

**Keras API.
Izgradnje duboke
neuronske mreže
za klasifikaciju.**



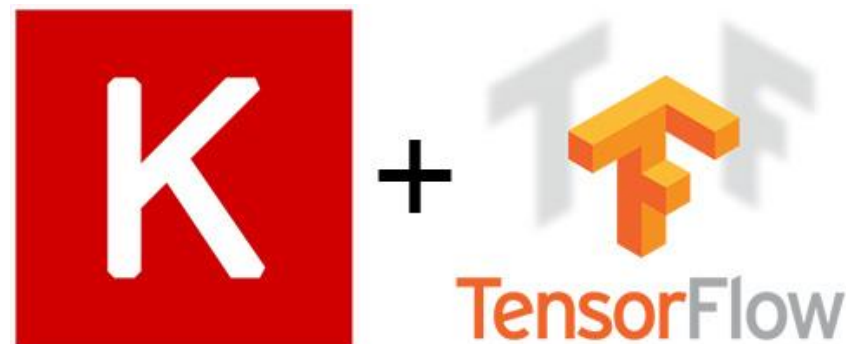
Sadržaj

- Keras API za duboko učenje
 - modeli, slojevi, aktivacijske funkcije
 - konfiguracija procesa učenja, učenje modela, evaluacija modela, spremanje modela
- Izgradnja mreže za klasifikaciju rukom pisanih brojeva (MNIST)
 - izgradnja, učenje i evaluacija potpuno povezane mreže
 - evaluacija potpuno povezane mreže nad vlastitim slikama
 - izgradnja, učenje i evaluacija konvolucijske neuronske mreže
 - detekcija i klasifikacija znamenki pomoću duboke mreže u video signalu
 - Klasifikator za složeniji dataset – Fashion-MNIST

Keras API za duboko učenje

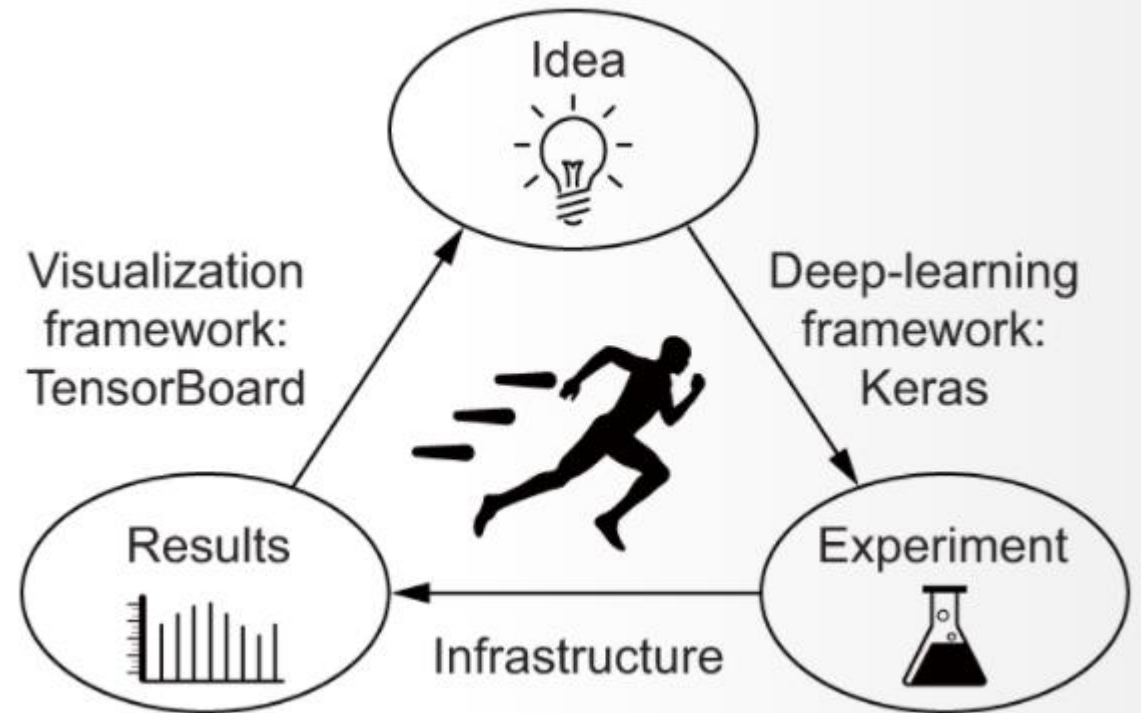
Keras

- Keras je API (engl. *Application Programming Interface*) za duboko učenje
- Kao platformu koristi Tensorflow 2 (prije je bio odvojen, danas dolazi zajedno s TensorFlow)
- Podrška za učenje na GPU!
- Omogućuje brzu izgradnju i eksperimentiranje s različitim modelima/algoritmima strojnog učenja



Keras

- Deep learning „za ljude”
- konzistentni i minimalni API
- odlična dokumentacija i različiti guideovi
- vrlo brzo se može izgraditi model te započeti učenje



Zadatak - Keras instalacija

- Napravite novo virtualno okruženje; Instalirati TensorFlow u novo okruženje:

```
pip install tensorflow
```

- Trenutno aktualna verzija je 2.7.0
- Nakon toga moguće je koristiti keras unutar python skripte, npr. na način:

```
from keras import layers
```

```
sloj = layers.Dense(units=10)  
print(sloj.activation)
```

- Ako navedeni kod ne proizvodi grešku prilikom pokretanja skripte, tada ste uspješno instalirali keras

Keras

- osnovni gradbeni blokovi Keras modela (models) su slojevi (layers) koji se kreiraju odgovarajućim klasama
- Za izgradnju modela i slojeva postoji API:
 - Models API
 - Layers API
- svaki sloj prima ulazne informacije, procesira ih i proizvodi izlaz
- Koriste se tenzori - generalizacija matrica i vektora na potencijalno veće dimenzije
- Keras tenzor je simbolički tenzor nalik objektu, kojem dodajemo određene attribute i koji nam omogućuju izgradnju Keras modela samo poznavanjem ulaza i izlaza modela
 - Za osnovnu upotrebu (danas) nije potrebno detaljno poznavanje tenzora

Keras Models API

- tri su načina kreiranja Keras modela:
 - **Sequential model** - vrlo jednostavno (jednostavan popis slojeva), ali je ograničeno na stack slojeva s jednim ulazom i s jednim izlazom (kako sam naziv kaže) → danas radimo
 - **Functional API** - jednostavan za korištenje, potpuno opremljen API koji podržava proizvoljne arhitekture modela. Dovoljno za većinu primjena.
 - **Model subclassing** – sve ide od nule. Ovo treba koristiti kada se grade specijalne mreže (tipično u istraživanju).

<https://keras.io/api/models/>

Keras Model class

- Klasa `Model` grupira slojeve u objekt kojeg je moguće trenirati i pomoću njega izvršiti inferenciju
 - Primjer kreiranja pomoću Functional API (o tome više iduće predavanje)
 - Sada samo obratite pažnju na zadnju liniju koda

- Važne metode ove klase:
 - `model.compile()`
 - `model.fit()`
 - `model.predict()`
 - `model.summary()`

```
import tensorflow as tf

inputs = tf.keras.Input(shape=(3,))
x = tf.keras.layers.Dense(4, activation=tf.nn.relu)(inputs)
outputs = tf.keras.layers.Dense(5, activation=tf.nn.softmax)(x)
model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

Keras Sequential

- najjednostavniji model u Kerasu je **Sequential model** - niz slojeva pri čemu svaki sloj ima točno jedan ulazni tenzor i točno jedan izlazni tenzor
- kreira se pomoću klase Sequential

The Sequential class

Sequential class

```
tf.keras.Sequential(layers=None, name=None)
```

Sequential groups a linear stack of layers into a **tf.keras.Model**.

Sequential provides training and inference features on this model.

Keras Sequential

- dva načina kreiranja modela:
 - prosljeđivanjem liste sa slojevima:

```
model = keras.Sequential(  
    [  
        layers.Dense(2, activation="relu"),  
        layers.Dense(3, activation="relu"),  
        layers.Dense(4),  
    ]  
)
```

- pomoću metode `.add()`

```
model = keras.Sequential()  
model.add(layers.Dense(2, activation="relu"))  
model.add(layers.Dense(3, activation="relu"))  
model.add(layers.Dense(4))
```

```
import tensorflow as tf  
from tensorflow import keras  
from tensorflow.keras import layers
```

Keras Sequential

- u prethodnom primjeru nije specificiran ulaz - stoga Keras ne zna koliko težina ima u sloju
- generalno, svaki sloj treba znati dimenziju svoga ulaza kako bi mogao kreirati težine

```
layer = layers.Dense(3)
layer.weights # Empty
```

- prilikom prvog pozivanja se definiraju težine:

```
# Call layer on a test input
x = tf.ones((1, 4))
y = layer(x)
layer.weights # Now it has weights, of shape (4, 3) and (3,)
```

Keras Sequential

- slično vrijedi i za `Sequential` model - ako ga se kreira bez veličine ulaza on još nije izgrađen (nema težine)

```
model = keras.Sequential(  
    [  
        layers.Dense(2, activation="relu"),  
        layers.Dense(3, activation="relu"),  
        layers.Dense(4),  
    ]  
) # No weights at this stage!  
  
# At this point, you can't do this:  
# model.weights  
  
# You also can't do this:  
# model.summary()  
  
# Call the model on a test input  
x = tf.ones((1, 4))  
y = model(x)  
print("Number of weights after calling the model:", len(model.weights)) # 6
```

Zašto se ispisuje broj 6?

Možete li reći koliko parametara ima u svakom sloju?

Što ispisuje `print(model.weights)`?

Kolika je veličina izlaza (`y.shape`) ako je ulaz `tf.ones((100,4))`?

Keras Sequential

- pomoću `model.summary()` prikazuje se tip pojedinog sloja, izlazna dimenzija i broj parametara u sloju
- ovo je moguće napraviti samo kada je “model” izgrađen

```
Model: "sequential"
Layer (type)                Output Shape              Param #
=====
dense (Dense)                (1, 2)                    10
dense_1 (Dense)              (1, 3)                     9
dense_2 (Dense)              (1, 4)                    16
=====
Total params: 35
Trainable params: 35
Non-trainable params: 0
```

Keras Sequential

- Keras `Sequential` klasa ima dvije metode:
 - `add` metoda – dodaje instancu sloja na vrh (kraj) niza slojeva
 - `pop` metoda – briše instancu sloja

`add` method

```
Sequential.add(layer)
```

`pop` method

```
Sequential.pop()
```

Keras Input objekt

- veličinu ulaznog tenzora moguće je definirati kod Sequential modela korištenjem `Input` objekta
- dodaj se kao ulazni sloj, ali ga `model.summary()` ne prikazuje

```
model = keras.Sequential()  
model.add(keras.Input(shape=(4,)))  
model.add(layers.Dense(2, activation="relu"))  
  
model.summary()
```

kada je ulaz jednodimenzionalni vektor

npr. za RGB sliku rezolucije 200x100:
`shape=(200,100,3)`

```
model = keras.Sequential()  
model.add(layers.Dense(2, activation="relu", input_shape=(4,)))  
  
model.summary()
```


Keras Layers API

- sloj je osnovni gradbeni blok neuronske mreže u Kerasu - to je tensor-in tensor-out računalna funkcija s određenim stanjem (sprema se u TensorFlow varijablu – to su težine sloja)
- Sloj se može pozvati kao bilo koja funkcija
- za razliku od funkcije, sloj sadržava svoje stanje, osvježava se tijekom učenja i sprema u `layer.weights`

```
from tensorflow.keras import layers

layer = layers.Dense(32, activation='relu')
inputs = tf.random.uniform(shape=(10, 20))
outputs = layer(inputs)
```

Keras Layers

- Keras Layers API omogućava korištenje gotovih slojeva kao i kreiranje vlastitih

Core layers

[Input object](#)
[Dense layer](#)
[Activation layer](#)
[Embedding layer](#)
[Masking layer](#)
[Lambda layer](#)

Convolution layers

[Conv1D layer](#)
[Conv2D layer](#)
[Conv3D layer](#)
[SeparableConv1D layer](#)
[SeparableConv2D layer](#)
[DepthwiseConv2D layer](#)
[Conv2DTranspose layer](#)
[Conv3DTranspose layer](#)

Pooling layers

[MaxPooling1D layer](#)
[MaxPooling2D layer](#)
[MaxPooling3D layer](#)
[AveragePooling1D layer](#)
[AveragePooling2D layer](#)
[AveragePooling3D layer](#)
[GlobalMaxPooling1D layer](#)
[GlobalMaxPooling2D layer](#)
[GlobalMaxPooling3D layer](#)
[GlobalAveragePooling1D layer](#)
[GlobalAveragePooling2D layer](#)
[GlobalAveragePooling3D layer](#)

Recurrent layers

[LSTM layer](#)
[GRU layer](#)
[SimpleRNN layer](#)
[TimeDistributed layer](#)
[Bidirectional layer](#)
[ConvLSTM2D layer](#)
[Base RNN layer](#)

Regularization layers

[Dropout layer](#)
[SpatialDropout1D layer](#)
[SpatialDropout2D layer](#)
[SpatialDropout3D layer](#)
[GaussianDropout layer](#)
[GaussianNoise layer](#)
[ActivityRegularization layer](#)
[AlphaDropout layer](#)

Normalization layers

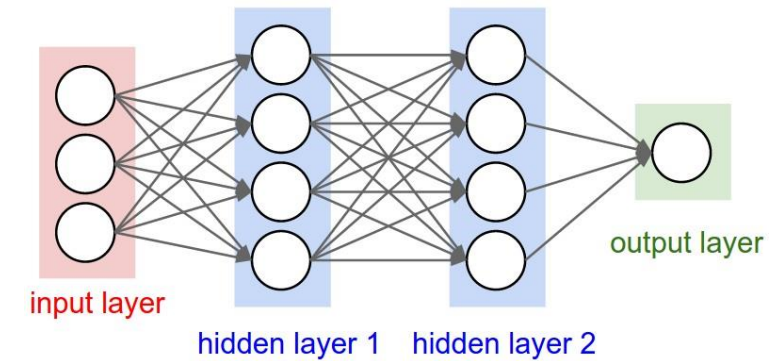
[BatchNormalization layer](#)
[LayerNormalization layer](#)

Dense layer

- potpuno povezani sloj

Dense class

```
tf.keras.layers.Dense(  
    units,  
    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,  
    **kwargs  
)
```


$$\text{output} = \text{activation}(\text{dot}(\text{input}, \text{kernel}) + \text{bias})$$

Dense layer

Argumenti:

units: Positive integer, dimensionality of the output space.

activation: Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: $a(x) = x$).

use_bias: Boolean, whether the layer uses a bias vector.

kernel_initializer: Initializer for the kernel weights matrix.

bias_initializer: Initializer for the bias vector.

kernel_regularizer: Regularizer function applied to the kernel weights matrix.

bias_regularizer: Regularizer function applied to the bias vector.

activity_regularizer: Regularizer function applied to the output of the layer (its "activation").

kernel_constraint: Constraint function applied to the kernel weights matrix.

bias_constraint: Constraint function applied to the bias vector.

Dense layer

Input shape

N-D tensor with shape: (batch_size, ..., input_dim). The most common situation would be a 2D input with shape (batch_size, input_dim).

Output shape

N-D tensor with shape: (batch_size, ..., units). For instance, for a 2D input with shape (batch_size, input_dim), the output would have shape (batch_size, units).

Dense layer - primjer

```
>>> # Create a `Sequential` model and add a Dense layer as the first layer.
>>> model = tf.keras.models.Sequential()
>>> model.add(tf.keras.Input(shape=(16,)))
>>> model.add(tf.keras.layers.Dense(32, activation='relu'))
>>> # Now the model will take as input arrays of shape (None, 16)
>>> # and output arrays of shape (None, 32).
>>> # Note that after the first layer, you don't need to specify
>>> # the size of the input anymore:
>>> model.add(tf.keras.layers.Dense(32))
>>> model.output_shape
(None, 32)
```

Conv2D layer

Conv2D class

```
tf.keras.layers.Conv2D(  
    filters,  
    kernel_size,  
    strides=(1, 1),  
    padding="valid",  
    data_format=None,  
    dilation_rate=(1, 1),  
    groups=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,  
    **kwargs  
)
```

**klasična konvolucija
(npr. prostorna
konvolucija nad
slikama)**

Conv2D layer

Neki argumenti:

filters: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).

kernel_size: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

strides: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value $\neq 1$ is incompatible with specifying any `dilation_rate` value $\neq 1$.

padding: one of "valid" or "same" (case-insensitive). "valid" means no padding. "same" results in padding with zeros evenly to the left/right or up/down of the input such that output has the same height/width dimension as the input.

data_format: A string, one of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape (batch_size, height, width, channels) while `channels_first` corresponds to inputs with shape (batch_size, channels, height, width). It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be `channels_last`.

activation: Activation function to use. If you don't specify anything, no activation is applied (see `keras.activations`).

use_bias: Boolean, whether the layer uses a bias vector.

kernel_initializer: Initializer for the kernel weights matrix (see `keras.initializers`). Defaults to 'glorot_uniform'.

bias_initializer: Initializer for the bias vector (see `keras.initializers`). Defaults to 'zeros'.

Conv2D layer

Input shape

4-D tensor with shape: batch_shape + (channels, rows, cols) if data_format='channels_first' or 4-D tensor with shape: batch_shape + (rows, cols, channels) if data_format='channels_last'.

Output shape

4-D tensor with shape: batch_shape + (filters, new_rows, new_cols) if data_format='channels_first' or 4-D tensor with shape: batch_shape + (new_rows, new_cols, filters) if data_format='channels_last'. rows and cols values might have changed due to padding.

Conv2D layer

- primjeri

```
>>> # The inputs are 28x28 RGB images with `channels_last` and the batch
>>> # size is 4.
>>> input_shape = (4, 28, 28, 3)
>>> x = tf.random.normal(input_shape)
>>> y = tf.keras.layers.Conv2D(
... 2, 3, activation='relu', input_shape=input_shape[1:])(x)
>>> print(y.shape)
(4, 26, 26, 2)
```

```
>>> # With `padding` as "same".
>>> input_shape = (4, 28, 28, 3)
>>> x = tf.random.normal(input_shape)
>>> y = tf.keras.layers.Conv2D(
... 2, 3, activation='relu', padding="same", input_shape=input_shape[1:])(x)
>>> print(y.shape)
(4, 28, 28, 2)
```

Pooling layers

- različiti slojevi koji rade sažimanje podataka

MaxPooling2D class

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

- sažima podatke po visini i širini uzimajući maksimalnu vrijednost u prozoru definiranom s **pool_size**
- **strides** definira za koliko se pomiče prozor u svakoj dimenziji

MaxPooling2D layer

Argumenti:

pool_size: integer or tuple of 2 integers, window size over which to take the maximum. (2, 2) will take the max value over a 2x2 pooling window. If only one integer is specified, the same window length will be used for both dimensions.

strides: Integer, tuple of 2 integers, or None. Strides values. Specifies how far the pooling window moves for each pooling step. If None, it will default to pool_size.

padding: One of "valid" or "same" (case-insensitive). "valid" means no padding. "same" results in padding evenly to the left/right or up/down of the input such that output has the same height/width dimension as the input.

data_format: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

MaxPooling2D layer

Input shape

If data_format='channels_last': 4D tensor with shape (batch_size, rows, cols, channels).

If data_format='channels_first': 4D tensor with shape (batch_size, channels, rows, cols).

Output shape

If data_format='channels_last': 4D tensor with shape (batch_size, pooled_rows, pooled_cols, channels).

If data_format='channels_first': 4D tensor with shape (batch_size, channels, pooled_rows, pooled_cols).

MaxPooling2D layer

- primjer

```
>>> x = tf.constant([[1., 2., 3.],  
...                  [4., 5., 6.],  
...                  [7., 8., 9.]])  
>>> x = tf.reshape(x, [1, 3, 3, 1])  
>>> max_pool_2d = tf.keras.layers.MaxPooling2D(pool_size=(2, 2),  
...      strides=(1, 1), padding='valid')  
>>> max_pool_2d(x)  
<tf.Tensor: shape=(1, 2, 2, 1), dtype=float32, numpy=  
  array([[[[5.,  
            [6.]],  
          [[8.],  
            [9.]]]], dtype=float32)>
```

- što će biti rezultat ako se postavi **strides** = (2,2) ?

Flatten layer

- pretvara ulazni tenzor u 1D

Flatten class

```
tf.keras.layers.Flatten(data_format=None, **kwargs)
```

- primjer

```
>>> model = tf.keras.Sequential()  
>>> model.add(tf.keras.layers.Conv2D(64, 3, 3, input_shape=(3, 32, 32)))  
>>> model.output_shape  
(None, 1, 10, 64)
```

```
>>> model.add(Flatten())  
>>> model.output_shape  
(None, 640)
```

Dropout layer

- primjenjuje Dropout tehniku na ulaz

Dropout class

```
tf.keras.layers.Dropout(rate, noise_shape=None, seed=None, **kwargs)
```

- nasumično se postavljaju vrijednosti ulaza na 0 s frekvencijom izbacivanja **rate** (preostale vrijednosti se skaliraju s konstantom **$1/(1 - \text{rate})$** kako bi suma svih ulaza ostala ista)
- aktivan je samo tijekom učenja

Dropout layer

Argumenti

rate: Float between 0 and 1. Fraction of the input units to drop.

noise_shape: 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape (batch_size, timesteps, features) and you want the dropout mask to be the same for all timesteps, you can use noise_shape=(batch_size, 1, features).

seed: A Python integer to use as random seed.

Call arguments

inputs: Input tensor (of any rank).

training: Python boolean indicating whether the layer should behave in training mode (adding dropout) or in inference mode (doing nothing).

Dropout layer

Primjer

```
>>> tf.random.set_seed(0)
>>> layer = tf.keras.layers.Dropout(.2, input_shape=(2,))
>>> data = np.arange(10).reshape(5, 2).astype(np.float32)
>>> print(data)
[[0. 1.]
 [2. 3.]
 [4. 5.]
 [6. 7.]
 [8. 9.]]
>>> outputs = layer(data, training=True)
>>> print(outputs)
tf.Tensor(
[[ 0.    1.25]
 [ 2.5   3.75]
 [ 5.    6.25]
 [ 7.5   8.75]
 [10.    0.  ]], shape=(5, 2), dtype=float32)
```

Zašto ovakav primjer?

Ako se Dropout doda u model, njegov utjecaj se ne vidi na izlazne podatke jer je isključen tijekom inferencije!

BatchNormalization layer

- ovaj sloj normalizira ulaze na način da imaju srednju vrijednost približno 0 i standardnu devijaciju približno 1

BatchNormalization class

```
tf.keras.layers.BatchNormalization(  
    axis=-1,  
    momentum=0.99,  
    epsilon=0.001,  
    center=True,  
    scale=True,  
    beta_initializer="zeros",  
    gamma_initializer="ones",  
    moving_mean_initializer="zeros",  
    moving_variance_initializer="ones",  
    beta_regularizer=None,  
    gamma_regularizer=None,  
    beta_constraint=None,  
    gamma_constraint=None,  
    **kwargs  
)
```

BatchNormalization layer

Neki argumenti:

axis: Integer, the axis that should be normalized (typically the features axis). For instance, after a Conv2D layer with `data_format="channels_first"`, set `axis=1` in BatchNormalization.

momentum: Momentum for the moving average.

epsilon: Small float added to variance to avoid dividing by zero.

center: If True, add offset of beta to normalized tensor. If False, beta is ignored.

scale: If True, multiply by gamma. If False, gamma is not used. When the next layer is linear (also e.g. `nn.relu`), this can be disabled since the scaling will be done by the next layer.

beta_initializer: Initializer for the beta weight.

gamma_initializer: Initializer for the gamma weight.

moving_mean_initializer: Initializer for the moving mean.

moving_variance_initializer: Initializer for the moving variance.

BatchNormalization layer

Input shape

Arbitrary. Use the keyword argument `input_shape` (tuple of integers, does not include the samples axis) when using this layer as the first layer in a model.

Output shape

Same shape as input.

Layer activation functions

- aktivacijsku funkciju moguće je primijeniti kroz zaseban aktivacijski sloj ili ju definirati izravno prilikom kreiranja sloja (kod svih unaprijednih slojeva)

```
model.add(layers.Dense(64, activation=activations.relu))
```

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

```
from tensorflow.keras import layers
from tensorflow.keras import activations

model.add(layers.Dense(64))
model.add(layers.Activation(activations.relu))
```

Relu activation function

relu function

```
tf.keras.activations.relu(x, alpha=0.0, max_value=None, threshold=0)
```

Argumenti:

x: Input tensor or variable.

alpha: A float that governs the slope for values lower than the threshold.

max_value: A float that sets the saturation threshold (the largest value the function will return).

threshold: A float giving the threshold value of the activation function below which values will be damped or set to zero.

Vraća:

A Tensor representing the input tensor, transformed by the relu activation function. Tensor will be of the same shape and dtype of input x.

Relu activation function

- Primjer

```
>>> foo = tf.constant([-10, -5, 0.0, 5, 10], dtype = tf.float32)
>>> tf.keras.activations.relu(foo).numpy()
array([ 0.,  0.,  0.,  5., 10.], dtype=float32)
>>> tf.keras.activations.relu(foo, alpha=0.5).numpy()
array([-5. , -2.5,  0. ,  5. , 10. ], dtype=float32)
>>> tf.keras.activations.relu(foo, max_value=5).numpy()
array([0., 0., 0., 5., 5.], dtype=float32)
>>> tf.keras.activations.relu(foo, threshold=5).numpy()
array([-0., -0.,  0.,  0., 10.], dtype=float32)
```


Sigmoid activation function

sigmoid function

```
tf.keras.activations.sigmoid(x)
```

Argumenti

x: Input tensor.

Vraća:

Tensor with the sigmoid activation: $1 / (1 + \exp(-x))$.

Elu activation function

elu function

```
tf.keras.activations.elu(x, alpha=1.0)
```

Argumenti:

x: Input tensor.

alpha: A scalar, slope of negative section. alpha controls the value to which an ELU saturates for negative net inputs.

Vraća:

The exponential linear unit (ELU) activation function: x if $x > 0$ and $\alpha * (\exp(x) - 1)$ if $x < 0$

Softmax activation function

softmax function

```
tf.keras.activations.softmax(x, axis=-1)
```

Argumenti:

x : Input tensor.

axis: Integer, axis along which the softmax normalization is applied.

Vraća:

Tensor, output of softmax transformation (all values are non-negative and sum to 1).

Softmax activation function

softmax function

```
tf.keras.activations.softmax(x, axis=-1)
```

Argumenti:

x : Input tensor.

axis: Integer, axis along which the softmax normalization is applied.

Vraća:

Tensor, output of softmax transformation (all values are non-negative and sum to 1).

Treniranje i evaluacija modela

- tipičan redoslijed prilikom izgradnje neuronske mreže:
 - Treniranje mreže
 - Validacija na validacijskom skupu (odvoji se dio iz raspoloživog skupa podataka za treniranje)
 - Kada smo završili izgradnju i podešavanje mreže - testiranje mreže na potpuno novom skupu
- Keras model API pruža gotove metode za podešavanje konfiguracije učenja, pokretanje učenja te dobivanje metrika poput točnosti mreže
- ovisno o veličini mreže i podatkovnog skupa za učenje može potrajati satima ili čak i danima (na GPU)

Konfiguracija procesa treninga

- `.compile()` metoda služi za definiranje kriterijske funkcije (loss function), metrike i numeričke metode za optimizaciju

lista; može se koristiti više metrika

```
model.compile(  
    optimizer=keras.optimizers.RMSprop(learning_rate=1e-3),  
    loss=keras.losses.SparseCategoricalCrossentropy(),  
    metrics=[keras.metrics.SparseCategoricalAccuracy()],  
)
```

```
model.compile(  
    optimizer="rmsprop",  
    loss="sparse_categorical_crossentropy",  
    metrics=["sparse_categorical_accuracy"],  
)
```

Konfiguracija procesa treninga

- postoje ugrađene metrike, kriterijske funkcije i metode optimizacije

Optimizers:

SGD() (with or without momentum)
RMSprop()
Adam()
itd.

Losses:

MeanSquaredError()
KLDivergence()
CosineSimilarity()
itd.


Metrics:

AUC()
Precision()
Recall()
MeanSquaredError()
itd.

Konfiguracija procesa treninga

- Metrike – prikazuju se tijekom učenja i logiraju se u objekt `History` kojeg vraća metoda `fit`

```
model.compile(  
    optimizer='adam',  
    loss='mean_squared_error',  
    metrics=[  
        metrics.MeanSquaredError(name='my_mse'),  
        metrics.AUC(name='my_auc'),  
    ]  
)
```



lista; može se koristiti više metrika u isto vrijeme

Treniranje modela

- `.fit()` metoda služi za pokretanje procesa treniranja
- najvažniji argumenti:
 - x,y**: numpy polja, TensorFlow tensor, tf.data dataset, generator
 - batch_size**: veličina batcha
 - epochs**: broj prolaska kroz sve trening podatke
 - verbose**: način ispisa rezultata (0, 1 ili 2)
 - callbacks**: lista callbacka koji se primjenjuju tijekom treniranja
 - validation_split**: ovo je postotak podataka koji se ne koristi za učenje već se koristi za validaciju na kraju svake epohe

fit method

```
Model.fit(  
    x=None,  
    y=None,  
    batch_size=None,  
    epochs=1,  
    verbose="auto",  
    callbacks=None,  
    validation_split=0.0,  
    validation_data=None,  
    shuffle=True,  
    class_weight=None,  
    sample_weight=None,  
    initial_epoch=0,  
    steps_per_epoch=None,  
    validation_steps=None,  
    validation_batch_size=None,  
    validation_freq=1,  
    max_queue_size=10,  
    workers=1,  
    use_multiprocessing=False,  
)
```

Treniranje modela - primjer

- primjer podešavanja procesa učenja i pokretanje učenja

```
model = keras.Sequential(  
    [  
        keras.Input(shape=input_shape),  
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Flatten(),  
        layers.Dropout(0.5),  
        layers.Dense(num_classes, activation="softmax"),  
    ]  
)
```

```
batch_size = 128  
epochs = 15  
  
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])  
  
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

Evaluacija modela

- izgrađeni model moguće je evaluirati pomoću metode `.evaluate()`
- vraća test loss (ili listu ako model ima više metrika)
- najvažniji argumenti:

x,y: numpy polja, TensorFlow tensor, tf.data dataset, generator

batch_size: veličina batcha

verbose: način ispisa rezultata (0 ili 1)

callbacks: lista callbacka koji se primjenjuju tijekom evaluacije

evaluate method

```
Model.evaluate(  
    x=None,  
    y=None,  
    batch_size=None,  
    verbose=1,  
    sample_weight=None,  
    steps=None,  
    callbacks=None,  
    max_queue_size=10,  
    workers=1,  
    use_multiprocessing=False,  
    return_dict=False,  
    **kwargs  
)
```

Evaluacija modela

- izlazne vrijednosti modela za ulazne podatke (inferencija) dobivaju se pomoću metode `.predict()`
- vraća numpy polje s predikcijama
- najvažniji argumenti:

x: numpy polje, TensorFlow tensor, tf.data dataset, generator

batch_size: veličina batcha (default je 32)

verbose: način ispisa rezultata (0 ili 1)

callbacks: lista callbacka koji se primjenjuju tijekom evaluacije

predict method

```
Model.predict(  
    x,  
    batch_size=None,  
    verbose=0,  
    steps=None,  
    callbacks=None,  
    max_queue_size=10,  
    workers=1,  
    use_multiprocessing=False,  
)
```

Evaluacija modela

- primjer evaluacije modela

```
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

Spremanje modela na disk

- keras model sadrži više dijelove (arhitekturu, konfiguraciju, težine, optimizacijsku metode, loss, metrike)
- spremanje modela može se napraviti pomoću metode `.save()`
- može se sve spremiti ili samo neki dijelovi (npr. težine - ovo je tipično tijekom učenja modela)
- metoda sprema `Tensorflow SavedModel` (preferirano) ili `HDF5` datoteku

save method

```
Model.save(  
    filepath,  
    overwrite=True,  
    include_optimizer=True,  
    save_format=None,  
    signatures=None,  
    options=None,  
    save_traces=True,  
)
```

Spremanje modela na disk

- spremanje modela

```
model = ... # Get model (Sequential, Functional Model, or Model subclass)
model.save('path/to/location')
```

- učitavanje modela

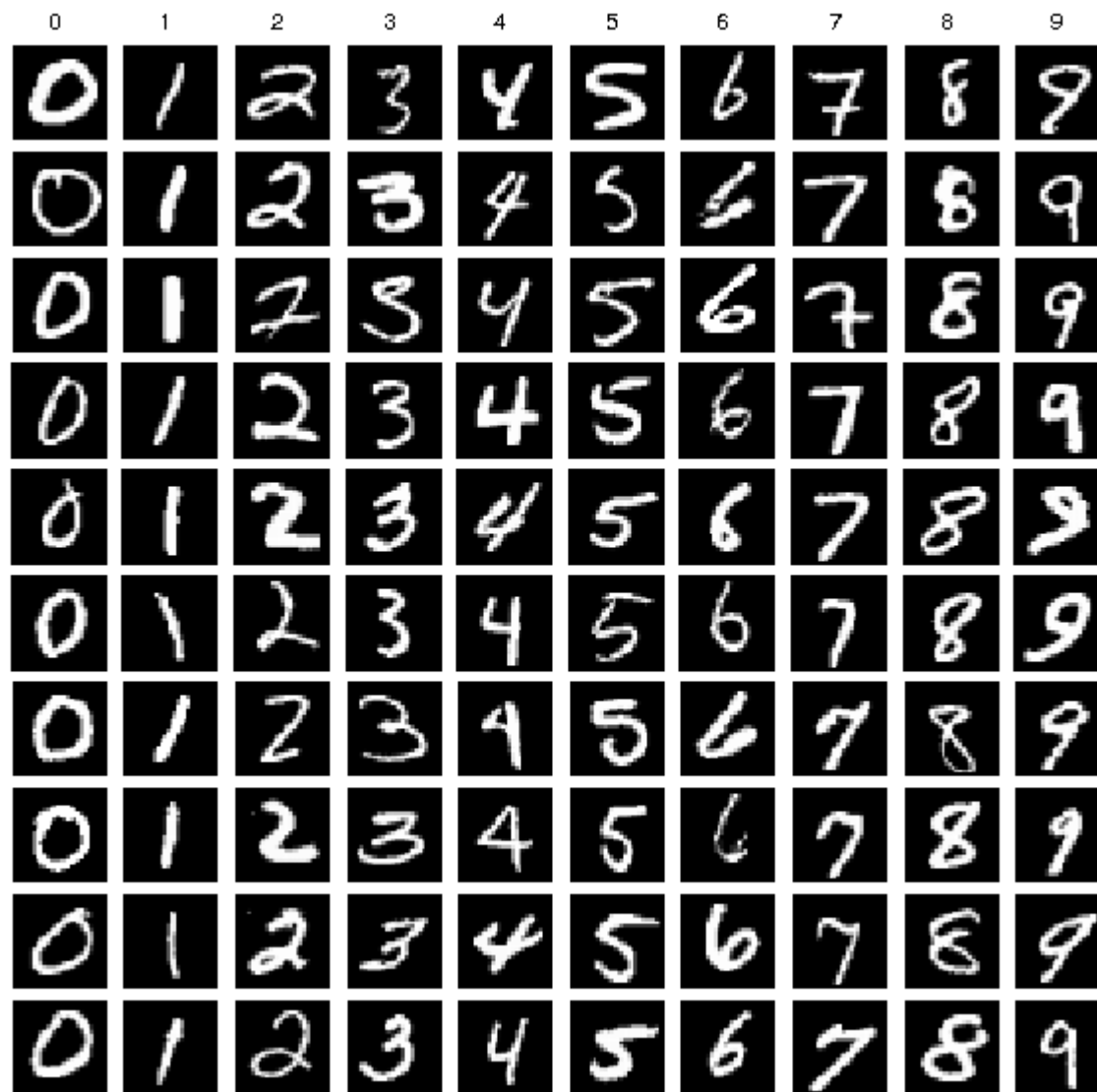
```
from tensorflow import keras
model = keras.models.load_model('path/to/location')
```

Izgradnja mreže za klasifikaciju rukom pisanih brojeva (MNIST)

MNIST

- MNIST je jednostavni dataset kojeg ćemo koristiti za ilustraciju rada s Kerasom
- MNIST - problem **klasifikacije** rukom pisanih brojeva
- Ovaj skup sadrži slike rukom pisanih brojeva koje su pisali zaposlenici u United States Census Bureau i američki studenti.
- Slike su zapisane u sivim tonovima odnosno svaki piksel na slici ima vrijednost u rasponu od 0 do 255.
- Slike su normirane na dimenziju 28 x 28 piksela.
- Svaka slika ima odgovarajuću oznaku tj. labelu (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).
- MNIST sadrži skup podataka za učenje od 60,000 slika, te skup podataka za testiranje koji sadrži 10,000 slika

MNIST



Obratite pažnju na način
pisanja
pojedinih znamenki!

(američki stil pisanja, pogledajte
znamenke 1, 4, 5, 7)

MNIST učitavanje u Kerasu

- Keras sadrži nekoliko jednostavnih datasetova poput MNIST, CIFAR10, Boston housing price dataset...
- postoji gotove funkcije za učitavanje MNIST podatkovnog skupa

`load_data` function

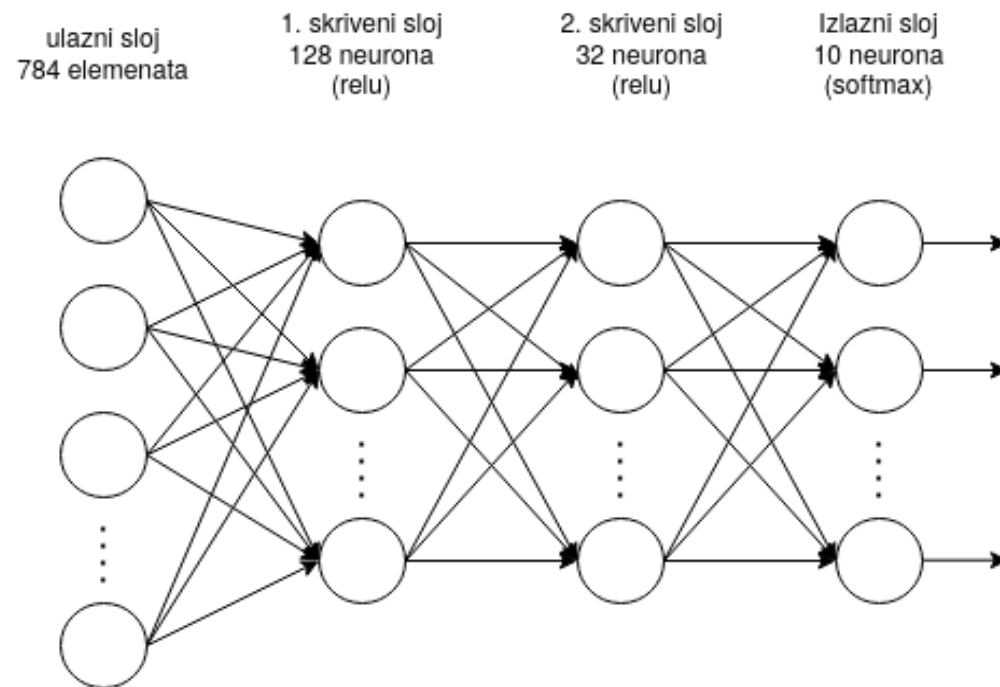
```
tf.keras.datasets.mnist.load_data(path="mnist.npz")
```

- funkcija vraća tuple train i test numpy polja:
`(x_train, y_train), (x_test, y_test)`

Zadatak 1 - izgradnja, učenje i evaluacija potpuno povezane mreže za klasifikaciju

- Otvorite `MNIST_FCN_training_AI.py`
- Korištenjem Keras API:

- učitajte MNIST
- prikažite nekoliko trening slika
 - koristite matplotlib
- pretprocesirajte ulazne podatke
- izgradite mrežu sa slike
- provedite učenje mreže
- evaluacija mreže na testnom skupu
 - matrica zabune
 - precision, recall, F1 mjera
- prikažite nekoliko dobro klasificiranih i nekoliko loše klasificiranih primjera

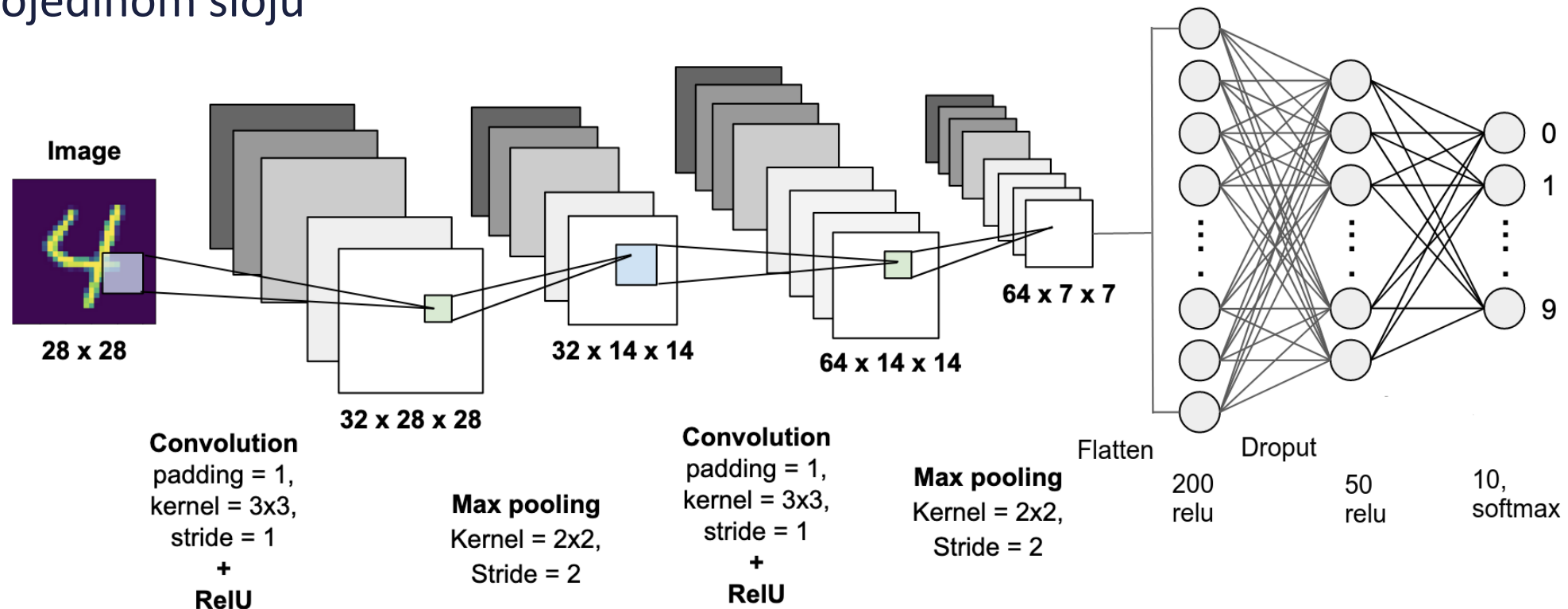


Zadatak 2 - evaluacija nad vlastitim slikama

- Skripta `MNIST_FCN_inference_AI.py` učitava `test.png` sliku u numpy polje
- Dodajte na odgovarajuća mjesta kod (TODO) kojim ćete izvršiti inferenciju za učitanu sliku
- Kako izgleda izlaz iz mreže?
- Prepoznaje li mreža točno o kojoj znamenki se radi?
- Mijenjate `test.png` sliku pomoću nekog alata za obradu slike (npr. Paint); upisujte redom znamenke od 0 - 9 i provjerite svaki puta što mreža daje na izlazu

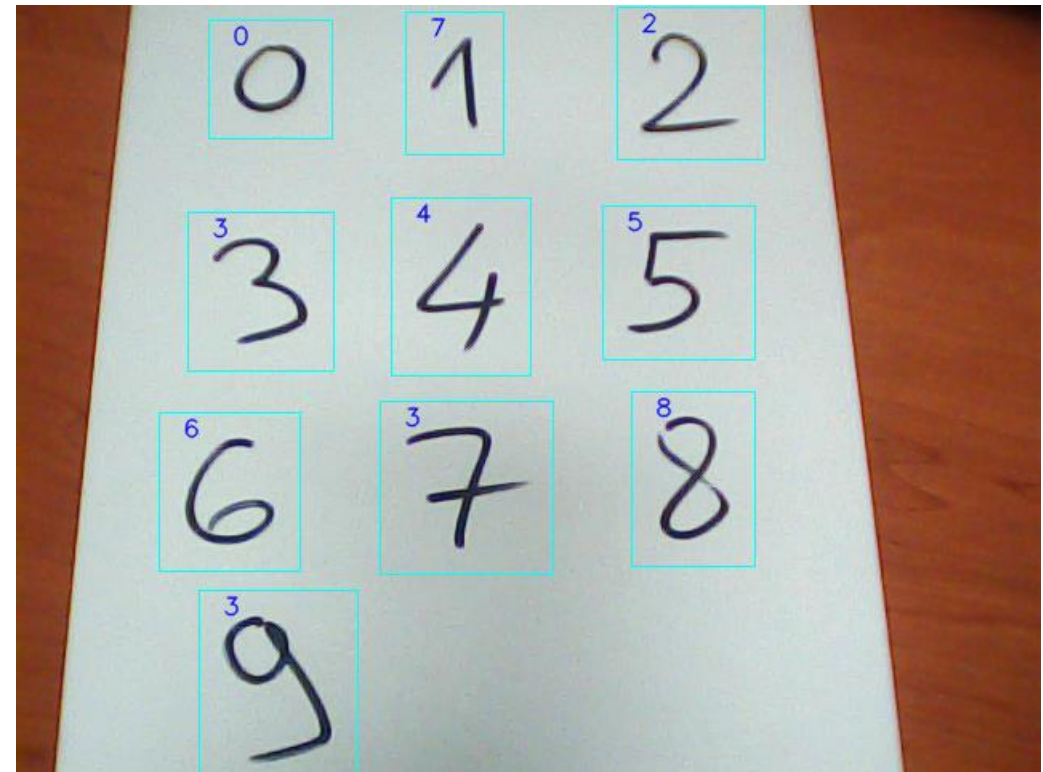
Zadatak 3 – izgradnja, učenje i evaluacija konvolucijske neuronske mreže

- ponovite zadatak 1, ali za slučaj konvolucijske neuronske mreže sa slike
- pomoću `model.summary()` provjerite imate li odgovarajuće dimenzije u pojedinom sloju



Zadatak 4 – detekcija i klasifikacija znamenki u video signalu

- skripta `MNIST_camera_AI.py` učitava video signal s kamere i provodi odgovarajuću obradu kako bi izdvojila znamenke sa slike (npr. kamera snima papir s napisanim znamenkama)
- dodajte kod kojim ćete izvršiti inferenciju pomoću naučene mreže
- napišite na papir s flomasterom znamenke od 0 - 9
- prepozna li mreža točno o kojima znamenkama se radi?



Zadatak 5 - Fashion MNIST

- MNIST dataset je jednostavan za moderne konvolucijske neuronske mreže – postižu 99.7%
- Fashion-MNIST → složeniji dataset (klase: T-shirt, Trouser, Pullover...) koji je pogodniji za benchmark dubokih modela
 - <https://github.com/zalandoresearch/fashion-mnist>
- Pokušajte napraviti model koji ima što veću točnost na testnom skupu
 - Pobjednik dobiva



Upute za CUDA i cuDNN - opcija

- Duboko učenje se može izvršiti na CPU samo za male datasetove poput MNIST
- Za iole složenije probleme proces učenja trajao bi predugo – koristi se GPU
- Instalirajte drive, CUDA i cuDNN kako biste mogli trenirati modele na GPU
- <https://docs.nvidia.com/deeplearning/cudnn/install-guide/index.html>
- <https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html>
- Nakon instalacije Keras će automatski koristiti GPU
- Pripazite koja verzija tensorflowa podržava koju CUDU i cuDNN, provjerite:
 - https://www.tensorflow.org/install/source#tested_build_configurations