Squeeze and Excite i SELU aktivacija
Tehnička dokumentacija
Verzija 1.0

Izradio: Igor Farszky

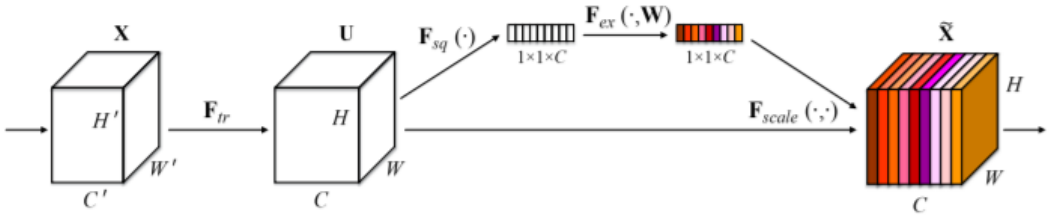
Nastavnik: Siniša Šegvić

Sadržaj

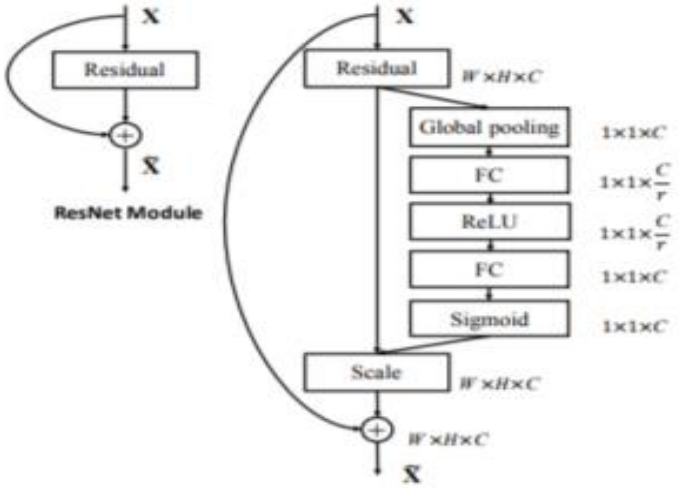
1. Opis razvijenog proizvoda

Sami zadatak projekta je bio istražiti metodu kojom se poboljšava učenje konvolucijske mreže koja je predstavljena u članku *Squeeze And Excite* [1]. Sama ideja se temelji na činjenici da se konvolucijski slojevi u mreži uče kako bi prepoznali lokalne aktivacije u ulaznoj slici te nemaju informaciju o cijelom kontekstu slike. Taj problem se želi riješiti povezanošću konvolucijskih slojeva, s navještavanjem njihovih značajki tako da nauče koristiti nekakvu globalnu informaciju s kojom bi pojačavali informativne značajke a potiskivali one manje značajne.

Grafički vrlo jednostavno ovu metodu opisuje slika 1, koja prikazuje kako se takvi SE blokovi mogu postaviti u mrežu u bilo kojem trenutku i na bilo koje mjesto. U ranijim slojevima, uči mrežu na više agnostični način, tako da mreža ne može razaznati da u slici nešto postoji, ali niti da ne postoji. Dok u kasnijim slojevima SE blokovi postaju veoma specijalizirani i djeluju vrlo osjetljivo na različite ulaze ovisno o njihovim razredima.



Slika 1 Ideja Squeeze And Excite bloka



Slika 2 Slojevi SE bloka ResNet50 modela

U samom programu je to implementirano uz pomoć TensorFlow programskog okvira s kojime se implementacija svodi na funkciju od par linija koda koja se zatim poziva na željenome mjestu u modelu mreže.

```
def squeeze_excite(self, input, filters=16, scope="SE",
se_conv_block_name="conv1"):

    filters1 = (int) (filters / constant.config['se_r'])

    with tf.variable_scope(scope) as scope:

        se = layers.global_pool(input, name="se_global_pool")
        se = layers.fc(se, filters1, activation_fn=layers.relu, name="se_fc_"
+ se_conv_block_name + "_1")
        se = layers.fc(se, filters, activation_fn=layers.sigmoid,
name="se_fc_" + se_conv_block_name + "_2")

    return se
```

Programski isječak 1 Squeeze And Excite blok u python-u

```
if constant.config['use_se'] == True:
    se = self.squeeze_excite(net[-1], filters=64)
    net += [tf.multiply(net[-1], se)]
```

Programski isječak 2 Poziv Squeeze And Excite bloka u modelu mreže

1.1 Squeeze

$$u_c = v_c * X = \sum_{s=1}^{C'} v_c^s * x^s \qquad (1)$$

Uc je rezultat konvolucije sa ulaznom slikom ili grupom slika X te težinama V_c veličine [batch_size x W x H x C], pri čemu je W širina slike, H visina slike, a C broj mapi značajki.

$$z_c = F_{sq}(u_c) = \frac{1}{W\ x\ H} \sum_{i=1}^{W} \sum_{j=1}^{H} u_c(i,j) \qquad (2)$$

Squeeze se računa kao “Global average pooling”, srednja vrijednost izlaza svih konvolucija (u_c) čime se na izlazu dobije matrica veličine [batch_size x 1 x 1 x C], gdje je C broj filtara konvolucije. Na taj način istisnemo nekakvu globalnu informaciju o slici.

1.2 Excite

Nakon istiskivanja važne informacije sa *Squeeze* operacijom, potrebno je vidjeti koji kanali su najznačajniji. Rekalibrirati ćemo slike tako da prvo izdvojimo samo [C/r] najvažnijih mapi značajki sa potpuno povezanim slojem i ReLU aktivacijom, a zatim vratimo nazad u [C] mapa značajki s drugim potpuno povezanim slojem te izračunamo važnost svake mape sa aktivacijskom funkcijom sigmoide koja će pojačati one doista važne mape (vrijednosti bliže 1), a smanjiti manje važne (vrijednost bliže 0). U jednadžbi (3) simbol * označava množenje u potpuno povezanom sloju, σ je funkcija sigmoide, W_1 su težine prvog potpuno povezanog sloja, a W_2 težine drugog potpuno povezanog sloja, z je rezultat Squeeze operacije.

$$s = F_{ex}(z, W) = \sigma\big(W_2 * ReLU(W_1 * z)\big) \qquad (3)$$

1.3 Skaliranje

Nakon Squeeze i Excite operacije s kojima imamo matricu veličine [batch_size x 1 x 1 x C] u kojoj smo izdvojili koje su nam mape važne. Sada tu informaciju primijeniti na daljnje učenje. To će se postići množenjem rezultata bloka SE s izlazom iz konvolucijskog sloja veličine [batch_size x W x H x C] prije primjene aktivacijske funkcije, koji je ujedno tako bio ulaz u SE blok, te zatim nad time se računa aktivacija. U jednadžbi (4) simbol s je rezultat Excite operacije, ujedno i izlaz SE bloka, te se matrično množi sa mapama značajki U.

$$\tilde{X} = F_{scale}(\mathrm{U}, s) = \mathrm{s} * U \qquad (4)$$

1.4 SELU

Do sada se u konvolucijskim neuronskim mrežama uglavnom koristila ReLU aktivacijska funkcija jer je davala najbolje rezultate. No i dalje se istražuju nove funkcije, te je jedan od primjera upravo SELU (eng. Scaled exponential linear units) [2] aktivacijska funkcija.

$$selu(x) = \lambda \begin{cases} x, & if\ x > 0 \\ \alpha e^x - \alpha, & if\ x \leq 0 \end{cases} \qquad (5)$$

Ideja SELU funkcije je da u ranijim slojevima smanjuje varijancu prema nuli, ako dođe

do pojave prevelike varijance, ili povećava s faktorom malo većim od 1 ako je varijanca premala. Simboli λ i α su konstante.

```
def selu(x):

    alpha = 1.6733
    scale = 1.0507

    return scale * tf.where(x > 0.0, x, alpha * tf.exp(x) - alpha, 'selu')
```

Programski isječak 3 SELU aktivacijska funkcija u Python-u

2. Tehničke značajke

Za izvođenje eksperimenata i treniranje implementiranog modela korišteno je računalo sa CPU Intel i7 2.4GHz, 16GB RAM-a i grafička kartica Nvidia GTX 1070.

2.1 Podaci

Eksperimenti su izvođeni na 3 skupa podataka. Mnist, CIFAR-10, CIFAR_128-10, ImageNet.

- Mnist: 55000 slika, veličine 28x28x1, 10 razreda (brojevi 0-1)
- CIFAR-10 (u nastavku cifar_128): 45000 slika, veličine 128x128x3, 10 razreda (objekti iz svijeta, npr. konj, auto, avion…)
- ImageNet: 14197122 slika, različitih veličina skalirane na 256x256x3, 1000 razreda (objekti iz svijeta)

2.2 Modeli

Sample model	SE_Sample model	ResNet50	SE_ResNet50
-	-	Conv-7x7, 64, 2 + BN + AF MaxPool-3x3, 2	
conv-5x5, 16 + AF maxpool-2x2, 2	conv-5x5, 16 + AF maxpool-2x2, 2	$\lceil conv-1x1, 64+BN+AF\rceil$ $\lvert conv-3x3, 64+BN+AF\rvert \quad x\ 3$ $\lfloor conv-1x1, 256+BN+AF\rfloor$	$\lceil conv-1x1, 64+BN+AF\rceil$ $\lvert conv-3x3, 64+BN+AF\rvert$ $\lvert conv-1x1, 256+BN\rvert \quad x\ 3$ $\lvert SE-16, 256\rvert$ $\lfloor AF\rfloor$
conv-5x5, 32 + AF maxpool-2x2, 2	conv-5x5, 32 + AF maxpool-2x2, 2	$\lceil conv-1x1, 128+BN+AF\rceil$ $\lvert conv-3x3, 128+BN+AF\rvert \quad x\ 4$ $\lfloor conv-1x1, 512+BN+AF\rfloor$	$\lceil onv-1x1, 128+BN+AF\rceil$ $\lvert conv-3x3, 128+BN+AF\rvert$ $\lvert conv-1x1, 512+BN\rvert \quad x\ 4$ $\lvert SE-32, 512\rvert$ $\lfloor AF\rfloor$
conv-5x5, 64 + AF maxpool-2x2, 2	conv-5x5, 64 + AF maxpool-2x2, 2	$\lceil conv-1x1, 256+BN+AF\rceil$ $\lvert conv-3x3, 256+BN+AF\rvert \quad x\ 6$ $\lfloor conv-1x1, 1024+BN+AF\rfloor$	$\lceil conv-1x1, 256+BN+AF\rceil$ $\lvert conv-3x3, 256+BN+AF\rvert$ $\lvert conv-1x1, 1024+BN\rvert \quad x\ 6$ $\lvert SE-64, 1024\rvert$ $\lfloor AF\rfloor$
-	SE-16, 64	$\lceil conv-1x1, 512+BN+AF\rceil$ $\lvert conv-3x3, 512+BN+AF\rvert \quad x\ 3$ $\lfloor conv-1x1, 2048+BN+AF\rfloor$	$\lceil conv-1x1, 512+BN+AF\rceil$ $\lvert conv-3x3, 512+BN+AF\rvert$ $\lvert conv-1x1, 2048+BN\rvert \quad x\ 3$ $\lvert SE-128, 2048\rvert$ $\lfloor AF\rfloor$
FC-256 + AF FC-128 + AF FC-c	FC-256 + AF FC-128 + AF FC-c	global_avg_pool-7x7, 1 If dataset ϵ (mnist, cifar) := $\begin{matrix}\text{FC}-512+AF\\ \text{FC}-256+AF\\ \text{FC}-c\end{matrix}$ Else : FC-1000	

Tablica 1 Arhitektura modela

Aktivacijska funkcija (AF) – ovisno o početnoj konfiguraciji koristi se ReLU ili SeLU
Batch normalization (BN) – normalizacija mini batch-a
Konvolucijski sloj (conv) – veličina konvolucije, broj filtara, korak konvolucije
Sloj sažimanja (maxpool, global_avg_pool) – veličina sažimanja, korak sažimanja
Potpuno povezani sloj (FC) – veličina izlaza, c označava broj razreda
Squeeze And Excite (SE) – veličina izlaza prvog i drugog potpuno povezanog sloja

2.3 Rezultati

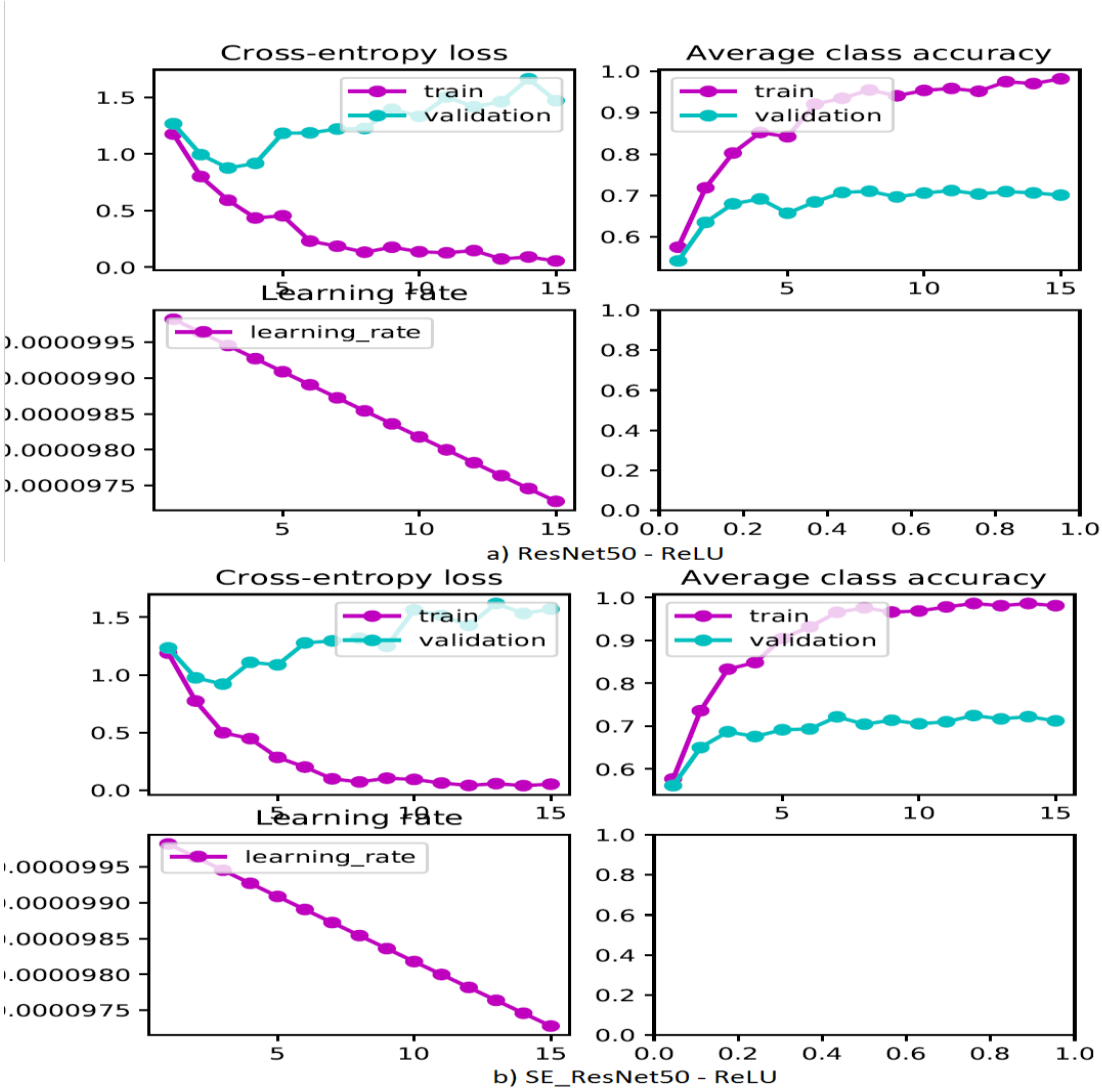
Svi rezultati su dobiveni eksperimentiranjem na skupu za treniranje i validaciju. Za gubitak je izabran najmanji gubitak tijekom svih epoha treniranja. Za točnost modela je izabran najveći postotak tijekom svih epoha treniranja. Vrijeme se odnosi na ukupno vrijeme kroz sve epohe treniranja bez vremena evaluacije. Prilikom treniranja modela ResNet50 na skupu podataka Mnist, nakon 6. epohe je došlo do divergencije.

	Gubitak				Točnost [%]				Vrijeme [hh:mm]	CPU/ GPU	Broj epoha
	relu		selu		relu		selu				
	train	valid	train	valid	train	valid	train	valid			
Sample/Mnist	0.0131	0.0250	0.0128	0.0221	99.85	99.05	**99.91**	98.45	00:28:40	CPU	20
SE_Sample/Mnist	0.0142	0.1600	0.0125	0.0458	**99.87**	98.87	99.90	98.75	00:34:20	CPU	20
ResNet50/Mnist	0.0227	0.0230	0.0110	**0.0120**	99.48	**99.06**	99.20	**99.18**	09:37:12	CPU	10
SE_ResNet50/Mnist	**0.0095**	**0.0098**	**0.0074**	0.0255	99.47	98.96	99.45	98.90	10:13:50	CPU	10
Sample/CIFAR_128	0.3363	1.7157	NaN	NaN	89.34	42.22	**10.10**	10.58	01:25	GPU	15
SE_Sample/CIFAR_128	0.1861	1.6097	-	-	94.27	48.42	-	-	01:08	GPU	15
ResNet50/ CIFAR_128	0.0553	0.9945	NaN	NaN	98.18	71.22	10.08	**10.70**	01:29	GPU	15
SE_ResNet50/ CIFAR_128	**0.0409**	**0.9209**	-	-	**98.62**	**72.22**	-	-	01:46	GPU	15
Sample/ImageNet	-	-	-	-	-	-	-	-	-	-	-
SE_Sample/ImageNet	-	-	-	-	-	-	-	-	-	-	-
ResNet50/ ImageNet	-	-	-	-	-	-	-	-	-	-	-
SE_ResNet50/ ImageNet	-	-	-	-	-	-	-	-	-	-	-

Tablica 2 Rezultati eksperimenata

Izdvojeni rezultati su najbolji dobiveni na danome skupu (mnist, cifar – train,valid. Slike iz skupa Mnist su prejednostavne te se svi modeli vrlo lagano nauče i postignu visoku točnost, te ćemo obraćati pozornost samo na CIFAR skup podataka. Lako se zaključi da "Sample" model mreže nije dobar, jer ne generalizira dobro podatke što možemo vidjeti iz niskog postotka točnosti na CIFAR slikama. Prilikom treniranja s aktivacijskom funkcijom SELU, model je divergirao te je točnost pripadno tome vrlo mala i na skupu za treniranje i validaciju. Točan razlog tome se ne zna. Stoga je jedino vrijedno komentirati rezultate na modelu „ResNet50", skupu slika CIFAR i aktivacijskoj funkciji ReLU. Gubitak je dakako puno manji na skupu za treniranje, te točnost od 98.62% nam govori da model vrlo dobro nauči podatke. S druge strane, na validacijskome skupu, gubitak je još uvijek malen, ali točnost od 72.22% je poprilično manja, no i dalje s tom brojkom bi mogli reći da model dosta dobro generalizira slike, no po navodima raznih izvora ona bi trebala biti još veća, pri čemu se može zaključiti da postoji pogreška u implementaciji modela

ResNet50 u samom programu. U usporedbi tih rezultata, koji su dobiveni prilikom korištenja Squeeze And Excite blokova i rezultata bez blokova, vidimo da smo dobili malo poboljšanje, gubitak je za 0.07 manji te točnost je veća za cijelih 1%, što nam zapravo navodi da SE blokovi doista unose malo poboljšanje u treniranje, te s obzirom da postoji greška u implementaciji, ta razlika bi mogla biti još i veća. Ako pogledamo kolika nam je dobit generalizacije naprama vremenu treniranja, mogli bi reći da se isplati unositi takve blokove, jer 1% razlike nije zapravo toliko malena dobit, a dobiva se sa samo 17 minuta više treniranja, odnosno 1 minuti više po epohi.



Slika 2 Rezultati treniranja modela ResNet50 – CIFAR

3. Upute za korištenje

Program je napisan u programskom jeziku Python 3.5.3. koristeći programski okvir TensorFlow za implementaciju većine operacija s matricama i samom neuronskom mrežom. Osim TensorFlow-a, program koristi i biblioteke poput numpy, os, matplotlib, skimage, sklearn, pickle…

Razred „Model“ u datoteci „Model.py“ implementira model na kojem se uče podaci. Pruža 2 implementacije, svojevoljni model u funkciji “model_net()” koji se sastoji od 3 konvolucijska sloja s pripadno izabranom aktivacijskom funkcijom , te iza svakog konvolucijskog sloja slijedi sloj sažimanja maksimalnom vrijednošću, zatim u ovisnosti dali je izabrano hoće li se koristiti “Squeeze And Excite” slijedi SE blok, i završava s 3 potpuno povezana sloja. Za optimizaciju gubitka koristi se adaptivni postupak optimizacije ADAM. Ovu mrežu je korisnik slobodan mijenjati na njemu odgovarajuće načine. Druga implementacija je ResNet50 predstavljena u članku “Deep Residual Learning for Image Recognition” [3] koju ne bi trebalo mijenjati.

Mreži se mogu vrlo jednostavno mijenjati parametri prije početka samog treniranja. U datoteci „constant.py“ se nalaze svi potrebni parametri i hiperparametri kojima se direktno utječe na model mreže.

a) Skup podataka za učenje

```
config['datasets'] = {'mnist' : 'mnist', 'cifar_32' : 'cifar_32', 'cifar_128' : 'cifar_128',
'imagenet' : 'imagenet'}
config['dataset_name'] = config['datasets']['cifar_128']
```

b) Direktoriji za spremanje rezultata treniranja i validacije

```
DATA_DIR = 'datasets/' + config['dataset_name'] + '/data_dir'
SAVE_DIR = 'datasets/' + config['dataset_name'] + '/save_dir'
FILTERS_SAVE_DIR = 'datasets/' + config['dataset_name'] + '/save_dir/filters'
PLOT_TRAINING_SAVE_DIR = 'datasets/' + config['dataset_name'] +
'/save_dir/plot_training'
```

c) Modeli mreže

```
config['models'] = {'custom_model' : 'custom_model', 'resnet50' : 'resnet50'}
config['model'] = config['models'][ 'resnet50']
```

d) Aktivacijska funkcija

```
config['activation_functions'] = {'selu': layers.selu, 'relu': layers.relu}
```

config['activation_fn'] = 'relu'

e) Hiperparametri

config['learning_rate'] = 1e-4 – stopa učenja
config['batch_size'] = 50 – veličina grupe za učenje u jednom koraku
config['max_epochs'] = 10 – ukupan broj iteracija učenja po svim podacima
config['output_shape'] = 10 – izlaz zadnjeg potpuno povezanog sloja
config['pool_size'] = [2, 2] – veličina sažimanja (koristi se samo u slojevima sažimanja)
config['strides'] = 2 – korak (koristi se samo u slojevima sažimanja)
config['decay_steps'] = 100000 – broj koraka propadanja stope učenja (postupak optimizacija s eksponencijalnim propadanjem stope učenja)
config['decay_base'] = 0.96 – baza propadanja (postupak optimizacija s eksponencijalnim propadanjem stope učenja)
config['se_r'] = 16 – parameter redukcije parametara SE bloka
config['use_se'] = False – izbor korištenja SE bloka u modelu mreže
config['log_every'] = 2500 – ispis stanja treniranja svakih x podataka

Nakon postavljanja svih željenih parametara program se pokreće pomoću skripte „main.py“ kojoj se ne prosljeđuju nikakvi parametri za pokretanje. Kada se program pokrene u konzoli ispisuje trenutno stanje treniranja.

```
epoch 10, step 40000 / 45000, loss = 0.19 (3.195 sec/batch)
epoch 10, step 42500 / 45000, loss = 0.33 (3.394 sec/batch)
epoch 10, step 45000 / 45000, loss = 0.67 (3.219 sec/batch)
EPOCH STATISTICS :
Train error: epoch 10 loss=0.20710094273090363 accuracy=0.9325555555555556
Validation error: epoch 10 loss=1.129801630973816 accuracy=0.6886
Plotting in:  datasets/cifar/save_dir/plot_training\resnet50_relu.pdf
```

Slika 3 Ispis evaluacije treniranja

Najvažnija python skripta koja se pokreće prilikom početka treniranja modela je main.py. U njoj se određuje naprama parametrima u constant.py, koji dataset će se inicijalizirati za treniranje te koja implementacija treniranja. Za skupove Mnist, CIFAR_32 se koristi učitavnje cijelog dataseta u numpy polja, dok za skup CIFAR_128 radi veličine slika kako se nebi napunila RAM memorija se koristi razred TFRecordsReader iz datoteke TFRecordsReader.py, koja učitava podatke u grupama od zadane veličine „batch_size“ direktno iz datoteke koja mora biti tipa „.tfrecords“.

```
def create_iterator(self, dataset_type, num_epochs, batch_size, capacity):

    self.feature = {dataset_type + '/image': tf.FixedLenFeature([],
tf.string),
                dataset_type + '/label': tf.FixedLenFeature([], tf.int64)}
```

```
    self.filename_queue =
tf.train.string_input_producer(self.fileDict[dataset_type], shuffle=True,
num_epochs=num_epochs)

    self.reader = tf.TFRecordReader()
    _, ser_example = self.reader.read(self.filename_queue)

    self.features = tf.parse_single_example(ser_example,
features=self.feature)

    self.image = tf.decode_raw(self.features[dataset_type + '/image'],
tf.float32)
    self.label = tf.cast(self.features[dataset_type + '/label'], tf.int32)

    self.image = tf.reshape(self.image, [128, 128, 3])

    self.images, self.labels = tf.train.shuffle_batch([self.image,
self.label], batch_size=batch_size,
                                                       capacity=capacity,
num_threads=1, min_after_dequeue=50)
```

Programski isječak 4 Čitanje podataka po grupama direktno iz datoteke

Nakon učitavanja podataka, ostatak treniranja se provodo uvijek na jednak način. U konstruktoru se inicijalizira model koji se trenira, točnije računski graf koji kreira Tensorflow, te ga pokuša inicijalizirati uz pomoć dostupne GPU jedinice ako ju pronađe, u protivnom će koristiti CPU.

```
def __init__(self):

    with tf.device('/device:GPU:0'):
        self.model = model.Model()

    config = tf.ConfigProto(allow_soft_placement=True,
log_device_placement=True)
    config.gpu_options.per_process_gpu_memory_fraction = 0.4

    self.sess = tf.Session(config=config)
```

Programski isječak 5 Inicijalizacija računskog grafa

Treniranje se izvodi po epohama i grupama. Nakon svakih n grupa se ispisuje redni broj epohe, broj naučenih slika i virjeme za treniranje jednge grupe. Prilikom završetka jedne epohe se izvodi evaluacija naučenog modela te se računa gubitak i točnost na skupu za treniranje i validaciju. Prilikom završetka ukupnog treniranja iscrtava se graf u pdf dokument koji prikazuje ovisnost gubitka i točnosti u trenutku određene epohe po skupu za treniranje i skupu za validaciju.

```
for epoch_num in range(1, max_epochs + 1):

    epoch_start_time = time.time()

    for step in range(num_batches):
```

```
        batch_x, batch_y = self.sess.run([self.tfTrainReader.images,
self.tfTrainReader.labels])
        batch_y = util.class_to_onehot(batch_y, constant.config['num_class'])

        start_time = time.time()

        ret_val = self.sess.run([self.model.train_op, self.model.loss,
self.model.logits],
                                feed_dict={self.model.X: batch_x,
self.model.Yoh: batch_y})
        _, loss_val, logits_val = ret_val

        duration = time.time() - start_time

        if (step + 1) * batch_size % log_every == 0:
            sec_per_batch = float(duration)
            self.log_step(epoch_num, step, batch_size, num_batches *
batch_size, loss_val, sec_per_batch)

    epoch_time = time.time() - epoch_start_time
    plot_data['epoch_time'] += [epoch_time]

    print("EPOCH STATISTICS : ")

    train_loss, train_acc = self.validate(epoch_num, self.tfTrainEvalReader,
"train")
    valid_loss, valid_acc = self.validate(epoch_num, self.tfValidEvalReader,
"valid")
    print("Total epoch time training: {}".format(epoch_time))

    lr = self.sess.run([self.model.learning_rate])

    plot_data['train_loss'] += [train_loss]
    plot_data['valid_loss'] += [valid_loss]
    plot_data['train_acc'] += [train_acc]
    plot_data['valid_acc'] += [valid_acc]
    plot_data['lr'] += [lr]

util.plot_training_progress(plot_data)
```

Programski isječak 6 Treniranje modela

Važna datoteka je layers.py u kojoj se nalaze metode koje implementiraju pojedine slojeve modela mreže, kao što su aktivacijske funkcije relu, selu, sigmod, konvolucijski sloj conv2d, slojevi sažimanja max_pool2d i global_pool, pretvaranje matrice slike u vektor odnosno matricu u slučaju grupe slike te se funkcija zove flatten, zatim potpuno povezani sloj fc i normalizaciju mini grup batchNormalization. Sve metode u pozadini koriste implementacije paketa „tensorflow.layers“.

```
def conv2d(input, filters=16, kernel_size=[5, 5], padding="same", stride=1,
activation_fn=relu, name="conv2d"):

   return tf.layers.conv2d(input, filters=filters, kernel_size=kernel_size,
```

```
padding=padding,
                            strides=stride, name=name,
activation=activation_fn)
```

Programski isječak 7 Primjer funkcije koja kreira konvolucijski sloj modela

Uz same razrede i funkcije koje su potrebne za treniranje mreže, korištena su i dva razreda za manipulaciju podataka. ImageResizer i TFRecordsWriter. ImageResizer služi kako bi uz pomoć Tensorflow-ovih funkcija na pametan način promijenio veličinu slike, te zadržao njene značajke, a to se izvodi pomoću bikubične interpolacije. ImageResizer je korišten prilikom povećavanja CIFAR slika sa 32x32x3 na veličinu 128x128x3 slikovnih elemenata.

```
batch_size = 10000
w = 128; h = 128; c = 3

with tf.variable_scope('image_resizer') as scope:

    tf_batch = tf.placeholder(dtype=tf.float32, shape=[batch_size, 32, 32,
3])
    resized_images = tf.image.resize_images(tf_batch, [w, h],
method=tf.image.ResizeMethod.BICUBIC)
    tf_data = tf.reshape(tf.transpose(resized_images, [0, 3, 1, 2]),
[batch_size, w * h * c])
    tf_data = tf.cast(tf_data, tf.uint8)

session = tf.Session()
session.run(tf.global_variables_initializer())
```

Programski isječak 8 Primjer funkcije koja kreira konvolucijski sloj modela

Nakon promjene veličine slika, TFRecordsWriter kreira datoteke tipa „.tfrecords“ koja je zapravo binarna datoteka i služi za učitavanje slika u grupama. TFRecordsWriter sprema niz mapa u kojima je ključna riječ niz znakova, te predstavlja identifikator da li se vrijednost odnosi na sliku ili oznaku razreda slike, a vrijednosti su slika u bajtovima ili broj razreda slike int64.

```
feature = {dataset_type + '/label' : util._int64_feature(np.asscalar(label)),
           dataset_type + '/image' :
util._bytes_feature(tf.compat.as_bytes(image.tostring()))}
```

Programski isječak 9 Zapis slike i njene oznake u binarnoj datoteci „.tfrecords“

4. Literatura

[1] Jie Hu, Li Shen, Gang Sun, Squeeze-and-Excitation Networks, 5. Rujna 2017, https://arxiv.org/pdf/1709.01507.pdf, 24.12.2017.

[2] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, Self-Normalizing Neural Networks, 7. Rujna 2017, https://arxiv.org/pdf/1706.02515.pdf, 24.12.2017.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition, 10. Prosinca 2015, https://arxiv.org/pdf/1512.03385.pdf, 24.12.2017.

Sl. 1 Squeeze And Excite blok (Jie Hu, Li Shen, Gang Sun, Squeeze-and-Excitation Networks, 5. Rujna 2017, https://arxiv.org/pdf/1709.01507.pdf, 24.12.2017, Sl. 1)

Sl. 2 Squeeze And Excite blok (Jie Hu, Li Shen, Gang Sun, Squeeze-and-Excitation Networks, 5. Rujna 2017, https://arxiv.org/pdf/1709.01507.pdf, 24.12.2017, Sl. 3)