```
import os
import cv2
import numpy as np
from tqdm import tqdm
import torchvision
from torchvision import transforms


REBUILD_DATA = False
IMG_SIZE = 300

class SnakesCucumbers():
   IMG_SIZE = 300
   SNAKES = "GreenImages/snake"
   CUCUMBERS = "GreenImages/cucumber"
   LABELS = {SNAKES: 0, CUCUMBERS: 1}
   training_data = []
   snakecount = 0
   cucumbercount = 0


   def make_training_data(self):
      for label in self.LABELS:
         print(label)
         for f in tqdm(os.listdir(label)):
            #f je samo ime fajla
            try:
               transform =
transforms.Compose([transforms.ToPILImage(),transforms.RandomHorizontalFlip(p=0.5),transforms.
RandomVerticalFlip(p=0.5)])
               path = os.path.join(label,f)
               #img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
               img = cv2.imread(path)
               img = cv2.resize(img, (self.IMG_SIZE, self.IMG_SIZE))
               img2 = transform(img)
               self.training_data.append([np.array(img),
                                 np.eye(2)[self.LABELS[label]]])
               self.training_data.append([np.array(img2),
                                 np.eye(2)[self.LABELS[label]]])
               if label == self.SNAKES:
                  self.snakecount += 1
               elif label == self.CUCUMBERS:
                  self.cucumbercount += 1
            except Exception as e:
               print(str(e))
               pass

      np.random.shuffle(self.training_data)
```

```
        np.save("green_data.npy", self.training_data)
        print("Snakes", self.snakecount)
        print("Cucumbers", self.cucumbercount)

if REBUILD_DATA:
    SnakesVSCucumbers = SnakesCucumbers()
    SnakesVSCucumbers.make_training_data()
```

Radi ustede vremena nakon sto formatiramo slike rebuild data se stavlja na false. Ukoliko menjamo svojstva slika, potrebno je staviti ga na true. Iznad je klasa koja slike sa specifiranog patha obradi I sacuva u .npy formatu za kasniju upotrebu. U ovom slucaju ih stavljamo da budu oblika 300x300, I imamo pipeline transformacija koji su composed u promenjivu transform koju kasnije stavljamo na svaku sliku. Na ovaj nacin vestacki prosirujemo dataset da sadrzi po jednu verziju originalne I jednu verziju editovane verzije slike.

```
training_data = np.load("green_data.npy", allow_pickle = True)
print(len(training_data))
```

Ucitamo data-u, I odsampamo kolicinu na ekran radi provere.

```
import torch

index = 1
print(training_data[index][1])
import matplotlib.pyplot as plt
#plt.imshow(training_data[index][0], cmap = "gray")
plt.imshow(training_data[index][0])
plt.show()
```

Pokretanjem ove celije cemo preko matplotlib-a na ekranu nacrtati sliku i odstampati njenu oznaku. Ovo je opet radi provere i nema veze sa radom same mreze.

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, 3)
        self.conv2 = nn.Conv2d(16, 32, 3)
        self.conv3 = nn.Conv2d(32, 64, 3)

        x = torch.randn(IMG_SIZE,IMG_SIZE, 3).view(-1,3,IMG_SIZE,IMG_SIZE)
        self._to_linear_ = None
        self.convs(x)
```

```
        self.fc1 = nn.Linear( self._to_linear_, 128)
        self.fc2 = nn.Linear(128, 2)

    def convs(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)),(2,2))
        x = F.max_pool2d(F.relu(self.conv2(x)),(2,2))
        x = F.max_pool2d(F.relu(self.conv3(x)),(2,2))

        #print(x[0].shape)

        if self._to_linear_ is None:
            self._to_linear_ = x[0].shape[0]*x[0].shape[1]*x[0].shape[2]
        return x

    def forward(self, x):
        x = self.convs(x)
        x = x.view(-1, self._to_linear_)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.softmax(x, dim = 1)

net = Net()
```

Definisemo izgled nase mreze. Za convulational nn je potrebno da poslednji slojevi budu fully connected, medjutim ne znamo kolikih ce dimenzija biti podaci nakon sto budu izbaceni iz poslednjeg conv sloja. Postoja formula kojom se to moze izracunati, medjutim to bi znacilo da moramo rucno sve menjati svaki put kad zelimo da promenimo broj slojeva ili neurona. Umesto toga je dodata funkcija convs(self,x), koja ce datu sliku provuci kroz conv slojeve. Ovo koristimo kao deo same forward(self, x) funkcije, ali I da kroz conv slojeve provucemo nasumicno generisanu sliku kako bi smo videli njene dimenzije – koje onda postavljamo kao velicinu ulaznog parametra fully connected layer-a.

```
import torch.optim as optim

x = torch.Tensor([i[0] for i in training_data]).view(-1,3,IMG_SIZE,IMG_SIZE)
x = x/255.0
y = torch.Tensor([i[1] for i in training_data])

VAL_PCT = 0.1
val_size = int(len(x)*VAL_PCT)

train_x = x[:-val_size]
train_y = y[:-val_size]

test_x = x[-val_size:]
```

```
test_y = y[-val_size:]

print(len(train_x), len(test_x))
```

Biramo procenat podataka koji zelimo da ostavimo za testiranje. Pixeli na slici imaju vrednost izmedju 1 I 255, te ih delimo sa 255 kako bi ih stavili u domet od 0 do 1 i olaksali obradu podataka. Niz delimo klasicno python-ski.

```
if torch.cuda.is_available():
    device = torch.device("cuda:0")
    print("Running on the GPU")
else:
    device = torch.device("cpu")
    print("Running on the CPU")
```

Zelimo mrezu da treniramo na grafickoj kartici jer je ona ona napravljena da vrsi puno malih operacija – generisanje poligona u 3d grafici, sto je dosta slicno popravljanju gomile weights I biases u nasoj mrezi, kao I obrade pixela iz slike. Ovim kodom proveramo da li imamo pristup GPU-u.

```
net.to(device)
```

Stavljamo mrezu na GPU. U kasnijem kodu cemo na GPU prebaciti I nase podake jer mreza ne moze da ih vidi ako nisu na istom mestu.

```
import time
optimizer = optim.Adam(net.parameters(), lr=0.001)
#optimizer = optim.SGD(net.parameters(), lr = 0.003)
loss_function = nn.MSELoss()

def fwd_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)]
    acc = matches.count(True)/len(matches)
    loss = loss_function(outputs, y)

    if train:
        loss.backward()
        optimizer.step()
    return acc, loss
```

Importujemo optimizatore I loss funkciju koju cemo koristiti. Fwd_pass za sliku x i “resenje” tj. obelezje te slike y racuna loss i koliko je mreza bila blizu tacnog odgovora. Opciono, ukoliko train stavimo na true, mreza ce popraviti svoje parametre u skladu sa optimizatorom I loss-om.

```
def test(size = 8):

   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():
      val_acc, val_loss = fwd_pass(x.view(-1,3,IMG_SIZE,IMG_SIZE).to(device), y.to(device))
   return val_acc, val_loss

val_acc, val_loss = test(size=8)
print(val_acc, val_loss)
```

Poziva fwd_pass od gore, ali sa test data-om, i sa size nasumicnih slika iz test data-e. Sluzi da bi smo proveravali da li dolazi do overfit-ovanja.

```
import time

MODEL_NAME = f"model-{int(time.time())}"
print(MODEL_NAME)

def train(net):
   BATCH_SIZE = 4
   EPOCHS = 16
   with open("model.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,IMG_SIZE,IMG_SIZE).to(device)
            batch_y = train_y[i:i+BATCH_SIZE].to(device)

            acc, loss = fwd_pass(batch_x, batch_y, train = True)
            if i % 1 == 0:
               val_acc, val_loss = test(size=19)
               f.write(f"{MODEL_NAME},{round(time.time(),3)},{round(float(acc),2)},{round(float(loss),
4)},{round(float(val_acc),2)},{round(float(val_loss),4)},{epoch}\n")

train(net)
```

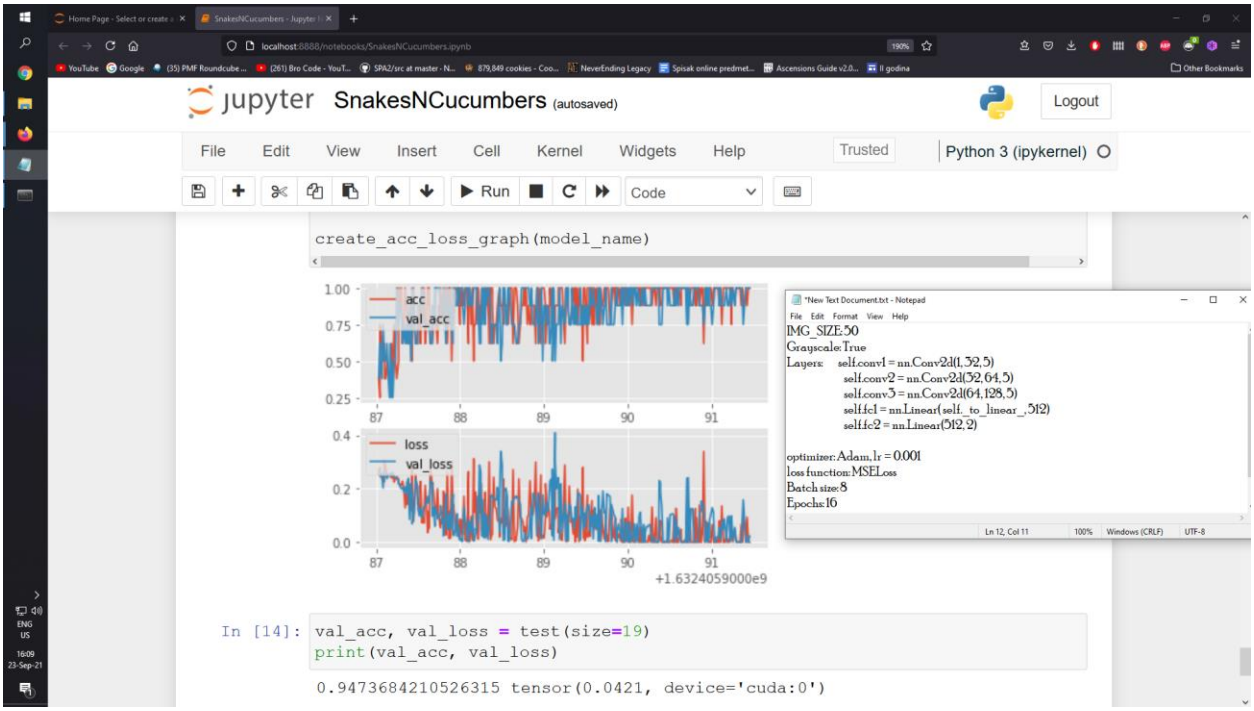
U spoljni fajl pisemo training I test acc I loss kroz vreme, da bi smo ga kasnije mogli procitati i bolje videti kako je tekao process ucenja. Osim toga imamo train funkciju koja kroz fwd_pass u train = true modu provlaci sve podatke. EPOCHS je broj puta koji ce svaka slika biti vidjena. Batch size odredjuje sa koliko slika cemo raditi u odjednom. S obzirom da je dataset mali, u log fajl pisemo svaki korak, ali to se moze smanjiti povecavanjem modula u poslednjem if-u.

```
import matplotlib.pyplot as plt
from matplotlib import style

style.use("ggplot")

model_name = MODEL_NAME


def create_acc_loss_graph(model_name):
    contents = open("model.log", "r").read().split("\n")

    times = []
    accuracies = []
    losses = []

    val_accs = []
    val_losses = []

    for c in contents:
        if model_name in c:
            name, timestamp, acc, loss, val_acc, val_loss, epoch = c.split(",")

            times.append(float(timestamp))
            accuracies.append(float(acc))
            losses.append(float(loss))

            val_accs.append(float(val_acc))
            val_losses.append(float(val_loss))


    fig = plt.figure()

    ax1 = plt.subplot2grid((2,1), (0,0))
    ax2 = plt.subplot2grid((2,1), (1,0), sharex=ax1)


    ax1.plot(times, accuracies, label="acc")
    ax1.plot(times, val_accs, label="val_acc")
    ax1.legend(loc=2)
    ax2.plot(times,losses, label="loss")
    ax2.plot(times,val_losses, label="val_loss")
    ax2.legend(loc=2)
    plt.show()

create_acc_loss_graph(model_name)
```

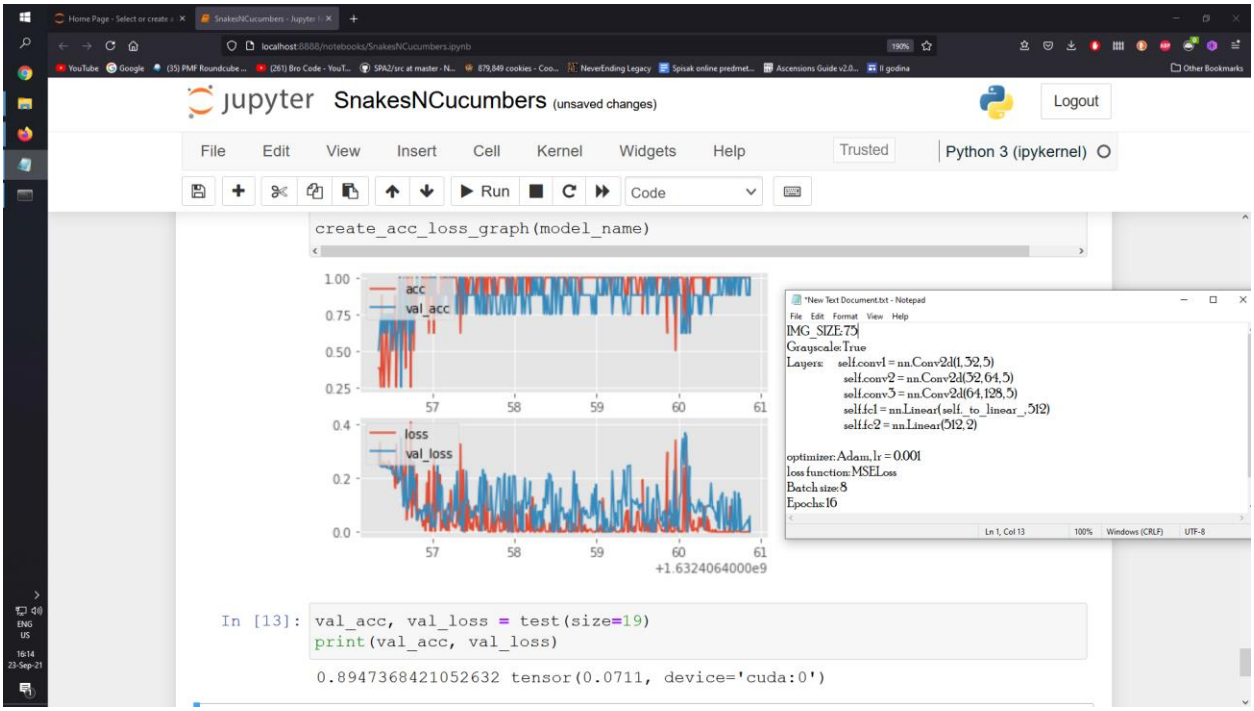
Na kraju pomocu matplotlib-a crtamo nas log fajl.

```
val_acc, val_loss = test(size=19)
print(val_acc, val_loss)
```

I na test podacima proveravamo finalnu preciznost mreze I loss. Sto vise povecamo size, rezultati ce biti precizniji, ali ne smemo otici iznad ukupne kolicine test data-e koju smo ostavili.

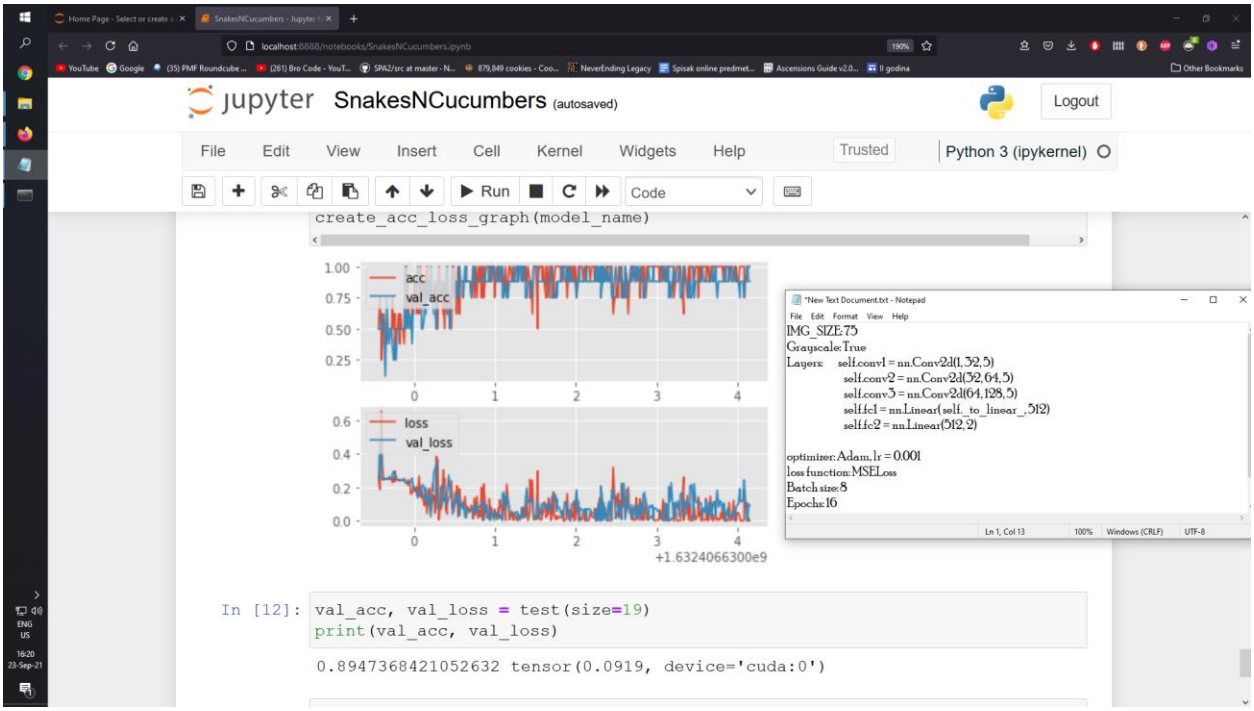


Kako izlgedaju rezultati nakon sto pokrenemo mrezu. Parametri su prikazani u notepad-u sa strane. Sa malom slikom I bez boja, rezultati su dosta haoticni ali imaju tendenciju da se popravljaju. Posle 16 epoha, preciznost mreze je 94%, sto je neobicno visoko, ali ne I zabrinjavajuce kad se uzme u obzir koliko je nas problem jednostavan.

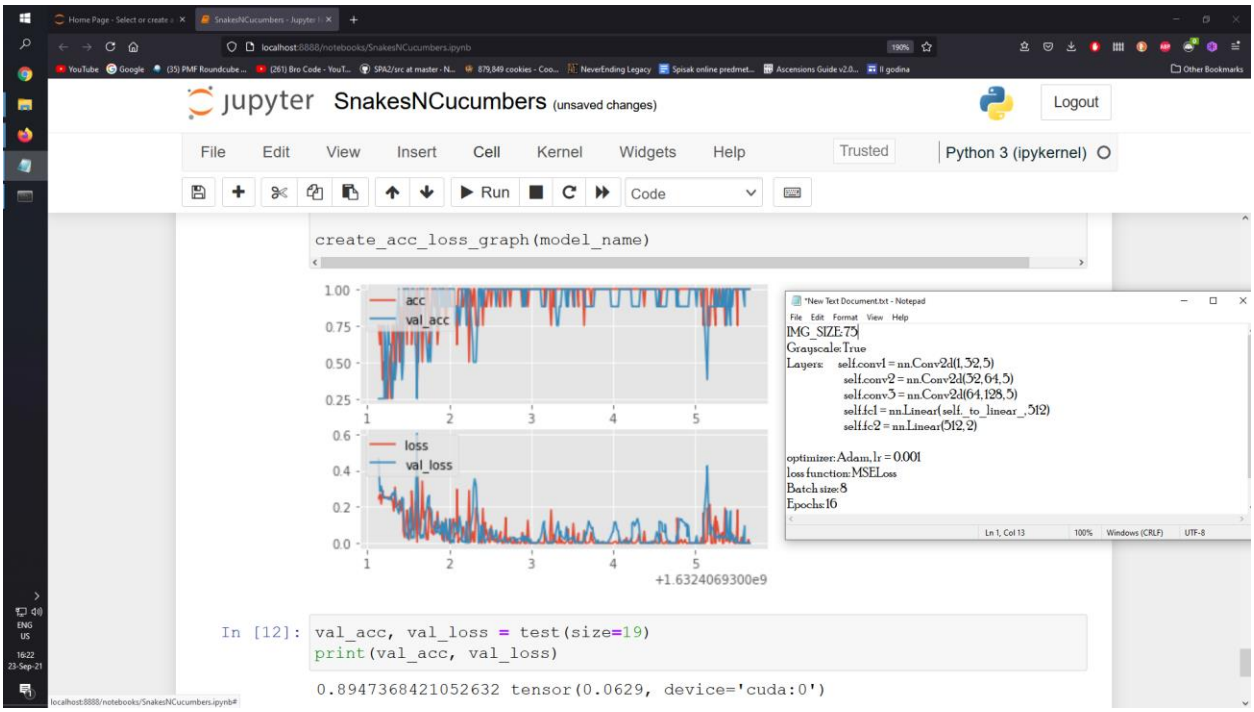


Malo veca slika daje manje haoticne rezultate. Prilikom svakog ucenja redosled slika koje su prikazane mrezi je drugaciji, pa je moguce dobiti razlicite rezultate sa istim parametrima samo zato sto je mreza drugacijim redosledom videla slike. Sto ima smisla, ukoliko vidi krstavce pa zmije naizmenicno, mnogo su

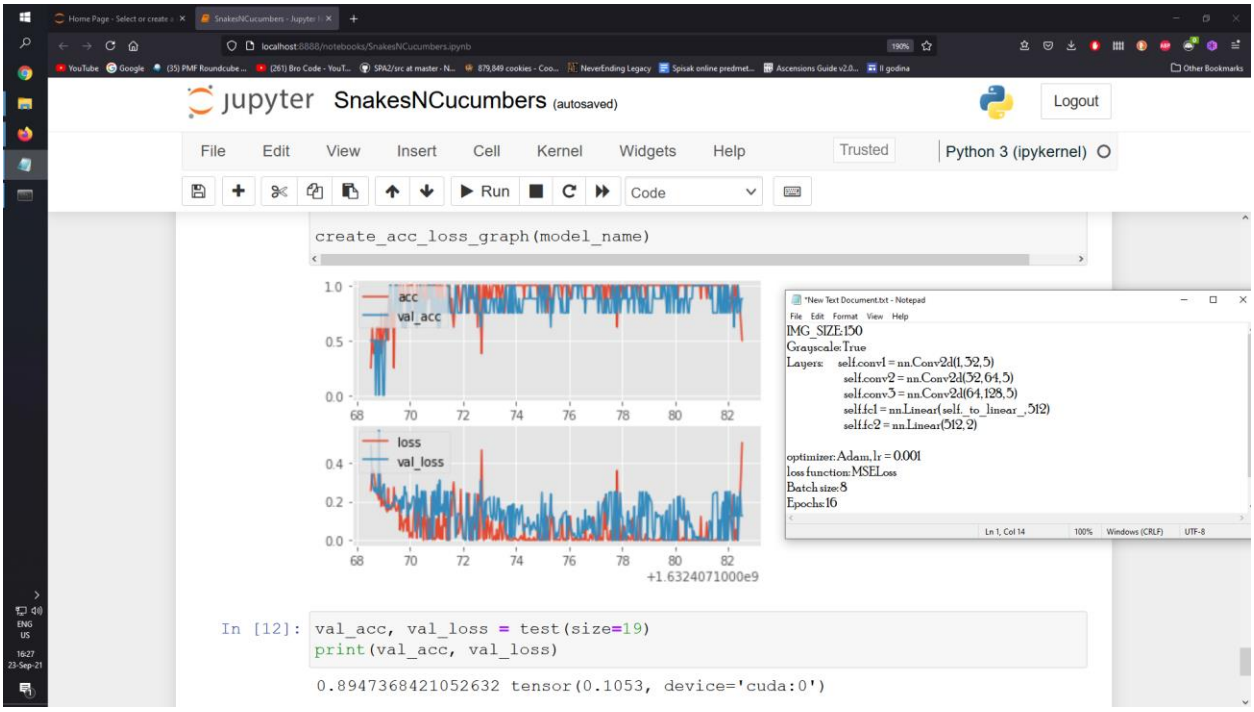
vece sanse da ce preciznost biti dobra nego ukoliko prvo vidi pedeset zmija, u kom slucaju ce imati tendenciju da misli da je sve zmija. Ovo se u velikoj meri javlja samo pri manjim kolicinama informacija, kao u mom slucaju. U kasnijim eksperimentima sa vise podataka, mreza odista cesce daje slicnije rezultate.



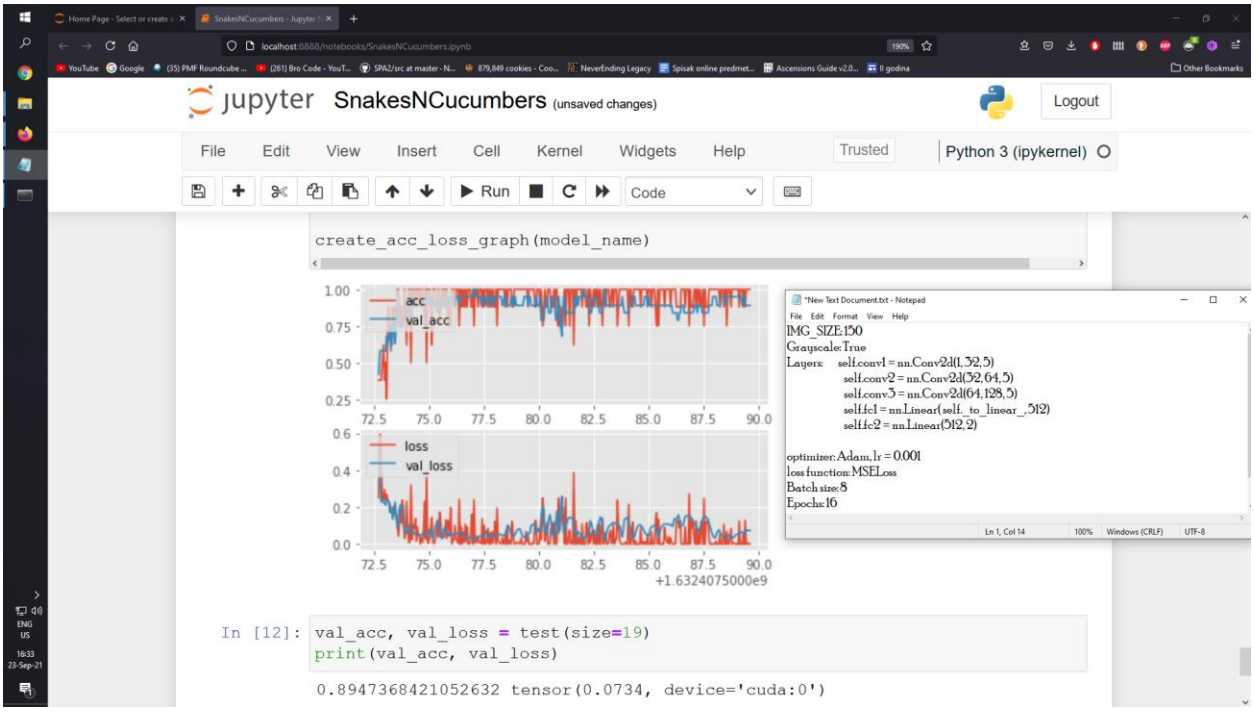
Drugi pokusaj sa istim parametrima.



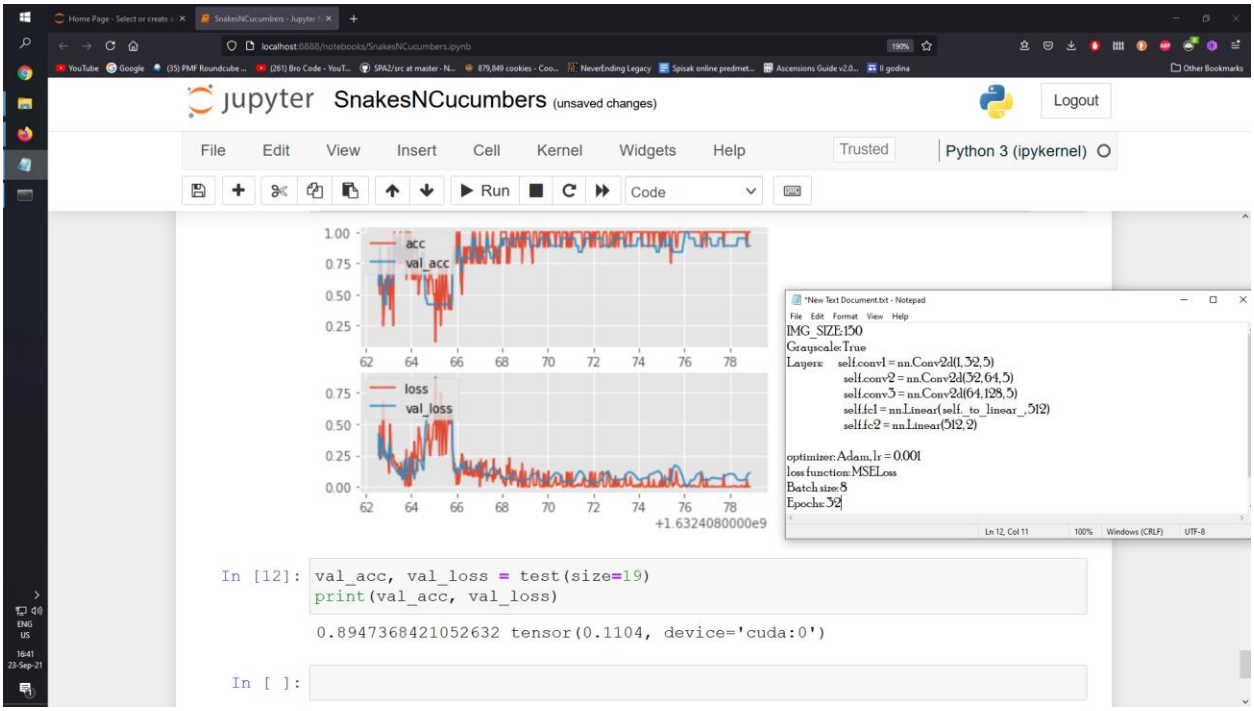
Treci pokusaj. Konacna preciznost ostaje ista, ali se vidi razlicit loss I izgled grafika.



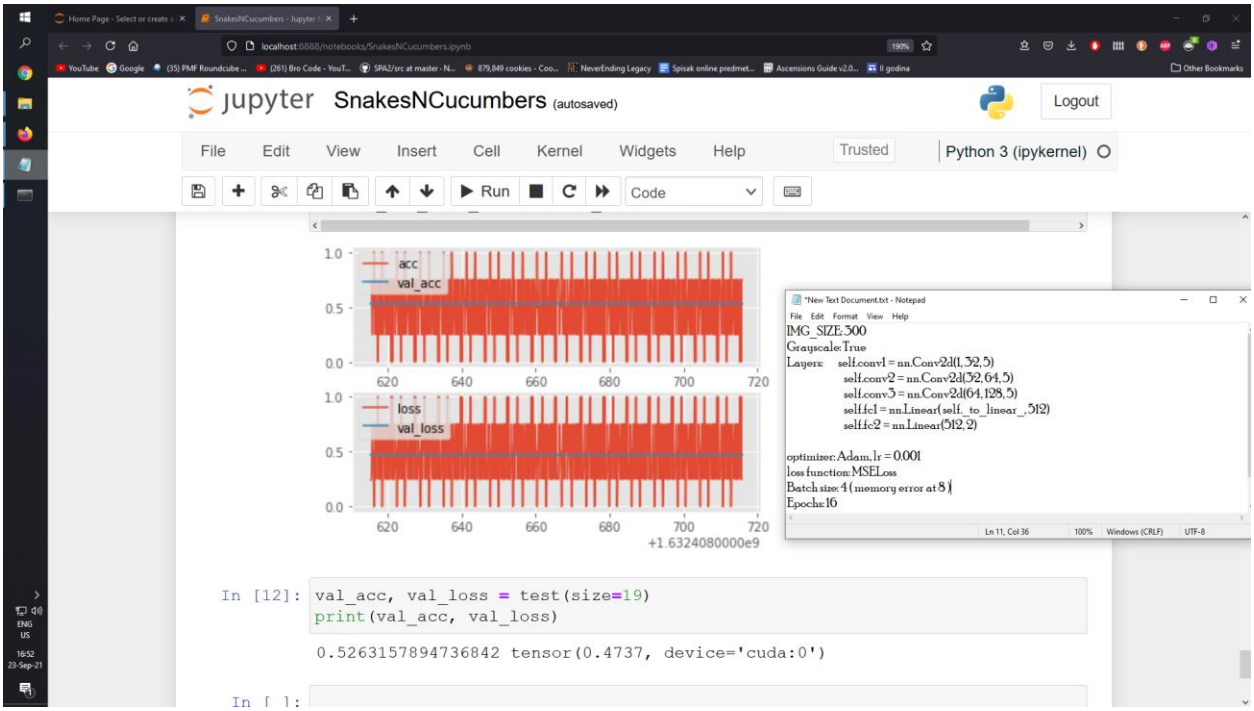
Povecanje dimenzija slika na 150x150. U kasnijim eksperimentima se pokazalo da je proporcija velicine slike I broja neurona bitna, zato su ovi rezultati, kontraintuitivno losiji.



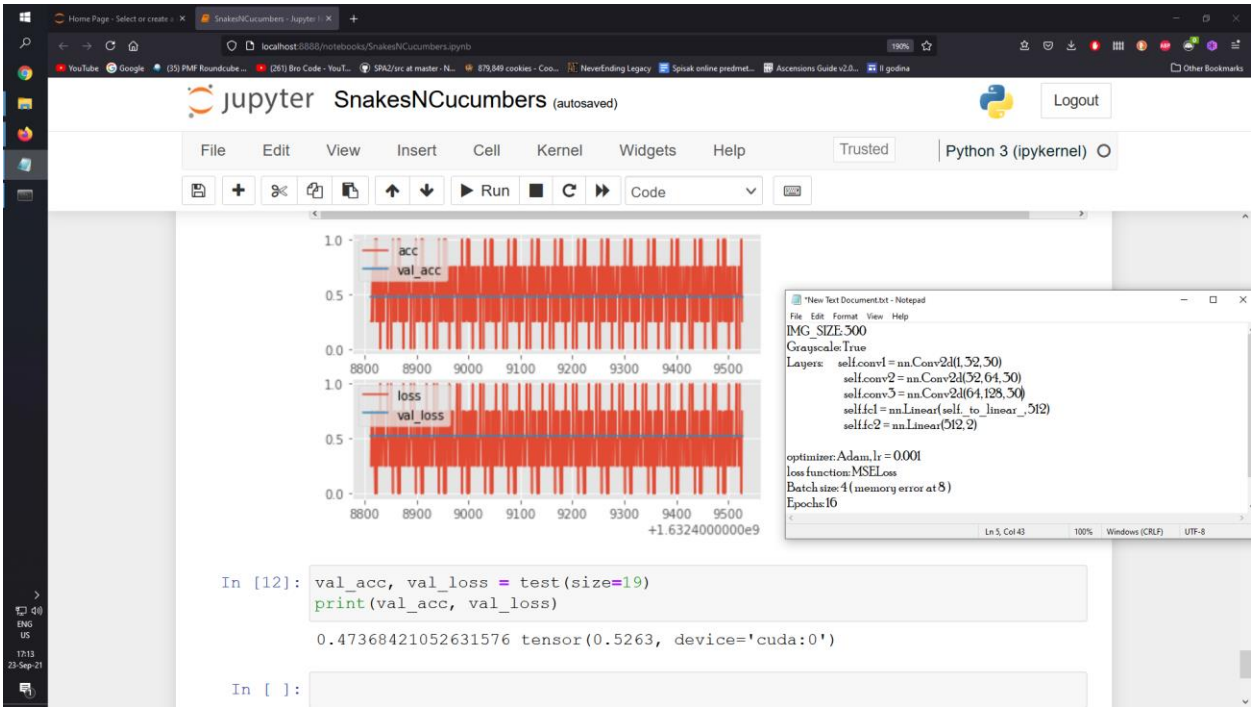
Jos jedan pokusaj sa istim parametrima.



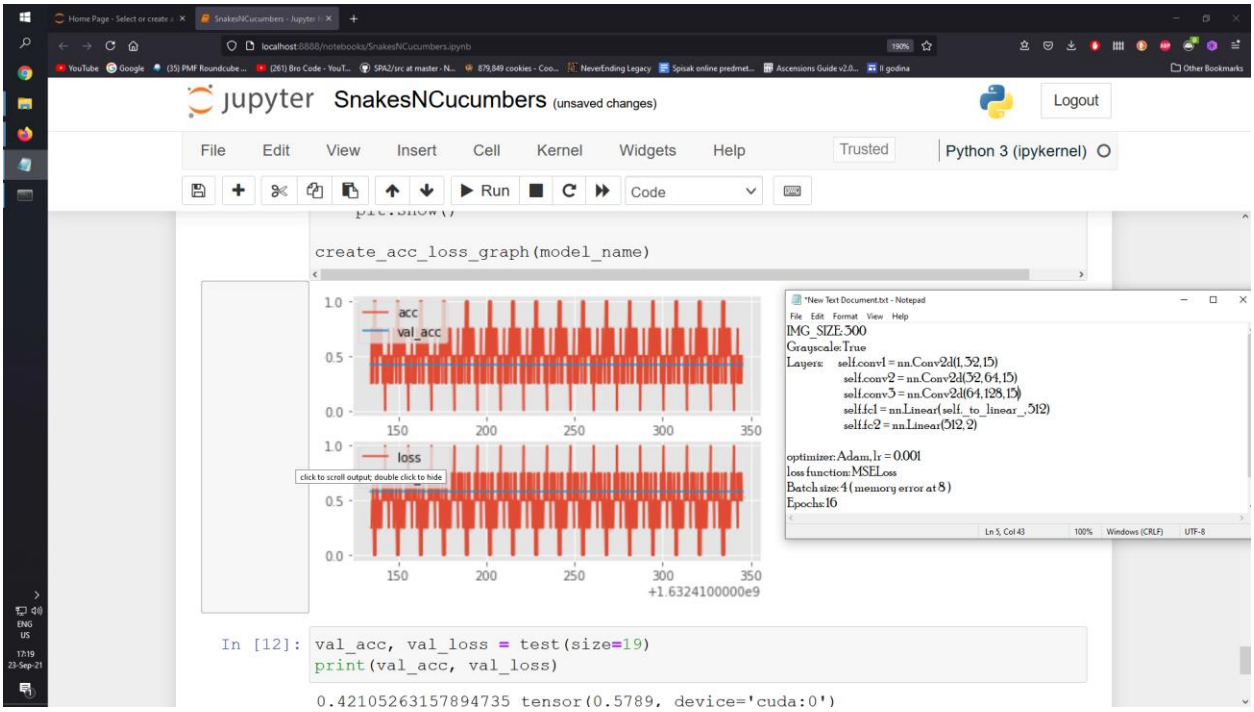
Treci pokusaj sa istim parametrima. Ponovo , uvek dobijamo istu preciznost ali razlicit loss.



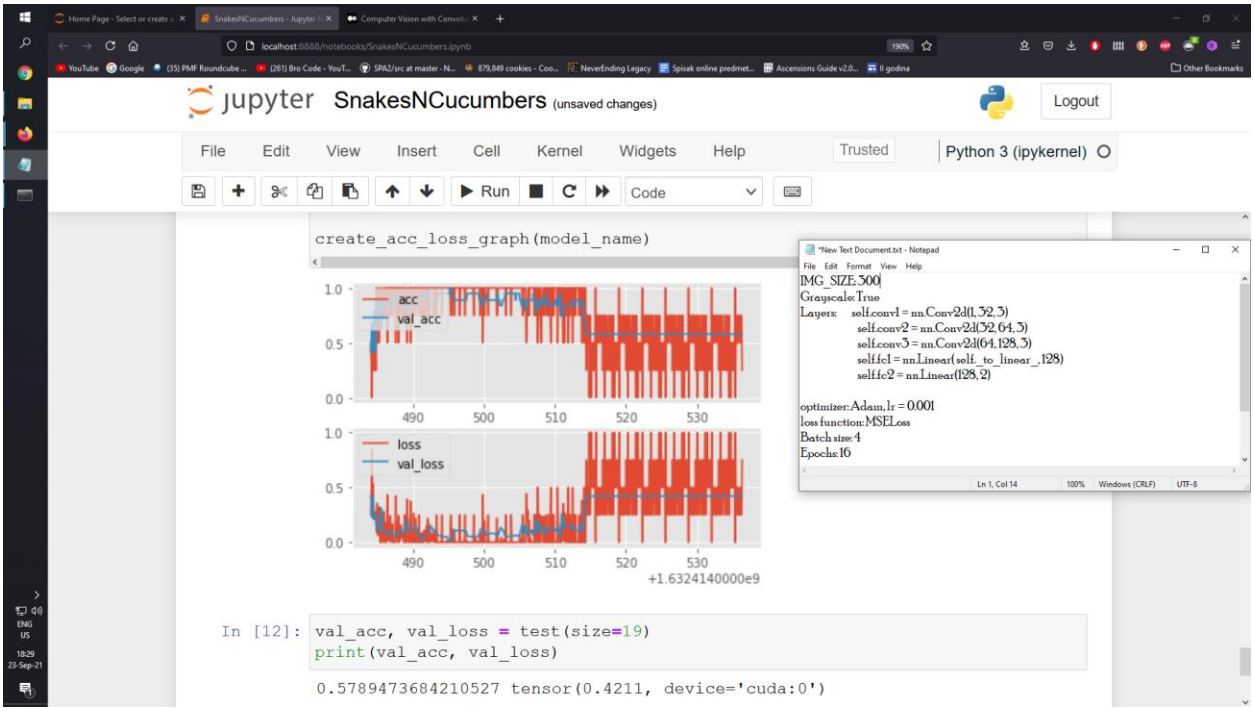
Ukoliko jos povecamo sliku, proporcija broja neurona I velicine slike dovodi do toga da se mreza vrti u mestu. Ovo se u teoriji moze popraviti boljim learning rate-om, ali nisam uspeo da ga nadjem.



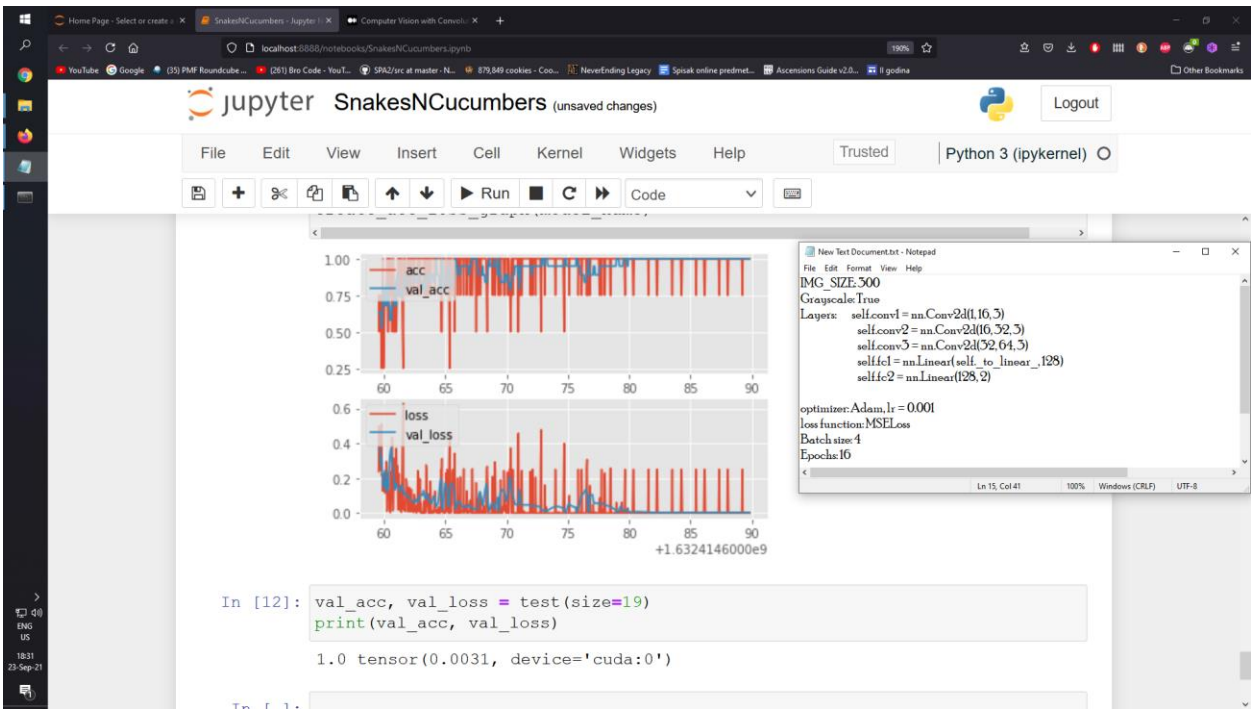
Zanimlo me je da li ce povecaje kernel u conv delu mreze uticati na rezultate. Nije pomoglo.



Mozda sam samo preterao. Ovo je neka sredina za kernel size. Idalje nista. Kasnije sam malo kopao, I saznao da je cak I originalnih 5x5 dosta veliko I da je najbolji pristup 3x3, sto se pokazalo tacnim.

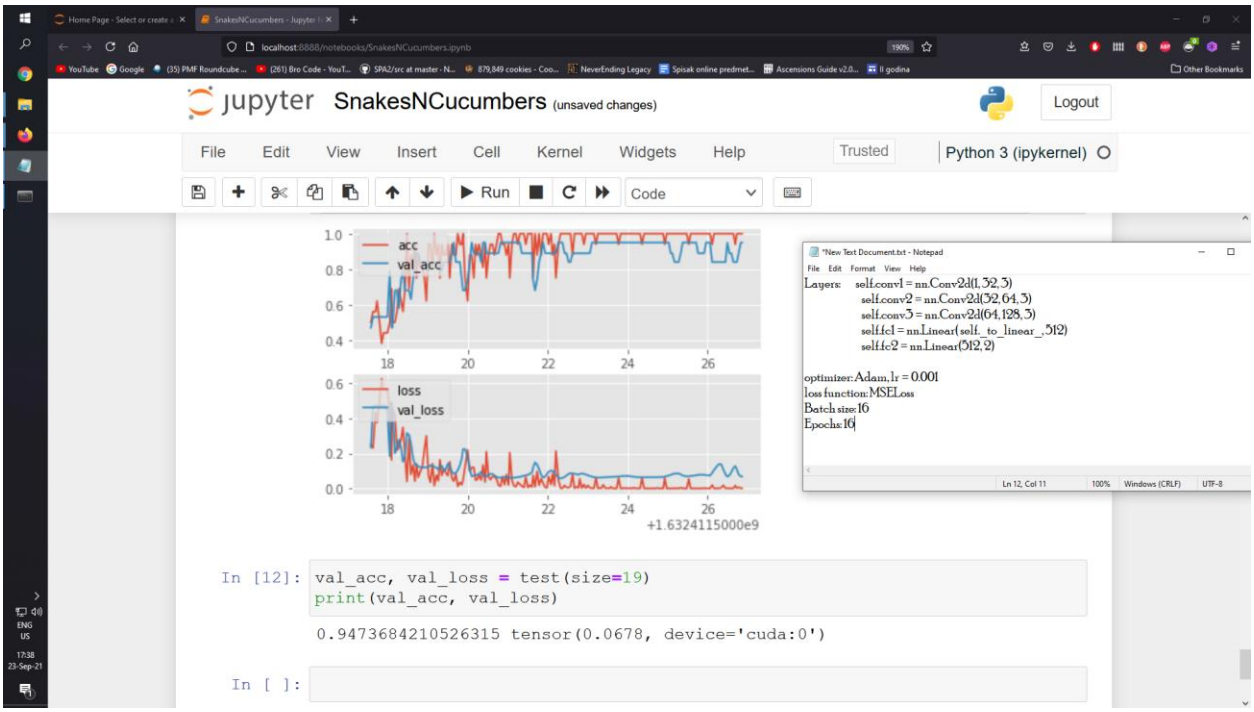


Presli smo na 3x3 kernel. Takodje me je zanimalo koliki uticaj na performanse mreze ima zadnji, fully connected sloj. Naizgled, veliki deo ucenja se desava I tu, a ne samo u conv sloju. Smanjenjem zadnjeg sloja dobijamo drasticno drugacije (mada ne I bolje) rezultate.

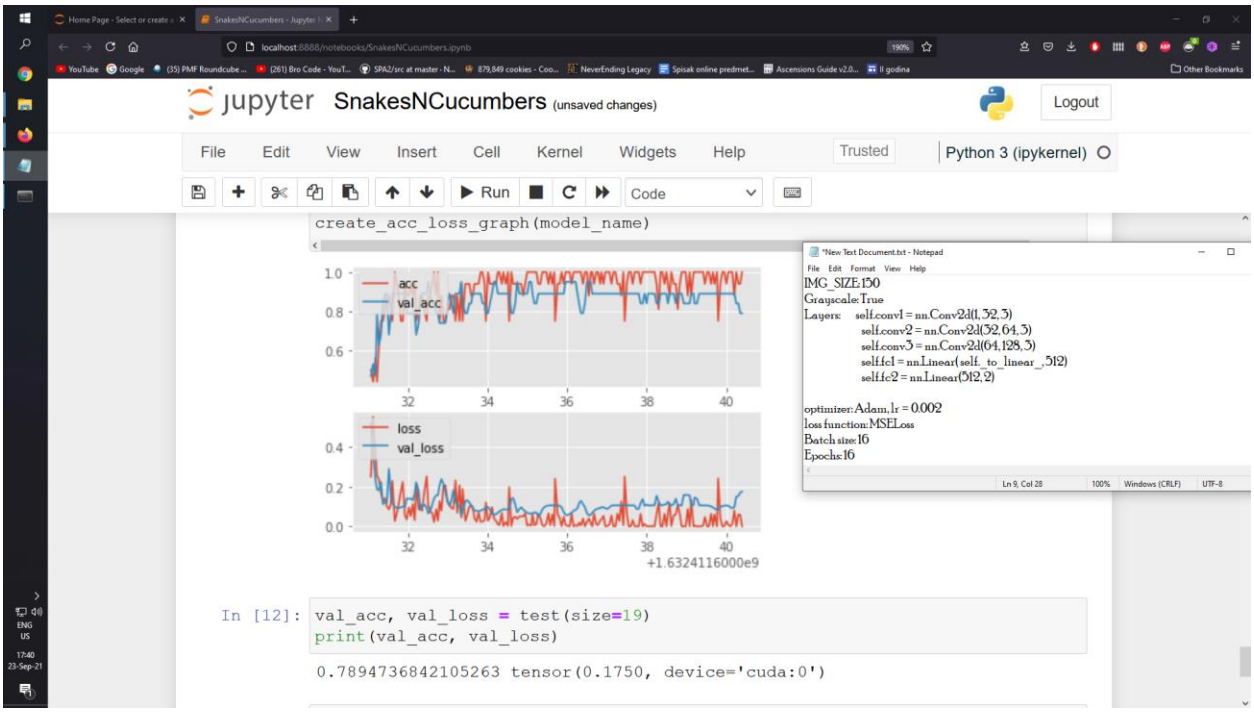


Sta se onda desi ako smanjimo broj neurona u celoj mrezi? Sa ovako velikom slikom, rezultati se poboljsavaju. Sto me navodi na sledece razmisljanje – deo informacija mreza izvlaci iz slike, ali deo su samo prepostavke na osnovu neurona. Ako imamo manju sliku, potrebno je imati vise neurona da bi se

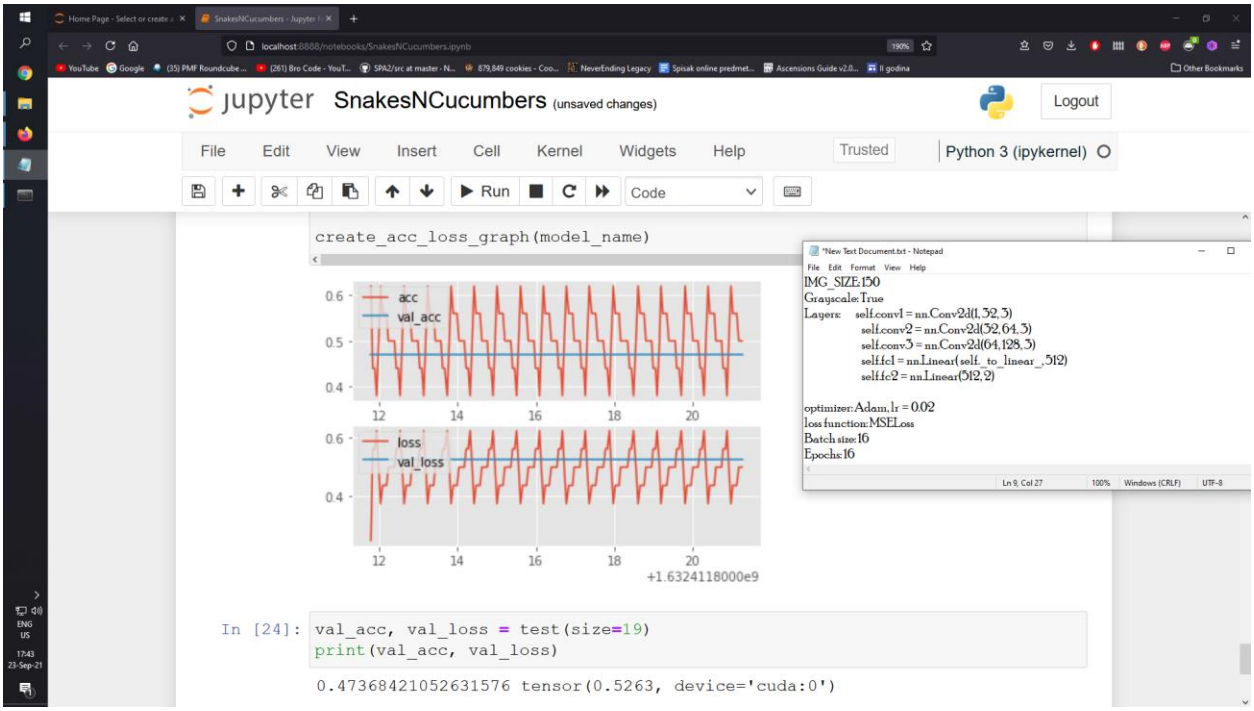
nadoknadio manjak informacija iz piksela. Medjutim ukoliko je slika velika, u kombinaciji sa velikim brojem neurona mreza ima previse parametara – idalje moze da uci, ali se learning rate mora smanjiti kako se ne bi zapucala na nekom mestu. U tom slucaju je bolje samo smanjiti broj neurona na slojevima, I pustiti mrezu da vecinu podataka koristi iz samih slika a manje predpostavlja.



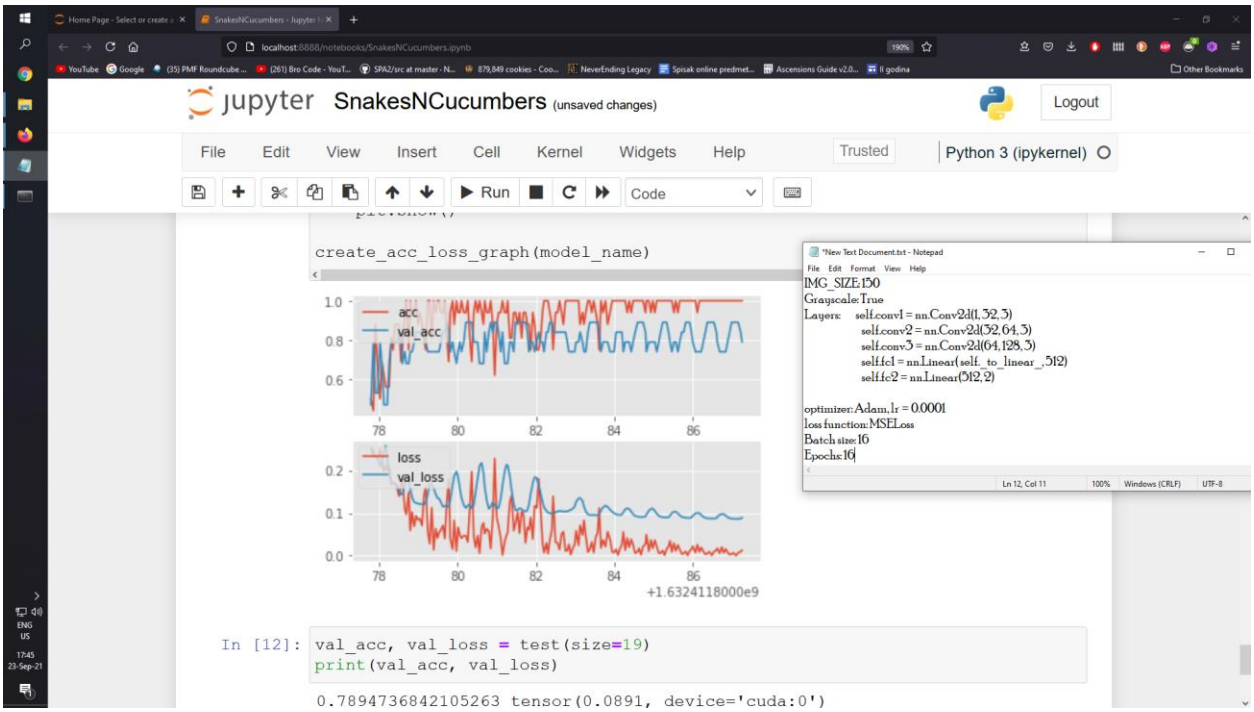
Isto to sam zatim isprobao na malo manjoj slici. Rezultati su idalje bili dobri.



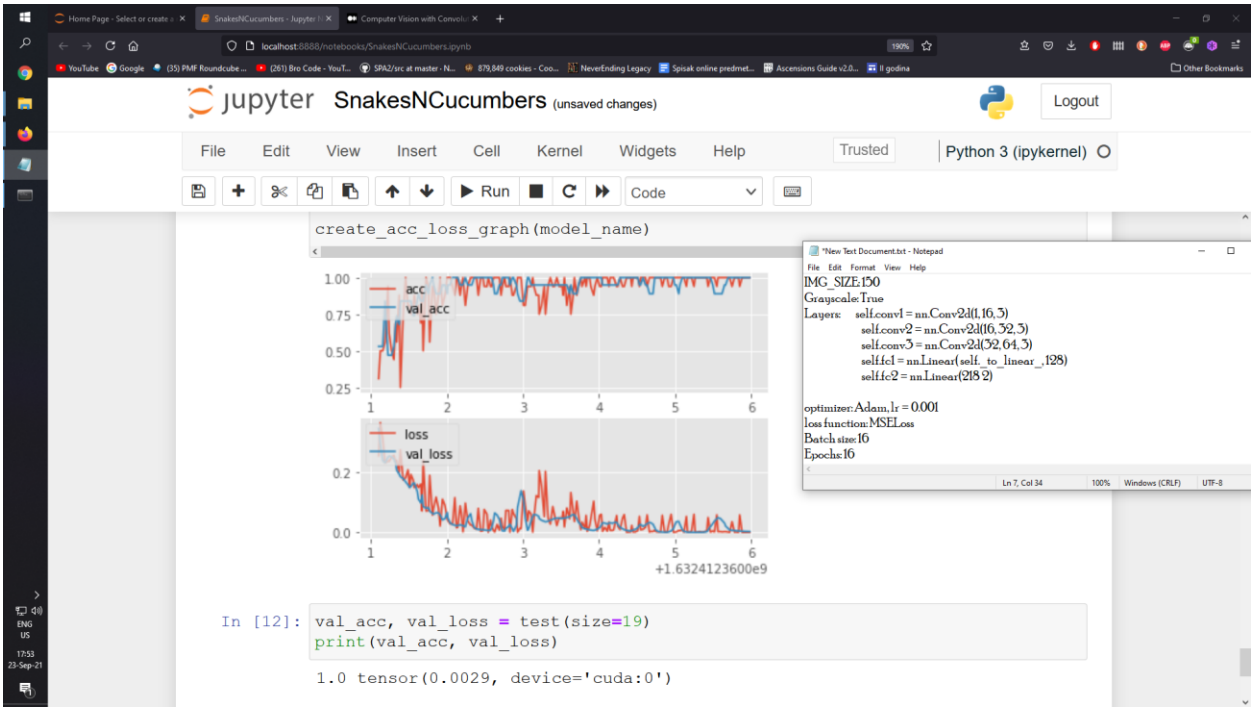
Zanimalo me je kako ce uticati povecanje learning rate-a. Kao sto se dalo I predvideti, brzo je doslo do overfittovanja podataka (sto se vidi po tome da je preciznost manja za test datu, a dobra za train-ing date-u (sto se vidi po tome da se plava linija odvaja od crvene)). Ispostavlja se da je originalni lr koji sam koristio najbolji.



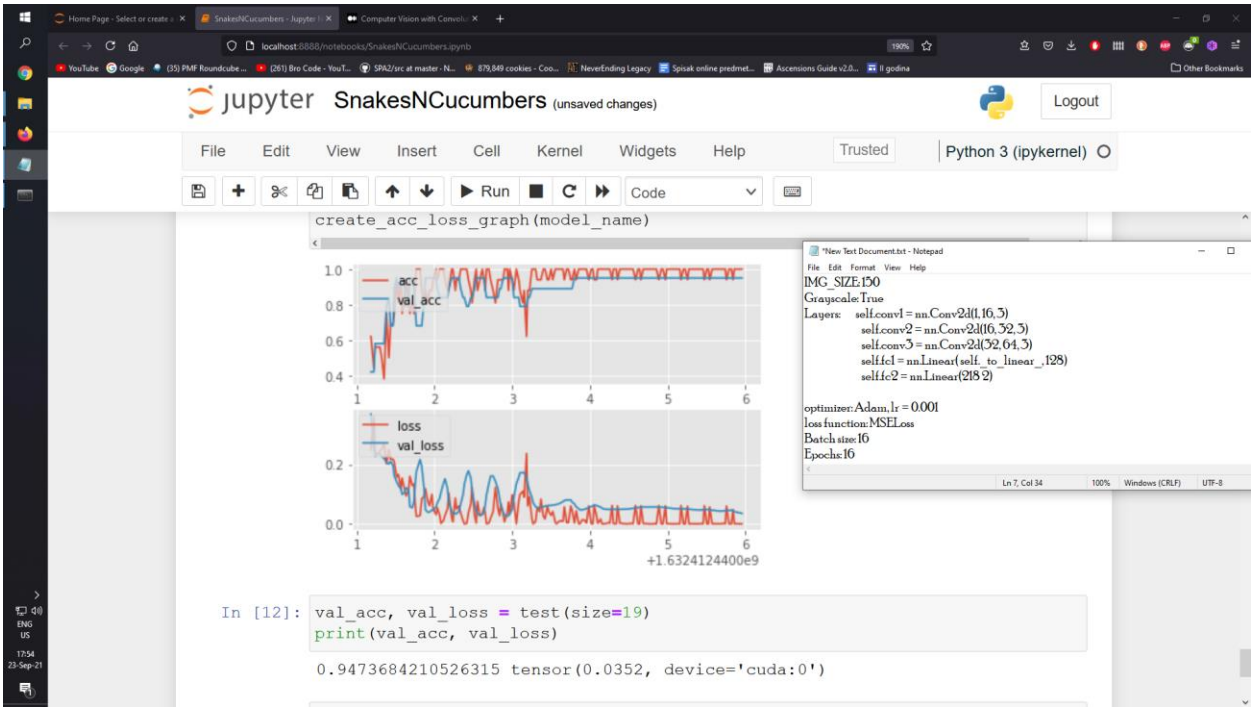
Dodatno povecan learning rate. Predvidiv rezultat toga da se mreza vrti u krug izmedju dva mesta.



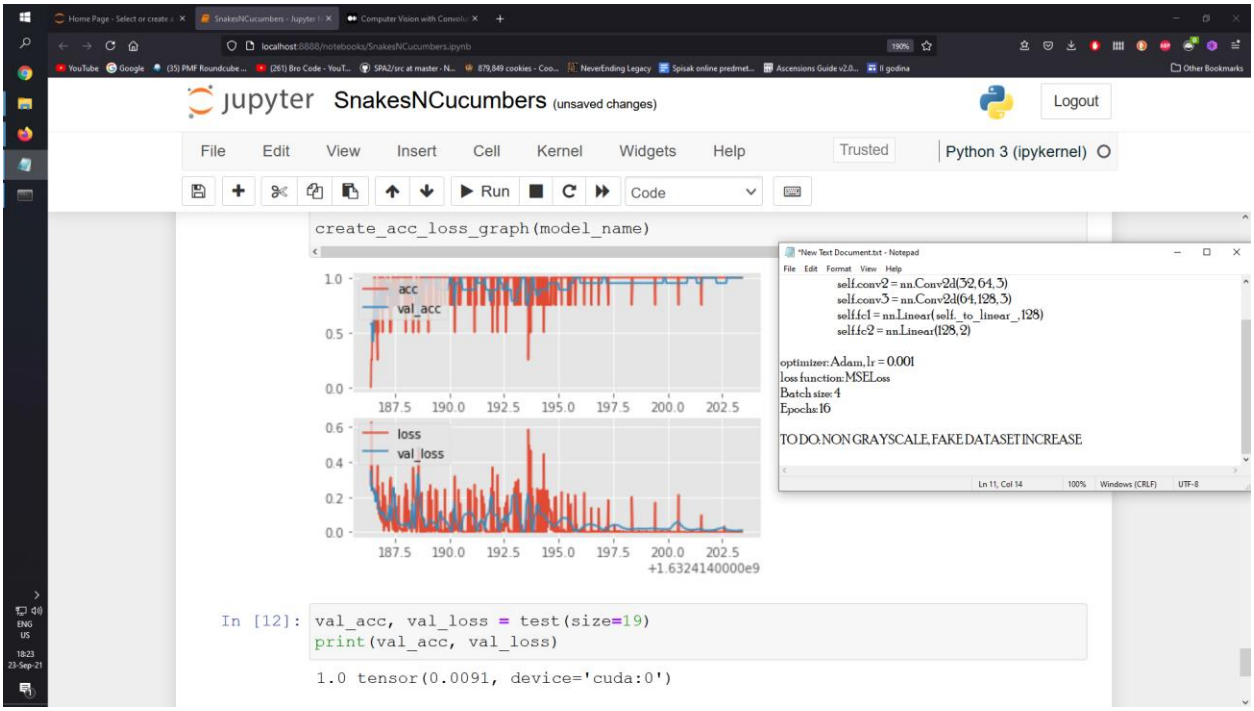
Iznenadjujuce, premali learning rate takodje vodi do overfittovanja. Nisam siguran zasto.



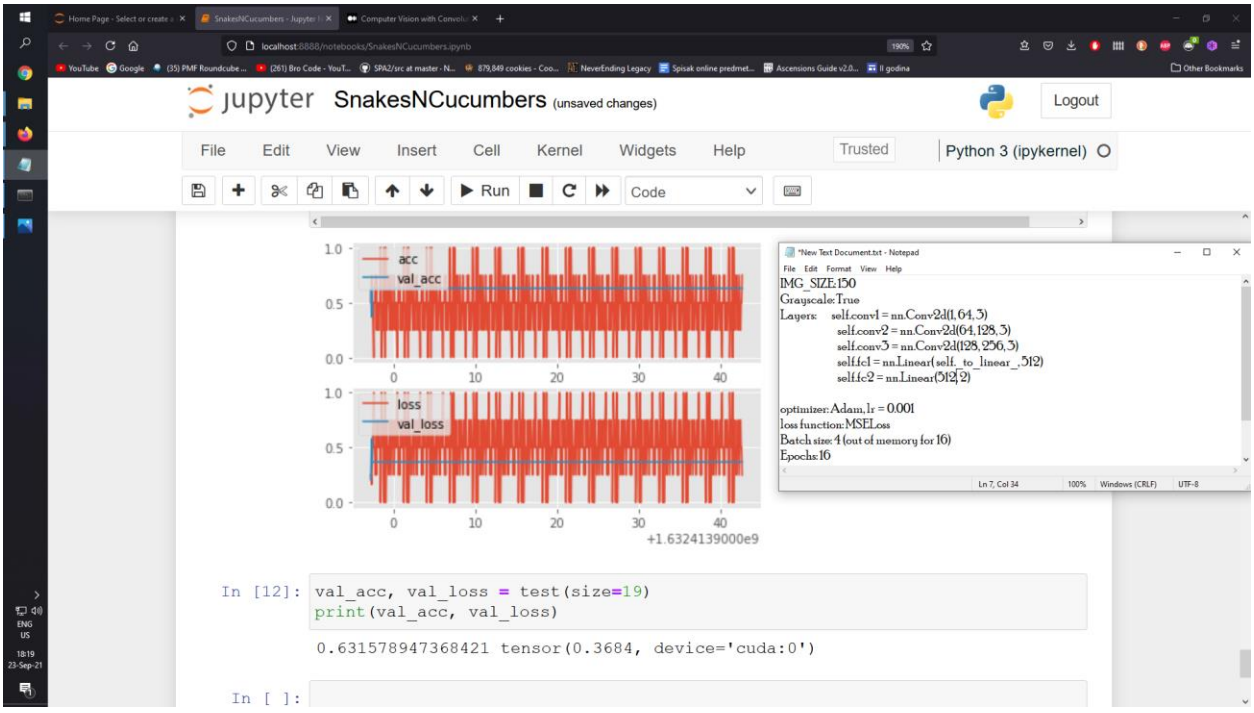
Probao sam istu smanjenu verziju mreze I na manjoj slici, sa odlicnim rezultatima. Izgleda da je oko 150x150 sasvim dovoljno da mreza izvuce sve informacije koje joj trebaju sa slike, jer se rezultati nisu preterano poboljsavali kako sam povecavao velicinu slike.



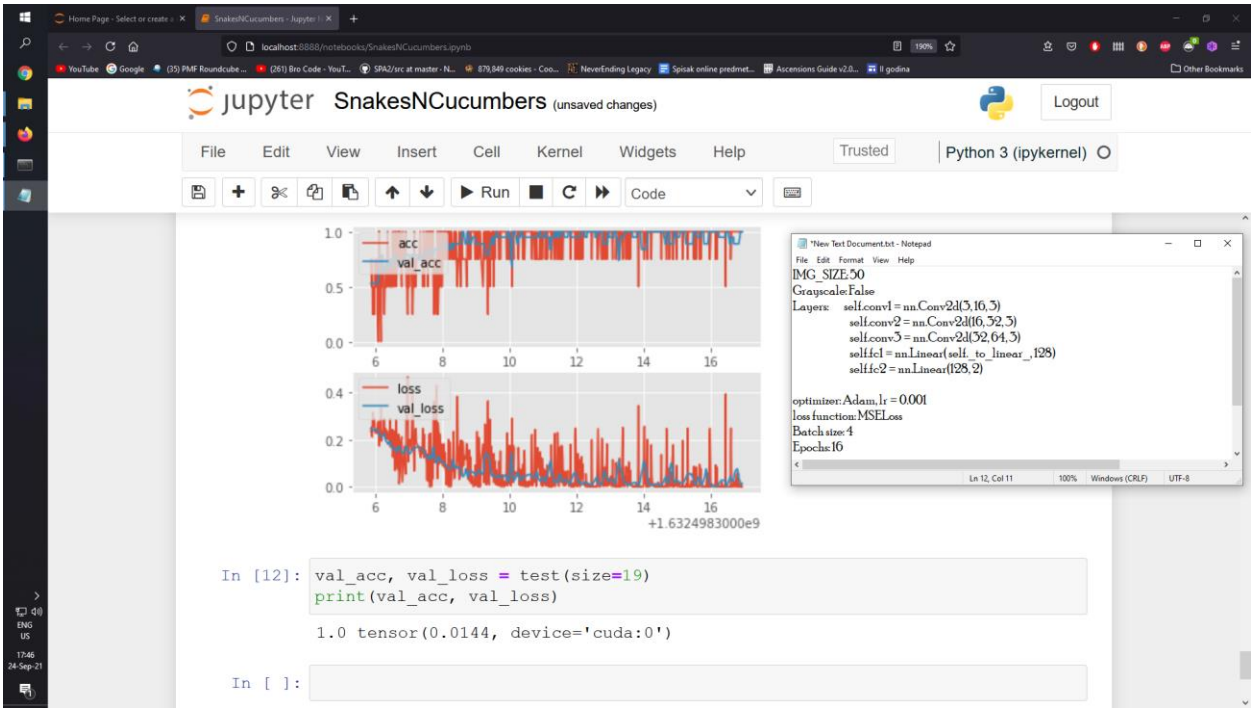
Jos jedan pokusaj sa istim parametrima.



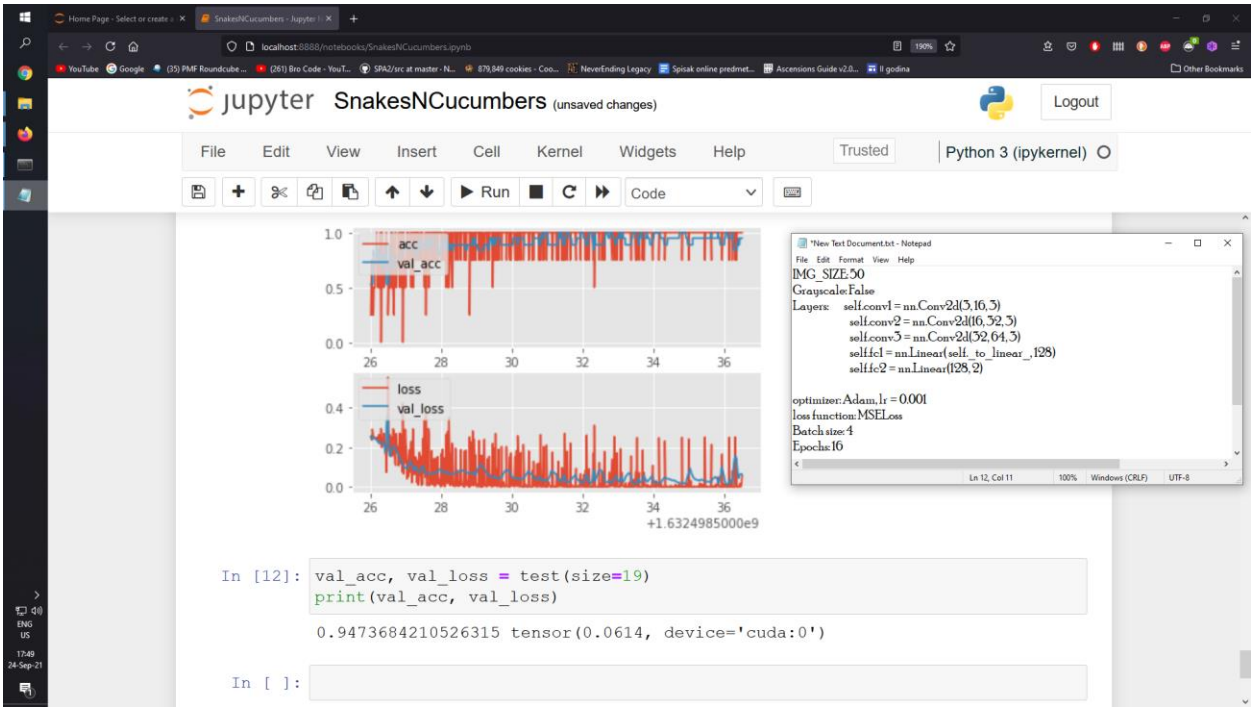
Posto je ranije smanjenje samo poslednjeg sloja dalo znacajne rezultate, zanimalo me je kako ce rezultati izgledati ako ga ostavim istim, a samo povecam conv slojeve. Mnogo bolje, pogotovo u kasnijim epohama. Dakle mozda nije toliko bitan odnos velicine slike I celokupne mreze, nego slike I fully connected sloja.



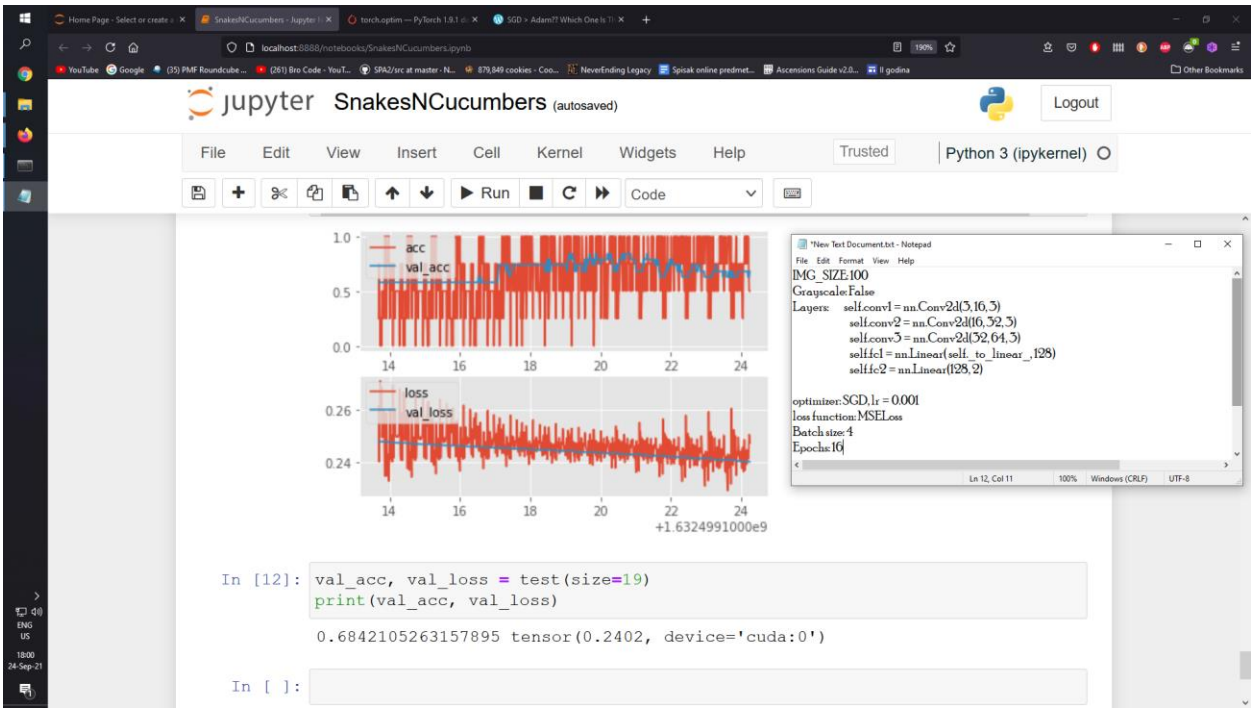
Da bih to testirao, ponovo sam povecao zadnji sloj. Predvidivo, dobijamo losije rezultate (opet, sa potencijalnim fix-om uz drugi lr, ali nisam uspeo da nadjem neki koji radi. To takodje deluje kao los metod posebno jer se ovaj koji koristim u predhodnim eksperimentima pokazao kao odlican.)



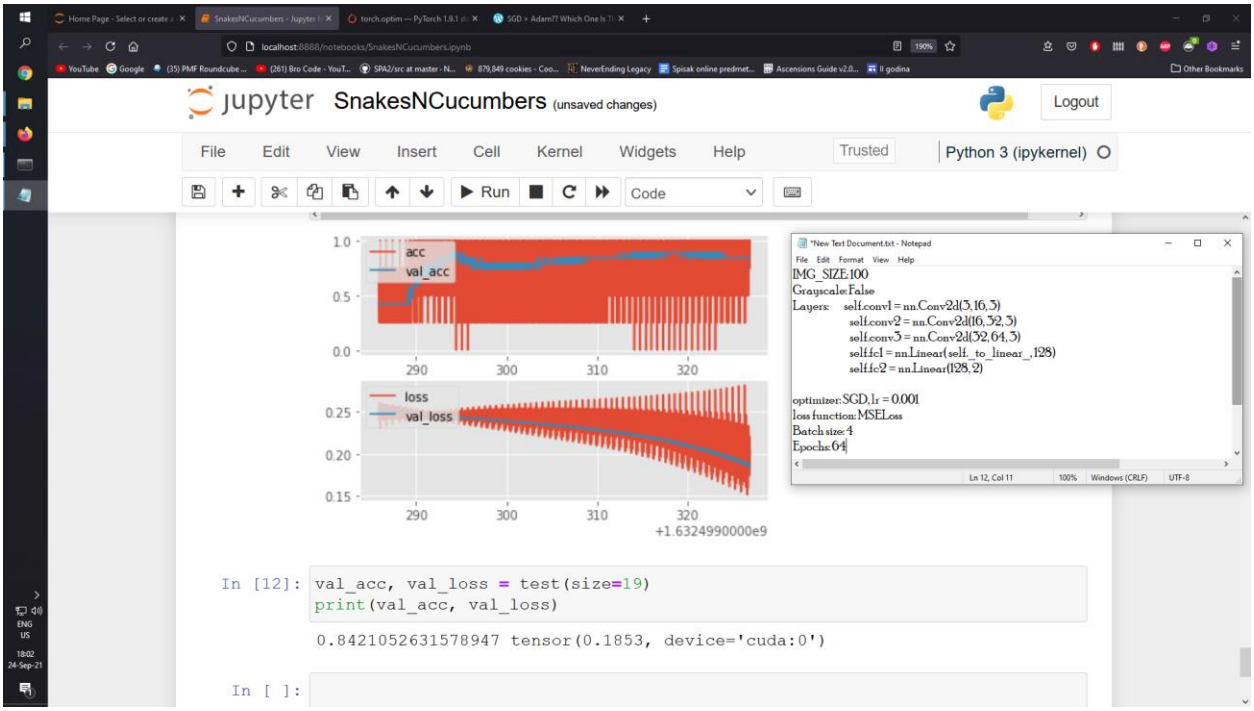
Namerno sam uzeo bas zmije I krastavce jer su slicne boje. Znao da zelim da mi jedan od eksperimenata bude uticaj boje na ucenje, I predvideo sam da u ovom slucaju nece biti znacajan, jer mreza ne razaznaje razliku izmedju zmija I krastavaca na osnovu boje vec oblika. To se pokazalo tacnim. Mreza iznad je skoro identicna po rezultatima a osim boje ista po parametrima mrezi 50x50 mrezi od ranije.



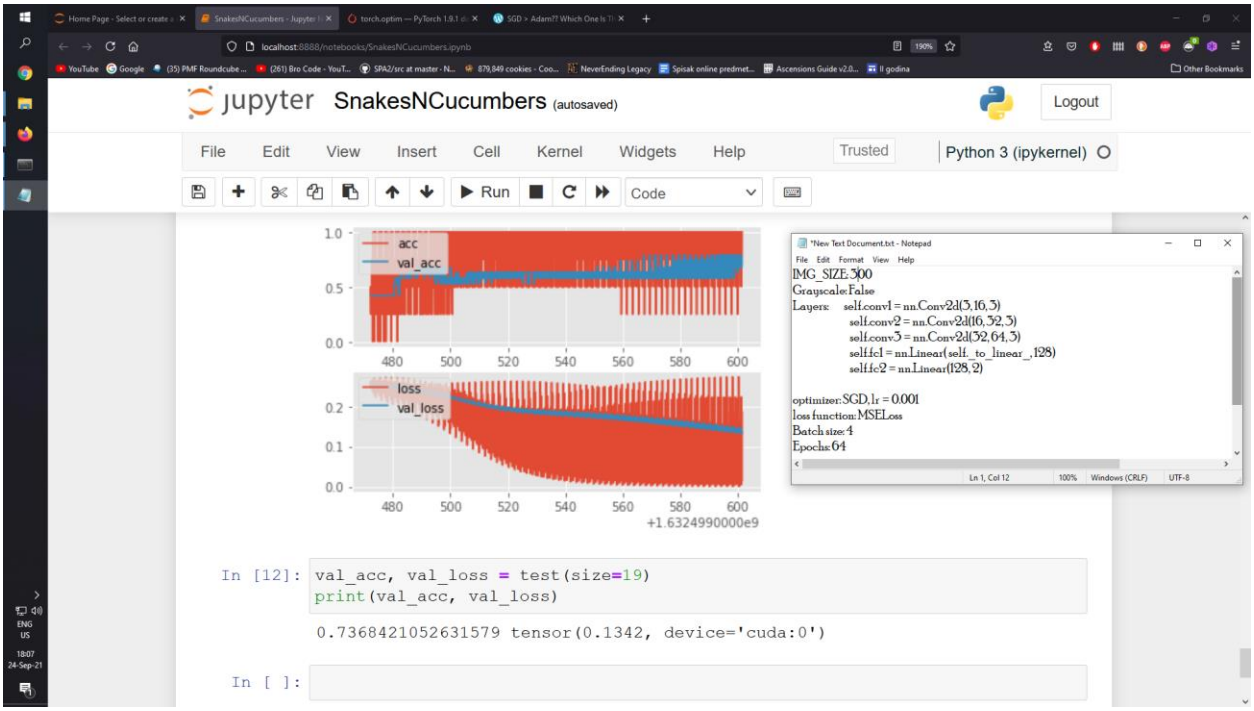
Naravno I ovde zbog shuffla ima mesta za sitne razlike medju razlicitim pokusajima sa istim parametrima.



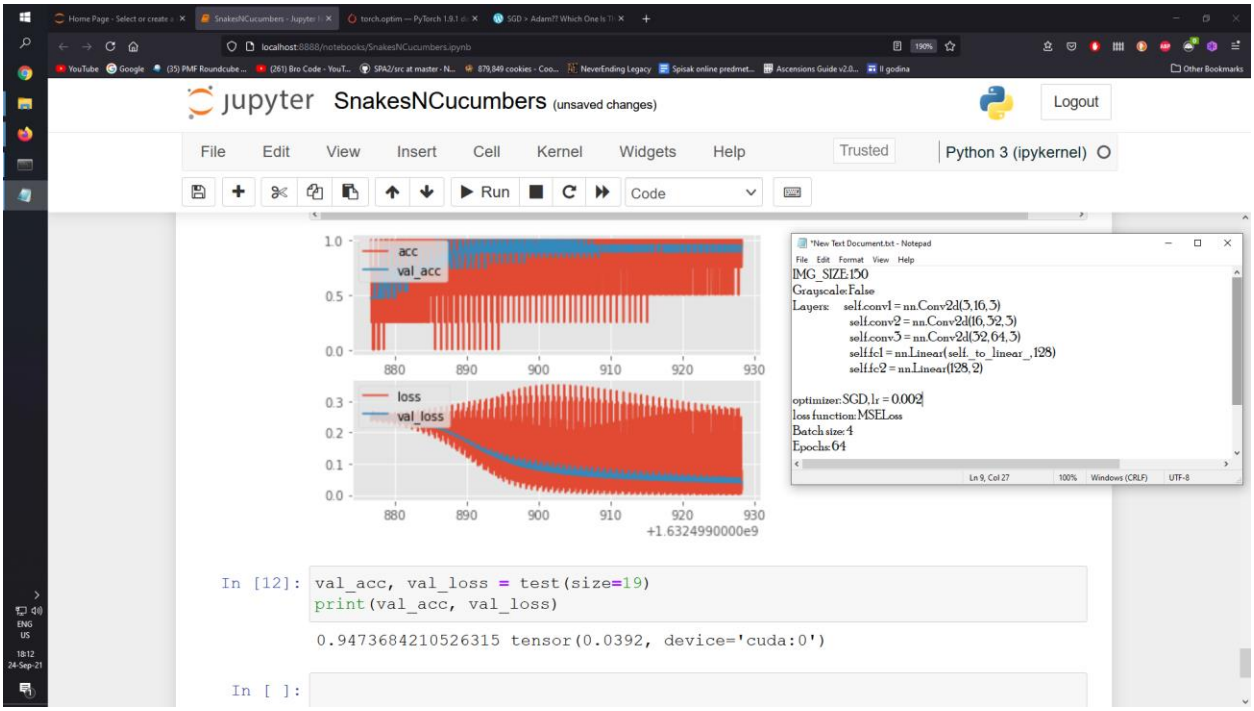
Zatim sam probao da povecam sliku I koristim drugi optimizator. Razlika u nacinu ucenja izmedju dva optimizatora je dosta ocigledna. Adam u prvih nekoliko epoha pravi najbolje rezultate, dok su promene u kasnijim zanemarljive. SGD se popravlja ravnomerno, polako ali sigurno. To se poklapa sa onim sto znamo o SGD-u – da genralno sporo uci I da mu je potrebno puno epoha za dobre rezultate.



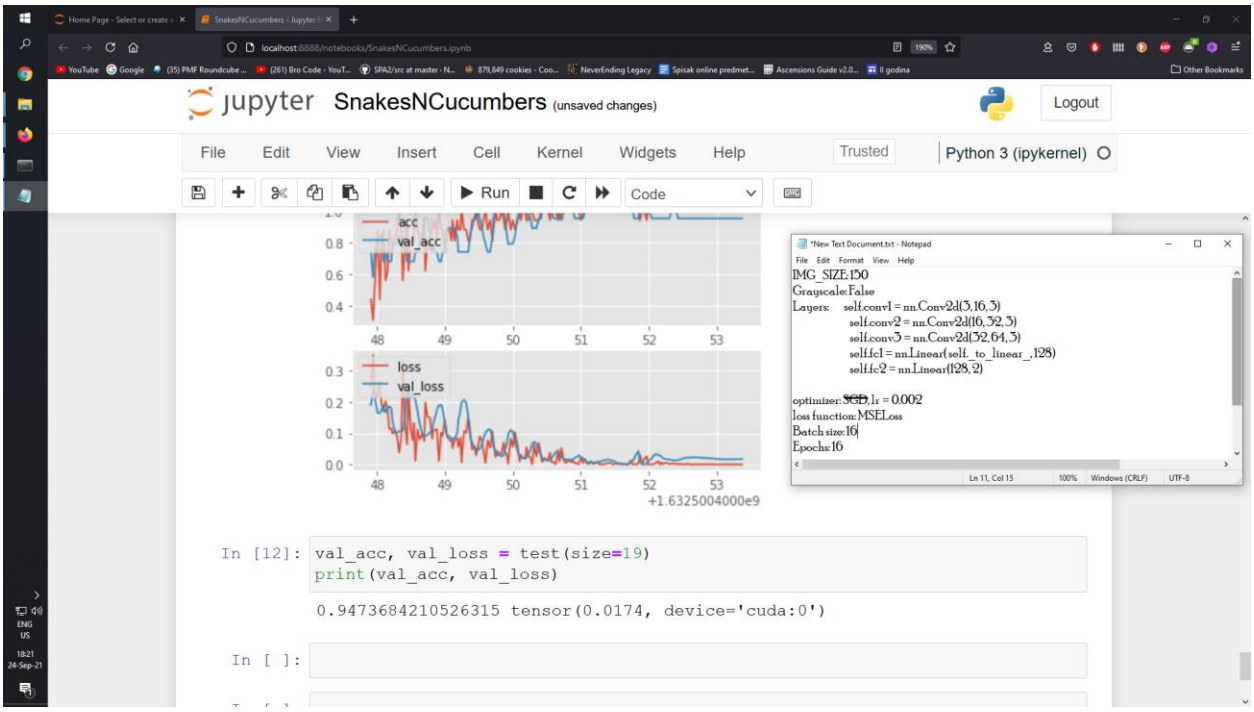
Tako da sam povecao broj epoha, I odista, rezultati su se polako ali sigurno popravljali.



Zanimalo me je kako ce velicina slike uticati na SGD. Sto je kolicina informacija veca, to SGD sporije uci. Posto mi je za ovaj rezultat bilo potrebno preko 10ak minuta ucenja (SGD je generalno dosta spor), odlucio sam da se vratim na Adam.

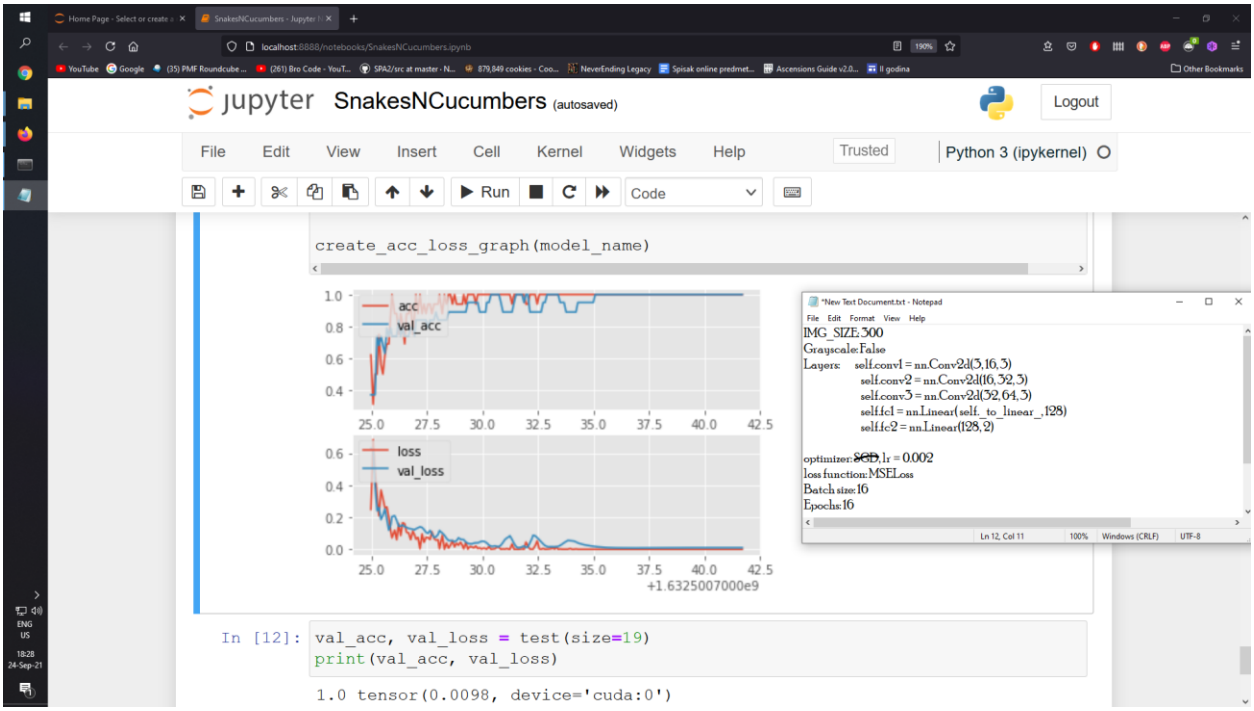


Za kraj sam probao da povecam learning rate SGD-a u nadi da cu ga ubrzati. Radilo je.

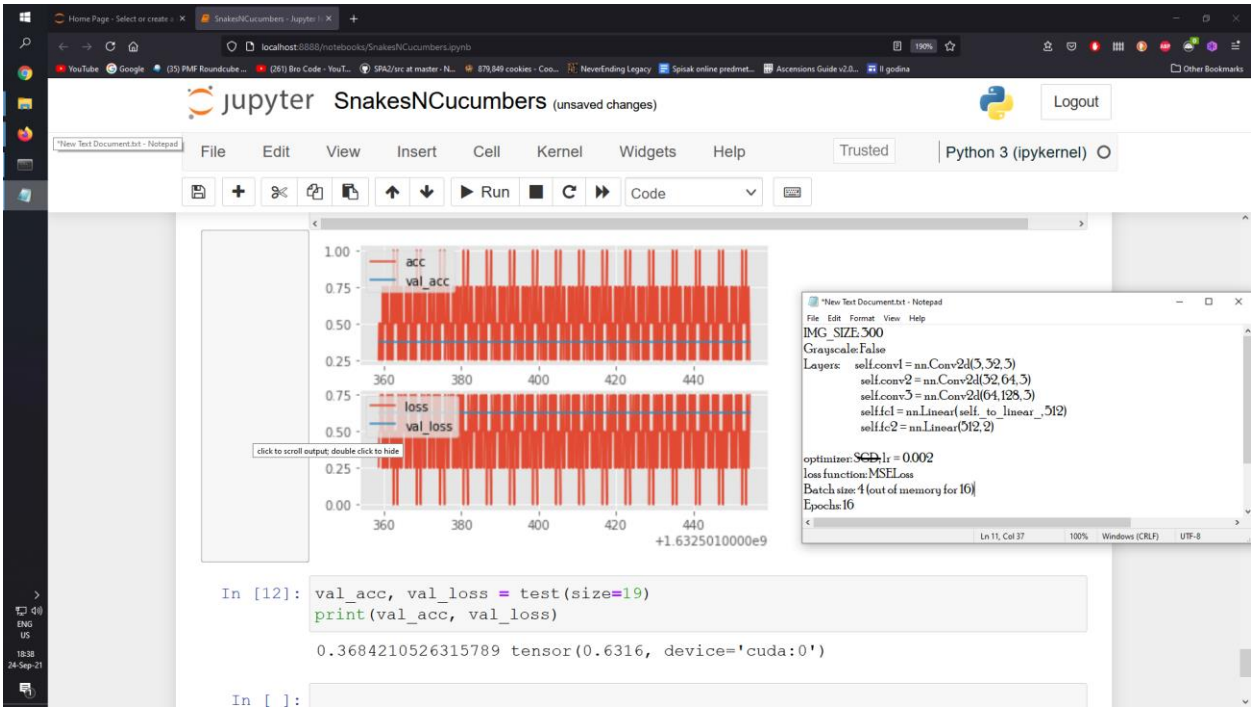


Zatim sam probao parametre koji su odlicno radili na grayscale 150x150 slikama da primenim na slike u boji (ova velicina mreze sa pokazala odlicnom) (na slici je greska – nadalje je optimizer Adam sa 0.001 lr.)

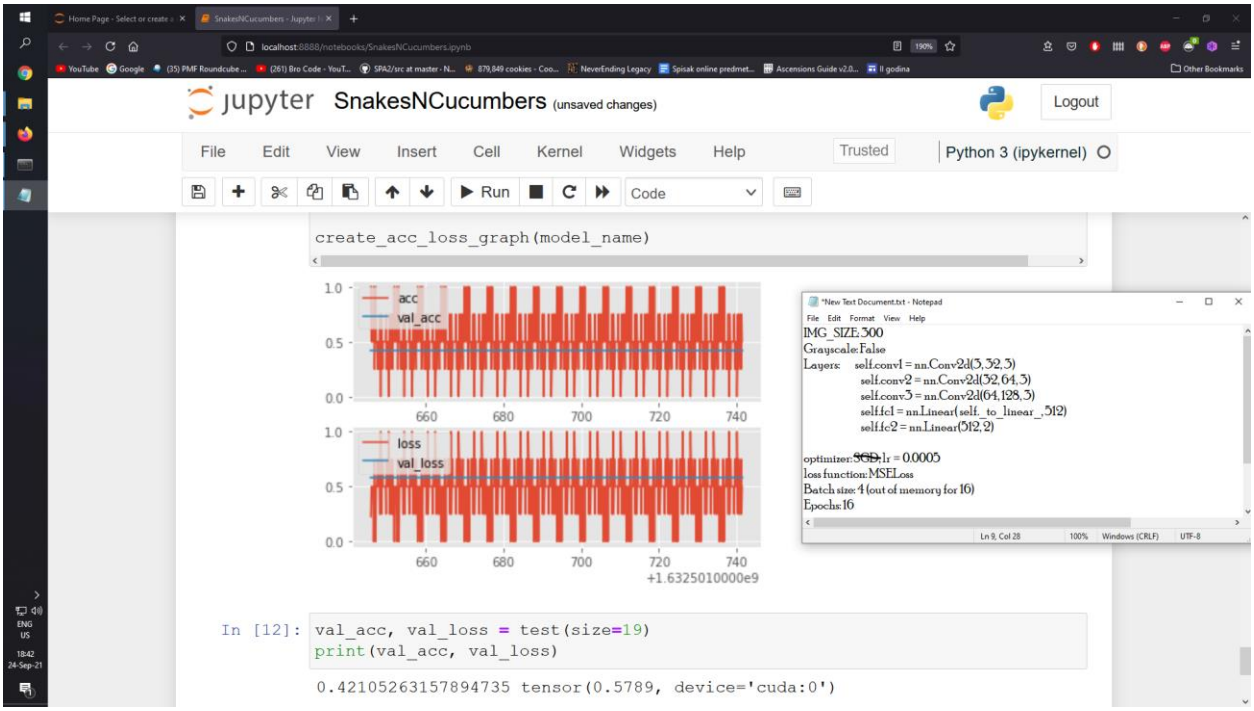
Ponovo sam dobio iste, odlicne, rezultate, koji potvrdjuju da boja stvarno nije bila bitna u ovom problemu.



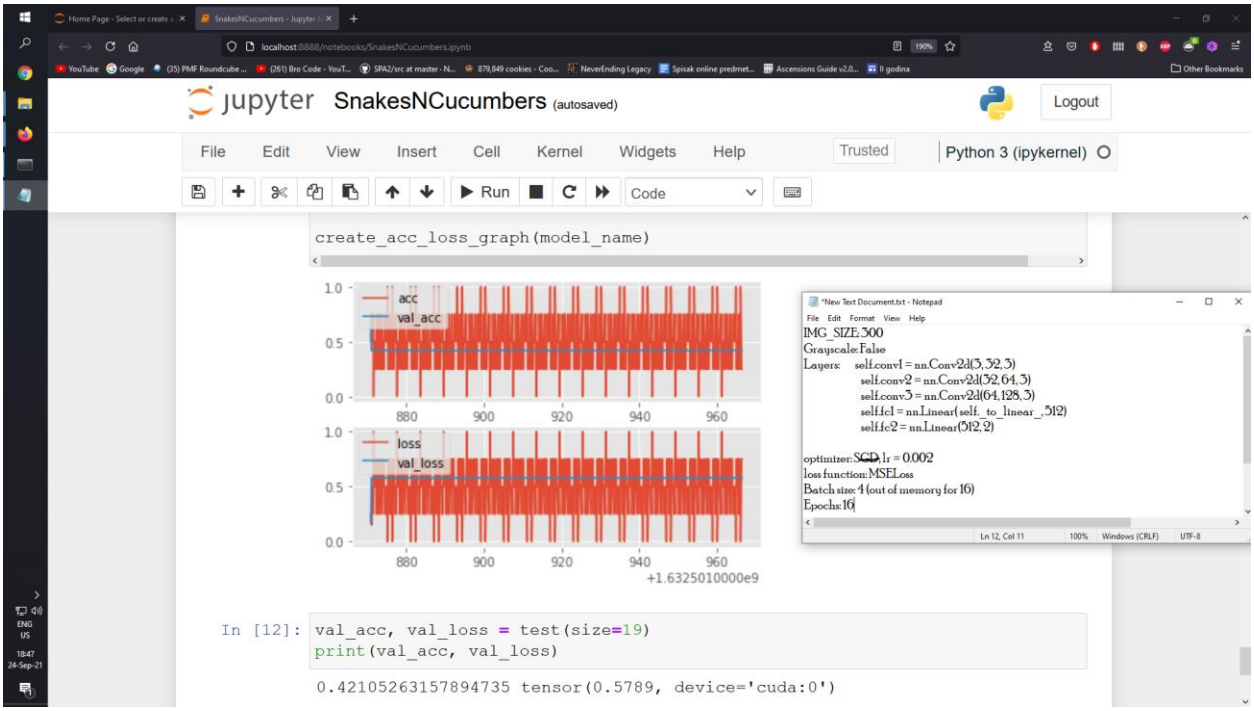
Isto kao malopre, ali sa vecom slikom. Analogni rezultati, kao sto sam se I nadao.



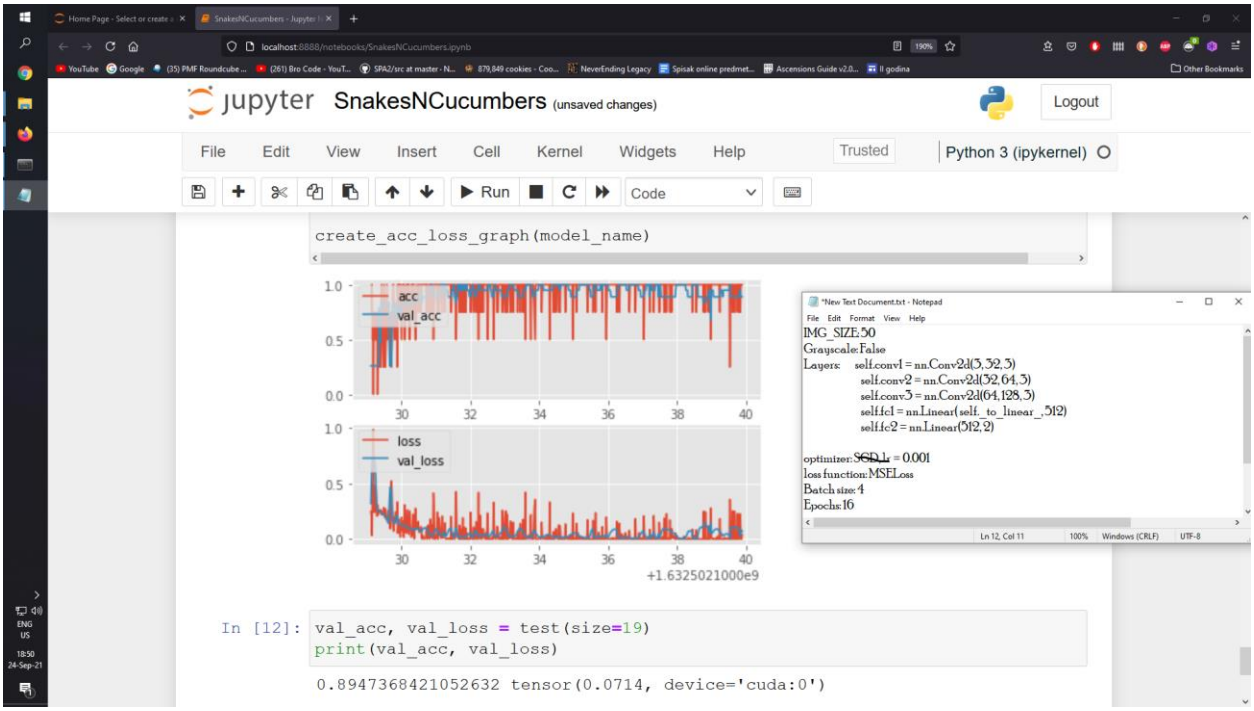
I kao i kada nismo imali boju, ako uz velike slike povecamo broj neurona, zaglavimo se u mestu. Boja na to ne utice.



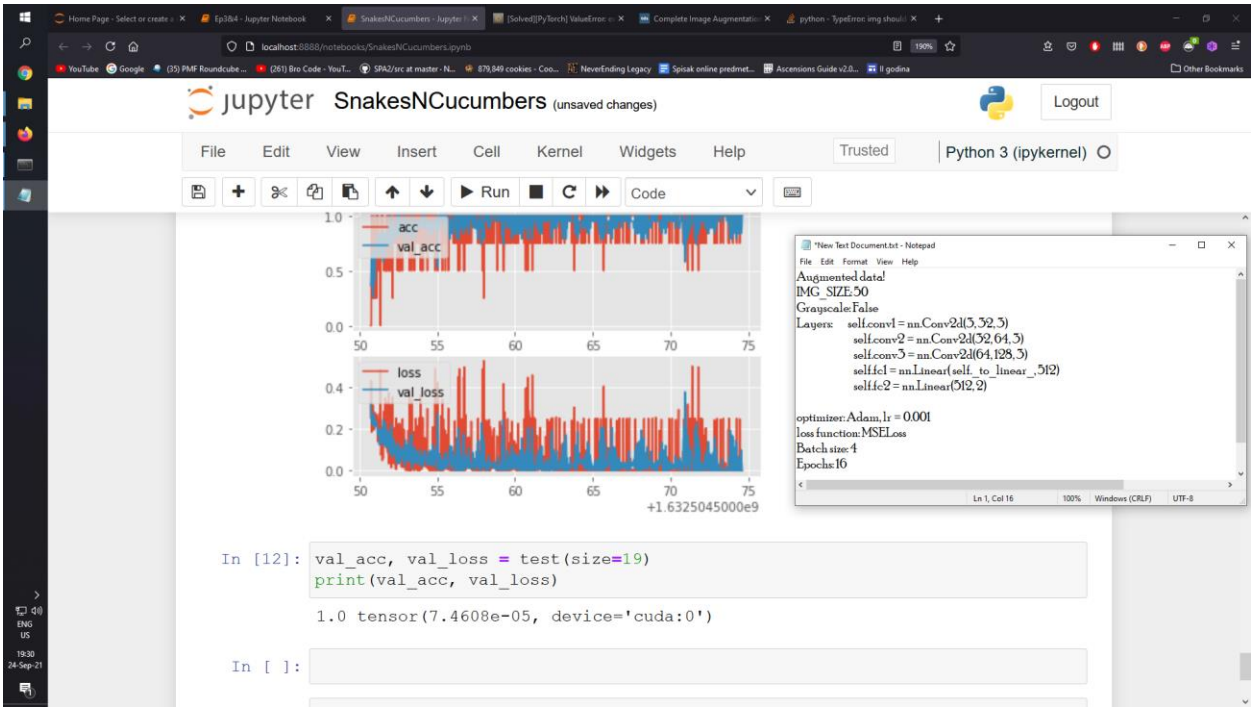
Probao sam da nadjem learning rate koji ce to resiti. Manji nije radio.



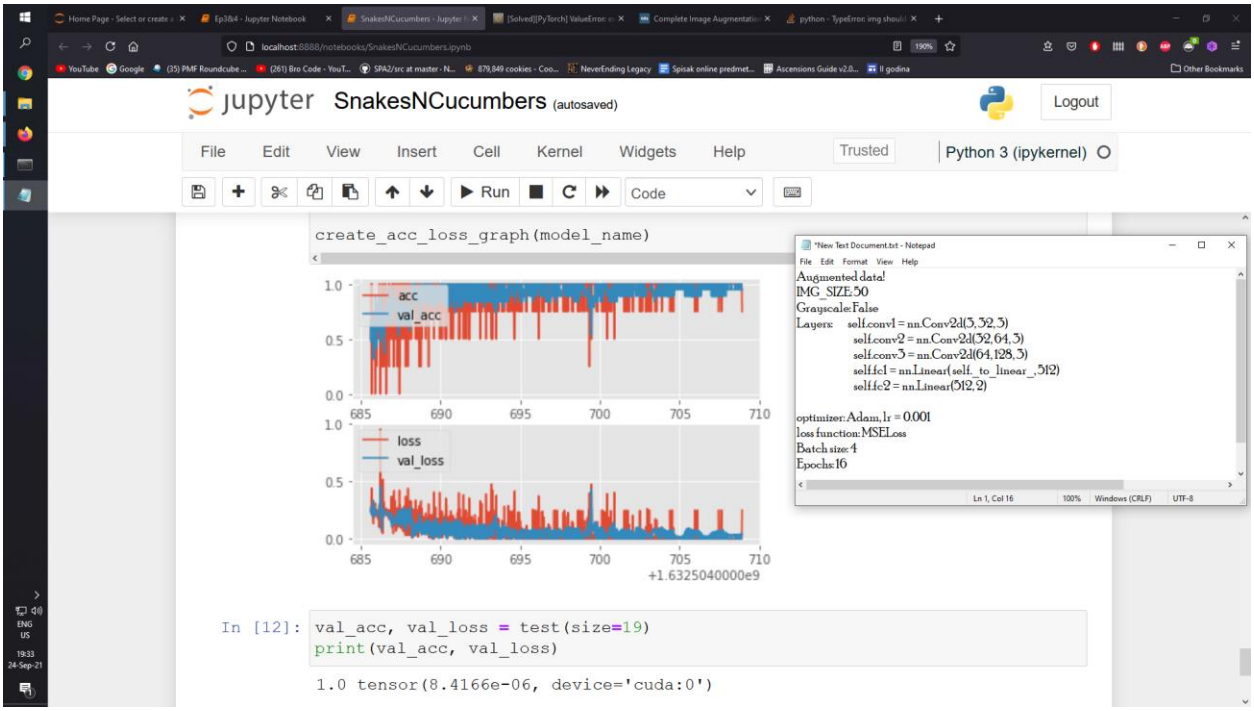
Veci takodje nije valjao.



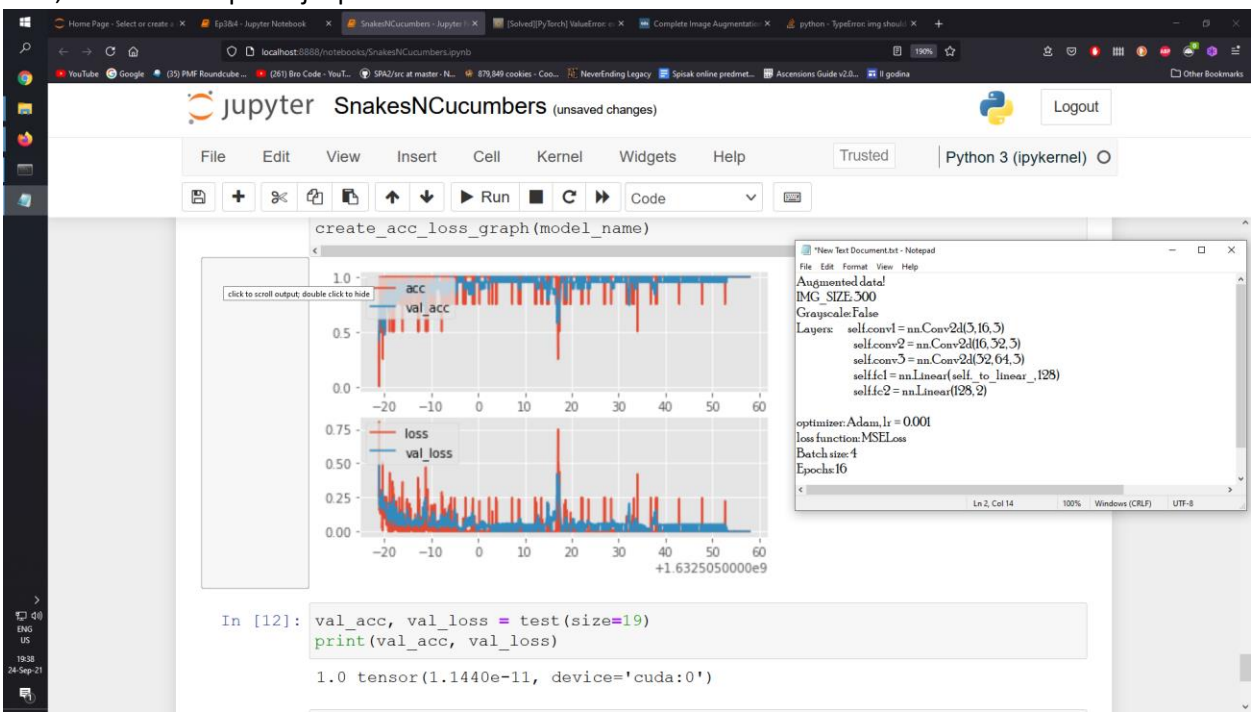
Medjutim, veca mreza je ponovo bila dobra uz manje slike.



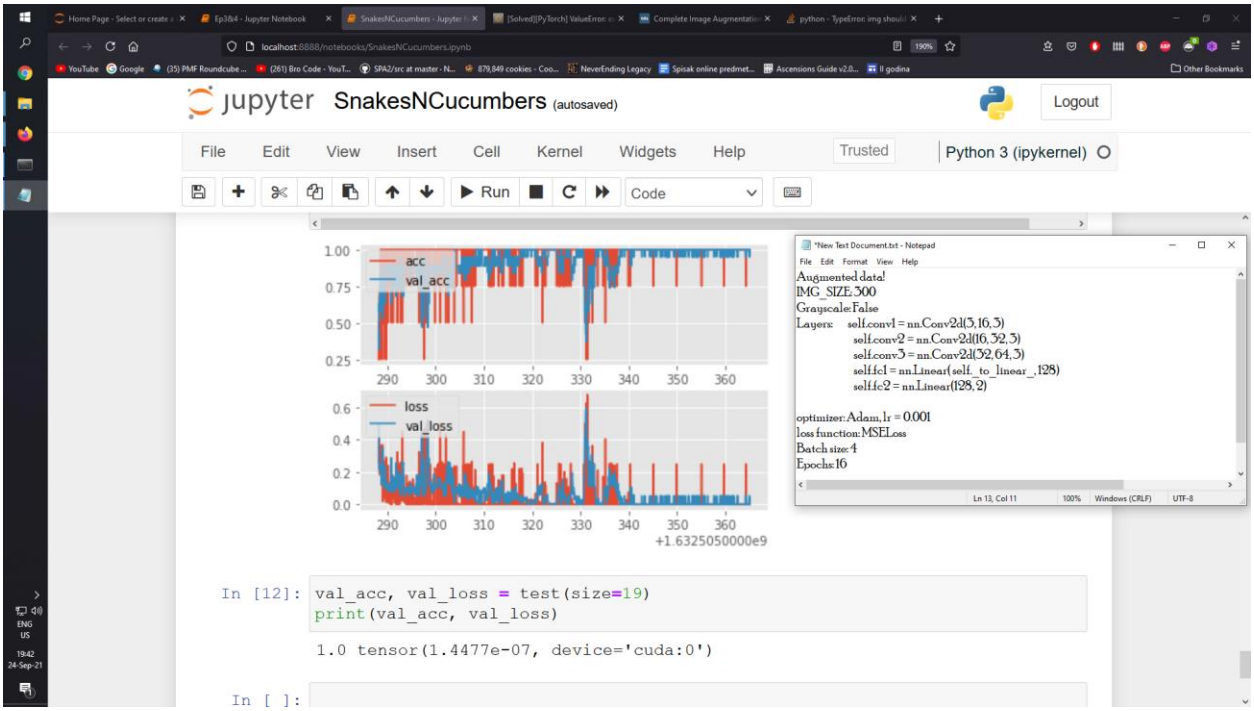
Sada sam presao na augmentovanje podataka. Nadao sam se da ce to uspeti da mi da vecu preciznost I manje razlike medju rezultatima zbog shuffle-ovanja redosleda slika, bez izazivanja overfitting-a. Tako I bese. Preciznost je I u predhodnim slucajevima bez dodatne data-e bila 100%, ali kao sto se vidi, loss je ovde mnogo bolji, sto znaci da je cak I na 100% preciznosti doslo do poboljsanja.



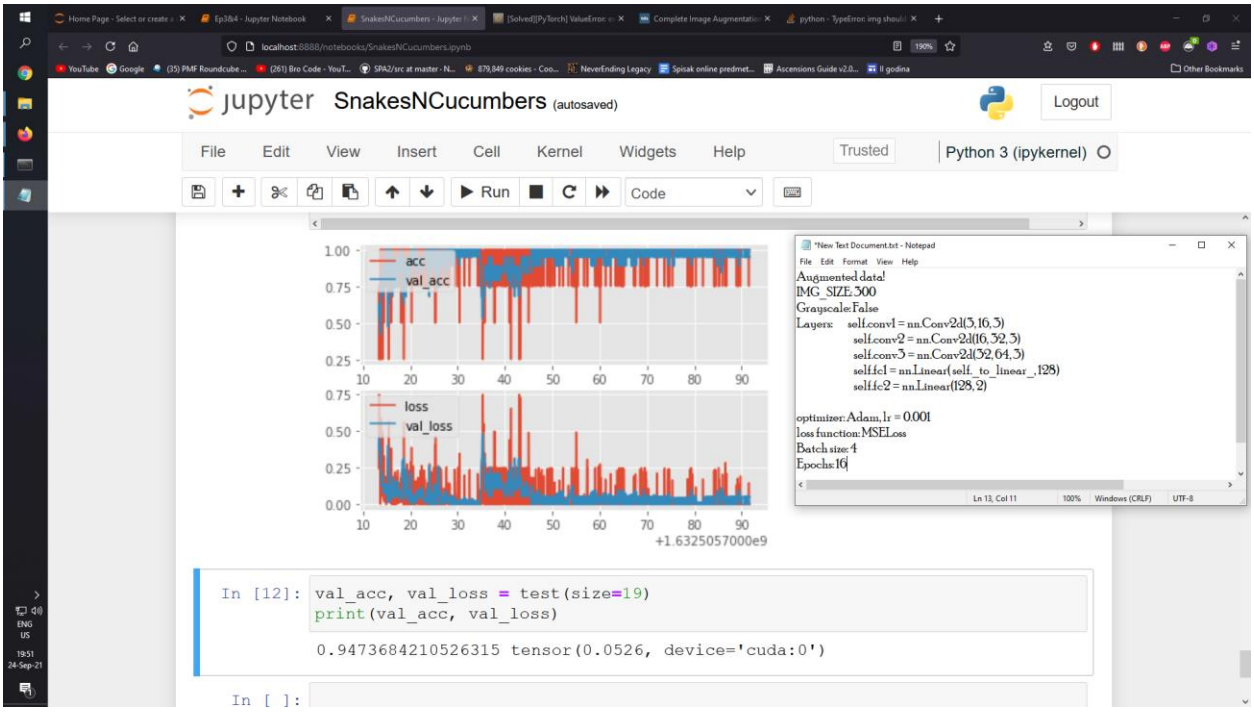
I rezultati su mnogo cesce bili slicni, za razliku od predhodnih puta gde sam pri istim parametrima dobijao mreze koje imaju bilo sta izmedju 80 – 100 % preciznosti. Nakon augmentacije imamo duplo vise slika, nasumicno flipovanjih po nasumicnim osama.



Najbolje se pokazala smanjena mreza za 300x300 slika. Sa dodatnom bojom i augemntovanim podacima, rezultati bi trebalo da su konstanto dobri.



Drugi pokusaj, isti parametri.



Treci pokusaj, isti parametri. Kao sto sam I ocekivao, sa vise podataka dolazi do mnogo manje varijabilnosti u rezultatima treninga. Ono sto me je iznenadilo je koliko je malo dodatnih slika potrebno za mnogo bolje rezultate.