

## Stranice predmeta Duboko učenje (FER)

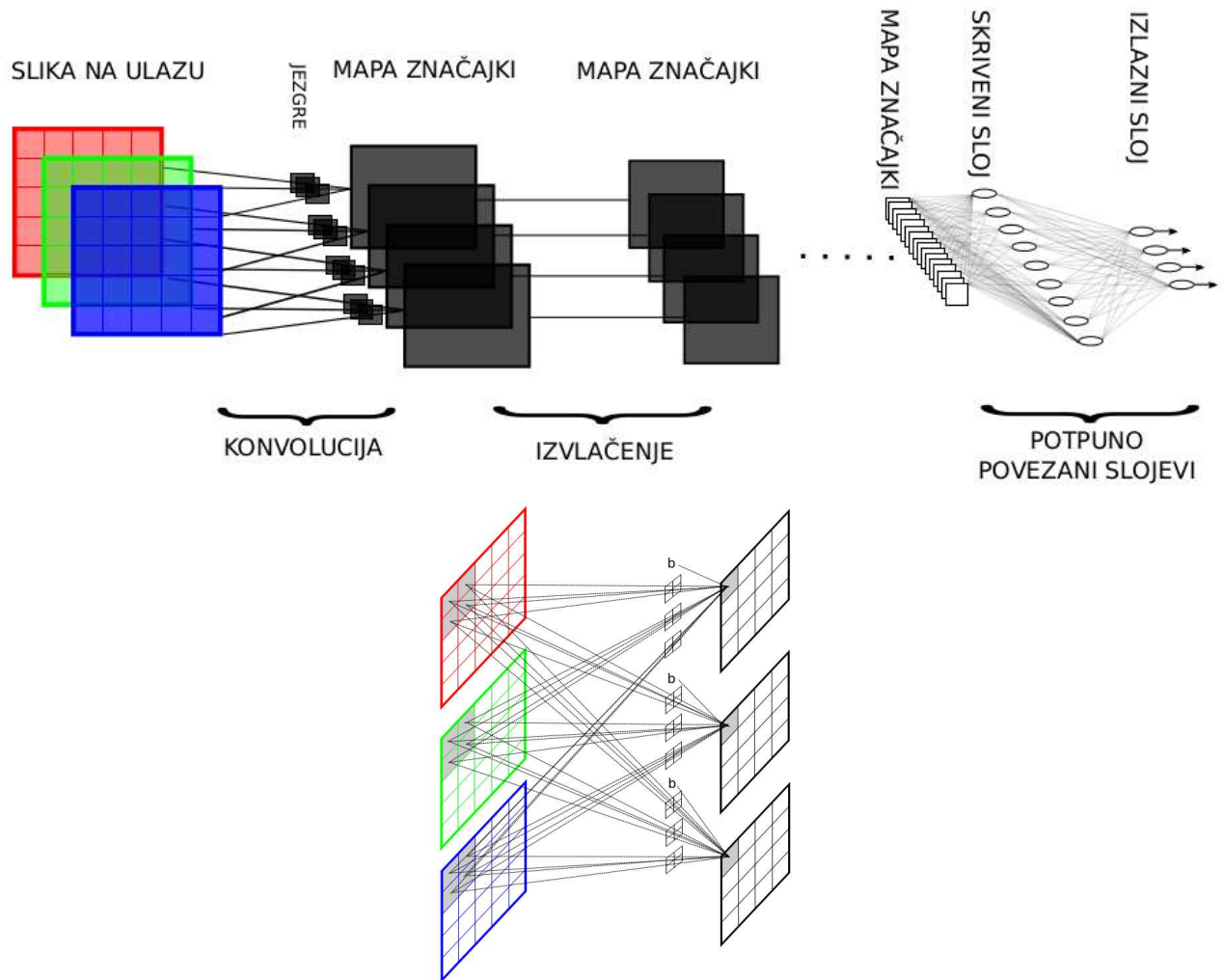
---

- Konvolucijska neuronska mreža
- Vježba
  - 1. zadatak
  - 2. zadatak
  - 3. zadatak
  - 4. zadatak
- Dodatni materijali

## 2. vježba: konvolucijske neuronske mreže (CNN)

U ovoj vježbi bavimo se konvolucijskim neuronskim mrežama. Konvolucijske mreže zamišljene su za obradu podataka koji imaju posebnu topologiju gdje je osobito važno ostvariti invarijantnost na translaciju (problem miješanja dimenzija u podacima). Dobar primjer takvih podataka su slike gdje se obično isti objekt može pojaviti na bilo kojem mjestu unutar slike. Ako bismo takvu sliku poslali na ulaz potpuno povezanog sloja tada bi jedan neuron vidio sve piksele slike. Takav pristup bi omogućio svim neuronima da se specijaliziraju za značajke objekta u djelovima slike u kojem se objekt pojavio što bi na kraju rezultiralo prenaučenošću i model bi loše generalizirao. Osim toga dodatni problem je što slike obično sadrže puno piksela. Na primjer, prosječne dimenzije slike iz poznatog dataseta ImageNet iznose  $3 \times 200 \times 200$  što znači da bi u tom slučaju jedan neuron u prvom sloju morao imati  $3 \times 200 \times 200 = 120,000$  težina. S većim brojem neurona bismo jako brzo ostali bez memorije.

Vidimo da bismo bili u puno boljoj situaciji ako bismo ostvarili da svaki neuron djeluje lokalno na samo jedan dio slike. Na taj način bi neuron imao rijetku povezanost sa slikom što bi uvelike smanjilo broj težina. Ideja je da neuroni imaju jako mala receptivna polja što znači da bi u prvom sloju mogli opisivati samo značajke jako niske razine poput linija i rubova. Kasniji slojevi bi imali sve veće receptivno polje što bi omogućilo da hijerarhijski grade kompleksnije značajke na temelju jednostavnijih. Dodatno, budući da želimo postići invarijantnost na translacije unutar slike i dalje želimo da svaki neuron djeluje nad čitavom slikom. To možemo ostvariti tako da za svaki neuron umjesto jednog izlaza kao do sada imamo više izlaza. Svaki izlaz tada bi odgovarao odzivu na drugom položaju u slici. Ovime smo postigli da se parametri jednog neurona dijele preko čitave slike. Neurone u konvolucijskim slojevima mreže obično nazivamo i filtrima. Konvolucijske mreže koriste tri važne ideje: rijetku povezanost, dijeljenje parametara i ekvivarijantnost reprezentacije.



Primjer konvolucijske mreže za klasifikaciju slika. Tipično se izmjenjuju konvolucijski slojevi i slojevi sažimanja (izvlačenja). Na kraju se dolazi do vektora značajki koji se potpuno povezanim slojem preslikava u konačnu distribuciju preko razreda.

(Preuzeto iz [diplomskog rada](#).)

## Vježba

Kod za prva dva zadatka nalazi se [ovdje](#). Biblioteke koje su vam potrebne za ovu vježbu su Tensorflow, NumPy, [Cython](#), [matplotlib](#) i [scikit-image](#). Pazite da sve biblioteke instalirate za Python 3. U datoteci `layers.py` nalaze se slojevi od kojih se tipično sastoji CNN. Svaki sloj sadrži dvije metode potrebne za izvođenje backpropagation algoritma. Metoda `forward` izvodi unaprijedni prolazak kroz sloj i vraća rezultat. Metode `backward_inputs` i `backward_params` izvode unazadni prolazak. Metoda `backward_inputs` računa gradijent s obzirom na ulazne podatke ( $\frac{\partial L}{\partial \mathbf{x}}$  gdje je  $\mathbf{x}$  ulaz u sloj). Metoda `backward_params` računa gradijent s obzirom na parametre sloja ( $\frac{\partial L}{\partial \mathbf{w}}$  gdje vektor  $\mathbf{w}$  vektor predstavlja sve parametre sloja)).

## 1. zadatak

Dovršite implementacije potpuno povezanog sloja, sloja nelinearnosti te funkcije gubitka u razredima `FC`, `ReLU` i `SoftmaxCrossEntropyWithLogits`. Podsjetimo se, gubitak unakrsne entropije računa udaljenost između točne distribucije i distribucije koju predviđa model i definiran je kao:

$$L = - \sum_{i=1}^C y_i \log(s_i(\mathbf{x}))$$

gdje je  $C$  broj razreda,  $\mathbf{x}$  ulaz funkcije softmaks kojeg možemo zvati klasifikacijska mjera ili logit,  $\mathbf{y}$  točna distribucija preko svih razreda za dani primjer (najčešće one-hot vektor), a  $s_i(\mathbf{x})$  izlaz Softmax funkcije za razred  $i$ . Radi jednostavnosti prikazali smo funkciju gubitka za samo jedan primjer dok u praksi definiramo gubitak nad skupom primjera pa će ukupan gubitak obično biti jednak prosječnom gubitku preko svih primjera. Da biste izveli unazadni prolazak kroz sloj potrebno je najprije izračunati gradijent ove funkcije s obzirom na ulaz  $\frac{\partial L}{\partial \mathbf{x}}$ . Postupak derivacije možemo pojednostavniti tako da uvrstimo definiciju Softmax funkcije:

$$\begin{aligned} \log(s_i(x)) &= \log\left(\frac{e^{x_i}}{\sum_{j=1}^C e^{x_j}}\right) = x_i - \log \sum_{j=1}^C e^{x_j} \\ L &= - \sum_{i=1}^C y_i \left( x_i - \log \sum_{j=1}^C e^{x_j} \right) = - \sum_{i=1}^C y_i x_i + \log \left( \sum_{j=1}^C e^{x_j} \right) \sum_{i=1}^C y_i \quad ; \quad \sum_{i=1}^C y_i = 1 \\ L &= \log \left( \sum_{j=1}^C e^{x_j} \right) - \sum_{i=1}^C y_i x_i \end{aligned}$$

Sada možemo jednostavno izračunati derivaciju funkcije cilja s obzirom na ulazni skalar  $x_k$ :

$$\begin{aligned} \frac{\partial L}{\partial x_k} &= \frac{\partial}{\partial x_k} \log \left( \sum_{j=1}^C e^{x_j} \right) - \frac{\partial}{\partial x_k} \sum_{i=1}^C y_i x_i \\ \frac{\partial}{\partial x_k} \log \left( \sum_{j=1}^C e^{x_j} \right) &= \frac{1}{\sum_{j=1}^C e^{x_j}} \cdot e^{x_k} = s_k(\mathbf{x}) \\ \frac{\partial L}{\partial x_k} &= s_k(\mathbf{x}) - y_k \end{aligned}$$

Konačno, gradijent s obzirom na sve ulaze sloja dobivamo tako da izračunamo razliku između vektora distribucije iz modela i točne distribucije:

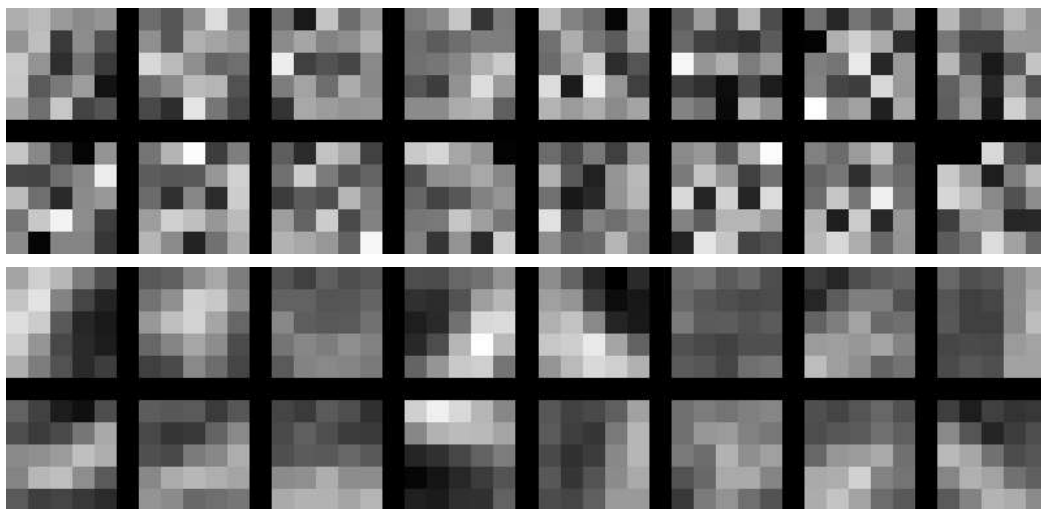
$$\frac{\partial L}{\partial \mathbf{x}} = \mathbf{s}(\mathbf{x}) - \mathbf{y}$$

Kako biste bili sigurni da ste ispravno napisali sve slojeva testirajte gradijente pozivom skripte `check_grads.py`. Zadovoljavajuća relativna greška bi trebala biti manja od  $10^{-5}$  ako vaši tenzori imaju dvostruku preciznost. Napokon, pokrenite učenje modela pozivom skripte `train.py`. Napomena: najprije postavite odgovarajuće puteve u varijable `DATA_DIR` i `SAVE_DIR` te prevedite Cython modul `im2col_cython.pyx` tako da izvršite `python3 setup_cython.py build_ext --inplace`.

Tijekom učenja možete promatrati vizualizaciju filtera koji se spremaju u `SAVE_DIR` direktorij. Budući da svaka težina odgovara jednom pikselu slike u vašem pregledniku isključite automatsko glađenje slike da biste mogli bolje vidjeti. Preporuka je da na Linuxu koristite preglednik Geeqie.

## 2. zadatak

U ovom zadatku trebate dodati podršku za L2 regularizaciju parametara. Dovršite implementaciju sloja `L2Regularizer` te naučite regularizirani model iz prethodnog zadatka koji se nalazi u `train_l2reg.py`. Proučite efekte regularizacijskog hiper-parametra tako da naučite tri različite mreže s  $\lambda = 1e^{-3}$ ,  $\lambda = 1e^{-2}$ ,  $\lambda = 1e^{-1}$  te usporedite naučene filtre u prvom sloju i dobivenu točnost.



Slučajno inicijalizirani filteri u prvom sloju na početku učenja (iznad) i naučeni filteri (ispod) s regularizacijom  $\lambda = 0.01$ .

## 3. zadatak - usporedba s Tensorflowom

U Tensorflowu definirajte i naučite model koji je ekvivalentan regulariziranom modelu iz 2. zadatka. Korisite identičnu arhitekturu i parametre učenja da biste reproducirali rezultate. Tijekom učenja vizualizirajte filtre u prvom sloju kao u prethodnoj vježbi. Kako biste u graf dodali operaciju konvolucije koristite `tf.nn.conv2d` ili

`tf.contrib.layers.convolution2d`. Prije toga proučite službenu dokumentaciju vezanu za [konvoluciju](#).

Primjer korištenja konvolucije iz `tf.contrib` paketa nalazi se ispod. Ako želite koristiti `tf.nn.conv2d` onda će vam od pomoći biti službeni [tutorial](#).

```
import tensorflow.contrib.layers as layers

def build_model(inputs, labels, num_classes):
    weight_decay = ...
    conv1sz = ...
    fc3sz = ...
    with tf.contrib.framework.arg_scope([layers.convolution2d],
        kernel_size=5, stride=1, padding='SAME', activation_fn=tf.nn.relu,
        weights_initializer=layers.variance_scaling_initializer(),
        weights_regularizer=layers.l2_regularizer(weight_decay)):

        net = layers.convolution2d(inputs, conv1sz, scope='conv1')
        # ostatak konvolucijskih i pooling slojeva
        ...

    with tf.contrib.framework.arg_scope([layers.fully_connected],
        activation_fn=tf.nn.relu,
        weights_initializer=layers.variance_scaling_initializer(),
        weights_regularizer=layers.l2_regularizer(weight_decay)):

        # sada definiramo potpuno povezane slojeve
        # ali najprije prebacimo 4D tenzor u matricu
        net = layers.flatten(inputs)
        net = layers.fully_connected(net, fc3sz, scope='fc3')

    logits = layers.fully_connected(net, num_classes, activation_fn=None, scope='logits')
    loss = ...

    return logits, loss
```

## 4. zadatak - Klasifikacija na CIFAR-10 skupu

[CIFAR-10](#) dataset sastoji se od 50000 slika za učenje i validaciju te 10000 slika za testiranje dimenzija 32x32 podijeljenih u 10 razreda. Najprije skinite dataset pripremljen za Python [ovdje](#). Iskorisite sljedeći kod kako biste učitali podatke i pripremili ih.

```
import os
import pickle
```

```
import numpy as np

def shuffle_data(data_x, data_y):
    indices = np.arange(data_x.shape[0])
    np.random.shuffle(indices)
    shuffled_data_x = np.ascontiguousarray(data_x[indices])
    shuffled_data_y = np.ascontiguousarray(data_y[indices])
    return shuffled_data_x, shuffled_data_y

def unpickle(file):
    fo = open(file, 'rb')
    dict = pickle.load(fo, encoding='latin1')
    fo.close()
    return dict

DATA_DIR = '/path/to/data/'

train_x = np.ndarray((0, img_height * img_width * num_channels), dtype=np.float32)
train_y = []
for i in range(1, 6):
    subset = unpickle(os.path.join(DATA_DIR, 'data_batch_%d' % i))
    train_x = np.vstack((train_x, subset['data']))
    train_y += subset['labels']
train_x = train_x.reshape((-1, num_channels, img_height, img_width)).transpose(3, 2, 1, 0)
train_y = np.array(train_y, dtype=np.int32)

subset = unpickle(os.path.join(DATA_DIR, 'test_batch'))
test_x = subset['data'].reshape((-1, num_channels, img_height, img_width)).transpose(3, 2, 1, 0)
test_y = np.array(subset['labels'], dtype=np.int32)

valid_size = 5000
train_x, train_y = shuffle_data(train_x, train_y)
valid_x = train_x[:valid_size, ...]
valid_y = train_y[:valid_size, ...]
train_x = train_x[valid_size:, ...]
train_y = train_y[valid_size:, ...]
data_mean = train_x.mean((0, 1, 2))
data_std = train_x.std((0, 1, 2))

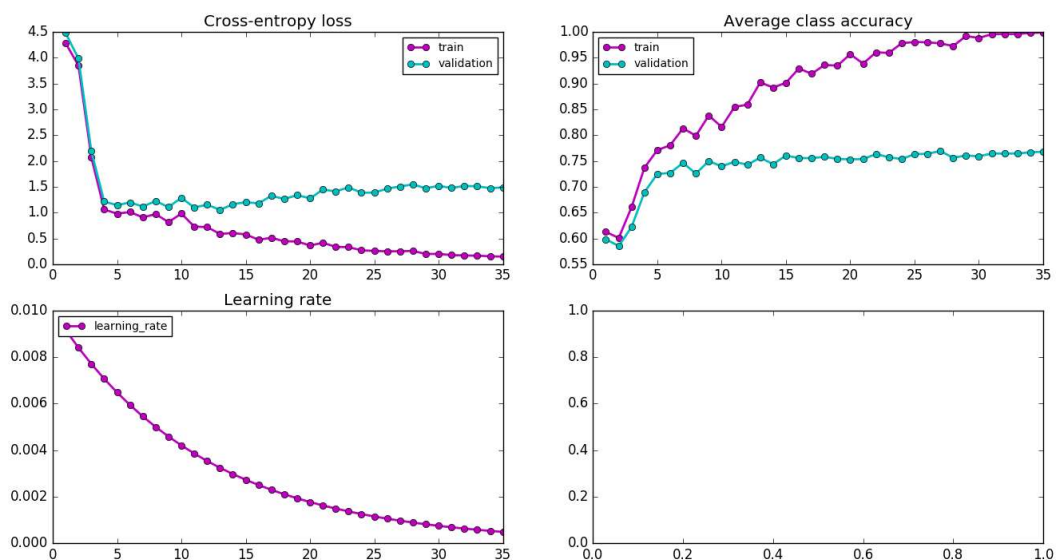
train_x = (train_x - data_mean) / data_std
valid_x = (valid_x - data_mean) / data_std
test_x = (test_x - data_mean) / data_std
```

Vaš zadatak je da u Tensorflowu naučite CNN na ovom skupu. U nastavku je prijedlog jednostavne mreže s kojom biste trebali dobiti ukupnu točnost oko 70%:

```
conv(16,5) -> relu() -> pool(3,2) -> conv(32,5) -> relu() -> pool(3,2) -> fc
```

gdje `conv(16,5)` predstavlja konvoluciju sa 16 mapa te dimenzijom filtra 5x5, a `pool(3,2)` max-pooling sloj s oknom veličine 3x3 i pomakom (*stride*) 2.

Napišite funkciju `evaluate` koja na temelju predviđenih i točnih indeksa razreda određuje pokazatelje klasifikacijske performanse: ukupnu točnost klasifikacije, matricu zabune (engl. confusion matrix) u kojoj retci odgovaraju točnim razredima a stupci predikcijama te mjere preciznosti i odziva pojedinih razreda. U implementaciji prvo izračunajte matricu zabune, a onda sve ostale pokazatelje na temelju nje. Tijekom učenja pozivajte funkciju `evaluate` nakon svake epohe na skupu za učenje i validacijskom skupu te na grafu pratite sljedeće vrijednosti: prosječnu vrijednost funkcije gubitka, stopu učenja te ukupnu točnost klasifikacije. Preporuka je da funkciji proslijedite podatke i potrebne Tensorflow operacije kako bi mogli izvesti samo unaprijedni prolazak kroz dane primjere i pritom izračunati matricu zabune. Pazite da slučajno ne pozovete i operaciju koja provodi učenje tijekom evaluacije. Na kraju funkcije možete izračunati ostale pokazatelje te ih isprintati.



Primjer kako bi trebao izgledati dobar graf tijekom učenja.

Vizualizirajte slučajno inicijalizirane i naučene filtre u prvom sloju. Da biste dohvatili varijablu u kojoj se nalaze težine prvog konvolucijskog sloja možete pozvati metodu `tf.contrib.framework.get_variables` kojoj kao argument predate *scope* pod kojim se varijabla nalazi u vašem modelu. Ispod je primjer kako to može izgledati gdje će *scope* u vašem slučaju ovisiti o tome kako ste ga nazvali tijekom definiranja grafa.

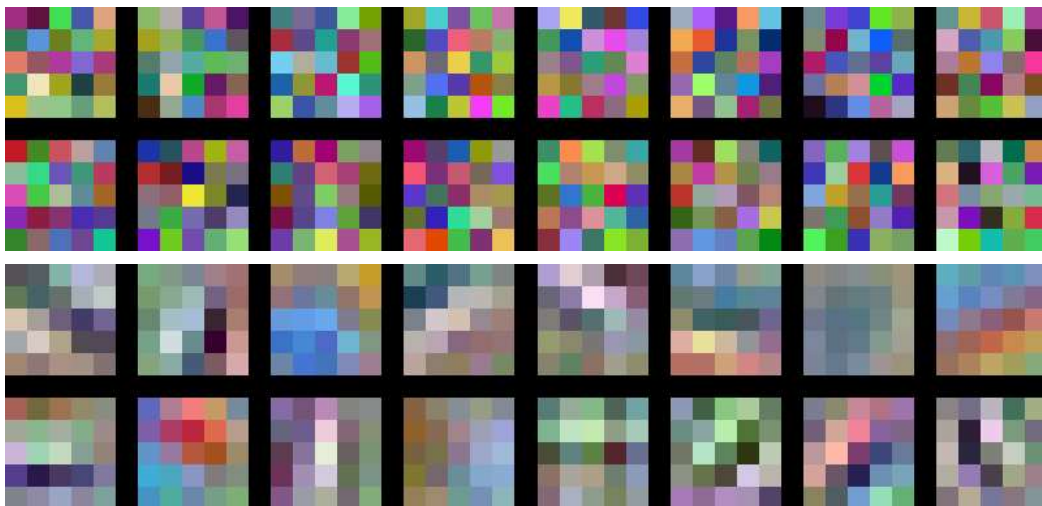
```
sess = tf.Session()
sess.run(tf.initialize_all_variables())
```



```
conv1_var = tf.contrib.framework.get_variables('model/conv1_1/weights:0')[0]
conv1_weights = conv1_var.eval(session=sess)
draw_conv_filters(0, 0, conv1_weights, SAVE_DIR)
```

U nastavku se nalazi kod koji možete koristiti za vizualizaciju:

```
def draw_conv_filters(epoch, step, weights, save_dir):
    w = weights.copy()
    num_filters = w.shape[3]
    num_channels = w.shape[2]
    k = w.shape[0]
    assert w.shape[0] == w.shape[1]
    w = w.reshape(k, k, num_channels, num_filters)
    w -= w.min()
    w /= w.max()
    border = 1
    cols = 8
    rows = math.ceil(num_filters / cols)
    width = cols * k + (cols-1) * border
    height = rows * k + (rows-1) * border
    img = np.zeros([height, width, num_channels])
    for i in range(num_filters):
        r = int(i / cols) * (k + border)
        c = int(i % cols) * (k + border)
        img[r:r+k, c:c+k, :] = w[:, :, :, i]
    filename = 'epoch_%02d_step_%06d.png' % (epoch, step)
    ski.io.imsave(os.path.join(save_dir, filename), img)
```



CIFAR-10: slučajno inicijalizirani filtri u prvom sloju na početku učenja (iznad) i naučeni filtri (ispod) s regularizacijom  $\lambda = 0.0001$ .



Prikažite 20 netočno klasificiranih slika s najvećim gubitkom te ispišite njihov točan razred i top-3 razreda za koje je mreža dala najveću vjerojatnost. Da biste prikazali sliku, morate najprije poništiti normalizaciju srednje vrijednosti i varijance:

```
import skimage as ski
import skimage.io

def draw_image(img, mean, std):
    img *= std
    img += mean
    img = img.astype(np.uint8)
    ski.io.imshow(img)
    ski.io.show()
```

Ispod se nalazi kod koji možete iskoristiti za crtanje grafova:

```
def plot_training_progress(save_dir, data):
    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(16,8))

    linewidth = 2
    legend_size = 10
    train_color = 'm'
    val_color = 'c'

    num_points = len(data['train_loss'])
    x_data = np.linspace(1, num_points, num_points)
    ax1.set_title('Cross-entropy loss')
    ax1.plot(x_data, data['train_loss'], marker='o', color=train_color,
             linewidth=linewidth, linestyle='-', label='train')
    ax1.plot(x_data, data['valid_loss'], marker='o', color=val_color,
             linewidth=linewidth, linestyle='-', label='validation')
    ax1.legend(loc='upper right', fontsize=legend_size)
    ax2.set_title('Average class accuracy')
    ax2.plot(x_data, data['train_acc'], marker='o', color=train_color,
             linewidth=linewidth, linestyle='-', label='train')
    ax2.plot(x_data, data['valid_acc'], marker='o', color=val_color,
             linewidth=linewidth, linestyle='-', label='validation')
    ax2.legend(loc='upper left', fontsize=legend_size)
    ax3.set_title('Learning rate')
    ax3.plot(x_data, data['lr'], marker='o', color=train_color,
             linewidth=linewidth, linestyle='-', label='learning_rate')
    ax3.legend(loc='upper left', fontsize=legend_size)

    save_path = os.path.join(save_dir, 'training_plot.pdf')
```

```
print('Plotting in: ', save_path)
plt.savefig(save_path)
```

```
plot_data = {}
plot_data['train_loss'] = []
plot_data['valid_loss'] = []
plot_data['train_acc'] = []
plot_data['valid_acc'] = []
plot_data['lr'] = []
for epoch_num in range(1, num_epochs + 1):
    train_x, train_y = shuffle_data(train_x, train_y)
    for step in range(num_batches):
        offset = step * batch_size
        # s ovim kodom pazite da je broj primjera djeljiv s batch_size
        batch_x = train_x[offset:(offset + batch_size), ...]
        batch_y = train_y[offset:(offset + batch_size)]
        feed_dict = {node_x: batch_x, node_y: batch_y}
        start_time = time.time()
        run_ops = [train_op, loss, logits]
        ret_val = sess.run(run_ops, feed_dict=feed_dict)
        _, loss_val, logits_val = ret_val
        duration = time.time() - start_time
        if (step+1) % 50 == 0:
            sec_per_batch = float(duration)
            format_str = 'epoch %d, step %d / %d, loss = %.2f (%.3f sec/batch)'
            print(format_str % (epoch_num, step+1, num_batches, loss_val, sec_per_

print('Train error:')
train_loss, train_acc = evaluate(logits, loss, train_x, train_y)
print('Validation error:')
valid_loss, valid_acc = evaluate(logits, loss, valid_x, valid_y)
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.eval(session=sess)]
plot_training_progress(SAVE_DIR, plot_data)
```

Ukoliko imate GPU, možda će vam biti zanimljivo pokušati dobiti bolje rezultate s moćnijom arhitekturom. U tom slučaju [ovdje](#) možete pronaći pregled članaka koji imaju najbolje rezultate na ovom skupu. Kao što vidite trenutni *state of the art* je oko 96% ukupne točnosti. Dva važna trika koje koriste najbolje arhitekture su skaliranje slika na veću rezoluciju kako bi omogućili da prvi konvolucijski slojevi uče značajke jako niske razine te proširivanje skupa

za učenje raznim modificiranjem slika (*data jittering*). Bez ovih trikova je jako teško preći preko 90% ukupne točnosti.


## Bonus zadatak - Multiclass hinge loss

Pokušajte u zadnjem zadatku unakrsnu entropiju zamijeniti s multiclass hinge lossom te usporedite rezultate. Objašnjenje multiclass hinge lossa možete pronaći [ovdje](#).

Proučite u Tensorflow dokumentaciji osnovne operacije nad tenzorima kako biste pronašli najlakši način da to ostvarite. Pomoć: jedna opcija kako to možete izvesti je da razdvojite logite (izlazi iz zadnjeg potpuno povezanog sloja) na matricu logita netočnih razreda i vektor logita na mjestima točnih razreda. To možete izvesti pomoću operacija `tf.dynamic_partition` i `tf.one_hot`. Zatim unutar `tf.maximum` računate razliku između matrice logita na netočnim razredima i vektora logita na točnim razredima. To možete napisati kao običnu razliku jer za tenzore različitih dimenzija Tensorflow po defaultu napravi *broadcasting* ako je to moguće.

## Dodatni materijali

- [Deep learning book](#)
- [CS231n Convolutional Neural Networks for Visual Recognition](#)

 [dlunizg](#)  
[ivan.kreso@fer.hr](mailto:ivan.kreso@fer.hr)