Handwritten Mathematical Operations Calculator Using Convolution Neural Network

Buğra Hüsrev Erdoğan

Doğuhan Ay

Abstract

Mathematics is known as the most difficult subject for young children who have just started education. The fact that other lessons are verbal, something is understood by reading and the logic is easier to understand is an indicator of why mathematics is more difficult to learn. We felt the need to develop this project because there are operations that children who are just learning mathematics cannot solve, and parents cannot always take care of their children. We developed this project, which recognizes handwritten numbers and operations and finds the result, using Convolution Neural Network (CNN), so that children can learn the answer to the question by taking a single photo.

Index Terms

Deep Learning, Convolutional Neural Network , Classification

I. Introduction

Mathematics can be said to be difficult not only for children but for everyone in general. Especially for children, it is difficult to understand issues such as the intelligibility of mathematical operations, the value of numbers and the priority of operations. Children who can't always get support from an adult on questions they can't do, on subjects they don't understand, are getting colder from the math they are already having difficulty with. The ability of today's children to use technology is undeniable compared to the older generation. This is exactly why it is a bigger problem for them to find the operations that they cannot do by using a calculator with numbers that they can hardly understand. Our approach in this project is to provide a quick solution to the questions that children cannot do by taking advantage of their tendency to technology. In order to do this, we first need to understand the problem. The primary problem here is to distinguish the numbers and operations written by the handwriting of children who have just started education. What makes this problem difficult is being able to recognize the operations correctly and show them the result, even if the children's writings are difficult to understand.

II. RELATED WORK

First of all, we had to properly recognize the numbers and signs written by children who have just started writing and education. In order to do this, it was necessary to assume that what they were going to do would be somewhat bad. In order to do this, we used a data set containing 8752 pictures consisting of numbers and figures as they would write.

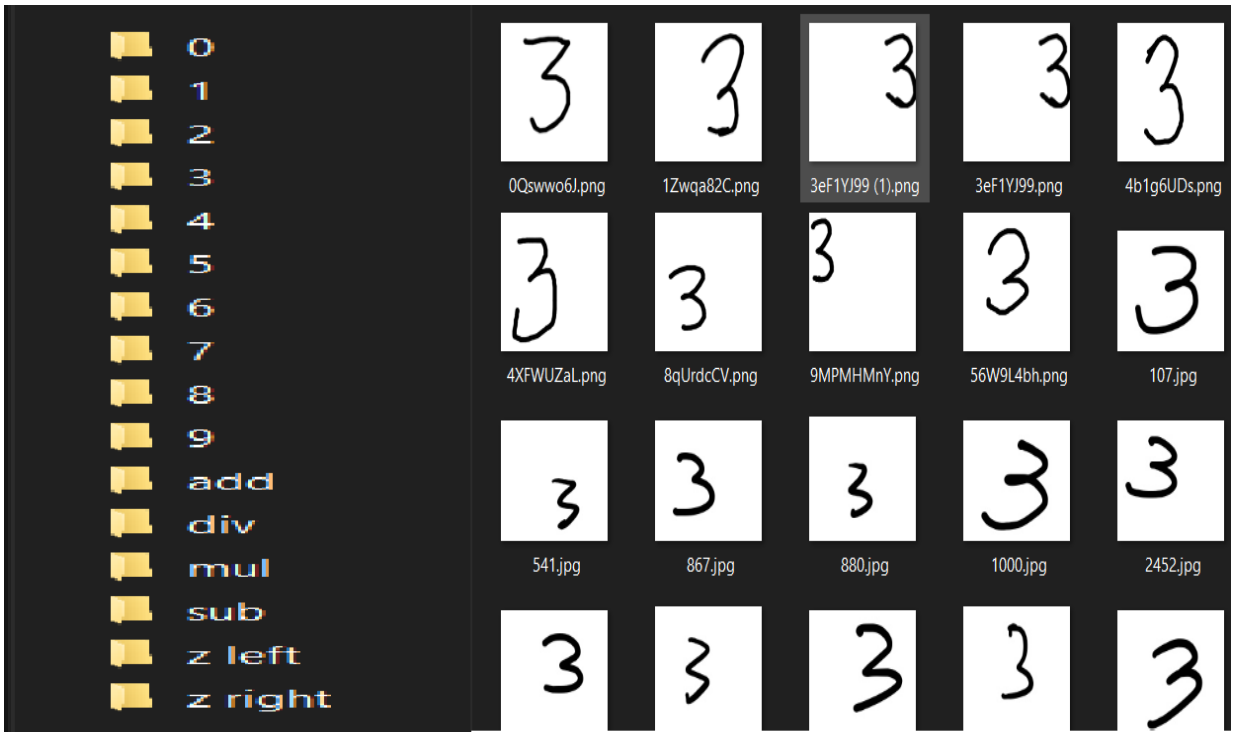
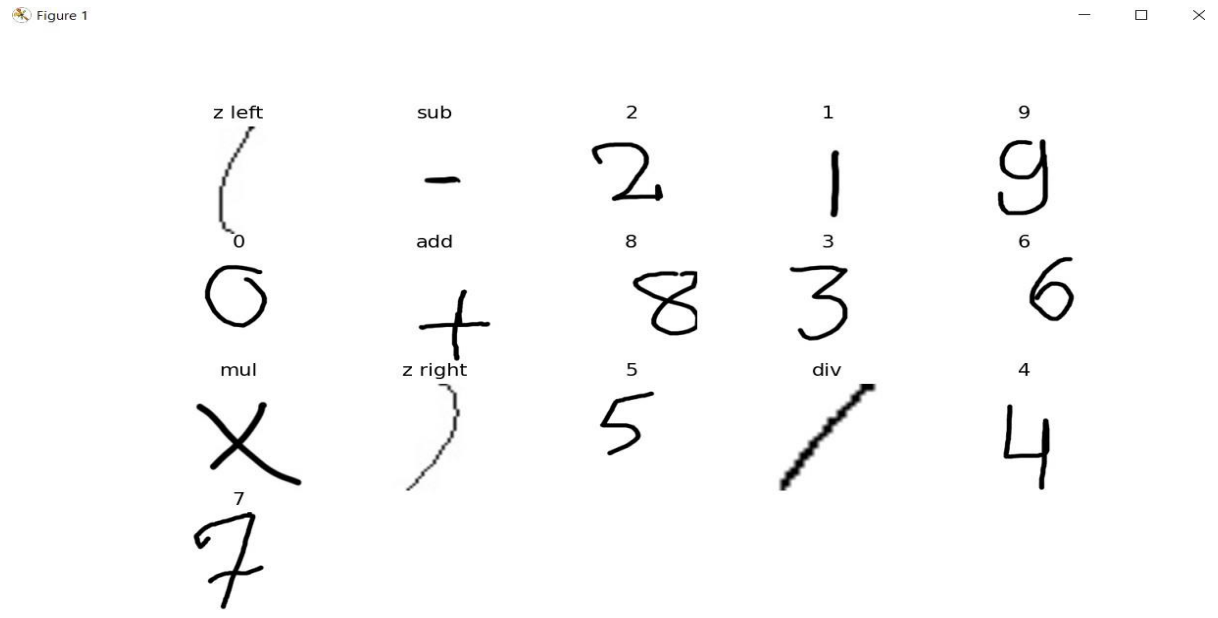


FIGURE 1. Dataset example

This data set, which consists of a total of 16 classes, consists of numbers and shapes drawn similar to those of children. We chose these classes considering that children who are just starting out in education will have more difficulty in basic level operations, operation priority and use of parentheses.

Before preparing our model for this process, we created a loop through the dataset file to introduce the classes in the dataset and showed a screen that introduces all the classes. This screen is seen by the developer, not the user.



We created two models in our project. Our first model is the AlexNet model. To use this model, we first resized the images in our dataset to 227*227. Then we used 'LabelEncoder' and separated train-test data. We divided the number of train data to make up 80% of the whole data set and the number of test data to make up 20% of the whole data set.

Next is creating the model. Our first model, AlexNet, was created by adhering to the traditions. For the first layer, 11*11 96 filters were used by using 227*227*1 input shape.

Layer	Type	Maps	Size	Kernel Size	Stride	Padding	Activation
Out	Fully Connected	–	1000	–	–	–	Softmax
F10	Fully Connected	–	4096	–	–	–	ReLU
F9	Fully Connected	–	4096	–	–	–	ReLU
S8	Max pooling	256	6X6	3X3	2	valid	–
C7	Convolution	256	13X13	3X3	1	same	ReLU
C6	Convolution	384	13X13	3X3	1	same	ReLU
C5	Convolution	384	13X13	3X3	1	same	ReLU
S4	Max pooling	256	13X13	3X3	2	valid	–
C3	Convolution	256	27X27	5X5	1	same	ReLU
S2	Max pooling	96	27X27	3X3	2	valid	–
C1	Convolution	96	55X55	11X11	4	valid	ReLU
In	Input	3(RGB)	227X227	–	–	–	–

Unlike normal, we converted both our dataset and input to gray scale image. Therefore, we used 227*227*1 input shape instead of 227*227*3. Also, unlike normal, we changed our size value in the last layer to 16 because we wanted it to choose between 16 classes, not 1000 classes. We used 'softmax' activation in the output layer because we wanted the model to give us a result value between 0 and 1, that is, to select a single class. We tried two different optimizers for the learning rate value, which we specified as '0.001'. We used SGD because we needed to choose between Adam and SGD and when we analyzed the results, we thought that we could classify better with SGD. We tried two different things inside the Loss function. One was 'categorical_crossentropy' and the other was 'sparse_categorical_crossentropy'. After analyzing and researching both , we realized that our classes are not completely mutually exclusive . While analyzing this , we found that the model had confusion for certain numbers in the test data we gave . That's why we decided to use 'categorical_crossentropy'. It's time to train the model we created. While doing this, we resized our test and train data, which we had changed before, by 0.1%. The reason for this was that the main value part of our dataset was not generally in the middle. The next challenge is to set the Batch_size and Epochs. First, we used the traditional AlexNet's 128 batch size. While doing this, we determined 30 epochs. But when we

analyzed the accuracy value and the result it gave to our test data, we decided that the epoch value was insufficient because when we analyzed the accuracy value, it was around 0.71. That's why we retrained using 50 epochs and 128 batch sizes. After about 3 hours of training, we got the following results.

FOR ALEX NET

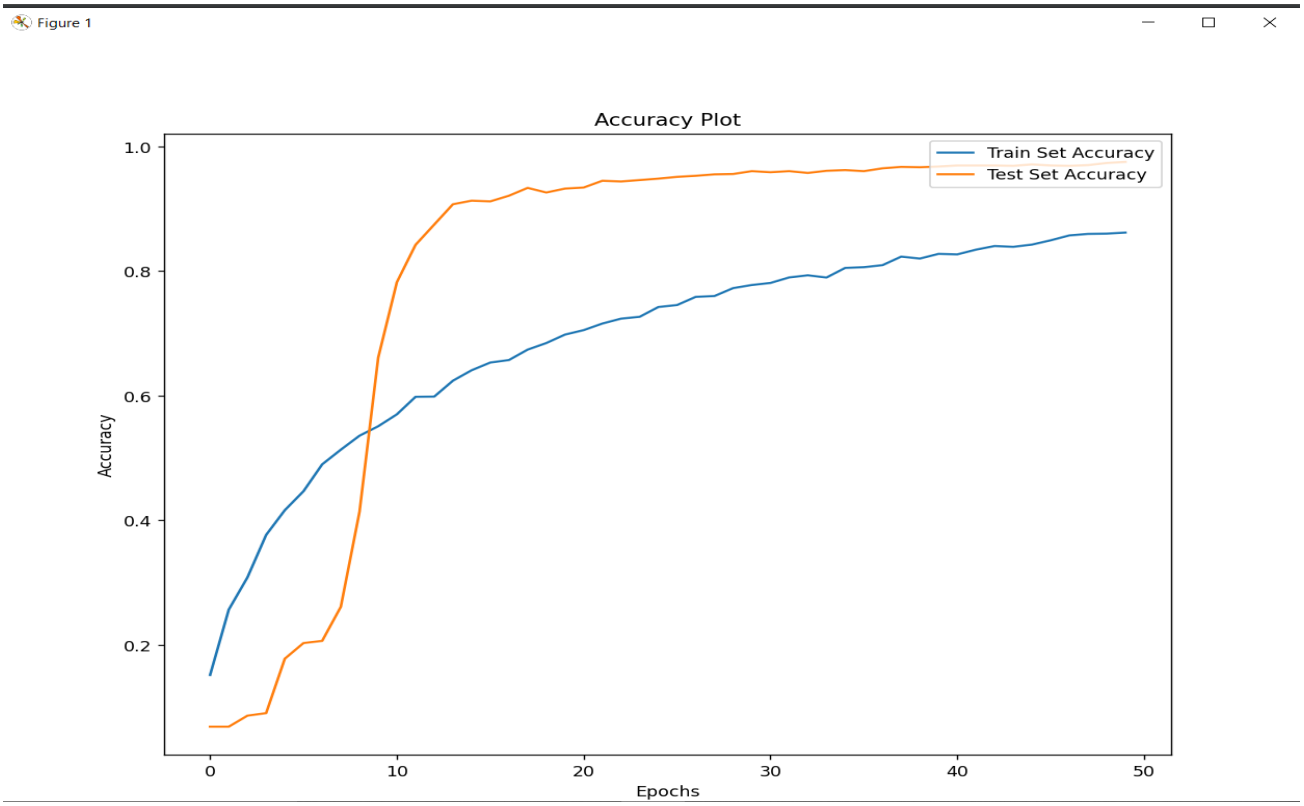
```
Epoch 50/50
55/55 [==============================] - 155s 3s/step - loss: 0.4270 - accuracy: 0.8614 - val_loss: 0.1072 - val_accuracy: 0.9749
55/55 [==============================] - 7s 133ms/step
              precision    recall  f1-score   support

           0       0.95      0.96      0.95       119
           1       0.97      0.93      0.95       102
           2       0.92      0.99      0.95        72
           3       0.97      0.98      0.98       105
           4       0.98      0.93      0.95       112
           5       0.98      0.96      0.97        93
           6       0.92      0.97      0.95       115
           7       0.96      0.96      0.96       108
           8       0.99      0.97      0.98       123
           9       0.96      0.97      0.97       110
          10       0.99      0.99      0.99       119
          11       1.00      1.00      1.00        87
          12       0.99      0.98      0.99       125
          13       0.99      1.00      1.00       120
          14       1.00      1.00      1.00       119
          15       1.00      1.00      1.00       122

    accuracy                           0.97      1751
   macro avg       0.97      0.97      0.97      1751
weighted avg       0.98      0.97      0.97      1751
```

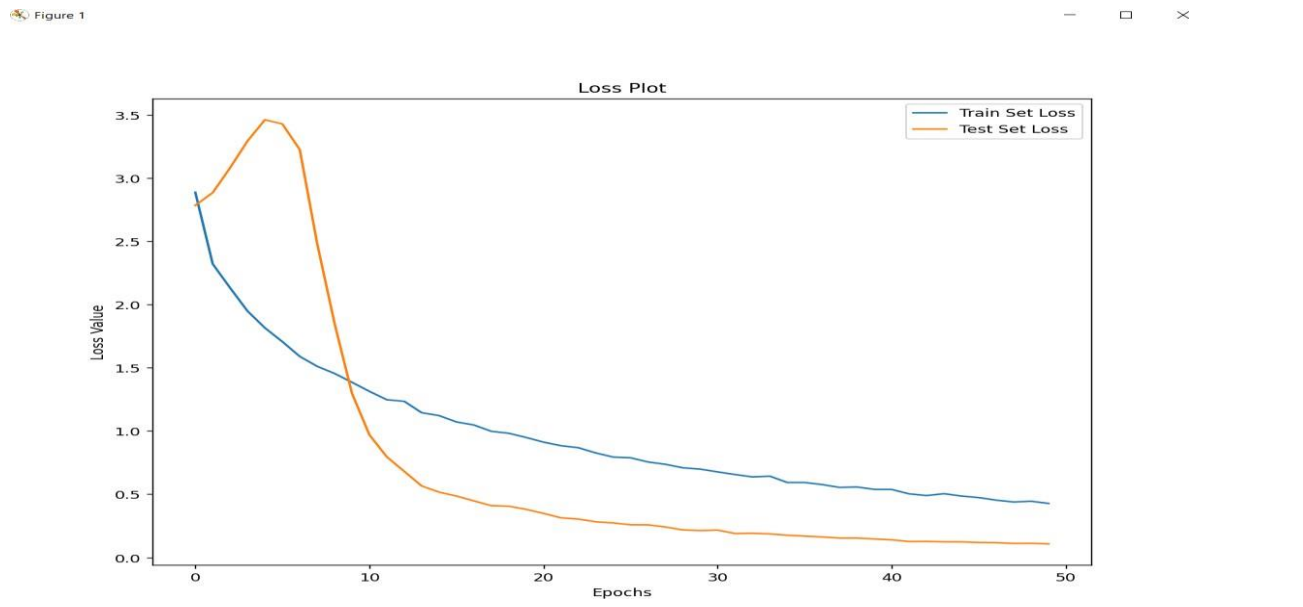
FOR OUR NET

```
Epoch 97/100
55/55 [==============================] - 3s 49ms/step - loss: 0.0864 - accuracy: 0.9847 - val_loss: 0.0901 - val_accuracy: 0.9897
Epoch 98/100
55/55 [==============================] - 3s 48ms/step - loss: 0.0838 - accuracy: 0.9863 - val_loss: 0.0862 - val_accuracy: 0.9897
Epoch 99/100
55/55 [==============================] - 3s 50ms/step - loss: 0.0752 - accuracy: 0.9854 - val_loss: 0.0994 - val_accuracy: 0.9857
Epoch 100/100
55/55 [==============================] - 3s 60ms/step - loss: 0.0842 - accuracy: 0.9843 - val_loss: 0.1032 - val_accuracy: 0.9874
55/55 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       119
           1       1.00      0.98      0.99        98
           2       0.99      0.99      0.99        76
           3       1.00      0.97      0.99       115
           4       0.97      0.99      0.98       115
           5       1.00      0.97      0.98       101
           6       0.97      0.99      0.98       118
           7       0.98      0.98      0.98       103
           8       0.99      0.96      0.98       108
           9       0.96      0.97      0.97       110
          10       1.00      1.00      1.00       133
          11       1.00      1.00      1.00        93
          12       0.99      1.00      1.00       104
          13       1.00      1.00      1.00       142
          14       0.99      1.00      1.00       102
          15       1.00      1.00      1.00       114

    accuracy                           0.99      1751
   macro avg       0.99      0.99      0.99      1751
weighted avg       0.99      0.99      0.99      1751


Process finished with exit code 0
```

When we analyzed the Accuracy and Loss graphs, we noticed a strangeness. Because the graphs moved very independently from each other and we couldn't get the result we wanted.

ALEX NET PLOTS



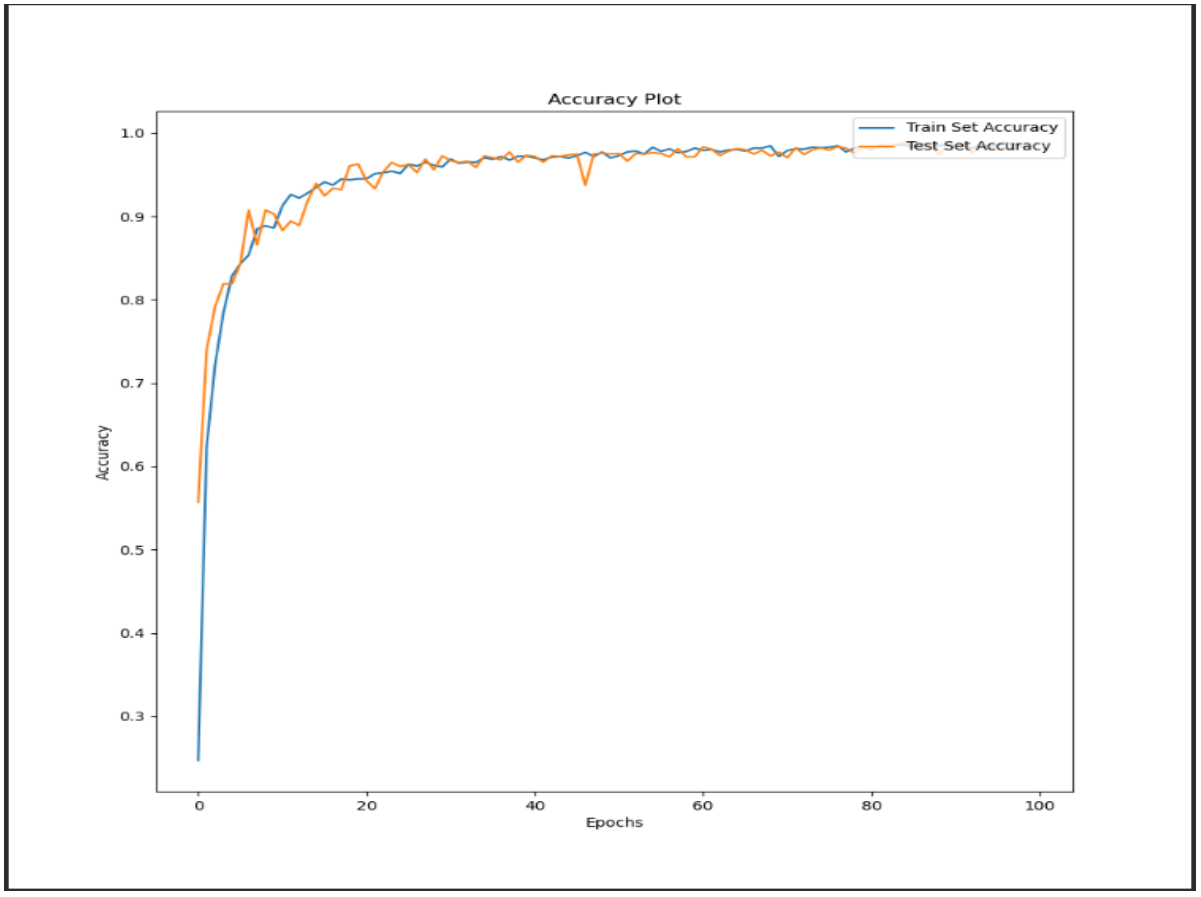


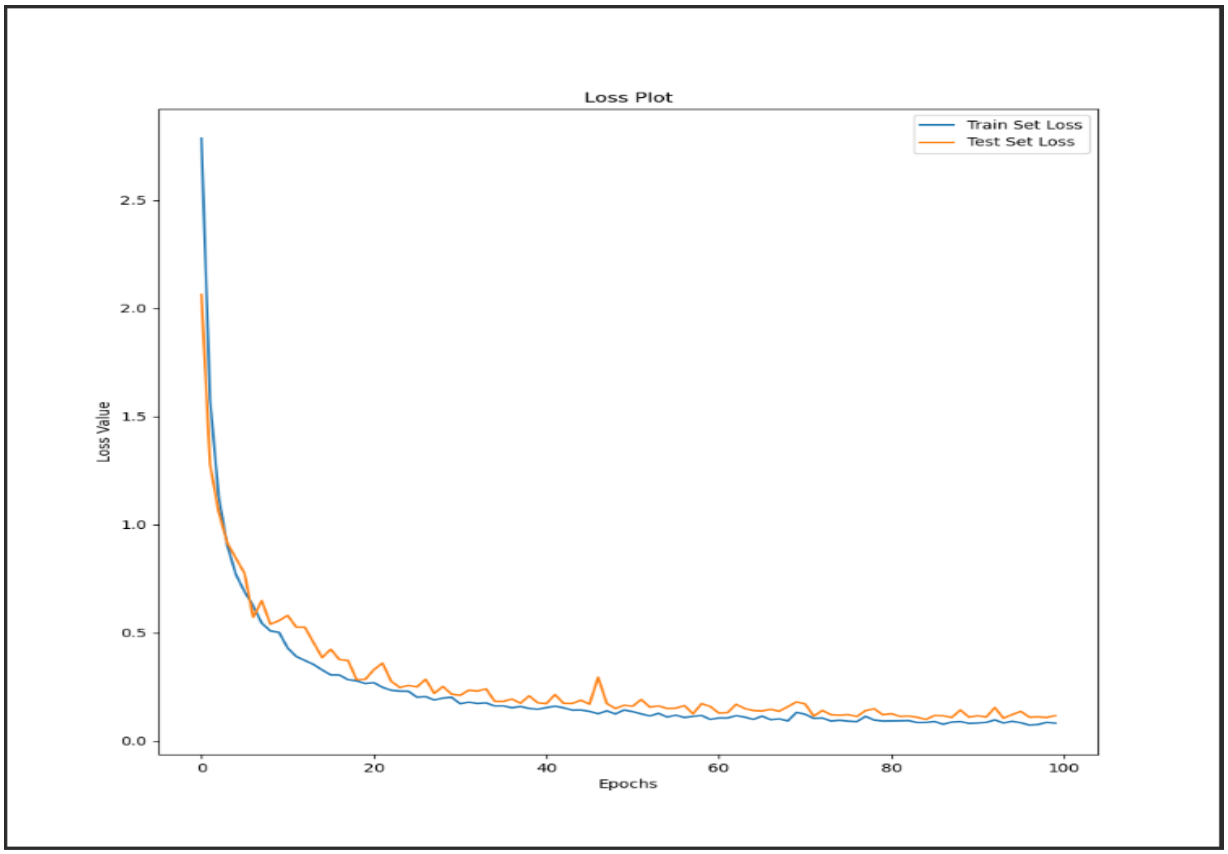
As a result, we changed the neuron values in batch size and fully connected steps. We tried batch size 64 and 256. We also reduced the number of neurons from 4096 to 128. However, we still could not get the exact result we wanted. Then we saved our model to avoid retraining it

every time. Then we started testing our model by giving test pictures. Our model uses the following steps to classify the picture we give.

First, it resizes the image to 800*800 and converts it to gray scale. It then begins to analyze the picture from left to right. While doing this, we got help from the OpenCV library. OpenCV first determines the numbers and characters on the image. Then, it transmits this determined shape to the model with the size of 227*227*1. The model analyzes that shape and compares it with the classes it has learned before. If he compares the shape to what he learned before, it's time to determine which shape it is. While doing this, it compares the output of the shape it determines with the names of the classes. When there is a matching class, it adds the value of that class (ex. '+', '2') to the String value we have created as an empty. It repeats this process until he finishes the examination of the whole picture. Finally, he finds the mathematical result of the String value he added and shows it to the child.

OUR MODEL PLOTS

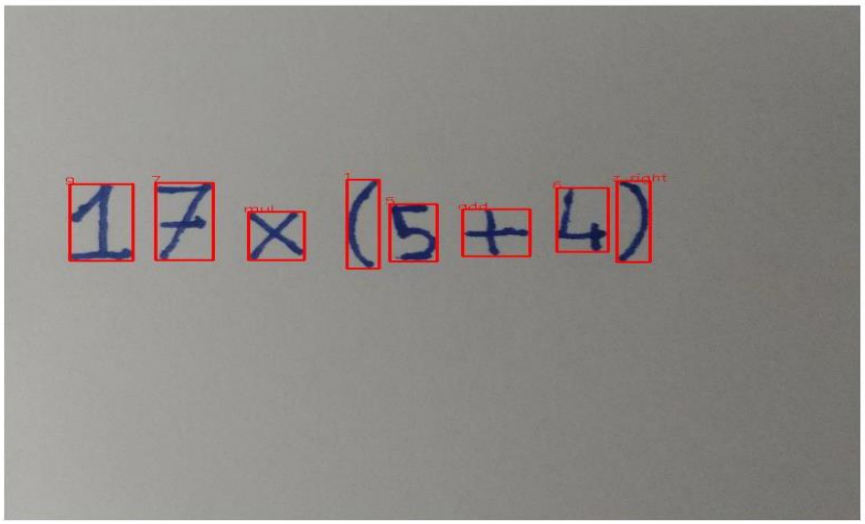




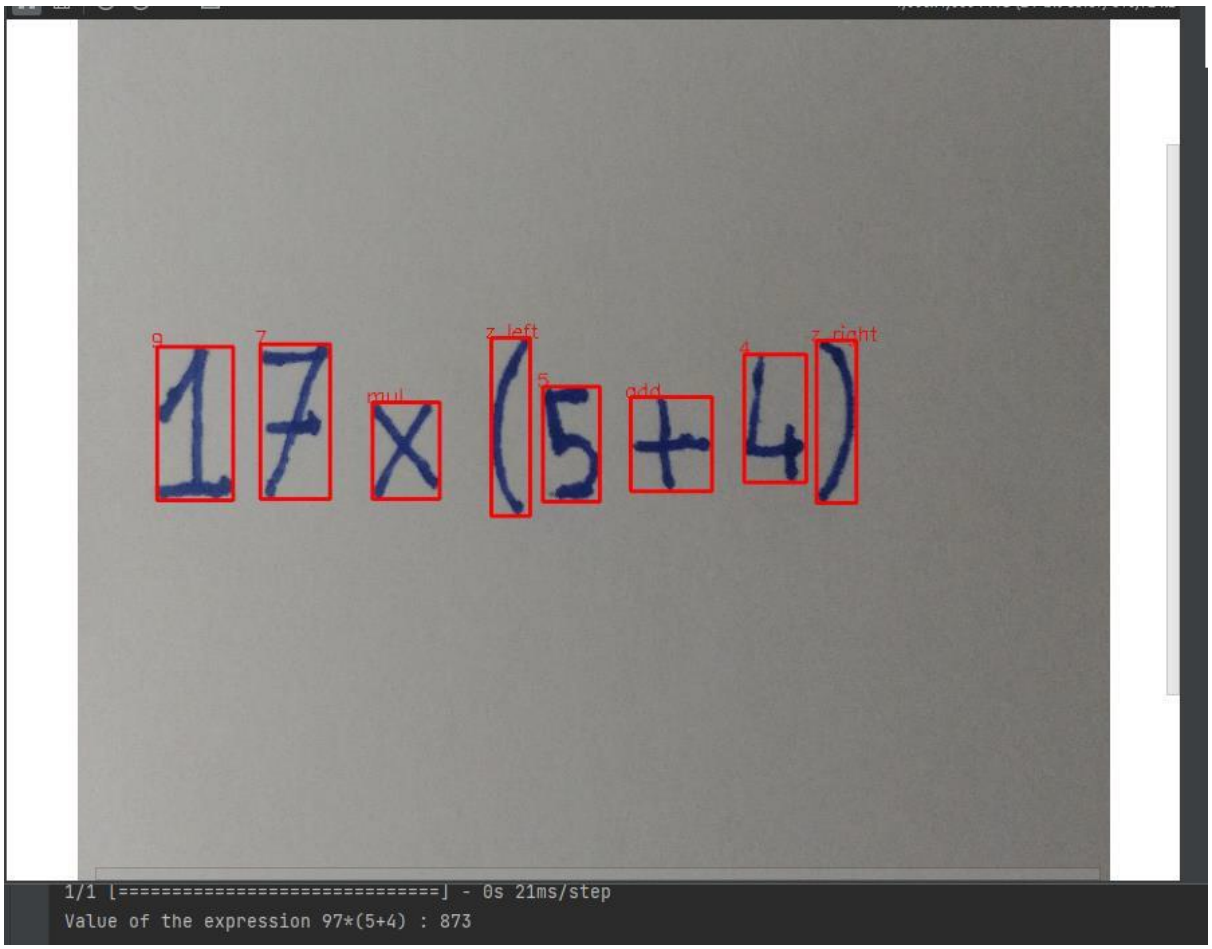
As a result, we changed the neuron values in batch size and fully connected steps. We tried batch size 128. We also reduced the number of neurons from 120 to 84. However, we still could not get the exact result we wanted. Then we saved our model to avoid retraining it every time. Then we started testing our model by giving test pictures. Our model uses the following steps to classify the picture we give.

First, it resizes the image to 800*800 and converts it to gray scale. It then begins to analyze the picture from left to right. While doing this, we got help from the OpenCV library. OpenCV first determines the numbers and characters on the image. Then, it transmits this determined shape to the model with the size of 32*32*1. The model analyzes that shape and compares it with the classes it has learned before. If he compares the shape to what he learned before, it's time to determine which shape it is. While doing this, it compares the output of the shape it determines with the names of the classes. When there is a matching class, it adds the value of that class (ex. '+', '2') to the String value we have created as an empty. It repeats this process until he finishes the examination of the whole picture. Finally, he finds the mathematical result of the String value he added and shows it to the child.

FOR ALEX NET



FOR OUR NET



Since we don't need to train the model every time anymore, it's enough to just send the picture of the process to the model we saved with the model.load() command.

USAGE OF PROJECT

THERE ARE 3 PYTHON FILE (alexnet, ourCnnModel, test)

To test your equation with one of the codes below, leave the comment line to the variable image_path and send it to the one for the relevant model as below.(on test.py)

FOR ALEX NET

```
new_model = tf.keras.models.load_model('saved_model/alexNet_ann_hw.h5')

image_path = 'ourtest/deneme3.jpg'
AlexNet.test_equation(image_path, new_model)
```

FOR OUR NET

```
our_test_model = tf.keras.models.load_model('saved_model/our_model_ann_hw.h5')
image_path = 'ourtest/deneme3.jpg'
OurCnnModel.test_equation(image_path, our_test_model)
```