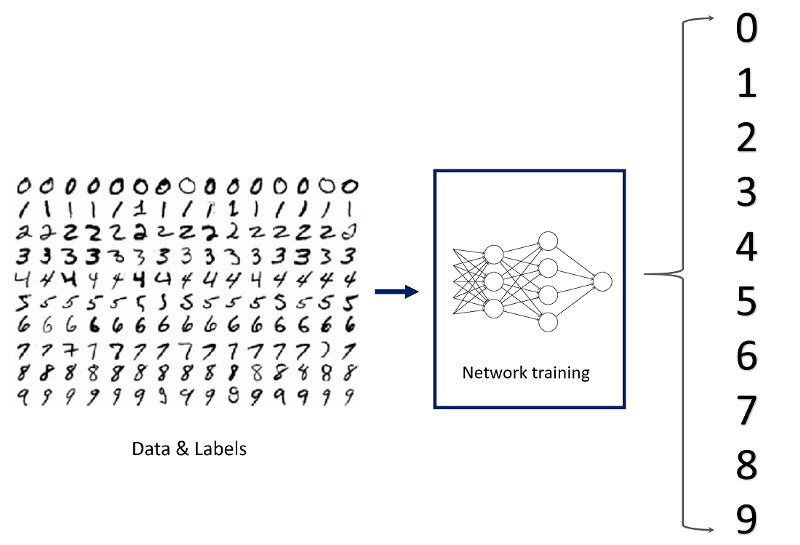
**4.1Exploratory Data Analysis**

**4.1.1 Figures and Tables**



**4.2 Data Modelling**

**Convolutional Neural Networks**

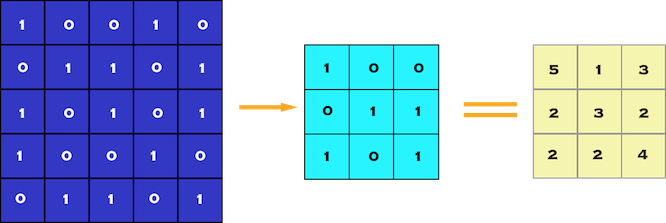
****

**Layers in a CNN**

We are capable of using many different layers in a convolutional neural network. However, convolution, pooling, and fully connect layers are the most important ones. Therefore, I will quickly introduce these layers before implementing them.

#### Convolutional Layers

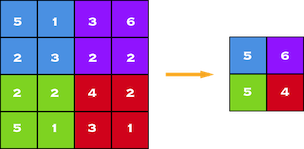
Convolutional layer is the very first layer where we extract features from the images in our datasets. Due to the fact that pixels are only related with the adjacent and close pixels, convolution allows us to preserve the relationship between different parts of an image. Convolution is basically filtering the image with a smaller pixel filter to decrease the size of the image without loosing the relationship between pixels. When we apply convolution to 5x5 image by using a 3x3 filter with 1x1 stride (1 pixel shift at each step). We will end up having a 3x3 output (64% decrease in complexity).



Convolution of 5 x 5 pixel image with 3 x 3 pixel filter (stride = 1 x 1 pixel)

**Pooling Layer**

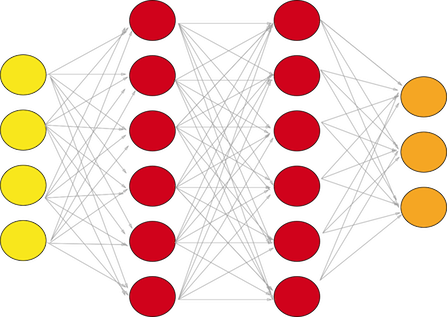
When constructing CNNs, it is common to insert pooling layers after each convolution layer to reduce the spatial size of the representation to reduce the parameter counts which reduces the computational complexity. In addition, pooling layers also helps with the overfitting problem. Basically we select a pooling size to reduce the amount of the parameters by selecting the maximum, average, or sum values inside these pixels. Max Pooling, one of the most common pooling techniques, may be demonstrated as follows:



Max Pooling by 2 x 2

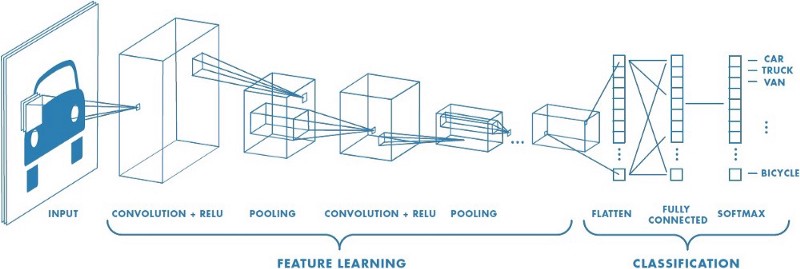
#### A Set of Fully Connected Layers

A fully connected network is our RegularNet where each parameter is linked to one another to determine the true relation and effect of each parameter on the labels. Since our time-space complexity is vastly reduced thanks to convolution and pooling layers, we can construct a fully connected network in the end to classify our images. A set of fully connected layers looks like this:



A fully connected layer with two hidden layers

Now that you have some idea about the individual layers that we will use, I think it is time to share an overview look of a complete convolutional neural network.



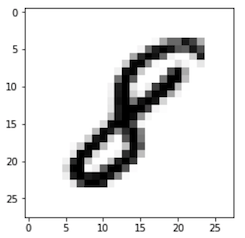
A Example for Convolutional Neural Network

And now that you have an idea of convolutional neural network that you can build for image classification, we can get the most cliche dataset for classification: MNIST dataset, which stands for Modified National Institute of Standards and Technology database. It is a large database of handwritten digits that is commonly used for training various image processing systems.

### Loading the Mnist Data

The MNIST dataset is one of the most common datasets used for image classification and accessible from many different sources. In fact, even Tensorflow and Keras allow us to import and download the MNIST dataset directly from their API. Therefore, I will start with the following two lines to import tensorflow and MNIST dataset under the Keras API.

When we run the code above, we will get the greyscale visualization of the RGB codes as shown below.



A visualization of the sample image at index 7777

We also need to know the shape of the dataset to channel it to the convolutional neural network. Therefore, I will use the “shape” attribute of numpy array with the following code:

You will get (60000, 28, 28). As you might have guessed 60000 represents the number of images in the train dataset and (28, 28) represents the size of the image: 28 x 28 pixel.

### Reshaping and Normalizing the Images

To be able to use the dataset in Keras API, we need 4-dims numpy arrays. However, as we see above, our array is 3-dims. In addition, we must normalize our data as it is always required in neural network models. We can achieve this by dividing the RGB codes to 255 (which is the maximum RGB code minus the minimum RGB code).

### Building the Convolutional Neural Network

We will build our model by using high level Keras API which uses either TensorFlow or Theano on the backend. I would like to mention that there are several high level TensorFlow APIs such as Layers, Keras, and Estimators which helps us create neural networks with high level knowledge. However, this may lead to confusion since they all varies in their implementation structure. Therefore, if you see completely different codes for the same neural network although they all use tensorflow, this is why. I will use the most straightforward API which is Keras. Therefore, I will import the Sequential Model from Keras and add Conv2D, MaxPooling, Flatten, Dropout, and Dense layers. I have already talked about Conv2D, Maxpooling, and Dense layers. In addition, Dropout layers fight with the overfitting by disregarding some of the neurons while training while Flatten layers flatten 2D arrays to 1D array before building the fully connected layers.

We may experiment with any number for the first Dense layer; however, the final Dense layer must have 10 neurons since we have 10 number classes (0, 1, 2, …, 9). You may always experiment with kernel size, pool size, activation functions, dropout rate, and number of neurons in the first Dense layer to get a better result.

### Compiling and Fitting the Model

we created an non-optimized empty CNN. Now it is time to set an optimizer with a given loss function which uses a metric. Then, we can fit the model by using our train data. We will use the following code for these tasks:

You can experiment with the optimizer, loss function, metrics, and epochs. However, I can say that adam optimizer is usually out-performs the other optimizers. I am not sure if you can actually change the loss function for multi-class classification. Feel free to experiment and comment below. Epoch number might seem a bit small. However, you will reach to 98–99% test accuracy. Since the MNIST dataset does not require heavy computing power, you may easily experiment with the epoch number as well.

### Evaluating the Model

Finally, you may evaluate the trained model with x\_test and y\_test using one line of code:

The results are pretty good for 10 epochs and for such simple model.

https://cdn-images-1.medium.com/max/800/1*5wiWlY_cHiqo51fySbQpHA.png

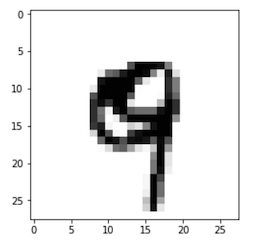
Evaluation shows 98.5% accuracy on test set!

We achieved 98.5% accuracy with such basic model. To be frank, in many image classification cases (e.g. for autonomous cars), we cannot even tolerate 0.1% error since, as an analogy, it will cause 1 accident in 1000 cases. However, for our first model, I would say the result is still pretty good.

**Prediction**

We can also make individual predictions by using predict() .We can perform prediction on test data and we load the data into Y\_predict variable , we see at what index what image is present And then we display the image in the training test and compare it if there is similarity if same then our prediction is correct and the model is working as below

Our model will classify the image as a ‘9’ and h is the visual of the image:



Our model correctly classifies this image as a 9 (Nine)

Although it is not really a good hand writing of the number 9, our model was able to classify it as 9.

**Conclusion**

In this we used Convolutional Neural Networks (CNN) for image classification using images form hand written MNIST data sets. This data sets used both and training and testing purpose using CNN. It provides the accuracy rate 98%. Images used in the training purpose are small and Grayscale images. Stacking the model with more layers and training the network with more image data using clusters of GPUs will provide more accurate results of classification of images. The future enhancement will focus on classifying the colored images of large size and its very useful for image segmentation process.