Neural Networks for Optics Pattern Recognition

Ajay Mehra
Department of Physics
Dyal Singh College, University of Delhi
Delhi, India
ajaymehra@duck.com

***Abstract*—The aim of this research was to develop a pattern recognition algorithm for predicting simple image patterns using actual simulated data from Experimental Hall C at Jefferson Lab. Using Keras library, a Convolution Neural Network(CNN) model in Python was designed with certain hidden layers like Colvolution, Maxpooling and Activation Layer, and each layer is aimed to extract different features from the images. While training it against 186 images of simulation data, the model worked very efficiently and reached an accuracy of 100% with a loss of 0.38. The models' predictive power was then tested with a 60 images, in which it achieved 80 % accuracy.**

***Index Terms*—Neural networks, Convolution, Optics pattern, Deep Learning,**

I. Introduction

"I visualize a time when we will be to robots what dogs are to humans, and I'm rooting for the machines." —*Claude Shannon*

Artificial intelligence, or AI, is a field of computer science that attempts to simulate characteristics of human intelligence or senses. These include learning, reasoning, and adapting. Machine Learning(ML) is a tool which AI uses to learn. ML can be explained as automating and improving the learning process of computers based on their experiences without being actually programmed i.e. without any human assistance. Deep learning is a subset of machine learning, which is essentially a neural network with some layers. These neural networks attempt to simulate the behavior of the human brain.

A neural network is a system that learns how to make predictions by following these steps:

1) Taking the input data
2) Making a prediction
3) Comparing the prediction to the desired output
4) Adjusting its internal state to predict correctly the next time

The neural network uses weights and biases in building these layers and the aim is to change and tune them such to minimise the prediction error. Each neuron takes inputs and have some values of weights and biases, and it gives some output.

In this Research paper, the implementation of Neural Networks in case of Hall C optics data of *Jefferson lab* is summarised.

II. Methodology

In this research, we analyze a total of 186 distinct simulated optics patterns from the *Super High Momentum Spectrometer* (SHMS) at Hall C of Jefferson Lab. There were six different optics correlations x_{fp} vs. y_{fp} , x_{fp} vs. $y^{'}_{fp}$, x_{fp} vs. $x^{'}_{fp}$, $x^{'}_{fp}$ vs. y_{fp} , $x^{'}_{fp}$ vs. $y^{'}_{fp}$, $y^{'}_{fp}$ vs. y_{fp} each pattern had 31 optics images with varying optics tunes [Q1, Q2, Q3], corresponding to the spectrometer quadrupole magnets, here summarized in Table I:

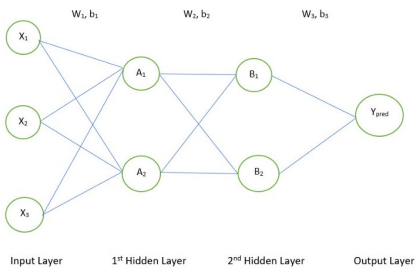

Fig. 1. Schematic representation of a neural network. Note: reprinted from Ref. [1]

Quadrupole Magnet	Range	Stepsize
$Q1$	[0.90, 1.10]	0.02
$Q2$	[0.95, 1.05]	0.01
$Q3$	[0.90, 1.10]	0.02

TABLE I
Quadrupoles Input Data Details

Each of the six 2D SHMS optics pattern correlations were trained separately, using 31 different optics tunes per correlation plot for a total of 186 images. The optics patters for testing the network consisted of only varying Q2 from 0.95 to 1.05 in steps of 0.01, while keeping Q1 and Q3 tunes fixed at unity. To test the neural network after it had been trained, a set of 10 images were used for each 2D optics correlation, where Q1 and Q3 tunes were kept fixed at unity while Q2 was varied from 0.955 to 1.055 in steps of 0.01 for a total of 10 Q2 tunes [2].

The data with specific [Q1,Q2,Q3] tunes were simulated using the standard Hall C simulation program (mc-single-arm) and the raw data output was written to a ROOTfile. A separate ROOT C++ script (make2Doptics.C) was used to form each of the six above mentioned 2D focal plane correlations which were stored in a separate ROOTfile as histogram objects. The 2D histograms were then converted to a 2D pixelated

array and stored in binary format (.h5) via a Python code (save2binary.py) array to be read by the Neural Network using Python Keras. Each optics image used was 200x200 pixels and was passed thorugh each of the hidden layers of the network described in Section 3 of this article.

III. Data Analysis Procedure

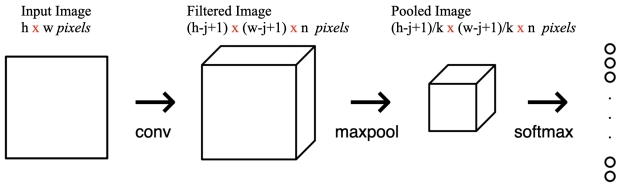

Fig. 2. CNN layers

The Neural Network used in this research consists of 5 layers in total (See Fig.2). The input and output layers, represent the raw input image and output model prediction, respectively, and intermediate hidden layers, *convolutional*, *pooling*, and *activation* layers, each with a specific image analysis task as described in the subsections below (see Ref. [3]).

A. Convolutional Layer

The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are input data, a filter, and a feature map. The feature map will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution.

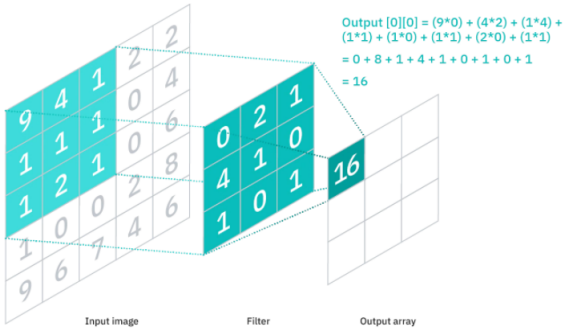

Fig. 3. Convolution Demo [4]

The feature detector is a two-dimensional *(2-D)* array of weights, which represents part of the image. While they can vary in size, In our case, we are using 12 filters of dimension *6x6*. The filter is then applied to an area of the image, and a dot product is calculated between the input pixels and the filter. This dot product is then fed into an output array. Afterwards, the filter shifts by a stride which is 1 in our case, repeating the process until the kernel has swept across the entire image. The final output from the series of dot products from the input and the filter is known as a feature map, activation map, or a convolved feature.

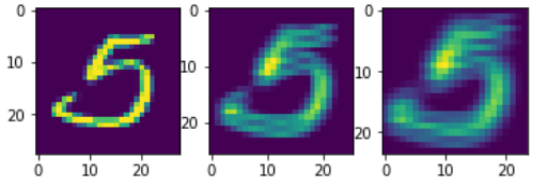

Fig. 4. *Example Figure: Convolution*: 28x28 image is converted to 26x26 to 24x24, with a filter of dimension 3x3 and stride of 1.

B. Pooling Layer

Pooling layers are used for dimensionality reduction, to reduce number of parameters. It is similar to convolutional layer, except its output is determined by some aggregation function.Although, in pooling layers,a lot of information is lost, it reduces complexity and solves the chances of overfitting.

In our case, we used *5x5* MaxPooling, in which filter (5x5) moves across the input, and returns the pixel with largest value to output array.

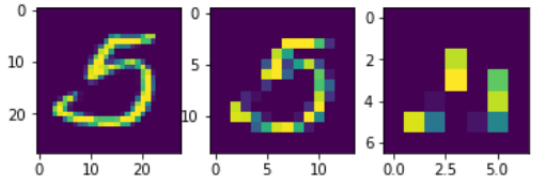

Fig. 5. *Example Figure: Maxpooling*: choosing 2x2 as our poolsize for our 28x28 image results a 14x14 image, and if applied again results into 7x7 image.

C. Activation Layer

This layer aims to classify the images on basis of its features of previous layer. In this layer, all the previous layer nodes connect directly to output nodes. In our case we used *Softmax Layer* [5], which uses Softmax function for its classification.

We can see in Fig 6, the first layer shows outputs from Maxpooling, and it is connected to the second layer via a neural network connection, which has certain number of classes, number of classes depends on the number of outputs we want, and then begins the softmax transformation.

Given a set of numbers, softmax function converts them into probabilities. Softmax performs the following transform on n numbers $x_1, \ldots, x_n$

$$S(x_n) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{1}$$

The reason for converting them into probability spectrum is to consider relative chances of being predicted correct, and how much our class' prediction is true than other classes,

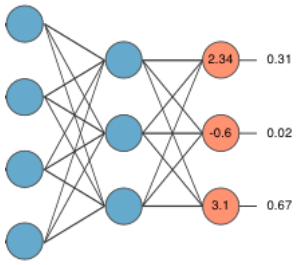

Fig. 6. An example of Softmax transformation in a Neural network. Image from Ref. [6]

which helps in training our model. The output of these layers are always less than 1 and sums up to 1.

Cross entropy loss: It is calculated by taking the logarithm of correct class' probability.

$$L = -\ln(p_c) \tag{2}$$

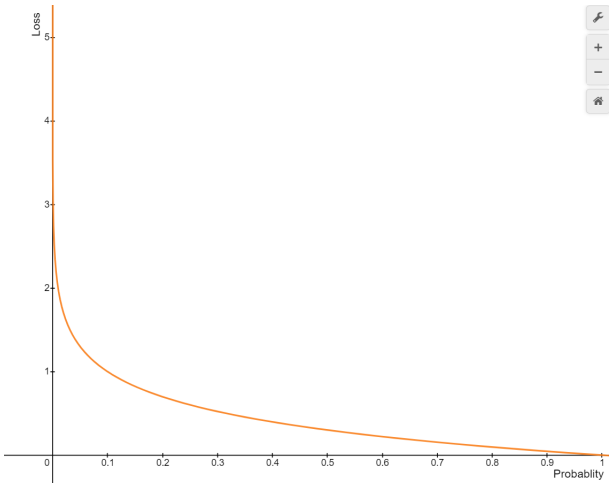

Fig. 7. Probability vs Loss graph [7]

If the probability of the correct class is close to one (means our prediction is better) then the loss is close to zero, but if the probability is close to 0 then penalty will be high and loss will approach infinity.

Backpropagation [8]: In forward phase, the inputs of one layer were passed to next layer and completely through the network, whereas in backward phase gradients are backpropagated through the network and weights are updated.

During the forward phase,the network will store the data like inputs, input size, intermediate values, etc, and each layer will receive a gradient of Loss with respect to output ($\frac{\partial L}{\partial out}$), and will give the gradient of Loss with respect to input ($\frac{\partial L}{\partial in}$). In our case, we are using loss as negative of logarithm of probability of correct class.

Next step was to update the values of weights and biases, we used the stochastic gradient descent method. In SGD method, all the variables are updated using following equation,

$$x \longleftarrow x - \alpha \frac{\partial L}{\partial x} \tag{3}$$

Where α is learning rate and it depends on this constant how fast or slow will our loss converge to zero. Overall, x will converge to that value which will result into minimum loss.

Figure 8 illustrates the transformations on our image, From 200x200x12 to 195x195x12 to 49x49x12 to 1x10.

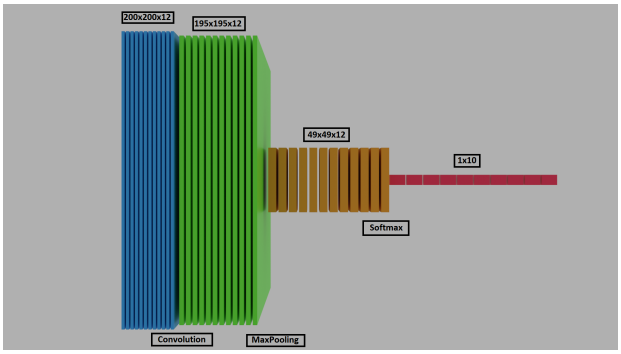

Fig. 8. Layers in our network [9]

IV. Results and Discussion

The purpose of this research was to teach a machine to recognize optics patterns that would otherwise be difficult to distinguish by the "human eye". With the help of Keras API, we were able to train and test a CNN by providing simulated optics data from Jefferson Lab, Hall C. Each of the six 2D optics correlation was trained with 31 optics tunes, and each were able to reach and plateau at an accuracy of 100 %, and a average loss of 0.3787 , in 100 epochs of training. We used 10 test images per each of the six 2D optics correlations, and network was able to correctly predict each of the patterns with average of 80% accuracy. The results of the training are shown in Fig.11 and Fig.12

- *Result: Accuracy vs Epochs*: In training graph 11, we can see that at 100 epochs, all the correlations have achieved 100 % accuracy.
- *Result: Loss vs Epochs*: Similarly, graph 12 tells us that loss is approaching 0.

The results of the test images is summarized in Table II. .

2D optics correlation	Correct predicted patterns	No. patterns	Accuracy
x_{fp} vs. x'_{fp}	8	10	80%
x_{fp} vs. y'_{fp}	8	10	80%
x_{fp} vs. y_{fp}	10	10	100%
x'_{fp} vs. y_{fp}	5	10	50%
x'_{fp} vs. y'_{fp}	10	10	100%
y_{fp} vs. y_{fp}	7	10	70%

TABLE II
Performance of our Neural Network

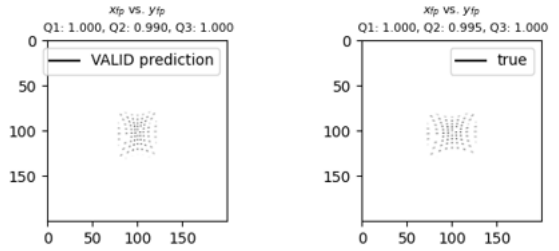

Fig. 9. A sample optics image and its prediction by our model.

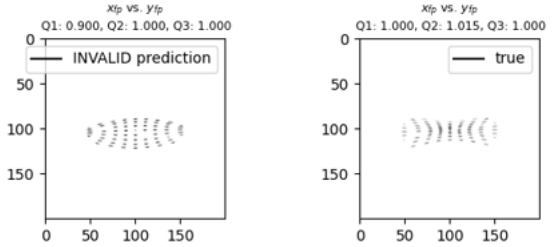

Fig. 10. A sample optics image and its prediction by our model.

Conclusion: Test results are pretty satisfactory, but particularly for $x^{'}_{fp}$ vs. y_{fp} and $y^{'}_{fp}$ vs. y_{fp} optics correlations, we were less accurate, about 50% and 70% respectively. There can be many factors to it, but it can certainly be improved by advancing the training Dataset, which can be done by decreasing the stepsize. Current stepsize (See Table I).

Overall, we are successful in developing a model, which is able to recognise patterns for our optics images with good accuracy and low loss, and with some improvements, it can certainly prove to be very helpful for automating this task of *Optics Pattern Recognition*, which would otherwise be very tedious for humans.

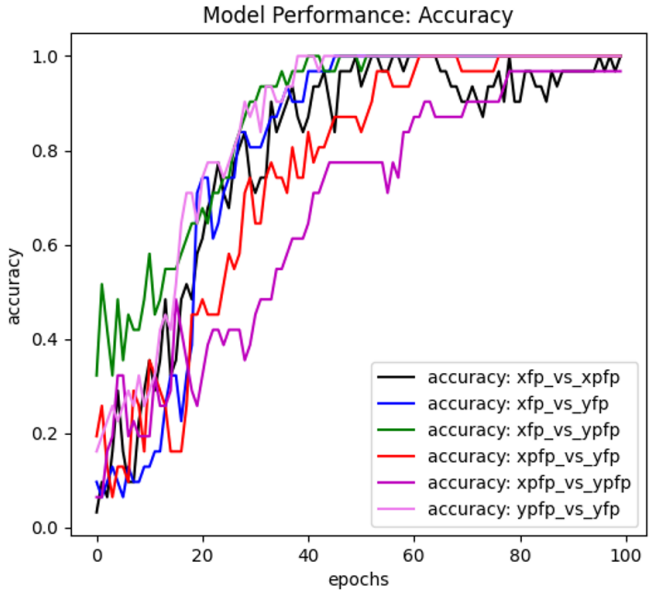

Fig. 11. Accuracy vs epochs plot

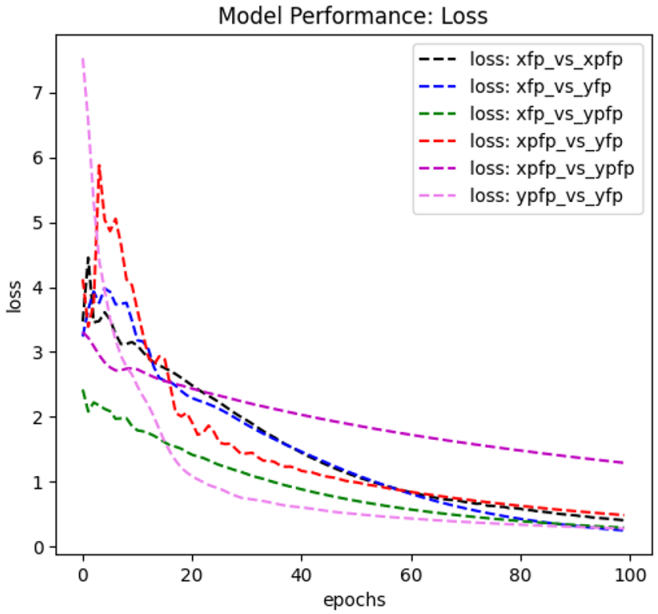

Fig. 12. Loss vs epochs plot

REFERENCES

[1] T. Bhattacharya. (2020) An Introduction to Neural Networks with Implementation from Scratch in Python. (Accessed on 2021-09-15). [Online]. Available: https://towardsdatascience.com/an-introduction-to-neural-networks-with-implementation-from-scratch-using-python-da4b6a45c05b/

[2] C. Yero, Private communication, August 2021.

[3] V. Zhou. (2019) CNNs, Part 1: An Introduction to Convolutional Neural Networks. (Accessed on 2021-08-20). [Online]. Available: https://victorzhou.com/blog/intro-to-cnns-part-1/

[4] I. C. Educaion. (2020) Convolutional Neural Networks. (Accessed on 2021-09-12). [Online]. Available: https://www.ibm.com/cloud/learn/convolutional-neural-networks

[5] V. Zhou. (2019) A Simple Explanation of the Softmax Function. (Accessed on 2021-08-27). [Online]. Available: https://victorzhou.com/blog/softmax/

[6] Z. Singer. (2019) Softmax and Uncertainty. (Accessed on 2021-09-12). [Online]. Available: https://towardsdatascience.com/softmax-and-uncertainty-c8450ea7e064

[7] A. Mehra, Used Desmos graphing Calculator, October 2021. [Online]. Available: https://www.desmos.com/calculator

[8] V. Zhou. (2020) CNNs, Part 2: Training a Convolutional Neural Network. (Accessed on 2021-08-27). [Online]. Available: https://victorzhou.com/blog/intro-to-cnns-part-2/

[9] A. Mehra, Used Blender Software, October 2021. [Online]. Available: https://blender.org