Predmet: "Vještačka inteligencija"
Laboratorijska vježba 6: Neuronske mreže (4/4)
Odgovorna nastavnica: Vanr. prof. dr Amila Akagić

Sadržaj vježbe:

1 Cilj vježbe

Cilj treće vježbe iz oblasti neuronskih mreža jeste upoznavanje sa slikom kao ulaznim podatkom u neuronsku mrežu, kao i sa arhitekturama koje se koriste za rad sa slikom. Poseban akcenat se stavlja na konvolucione neuronske mreže. Nakon uspješno urađene vježbe, studenti su u stanju da identificiraju osnovne elemente arhitekture konvolucione neuronske mreže, te da primijene stečeno znanje na jednostavne zadatke iz oblasti računarske vizije.

2 Konvolucione neuronske mreže

Nakon što smo se, kroz prethodne dvije laboratorijske vježbe, upoznali sa osnovama rada u Keras-u, kao i sa osnovnim tehnikama za kreiranje sistema za klasifikaciju i regresiju na bazi tekstualnih podataka, na ovoj vježbi ćemo se upoznati sa korištenjem neuronskih mreža za rad sa slikom. Kroz vježbu ćemo se upoznati sa tipom dubokih neuronskih mreža koji se gotovo pa univerzalno koristi u oblasti računarske vizije - konvolucione neuronske mreže (eng. *Convolutional Neural Networks* - CNN ili *convnets*).

2.1 Uvod u CNN

Na laboratorijskoj vježbi 2 se kao uvodni primjer u Keras razvojni okvir vršila klasifikacija rukom pisanih cifara na MNIST skupu podataka. U tu svrhu, korištena je jednostavna neuronske mreža sa dva potpuno povezana (*dense*) sloja. Pri tome, postignuta je odlična tačnost nad testnim skupom podataka - oko 97%. Međutim, za ovakav zadatak (koji je klasični zadatak računarske vizije), se koristi drugi tip neuronske mreža - konvolucione neuronske mreže. Prije nego što teoretski objasnimo CNN-ove i razlog njihove uspješnosti i popularnosti, pogledajmo praktični primjer kreiranja jednostavnog CNN modela za isti zadatak koji smo već riješili na vježbi 2.

Kod koji slijedi pokazuje kako izgleda jedan CNN model - to je u suštini lanac `Conv2D` i `MaxPooling2D` slojeva. Tačne uloge ovih slojeva će biti objašnjene u nastakvu vježbe.

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
```

```
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Bitno je naglasiti da je ulaz u CNN uvijek tenzor oblika `(visina_slike, širina_slike, broj_kanala`[1]`)`. U isječku koda iznad, ulaz je oblika `(28, 28, 1)`, jer se u MNIST skupu podataka radi o slikama u nijansama sive boje (eng. *grayscale*), dimenzija 28×28. Ukoliko se arhitektura ove mreže ispiše pomoću naredbe `model.summary()`, dobija se izlaz kao što je prikazano na slici 1.

```
[2]: model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

Slika 1: Prikaz arhitekture mreže.

Treba primijetiti da je izlaz iz svakog `Conv2D` i `MaxPooling2D` sloja 3D tenzor oblika `(visina, širina, filteri)`. Visina i širina se smanjuju kako se ulazi dublje u mrežu. Broj filtera se kontroliše pomoću prvog argumenta koji se pošalje `Conv2D` slojevima (za prvi sloj je do 32, a za druga dva 64).

Sljedeći korak jeste proslijediti trenutni izlazni tenzor (oblika `(3, 3, 64)`) na poptuno povezanu klasifikacijsku mrežu kao one sa kojima smo se upoznali ranije (lanac potpuno povezanih - `Dense` - slojeva). Pošto ti slojevi procesiraju vektore, koji su jednodimenzionalni, a naš trenutni izlaz iz mreže je trodimenzionalan, potrebno je prvo 3D podatke pretvoriti u 1D podatke. To se radi pomoću `Flatten` sloja, koji uzima podatke, i iste poravna na način da sada predstavljaju 1D vektor, uklanjajući sve dimenzije osim jedne.

U model neuronske mreže je, dakle, potrebno dodati sljedeće slojeve:

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Posljednji sloj ima 10 neurona, obzirom da postoji 10 mogućih klasa (po jedna za svaku cifru). Pošto se radi o tipičnom problemu klasifikacije, aktivacijska funkcija je `softmax`. Ukoliko se sada pogleda arhitektura mreže pomoću naredbe `summary`, dobija se izlaz prikazan na slici 2.

[1] Broj kanala se odnosi na kanale boja slike, npr. 1 za *grayscale slike*, 3 za RGB slike, itd.

```
[4]: model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

Slika 2: Arhitektura mreže nakon dodavanja `Flatten` i `Dense` slojeva.

Može se primijetiti da je izlaz oblika `(3, 3, 64)` pretvoren u oblik `(576,)` pomoću `Flatten` sloja, te se sada može proslijediti potpuno povezanim slojevima.

Ovim je osnovna arhitektura konvolucione neuronske mreže završena. U najjednostavnijem obliku, CNN se sastoji od tri dijela:

1. Lanac konvolucionih blokova (parova `Conv2D` i `MaxPooling2D` slojeva);
2. `Flatten` sloj;
3. Lanac izlaznih `Dense` slojeva.

Sada je mreža spremna za treniranje i evaluaciju. Kod koji se koristi za to je u osnovi isti kao kod iz laboratorijske vježbe 2:

```
from keras.datasets import mnist
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_images,
          train_labels,
          epochs=5,
          batch_size=64)

```

```
test_loss, test_acc = model.evaluate(test_images, test_labels)

print(test_acc)
```

Nakon treniranja, tačnost nad testnim skupom podataka će biti čak 99.3%! Ovo znači da se relativna pojava greške u odnosu na običnu neuronsku mrežu iz vježbe 2 smanjila za čak 68%. U nastavku ćemo se upoznati sa operacijama konvolucije i max-pooling, to jeste sa osnovnim slojevima koji čine CNN.

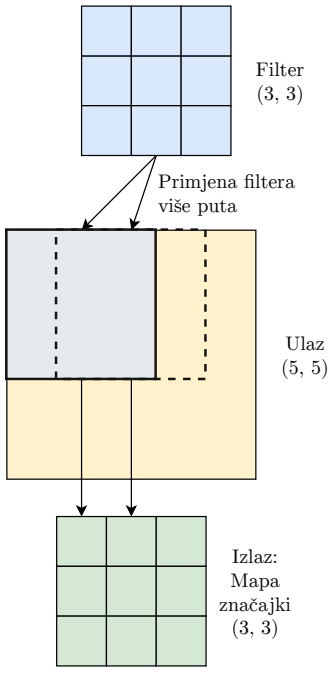
2.2 Konvolucioni sloj

Konvolucija predstavlja jednostavnu primjenu filtera na neki ulaz, što rezultira u nekoj aktivaciji - aktivacija ulaz u neuron pretvara u izlaz koji se šalje na naredni sloj. Ukoliko se primjena ponovi više puta, rezultat koji se dobija jeste mapa aktivacija, ili mapa značajki, koja ukazuje na mjesta i intenzitet prepoznate značajke na ulazu kao što je slika. U kontekstu dubokog učenja, značajka na slici je širok pojam, te može biti bilo šta od jednostavne ivice, pa sve do očiju životinje (npr. oči mačke ukoliko se radi na sistemu za prepoznavanje mačaka).

2.2.1 Konvolucioni filter

U kontekstu CNN, konvolucija je linearna operacija koja uključuje množenje skupa težina sa ulazom. Kako je tehnika napravljena za dvodimenzionalne ulaze kao što je slika, množenje se vrši između niza ulaznih podataka, i dvodimenzionalnog niza težina, koji se još naziva filter ili kernel. Filter je uvijek manji od ulaznih podataka (slike), a množenje koje se vrši između filtera i ulaza je skalarno množenje. Skalarno množenje je potrebno iz razloga što se kao reultat ove operacije mora dobiti jedna vrijednost (skalar) a ne niz ili matrica novih vrijednosti.

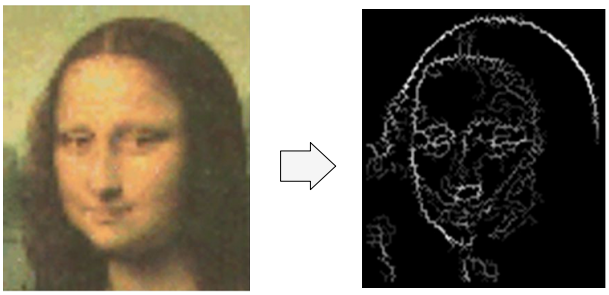
Korištenje filtera koji je manji od ulaznih podataka omogućava primjenu jednog filtera nad istim ulaznim podacima više puta, to jeste na različitim mjestima slike. Filter se, dakle, sistematski primijenjuje na svaki dio slike koji je veličine filtera, s lijeva na desno, počevši od gornjeg lijevog ugla. Sistematska primjena filtera nad čitavom slikom je upravo ono što konvolucionom sloju daje na značaju. Ako je filter napravljen da otkrije specifičnu značajku na slici, onda sistematska primjena tog filtera nad čitavom slikom stvara mogućnost detekcije te značajke na bilo kojem mjestu u slici[2]. Primjena filtera i operacija konvolucije su prikazani na slici 3.



Slika 3: Ilustracija operacije konvolucije.

Izlazni dvodimenzionalni niz koji se dobija primjenom filtera naziva se mapa značajki. Primjer jedne mape značajki je dat na slici 4. Nakon što se mapa značajki kreira, ona se može proslijediti dalje kroz mrežu, na primjer kroz slojeve sa ReLU aktivacijskom funkcijom, slično kao kod običnih neuronskih mreža.

[2]Ova sposobnost se u literaturi često naziva *translation invariance*.



Slika 4: Ulaz i mapa značajki nakon primjene konvolucije. U ovom slučaju, filter je služio za detekciju ivica na slici.

Ono što se može primijetiti sa primjera iz slike 3 jeste da rezultat konvolucije nema nužno iste dimenzije kao ulazna slika. Za to postoje dva osnovna razloga:

- Granični efekti, koji se mogu riješiti upotrebom *padding*-a;
- Primjena pomaka kod konvolucije.

2.3 Max-pooling sloj

U primjeru sa početka vježbe treba primijetiti da se veličina mapa značajki polovi nakon svakog `MaxPooling2D` sloja. Na primjer, prije prvog `MaxPooling2D` sloja, mapa značajki je dimenzija 26×26, ali ju max-pooling operacija polovi na 13×13. Upravo to je uloga max pooling: smanjivanje dimenzija mapa značajki, slično kao konvolucije sa pomakom.

Max pooling se sastoji od ekstrakcije dijelova iz ulazne mape značajki, te prosljeđivanja maksimalne vrijednosti unutar tog dijela na izlaz (ukoliko se radi sa slikama koje imaju više kanala, npr. RGB, onda se operacija radi posebno za svaki kanal).

Ova operacija se obično radi sa prozorom veličine 2×2, i pomakom od 2, dok se konvolucija radi sa 3×3 kernelom, i pomakom od 1 (tj. bez pomaka).

Alternativa max pooling operaciji je average pooling. Kod ove operacije, umjesto maksimuma iz dijela slike koji se posmatra, na izlaz se proslijedi prosječna vrijednost tog dijela slike. Međutim, u praksi se max pooling pokazao boljim, kao i lakšim za procesiranje na računaru. Razlog iza toga leži u činjenici da je više informacija sadržano posmatranjem maksimalne prisutnosti neke značajke u određenom dijelu slike, nego prosječne prisutnosti.

2.4 CNN u Keras-u

Keras sadrži ugrađene mehanizme za jednostavno kreiranje konvolucionih i max pooling slojeva. U nastavku će ovi slojevi biti objašnjeni, kao i njihovi parametri.

2.4.1 Keras `Conv2D` sloj

Keras `Conv2D` sloj ima mnogo parametara koje je moguće postaviti. Konstruktor ove klase je dat u isječku koda kako slijedi:

```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1),
  padding='valid', data_format=None, dilation_rate=(1, 1),
  activation=None, use_bias=True, kernel_initializer='glorot_uniform',
  bias_initializer='zeros', kernel_regularizer=None,
  bias_regularizer=None, activity_regularizer=None,
  kernel_constraint=None, bias_constraint=None)
```

Najbitniji parametri su objašnjeni u nastakvu:

1. `filters` - broj filtera iz kojih će konvolucijski sloj učiti. Također određuje broj izlaznih filtera konvolucije. Ovo je obavezan parametar;
2. `kernel_size` - dimenzije kernela. Obavezan parametar. Mora biti neparan broj;

3. `strides` - pomak. Može biti cijeli broj ili par od dva cijela broja. Podrazumijevana vrijednost je (1, 1);
4. `padding` - *padding*. Vrijednost 'valid' označava da nema *padding*-a, dok vrijednost 'same' označava da treba primijeniti *padding*. Podrazumijevana vrijednost je bez *padding*-a.
5. `data_format` - parametar koji je potreban radi kompatibilnosti sa TensorFlow. Obično se ostavi podrazumijevana vrijednost;
6. `dilation_rate` - označava dilaciju konvolucije[3]. Obično se ostavlja na podrazumijevanoj vrijednosti 1.
7. `activation` - aktivacijska funkcija. Ukoliko se ne navede, nema aktivacijske funkcije;

Ostali parametri izlaze van okvira ove laboratorijske vježbe.

2.4.2 Keras `MaxPooling2D` sloj

Operacija max pooling je podržana u Keras razvojnom okviru kroz `MaxPooling2D` sloj. Konstruktor ove klase je dat u sljedećem isječku koda:

```
tf.keras.layers.MaxPooling2D(
    pool_size=(2, 2), strides=None, padding="valid", data_format=None
)
```

Značenje parametar je identično kao kod `Conv2D` sloja. Novi parametar, `pool_size`, označava veličinu prozora za max pooling operaciju (analogno veličini kernela kod konvolucije).

[3]Konvolucija sa dilacijom je takva konvolucija kod koje postoje praznine u filteru.

3 Zadaci za rad u laboratoriji

Predviđeno je da se svi zadaci u nastavku rade u sklopu Jupyter Notebook okruženja. Svaki podzadatak treba biti zasebna Jupyter ćelija.

Zadatak 1 - Korištenje CNN za klasifikaciju

U ovom zadatku ćemo koristiti CNN kako bismo riješili problem klasifikacije slika. Skup podataka koji ćemo koristiti je CIFAR-10. Ovaj skup predstavlja jedan od standarnih skupova podataka koji se koriste u računarskoj viziji i dubokom učenju. Iako se klasifikacijski zadatak na ovom skupu podataka smatra 'riješenim', može se koristiti kao osnova za učenje i vježbanje razvoja, evaluacije, i korištenja konvolucionih neuronskih mreža za zadatak klasifikacije slika.

CIFAR-10 je skraćenica od *Canadian Institute For Advanced Research*. Sadrži 60 000 RGB slika dimenzija 32×32 piksela. Slike su različitih objekata, i pripadaju 10 klasa, i to: *airplane (0), automobile (1), bird (2), cat (3), deer (4), dog (5), frog (6), horse (7), ship (8), truck (9)*. Ovaj skup podataka je dosta dobro istražen, te se često koristi za testiranje algoritama računarske vizije iz oblasti mašinskog učenja. Relativno je jednostavno postići tačnost od oko 80% prilikom klasifikacije. Bolje rezultate od toga je moguće postići sa CNN. Ovaj skup podataka je dostupan uz Keras, te ga je moguće veoma jednostavno učitati:

```
from keras.datasets import cifar10
# ucitaj skup podataka
(trainX, trainy), (testX, testy) = cifar10.load_data()
```

a) Učitati CIFAR-10 skup podataka, i ispisati oblik trening i test podataka i labela;

b) Pomoću Matplotlib (tj. pyplot) iscrtati prvih 9 slika ovog skupa podataka. Šta možete reći o slikama? Da li ova rezolucija odgovara rezoluciji modernih fotografija? Kako bi rezolucija slike mogla djelovati na rad algoritama i modela mašinskog učenja prilikom klasifikacije?

Nakon što su podaci učitani, potrebno je iste predprocesirati. U tu svrhu, potrebno je labele pretvoriti u nizove od 10 elemenata (to jeste izvršiti *one-hot* enkodiranje), a same fotografije (to jeste podatke) je potrebno pretvoriti u tip `float32` te skalirati na opsegu od 0 do 1. Obzirom da znamo da je maksimalna vrijednost nekog kanala 255, skaliranje možemo postići jednostavnim dijeljenjem sa 255.

c) Pripremiti podatke za treniranje. Za labele, potrebno je koristiti funkciju `to_categorical`, dok je za trening podatke potrebno koristiti `astype` za pretvorbu u *floating-point* tip, te dijeljenje sa 255 za korak skaliranja podataka.

Naredni korak je definisanje modela. Obično se kreira pomoćna funkcija koja kao rezultat vraća gotov model. To se radi iz razloga što je često potrebno mijenjati arhitekturu modela, te eksperimentisati sa različitim slojevima. Funkcija za definisanje modela obično izgleda ovako:

```
# pomocna funkcija za definisanje modela
def define_model():
  model = Sequential()
  # ... slojevi ...
  return model
```

d) Definisati pomoćnu funkciju za kreiranje modela. Za sada, funkcija će praviti jednostavan CNN (ovo se još naziva i *baseline* model). Model treba imati sljedeće slojeve:

- `Conv2D` sloj sa 32 filtera, dimenzija 3×3, sa uključenim *padding-om*;
- `Conv2D` sloj sa 32 filtera, dimenzija 3×3, sa uključenim *padding-om*;
- `MaxPooling2D` sloj dimenzija 2×2;
- `Conv2D` sloj sa 64 filtera, dimenzija 3×3, sa uključenim *padding-om*;
- `Conv2D` sloj sa 64 filtera, dimenzija 3×3, sa uključenim *padding-om*;
- `MaxPooling2D` sloj dimenzija 2×2;

- `Flatten` sloj;
- `Dense` sloj sa 128 neurona, koji koristi `relu` aktivacijsku funkciju;
- Izlazni `Dense` sloj sa 10 neurona i `softmax` aktivacijskom funkcijom.

e) Dodati potrebni kod za kompajliranje modela u funkciju `define_model`. Optimizator treba biti *gradient descent* uz stopu učenja od 0.001 i momentum od 0.9, to jeste:

```
opt = SGD(lr=0.001, momentum=0.9)
```

Kompajliranje se treba vršiti na način da se za funkciju gubitka koristi `categorical_crossentropy`, a za metrike `accuracy`. Liniju za kompajliranje modela dodati na kraj pomoćne funkcije. Optimizator treba biti SGD definisan u prethodnom podzadatku.

Sada je potrebno istrenirati model. Parametri koji će biti korišteni za treniranje su sljedeći:

- `epochs=15`;
- `batch_size=64`;
- `validation_data=(testX, testY)` (obično bi se kreirao poseban skup za validaciju, ali u ovom primjeru ćemo koristiti testni skup);

f) Izvršiti treniranje i evaluaciju modela. Kolika je postignuta tačnost? Da li je ovo zadovoljavajuće?

g) Izvršiti plot ponašanja tačnost i i gubitka kroz epohe. Šta možete zaključiti iz plotova?

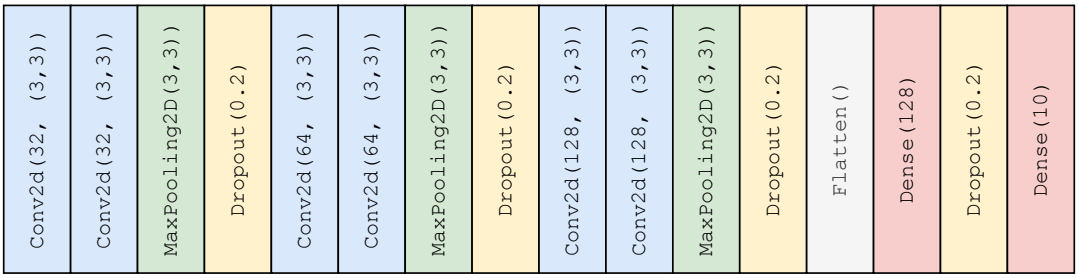
Nakon što smo napravili *baseline* model, možemo isti optimizirati. U prethodnom modelu nalazi se potpuno povezani sloj od 128 neurona. Ovaj sloj zapravo uzima sve značajke i koristi ih za treniranje. Međutim, kako bi se spriječio *overfitting*, moguće je nasumično onemogućiti neke neurone tokom treniranja. Kako bi se to postiglo, koristi se `Dropout` sloj. Ovo pomaže robusnosti mreže, te spriječava *overfitting*, na način da tokom treniranja neuroni neće učiti uvijek, već imaju šansu da budu odbačeni (od čega i naziv sloja). Na taj način, može se spriječiti da model trening značajke nauči previše dobro.

h) Dodati `Dropout` sloj u model, sa vjerovatnoćom od 0.5, prije prvog `Dense` sloja. Ponoviti treniranje, evaluaciju, i plotanje gubitka i tačnosti. Da li su postignuti rezultati bolji? Da li sada nastaje *overfitting*?

i) Sada ćemo napraviti konačnu verziju našeg modela. Nad postojećim modelom, izvršiti sljedeće izmjene:

1) Prije `Flatten` sloja, dodati još jedan blok od dva `Conv2D` sloja i jednog `MaxPooling2D` sloja, po uzoru na već postojeće blokove. Postaviti broj filtera za dva `Conv2D` sloja na 128;
2) Postojeći `Dropout` promijeniti na 0.2, te ga pomjeriti nakon prvog `Dense` sloja;
3) Nakon svakog `MaxPooling2d` sloja, dodati `Dropout` sloj sa vjerovatnoćom 0.2.

Konačni model bi trebao izledati kao na slici 5.



Slika 5: Arhitektura finalnog modela za CIFAR-10.

Ovakav model je potrebno trenirati 30 epoha. Koja je dobijena tačnost? Da li dolazi do *overfitting*-a?

Na kreiranom modelu je moguće još eksperimentisati (npr. promjenom vrijednosti i pozicije `Dropout` slojeva), te je moguće primijeniti još tehnika regularizacije, kao što *weight decay*, *batch normalization*, itd. Međutim, često nije potrebno praviti vlastiti model od nule, već se mogu koristiti gotovi, unaprijed trenirani modeli, koje je moguće prilagoditi vlastitim potrebama. Ovakav pristup naziva se *transfer learning*.