A Complete Sudoku Solver Using Machine Learning.

Tenzin Tashi

Prof: Dr. Erik Grimmelmann

CSC 44700 – Introduction to Machine Learning

May 20, 2022

Table of Contents

Objective:

The main objective of this project is to create an app that will solve the given unsolved Sudoku problem from an image by using different machine learning models. Sudoku is a logic-based, combinatorial number-placement puzzle (Sudoku). The classic Sudoku is made of 9x9 grid with digits, and the objective is to fill each cell with digits such that each column, each row, and each of the nine 3x3 sub-grids contain all the digits from 1 to 9 without repetition.

Methodology:

To accomplish this goal, we will be using 3 main libraries that are OpenCV and PIL, for image processing, and TensorFlow which will be doing the heavy lifting of building and training the models. I have divided the whole project into 3 parts to make it clear and easy to understand. The Part I deals with building digit classification using Convolutional Neural Network (CNN), Part II deals with Image Processing where we create an Optical Character Recognition (OCR) which detects each digit inside the given Sudoku puzzle image using the part I neural network model, and finally in the Part III we build a Sudoku Puzzle Solver model using CNN.

Part I: Digit Classification Model

Introduction:

In this part we will be first building and training a neural network on the MNIST dataset of handwritten digits. This model will help to classify the digits from the images.

Dataset:

The dataset we are using for this part is from the MNIST Dataset, which has a training set of 60,000 examples and a test set of 10,000 examples. We will split the training set into 80% to training and the remaining 20% to validation set It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image of 28x28 pixel.

Data Visualization:

After importing the dataset and splitting it into training, validation, and testing, we first visualize the data to see how the data is being given and the distribution of the data.

```
plt.figure(figsize=(10,5))
plt.bar(range(0,num_classes), num_samples)
plt.title('No of images for each classes')
plt.xlabel("Class ID")
plt.ylabel('Number of Images')
plt.show()
```

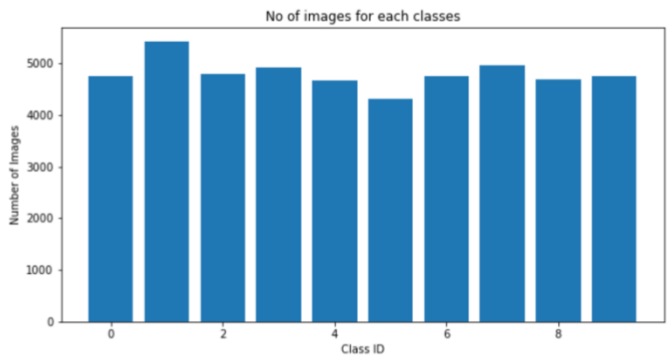


Figure 1 Code segment showing the distribution of data

In the above figure the x-axis shows that we have total of 10 classes labeled from (0 to 9) and the y-axis show the number of images per class. We can infer from the above figure that the dataset is not biased towards any one of the classes, which means the model we build using the above dataset won't we biases to any of the class.

Then to further visualize the images per given class we wrote the code segment shown in the figure 2.

```
### visualize
f, ax = plt.subplots(1, num_classes, figsize=(20,20))

for i in range(num_classes):
    sample = x_train[y_train == i][0]
    ax[i].imshow(sample, cmap='gray')
    ax[i].set_title('Lable: {}'.format(i), fontsize=16)
```

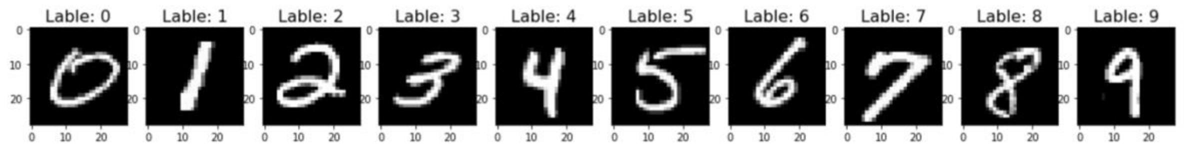
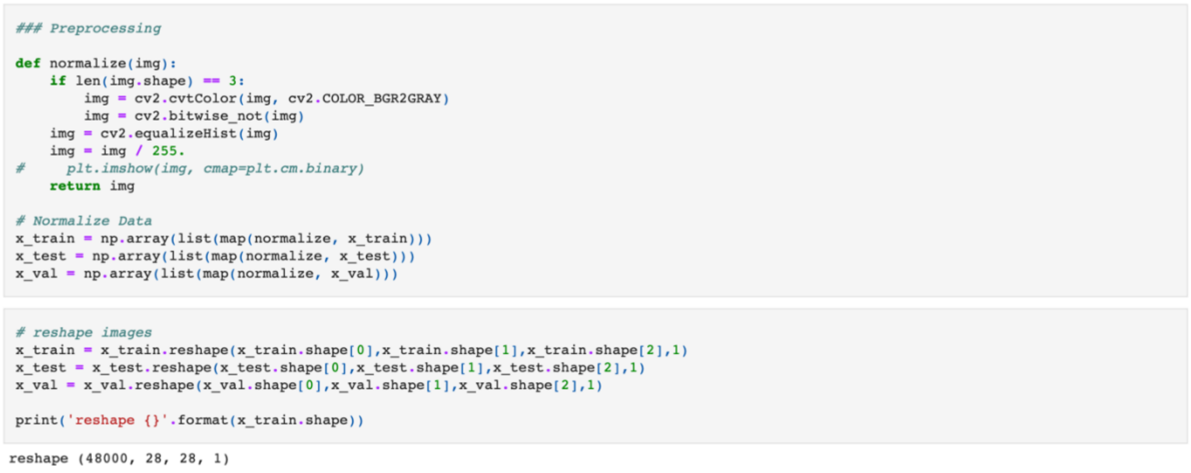


Figure 2 Images per classes

Data Preparation:

Now that we have visualize the data, the next step is to prepare the data to make it ready to be used in the neural network. In a preprocessing step, we preprocess the features (images) into grayscale, normalizing and enhancing them with histogram equalization. After that, convert them to NumPy arrays then reshaping them. We also preprocess the labels (classes) using the one-hot encoding.

The goal of data normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information.



```python
### Preprocessing

def normalize(img):
    if len(img.shape) == 3:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        img = cv2.bitwise_not(img)
    img = cv2.equalizeHist(img)
    img = img / 255.
#     plt.imshow(img, cmap=plt.cm.binary)
    return img

# Normalize Data
x_train = np.array(list(map(normalize, x_train)))
x_test = np.array(list(map(normalize, x_test)))
x_val = np.array(list(map(normalize, x_val)))
```

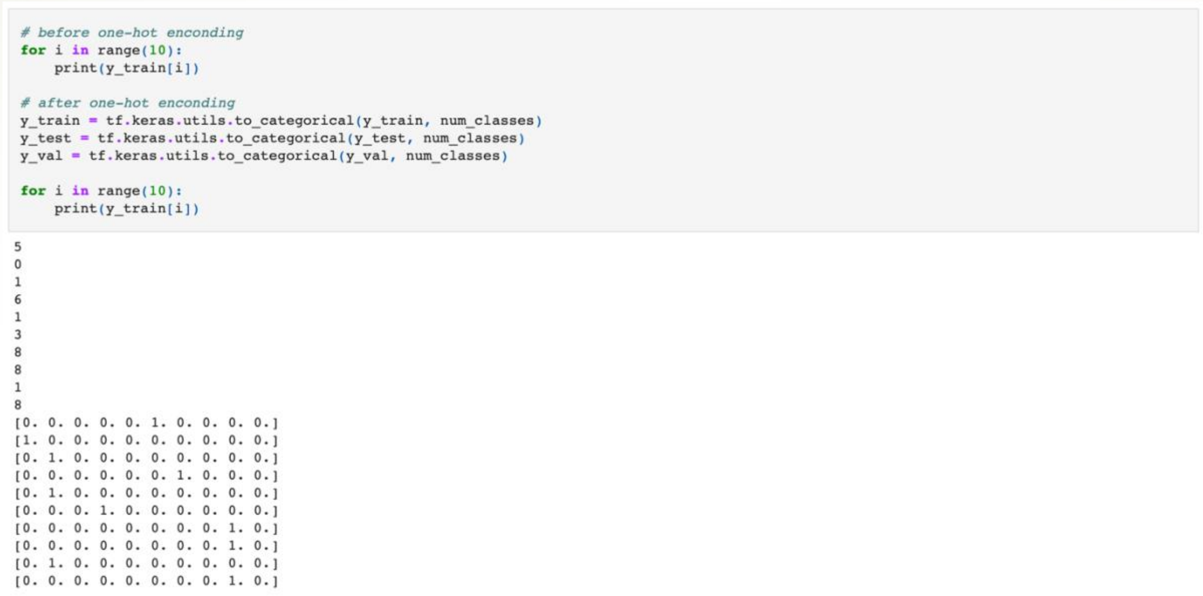
```python
# reshape images
x_train = x_train.reshape(x_train.shape[0],x_train.shape[1],x_train.shape[2],1)
x_test = x_test.reshape(x_test.shape[0],x_test.shape[1],x_test.shape[2],1)
x_val = x_val.reshape(x_val.shape[0],x_val.shape[1],x_val.shape[2],1)

print('reshape {}'.format(x_train.shape))
```

```
reshape (48000, 28, 28, 1)
```

Figure 3 Preprocessing the Features

One Hot Encoding is a common way of preprocessing categorical features for machine learning models. This type of encoding creates a new binary feature for each possible category and assign a value of 1 to the feature of each sample that corresponds to its original category as shown in the figure 4 below.



```
# before one-hot enconding
for i in range(10):
    print(y_train[i])

# after one-hot enconding
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)
y_val = tf.keras.utils.to_categorical(y_val, num_classes)

for i in range(10):
    print(y_train[i])
```

```
5
0
1
6
1
3
8
8
1
8
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
```

Figure 4 Preprocessing of Label

Data Augmentation:

Data augmentation is a set of techniques to artificially increase the amount of data by generating new data points from existing data. This includes making small changes to data to generate new data points. It is useful to improve performance and outcome of machine learning models by forming new and different examples to train datasets.

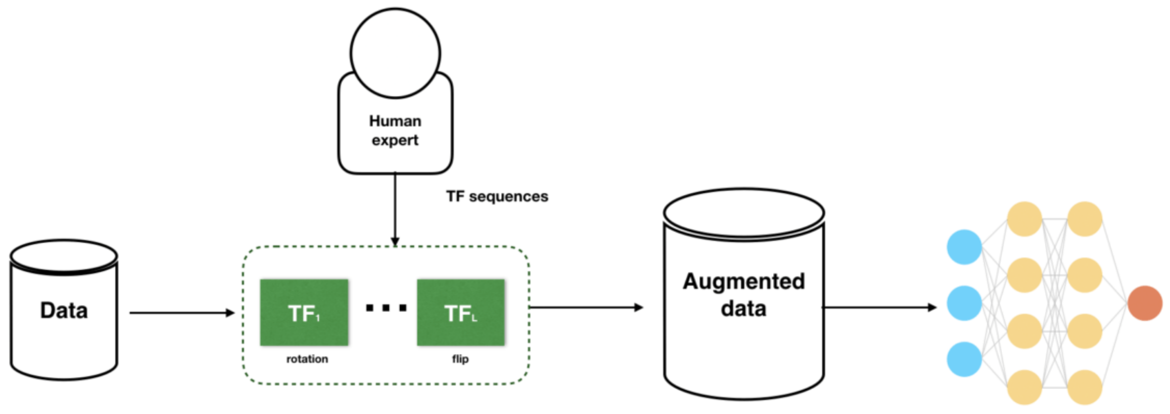


Figure 5 Data augmentation

Model:

We are using a convolutional neural network for model building.

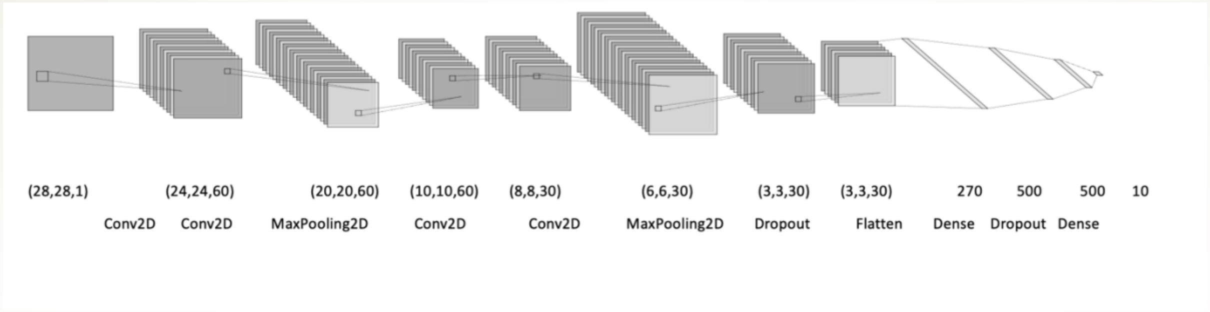


Figure 6 Architecture of CNN model for Digit Recognition

The TensorFlow library provides us ready to use API to build the model we want. We begin with creating an instance of the Sequential model. We then add individual layers into the model. The first layer is a convolutional layer that processes input image of 28x28. We define the kernel size as 5 and create 60 of such kernels to create an output of 60 frames of size 24x24 (28-5+1=24).

We repeat the above step of convolutional layer of same kernel size to create an output of 60 frames of size 20x20.

Then it is followed by max pooling layer of 2x2. This reduces the dimensions from 20x20 to 10x10. We used max pooling because we know that the essence of the problem is based on edges, and we know that edges show up as high values in a convolution.

We follow the above 3 steps again, which result in the 30 frames of 3x3 frame size.

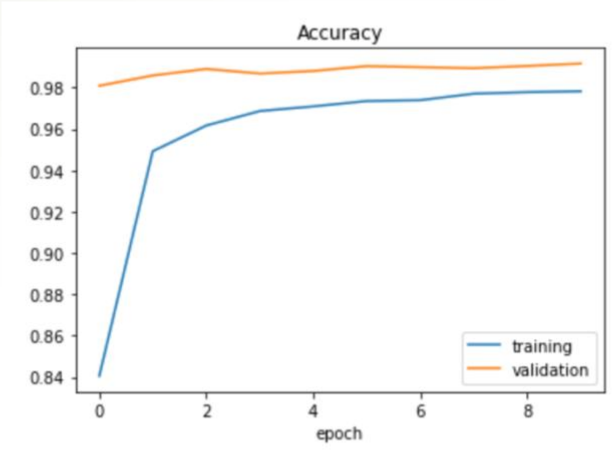
Then we dropout the 50% of the data, to prevent it from overfitting.

It is then flatten and feed to a dense layer of 500. Again, we dropout the 50% of the data and finally feed it to the dense layer that has outputs corresponding to the 10 required values.

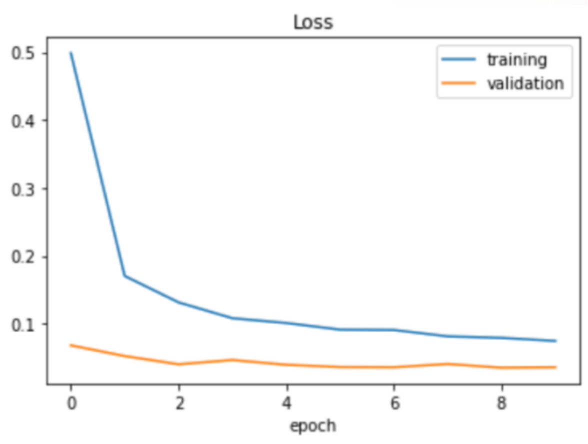
The full convolutional architecture is shown in the figure 6 above.

Result:

After the building the model we train it with the data we have for 10 epochs. And in just 10 epoch we were able to achieve and accuracy of 99.20% with loss of 2.39% on the testing dataset. The Graphs below shows the model's accuracy vs epoch and loss vs epoch.



Graph 2 Accuracy vs Epoch



Graph 1 Loss vs Epoch

Figure 7 Model Evaluation with Test Image

Then we created 28x28 image of some digits and ran our newly build model to predict this digit we can see the digits like 1 to 9 except (0 or black image) is being detected correctly with almost 100% accuracy but the blank image is being precited as 1 (wrong prediction) but with low accuracy, we will use this knowledge in next part.

Part II: Reading and Detecting the Sudoku from an Image

Introduction:

The goal of this part is to be able to detect and grab each 81 cells of the Sudoku board from the given image and run the digit classification model of part I to get the 2D array of digit that represent the given Sudoku image. To accomplish this goal the following steps were considered.

Read Image:

The first and foremost step is reading an image. We will be using OpenCV module to read the image.

Figure 8 Code snippet of reading image using OpenCV

Find the sudoku box outline:

The next step is getting the outline of the Sudoku box from the given image. This step is important to find the outline of the Sudoku board from the whole image helping us ignore rest of the unneeded pixels.

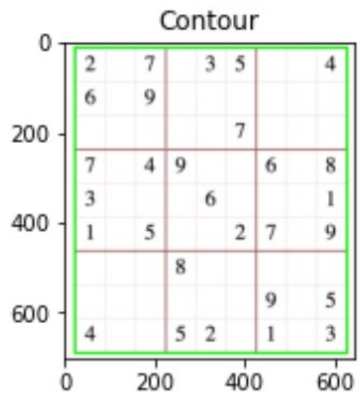


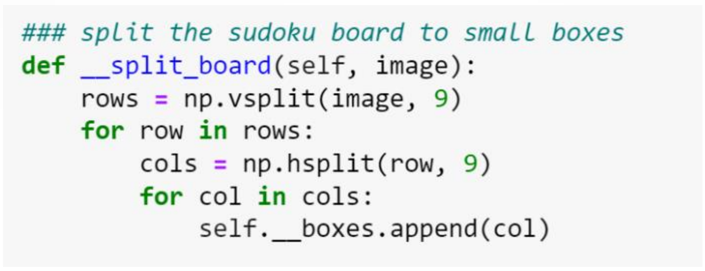
Figure 9 Outer Contour of the Sudoku box

Extract the location of the 4 corners from the sudoku outline:

After finding the outer contour of the Sudoku board from the input image, the next step is to get the find the 4 end points of the Sudoku box and rewrap the image using those end points.

Divide it into 9 rows and 9 cols:

After getting the wrapped image of just Sudoku box we split the rows and columns into 9 equal parts. It is done using the NumPy methods: vsplit (vertical split) for the rows and hsplit (horizontal split) for the columns.



```python
### split the sudoku board to small boxes
def __split_board(self, image):
    rows = np.vsplit(image, 9)
    for row in rows:
        cols = np.hsplit(row, 9)
        for col in cols:
            self.__boxes.append(col)
```

Figure 10 Code snippet of using NumPy vsplit and hsplit

Detect the Digit:

The Final Step of this part is to run the digit recognition model that we build in the Part I to each of the cells (total of 81) that we split in the previous step and store it into a list. One thing to keep in mind during this step is that in the Part I result I have shown when we pass an image with no digit it detects that image as 1 with the probability less than 90%, so we will be using that knowledge while running the digit recognition in each cell and if the probability is less than 90% we will add 0 in the list else we will add the digit that was classified by the model. The reason of adding 0 if the probability is too low is because in Sudoku puzzle, we have digit only from 1 to 9 and 0 means the empty cell that needs to be filled by the Sudoku solver model which will be shown in the next Part.

Part III: Puzzle Solver Model

Introduction

This is the final section of the project. In this part we will be training and building neural network model than will take an input of integer 2D array representing the Sudoku Puzzle and output its solution array. This is the section that sets apart this project from the rest. The other paper that I have researched does this section by backtracking algorithm but this section of mine does it by building neural network that learns and solves the Sudoku problem.

Dataset:

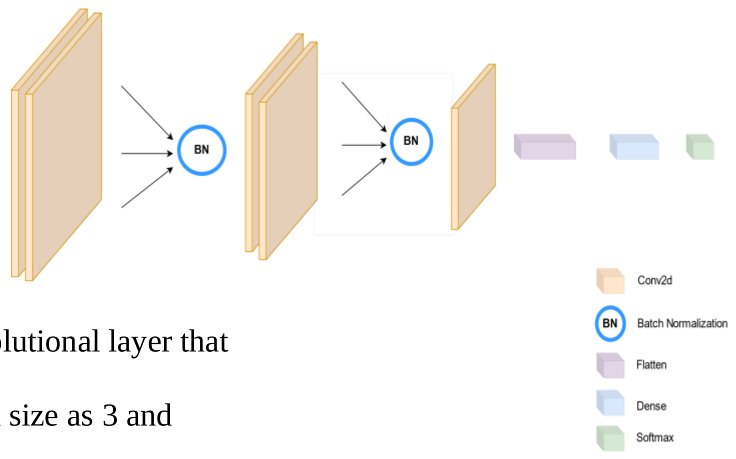
The dataset that I will be using for this section is from the Kaggle, it a total of 1 million Sudoku Games. This dataset contains 2 columns: quizzes, the unsolved sudoku puzzle, which will be used as the features and solutions, the solved sudoku puzzle, which will be used as the labels.

Data Preparation:

After getting the features and labels from the dataset, and then splitting it into train and test, the preprocessing step that we have done are first changing the features and labels into appropriate shape so it can be feed onto the machine learning model. Finally, the features were normalized by dividing it by 9.

Model:

Similar to the Part I model we will be using TensorFlow library to build the model we want. We begin with creating an instance of the Sequential model. We then add individual layers into the model. The first layer is a convolutional layer that processes input of 9x9x1. We define the kernel size as 3 and



create 64 of such kernels to create an output of 64 frames of size 9x9.

Then it is followed by Batch Normalization.

We repeat the same two steps again. Then it is followed by the convolutional layer to kernel size 1, and we created 128 frames of it.

After that we flatten the layers and dense it by 729. And finally, we reshape it into the desire shape of sudoku.

Result:

After the building the model we train it with the data we have for 10 epochs. And in just 10 epoch we were able to achieve and accuracy of 100% on the testing dataset. The Graphs below shows the model's accuracy vs epoch and loss vs epoch.

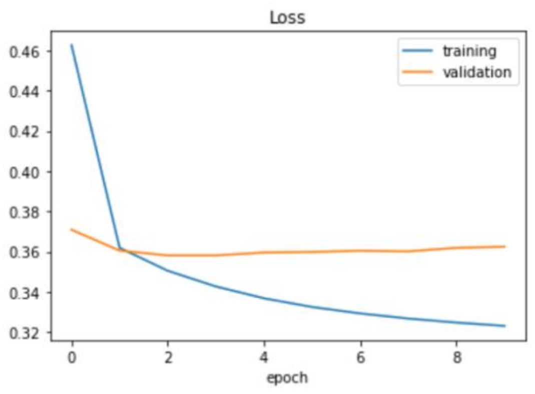


Figure 12 Loss vs Epoch

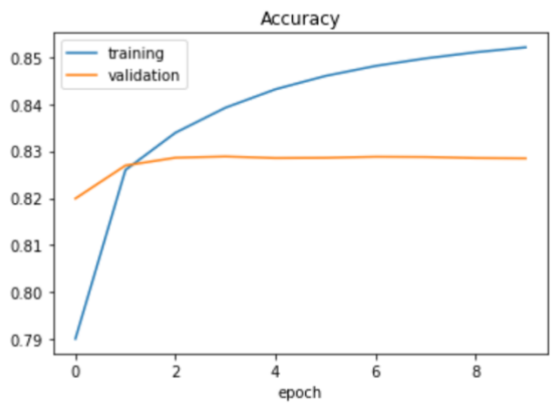
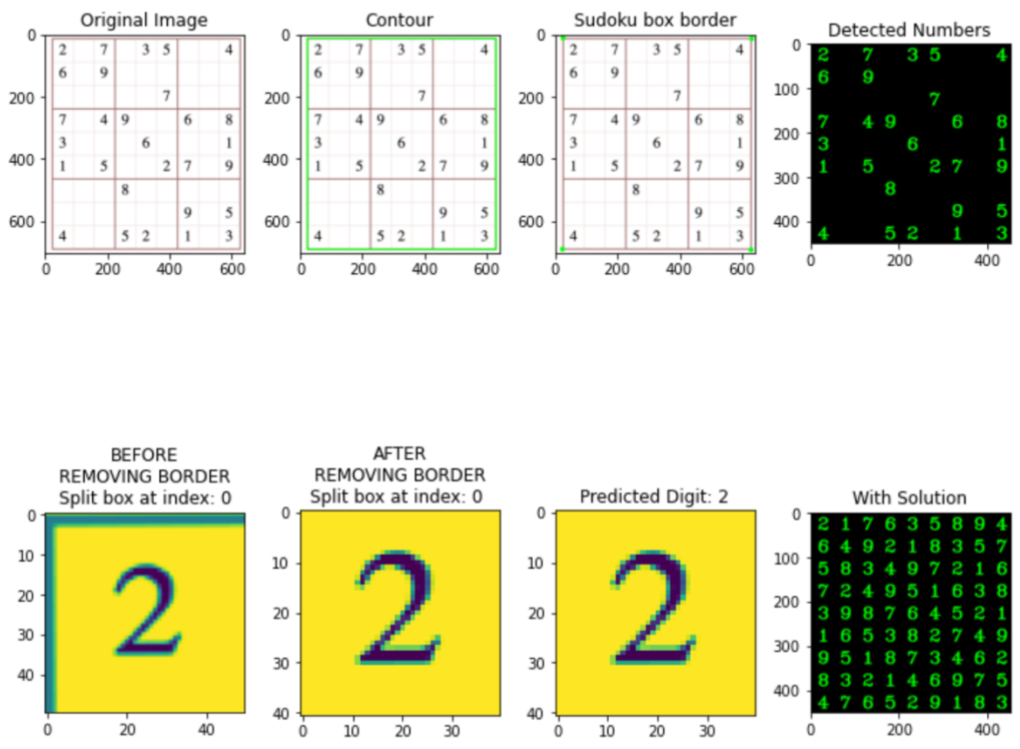


Figure 11 Accuracy vs Epoch

What we can infer from these graphs above is that after 2 epoch the validation graph (orange line stays constant) while the training graph (blue line converges) meaning the model is not overfitting.

Conclusion:

Putting it all together, In the end we combined all the 3 Parts together and was able to accomplish the goal we set out at the beginning of this paper. The figure below shows the result that we accrue from this project. The upper leftmost image labeled as “Original Image” is the Sudoku Puzzle Image that we will be sending in as the initial input, then the two image that follows it are the Image processing part where we get each cell of the sudoku box. The fourth image labeled as “Detected Numbers” is the result 2D array after running the Digit Recognition to each cell of the Sudoku box. One thing that was not explained in the Part II was that before running the Digit Recognition to each cell we had to do a little bit of Image segmentation to each cell as we can see the cell that we get contains some trace of border as show in the lower leftmost image, then when we feed the segmented image to the model we get the correct prediction. Once we get the 2D array of Detected numbers we feed it to the Sudoku solver model build in the Part III to get the solution as shown in the lower rightmost image.



Reference:

Aysegul TakimogluAysegul has experience in HR management, and Name *. “What Is Data Augmentation? Techniques & Examples in 2022.” *AIMultiple*, 18 Apr. 2022, https://research.aimultiple.com/data-augmentation/.

Park, Kyubyong. “1 Million Sudoku Games.” *Kaggle*, 29 Dec. 2016, https://www.kaggle.com/datasets/bryanpark/sudoku.

“Sudoku.” *Wikipedia*, Wikimedia Foundation, 16 May 2022, https://en.wikipedia.org/wiki/Sudoku.

“The Mnist Database.” *MNIST Handwritten Digit Database, Yann LeCun, Corinna Cortes and Chris Burges*, http://yann.lecun.com/exdb/mnist/.