Biologically inspired artificial intelligence

Fruit recognition

Mikołaj Habarta
Franciszek Krawczyk

1. Main goals

Main goal of the project was to create a convolutional neural network that will be able to distinguish an apple and an orange on the picture.

2. Theory and tools

As recently we can observe the growth in popularity of the convolutional neural networks – a model that is supposed to do well on image recognition tasks seems to to well in non-image-related areas. Our model consists of 3 convolutional layers and 2 fully-connected, classical layers. The first convolutional layer takes a 3-channel input, as the network is working with colored images and they represent RGB channels.

```
self.conv1=nn.Conv2d(3,32,5)
self.conv2=nn.Conv2d(32,64,5)
self.conv3=nn.Conv2d(64,128,5)
```

1: Convolutional layers

```
self.fc1= nn.Linear(self._to_linear,248)
self.fc2 = nn.Linear(248,2)
```

Fully-connected layers

As for the tools, we decided to use PyTorch library. It's relatively easy to learn, and it's integrated with NumPy – it has build-in conversion between PyTorch types (such as tensors) and NumPy types (numpy Arrays). It also enables the code to be run with CUDA device, meaning we could use our GPU power to help train and validation of our network, which speeds up the process quite drastically,

3. Implementation

Data preparation

The data contains 6024 pictures of apples and oranges (3012 of each) in 320 x 258 resolution. To prepare our data for the neural network we have to save it to the file in the format that will be usable for our neural network. For this we can think of our dataset as of 6024 tuples – each one containing an image and a one-hot vector that corresponds to the image. The one-hot vector is just a vector of values [0,1] or [1,0], where to „1" means which fruit the image contains. The image is scaled down to 80 x 65 resolution in order to save space and speed up calculations. Therefore every image is stored as an numpy array of size (3,80,65) where 3 stands for RGB channels. All the tuples create a list.

There are the same amount of pictures of apples and oranges, therefore our dataset will be balanced, however it is import to shuffle the data before feeding it to neural network.

Below you can see resizing the image (using cv2) and adding it to the list. np.array(img) is a numpy array containing an image, np.eye(2)[…] returns [0,1] vector if the label is apple and [1,0] vector if the label is orange.

```
img=cv2.resize(img,(x_size,y_size))
self.training_data.append([ np.array(img),np.eye(2)[self.labels[label]]])
```

Neural network

At the beginning the data is being split into two separate sets, one for training and the other for validation. The sets are then split into set of images(first in the tuple) and set of one-hot vectors(second in the tuple). The former will be referred later as the train_X or test_X, the latter as test_T or train_Y .

```
train_X = X[-training_size:]
train_Y = Y[-training_size:]
print("training set:",len(train_X))
test_X = X[:-training_size]
test_Y = Y[:-training_size]
print("testing set:",len(test_X))
```

The network is trained and tested with 3 functions: `forward_pass`, `test` and `train`.

```
def train():
    MODEL_NAME = f"model-{int(time.time())}"
    BATCH_SIZE=150
    EPOCHS = 15
    with open("model(150,15,).log","a") as f:
        for epoch in range(EPOCHS):
            for i in tqdm(range(0,len(train_X),BATCH_SIZE)):
                batch_X=train_X[i:i+BATCH_SIZE].view(-1,3,x_size,y_size).to(device)
                batch_Y =train_Y[i:i+BATCH_SIZE].to(device)
                acc,loss=forward_pass(batch_X,batch_Y,train=True)
                if i % 35 == 0:
                    val_acc,val_loss = test(size=54)
                    f.write(f"{MODEL_NAME},{round(time.time(),3)},{epoch},{round(float(val_acc),3)},{round(float(val_loss),4)},{round(float(acc),3)},{round(float(loss),4)}\n")
```

The train function is responsible for training, testing, and recording the results of the neural network. It divides the input data (in train_X) into the batches, and then trains the network with those batches of data (by invoking forward pass). Because testing every batch would be every resource-consuming, we test only every 35 batches. After testing we write down the results, that is: validation accuracy, validation loss, training accuracy and training loss to the log file.

```
def forward_pass(X,Y,train=False):
    if train:
        net.zero_grad()
    outputs = net(X)
    matches = [torch.argmax(i)==torch.argmax(j) for i,j in zip(outputs,Y)]
    accuracy = matches.count(True)/len(matches)
    loss = loss_function(outputs,Y)
    if train:
        loss.backward()
        optimizer.step()
    return accuracy,loss
```

The forward_pass function runs the data through the network and trains the network if the flag „train” is set to true. It returns accuracy and loss values for the data it took.

```
def test(size=35):
    random_start = np.random.randint(len(test_X)-size)
    X,Y = test_X[random_start:random_start+size],test_Y[random_start:random_start+size]
    with torch.no_grad():
        vall_acc, val_loss = forward_pass(X.view(-1,3,x_size,y_size).to(device),Y.to(device))
    return vall_acc, val_loss
```

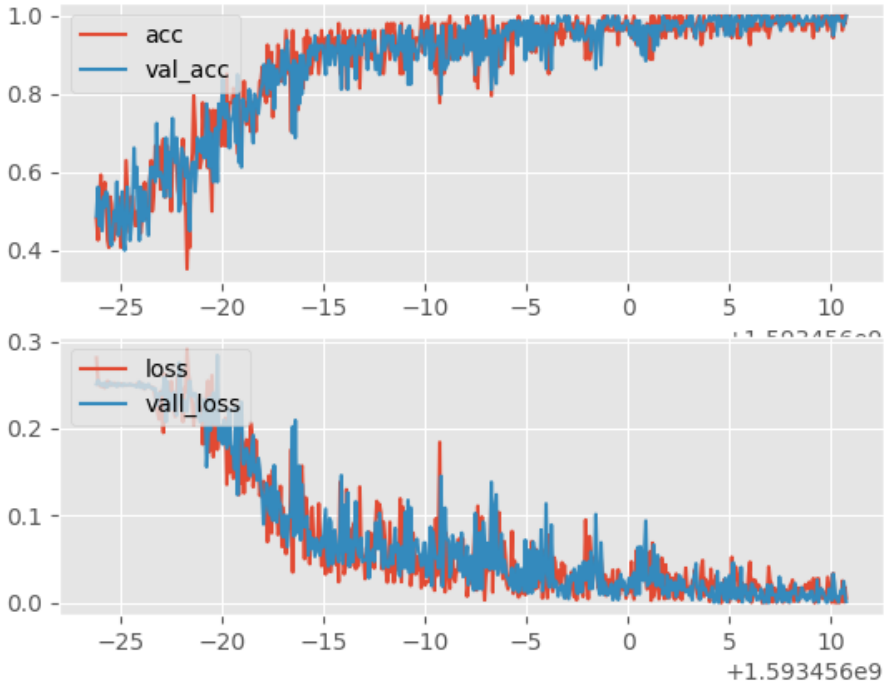
The test function takes a random slice of a given length from a validation data and runs it through the network without training it. It returns the validation accuracy and validation loss.

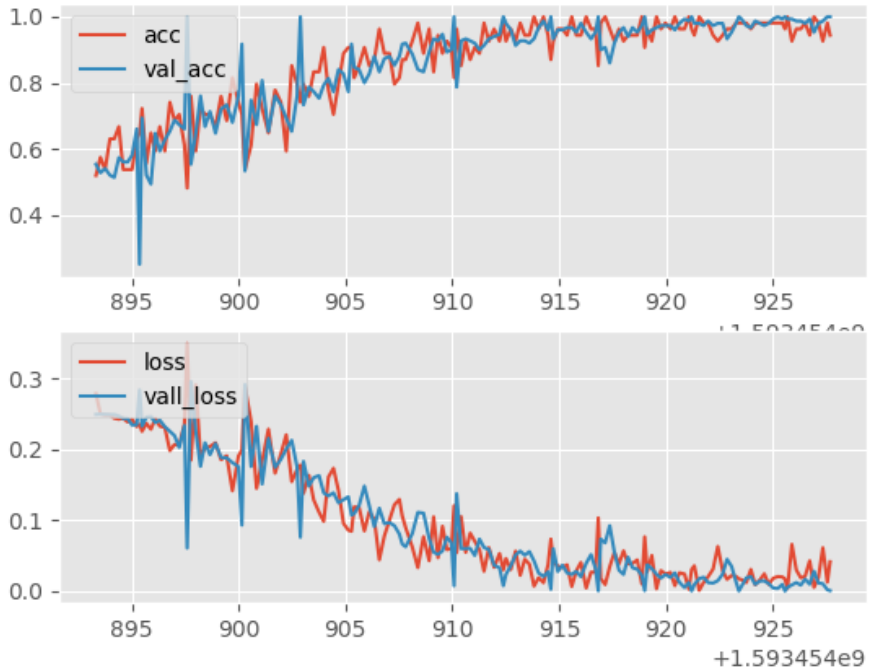
4.a Results and graphs

PLESE NOTE: The colors in legend are wrong, it, the red ones are the validation test and the blue ones are the training

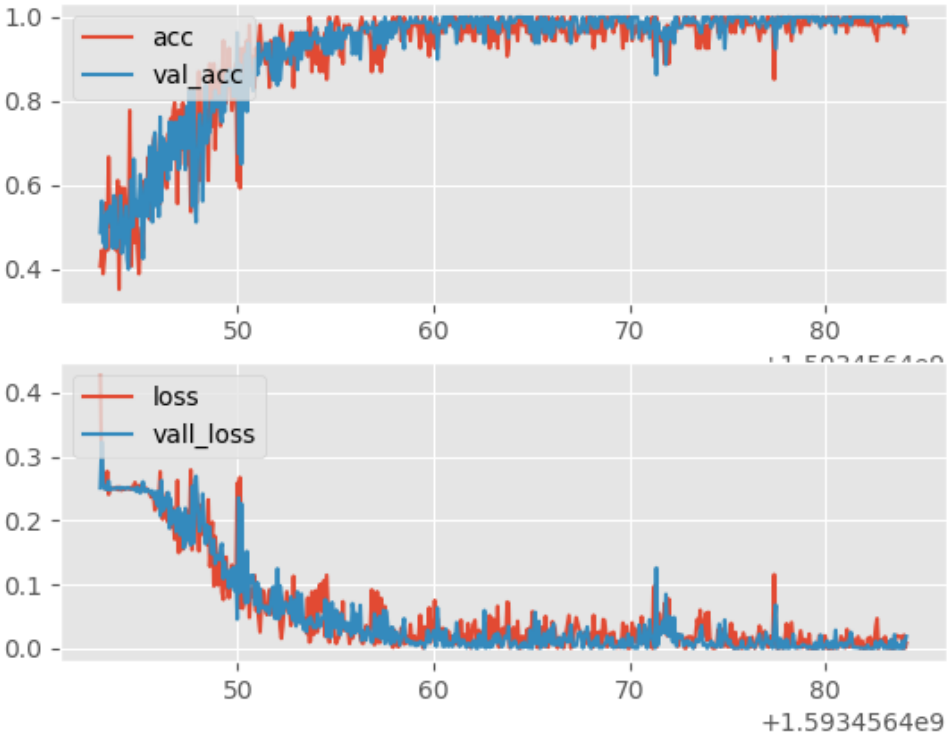
Below are shown graph the illustrate how the neural network behaves depending on the number on epochs/batches. Red values correspond to the off-sample test, blue one to the in-sample test.

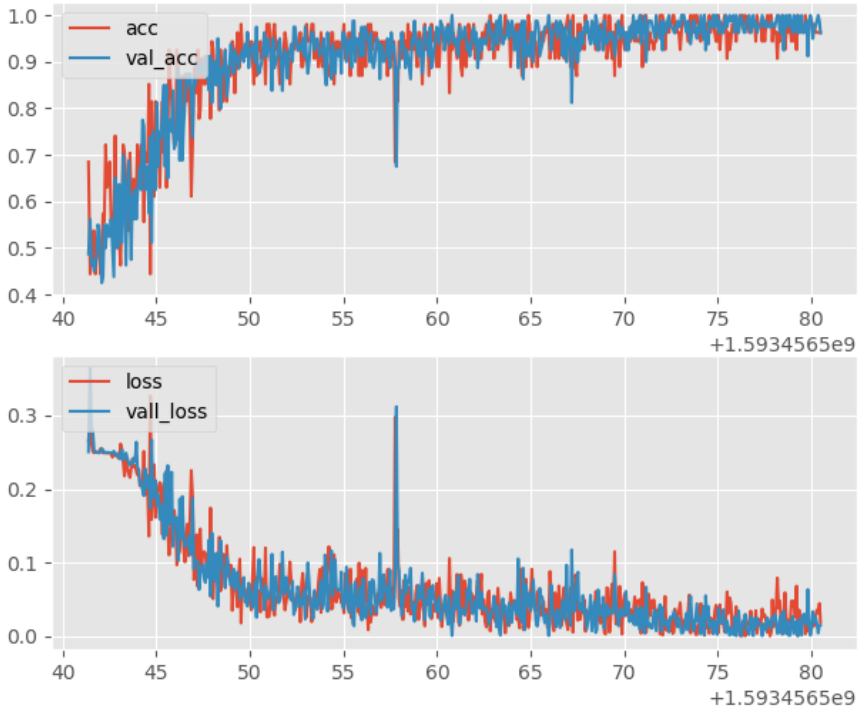
Below 15 epochs, batch size is 80 and 150, respectively:

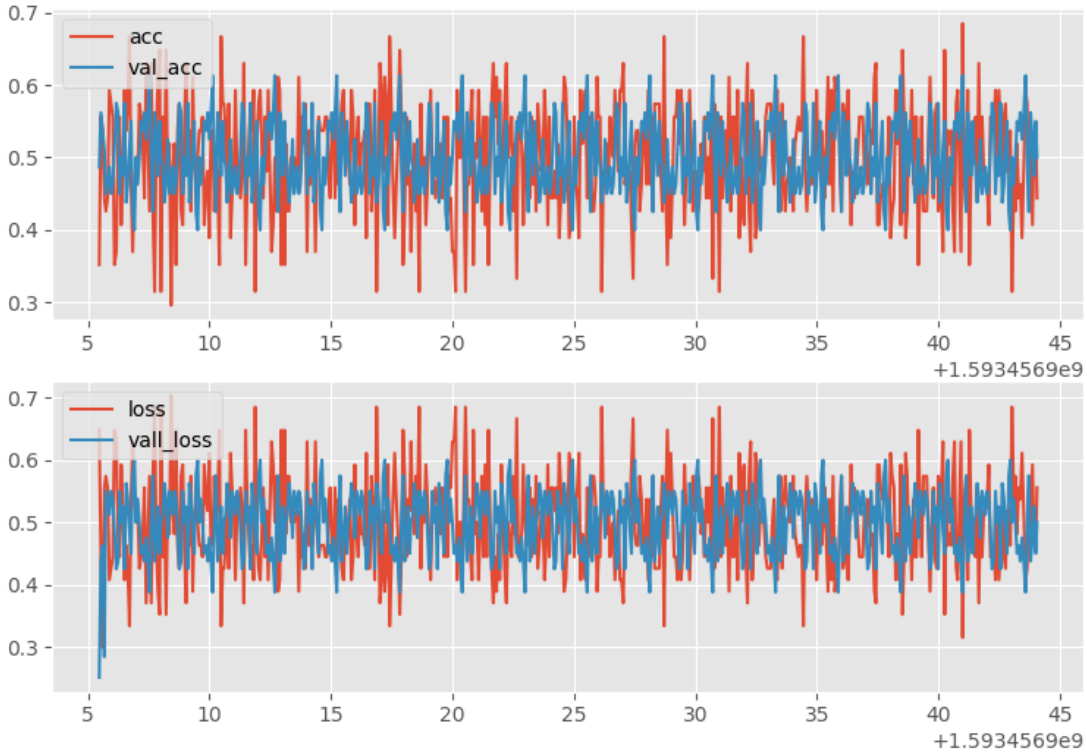




Previous graphs were made with learning rate 0.001
5 epochs, batch size 80, learning rate 0.002, 0.003, 0.004, respectively

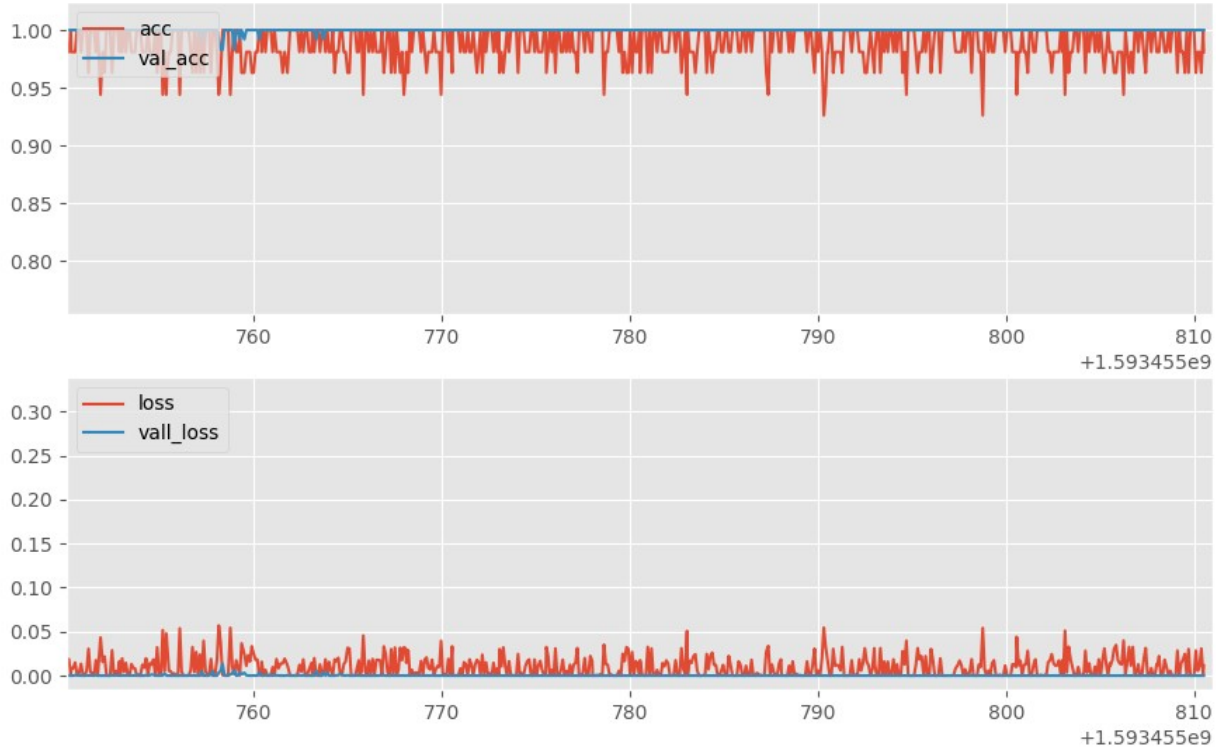






Learning rate decides how “step” the gradient will be. Greater values cause more rapid development, however the cost of this is more instability. Learning rate 0.001, batch size 120, 150 epochs

In the last example we can see an example of over-fitting. With large number of epochs the network is no linger recognizing features of the images and it is starting to memorize all the data. The accuracy on the in-sample data hits 100% (blue line) as the network memorized all the images, but the validation accuracy cannot raise as the network doesn't learn of those images.



4.b The data

The data is taken form
https://www.kaggle.com/chrisfilo/fruit-recognition
The original dataset contains 44406 pictures of different fruits, we are using only a small part of it.

5. Conclusion

We manged to achieve the goal of this project. Our network is consistently hitting more than 99% accuracy, which is a great result. During the project we came to conclusion, that the biggest problem with the neural network is to find the data to feed it with. Once it was done, learning new technology, along with a new language was a very developing experience.