9. hét / Prológus

A mai órán a következőkről lesz szó:

- Elméleti bevezetés a Deep Learningbe (~ 20 perc)
- Regresszió mélytanulás segítségével: California Housing Prices (~ 30 perc)
- Klasszifikáció mélytanulás segítségével: MNIST (~ 30 perc)

9. hét / I. Elméleti bevezetés a Deep Learningbe

Ismétlés: Machine Learning feladatok megoldása

Jellemzően a gépi tanulás feladatai az alábbi lépésekből épülnek fel:

1. **Adatgyűjtés**, mérés: Az adattudományok nem fognak működni adatok nélkül! Fontos, hogy a mérések elvégzése előtt tegyünk becsléseket arra vonatkozóan, hogy mennyire sok adatra van szükségünk és mekkora paramétertérrel akarunk dolgozni! Minél több paramétert mérünk, annál több adatra van szükségünk! (A paramétertér dimenziójával exponenciálisan növekszik a szükséges adattér "térfogata".)

2. **Adatfeltérképezés**: alapvető - statisztikai eszközökkel történő - megismerkedés az adatbázisunkkal. Például ábrázoljuk az egyes attribútumok eloszlását, szórását, vagy kirajzoljuk a térbeli és időbeli folyamatokat.

3. ! **Adatok előkészítése**, data preprocessing: az adathalmaz előkészítése a modellek betanításának megfelelően. Ez maga a "kreatív agya" az adattudományoknak, amikor a nyers adatbázisból kiszűrjük a számunkra lényeges részeket. Az adat minőségi paramétereitől függően teljesen eltérő eszközöket igényel: például egy képet gyakran fekete-fehérre átállítunk, átméretezünk, megkeressük és kiemeljük rajta az éleket, hogy kizárólag a lényeges információtartalommal rendelkező adattal dolgozzon az algoritmusunk. Minél ügyesebben tudjuk elvégezni az adathalmaz előkészítését, annál több nehézségtől mentjük meg magunkat meg a tanítás során!

4. **Modellválasztás**: a tanító modell meghatározása. (*Megjegyzés: valójában a modellt ismernünk kell már a preprocessing előtt, elvégre a választott modellre fogjuk optimalizálni a nyers adathalmazt!*) Ez mindig azzal kezdődik, hogy definiáljuk a problémát (például klasszifikációt, vagy regressziót akarunk-e csinálni) és ennek megfelelően keresünk a szakirodalomban megfelelő megoldásokat. Ezek alapján összeállítjuk a modellünket.

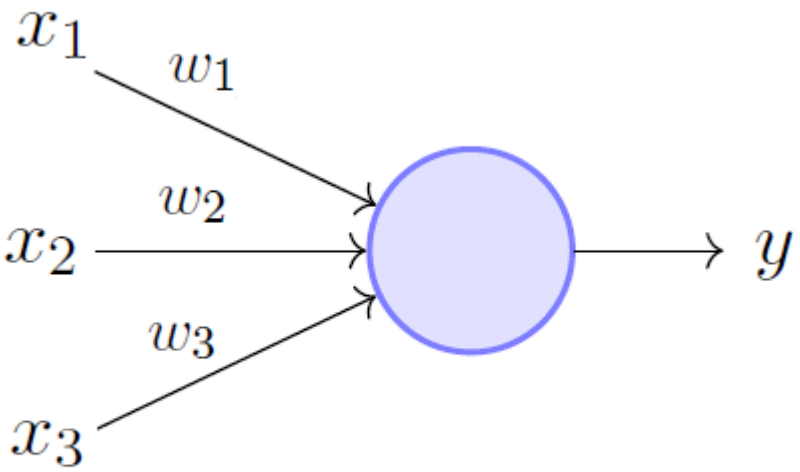
5. **Modell illesztés**, tanítás: a választott modell betanítása.

6. **Kiértékelés**: a tanító folyamat kiértékelése megadott teljesítménymetrikák (például *precision*, *recall*, vagy *accuracy*) alapján. Érdemes ezek változását is ábrázolni a tanítás során.

7. **Modell paramétereinek finomhangolása**: az előfeldolgozás, vagy a választott modellünk módosítása a teljesítménymetrikák szerint; illetve a tanítás során fellépő rendellenességek és anomáliák feltérképezése. Gyakran előfordul, hogy ugyan működik is a modellünk, de esetleg nem elég pontos, nem elég megbízható, nem tud eléggé általánosítani, túltanult. Ilyenkor az első négy lépésen visszafelé érdemes elkezdeni végighaladni és onnan újrakezdve optimalizálni a modellt.

8. **Alkalmazás**: az előre meghatározott célnak megfelelő modell telepítése.

A perceptron modell



Perceptron Model (Minsky-Papert in 1969)

9. hét / II. Regresszió: California Housing Prices

Az alábbiakban meg fogunk ismerkedni a neurális hálók egy klasszikus "Hello World!" feladatával.

Adott egy adathalmazunk, amely a California állambeli házak árát tartalmazza 8 paraméter függvényében. Feladatunk az, hogy készítsünk egy olyan matematikai modellt, amely képes predikciókat tenni arra vonatkozóan, hogy az adatbázisba egy újonnan bekerülő ingatlant mennyi pénzbe kerülhet.

Az adatbázis elérhető az alábbi linken.

Megoldás:

Szükséges framework telepítése:

In [ ]:
```
%pip install tensorflow
```

Szükséges importok

In [ ]:
```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import load_model
import copy
import pandas as pd
from tensorflow.keras.optimizers import SGD
from sklearn.preprocessing import StandardScaler

nyolc = 9
np.random.seed(nyolc)
```

1. Adatgyűjtés

Az adatgyűjtés problémáját a megfelelő szaktárgyak fedik le, mint például a Méréstechnika, Szenzor- és aktuátortechnika, Mikrovezérlők programozása, illetve egyes specializációkon ebben még részletesebben is elmélyülhetnek az érdeklődők.

Jelen kurzus nem fektet különösebb hangsúlyt arra, hogy a résztvevők a mérésadatgyűjtés és jelfeldolgozás eszközeiben jobban elmélyedjenek. Ennek elsődleges oka, hogy ezekről már külön-külön is egy teljes féléves anyagot össze lehetne állítani. A másik nagyon fontos dolog, hogy mérni (különösen akkora méretekben, mint ahogy azt az adattudományok igénylik! Emlékeztetőül: ha a vizsgált attribútumok száma n, akkor a szükséges adathalmaz mérete 2^n-nel arányos!) roppant **költséges** folyamat! Gondoljunk csak arra, hogy a pontos mérésekhez mennyire sok szenzorra és mindeneklőtt üzemórára van szükségünk.

Ezen megfontolásokkal a tárgy keretein belül mindig feltételezzük, hogy **a megoldani kívánt feladat szempontjából alkalmas adatbázis áll rendelkezésre!**

In [ ]:
```
from sklearn.datasets import fetch_california_housing

california_housing = fetch_california_housing(as_frame=True)
```

2. Adatfeltérképezés

Jellemzően az adatfeltérképezés során rengeteg ábrát és statisztikai eszközt érdemes ráereszteni az adatbázisunkra. Mivel jelen esetben egy nagyon jól ismert adathalmazról van szó, ettől most eltekinthetünk. Részletes leírást az alábbi linken kaphatunk.

Többek között azért fontos ez, hogy például ki tudjunk választani olyan attribútumokat, vagy változókat amelyek feltehetően teljesen irrelevánsak. Vagy éppen előre megjósolhassuk, hogy vajon mik lesznek a fontos paraméterek.

In [ ]:
```
california_housing.frame
```

Out[ ]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422
...	...	...	...	...	...	...	...	...	...
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.09	0.781
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.21	0.771
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.22	0.923
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.32	0.847
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.24	0.894

20640 rows × 9 columns

3. Data preprocessing

3.1 Az adathalmaz felbontása

Nagyon fontos lépés, hogy miután preprocesszáltuk a nyers adatot, utána osszuk fel három különböző részre. Ezek mind más funkciót fognak betölteni a *tanítás* során!

- **Training set**, vagy *tanító adathalmaz*: az adatnak az a része, amelyet ténylegesen fel fog dolgozni a neurális háló. Elsősorban ez az adathalmaz fogja definiálni a neurális háló főbb paramétereit.
- **Validation set**, vagy *validációs adathalmaz*: a kiindulási adathalmaz azon része, amelyet ugyan tanítás közben folyamatosan lát a neurális háló, de az ebből szerzett információt csak a modell finomhangolására és hiperparaméteroptimalizálására használja. Például a validációs adathalmaz segítségével könnyen ellenőrizhetővé válik, hogy hány *epoch* után érdemes leállítani a tanítást, hogy elkerüljük az overfitting-et! (A validációs adathalmazt definiálni nem kötelező, de az esetek túlnyomó részében több, mint érdemes!)
- **Test set**, vagy *tesztelési adathalmaz*: azon adathalmaz, amelyet a tanítás végeztével arra használunk, hogy a neurális háló metrikáit kiértékeljük. Ezt az adathalmazt tanítás során semmiképp sem láthatja a neurális háló, ugyanis az meghamisítaná a modell teljesítményét!

Ökölszabályként érdemes megjegyezni, hogy jellemzően 60%-20%-20%, 70%-15%-15% arányokat szoktunk választani! Viszont fontos emellett, hogy ezen halmazoknak nem is az aránya, sokkal inkább a homogenitása ami mérvadó! Nagyon fontos, hogy mindhárom halmazban minél ideálisabban legyen reprezentálva a teljes adathalmaz, különben sok anomáliával szembesülhetünk! *(Például fontos, hogy egy rendezett adathalmazt ne egymás utáni részekre osszuk fel, hanem véletlenszerűen válasszuk ki az egyes elemeket! Ha nem így teszünk, akkor könnyen előfordulhat, hogy tanítás során például közel 100%-os accuracy értéket kapunk, a tesztelés során pedig ettől jelentősen kevesebb érték jelenik meg.)*

3.2 Az adathalmaz standardizálása

Szigorúan az adathalmaz felosztása után érdemes egyenként a tanító, validációs és tesztelési adathalmazainkat standardizálni. Ez azt jelenti, hogy a $\mathbf{X}$ adathalmazunk elemeit **normalizál**juk, azaz úgy transzformáljuk, hogy az E **várható érték**e (jelen esetben: számtani közép, amit szokás még μ-vel is jelölni) **zérus**, illetve a σ **szórás**a **egységnyi** legyen! $\hat{\mathbf{X}} = \frac{\mathbf{X}-E}{\sigma}$ Erre azért van szükség, hogy egyrészt ne kelljen feleslegesen nagy számokkal dolgoznia a neurális hálónak, másrészt pedig azért, hogy könnyebben tudjon általánosítani a modell.

(Megjegyzés: miért nem mindegy, hogy az adathalmaz felbontása előtt, vagy után standardizáljuk az adatot? Melyik adathalmaznak és melyik aspektusa sérülne, ha fordítva járnánk el?)

In [ ]:
```
dataset = california_housing.frame.values

# Tanító-validációs-tesztelő adathalmazok méretének definiálása
test_split = 0.1
valid_split = 0.1

X = dataset[:,0:-1]
Y = dataset[:,-1]

# A megfelelő arányokban felosztjuk az egyes adathalmazokat
v_index = int(X.shape[0]*(1-valid_split-test_split))
t_index = int(X.shape[0]*(1-test_split))

# Tesztelő adat: 90%-tól végig
X_test = X[t_index:]
```

```
Y_test = Y[t_index:]

# Validációs adat: 80%-90%-ig
X_valid = X[v_index:t_index]
Y_valid = Y[v_index:t_index]

# Tanító adat: elejétől a 80%-ig
X = X[:v_index]
Y = Y[:v_index]

# Normalizáljuk az adatot (σ = 1, E = 0)
scaler = StandardScaler().fit(X)
X = scaler.transform(X)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
```

4. Modellválasztás

In [ ]:
```
# Callback függvények definiálása
patience=50
early_stopping=EarlyStopping(patience=patience, verbose=1)
checkpointer=ModelCheckpoint(filepath='weights.keras', save_best_only=True, verbose=1)
```

```
# Modell felépítése
model = Sequential()
model.add(Dense(units=200, input_dim=X.shape[1]))
model.add(Activation('sigmoid'))
model.add(Dropout(0.5))
model.add(Dense(units=1, activation='linear'))
model.summary()

# Model tanításának hiperparaméterei
sgd = SGD(learning_rate=1e-3, weight_decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mse', optimizer=sgd)
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 200)	1,800
activation_3 (Activation)	(None, 200)	0
dropout_3 (Dropout)	(None, 200)	0
dense_7 (Dense)	(None, 1)	201

Total params: 2,001 (7.82 KB)

Trainable params: 2,001 (7.82 KB)

Non-trainable params: 0 (0.00 B)

5. Modell illesztés

Miután definiáltuk az összes szükséges paramétert a modellünkben, kezdődhet maga a tanítási folyamat. Fontos megjegyezni, hogy ez egy masszívan számításigényes folyamat, ezért egy-egy tanítás könnyűszerrel igénybevehet több órát, de akár napokat is! Emiatt mindig érdemes egyszerűbb modelleket felépíteni, kevesebb réteggel és kevesebb neuronnal, a gyors betanulás érdekében. Ezen csak akkor érdemes javítani, ha a cél szempontjából jelentősen alulteljesít a modellünk.

In [ ]:
```
history=model.fit(X,Y,epochs=10000,
                  batch_size=16,
                  verbose=2,
                  validation_data=(X_valid, Y_valid),
                  callbacks=[checkpointer, early_stopping])
```

```
Epoch 1/10000

Epoch 1: val_loss improved from inf to 1.06768, saving model to weights.keras
1032/1032 - 2s - 2ms/step - loss: 1.0694 - val_loss: 1.0677
Epoch 2/10000

Epoch 2: val_loss did not improve from 1.06768
1032/1032 - 1s - 1ms/step - loss: 0.8673 - val_loss: 1.0706
Epoch 3/10000

Epoch 3: val_loss improved from 1.06768 to 0.87741, saving model to weights.keras
1032/1032 - 1s - 1ms/step - loss: 0.8101 - val_loss: 0.8774
Epoch 4/10000

Epoch 4: val_loss did not improve from 0.87741
1032/1032 - 1s - 1ms/step - loss: 0.7799 - val_loss: 0.9797
Epoch 5/10000

Epoch 5: val_loss improved from 0.87741 to 0.73711, saving model to weights.keras
1032/1032 - 1s - 1ms/step - loss: 0.7491 - val_loss: 0.7371
Epoch 6/10000

Epoch 6: val_loss improved from 0.73711 to 0.73673, saving model to weights.keras
1032/1032 - 1s - 1ms/step - loss: 0.7382 - val_loss: 0.7367
Epoch 7/10000

Epoch 7: val_loss improved from 0.73673 to 0.65492, saving model to weights.keras
1032/1032 - 1s - 1ms/step - loss: 0.7182 - val_loss: 0.6549
Epoch 8/10000

Epoch 8: val_loss did not improve from 0.65492
1032/1032 - 1s - 1ms/step - loss: 0.7057 - val_loss: 0.7361
Epoch 9/10000

Epoch 9: val_loss did not improve from 0.65492
1032/1032 - 1s - 1ms/step - loss: 0.7014 - val_loss: 0.8503
Epoch 10/10000

Epoch 10: val_loss did not improve from 0.65492
1032/1032 - 2s - 1ms/step - loss: 0.6877 - val_loss: 0.9129
Epoch 11/10000

Epoch 11: val_loss did not improve from 0.65492
1032/1032 - 2s - 2ms/step - loss: 0.6824 - val_loss: 0.6870
Epoch 12/10000

Epoch 12: val_loss did not improve from 0.65492
1032/1032 - 2s - 2ms/step - loss: 0.6778 - val_loss: 0.6609
Epoch 13/10000

Epoch 13: val_loss did not improve from 0.65492
1032/1032 - 1s - 1ms/step - loss: 0.6641 - val_loss: 0.9203
Epoch 14/10000

Epoch 14: val_loss did not improve from 0.65492
1032/1032 - 2s - 2ms/step - loss: 0.6660 - val_loss: 0.8083
Epoch 15/10000

Epoch 15: val_loss did not improve from 0.65492
1032/1032 - 2s - 2ms/step - loss: 0.6613 - val_loss: 0.6572
Epoch 16/10000

Epoch 16: val_loss did not improve from 0.65492
1032/1032 - 3s - 3ms/step - loss: 0.6452 - val_loss: 0.8144
Epoch 17/10000

Epoch 17: val_loss did not improve from 0.65492
1032/1032 - 1s - 1ms/step - loss: 0.6476 - val_loss: 0.7051
Epoch 18/10000
```

```
Epoch 18: val_loss did not improve from 0.65492
1032/1032 - 1s - 1ms/step - loss: 0.6425 - val_loss: 0.7654
Epoch 19/10000

Epoch 19: val_loss improved from 0.65492 to 0.64153, saving model to weights.keras
1032/1032 - 1s - 1ms/step - loss: 0.6404 - val_loss: 0.6415
Epoch 20/10000

Epoch 20: val_loss did not improve from 0.64153
1032/1032 - 1s - 1ms/step - loss: 0.6331 - val_loss: 0.8400
Epoch 21/10000

Epoch 21: val_loss did not improve from 0.64153
1032/1032 - 1s - 1ms/step - loss: 0.6282 - val_loss: 0.6612
Epoch 22/10000

Epoch 22: val_loss did not improve from 0.64153
1032/1032 - 1s - 1ms/step - loss: 0.6215 - val_loss: 0.6682
Epoch 23/10000

Epoch 23: val_loss did not improve from 0.64153
1032/1032 - 1s - 1ms/step - loss: 0.6167 - val_loss: 0.6553
Epoch 24/10000

Epoch 24: val_loss did not improve from 0.64153
1032/1032 - 1s - 1ms/step - loss: 0.6216 - val_loss: 0.7703
Epoch 25/10000

Epoch 25: val_loss improved from 0.64153 to 0.59446, saving model to weights.keras
1032/1032 - 1s - 1ms/step - loss: 0.6130 - val_loss: 0.5945
Epoch 26/10000

Epoch 26: val_loss did not improve from 0.59446
1032/1032 - 1s - 1ms/step - loss: 0.6073 - val_loss: 0.6486
Epoch 27/10000

Epoch 27: val_loss did not improve from 0.59446
1032/1032 - 1s - 1ms/step - loss: 0.6067 - val_loss: 0.7002
Epoch 28/10000

Epoch 28: val_loss improved from 0.59446 to 0.58177, saving model to weights.keras
1032/1032 - 1s - 1ms/step - loss: 0.6044 - val_loss: 0.5818
Epoch 29/10000

Epoch 29: val_loss did not improve from 0.58177
1032/1032 - 1s - 1ms/step - loss: 0.6002 - val_loss: 0.6317
Epoch 30/10000

Epoch 30: val_loss did not improve from 0.58177
1032/1032 - 1s - 1ms/step - loss: 0.5987 - val_loss: 0.6642
Epoch 31/10000

Epoch 31: val_loss did not improve from 0.58177
1032/1032 - 1s - 1ms/step - loss: 0.5970 - val_loss: 0.6037
Epoch 32/10000

Epoch 32: val_loss improved from 0.58177 to 0.55193, saving model to weights.keras
1032/1032 - 1s - 1ms/step - loss: 0.5936 - val_loss: 0.5519
Epoch 33/10000

Epoch 33: val_loss did not improve from 0.55193
1032/1032 - 1s - 1ms/step - loss: 0.5936 - val_loss: 0.6184
Epoch 34/10000

Epoch 34: val_loss did not improve from 0.55193
1032/1032 - 1s - 1ms/step - loss: 0.5848 - val_loss: 0.6605
Epoch 35/10000
```

```
Epoch 35: val_loss did not improve from 0.55193
1032/1032 - 1s - 1ms/step - loss: 0.5865 - val_loss: 0.5662
Epoch 36/10000

Epoch 36: val_loss did not improve from 0.55193
1032/1032 - 1s - 1ms/step - loss: 0.5810 - val_loss: 0.6400
Epoch 37/10000

Epoch 37: val_loss did not improve from 0.55193
1032/1032 - 1s - 1ms/step - loss: 0.5843 - val_loss: 0.6476
Epoch 38/10000

Epoch 38: val_loss did not improve from 0.55193
1032/1032 - 1s - 1ms/step - loss: 0.5749 - val_loss: 0.6432
Epoch 39/10000

Epoch 39: val_loss did not improve from 0.55193
1032/1032 - 1s - 1ms/step - loss: 0.5739 - val_loss: 0.6110
Epoch 40/10000

Epoch 40: val_loss did not improve from 0.55193
1032/1032 - 1s - 1ms/step - loss: 0.5746 - val_loss: 0.6753
Epoch 41/10000

Epoch 41: val_loss did not improve from 0.55193
1032/1032 - 1s - 1ms/step - loss: 0.5755 - val_loss: 0.5606
Epoch 42/10000

Epoch 42: val_loss did not improve from 0.55193
1032/1032 - 1s - 1ms/step - loss: 0.5653 - val_loss: 0.6405
Epoch 43/10000

Epoch 43: val_loss did not improve from 0.55193
1032/1032 - 1s - 1ms/step - loss: 0.5669 - val_loss: 0.6713
Epoch 44/10000

Epoch 44: val_loss did not improve from 0.55193
1032/1032 - 1s - 1ms/step - loss: 0.5673 - val_loss: 0.5772
Epoch 45/10000

Epoch 45: val_loss improved from 0.55193 to 0.55031, saving model to weights.keras
1032/1032 - 1s - 1ms/step - loss: 0.5655 - val_loss: 0.5503
Epoch 46/10000

Epoch 46: val_loss did not improve from 0.55031
1032/1032 - 1s - 1ms/step - loss: 0.5661 - val_loss: 0.6810
Epoch 47/10000

Epoch 47: val_loss did not improve from 0.55031
1032/1032 - 1s - 1ms/step - loss: 0.5622 - val_loss: 0.6046
Epoch 48/10000

Epoch 48: val_loss improved from 0.55031 to 0.53880, saving model to weights.keras
1032/1032 - 2s - 2ms/step - loss: 0.5613 - val_loss: 0.5388
Epoch 49/10000

Epoch 49: val_loss did not improve from 0.53880
1032/1032 - 1s - 1ms/step - loss: 0.5511 - val_loss: 0.5852
Epoch 50/10000

Epoch 50: val_loss did not improve from 0.53880
1032/1032 - 1s - 1ms/step - loss: 0.5540 - val_loss: 0.6754
Epoch 51/10000

Epoch 51: val_loss did not improve from 0.53880
1032/1032 - 1s - 1ms/step - loss: 0.5594 - val_loss: 0.5476
Epoch 52/10000

Epoch 52: val_loss did not improve from 0.53880
```

```
1032/1032 - 1s - 1ms/step - loss: 0.5480 - val_loss: 0.5667
Epoch 53/10000

Epoch 53: val_loss improved from 0.53880 to 0.53806, saving model to weights.keras
1032/1032 - 1s - 1ms/step - loss: 0.5453 - val_loss: 0.5381
Epoch 54/10000

Epoch 54: val_loss did not improve from 0.53806
1032/1032 - 1s - 1ms/step - loss: 0.5477 - val_loss: 0.5677
Epoch 55/10000

Epoch 55: val_loss did not improve from 0.53806
1032/1032 - 1s - 1ms/step - loss: 0.5441 - val_loss: 0.6397
Epoch 56/10000

Epoch 56: val_loss did not improve from 0.53806
1032/1032 - 1s - 1ms/step - loss: 0.5490 - val_loss: 0.6335
Epoch 57/10000

Epoch 57: val_loss improved from 0.53806 to 0.53784, saving model to weights.keras
1032/1032 - 2s - 1ms/step - loss: 0.5424 - val_loss: 0.5378
Epoch 58/10000

Epoch 58: val_loss did not improve from 0.53784
1032/1032 - 1s - 1ms/step - loss: 0.5471 - val_loss: 0.7138
Epoch 59/10000

Epoch 59: val_loss did not improve from 0.53784
1032/1032 - 1s - 1ms/step - loss: 0.5396 - val_loss: 0.6075
Epoch 60/10000

Epoch 60: val_loss did not improve from 0.53784
1032/1032 - 1s - 1ms/step - loss: 0.5373 - val_loss: 0.6033
Epoch 61/10000

Epoch 61: val_loss did not improve from 0.53784
1032/1032 - 1s - 1ms/step - loss: 0.5439 - val_loss: 0.5990
Epoch 62/10000

Epoch 62: val_loss improved from 0.53784 to 0.52510, saving model to weights.keras
1032/1032 - 2s - 1ms/step - loss: 0.5393 - val_loss: 0.5251
Epoch 63/10000

Epoch 63: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5362 - val_loss: 0.5744
Epoch 64/10000

Epoch 64: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5356 - val_loss: 0.5686
Epoch 65/10000

Epoch 65: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5387 - val_loss: 0.5909
Epoch 66/10000

Epoch 66: val_loss did not improve from 0.52510
1032/1032 - 2s - 2ms/step - loss: 0.5307 - val_loss: 0.5481
Epoch 67/10000

Epoch 67: val_loss did not improve from 0.52510
1032/1032 - 2s - 1ms/step - loss: 0.5348 - val_loss: 0.6118
Epoch 68/10000

Epoch 68: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5293 - val_loss: 0.5542
Epoch 69/10000

Epoch 69: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5317 - val_loss: 0.5350
```

```
Epoch 70/10000

Epoch 70: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5301 - val_loss: 0.5983
Epoch 71/10000

Epoch 71: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5309 - val_loss: 0.5849
Epoch 72/10000

Epoch 72: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5309 - val_loss: 0.5265
Epoch 73/10000

Epoch 73: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5253 - val_loss: 0.6071
Epoch 74/10000

Epoch 74: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5232 - val_loss: 0.5863
Epoch 75/10000

Epoch 75: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5264 - val_loss: 0.6499
Epoch 76/10000

Epoch 76: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5273 - val_loss: 0.6208
Epoch 77/10000

Epoch 77: val_loss did not improve from 0.52510
1032/1032 - 2s - 2ms/step - loss: 0.5233 - val_loss: 0.5998
Epoch 78/10000

Epoch 78: val_loss did not improve from 0.52510
1032/1032 - 2s - 2ms/step - loss: 0.5203 - val_loss: 0.5370
Epoch 79/10000

Epoch 79: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5270 - val_loss: 0.6042
Epoch 80/10000

Epoch 80: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5207 - val_loss: 0.5395
Epoch 81/10000

Epoch 81: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5213 - val_loss: 0.6686
Epoch 82/10000

Epoch 82: val_loss did not improve from 0.52510
1032/1032 - 2s - 2ms/step - loss: 0.5181 - val_loss: 0.5719
Epoch 83/10000

Epoch 83: val_loss did not improve from 0.52510
1032/1032 - 2s - 2ms/step - loss: 0.5167 - val_loss: 0.5841
Epoch 84/10000

Epoch 84: val_loss did not improve from 0.52510
1032/1032 - 2s - 1ms/step - loss: 0.5139 - val_loss: 0.5344
Epoch 85/10000

Epoch 85: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5213 - val_loss: 0.5505
Epoch 86/10000

Epoch 86: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5099 - val_loss: 0.6516
Epoch 87/10000
```

```
Epoch 87: val_loss did not improve from 0.52510
1032/1032 - 2s - 2ms/step - loss: 0.5152 - val_loss: 0.6284
Epoch 88/10000

Epoch 88: val_loss did not improve from 0.52510
1032/1032 - 1s - 1ms/step - loss: 0.5128 - val_loss: 0.5349
Epoch 89/10000

Epoch 89: val_loss improved from 0.52510 to 0.52388, saving model to weights.keras
1032/1032 - 1s - 1ms/step - loss: 0.5127 - val_loss: 0.5239
Epoch 90/10000

Epoch 90: val_loss did not improve from 0.52388
1032/1032 - 2s - 2ms/step - loss: 0.5072 - val_loss: 0.5599
Epoch 91/10000

Epoch 91: val_loss did not improve from 0.52388
1032/1032 - 2s - 2ms/step - loss: 0.5037 - val_loss: 0.5247
Epoch 92/10000

Epoch 92: val_loss improved from 0.52388 to 0.52008, saving model to weights.keras
1032/1032 - 1s - 1ms/step - loss: 0.5074 - val_loss: 0.5201
Epoch 93/10000

Epoch 93: val_loss did not improve from 0.52008
1032/1032 - 1s - 1ms/step - loss: 0.5055 - val_loss: 0.5757
Epoch 94/10000

Epoch 94: val_loss did not improve from 0.52008
1032/1032 - 1s - 1ms/step - loss: 0.5122 - val_loss: 0.5304
Epoch 95/10000

Epoch 95: val_loss did not improve from 0.52008
1032/1032 - 1s - 1ms/step - loss: 0.5082 - val_loss: 0.5517
Epoch 96/10000

Epoch 96: val_loss did not improve from 0.52008
1032/1032 - 1s - 1ms/step - loss: 0.5047 - val_loss: 0.5818
Epoch 97/10000

Epoch 97: val_loss did not improve from 0.52008
1032/1032 - 1s - 1ms/step - loss: 0.5090 - val_loss: 0.5513
Epoch 98/10000

Epoch 98: val_loss did not improve from 0.52008
1032/1032 - 1s - 1ms/step - loss: 0.5038 - val_loss: 0.5280
Epoch 99/10000

Epoch 99: val_loss did not improve from 0.52008
1032/1032 - 1s - 1ms/step - loss: 0.4982 - val_loss: 0.5233
Epoch 100/10000

Epoch 100: val_loss did not improve from 0.52008
1032/1032 - 1s - 1ms/step - loss: 0.5039 - val_loss: 0.6012
Epoch 101/10000

Epoch 101: val_loss did not improve from 0.52008
1032/1032 - 1s - 1ms/step - loss: 0.5061 - val_loss: 0.5471
Epoch 102/10000

Epoch 102: val_loss did not improve from 0.52008
1032/1032 - 1s - 1ms/step - loss: 0.5028 - val_loss: 0.5320
Epoch 103/10000

Epoch 103: val_loss did not improve from 0.52008
1032/1032 - 2s - 2ms/step - loss: 0.4987 - val_loss: 0.5412
Epoch 104/10000
```

```
Epoch 104: val_loss did not improve from 0.52008
1032/1032 - 1s - 1ms/step - loss: 0.4968 - val_loss: 0.5544
Epoch 105/10000

Epoch 105: val_loss did not improve from 0.52008
1032/1032 - 1s - 1ms/step - loss: 0.4930 - val_loss: 0.5339
Epoch 106/10000

Epoch 106: val_loss improved from 0.52008 to 0.51634, saving model to weights.keras
1032/1032 - 1s - 1ms/step - loss: 0.5028 - val_loss: 0.5163
Epoch 107/10000

Epoch 107: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4914 - val_loss: 0.5385
Epoch 108/10000

Epoch 108: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4913 - val_loss: 0.5455
Epoch 109/10000

Epoch 109: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.5021 - val_loss: 0.5257
Epoch 110/10000

Epoch 110: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4956 - val_loss: 0.5344
Epoch 111/10000

Epoch 111: val_loss did not improve from 0.51634
1032/1032 - 2s - 1ms/step - loss: 0.4985 - val_loss: 0.5189
Epoch 112/10000

Epoch 112: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4968 - val_loss: 0.5410
Epoch 113/10000

Epoch 113: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4903 - val_loss: 0.5187
Epoch 114/10000

Epoch 114: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4919 - val_loss: 0.5520
Epoch 115/10000

Epoch 115: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4977 - val_loss: 0.5205
Epoch 116/10000

Epoch 116: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4869 - val_loss: 0.5794
Epoch 117/10000

Epoch 117: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4851 - val_loss: 0.5336
Epoch 118/10000

Epoch 118: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4889 - val_loss: 0.5317
Epoch 119/10000

Epoch 119: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4954 - val_loss: 0.5420
Epoch 120/10000

Epoch 120: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4891 - val_loss: 0.5385
Epoch 121/10000

Epoch 121: val_loss did not improve from 0.51634
```

```
1032/1032 - 1s - 1ms/step - loss: 0.4884 - val_loss: 0.5281
Epoch 122/10000

Epoch 122: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4896 - val_loss: 0.5258
Epoch 123/10000

Epoch 123: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4945 - val_loss: 0.6069
Epoch 124/10000

Epoch 124: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4879 - val_loss: 0.5594
Epoch 125/10000

Epoch 125: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4823 - val_loss: 0.6107
Epoch 126/10000

Epoch 126: val_loss did not improve from 0.51634
1032/1032 - 2s - 2ms/step - loss: 0.4877 - val_loss: 0.5652
Epoch 127/10000

Epoch 127: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4869 - val_loss: 0.5292
Epoch 128/10000

Epoch 128: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4819 - val_loss: 0.5284
Epoch 129/10000

Epoch 129: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4824 - val_loss: 0.5594
Epoch 130/10000

Epoch 130: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4853 - val_loss: 0.5322
Epoch 131/10000

Epoch 131: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4845 - val_loss: 0.5183
Epoch 132/10000

Epoch 132: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4874 - val_loss: 0.5166
Epoch 133/10000

Epoch 133: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4850 - val_loss: 0.5170
Epoch 134/10000

Epoch 134: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4798 - val_loss: 0.5223
Epoch 135/10000

Epoch 135: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4877 - val_loss: 0.5209
Epoch 136/10000

Epoch 136: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4835 - val_loss: 0.5460
Epoch 137/10000

Epoch 137: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4811 - val_loss: 0.5285
Epoch 138/10000

Epoch 138: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4859 - val_loss: 0.5487
```

```
Epoch 139/10000

Epoch 139: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4760 - val_loss: 0.5224
Epoch 140/10000

Epoch 140: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4803 - val_loss: 0.6774
Epoch 141/10000

Epoch 141: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4814 - val_loss: 0.5410
Epoch 142/10000

Epoch 142: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4774 - val_loss: 0.6160
Epoch 143/10000

Epoch 143: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4791 - val_loss: 0.5753
Epoch 144/10000

Epoch 144: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4817 - val_loss: 0.5347
Epoch 145/10000

Epoch 145: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4815 - val_loss: 0.5364
Epoch 146/10000

Epoch 146: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4775 - val_loss: 0.5267
Epoch 147/10000

Epoch 147: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4697 - val_loss: 0.5198
Epoch 148/10000

Epoch 148: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4864 - val_loss: 0.5298
Epoch 149/10000

Epoch 149: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4737 - val_loss: 0.5439
Epoch 150/10000

Epoch 150: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4758 - val_loss: 0.5265
Epoch 151/10000

Epoch 151: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4773 - val_loss: 0.5199
Epoch 152/10000

Epoch 152: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4754 - val_loss: 0.5219
Epoch 153/10000

Epoch 153: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4749 - val_loss: 0.6012
Epoch 154/10000

Epoch 154: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4678 - val_loss: 0.5325
Epoch 155/10000

Epoch 155: val_loss did not improve from 0.51634
1032/1032 - 2s - 1ms/step - loss: 0.4767 - val_loss: 0.5690
Epoch 156/10000
```

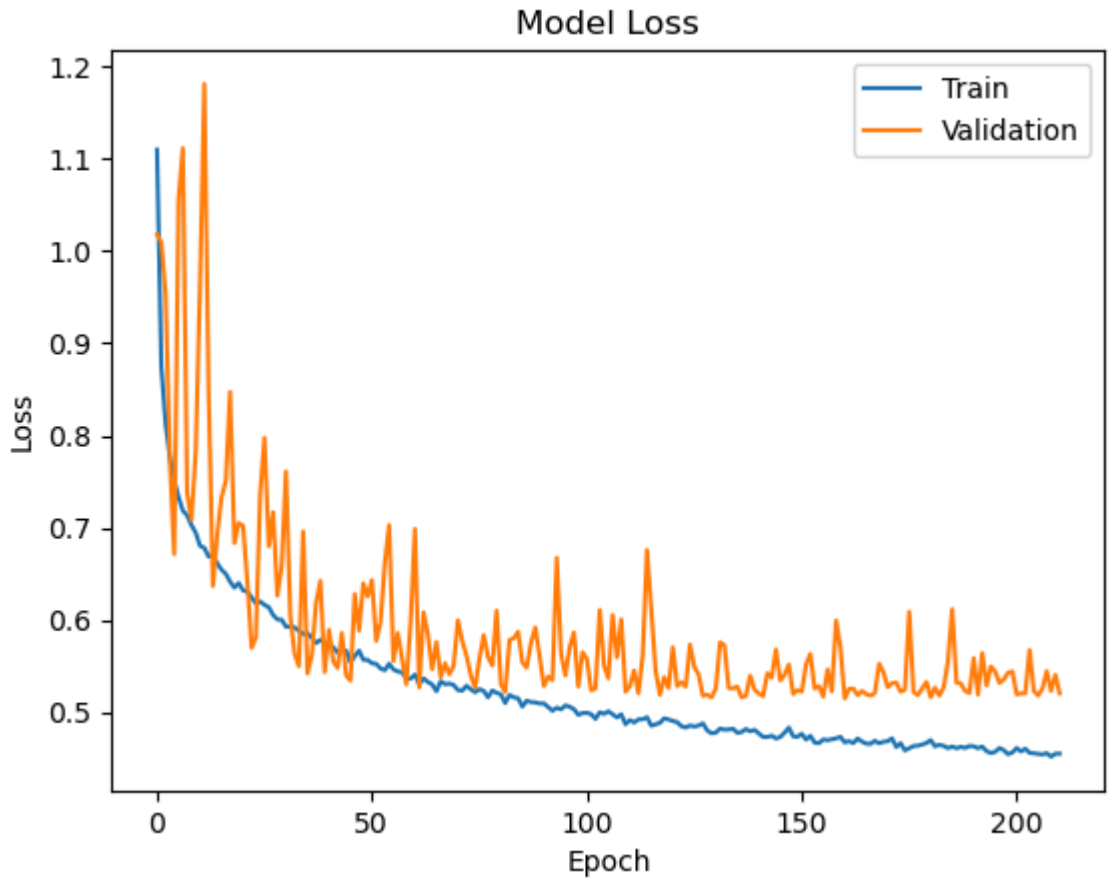
```
Epoch 156: val_loss did not improve from 0.51634
1032/1032 - 1s - 1ms/step - loss: 0.4677 - val_loss: 0.5174
Epoch 156: early stopping
```

6. Kiértékelés

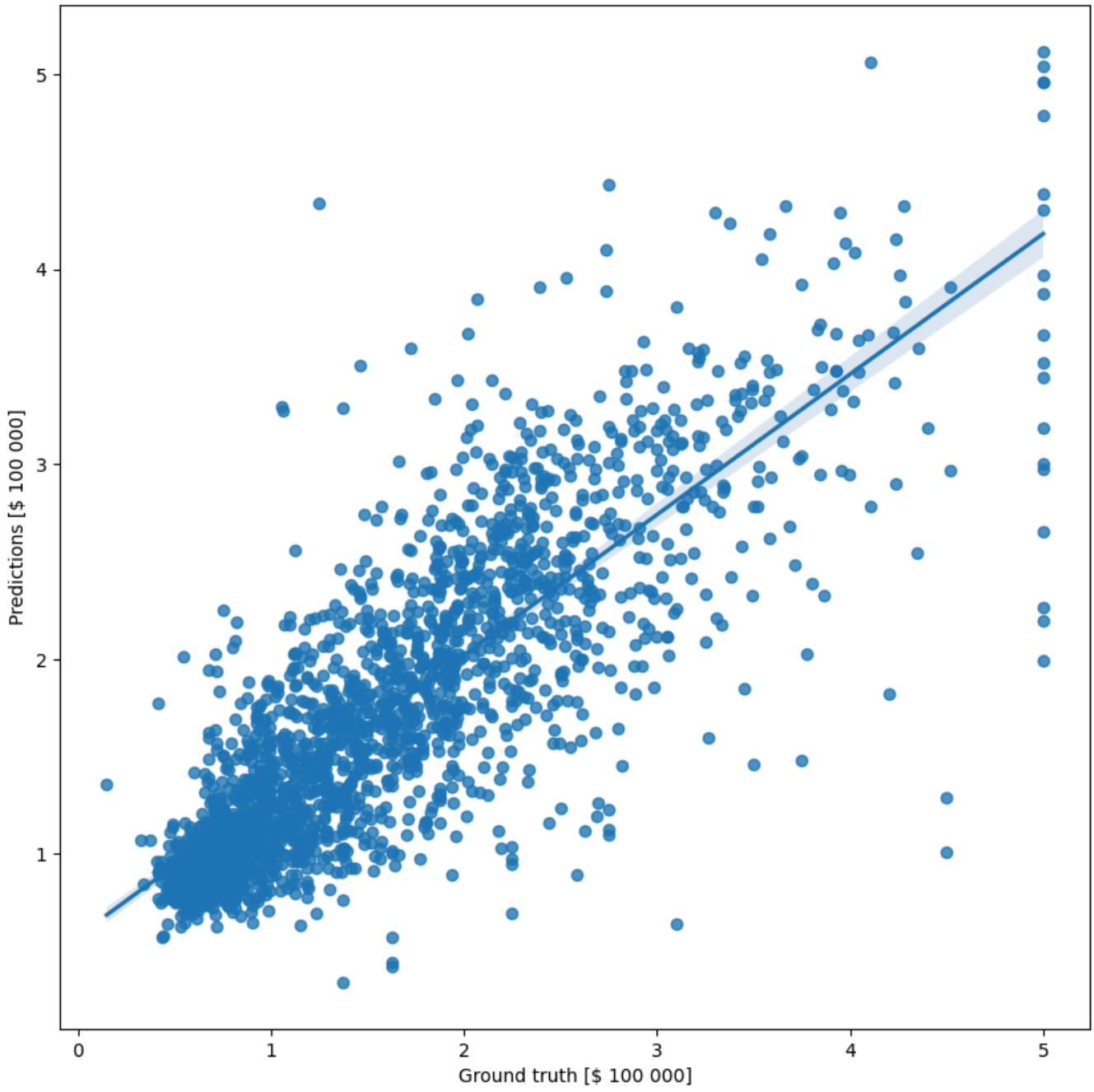
In [ ]:
```python
from sklearn.metrics import mean_squared_error
import math

# Betöltjük az elmentett modellt
model = load_model('weights.keras')

# Predikálunk a teszt adatbázison
preds = model.predict(X_test)

# Kiértékeljük a tesztelési adatbázison a modell hibáját
test_err = mean_squared_error(Y_test,preds)
print("\nTeszt hiba: %f" % (test_err))
print(f"Ez az jelenti, hogy sqrt({test_err:0.3f}))={math.sqrt(test_err):0.3f}-et hibáz átlago
print(f"Ez annyit jelent, hogy átlagosan {math.sqrt(test_err)*100000:0.0f}$-t téved a modellü
```

```
65/65 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step

Teszt hiba: 0.285780
Ez az jelenti, hogy sqrt(0.286))=0.535-et hibáz átlagosan a modell a teszt adatokon.
Ez annyit jelent, hogy átlagosan 53458$-t téved a modellünk.
```

In [ ]:
```python
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Train')
plt.plot(history.history['val_loss'], label='Validation')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



In [ ]:
```
import seaborn as sns

plt.figure(figsize=(10,10))
fig = sns.regplot(x=Y_test, y=preds.reshape(-1));
# fig.set(xlim=(10,30),ylim=(10,30))
plt.xlabel("Ground truth [$ 100 000]")
plt.ylabel("Predictions [$ 100 000]")
plt.show()
```



7. A modell finomhangolása

Jelen esetben megállapíthatjuk, hogy a modell tanítása során nem jelentkezett semmilyen anomálisa, vagy rendellenesség. Nem állapítható meg túltanulás sem, ezért a kapott modellt **elfogadom**.

Ezen felül még érdemes figyelni, hogy mennyire gyorsan tud predikálni a modell és figyelembe venni, hogy vajon emellett képesek vagyunk-e valós idejű alkalmazást létrehozni. Ha nem, akkor célszerű egy egyszerűbb modellt választani.

8. Alkalmazás

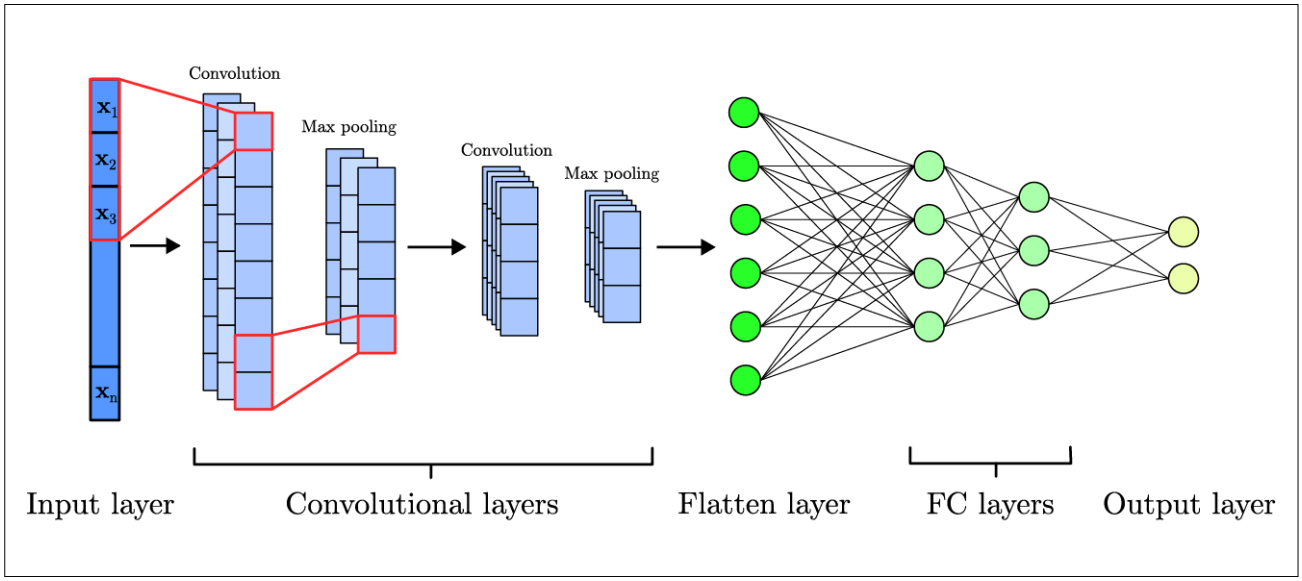
Ha már nem tartunk igényt a modell további finomhangolására, akkor a `weights.keras` file-t elmentve készen állunk arra, hogy alkalmazásba ültessük a neurális hálónkat.

9. hét / III. Klasszifikáció: MNIST

Jelen példában ismételten egy "Hello World" típusú feladatot fogunk megvizsgálni, viszont jelen esetben már nem egy MLP (Multi Layer Perceptron) modellt fogunk segítségül hívni, hanem egy úgynevezett Konvolúciós Neurális Hálót, röviden CNN-t!

A feladat a következő: adott az MNIST adatbázis, amely rengeteg kézzel írott számjegyről tartalmaz 256x256-os felbontásban képeket. A feladatunk az, hogy készítsünk egy olyan modellt, amely képes egy újonnan beadott képről eldönteni, hogy azon milyen számjegy található!

Mielőtt mélyebbre áskálódnánk, fontos tisztázni, hogy miből is épül fel egy CNN! Tekintsük az alábbi példát:



In [ ]:
```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Betöltjük az adatbázist - itt eleve szét van szedve tanító és teszt adathalamzra
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [ ]:
```
x_train = x_train / 255
x_test = x_test / 255

x_train = x_train.reshape(-1,28,28,1)
x_test = x_test.reshape(-1,28,28,1)
x_train = x_train.astype("float32")
x_test = x_test.astype("float32")

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

batch_size=128
```

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Dropout, Flatten, Embedding
from tensorflow.keras.optimizers import SGD

model = Sequential()
model.add(Conv2D(32, kernel_size=(5, 5), activation='relu', input_shape=(28,28,1,)))
model.add(Conv2D(64, (5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

print(model.summary())
```

```
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"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 24, 24, 32)	832
conv2d_1 (Conv2D)	(None, 20, 20, 64)	51,264
max_pooling2d (MaxPooling2D)	(None, 10, 10, 64)	0
dropout (Dropout)	(None, 10, 10, 64)	0
flatten (Flatten)	(None, 6400)	0
dense (Dense)	(None, 128)	819,328
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Total params: 872,714 (3.33 MB)

Trainable params: 872,714 (3.33 MB)

Non-trainable params: 0 (0.00 B)

None

In [ ]:
```
# Early stopping, amellyel figyeljük a validációs hibát (alap beállítás)
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, TensorBoard
patience=10
early_stopping=EarlyStopping(patience=patience, verbose=1)
# Szintén a validációs hibát figyeljük, és elmentjük a legjobb modellt
checkpointer=ModelCheckpoint(filepath='mopdel.keras', save_best_only=True, verbose=1)
tb = TensorBoard(log_dir='logs', histogram_freq=1, write_graph=1)
```

In [ ]:
```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

network_history = model.fit(x_train, y_train, batch_size=128, epochs=30, verbose=1, validatio
```

```
Epoch 1/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 183ms/step - accuracy: 0.8315 - loss: 0.5210
Epoch 1: val_loss improved from inf to 0.05900, saving model to mopdel.keras
375/375 ━━━━━━━━━━━━━━━━━━━━ 78s 201ms/step - accuracy: 0.8317 - loss: 0.5203 - val_accuracy:
0.9826 - val_loss: 0.0590
Epoch 2/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 175ms/step - accuracy: 0.9714 - loss: 0.0967
Epoch 2: val_loss improved from 0.05900 to 0.04606, saving model to mopdel.keras
375/375 ━━━━━━━━━━━━━━━━━━━━ 71s 188ms/step - accuracy: 0.9714 - loss: 0.0967 - val_accuracy:
0.9867 - val_loss: 0.0461
Epoch 3/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 173ms/step - accuracy: 0.9791 - loss: 0.0686
Epoch 3: val_loss improved from 0.04606 to 0.03502, saving model to mopdel.keras
375/375 ━━━━━━━━━━━━━━━━━━━━ 76s 204ms/step - accuracy: 0.9791 - loss: 0.0685 - val_accuracy:
0.9899 - val_loss: 0.0350
Epoch 4/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 194ms/step - accuracy: 0.9859 - loss: 0.0476
Epoch 4: val_loss improved from 0.03502 to 0.03394, saving model to mopdel.keras
375/375 ━━━━━━━━━━━━━━━━━━━━ 78s 209ms/step - accuracy: 0.9859 - loss: 0.0476 - val_accuracy:
0.9903 - val_loss: 0.0339
Epoch 5/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 205ms/step - accuracy: 0.9873 - loss: 0.0418
Epoch 5: val_loss did not improve from 0.03394
375/375 ━━━━━━━━━━━━━━━━━━━━ 86s 218ms/step - accuracy: 0.9873 - loss: 0.0418 - val_accuracy:
0.9895 - val_loss: 0.0396
Epoch 6/30
  4/375 ━━━━━━━━━━━━━━━━━━━━ 1:00 164ms/step - accuracy: 0.9840 - loss: 0.0349
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[6], line 3
      1 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accurac
y'])
----> 3 network_history = model.fit(x_train, y_train, batch_size=128, epochs=30, verbose=1, v
alidation_split=0.2, callbacks=[early_stopping, checkpointer, tb])

File c:\Users\mbenc\anaconda3\lib\site-packages\keras\src\utils\traceback_utils.py:117, in fi
lter_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_spli
t, validation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, val
idation_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:15
0, 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_inpu
ts)
   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 C
ontext.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 qui
ck_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:
```

In [ ]:

```
from keras.models import load_model
model = load_model("mopdel.keras")
test_err = model.evaluate(x_test,y_test)
print("Teszt hiba:", test_err[0], "Teszt pontosság:", test_err[1])
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - accuracy: 0.9892 - loss: 0.0306
Teszt hiba: 0.025237729772925377 Teszt pontosság: 0.991599977016449
```

In [ ]:
```python
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_curv
# először is nyerjük ki a predikciókat (valószínűség és hozzá tartozó pontosságot)
y_pred = model.predict(x_test)
y_pred = np.argmax(y_pred,1)
y_true = np.argmax(y_test,1)

print("test accuracy: %g" %(accuracy_score(y_true, y_pred)))
print("Precision", precision_score(y_true, y_pred, average="macro"))
print("Recall", recall_score(y_true, y_pred, average="macro"))
print("f1_score", f1_score(y_true, y_pred, average="macro"))
print("\nKonfúziós mátrix: ")
conf=confusion_matrix(y_true, y_pred)
print(conf)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step
test accuracy: 0.9916
Precision 0.991563758119932
Recall 0.991473706269493
f1_score 0.9915050484727665

Konfúziós mátrix:
[[ 973    0    0    0    0    0    4    1    2    0]
 [   0 1133    0    1    0    0    1    0    0    0]
 [   0    2 1025    0    1    0    0    4    0    0]
 [   0    0    1 1005    0    1    0    2    1    0]
 [   0    0    0    0  978    0    0    1    0    3]
 [   0    0    0    7    0  881    2    1    0    1]
 [   0    2    1    0    2    4  947    0    2    0]
 [   0    1    2    3    1    0    0 1020    1    0]
 [   0    0    2    1    0    1    0    1  968    1]
 [   1    0    1    2    5    4    1    6    3  986]]
```

In [ ]:
```python
import seaborn as sns
ax = sns.heatmap(conf, annot=True, fmt='d', vmax=20, cmap='Blues') # a vmax paraméterrel állí
ax.set(xlabel='Predicted Label',
       ylabel='True label');
```