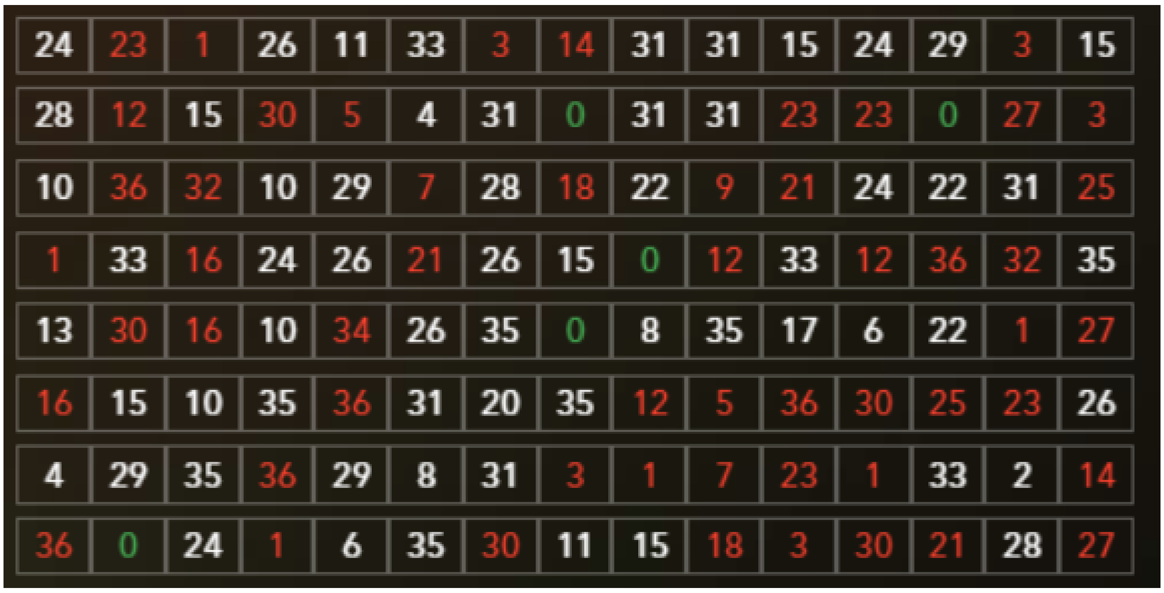
Digit recognition with convolutional neural network

It all started from the debate between me and my friend. The debate was about if it's possible to recognize numbers (8x15) from the picture and classify them properly. The picture below represents the problem we faced. There were thousands of these kind of pictures with lot of numbers. So, my friend needed a software which could do the labelling for him and I was certain that it was possible. Therefore, I accepted the challenge.



24	23	1	26	11	33	3	14	31	31	15	24	29	3	15
28	12	15	30	5	4	31	0	31	31	23	23	0	27	3
10	36	32	10	29	7	28	18	22	9	21	24	22	31	25
1	33	16	24	26	21	26	15	0	12	33	12	36	32	35
13	30	16	10	34	26	35	0	8	35	17	6	22	1	27
16	15	10	35	36	31	20	35	12	5	36	30	25	23	26
4	29	35	36	29	8	31	3	1	7	23	1	33	2	14
36	0	24	1	6	35	30	11	15	18	3	30	21	28	27

At that point I had heard about machine learning and digit recognition, but I wasn't familiar with those topics at all. I did my first attempt with MATLAB and I tried to make software to recognize all the digits at one. It wasn't successful attempt, but I learned a lot.

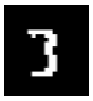
After that I heard about ongoing deep learning course in my university. I decided to join to the group. The course was about learning from theory as a group and then building up your own machine learning software as a project work. I had a perfect "real-world"- project for that purpose, so I was excited to give it a shot.

At that time, I already had enough knowledge about data sciences hence I knew the meaning of proper data manipulation in a first place. It was obvious that manipulating pictures in a right way would make training the model a lot easier. So that was my starting point.

There are lot of code available online for digit recognition, but they are mostly for classifying numbers from 0…9. As we can notice I had different problem. I had to make a software that was able to recognize digits from 0…36 and it was out of question to recognize them separately since 3 with 0 has different value than 30. So, there would be 37 different classes to classify these pictures. Another problem was the training dataset. How would I be able to generate enough training samples for the training. There are lot of samples for hand written digits from 0…9 but not from 0…36.

Steps:

1. Crop every single number from the picture above and turn them into black and white pictures

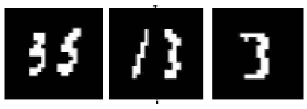
2. Load MNIST-handwritten dataset and see if I can utilize it.  Digits were not only numbers from 0…9 but they were also bigger than mine. So, it required some more coding to make them match better.

3. Manipulate MNIST-dataset by melting single digits and labels together in order to produce labelled dataset from 10…36.

At this point I had dataset for training my model. I created generator which would produce enough different samples from MNIST-dataset. I generated 250 000 samples from the data.

4. Classify digits manually. This was the boring part. I created a code for labelling digits in order to make enough samples from my data domain. I was ready for hire someone to do this, but I managed to label like 3000 samples

5. Build model. I knew I didn't have enough data from my data domain, so I tried to train model with generated MNIST-dataset (250 000) samples. At this point I was concerned about my computer's wellbeing while training the model. Training samples were like this.



The structure of used convolutional neural network. It was more like try and mistake.

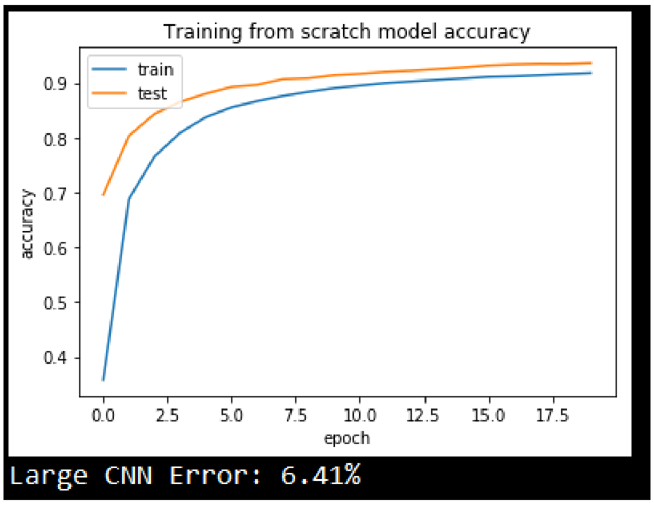
```
# Create model

model = Sequential()
model.add(Conv2D(30, (5, 5), input_shape=(28, 28, 1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(15, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(37, activation='softmax'))

# Compile model

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']
```

In the picture below is shown the best result I got. I'm still not sure how did it work with this kind of accuracy, but curves looked pretty good to me. Parameters were epochs= 20 and batch_size = 200



```
Large CNN Error: 6.41%
```

So, the results were promising, and I had model that could predict MNIST-type of data accurately. I was still concerned that I just don't have enough data for the training and I didn't feel like labelling anymore digits manually. At this point I heard about technique called fine-tuning. Fine-tuning is about modifying existing parameters of your model just a little bit. But this requires that distribution of the data is very similar with the other data.

6. Fine-tune existing model with manually labelled dataset. I knew I would have to modify parameters for fine-tuning because otherwise it would tune too much my model and achieved accuracy would be gone. After couple of attempts I ended up using epoch = 10 and batch_size = 10. Again, it was more like try and make mistake type of approach.

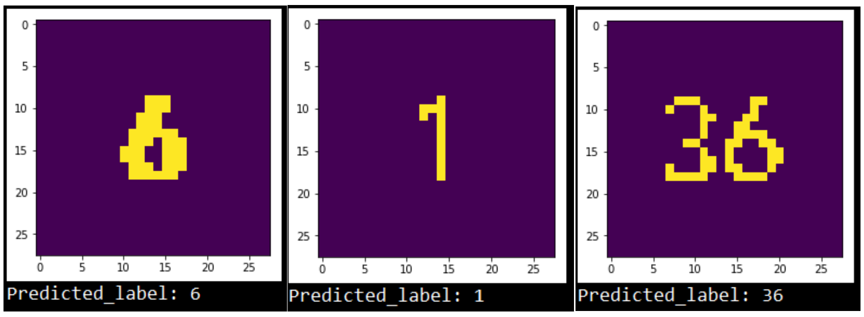
   # Fits the model on batches with real-time data augmentation, no augmentation used here:

   loaded_model.fit_generator(datagen.flow(X_train, y_train, batch_size=10), steps_per_epoch=len(X_train)/10, epochs = 10)

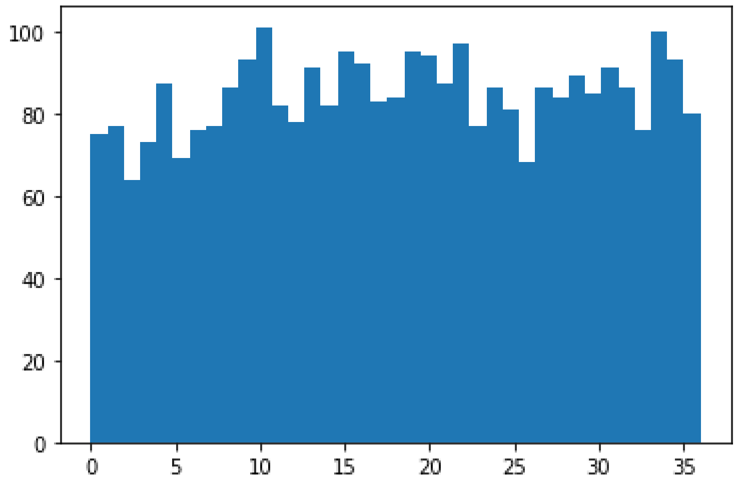
Results:

```
Epoch 1/10
300/300 [==============================] - 4s 12ms/step - loss: 0.3209 - acc: 0.9040
Epoch 10/10
300/300 [==============================] - 3s 11ms/step - loss: 0.0151 - acc: 0.9960
```

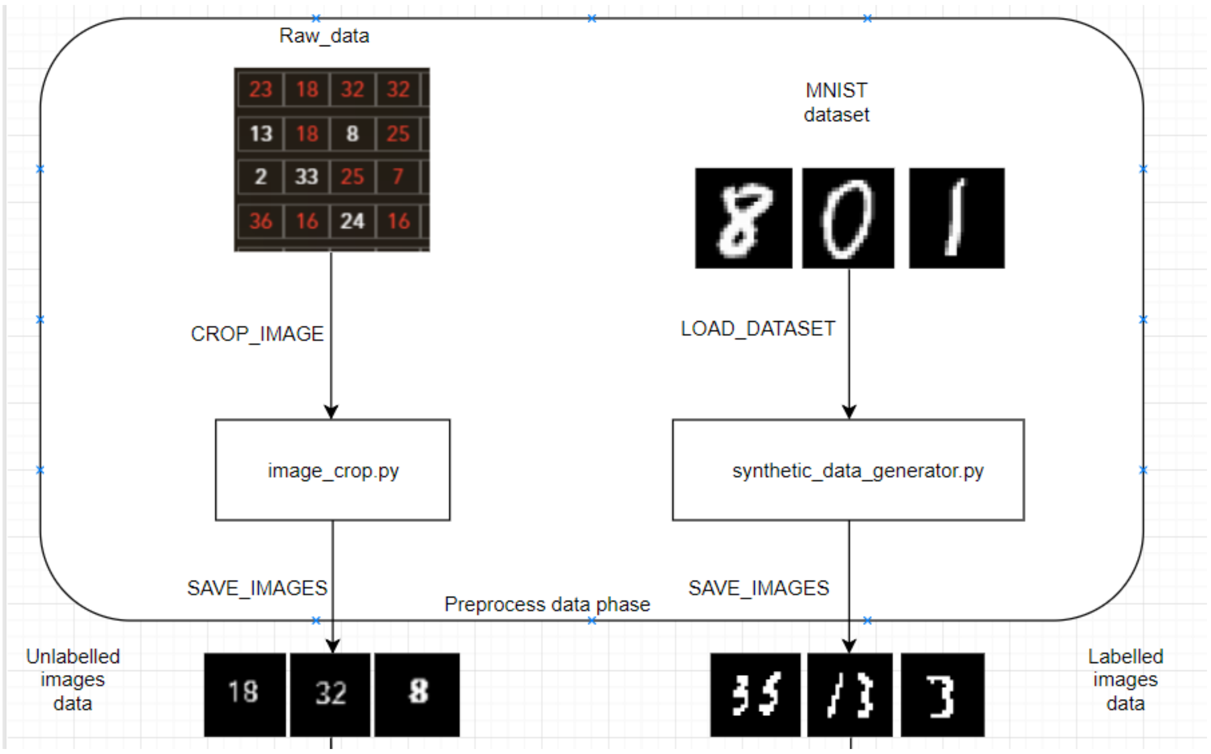
Over 99% accuracy is very suspicious, but it works for this purpose. This high accuracy is a first sign of overfitting model, but I believe that I managed to make distribution of my dataset narrow enough. Obviously, I tested this fine-tuned model with my unlabeled data. Below we can see the results. Pictures are still black and white, but the user interface turns them into purple and yellow. The model hasn't been shown these digits before, but apparently it has accurate idea about their classes.

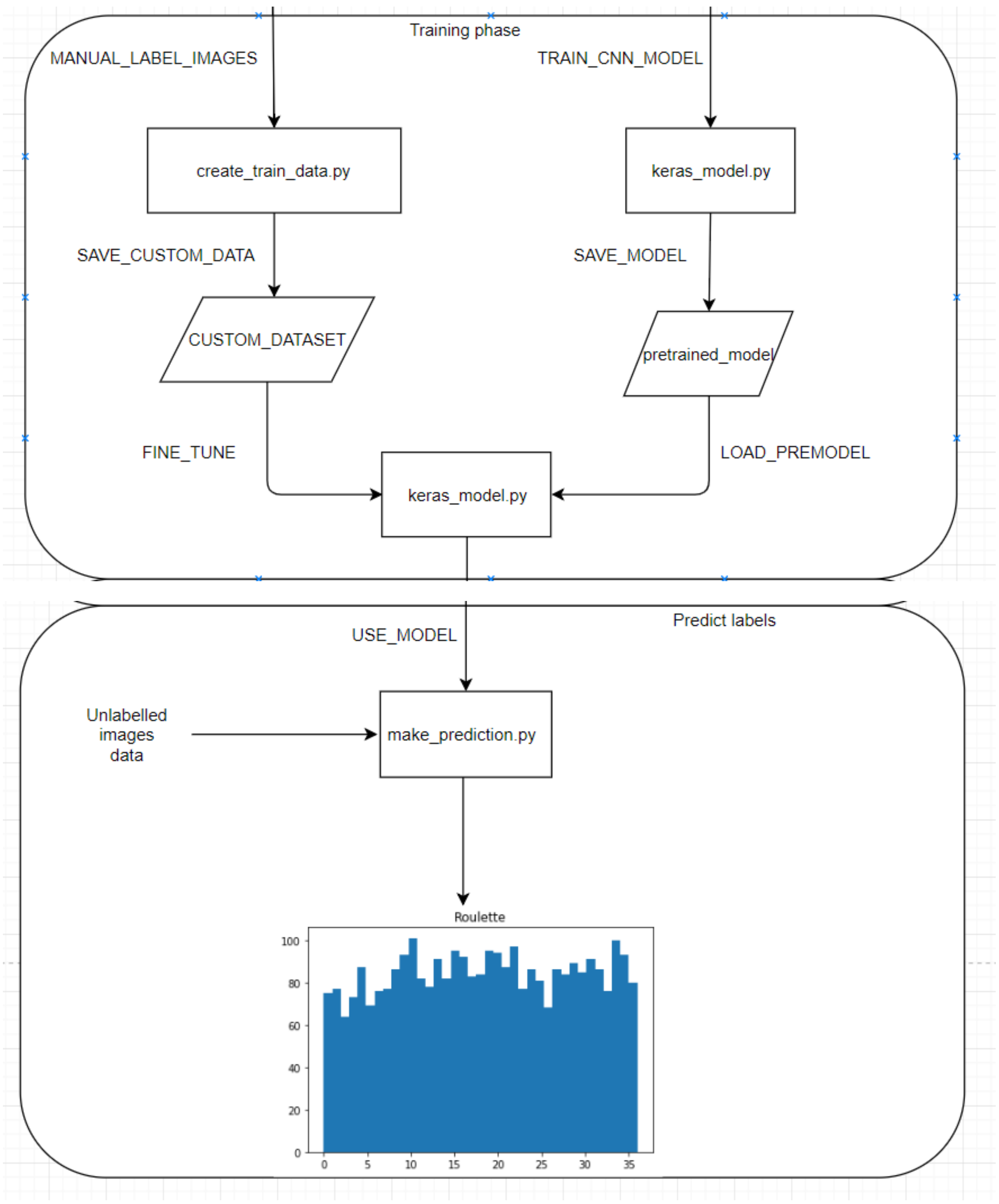


In the picture below, we can see the distribution of 3000 labelled digits. I was hoping to see some peaks there, but it wasn't the case.



Flowchart of the software:





Conclusion:

It is possible to recognize images with convolutional neural network. This model hasn't ever been in use because providing pictures to it requires lot of manual work. Utilizing RPA would be the solution for that. The robot would log-in to the webpage and take screenshots from tables of digits. For this purpose, I started to create Python based solution which utilizes selenium test automation but it's under construction. Other part that would need more research is model's accuracy if we provide digits from different data domain. I think it won’t be able to predict accurately digits from slightly different dataset. That’s why it was important to manipulate data into very similar shape. For this challenge it was enough.