Keras API. Izgradnje duboke neuronske mreže za klasifikaciju prometnih znakova.

Sadržaj

- Keras API za duboko učenje
  - metode učitavanje podataka s diska
  - izgradnja modela s funkcijskim API
  - augmentacija skupa podataka
  - callbacks
  - transfer learning
- Izgradnja mreže za klasifikaciju prometnih znakova
  - priprema skupova podataka za GTSRB dataset
  - izgradnja učenje i evaluacija konvolucijske neuronske mreže
  - inferencija
  - implementacija callbacka
  - augmentacija skupa za učenje
  - korištenje dostupnih popularnih dubokih neuronskih mreža
  - DFGTSD skup podataka

Keras API za duboko učenje

Metode učitavanja podataka s diska

- Keras omogućuje učitavanje podataka na tri načina:
  - numpy polja → ovo smo već vidjeli (MNIST)
  - TensorFlow Dataset objekti
  - Python generatori
- moderni datasetovi su često vrlo veliki (nekoliko GB) te ih je nepraktično cijele učitavati u numpy polja (ili jednostavno ne stanu)
- ako se model trenira na GPU – treba koristiti Dataset objekt jer on automatski asinkrono pretprocesira podatke na CPU dok je GPU zauzet izračunom te bufferira podatke

Metode učitavanja podataka s diska

- ako imamo na raspolaganje slike na disku strukturirane na način (svaka klasa u zasebnom poddirektoriju):

```
training_data/
...class_a/
......a_image_1.jpg
......a_image_2.jpg
...class_b/
......b_image_1.jpg
......b_image_2.jpg
etc.
```

- onda je moguće koristiti metodu `image_dataset_from_directory` za kreiranje TensorFlow dataset objekta

Metode učitavanja podataka s diska

Neki argumenti

- **directory**: direktorij gdje su podaci smješteni
- **labels**: ako je 'infered' onda se labele generiraju iz strukture direktorija, None (nema labela) ili lista/tuple integer labela koja je iste veličine kao broj slika u direktoriju
- **label_mode**: 'int' znači da su labele kodirane kao integeri (npr. za sparse_categorical_crossentropy loss; ‘categorical’ znači da su labele kodirane kao kategorički vektori; 'binary’ znači da su labele kodirane s float32 s vrijednostima 0 ili 1
- **class_names**: ako su "labels" "inferred„ onda se koristi. Ovo je lista imena klasa (mora odgovarati imenima poddirektorija i zapravo služi za kontrolu redoslijeda klasa (inače se koristi alfanumerički redoslijed)
- **batch_size**: veličina bacha podataka
- **image_size**: veličina na koju će se skalirati slike nakon učitavanja s diska
- **shuffle**: trebali li nasumično rasporediti podatke; uključeno po defaultu
- **validation_split**: float između 0 i 1 i definira udio podataka koji će se rezervirati za validaciju

Metode učitavanja podataka s diska

```
from tensorflow import keras
from tensorflow.keras.preprocessing.image import image_dataset_from_directory

train_ds = image_dataset_from_directory(
    directory='training_data/',
    labels='inferred',
    label_mode='categorical',
    batch_size=32,
    image_size=(256, 256))
validation_ds = image_dataset_from_directory(
    directory='validation_data/',
    labels='inferred',
    label_mode='categorical',
    batch_size=32,
    image_size=(256, 256))

model = keras.applications.Xception(weights=None, input_shape=(256, 256, 3), classes=10)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
model.fit(train_ds, epochs=10, validation_data=validation_ds)
```

- Dataset objekti koji vraćaju batch slika (slike + labele kodirane kao integeri)
- labela odgovara rednom broju foldera (sortiranom u alfanumeričkom redoslijedu)
- moguće dati i vlastite nazive klasama pomoću `class_names` liste

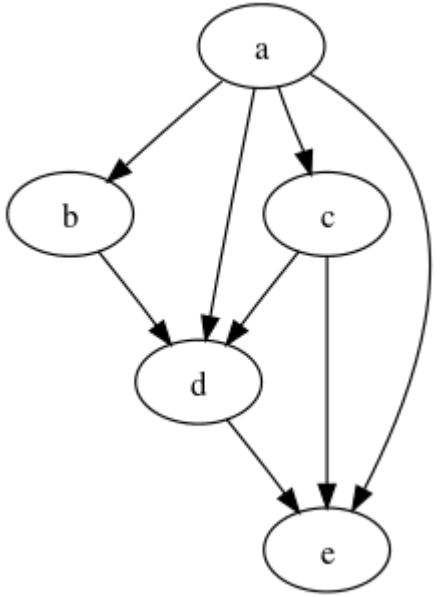
Metode učitavanja podataka s diska

- kada je na raspolaganju TensorFlow Dataset moguće ga je uzorkovati
- primjer:

```
# Create a dataset.
dataset = keras.preprocessing.image_dataset_from_directory(
  'path/to/main_directory', batch_size=64, image_size=(200, 200))

# For demonstration, iterate over the batches yielded by the dataset.
for data, labels in dataset:
   print(data.shape)  # (64, 200, 200, 3)
   print(data.dtype)  # float32
   print(labels.shape)  # (64,)
   print(labels.dtype)  # int32
```

Izgradnja modela s funkcijskim API

- funkcijski API omogućava izgradnju složenijih modela nego što omogućava Sequential model (dijeljeni slojevi, više ulaza i izlaza…)
- glavna ideja je predstavljanje mreže u smislu directed acyclic graph (DAG) koji se sastoji od slojeva
- praktički model je mreža s dodatnim rutinama za treniranje i evaluaciju
- funkcijski API upravo je način kako izgraditi takvu strukturu na jednostavan način
- kreće se od ulaza, dodavaju se slojevi dok se ne dobije izlaz



Izgradnja modela s funkcijskim API

- **Primjer**: konvolucijska mreža
- RGB slike mogu biti proizvoljne veličine
- slike se cropaju na 150x150 piksela
- vrijednosti piksela skaliraju se na raspon [0,1]
- niz konvolucijskih slojeva + maxpooling
- na izlazu 10 neurona

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None, None, 3)]   0
_________________________________________________________________
center_crop_1 (CenterCrop)   (None, 150, 150, 3)       0
_________________________________________________________________
rescaling_1 (Rescaling)      (None, 150, 150, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 47, 47, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 32)        9248
_________________________________________________________________
global_average_pooling2d (Gl (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 10)                330
=================================================================
Total params: 19,722
Trainable params: 19,722
Non-trainable params: 0
_________________________________________________________________
```

Izgradnja modela s funkcijskim API

- slike mogu biti proizvoljne veličine

```
# Let's say we expect our inputs to be RGB images of arbitrary size
inputs = keras.Input(shape=(None, None, 3))
```

- za potrebe cropanja i skaliranja potrebno je importati metode za preprocessing:

```
from tensorflow.keras.layers.experimental.preprocessing import CenterCrop
from tensorflow.keras.layers.experimental.preprocessing import Rescaling
```

Izgradnja modela s funkcijskim API

- definiranje transformacija od ulaza ka izlazu
- što je novo?
  - korištenje preprocessing slojeva
  - GlobalAveragePooling2D
- kreiranje modela:

```
model = keras.Model(inputs=inputs, outputs=outputs)
```

```
from tensorflow.keras import layers

# Center-crop images to 150x150
x = CenterCrop(height=150, width=150)(inputs)
# Rescale images to [0, 1]
x = Rescaling(scale=1.0 / 255)(x)

# Apply some convolution and pooling layers
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(3, 3))(x)
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(3, 3))(x)
x = layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu")(x)

# Apply global average pooling to get flat feature vectors
x = layers.GlobalAveragePooling2D()(x)

# Add a dense classifier on top
num_classes = 10
outputs = layers.Dense(num_classes, activation="softmax")(x)
```

Izgradnja modela s funkcijskim API

- model se ponaša kao jedan veliki sloj
- npr. mogu mu se predati podaci (batch) i očitati izlaz

```
data = np.random.randint(0, 256, size=(64, 200, 200, 3)).astype("float32")
processed_data = model(data)
print(processed_data.shape)
```

- što će se ispisati na ekranu?

- trening, evaluacija i pohrana modela izgrađenog funkcijskim API je jednaka kao za `Sequential` modele (koristiti metode `.fit()` i `.evaluate()`, `.save()`)

Augmentacija skupa podataka

- augmentacija (umjetno) povećavanje skupa za učenje je česta tehnika kojom se mogu dobiti robusniji modeli i modeli koji bolje generaliziraju
- Keras pruža ImageDataGenerator klasu koja omogućava vrlo laku implementaciju augmentaciju u real-time:
  - rotacija
  - pomak
  - flipanje
  - promjena osvjetljenja
  - zoomiranje
- objekt generira batcheve slika

Augmentacija skupa podataka

- Neki argumenti:

rotation_range: Int. Degree range for random rotations.

brightness_range: Tuple or list of two floats. Range for picking a brightness shift value from.

zoom_range: Float or [lower, upper]. Range for random zoom. If a float, [lower, upper] = [1-zoom_range, 1+zoom_range].

horizontal_flip: Boolean. Randomly flip inputs horizontally.

vertical_flip: Boolean. Randomly flip inputs vertically.

rescale: rescaling factor. Defaults to None. If None or 0, no rescaling is applied, otherwise we multiply the data by the value provided (after applying all other transformations).

preprocessing_function: function that will be applied on each input. The function will run after the image is resized and augmented. The function should take one argument: one image (Numpy tensor with rank 3), and should output a Numpy tensor with the same shape.

`ImageDataGenerator` class

```
tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    zca_epsilon=1e-06,
    rotation_range=0,
    width_shift_range=0.0,
    height_shift_range=0.0,
    brightness_range=None,
    shear_range=0.0,
    zoom_range=0.0,
    channel_shift_range=0.0,
    fill_mode="nearest",
    cval=0.0,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=None,
    preprocessing_function=None,
    data_format=None,
    validation_split=0.0,
    dtype=None,
)
```

Augmentacija skupa podataka

- primjer upotrebe
- primijetite kako train slike augmentiramo dok za testne slike samo skaliramo vrijednosti piksela na range [0,1]
- ulazne slike se učitavaju s diska pomoću metode `.flow_from_directory`

```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
        'data/train',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')
validation_generator = test_datagen.flow_from_directory(
        'data/validation',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')
model.fit(
        train_generator,
        steps_per_epoch=2000,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=800)
```

Callbacks

- callback je općenito nekakav izvršni kod koji se prosljeđuje kao argument drugom kodu i od drugog koda se očekuje da pozove dani kod u određenom trenutku
- keras callback je objekt koji se aktivira tijekom učenja (npr. na početku ili na kraju epohe, nakon jednog batcha...)
- postoje gotovi callbacks, ali se mogu kreirati i vlastiti
- callbacks se predaju metodi `.fit()` u obliku liste koja sadrži sve callbacke koje želimo primijeniti tijekom procesa učenja
- mogu se koristiti za različite korisne radnje: periodičko spremanje modela na disk, rano zaustavljanje, pisanje u TensorBoard logove nakon svakog batcha kako bi mogli pratiti tijek učenja pomoću metrika, podešavanje stope učenja...

Checkpointing models

- učenje duboke neuronske mreže često traje satima pa čak i danima
- duboke mreže mogu lako overfitati skup za učenje
- to bi značilo da moramo konstantno pratiti tijek učenja
- model checkpointing predstavlja callback koji sprema model na disk pod određenim uvjetima
- na taj način možemo koristiti odabrati od modela iz prethodnih epoha učenja; nastaviti učenje od određenog spremljenog modela i sl.
- također uvijek se može dogoditi nekakva pogreška (npr. restart računala i sl.) pa je spremanje modela poželjno

Checkpointing models

- Argumenti:
  - **filepath**: putanja gdje se spremaju modeli
  - **save_best_only**: spremanje samo najboljeg modela do sada ili spremanje modela nakon svake epohe
  - **monitor**: koja metrika se prati (npr. val_loss)
  - **mode**: definicija najboljeg modela (minimum ili maksimum)
  - **save_freq**: na kraju svake epohe ili nakon određenog broja batcheva

`ModelCheckpoint` class

```
tf.keras.callbacks.ModelCheckpoint(
    filepath,
    monitor="val_loss",
    verbose=0,
    save_best_only=False,
    save_weights_only=False,
    mode="auto",
    save_freq="epoch",
    options=None,
    **kwargs
)
```

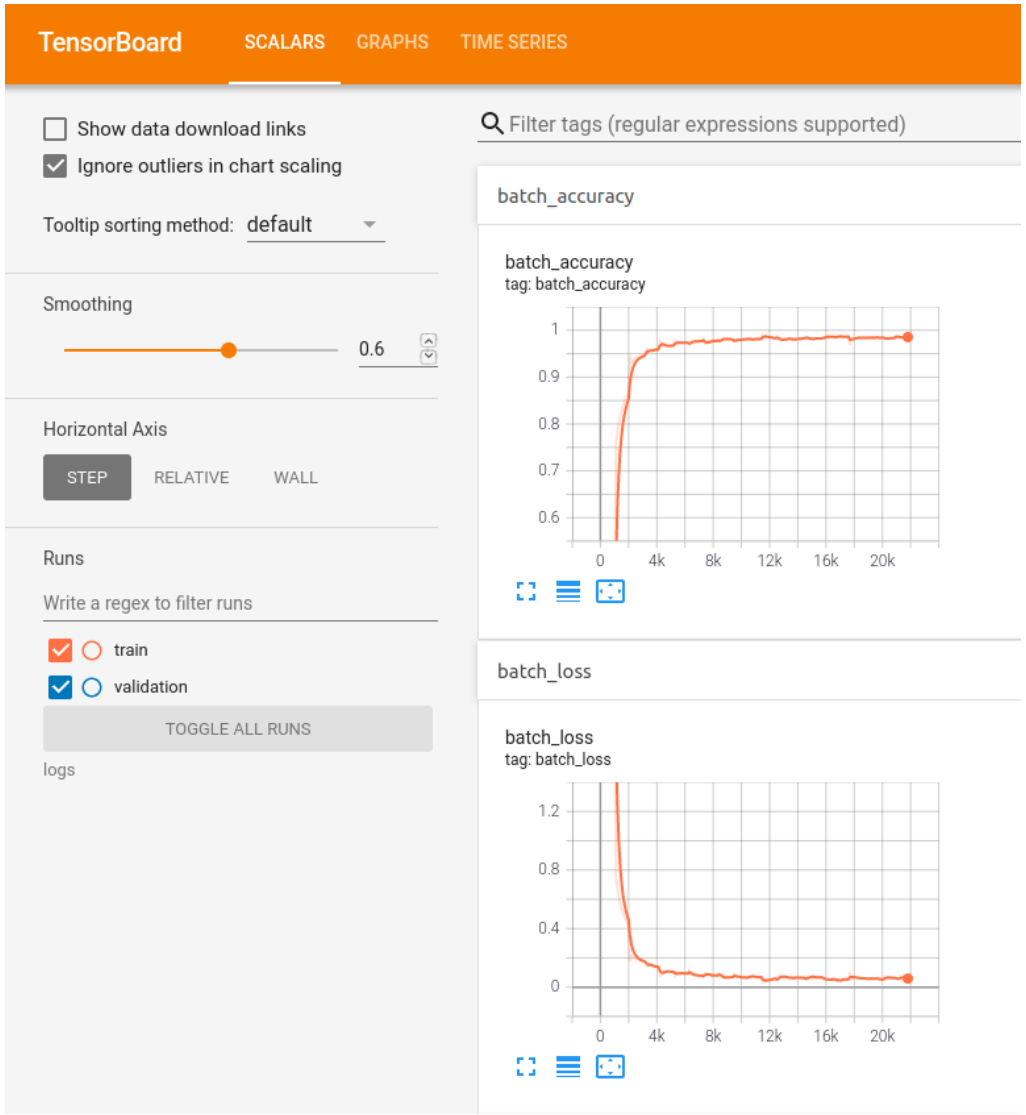
Checkpointing models

- primjer: spremanja najboljeg modela prema točnosti klasifikacije na validacijskom skupu; spremaju se samo težine (tako manje prostora model zauzima na disku)

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

EPOCHS = 10
checkpoint_filepath = '/tmp/checkpoint'
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

# Model weights are saved at the end of every epoch, if it's the best seen
# so far.
model.fit(epochs=EPOCHS, callbacks=[model_checkpoint_callback])

# The model weights (that are considered the best) are loaded into the model.
model.load_weights(checkpoint_filepath)
```

Vizualizacija loss-a i metrika tijekom treniranja pomoću TensorBoard-a

- TensorBoard je alat za pružanje uvida u mjerenja i vizualizacija tijekom procesa učenja modela
- Omogućuje praćenje mjernih podataka eksperimenta poput loss-a i točnosti, vizualiziranje grafa modela, aktivacija i sl.
- Potrebno je tijekom učenja zapisivati određene logove kojima se onda pristupa putem web browsera
- logovi se u Kerasu zapisuju pomoću odgovarajućeg callbacka



Vizualizacija loss-a i metrika tijekom treniranja pomoću TensorBoard-a

- potrebno je napisati odgovarajući callback, **argumenti**:
- **log_dir**: direktorij gdje se spremaju logovi
- **histogram_freq**: frekvencija u epohama s kojom se računaju aktivacije i histogram težina za slojeve modela
- **write_graph**: vizualizacija grafova u TensorBoard
- **write_images**: zapisivanje težina modela u obliku slika
- **write_steps_per_second**: logiranje broja trening koraka po sekundi
- **update_freq**: 'batch' ili 'epoch' (zapisuju li se rezultati u TensorBoard nakon svake epohe ili nakon svakog batcha); ako je integer to definira nakon koliko batcheva

`TensorBoard` class

```
tf.keras.callbacks.TensorBoard(
    log_dir="logs",
    histogram_freq=0,
    write_graph=True,
    write_images=False,
    write_steps_per_second=False,
    update_freq="epoch",
    profile_batch=2,
    embeddings_freq=0,
    embeddings_metadata=None,
    **kwargs
)
```

Vizualizacija loss-a i metrika tijekom treniranja pomoću TensorBoard-a

- funkciji `.fit()` predati callback

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir="./logs")
model.fit(x_train, y_train, epochs=2, callbacks=[tensorboard_callback])
# Then run the tensorboard command to view the visualizations.
```

- TensorBoard se pokreće sljedećom naredbom u terminalu:

```
tensorboard --logdir=path_to_your_logs
```

- pokrenuti web browser i upisati adresu:

```
http://localhost:6006/
```

Learning rate scheduling

- tijekom učenja modela moguće je automatski mijenjati stopu učenja (npr. eksponencijalno ju smanjivati) pomoću `LearningRateScheduler` callbacka

`LearningRateScheduler` class

```
tf.keras.callbacks.LearningRateScheduler(schedule, verbose=0)
```

- **schedule**: funkcija koja prima indeks epohe (kreće od 0) i trenutnu stopu učenja (float), a vraća novu vrijednost stope učenja (float)
- **verbose**: int. 0: bez ispisa, 1: update poruka

Smanjivanje stope učenja kada metrika dostigne plato

- dodatno fino podešavanje modela moguće je postići smanjivanjem stope učenja kada proces učenja počne stagnirati pomoću `ReduceLROnPlateau` calbacka
- neki argumenti:
- **monitor**: koja metrika se prati
- **factor**: koliko puta se smanjuje stopa učenja
- **patience**: koliko treba proći epoha bez promjene metrike nakon čega se smanjuje stopa učenja
- **cooldown**: broj epoha prije nego opet započne praćenje metrike i patience broja

`ReduceLROnPlateau` class

```
tf.keras.callbacks.ReduceLROnPlateau(
    monitor="val_loss",
    factor=0.1,
    patience=10,
    verbose=0,
    mode="auto",
    min_delta=0.0001,
    cooldown=0,
    min_lr=0,
    **kwargs
)
```

Transfer learning

- transfer learning je princip preuzimanja mreže naučene na jednom skupu podataka i iskorištavanje za rješavanje novog sličnog problema
- obično se primjenjuje u zadacima gdje podataka nema dovoljno kako bi se trenirao duboki model “od nule”
- postupak je sljedeći:
  - preuzeti slojeve prethodnog naučene mreže (bazni model)
  - zamrznuti ove slojeve (spriječiti mogućnost njihovog podešavanja) tijekom učenja
  - dodavanje novih slojeva na postojeće (zamrznute) slojeve
  - treniranje novih slojeva na vlastitom skupu podataka
  - (opcija: fino prepodešavanje cijelog modela s malom stopom učenja na vlastitom skupu podataka)

Transfer learning - Keras trainable

- slojevi sadrže tri atributa:
  - **weights**: lista svih težina sloja
  - **trainable_weights**: lista težina koje se mogu osvježiti tijekom učenja kako bi se smanjio loss
  - **non_trainable_weights**: lista težina koje se ne mijenjaju tijekom učenja

- ideja je kod transfer learning “zamrznuti” slojeve postavljanjem `trainable` parametra na False

Transfer learning - Keras trainable

- primjer: “zamrzavanje” prvog sloja - vrijednosti težina prije i poslije učenja će biti jednake

```
# Make a model with 2 layers
layer1 = keras.layers.Dense(3, activation="relu")
layer2 = keras.layers.Dense(3, activation="sigmoid")
model = keras.Sequential([keras.Input(shape=(3,)), layer1, layer2])

# Freeze the first layer
layer1.trainable = False

# Keep a copy of the weights of layer1 for later reference
initial_layer1_weights_values = layer1.get_weights()

# Train the model
model.compile(optimizer="adam", loss="mse")
model.fit(np.random.random((2, 3)), np.random.random((2, 3)))
```

Transfer learning - Keras trainable

- uobičajeno je uzeti neki od modela koji su istrenirani na ImageNetu (npr. VGG16, Resnet50, InceptionV3...)

```
base_model = keras.applications.Xception(
    weights='imagenet',  # Load weights pre-trained on ImageNet.
    input_shape=(150, 150, 3),
    include_top=False)  # Do not include the ImageNet classifier at the top.
```

- zamrznuti težine baznog modela

```
base_model.trainable = False
```

Transfer learning - Keras trainable

- dodati vlastite slojeve na bazni model

```
inputs = keras.Input(shape=(150, 150, 3))
# We make sure that the base_model is running in inference mode here,
# by passing `training=False`. This is important for fine-tuning, as you will
# learn in a few paragraphs.
x = base_model(inputs, training=False)
# Convert features of shape `base_model.output_shape[1:]` to vectors
x = keras.layers.GlobalAveragePooling2D()(x)
# A Dense classifier with a single unit (binary classification)
outputs = keras.layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
```

Transfer learning - Keras trainable

- trenirati model na vlastitim podacima

```
model.compile(optimizer=keras.optimizers.Adam(),
              loss=keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=[keras.metrics.BinaryAccuracy()])
model.fit(new_dataset, epochs=20, callbacks=..., validation_data=...)
```

Transfer learning - Keras trainable

- opcija - provesti fino podešavanje parametara
- paziti da se ne dogodi overfit (ići s manjom stopom učenja)!

```
# Unfreeze the base model
base_model.trainable = True

# It's important to recompile your model after you make any changes
# to the `trainable` attribute of any inner layer, so that your changes
# are take into account
model.compile(optimizer=keras.optimizers.Adam(1e-5),  # Very low learning rate
              loss=keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=[keras.metrics.BinaryAccuracy()])

# Train end-to-end. Be careful to stop before you overfit!
model.fit(new_dataset, epochs=10, callbacks=..., validation_data=...)
```

Izgradnja mreže za klasifikaciju prometnih znakova

GTSRB

- German Traffic Sign Recognition Benchmark (GTSRB) je dataset koji se sastoji od približno 50 000 slika prometnih znakova
- ukupno 43 klase

Izvor

Zadatak 1 – pripremanje podatkovnih skupova

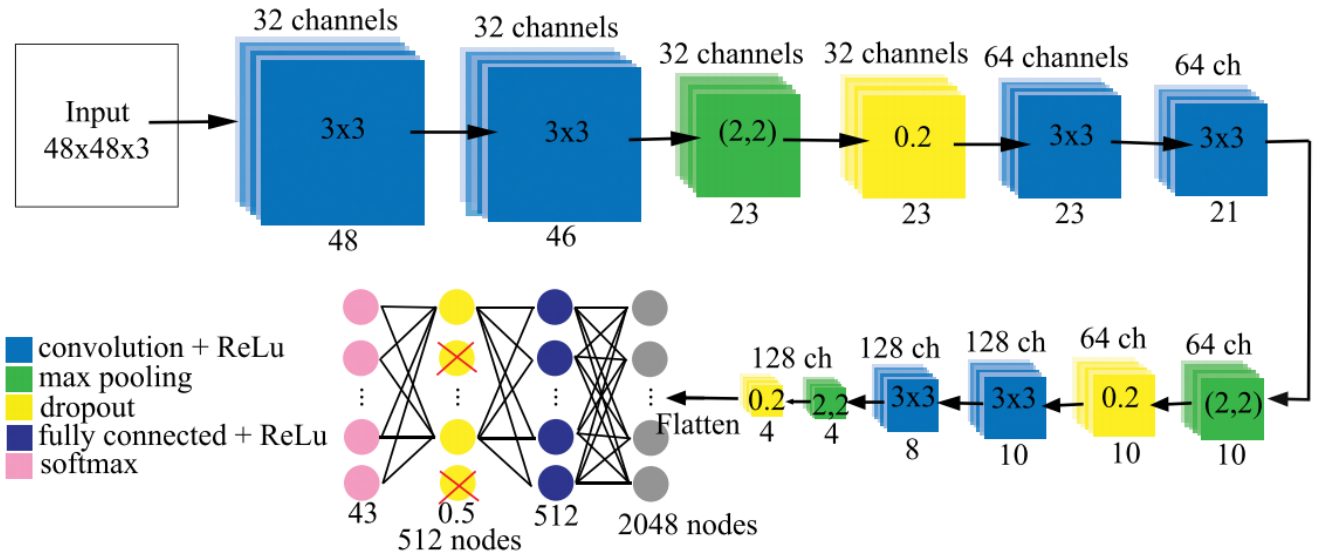
1. preuzmite dataset na adresi https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

2. Raspakirajte ga u direktorij naziva `gtsrb_dataset` i upoznajte se sa strukturom dataseta.

3. Podaci za treniranje pohranjeni su u poddirektorijima `Train/0`, `Train/1`, `Train/2` ... `Train/42`. Na isti način potrebno je strukturirati i testne podatke u direktoriju `Test`.
   - koristite `Test.csv` kako bi saznali koju sliku treba kopirati u koji podirektorij
   - npr. u petlji čitajte red po red iz navedenog csv-a te koristite `shutil.copy()` za kopiranje slika

Zadatak 2 – Izrada, učenje i evaluacija konvolucijske neuronske mreže

1. učitavanje podataka učinite pomoću `image_dataset_from_directory`
2. prikažite nekoliko slika iz skupa za učenje i skupa za testiranje
3. izradite konvolucijsku neuronsku mrežu sa slike pomoću funkcijskog API; provedite učenje mreže i evaluaciju na testnom skupu
4. spremite mrežu na disk



Izvor

Zadatak 3 – Inferencija

1. učitajte mrežu iz prošlog zadatka
2. izradite testni generator
3. izračunajte matricu zabune za testne podatke
4. dohvatite nekoliko slika testnih slika i prikažite rezultat inferencije

Zadatak 4 – implementacija callbacka

Implementirajte sljedeće callbacks i pokrenite učenje mreže:

1. `EarlyStopping` - nakon što se val_loss metrika ne povećava unutar 5 epoha
2. `ModelCheckpoint` – spremanje samo najboljeg modela na na temelju val_accuracy metrike
3. `ReduceLROnPlateau` – ako se val_loss ne smanji unutar tri epohe smanjiti za faktor 0.1
4. `TensorBoard` – pokrenite TensorBoard i pratite tijek učenja mreže

Zadatak 5 – augmentacija skupa za učenje

1. dodajte u rješenje prethodnog zadatka augmentaciju podataka pomoću klase `ImageDataGenerator`. Pri tome koristite metodu `flow_from_directory()` za dohvaćanje slika iz određenog direktorija. Prilikom definiranja transformacija u okviru augmentacije podataka uzmite u obzir da radite klasifikator prometnih znakova!
2. Prikažite nekoliko augmentiranih slika za učenje.
3. Pokrenite učenje mreže i izvršite evaluaciju mreže na testnom skupu. Dobivate li bolje rezultate na testnom skupu?

Zadatak 6 – korištenje dostupnih popularnih dubokih neuronskih mreža

1. Učitaje neki od postojećih modela u Keras-u (npr. Resnet50 ili MobileNetV2)
   - koristite opcije: `weights: „imagenet”, include_top = True`
2. učitajte sliku po želji (npr. sliku automobila, banane..) pomoću Keras funkcije `load_img`
3. Napravite predikciju za učitanu sliku (pogledajte help funkcije `load_img`).
4. Budući da se radi o mrežni naučenoj na ImageNet-u, pomoću funkcije `keras.applications.imagenet_utils.decode_predictions` pretvorite predikciju u odgovarajuću klasu
5. „igrajte” se s mrežom (izvršite inferenciju za druge primjere slika)

Zadatak 7 – korištenje dostupnih popularnih dubokih neuronskih mreža

- Zamijenite model iz 5 zadatka s nekim od gotovih modela dostupnih u Kerasu (npr. MobileNetV2).
  1. učitajte model uz opciju `include_top=False`; definirajte odgovarajući `input_tensor` i `input_shape`
  2. Na učitani model dodajte vlastite slojeve (npr. izlazni softmax sloj od 43 elementa)
  3. Provedite učenje cjelokupne mreže i evaluaciju na testnom skupu podataka

Zadatak 8 – DFGTSD skup podataka

- Na raspolaganju je skup podataka naziva DFGTSD: https://www.vicos.si/Downloads/DFGTSD
- Ovaj skup podataka ima čak 200 klasa

- Trening i testni skup podataka nalaze se na google driveu; nema zasebnog validacijskog skupa
- Stoga, podijelite trening skup u omjeru 85% (trening) 15% (validacija) pomoću python biblioteke split-folders
- Izgradite mrežu koja će imati što bolja predikcijska svojstva na testnom skupu – koristite sve tehnike za koje mislite da bi vam mogle pripomoći u dobivanju konačnog modela s dobrim generalizacijom (postojeće strukture, transfer learning…)
- kada smatrate da imate konačnu mrežu, provjerite performanse mreže na testnom skupu