Building a neural network for translating the arabic sign language

Sohaila Kandil[a]

[a] *Egypt Japan university of science and technology,*

Abstract

In this project, I created an Artificial Neural Network (ANN) from scratch to predict digits. The training data consisted of 42000 images of handwritten digits, and the model achieved an accuracy of approximately 85%.

Later, I developed a Convolutional Neural Network (CNN), which is better suited for image training, to build a sign language model. This model was trained using Keras and TensorFlow, using a dataset of 50,000 images representing 32 different Arabic sign letters and it was able to predict the sign language images with accuracy of 95.9%.

Keywords: ANN, CNN, Forward propagation, Backward propagation

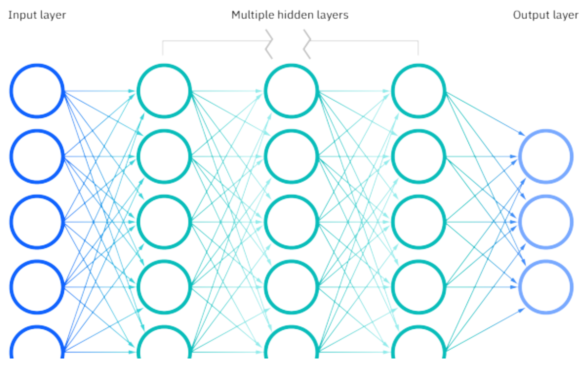


Figure 1: A simple ANN (1)

1. Introduction

1.1. The principles of ANN

Artificial Neural Network (ANN) is a network composed of multiple layers, where each layer contains nodes representing probabilities of patterns in an image. Each layer can be visualized as a vector.

Let's consider Figure 1 to understand this concept better. If we have an image of the number 1, the input layer, which is the first layer, contains the pixel values of all the image's pixels. The first hidden layer consists of nodes representing various patterns that could exist in the images. When we input the pixel values of the digit 1 image, each node in the first layer will receive probabilistic values indicating the likelihood of the pattern represented by that node. These probability values are influenced by the input pixels.

Similarly, each node in the second hidden layer represents the probability of a larger pattern existing in the dataset of digits. The probabilistic values in the first hidden layer will impact the probabilistic values in the second hidden layer. Finally, the last hidden layer comprises 10 nodes, with each node representing the probability of the input image being a specific digit from zero to nine.By visualizing the probabilistic values in the last hidden layer, the node with the highest probability corresponds to our prediction for the digit.

To construct this neural network, we assign weights that connect each layer to the subsequent layer. Two essential operations are involved.

First, we perform forward propagation, where predictions are made in the output layer based on the input layer. It also calculates the error incurred during the prediction.

Second, we execute backward propagation, which adjusts the weights in the network to minimize the error. This step helps in fine-tuning the network's performance.

1.1.1. Forward propagation

Consider the input layer of our neural network represented by the matrix:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

In this example, each column corresponds to the pixel values of one image, indicating that we have three images in total in this specific example. We denote each hidden layer in the network as $z^{[x]}$, with x representing the layer number. Within layer j, the node n represents a probabilistic value. Let the subsequent layer i contain the node m, which is influenced by the node n through the application of some weights and biases. Hence the value of each node in layer i is calculated by summing the products of the nodes in the previous layer, multiplied by their respective weights w_{nm}, such that n represent a node in the layer i and m represents a node in layer j, and adding a specific bias b_j. Hence layer $z^{[i]}$ can be calculated through the following equation:

$$\begin{bmatrix} w^i_{00} & w^i_{01} & w^i_{02} & \cdots & w^i_{0p} \\ w^i_{10} & w^i_{11} & w^i_{12} & \cdots & w^i_{1p} \\ w^i_{20} & w^i_{21} & w^i_{22} & \cdots & w^i_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w^i_{k0} & w^i_{k1} & w^i_{k2} & \cdots & w^i_{kp} \end{bmatrix} \times \begin{bmatrix} z^j_0 \\ z^j_1 \\ z^j_2 \\ \vdots \\ z^j_p \end{bmatrix} + \begin{bmatrix} b^i_0 \\ b^i_1 \\ b^i_2 \\ \vdots \\ b^i_p \end{bmatrix} = \begin{bmatrix} z^i_0 \\ z^i_1 \\ z^i_2 \\ \vdots \\ z^i_k \end{bmatrix} \quad (1)$$

such that p is the number of nodes in layer j, k is the number of nodes in the subsequent layer i, b^i_x is the bias of node x at layer i and z^i_x is the value of the node x at layer i.

For non-binary models, which classify three or more objects, we need the relation between the layers not to be linear. Therefore, after computing the value of each node in layer i, it is necessary to apply an activation function to each node in that layer. Thus, we have

$$a^i = f(z^i) \tag{2}$$

where f represents the activation function, such as the sigmoid or ReLU function, and a^i denotes the hidden layer i after the activation function has been applied.

The next step involves calculating the prediction error. This is done by employing one-hot encoding on the true value. In the context of classifying dogs and cats, we assign the label "cats" the number 0 and "dogs" the number 1. Consequently, we construct a vector with two elements. When the predicted value corresponds to a cat, the first index of the vector is set to one and the second index is set to zero. Conversely, if the predicted value corresponds to a dog, the values are reversed, with the first index being zero and the second index being one.

Using the one-hot encoded vector y, we can quantify the error by subtracting the predicted value from the network from the actual value indicated by the one-hot encoded vector and squaring the result to make the error vector contain all positive values. The following equation calculates the error:

$$l = (a^B - y)^2 \tag{3}$$

such that a^B is the last hidden layers of the network.

1.1.2. Backward propagation

To enhance the neural network's performance, it is crucial to fine-tune the weights and biases linked to each layer within the network. These adjustments in weights and biases should be proportionate to their respective partial derivatives: $\frac{\partial l}{\partial w^n}$ and $\frac{\partial l}{\partial b^n}$. Let's now proceed with the computation of these derivatives.

$$z^n_i = \sum_j \left((w^n_{i,j}) \cdot (a^{n-1}_j)\right) + (b^n_i) \tag{4}$$

Hence from equation 4:

$$(\frac{\partial z^n_i}{\partial b^n_i} = 1) \tag{5}$$

from the chain rule

$$\frac{\partial l}{\partial b^n_i} = \frac{\partial l}{\partial z^n_i} * \frac{\partial z^n_i}{\partial b^n_i} \tag{6}$$

Now from equation 5 and 6 we conclude that:

$$\frac{\partial l}{\partial b^n_i} = \frac{\partial l}{\partial z^n_i} \tag{7}$$

Let $\nabla_x y$ denote the matrix derivative of x with respect to y. then

$$\nabla_{(b^n)} l = \nabla_{(z^n)} l \tag{8}$$

Now that we have obtained $\nabla_{(b^n)} l$ in terms of $\nabla_{(z^n)} l$, we proceed to calculate $\nabla_{(w^n)} l$.

from chain rule:

$$\frac{\partial l}{\partial w^n_{i,j}} = \frac{\partial l}{\partial z^n_i} * \frac{\partial z^n_i}{\partial w^n_{i,j}} \tag{9}$$

and from equation 4:

$$\frac{\partial l}{\partial w^n_{i,j}} = \frac{\partial l}{\partial z^n_i} * a^{n-1}_j \tag{10}$$

and hence from equation 7:

$$\nabla_{(w^n)} l = \nabla_{(z^n)} l \cdot (a^{n-1})^T \tag{11}$$

Now, to calculate $\nabla_{(b^n)} l$ and $\nabla_{(w^n)} l$ in equations 8 and 11, respectively, we need to compute $\nabla_{(z^n)} l$.

According to equation 3:

$$\nabla_{(z^B)} l = 2 \cdot (a^B - y) \tag{12}$$

Here, z^B represents the last hidden layer in the network. With this, we can proceed to calculate $\nabla_{(b^B)} l$ and $\nabla_{(w^B)} l$. However, what about the other layers of the network? To calculate $\nabla_{(b^n)} l$ and $\nabla_{(w^n)} l$ for all layers n in the network, we perform the following computations:

from chain rule:

$$\frac{\partial l}{\partial a^{n-1}_j} = \sum_i \frac{\partial l}{\partial z^n_i} \cdot \frac{\partial z^n_i}{\partial a^{n-1}_j} \tag{13}$$

this is because node a^{n-1}_j affects all nodes a^n_i for all nodes i in layer n hence we need to calculate the average change that each node in layer n makes to all nodes in layer $n-1$.

From equation 4:

$$\frac{\partial l}{\partial a^{n-1}_j} = \sum_i \frac{\partial l}{\partial z^n_i} \cdot w^n_{i,j} \tag{14}$$

Hence:

$$\nabla_{(a^{n-1})} l = (w^n)^T \cdot \nabla_{(z^n)} l \tag{15}$$

Finally from the chain rule:

$$\frac{\partial l}{\partial z^{n-1}_i} = \frac{\partial l}{\partial a^{n-1}_i} \cdot \frac{\partial a^{n-1}_i}{\partial z^{n-1}_i} \tag{16}$$

Now from equation 2:

$$\frac{\partial l}{\partial z^{n-1}_i} = \frac{\partial l}{\partial a^{n-1}_i} \cdot F'(z^{(n-1)}_i) \tag{17}$$

Hence:

$$\nabla_{(z^{n-1})} l = \nabla_{(a^{n-1})} l \cdot \nabla_{(z^{n-1})} F \tag{18}$$

Now from equation 14:

$$\nabla_{(z^{n-1})} l = (w^n)^T \cdot \nabla_{(z^n)} l \cdot \nabla_{(z^{n-1})} F \tag{19}$$

Now, after obtaining $\nabla_{(z^n)} l$ in equation 12, we can calculate $\nabla_{(z^{n-1})} l$, and consequently calculate $\nabla_{(b^{n-1})} l$ and $\nabla_{(w^{n-1})} l$ as in equations 8 and 11, respectively.(2)

Now as we want to reduce the error in the biases and weights we can update our weights and biases as in the following two equations:

$$(w^n) = (w^n) - \alpha(\nabla_{(w^n)} l) \tag{20}$$

$$(b^n) = (b^n) - \alpha(\nabla_{(w^n)} l) \tag{21}$$

1.2. Principles of CNN

One limitation of artificial neural networks (ANNs) is that their predictions can heavily rely on the presence of specific patterns in specific locations within the image. If an image is altered or its position is changed relative to the original training data, the model may struggle to make accurate predictions.

In a CNN, instead of individual nodes representing the probability of a specific pattern's existence, each node in a convolutional layer is associated with a matrix known as a filter or kernel. The entries of this matrix represent the probabilities or weights indicating the presence of certain patterns in specific parts of the image. These filters allow the CNN to capture spatial hierarchies of features, starting from low-level patterns (such as edges) and progressing to more complex patterns (such as shapes and objects).

In Convolutional Neural Networks (CNNs), the process of obtaining values in the nodes is accomplished through image convolution. Convolution involves using a kernel, which is a smaller matrix, typically smaller than the size of the input image. The kernel matrix represents a specific pattern or feature that the CNN aims to identify.

During the convolution operation, the kernel matrix is multiplied with different portions of the original image. This multiplication process highlights regions in the image that have a higher probability of containing the specific pattern represented by the kernel. As a result, the parts of the image with a strong likelihood of containing the desired pattern will generate larger values.

During the training process, the network learns to adjust the values within the kernels to effectively detect specific patterns or features in the input data. This learning occurs through an iterative optimization process, such as backward propagation, where the network adjusts the kernel values based on the desired output and the associated loss function.

2. Methodology

2.1. Building a Digit Prediction Artificial Neural Network (ANN) from Scratch

Description: In this project, we built a neural network to predict digits represented in gray scale images. The dataset consists of 42,000 images stored in an Excel sheet. Each row represents the pixel values of an image, with a size of 28 by 28 pixels (784 total pixels).

Network Architecture: The neural network consists of two hidden layers, each with ten nodes. The input layer has a size of 784 by 42,000, where each column represents the pixel values of an image, The output layer consists of 10 nodes, each representing the probability that the input belongs to a certain digit for all digits from 0 to 9. Building the Artificial Neural Network (ANN) is done through three phases, as follows:

1. Divide the available data into a training set and a testing set.
2. Rescale the pixel values of the dataset by dividing each pixel value by 255. This enhances the significance of weights in predicting the images and improves accuracy.

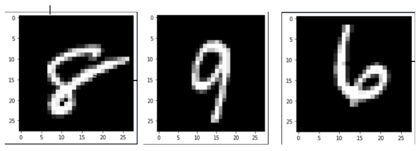
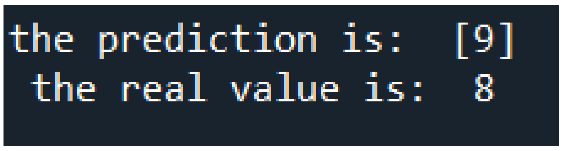


Figure 2: Three digits used in testing The model.



```
the prediction is:  [9]
 the real value is:  8
```

Figure 3: The output prediction of figure

3. Initialize the weights and biases with random values.
4. Perform the forward propagation function. In this step, the network predicts the inputs and calculates the loss function (error).
5. Execute the backward propagation function to adjust the weights and biases. This returns the updated weights and biases that reduce the error.
6. Repeat steps 4 and 5, where each forward propagation uses the weights and biases obtained from the previous backward propagation. Measure accuracy by comparing the predicted output with the true values. Stop repeating when a satisfactory accuracy is achieved.
7. Evaluate the network's performance using the test dataset.

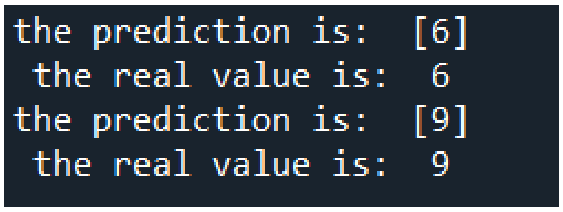
2.2. Testing the digit recognition model

Figure 2 represent a sample of three images used in testing. In Figures 3 and 4, we observe the predictions for three digits in figure 2. Notably, one prediction for image 2 is incorrect.

Considering the low accuracy of 85% achieved by this model, our objective is to develop a sign language model capable of addressing the complexity inherent in sign language images. Due to their intricate patterns, we will transition from using an Artificial Neural Network (ANN) to employing a Convolutional Neural Network (CNN) in the design of our new sign language model.

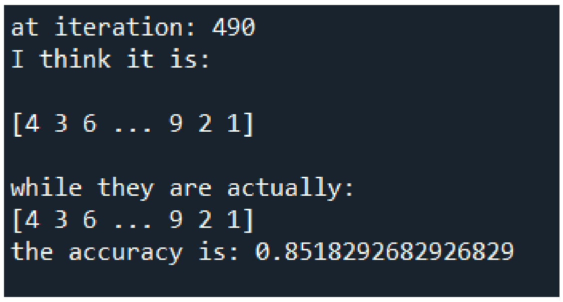
Figure 5 displays the final epoch of training the model, with each epoch representing an iteration of training on each image in the dataset. The image includes the following information:

1. Iteration number: It indicates the specific iteration within the epoch.



```
the prediction is:  [6]
 the real value is:  6
the prediction is:  [9]
 the real value is:  9
```

Figure 4: The output correct prediction of figure 2.



```
at iteration: 490
I think it is:

[4 3 6 ... 9 2 1]

while they are actually:
[4 3 6 ... 9 2 1]
the accuracy is: 0.8518292682926829
```

Figure 5: The last epoch of training the model it shows a list of predictions and a list of real values and accuracy of 85.1%

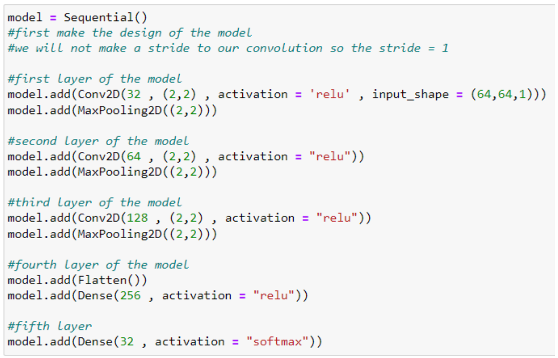
2. Predicted values: These are the output values predicted by the model for the input images during that iteration.
3. Real values: These are the actual values of the corresponding input images in the dataset.
4. Accuracy of the prediction: It represents the accuracy achieved by the model in predicting the output values for the given input images.

2.3. *Building a sign language model*

Dataset: For this project, we utilized a dataset containing 53,392 images representing the 32 letters of the Arabic sign language (ArSL). Each image has dimensions of 64 by 64 pixels, resulting in a total of 4,096 pixels per image. All pixel values range from 0 for black and 255 for white. The dataset is composed of several JPG images.(3)

The model training and construction involve the following steps:

1. Data Preparation: The images are converted into pixel values and stored in an Excel sheet. Each row in the sheet represents the pixel values of a single image.
2. Train-Test Split: The data is divided into training images (50,000) and testing images (remaining).
3. Shuffling: The data is shuffled to enhance the neural network's performance and eliminate potential biases.
4. Reshaping: The pixel values of each image are reshaped into a 64 by 64 format. This is necessary for convolutional operations using kernels, as the input to the CNN must have the same dimensions as the original image.
5. Label and Pixel Arrays: The training and testing datasets are separated into label arrays and pixel arrays. The index i of the labels array corresponds to the label of the pixel array at index i.
6. One-Hot Encoding: The labels are assigned unique numbers, and one-hot encoding is applied to transform these numbers into binary vectors.
7. Rescaling the pixel values of the dataset: To enhance the significance of weights in image prediction and improve overall accuracy, we rescale each pixel value by dividing it by 255.
8. Designing the model: In this step, as shown in figure 6, we construct the layers of the model. Firstly, we add a convolutional layer with 32 nodes, which allows the model to differentiate between 32 patterns. Next, we incorporate two additional convolutional layers with 64 and 128 nodes respectively. Following that, we flatten the layers and create a dense layer similar to those in an Artificial Neural Network (ANN). The purpose of this dense layer is to produce the final prediction.



```python
model = Sequential()
#first make the design of the model
#we will not make a stride to our convolution so the stride = 1

#first layer of the model
model.add(Conv2D(32 , (2,2) , activation = 'relu' , input_shape = (64,64,1)))
model.add(MaxPooling2D((2,2)))

#second layer of the model
model.add(Conv2D(64 , (2,2) , activation = "relu"))
model.add(MaxPooling2D((2,2)))

#third layer of the model
model.add(Conv2D(128 , (2,2) , activation = "relu"))
model.add(MaxPooling2D((2,2)))

#fourth layer of the model
model.add(Flatten())
model.add(Dense(256 , activation = "relu"))

#fifth layer
model.add(Dense(32 , activation = "softmax"))
```

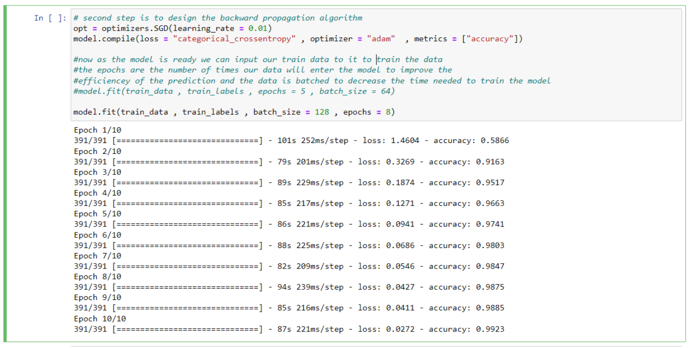
Figure 6: The designing of the sign language neural network

9. Model compilation: During this step, you specify the learning algorithm used and the techniques employed to enhance the model's performance. We have specified the categorical cross-entropy loss function, which is commonly used for classifying non-binary models that differentiate between three or more classes.
10. Model fitting: In this step, as shown in figure 7, we initiate the training process by inputting the data into the model and performing forward propagation and backward propagation multiple times. For this particular project, we conducted the fitting process over eight epochs. To streamline the training procedure and improve efficiency, we divided the data into 391 batches, with each batch containing 128 samples. This batching approach reduces complexity and enhances the overall runtime of the training process.
11. Model evaluation: During this step, we assess the performance of the model by inputting the test data and making predictions using the model. We then calculate the accuracy by comparing the predicted outputs with the actual outputs in the test dataset. The accuracy is determined by calculating the average error across the entire test dataset.
12. Save the model: In this step we save the model as a .h format so that each time we make predictions with the model we do not have to train the model again.

2.4. *Testing the sign language model*

Figure 8 illustrates the evaluation of the model, which involves testing the test data on the trained model. The achieved accuracy for the model is reported to be 95.9

Figure 9 showcases the output of the model, specifically predicting the letters ”raa”. Notably, the prediction made by the model aligns with the ground truth, indicating a correct prediction.



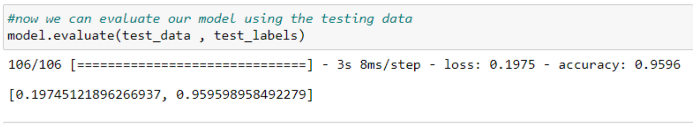
```
In [ ]: # second step is to design the backward propagation algorithm
        opt = optimizers.SGD(learning_rate = 0.01)
        model.compile(loss = "categorical_crossentropy" , optimizer = "adam"  , metrics = ["accuracy"])

        #now as the model is ready we can input our train data to it to train the data
        #the epochs are the number of times our data will enter the model to improve the
        #efficiencey of the prediction and the data is batched to decrease the time needed to train the model
        #model.fit(train_data , train_labels , epochs = 5 , batch_size = 64)

        model.fit(train_data , train_labels , batch_size = 128 , epochs = 8)

        Epoch 1/10
        391/391 [==============================] - 101s 252ms/step - loss: 1.4604 - accuracy: 0.5866
        Epoch 2/10
        391/391 [==============================] - 79s 201ms/step - loss: 0.3269 - accuracy: 0.9163
        Epoch 3/10
        391/391 [==============================] - 89s 229ms/step - loss: 0.1874 - accuracy: 0.9517
        Epoch 4/10
        391/391 [==============================] - 85s 217ms/step - loss: 0.1271 - accuracy: 0.9663
        Epoch 5/10
        391/391 [==============================] - 86s 221ms/step - loss: 0.0941 - accuracy: 0.9741
        Epoch 6/10
        391/391 [==============================] - 88s 225ms/step - loss: 0.0686 - accuracy: 0.9803
        Epoch 7/10
        391/391 [==============================] - 82s 209ms/step - loss: 0.0546 - accuracy: 0.9847
        Epoch 8/10
        391/391 [==============================] - 94s 239ms/step - loss: 0.0427 - accuracy: 0.9875
        Epoch 9/10
        391/391 [==============================] - 85s 216ms/step - loss: 0.0411 - accuracy: 0.9885
        Epoch 10/10
        391/391 [==============================] - 87s 221ms/step - loss: 0.0272 - accuracy: 0.9923
```

Figure 7: Compiling and training of the model and the output correspondent to it.

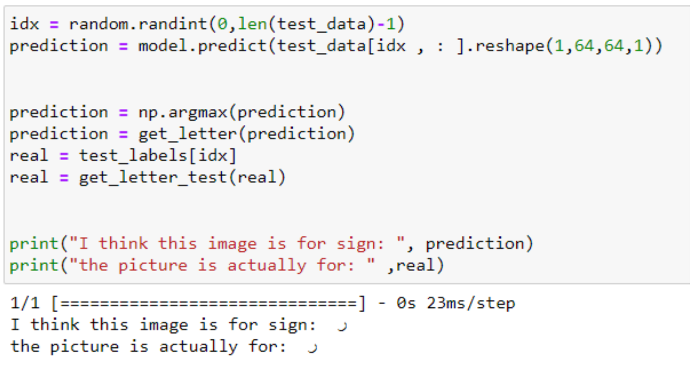


```
#now we can evaluate our model using the testing data
model.evaluate(test_data , test_labels)

106/106 [==============================] - 3s 8ms/step - loss: 0.1975 - accuracy: 0.9596

[0.19745121896266937, 0.9595989584922791]
```

Figure 8: The evaluation of the model. The first indix of the output represents the loss percentage and the second index represents the accuracy



```
idx = random.randint(0,len(test_data)-1)
prediction = model.predict(test_data[idx , : ].reshape(1,64,64,1))


prediction = np.argmax(prediction)
prediction = get_letter(prediction)
real = test_labels[idx]
real = get_letter_test(real)


print("I think this image is for sign: ", prediction)
print("the picture is actually for: " ,real)

1/1 [==============================] - 0s 23ms/step
I think this image is for sign:  ر
the picture is actually for:  ر
```

Figure 9: The evaluation of the model. The first indix of the output represents the loss percentage and the second index represents the accuracy

Acknowledgements

I want to express my gratitude to Dr. Rami for his invaluable assistance throughout the project. His guidance and support have been instrumental in its success. Additionally, I would like to extend my thanks to Eng Zeyad for his valuable help, which greatly contributed to the project's accomplishments.

References

[1] "IBM," https://www.ibm.com/topics/neural-networks, accessed: May 23, 2023.

[2] E. Demaine, "Backpropagation," Webpage, 2006, mIT Computer Science and Artificial Intelligence Laboratory. [Online]. Available: https://fab.cba.mit.edu/classes/864.20/people/erik/notes/backpropagation.html

[3] G. Latif, N. Mohammad, J. Alghazo, R. AlKhalaf, and R. AlKhalaf, "Arasl: Arabic alphabets sign language dataset," *Data in Brief*, vol. 23, p. 103777, April 2019, open Access. [Online]. Available: https://www.data-in-brief.com/article/S2352-3409(19)30128-3/fulltext