10. hét / Prológus

A mai órán a következőkről lesz szó:

- Idősorok elemzése 1D-CNN és RNN segítségével
- Zajcsökkentés konvolúciós rétegekből felépített Autoencoder segítségével
- Képszegmentáció haladó konvolúciós hálóval

10. hét / I. Idősorok elemzése

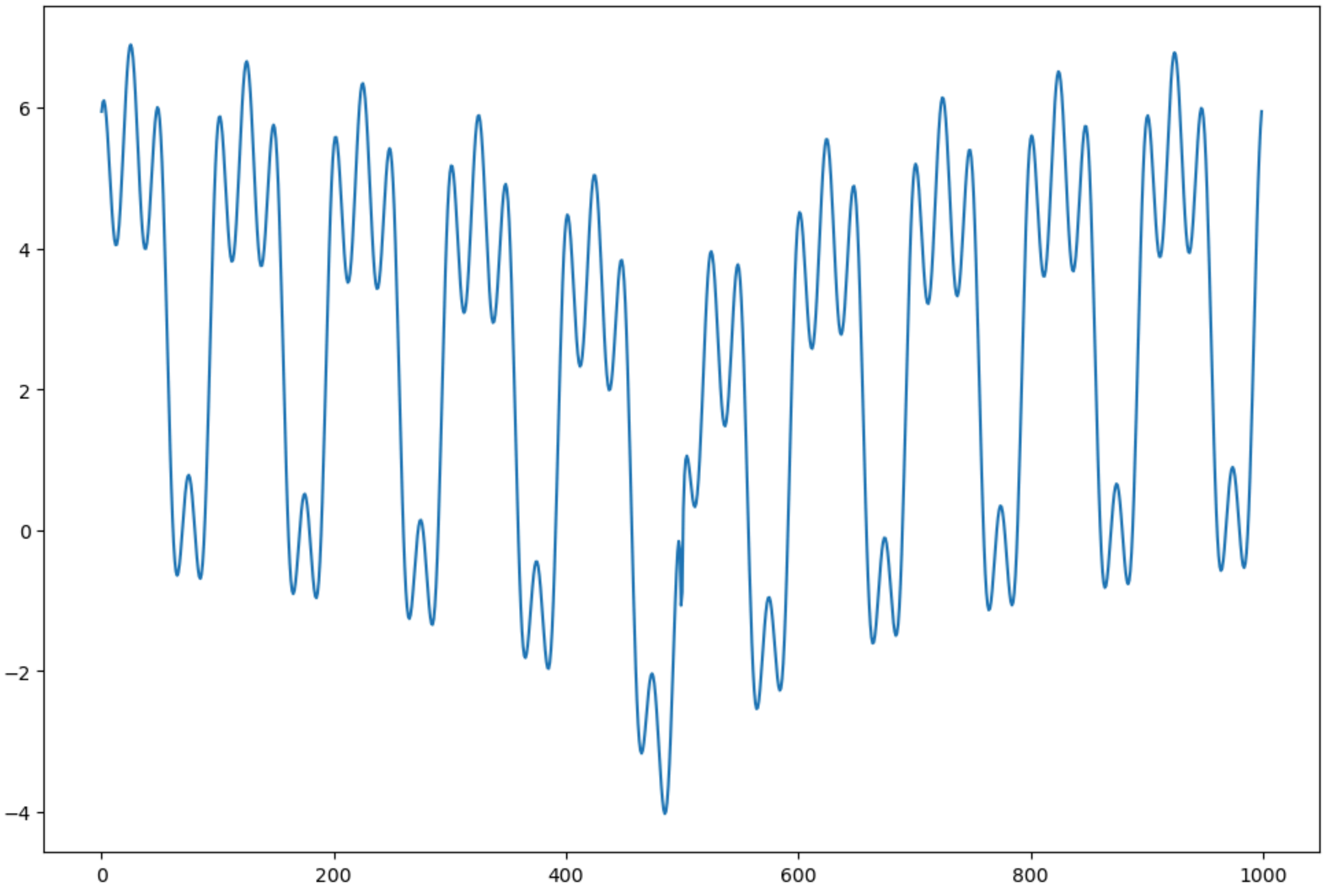
Szükséges Importok

In [ ]:
```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, Flatten, Conv1D, MaxPooling1D, LSTM
from tensorflow.keras.optimizers import SGD
import matplotlib.pyplot as plt
```

1. Adatgyűjtés

In [ ]:
```
# Adatok elkészítése
lp      = np.linspace(-10*np.pi,10*np.pi,1000)
X       = np.sin(lp)*3+np.cos(lp*2)+np.sin(np.pi/2+lp*4)*1.5+np.log(np.abs(lp))

plt.figure(figsize=(12,8))
plt.plot(X)
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x1d000e34ac0>]
```



2-3. Adatok feltérképezése és preprocesszálása

Itt nem lényeges.

4. Modell választása

In [ ]:

```
# 1D konvolúció alapú háló
def make_1d_convnet(window_size, filter_length, nb_input_series=1, nb_outputs=1, nb_filter=4)
    model = Sequential()
    model.add(Conv1D(filters=nb_filter, kernel_size=filter_length, activation='relu', input_s
    model.add(MaxPooling1D())
    model.add(Conv1D(filters=nb_filter, kernel_size=filter_length, activation='relu'))
    model.add(MaxPooling1D())
    model.add(Flatten())
    model.add(Dense(nb_outputs, activation='linear'))
    model.compile(loss='mse', optimizer='adam', metrics=['mae'])
    return model

# LSTM alapú háló
def make_LSTM(window_size, nb_input_series=1, nb_outputs=1):
    model = Sequential()
    model.add(LSTM(units = 100, activation='relu', input_shape=(window_size, nb_input_series)
    model.add(Dense(nb_outputs, activation='linear'))
    model.compile(loss='mse', optimizer='adam', metrics=['mae'])
    return model
```

In [ ]:

```
def make_timeseries_instances(timeseries, window_size):
    timeseries = np.asarray(timeseries)
    assert 0 < window_size < timeseries.shape[0] , "Out of range 0 < {} < {} ".format(window_
    X = np.atleast_3d(np.array([timeseries[start:start + window_size] for start in range(0, t
    Y = timeseries[window_size:]
    return X, Y
```

In [ ]:

```
def evaluate_timeseries(timeseries, window_size, valid_split=0.15, test_split=0.15):
    filter_length = 5
    nb_filter = 4
    timeseries = np.atleast_2d(timeseries)
    if timeseries.shape[0] == 1:
        timeseries = timeseries.T       # 1D vektor -> 2D matrix
    nb_samples, nb_series = timeseries.shape
    model = make_1d_convnet(window_size=window_size, filter_length=filter_length, nb_input_se
    model.summary()
    X, Y = make_timeseries_instances(timeseries, window_size)

    valid_size = int(nb_samples*(1-test_split-valid_split))
    test_size = int(nb_samples*(1-test_split))
    X_train, Y_train = X[:valid_size], Y[:valid_size]
    X_valid, Y_valid = X[valid_size:test_size], Y[valid_size:test_size]
    X_test, Y_test   = X[test_size:], Y[test_size:]

    model.fit(X_train, Y_train, epochs=50, batch_size=16, validation_data=(X_valid, Y_valid),

    preds = model.predict(X_test)

    return Y_test, preds
```

5. Modell illesztése

In [ ]:

```
window_size = 20
targets, preds = evaluate_timeseries(X, window_size)
```

```
c:\Users\mbenc\anaconda3\lib\site-packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(
```

Model: "sequential_16"

Layer (type)	**Output Shape**	**Param #**
conv1d_2 (Conv1D)	(None, 16, 4)	24
max_pooling1d_2 (MaxPooling1D)	(None, 8, 4)	0
conv1d_3 (Conv1D)	(None, 4, 4)	84
max_pooling1d_3 (MaxPooling1D)	(None, 2, 4)	0
flatten_1 (Flatten)	(None, 8)	0
dense_4 (Dense)	(None, 1)	9

Total params: 117 (468.00 B)

Trainable params: 117 (468.00 B)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
44/44 - 2s - 38ms/step - loss: 13.4507 - mae: 3.1027 - val_loss: 16.0147 - val_mae: 3.4211
Epoch 2/50
44/44 - 0s - 3ms/step - loss: 11.5044 - mae: 2.8857 - val_loss: 14.4014 - val_mae: 3.2520
Epoch 3/50
44/44 - 0s - 3ms/step - loss: 10.0514 - mae: 2.7086 - val_loss: 10.7919 - val_mae: 2.8563
Epoch 4/50
44/44 - 0s - 6ms/step - loss: 6.6575 - mae: 2.1615 - val_loss: 5.7871 - val_mae: 1.9623
Epoch 5/50
44/44 - 0s - 4ms/step - loss: 4.7886 - mae: 1.7409 - val_loss: 4.7022 - val_mae: 1.7603
Epoch 6/50
44/44 - 0s - 3ms/step - loss: 3.3546 - mae: 1.4691 - val_loss: 2.7665 - val_mae: 1.3490
Epoch 7/50
44/44 - 0s - 3ms/step - loss: 2.3999 - mae: 1.2488 - val_loss: 2.0262 - val_mae: 1.1426
Epoch 8/50
44/44 - 0s - 3ms/step - loss: 1.9132 - mae: 1.1117 - val_loss: 1.6111 - val_mae: 1.0087
Epoch 9/50
44/44 - 0s - 3ms/step - loss: 1.5687 - mae: 1.0043 - val_loss: 1.2820 - val_mae: 0.8941
Epoch 10/50
44/44 - 0s - 4ms/step - loss: 1.3047 - mae: 0.9058 - val_loss: 1.0414 - val_mae: 0.7949
Epoch 11/50
44/44 - 0s - 5ms/step - loss: 1.1081 - mae: 0.8310 - val_loss: 0.8691 - val_mae: 0.7204
Epoch 12/50
44/44 - 0s - 4ms/step - loss: 0.9699 - mae: 0.7638 - val_loss: 0.7376 - val_mae: 0.6560
Epoch 13/50
44/44 - 0s - 3ms/step - loss: 0.8504 - mae: 0.7164 - val_loss: 0.6725 - val_mae: 0.6229
Epoch 14/50
44/44 - 0s - 4ms/step - loss: 0.7780 - mae: 0.6861 - val_loss: 0.5925 - val_mae: 0.5917
Epoch 15/50
44/44 - 0s - 4ms/step - loss: 0.7163 - mae: 0.6569 - val_loss: 0.5442 - val_mae: 0.5706
Epoch 16/50
44/44 - 0s - 4ms/step - loss: 0.6733 - mae: 0.6409 - val_loss: 0.5134 - val_mae: 0.5602
Epoch 17/50
44/44 - 0s - 3ms/step - loss: 0.6476 - mae: 0.6319 - val_loss: 0.4973 - val_mae: 0.5616
Epoch 18/50
44/44 - 0s - 3ms/step - loss: 0.6080 - mae: 0.6105 - val_loss: 0.4629 - val_mae: 0.5439
Epoch 19/50
44/44 - 0s - 4ms/step - loss: 0.5774 - mae: 0.5930 - val_loss: 0.4415 - val_mae: 0.5350
Epoch 20/50
44/44 - 0s - 3ms/step - loss: 0.5531 - mae: 0.5828 - val_loss: 0.4195 - val_mae: 0.5183
Epoch 21/50
44/44 - 0s - 4ms/step - loss: 0.5298 - mae: 0.5760 - val_loss: 0.4153 - val_mae: 0.5217
Epoch 22/50
44/44 - 0s - 3ms/step - loss: 0.5098 - mae: 0.5554 - val_loss: 0.3979 - val_mae: 0.5106
Epoch 23/50
44/44 - 0s - 3ms/step - loss: 0.4816 - mae: 0.5433 - val_loss: 0.3818 - val_mae: 0.4993
Epoch 24/50
44/44 - 0s - 3ms/step - loss: 0.4633 - mae: 0.5310 - val_loss: 0.3591 - val_mae: 0.4755
Epoch 25/50
44/44 - 0s - 3ms/step - loss: 0.4401 - mae: 0.5116 - val_loss: 0.3382 - val_mae: 0.4611
Epoch 26/50
44/44 - 0s - 4ms/step - loss: 0.4374 - mae: 0.5148 - val_loss: 0.3290 - val_mae: 0.4589
Epoch 27/50
44/44 - 0s - 4ms/step - loss: 0.4173 - mae: 0.5053 - val_loss: 0.3113 - val_mae: 0.4438
Epoch 28/50
44/44 - 0s - 3ms/step - loss: 0.3977 - mae: 0.4930 - val_loss: 0.2956 - val_mae: 0.4250
Epoch 29/50
44/44 - 0s - 4ms/step - loss: 0.3782 - mae: 0.4764 - val_loss: 0.2845 - val_mae: 0.4129
Epoch 30/50
44/44 - 0s - 3ms/step - loss: 0.3665 - mae: 0.4718 - val_loss: 0.2710 - val_mae: 0.4106
Epoch 31/50
44/44 - 0s - 3ms/step - loss: 0.3511 - mae: 0.4535 - val_loss: 0.2990 - val_mae: 0.4362
Epoch 32/50
44/44 - 0s - 3ms/step - loss: 0.3425 - mae: 0.4463 - val_loss: 0.2654 - val_mae: 0.4099
Epoch 33/50
44/44 - 0s - 4ms/step - loss: 0.3195 - mae: 0.4308 - val_loss: 0.2429 - val_mae: 0.3894
Epoch 34/50
44/44 - 0s - 4ms/step - loss: 0.3033 - mae: 0.4203 - val_loss: 0.2242 - val_mae: 0.3701
Epoch 35/50
```

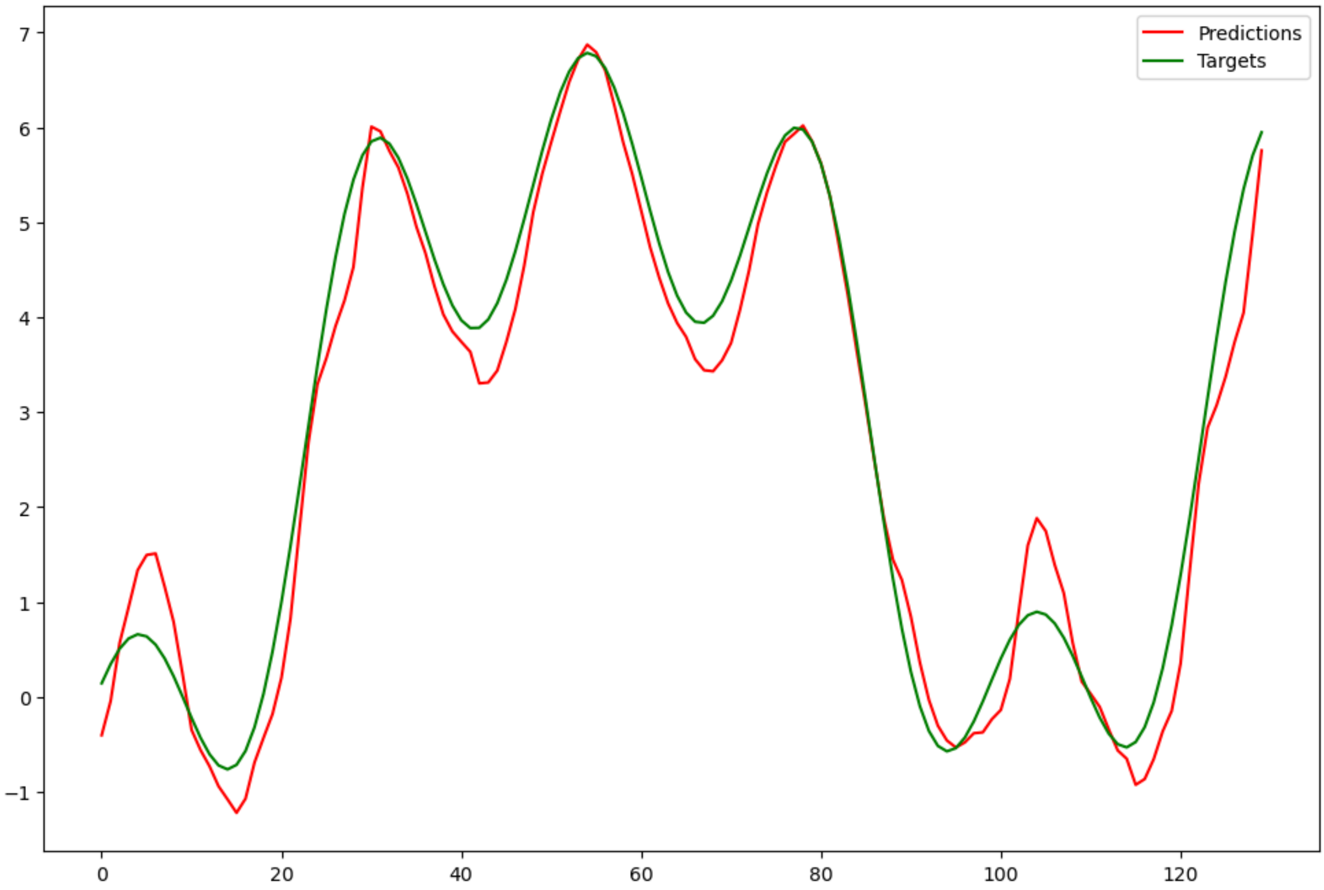
```
44/44 - 0s - 3ms/step - loss: 0.2866 - mae: 0.4073 - val_loss: 0.2009 - val_mae: 0.3438
Epoch 36/50
44/44 - 0s - 3ms/step - loss: 0.2730 - mae: 0.3904 - val_loss: 0.1945 - val_mae: 0.3402
Epoch 37/50
44/44 - 0s - 3ms/step - loss: 0.2622 - mae: 0.3877 - val_loss: 0.2012 - val_mae: 0.3441
Epoch 38/50
44/44 - 0s - 3ms/step - loss: 0.2566 - mae: 0.3817 - val_loss: 0.1743 - val_mae: 0.3218
Epoch 39/50
44/44 - 0s - 3ms/step - loss: 0.2465 - mae: 0.3718 - val_loss: 0.1760 - val_mae: 0.3208
Epoch 40/50
44/44 - 0s - 3ms/step - loss: 0.2366 - mae: 0.3624 - val_loss: 0.1639 - val_mae: 0.3105
Epoch 41/50
44/44 - 0s - 3ms/step - loss: 0.2309 - mae: 0.3572 - val_loss: 0.1506 - val_mae: 0.2946
Epoch 42/50
44/44 - 0s - 4ms/step - loss: 0.2215 - mae: 0.3495 - val_loss: 0.1459 - val_mae: 0.2907
Epoch 43/50
44/44 - 0s - 3ms/step - loss: 0.2152 - mae: 0.3432 - val_loss: 0.1411 - val_mae: 0.2887
Epoch 44/50
44/44 - 0s - 3ms/step - loss: 0.2087 - mae: 0.3385 - val_loss: 0.1352 - val_mae: 0.2764
Epoch 45/50
44/44 - 0s - 4ms/step - loss: 0.2027 - mae: 0.3302 - val_loss: 0.1318 - val_mae: 0.2780
Epoch 46/50
44/44 - 0s - 4ms/step - loss: 0.1977 - mae: 0.3261 - val_loss: 0.1251 - val_mae: 0.2691
Epoch 47/50
44/44 - 0s - 4ms/step - loss: 0.1945 - mae: 0.3226 - val_loss: 0.1221 - val_mae: 0.2668
Epoch 48/50
44/44 - 0s - 4ms/step - loss: 0.1895 - mae: 0.3158 - val_loss: 0.1155 - val_mae: 0.2596
Epoch 49/50
44/44 - 0s - 3ms/step - loss: 0.1840 - mae: 0.3105 - val_loss: 0.1126 - val_mae: 0.2585
Epoch 50/50
44/44 - 0s - 4ms/step - loss: 0.1852 - mae: 0.3122 - val_loss: 0.1225 - val_mae: 0.2728
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

In [ ]:

```
plt.figure(figsize=(12,8))
plt.plot(preds, color='r', label="Predictions")
plt.plot(targets, color='g', label="Targets")
plt.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x1d000ae4fa0>
```



10. hét / II. Autoencoder alapú zajcsökkentés

Szükséges Importok

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf

from tensorflow import keras

from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
```

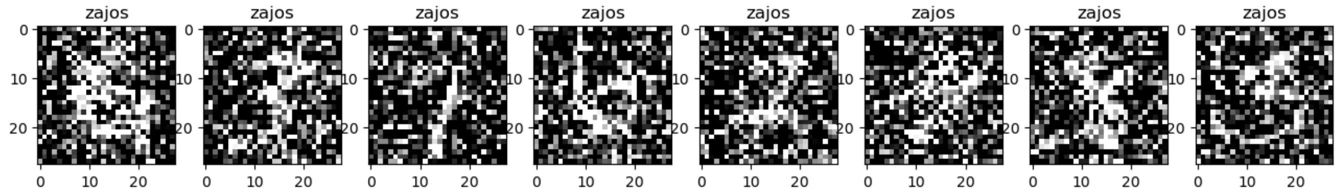
1. Adatgyűjtés

In [ ]:
```
# tanítóadatok, mint az előző példában

(x_train, _) ,(x_test, _) = mnist.load_data()

print(x_train.shape, x_test.shape)

# teszt részt kettéválasztjuk még, hogy legyen külön validációs halmaz
from sklearn.model_selection import train_test_split
[x_test, x_valid] = train_test_split(x_test, test_size=0.5)

x_train = x_train / 255
x_valid = x_valid / 255
x_test = x_test / 255

print(x_train.shape, x_valid.shape, x_test.shape)
```

```
(60000, 28, 28) (10000, 28, 28)
(60000, 28, 28) (5000, 28, 28) (5000, 28, 28)
```

2-3. Adatfeltérképezés és data preprocessing

In [ ]:
```
# Zaj hozzáadása
noise_factor = 0.7
x_train_noisy = x_train + noise_factor * np.random.normal(size=x_train.shape)
x_valid_noisy = x_valid + noise_factor * np.random.normal(size=x_valid.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(size=x_test.shape)

# Klippelés 0 és 1 közé
x_train_noisy = np.clip(x_train_noisy, 0, 1)
x_valid_noisy = np.clip(x_valid_noisy, 0, 1)
x_test_noisy = np.clip(x_test_noisy, 0, 1)

# Vizualizáció
n = 8
plt.figure(figsize=(16, 4))
for i in range(n):
  # eredeti
  ax = plt.subplot(1, n, i + 1)
  ax.set_title('zajos')
  plt.imshow(x_test_noisy[i].reshape(28,28))
  plt.gray()
```



4. Modell választás

In [ ]:
```
autoencoder = keras.models.Sequential([
    keras.layers.Reshape([28, 28, 1], input_shape=[28, 28]),
    keras.layers.Conv2D(16, kernel_size=(3, 3), padding="same", activation="relu"),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Conv2D(32, kernel_size=(3, 3), padding="same", activation="relu"),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Conv2D(64, kernel_size=(3, 3), padding="same", activation="relu"),
    keras.layers.MaxPool2D(pool_size=2),

    keras.layers.Conv2DTranspose(32, kernel_size=(3, 3), strides=2, padding="valid", activati
                                 input_shape=[3, 3, 64]),
    keras.layers.Conv2DTranspose(16, kernel_size=(3, 3), strides=2, padding="same", activatio
    keras.layers.Conv2DTranspose(1, kernel_size=(3, 3), strides=2, padding="same"),
    keras.layers.Reshape([28, 28])
])

autoencoder.summary()

autoencoder.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

```
c:\Users\mbenc\anaconda3\lib\site-packages\keras\src\layers\reshaping\reshape.py:39: UserWarn
ing: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential mode
ls, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
c:\Users\mbenc\anaconda3\lib\site-packages\keras\src\layers\convolutional\base_conv_transpos
e.py:94: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When usin
g Sequential models, prefer using an `Input(shape)` object as the first layer in the model in
stead.
  super().__init__(
```

Model: "sequential_23"

Layer (type)	Output Shape	Param #
reshape_3 (Reshape)	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 28, 28, 16)	160
max_pooling2d (MaxPooling2D)	(None, 14, 14, 16)	0
conv2d_1 (Conv2D)	(None, 14, 14, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 32)	0
conv2d_2 (Conv2D)	(None, 7, 7, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 3, 3, 64)	0
conv2d_transpose_15 (Conv2DTranspose)	(None, 7, 7, 32)	18,464
conv2d_transpose_16 (Conv2DTranspose)	(None, 14, 14, 16)	4,624
conv2d_transpose_17 (Conv2DTranspose)	(None, 28, 28, 1)	145
reshape_4 (Reshape)	(None, 28, 28)	0

Total params: 46,529 (181.75 KB)

Trainable params: 46,529 (181.75 KB)

Non-trainable params: 0 (0.00 B)

5. Modell illesztés

In [ ]:
```
autoencoder.fit(x_train_noisy, x_train,
                epochs = 10,
                batch_size = 128,
                shuffle = True,
                validation_data = (x_valid_noisy, x_valid))
```

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 17s 32ms/step - loss: 0.0614 - val_loss: 0.0314
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 15s 31ms/step - loss: 0.0301 - val_loss: 0.0269
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 29ms/step - loss: 0.0266 - val_loss: 0.0249
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 14s 29ms/step - loss: 0.0249 - val_loss: 0.0237
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 16s 33ms/step - loss: 0.0237 - val_loss: 0.0230
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 15s 33ms/step - loss: 0.0230 - val_loss: 0.0224
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 15s 32ms/step - loss: 0.0224 - val_loss: 0.0221
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 16s 34ms/step - loss: 0.0219 - val_loss: 0.0215
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 16s 33ms/step - loss: 0.0215 - val_loss: 0.0214
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 15s 32ms/step - loss: 0.0214 - val_loss: 0.0210
```

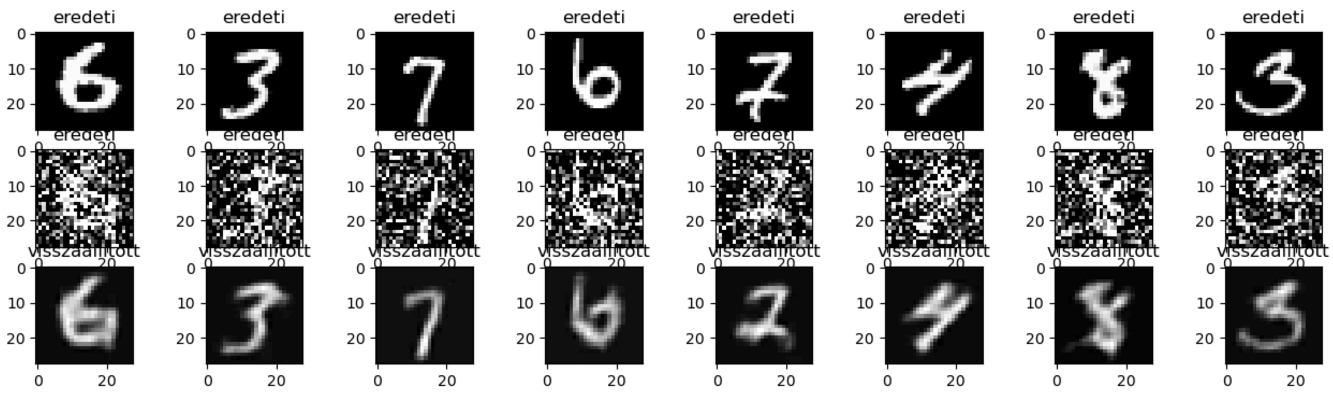
Out[ ]:
```
<keras.src.callbacks.history.History at 0x1d051d87e80>
```

In [ ]:
```
decoded_imgs = autoencoder.predict(x_test_noisy)
```

```
157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
```

6. Kiértékelés

In [ ]:
```python
n = 8
plt.figure(figsize=(16, 4))
for i in range(n):
  # eredeti
  ax = plt.subplot(3, n, i + 1)
  ax.set_title('eredeti')
  plt.imshow(x_test[i].reshape(28,28))
  plt.gray()

  # zajos
  ax = plt.subplot(3, n, i + 1 + n)
  ax.set_title('eredeti')
  plt.imshow(x_test_noisy[i].reshape(28,28))
  plt.gray()

  # visszaállított
  ax = plt.subplot(3, n, i + 1 + 2 * n)
  ax.set_title('visszaállított')
  plt.imshow(decoded_imgs[i].reshape(28,28))
  plt.gray()
plt.show()
```

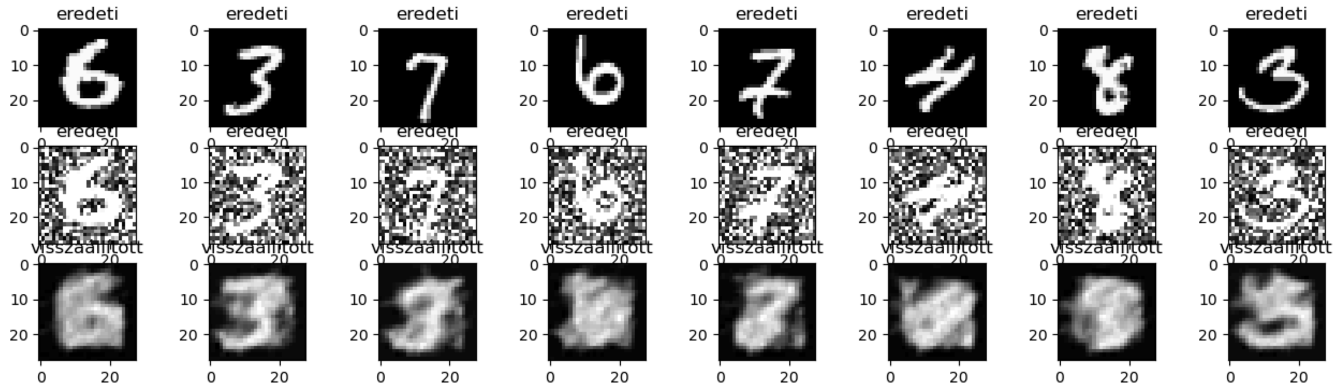


7. Modell finomítása

In [ ]:
```python
noise_factor = 0.4
x_test_noisy = x_test + noise_factor * np.random.lognormal(0, 1, size=x_test.shape)
x_test_noisy = np.clip(x_test_noisy, 0, 1)

decoded_imgs = autoencoder.predict(x_test_noisy)

# eredeti, zajos és visszaállított képek kirajzolása
n = 8
plt.figure(figsize=(16, 4))
for i in range(n):
  # eredeti
  ax = plt.subplot(3, n, i + 1)
  ax.set_title('eredeti')
  plt.imshow(x_test[i].reshape(28,28))
  plt.gray()

  # zajos
  ax = plt.subplot(3, n, i + 1 + n)
  ax.set_title('eredeti')
  plt.imshow(x_test_noisy[i].reshape(28,28))
  plt.gray()

  # visszaállított
  ax = plt.subplot(3, n, i + 1 + 2 * n)
  ax.set_title('visszaállított')
```

```
    plt.imshow(decoded_imgs[i].reshape(28,28))
    plt.gray()
plt.show()
```

```
157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
```



10. hét / III. Képszegmentáció haladó konvolúciós háló segítségével

Szükséges könyvtárak

1. **TensorFlow: Datasets** - Olyan könyvtár, amelyben több, előre összeállított és feldolgozott adatbázis található
2. **TensorFlow: Examples** - TensorFlow oktatóanyagok és adathalmazok tárolására létrehozott könyvtár
3. **Pydot** - Neurális hálók vizaulizációjához szükséges könyvtár

In [ ]:
```
%pip install tensorflow-datasets
%pip install tensorflow-examples
%pip install pydot
```

In [ ]:
```
#%pip install -q git+https://github.com/tensorflow/examples.git
```

Szükséges importok

In [ ]:
```
# Deep Learning keretrendszer
import tensorflow as tf
import tensorflow_datasets as tfds

# Adathalmaz
from tensorflow_examples.models.pix2pix import pix2pix

# Megjelenítés
from IPython.display import clear_output
import matplotlib.pyplot as plt
```

1. Adatgyűjtés

In [ ]:
```
dataset, info = tfds.load('oxford_iiit_pet:3.*.*', with_info=True)
```

2. Adatfeltérképezés + 3. Adat előkészítés

In [ ]:
```
# Vizuális adatot noramlizáljuk 0 és 1 közé
def normalize(input_image, input_mask):
    input_image = tf.cast(input_image, tf.float32) / 255.0
    input_mask -= 1
```

```python
    return input_image, input_mask

# Az adatok betöltése
def load_image(datapoint):
  # Érdemes átskálázni a bemeneti kép méretét
  input_image = tf.image.resize(datapoint['image'], (128, 128))
  input_mask = tf.image.resize(
    datapoint['segmentation_mask'],
    (128, 128),

    # Azért, hogy ne 'hígguljon' fel az adat, nearest_neighbor algoritmust használunk
    method = tf.image.ResizeMethod.NEAREST_NEIGHBOR,
  )

  # A betöltés végén a képet normalizáljuk
  input_image, input_mask = normalize(input_image, input_mask)

  # Visszatérünk a preprocesszált képekkel
  return input_image, input_mask
```

In [ ]:

```python
# Betöltjük az adatot
train_images = dataset['train'].map(load_image, num_parallel_calls=tf.data.AUTOTUNE)
test_images = dataset['test'].map(load_image, num_parallel_calls=tf.data.AUTOTUNE)
```

In [ ]:

```python
# Augmentáció
class Augment(tf.keras.layers.Layer):
  def __init__(self, seed=42):
    super().__init__()
    # Ha ugyanazzal a random seeddel dolgozunk, akkor ugyanazok fognak megfordulni
    self.augment_inputs = tf.keras.layers.RandomFlip(mode="horizontal", seed=seed)
    self.augment_labels = tf.keras.layers.RandomFlip(mode="horizontal", seed=seed)

  def call(self, inputs, labels):
    inputs = self.augment_inputs(inputs)
    labels = self.augment_labels(labels)
    return inputs, labels
```

In [ ]:

```python
# Előkészítés a tanításhoz
TRAIN_LENGTH = info.splits['train'].num_examples
BATCH_SIZE = 64
BUFFER_SIZE = 1000
STEPS_PER_EPOCH = TRAIN_LENGTH // BATCH_SIZE

train_batches = (
    train_images
    .cache()
    .shuffle(BUFFER_SIZE)
    .batch(BATCH_SIZE)
    .repeat()
    .map(Augment())
    .prefetch(buffer_size=tf.data.AUTOTUNE))

test_batches = test_images.batch(BATCH_SIZE)
```

In [ ]:

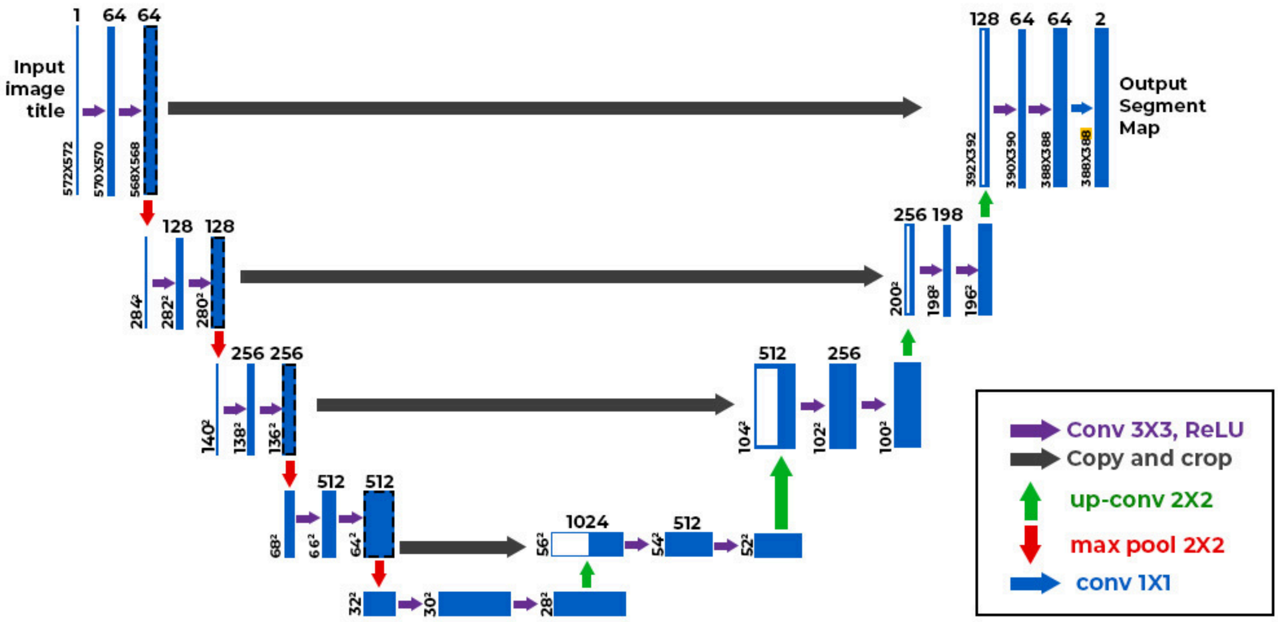
```python
# Megjelenítés
def display(display_list):
  plt.figure(figsize=(15, 15))

  title = ['Input Image', 'True Mask', 'Predicted Mask']

  for i in range(len(display_list)):
    plt.subplot(1, len(display_list), i+1)
    plt.title(title[i])
    plt.imshow(tf.keras.utils.array_to_img(display_list[i]))
    plt.axis('off')
  plt.show()
```

```
for images, masks in train_batches.take(1):
  sample_image, sample_mask = images[0], masks[0]
  display([sample_image, sample_mask])
```

Input Image

True Mask

4. Modellválasztás




```
In [ ]: base_model = tf.keras.applications.MobileNetV2(input_shape=[128, 128, 3], include_top=False)

# "Legózás"
layer_names = [
    'block_1_expand_relu',   # 64x64
    'block_3_expand_relu',   # 32x32
    'block_6_expand_relu',   # 16x16
    'block_13_expand_relu',  # 8x8
    'block_16_project',      # 4x4
]
base_model_outputs = [base_model.get_layer(name).output for name in layer_names]

# Contracting (Feature Extractor) path létrehozása
down_stack = tf.keras.Model(inputs=base_model.input, outputs=base_model_outputs)

down_stack.trainable = False

# Expansive path létrehozása
up_stack = [
```

```python
    pix2pix.upsample(512, 3),  # 4x4 -> 8x8
    pix2pix.upsample(256, 3),  # 8x8 -> 16x16
    pix2pix.upsample(128, 3),  # 16x16 -> 32x32
    pix2pix.upsample(64, 3),   # 32x32 -> 64x64
]
```

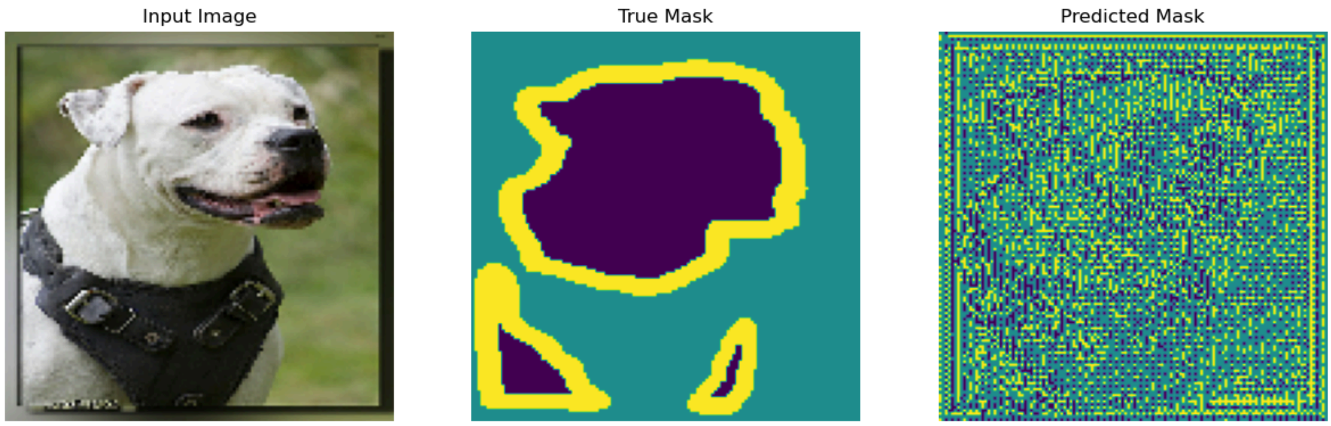
In [ ]:
```python
def unet_model(output_channels:int):
  inputs = tf.keras.layers.Input(shape=[128, 128, 3])

  # Downsampling
  skips = down_stack(inputs)
  x = skips[-1]
  skips = reversed(skips[:-1])

  # Upsampling
  for up, skip in zip(up_stack, skips):
    x = up(x)
    concat = tf.keras.layers.Concatenate()
    x = concat([x, skip])

  # Utolsó réteg
  last = tf.keras.layers.Conv2DTranspose(
      filters=output_channels, kernel_size=3, strides=2,
      padding='same')  #64x64 -> 128x128

  x = last(x)

  return tf.keras.Model(inputs=inputs, outputs=x)
```

In [ ]:
```python
OUTPUT_CLASSES = 3

model = unet_model(output_channels=OUTPUT_CLASSES)
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [ ]:
```python
def create_mask(pred_mask):
  pred_mask = tf.math.argmax(pred_mask, axis=-1)
  pred_mask = pred_mask[..., tf.newaxis]
  return pred_mask[0]

def show_predictions(dataset=None, num=1):
  if dataset:
    for image, mask in dataset.take(num):
      pred_mask = model.predict(image)
      display([image[0], mask[0], create_mask(pred_mask)])
  else:
    display([sample_image, sample_mask,
             create_mask(model.predict(sample_image[tf.newaxis, ...]))])
```

In [ ]:
```python
show_predictions()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```



In [ ]:

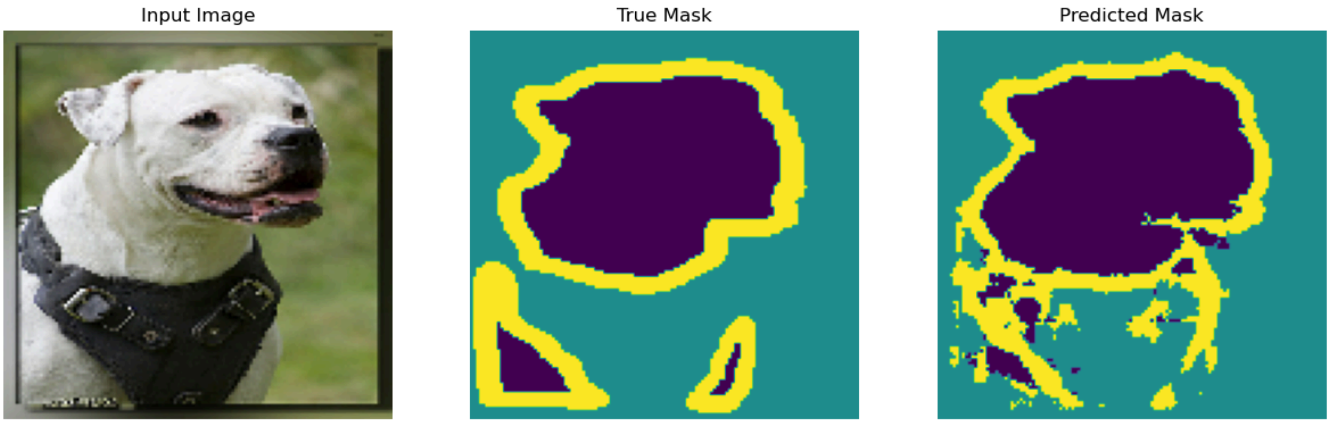
```
class DisplayCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs=None):
    clear_output(wait=True)
    show_predictions()
    print ('\nSample Prediction after epoch {}\n'.format(epoch+1))
```

5. Modell illesztése

In [ ]:

```
EPOCHS = 40
VAL_SUBSPLITS = 5
VALIDATION_STEPS = info.splits['test'].num_examples//BATCH_SIZE//VAL_SUBSPLITS

model_history = model.fit(train_batches,
                           epochs=EPOCHS,
                          steps_per_epoch=STEPS_PER_EPOCH,
                          validation_steps=VALIDATION_STEPS,
                          validation_data=test_batches,
                          callbacks=[DisplayCallback()])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
```




```
Sample Prediction after epoch 13

57/57 ━━━━━━━━━━━━━━━━━━━━ 206s 4s/step - accuracy: 0.9172 - loss: 0.2059 - val_accuracy: 0.9
037 - val_loss: 0.2581
Epoch 14/40
26/57 ━━━━━━━━━━━━━━━━━━━━ 1:34 3s/step - accuracy: 0.9156 - loss: 0.2094
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[56], line 5
      2 VAL_SUBSPLITS = 5
      3 VALIDATION_STEPS = info.splits['test'].num_examples//BATCH_SIZE//VAL_SUBSPLITS
----> 5 model_history = model.fit(train_batches,
      6                            epochs=EPOCHS,
      7                           steps_per_epoch=STEPS_PER_EPOCH,
      8                           validation_steps=VALIDATION_STEPS,
      9                           validation_data=test_batches,
     10                           callbacks=[DisplayCallback()])

File c:\Users\mbenc\anaconda3\lib\site-packages\keras\src\utils\traceback_utils.py:117, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    115 filtered_tb = None
    116 try:
--> 117     return fn(*args, **kwargs)
    118 except Exception as e:
    119     filtered_tb = _process_traceback_frames(e.__traceback__)

File c:\Users\mbenc\anaconda3\lib\site-packages\keras\src\backend\tensorflow\trainer.py:329, in TensorFlowTrainer.fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, validation_batch_size, validation_freq)
    327 for step, iterator in epoch_iterator.enumerate_epoch():
    328     callbacks.on_train_batch_begin(step)
--> 329     logs = self.train_function(iterator)
    330     callbacks.on_train_batch_end(
    331         step, self._pythonify_logs(logs)
    332     )
    333     if self.stop_training:

File c:\Users\mbenc\anaconda3\lib\site-packages\tensorflow\python\util\traceback_utils.py:150, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    148 filtered_tb = None
    149 try:
--> 150   return fn(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)

File c:\Users\mbenc\anaconda3\lib\site-packages\tensorflow\python\eager\polymorphic_function\polymorphic_function.py:833, in Function.__call__(self, *args, **kwds)
    830 compiler = "xla" if self._jit_compile else "nonXla"
    832 with OptionalXlaContext(self._jit_compile):
--> 833   result = self._call(*args, **kwds)
    835 new_tracing_count = self.experimental_get_tracing_count()
    836 without_tracing = (tracing_count == new_tracing_count)

File c:\Users\mbenc\anaconda3\lib\site-packages\tensorflow\python\eager\polymorphic_function\polymorphic_function.py:878, in Function._call(self, *args, **kwds)
    875 self._lock.release()
    876 # In this case we have not created variables on the first call. So we can
    877 # run the first trace but we should fail if variables are created.
--> 878 results = tracing_compilation.call_function(
    879     args, kwds, self._variable_creation_config
    880 )
    881 if self._created_variables:
    882   raise ValueError("Creating variables on a non-first call to a function"
    883                    " decorated with tf.function.")

File c:\Users\mbenc\anaconda3\lib\site-packages\tensorflow\python\eager\polymorphic_function\tracing_compilation.py:139, in call_function(args, kwargs, tracing_options)
    137 bound_args = function.function_type.bind(*args, **kwargs)
    138 flat_inputs = function.function_type.unpack_inputs(bound_args)
--> 139 return function._call_flat(  # pylint: disable=protected-access
    140     flat_inputs, captured_inputs=function.captured_inputs
    141 )
```

```
File c:\Users\mbenc\anaconda3\lib\site-packages\tensorflow\python\eager\polymorphic_function\concrete_function.py:1322, in ConcreteFunction._call_flat(self, tensor_inputs, captured_inputs)
   1318 possible_gradient_type = gradients_util.PossibleTapeGradientTypes(args)
   1319 if (possible_gradient_type == gradients_util.POSSIBLE_GRADIENT_TYPES_NONE
   1320     and executing_eagerly):
   1321   # No tape is watching; skip to running the function.
-> 1322   return self._inference_function.call_preflattened(args)
   1323 forward_backward = self._select_forward_and_backward_functions(
   1324     args,
   1325     possible_gradient_type,
   1326     executing_eagerly)
   1327 forward_function, args_with_tangents = forward_backward.forward()

File c:\Users\mbenc\anaconda3\lib\site-packages\tensorflow\python\eager\polymorphic_function\atomic_function.py:216, in AtomicFunction.call_preflattened(self, args)
    214 def call_preflattened(self, args: Sequence[core.Tensor]) -> Any:
    215   """Calls with flattened tensor inputs and returns the structured output."""
--> 216   flat_outputs = self.call_flat(*args)
    217   return self.function_type.pack_output(flat_outputs)

File c:\Users\mbenc\anaconda3\lib\site-packages\tensorflow\python\eager\polymorphic_function\atomic_function.py:251, in AtomicFunction.call_flat(self, *args)
    249 with record.stop_recording():
    250   if self._bound_context.executing_eagerly():
--> 251     outputs = self._bound_context.call_function(
    252         self.name,
    253         list(args),
    254         len(self.function_type.flat_outputs),
    255     )
    256   else:
    257     outputs = make_call_op_in_graph(
    258         self,
    259         list(args),
    260         self._bound_context.function_call_options.as_attrs(),
    261     )

File c:\Users\mbenc\anaconda3\lib\site-packages\tensorflow\python\eager\context.py:1500, in Context.call_function(self, name, tensor_inputs, num_outputs)
   1498 cancellation_context = cancellation.context()
   1499 if cancellation_context is None:
-> 1500   outputs = execute.execute(
   1501       name.decode("utf-8"),
   1502       num_outputs=num_outputs,
   1503       inputs=tensor_inputs,
   1504       attrs=attrs,
   1505       ctx=self,
   1506   )
   1507 else:
   1508   outputs = execute.execute_with_cancellation(
   1509       name.decode("utf-8"),
   1510       num_outputs=num_outputs,
   (...)
   1514       cancellation_manager=cancellation_context,
   1515   )

File c:\Users\mbenc\anaconda3\lib\site-packages\tensorflow\python\eager\execute.py:53, in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     51 try:
     52   ctx.ensure_initialized()
---> 53   tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     54                                       inputs, attrs, num_outputs)
     55 except core._NotOkStatusException as e:
     56   if name is not None:

KeyboardInterrupt:
```

6. Modell kiértékelése

In [ ]:

```
loss = model_history.history['loss']
val_loss = model_history.history['val_loss']

plt.figure()
plt.plot(model_history.epoch, loss, 'r', label='Training loss')
plt.plot(model_history.epoch, val_loss, 'bo', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss Value')
plt.ylim([0, 1])
plt.legend()
plt.show()
```

In [ ]:

```
show_predictions(test_batches, 3)
```