האוניברסיטה
העברית
בירושלים

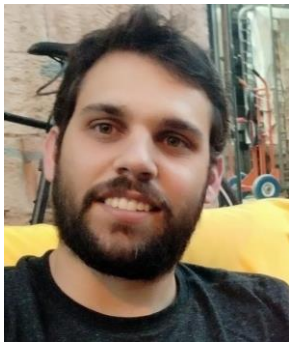
Hackathon - 3D Data Processing in Structural Biology

Puzzle Assembling

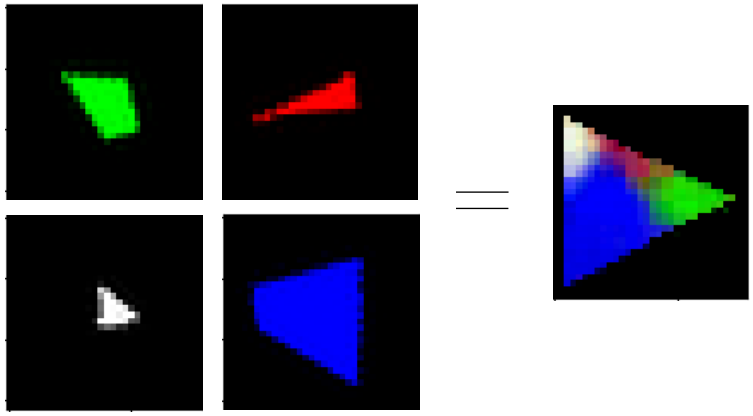
Team members:


Eden Elmaliah


Omer Dan


Rafael Horowitz


Shahar Jacob



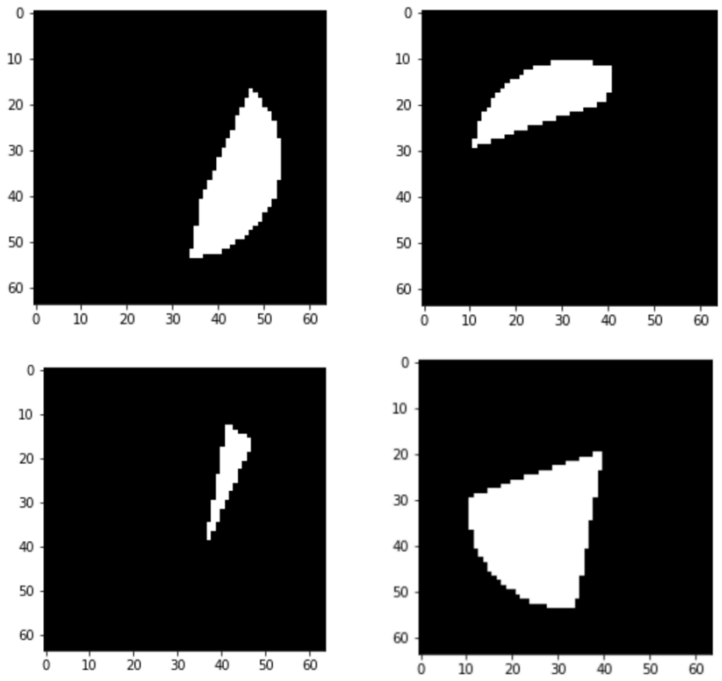
The problem:

Compose a puzzle given its pieces.

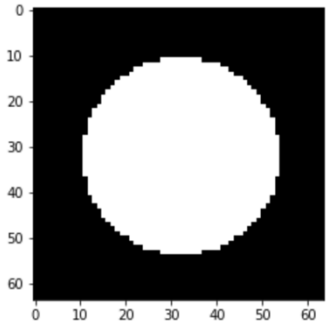
Input:

m puzzle pieces (each piece is of shape (N, N)), and their label (the label is the right way to compose the puzzle) the label shape is also (N, N). Overall, the input shape is (M, N, N).

For example, with M = 4, N = 64:



And the label is:



Working process:

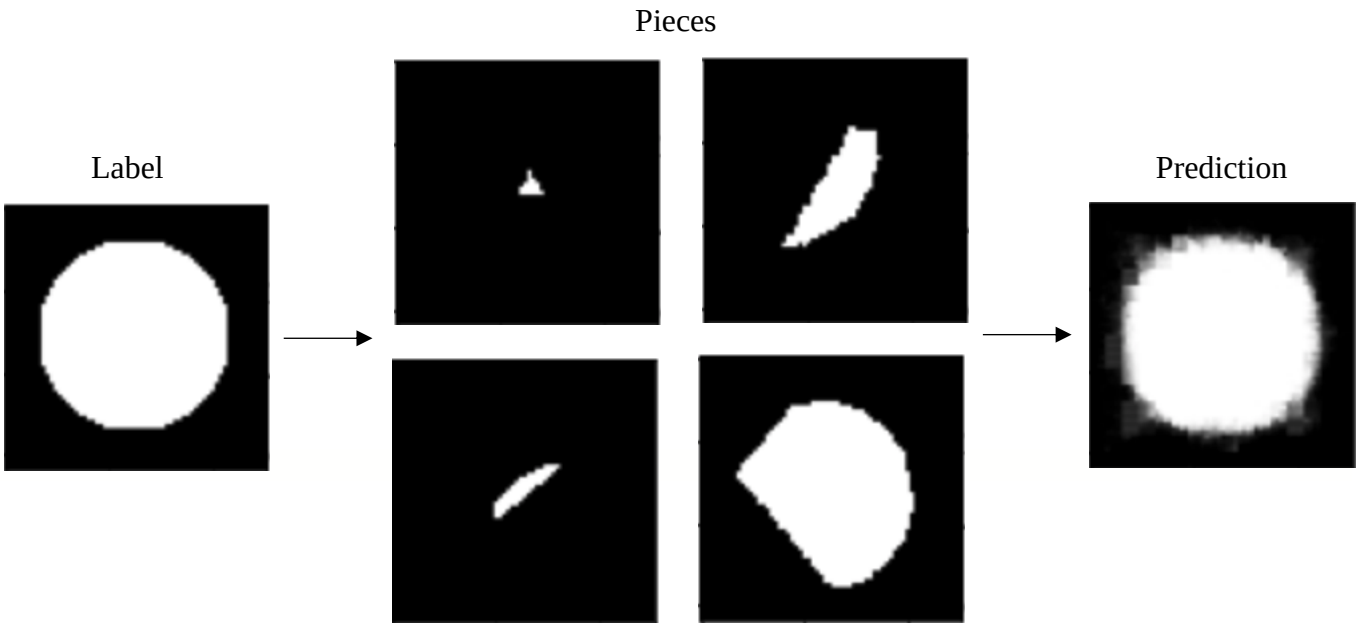
1st step: We have created images as the ones above. Each image contains one of three random shapes: triangle, rectangle, and circle, when the background is black, and the shape is white. The shapes also have a random size (but we kept it quite big).

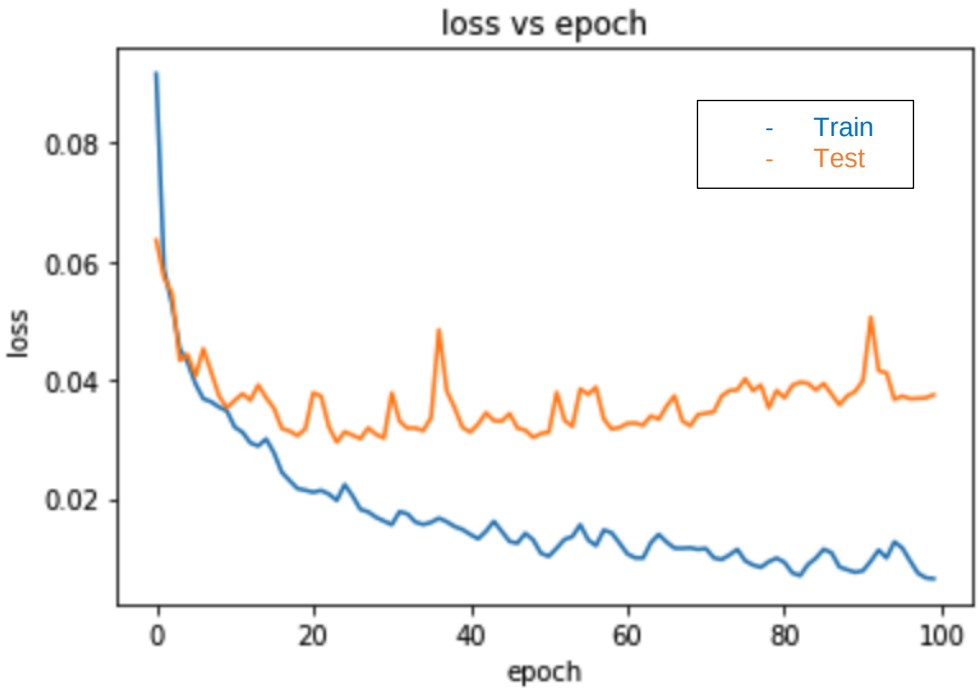
Next, each image was sliced randomly to 4 different pieces, while keeping minimum number of white pixels in each slice and centering each piece.

Each slice is of the same size of the input – NxN pixels.

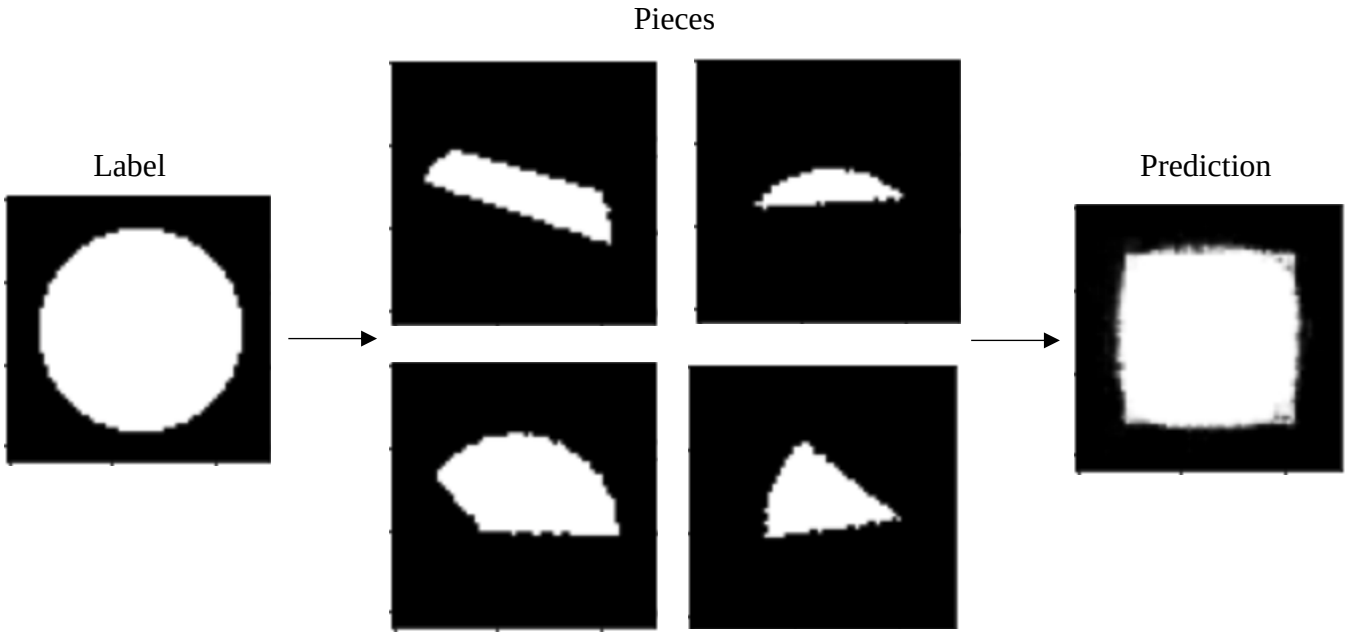
After generating the data, we trained a neural network, where the input is 4 pieces of an image and the label is the original image.

Results: We got pretty good visual results after very few epochs. We believe that the NN learned how to "guess" what kind of shape should be in the output and just constructed it in the right size.

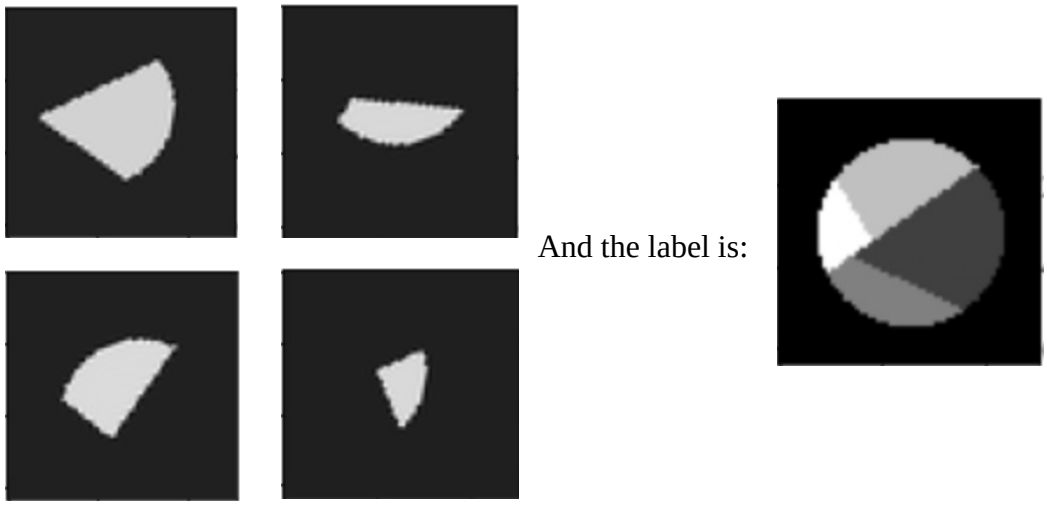


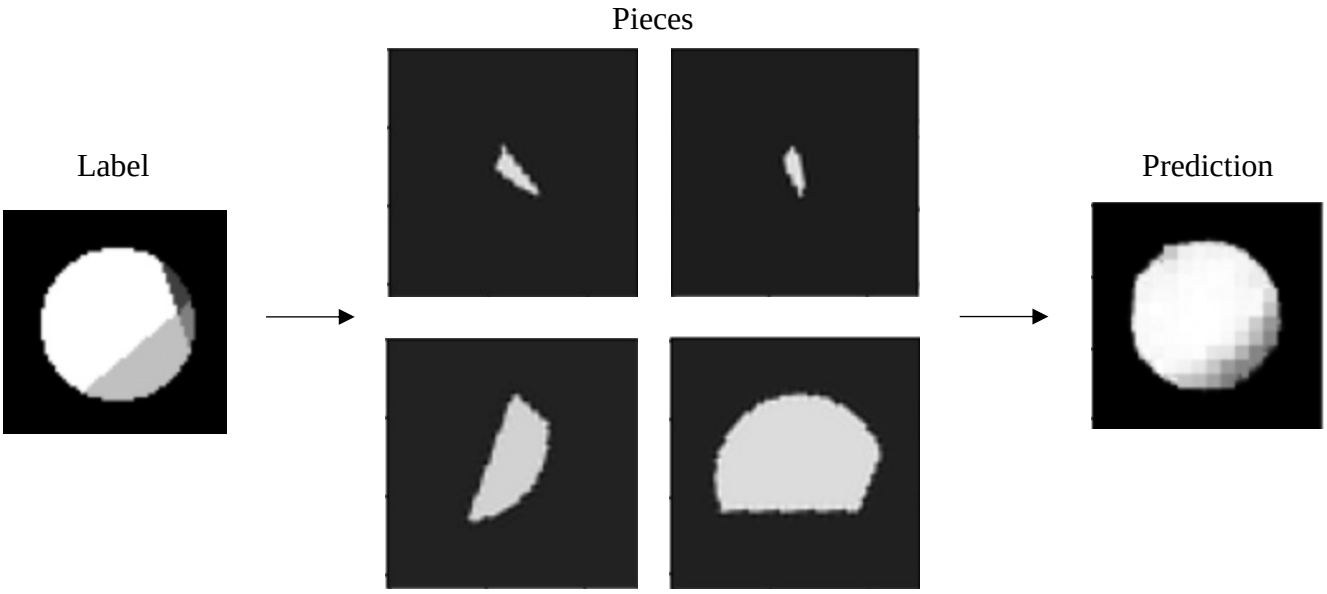


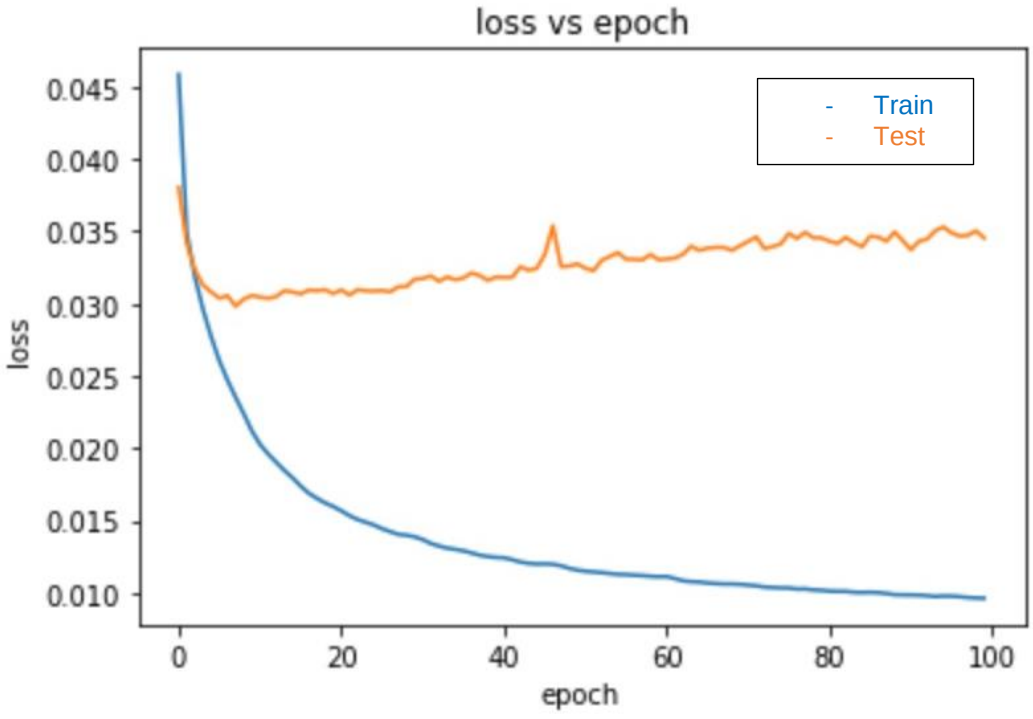
In the following example we can see that the model tried to guess is it a circle or triangle, and predicted some kind of mix:



2nd step: we added few changes to the data to prevent the case of the first step, where the NN just figured out the target shape and constructed it.
First, we painted each piece in a different shade of gray and reconstructed the original shape with the new colors. Next, each piece was centered and then rotated in a random angle.
For example:



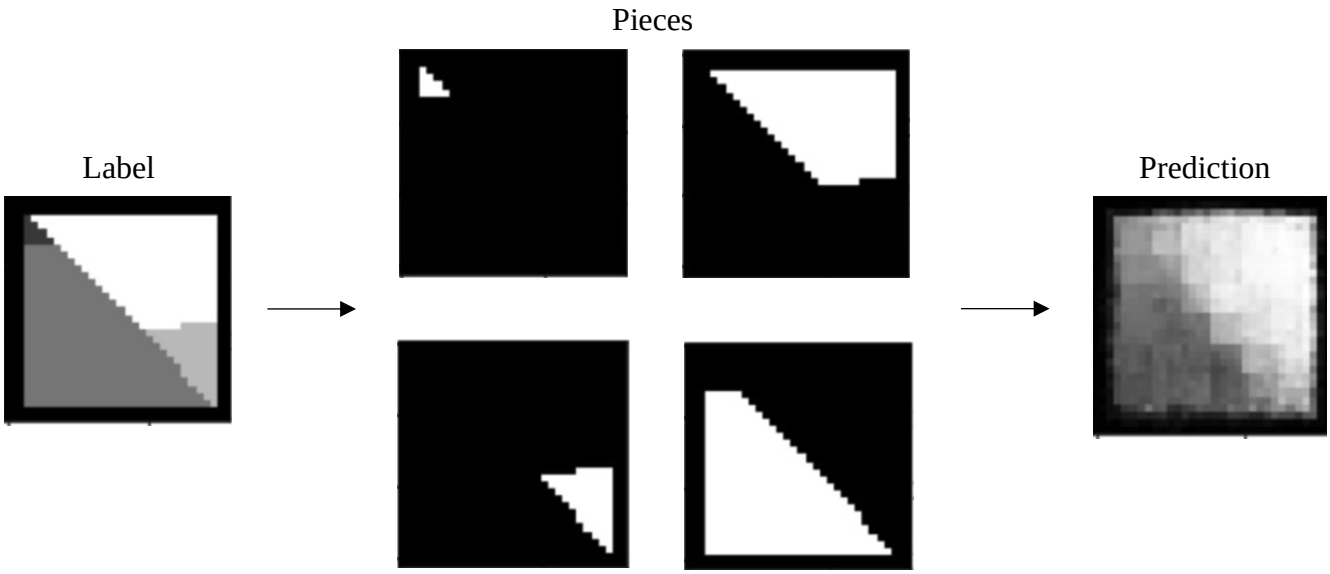


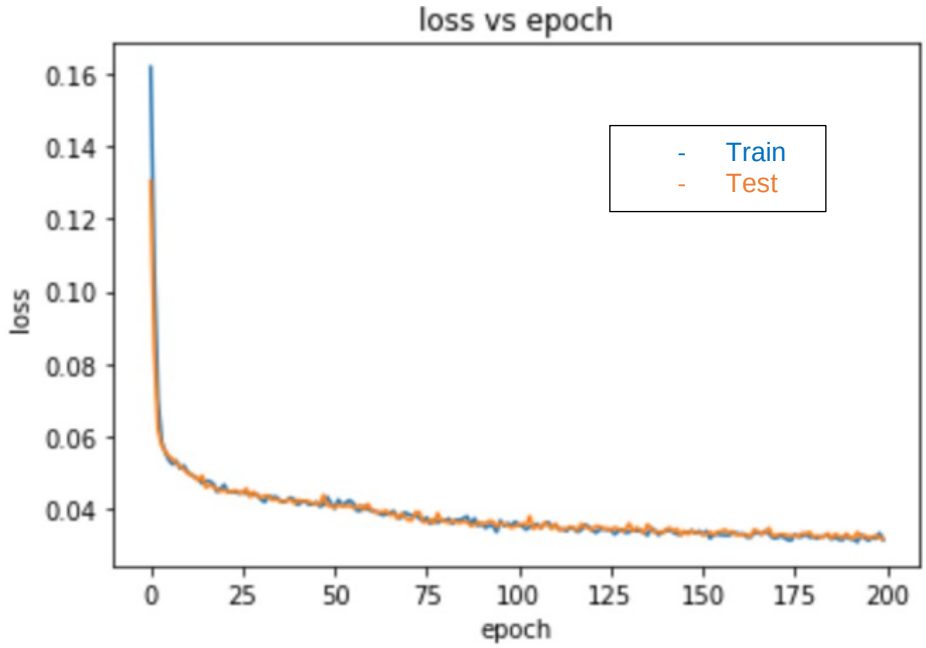


We can see here pretty big overfit to the data

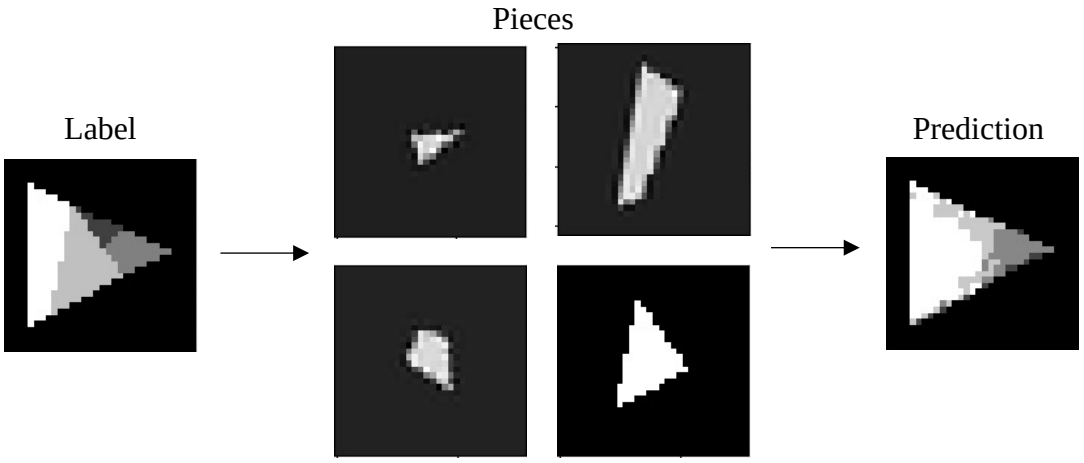
3rd step: we added the ability to create data on-the-fly, instead of creating it in advanced.
The results were much better, because this way we could prevent the overfit we had before.
Also, when the data is created this way, the training can run "forever" and can be stopped by request.

Results: the results were better but still not clear enough (the pieces were taken before their centering):

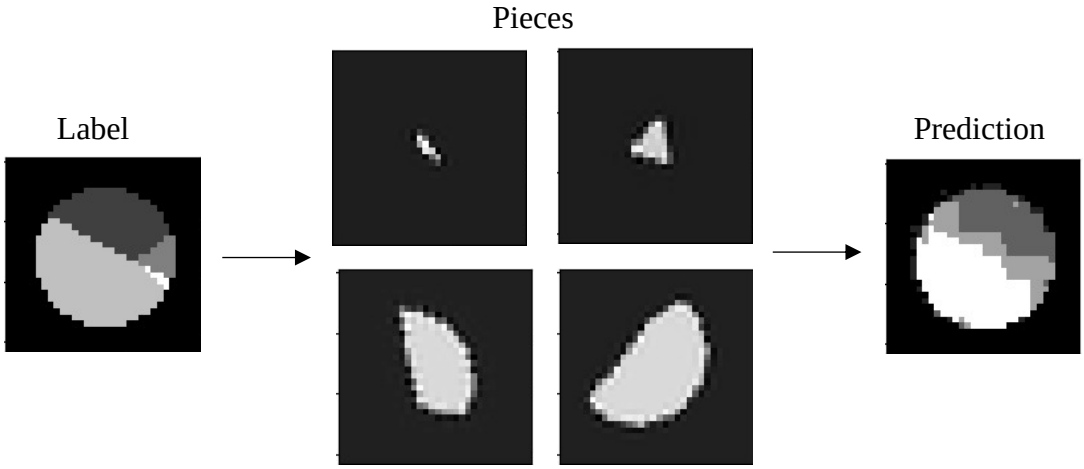




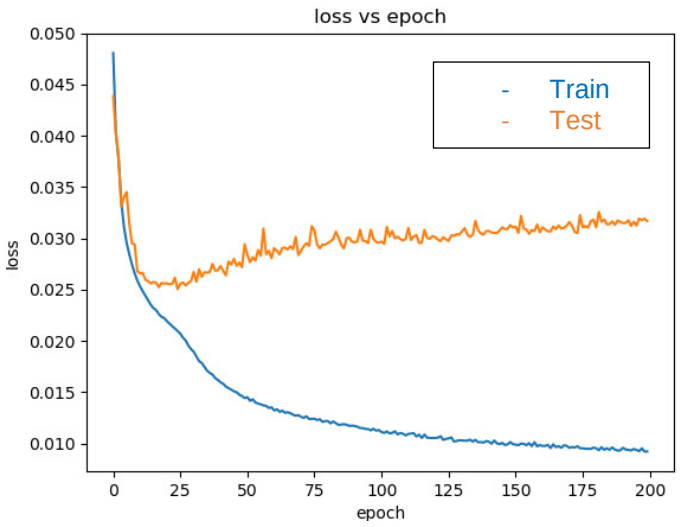
4th step: we added quantization to the prediction images, in order to get better visual results.
We worked in parallel on some steps, so the following results were taken from a run with existing data (no on-the-fly data creation). So, after 30-40 epochs we got overfit.



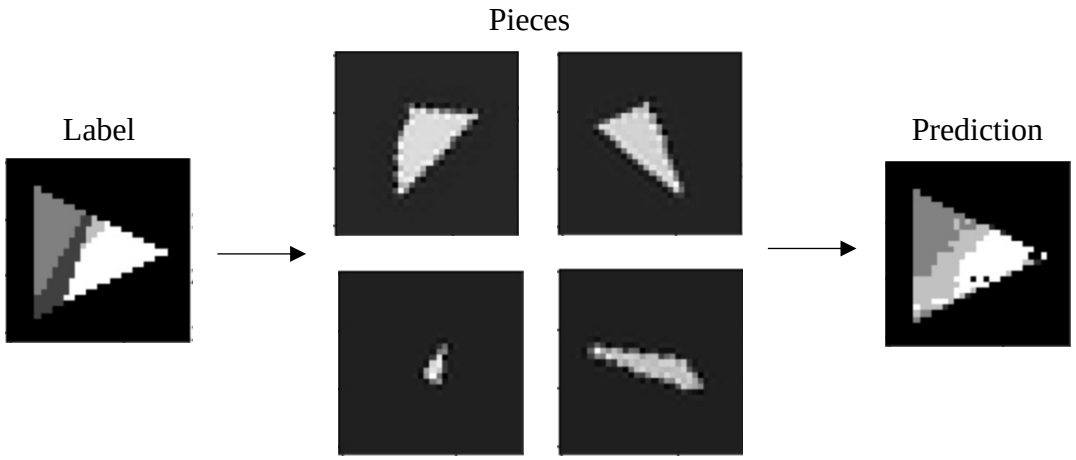
After 20 epochs

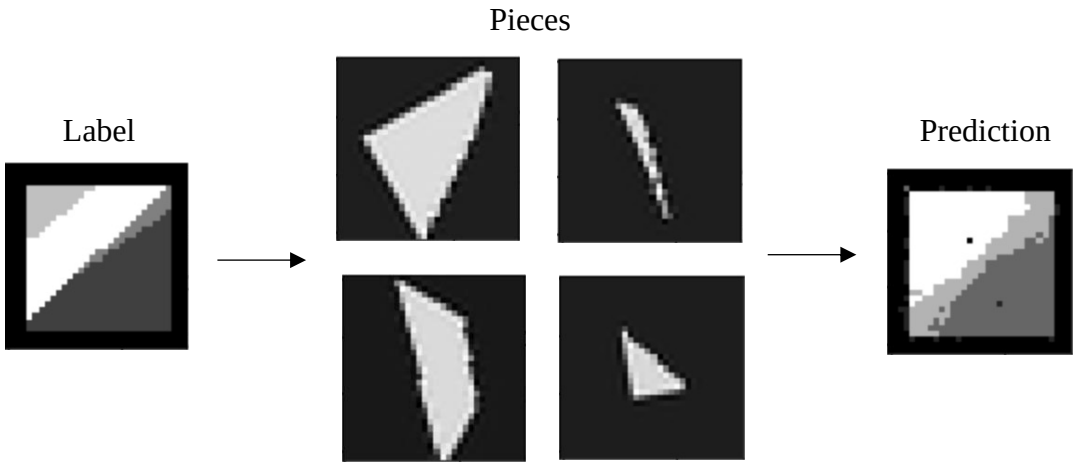


After 40 epochs

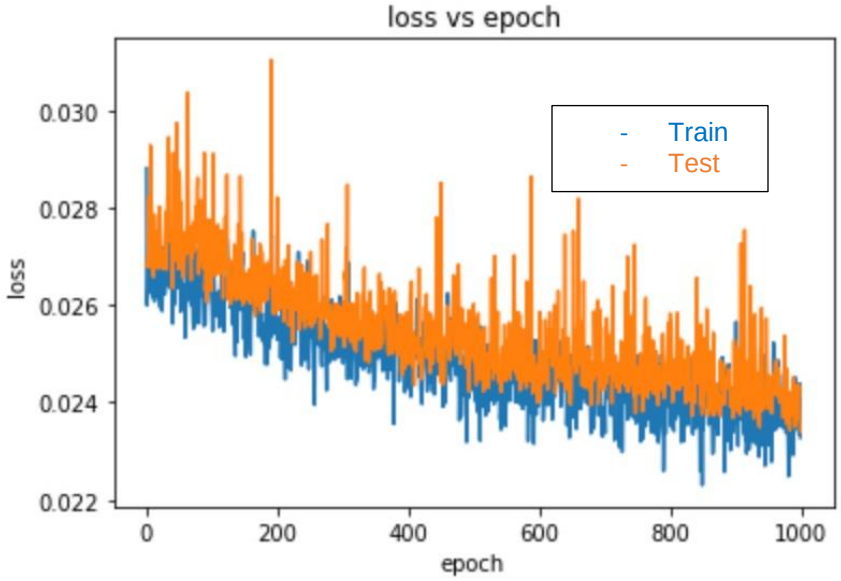


5th step: we combined two previous methods: quantization + on-the-fly data creation.
The results here are much better:



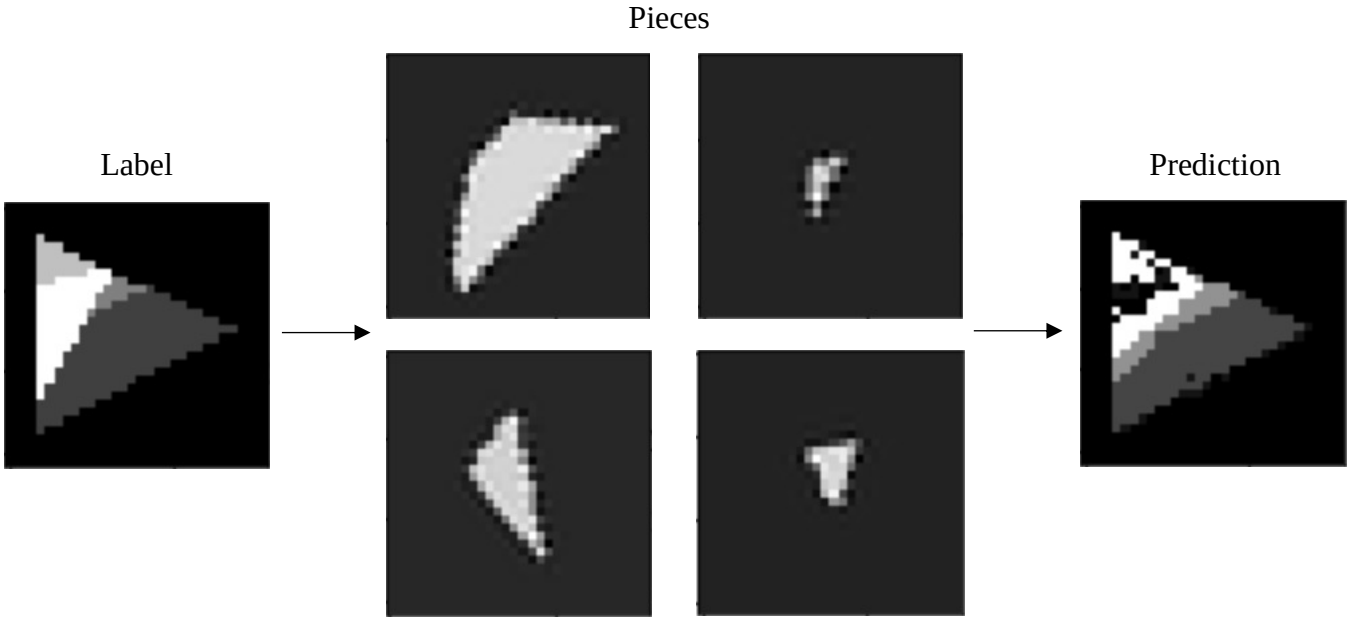


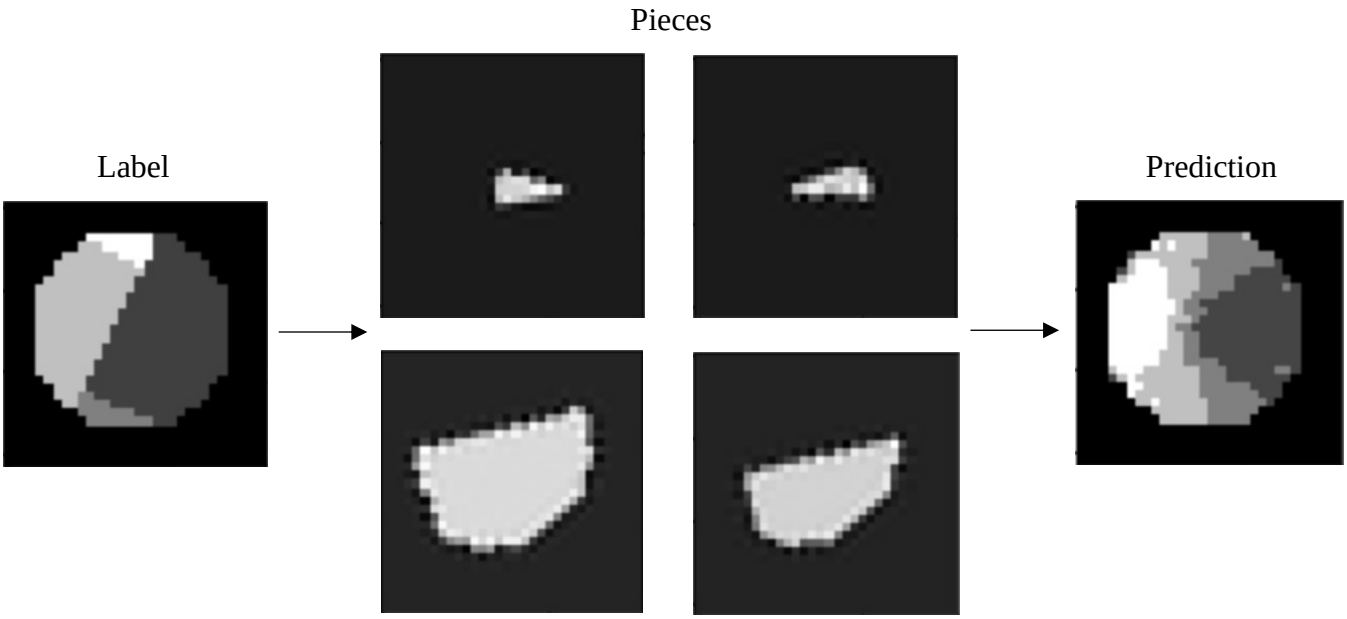
The loss after many epochs (the loss here is very small, so the oscillation are also quite small):

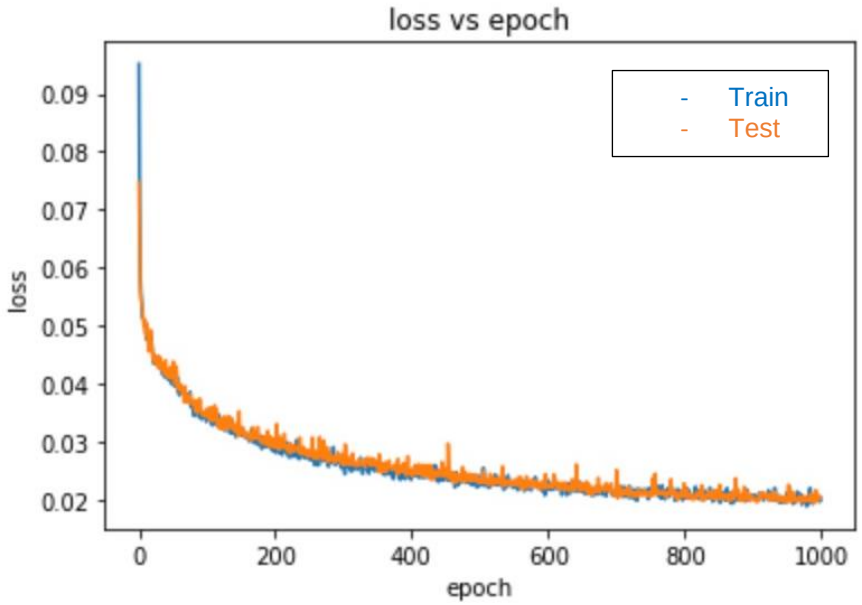


6th step: We have changed the network architecture (added an additional fully connected layer to the encoder and decoder and added another convolutional and deconvolutional layer to the encoder/decoder respectively).

Results:



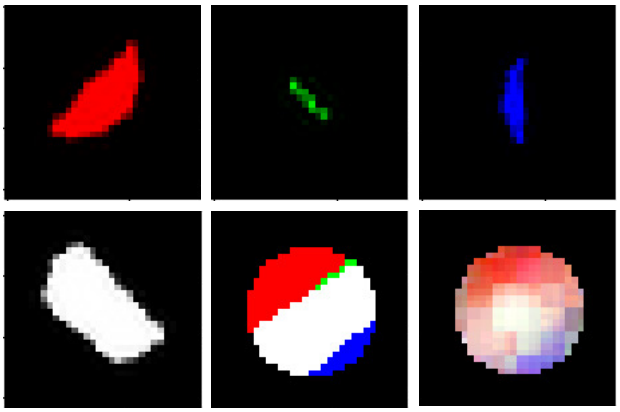


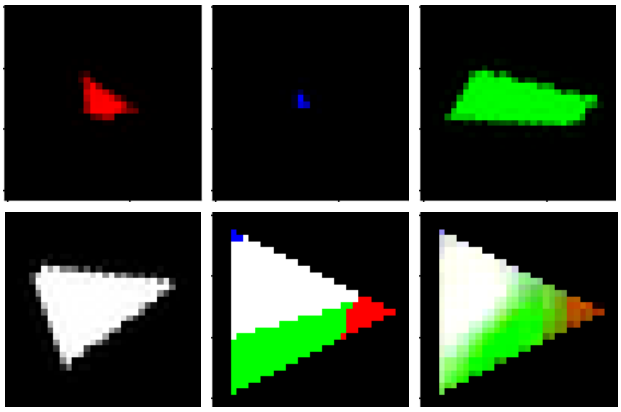


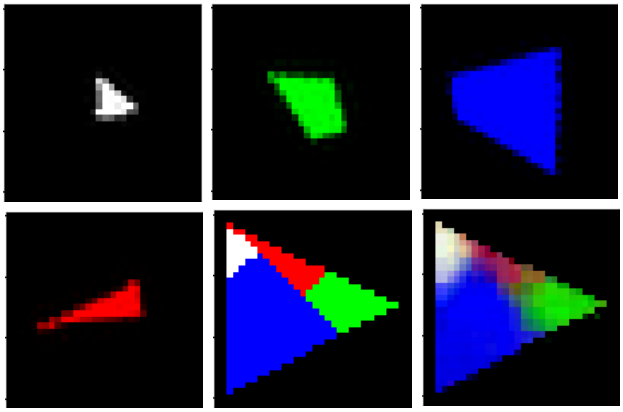
7th step: we tried to change the model, so it will output the transformation of each piece (rotation and translation). After receiving this output, we applied the transformations on each piece and combined them into one image, and the loss was computed between this combined image to the original one. We had some issues with this approach because some of the variables were not differentiable (so the back propagation failed), so we decided to drop this approach and improve the previous ones.

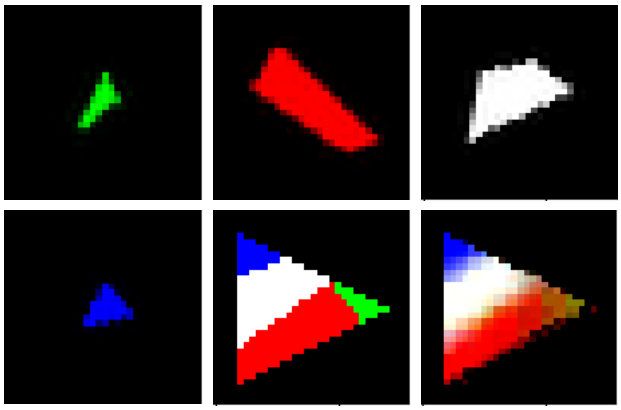
8th step: we decided to paint the pieces in RGB colors, so that the difference between them will be bigger, we didn't add quantization.

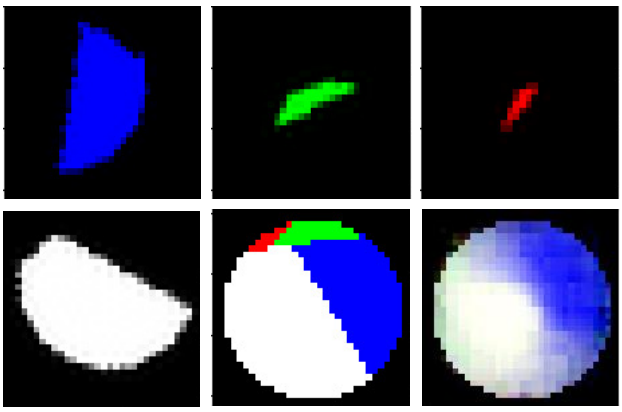
Results (the label is in the second row in the center, and the prediction is the rightmost image in the second row, all the others are the pieces):

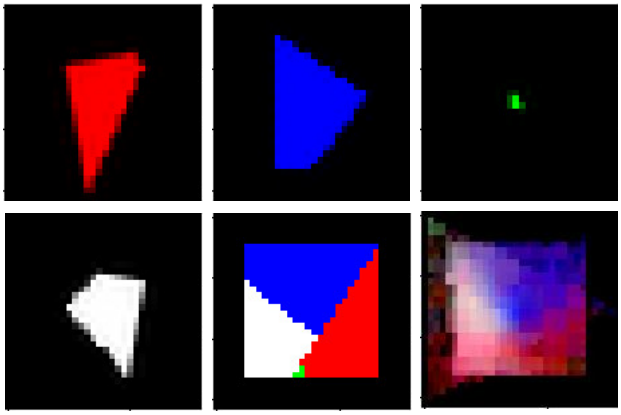






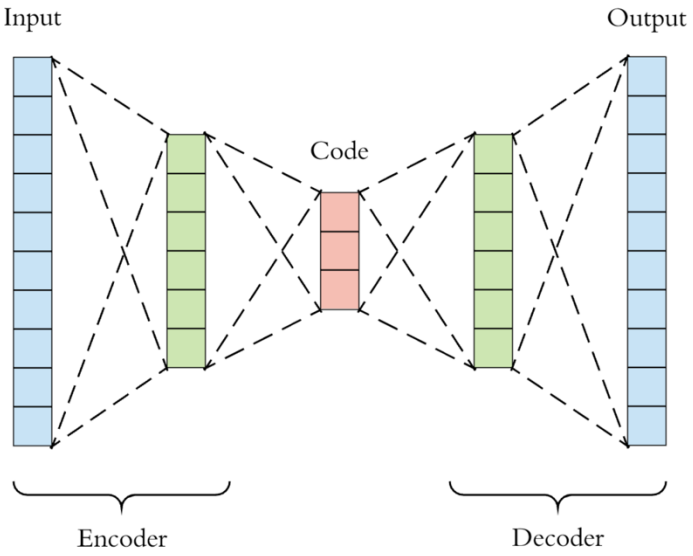






Neural network architecture:

We used an architecture of **autoencoder.** We used the fundamental ideas of this architecture but built it from scratch on our own.



In the Encoder, we used 3 convolutional layers and then 2 fully connected layers. The latent vector size is 128. In the Decoder we used 2 fully connected layers and then 3 de-convolutional layers.

From our code (inputs shape here is 4x32x32):

```
class Encoder(Model):
    def __init__(self):
        super(Encoder, self).__init__()
        self.d1 = Dense(512, activation='relu')
        self.d2 = Dense(2048, activation='relu')
        self.reshape = Reshape((8, 8, 32))
        self.deconv1 = Conv2DTranspose(16, 2, activation='relu', strides=2)
        self.deconv2 = Conv2DTranspose(1, 2, activation='relu', strides=2)

class Decoder(Model):
    def __init__(self):
        super(Decoder, self).__init__()
        self.reshape = Reshape((32, 32, 4))
        self.conv1 = Conv2D(16, 3, activation='relu', strides=2, padding='same')
        self.conv2 = Conv2D(32, 3, activation='relu', strides=2, padding='same')
        self.conv3 = Conv2D(64, 3, activation='relu', strides=2, padding='same')
        self.flatten = Flatten()
        self.d1 = Dense(512, activation='relu')
        self.d2 = Dense(128, activation='relu')
```

Train our model on your own data

To train our model on different data, the pieces and their labels should be of size 32x32 where the backgroung is black and the desired shapes are grayscale.
So in order to use protein images, you'll need to use a code to project them from 3D to 2D images and quantize it to a grayscale image (you can also quantize to 2 colors, so the images will be binary).
The network expect to receive 4 images to reconstruct, thus an input of four 32x32 graysclae images, with label of 32x32 grayscale image which represents the correct reconstruction of the images.
To input images of other size, there is a need to change the architecture of the neural network itself, which results with an extra overhead of redesigning the network.

Future work:

Improving the data creation: because the data is created on-the-fly, it can be inifinit. One can add more complex shapes and sizes in order to increase the diversity of the data.
This way, an overfit situation can be prevented, which is a very powerful capability.
Also, when increasing the diversity of the data, the network is learning how to work without learning the shapes themselves.
We believe that if someone with biological bakground improve the data creation (the shapes kinds and its randomness), the model will have a much better fit to biological problems.

Comments:

- In all the graphs of the steps with the on-the-fly data creation, the test and train curves are pretty much the same. The reason is that the model doesn't see any sample twice, so there is no difference between test and train run and it was more of a sanity check for us.

- In the steps mentioned above, we want to point out that the epochs are not really epochs, from the same reason that we don't really have a dataset to run the model on and divide to epochs. So, each epoch is more of an iteration of the training.

- The training takes longer when creating data on-the-fly, because each iteration requires to create images, slice them, rotate and translate the slices etc. But the outcome is much better. Also, there is no limit for training, you can "train forever".

- **Our Git repository:** https://github.cs.huji.ac.il/shahar-jacob/3D-bio-hackathon