**Projekt Dokumentáció:**

**Útjelző Táblák Felismerése TensorFlow Technológiával**

# Bevezetés

A közlekedésbiztonság és a vezetői segédrendszerek fejlődése napjainkban egyre fontosabbá válik. Ezen projekt célja egy innovatív rendszer kifejlesztése, amely képes az útjelző táblák felismerésére és a rajtuk szereplő információk digitalizálására. A modern képfeldolgozási technikák és gépi tanulási modellek alkalmazásával, mint például a TensorFlow, a rendszer elemzi a közlekedési jelzéseket és azonosítja a szimbólumokat.

### Megoldandó Feladat Kifejtése

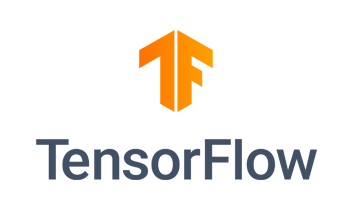
A projekt fő célja egy olyan modell létrehozása, amely képes felismerni és azonosítani különböző útjelző táblákat. Ez magában foglalja egy interaktív grafikus felhasználói interfész (GUI) kialakítását is, amely lehetővé teszi a felhasználók számára, hogy képeket töltsenek fel és azonnali visszajelzést kapjanak az elemzés eredményeiről.

# 2. Megoldáshoz Szükséges Elméleti Háttér

## Gépi Látás és Képfeldolgozás

A gépi látás a mesterséges intelligencia egy ága, amely a képek és videók automatikus elemzésével és értelmezésével foglalkozik. Ennek alapja a képfeldolgozás, amely magában foglalja a digitális képek különböző módszerekkel történő feldolgozását és elemzését. A képfeldolgozás alapvető lépései közé tartozik a képek előfeldolgozása, zajszűrés, éldetektálás és szegmentáció.

## TensorFlow és Mély Tanulás

A TensorFlow egy nyílt forráskódú szoftverkönyvtár, amelyet a Google fejlesztett ki a gépi tanulás és mély tanulás alkalmazására. A TensorFlow lehetővé teszi a különböző neurális hálózati modellek egyszerű és hatékony kialakítását, tréningelését és kiértékelését. A mély tanulás alapja a mély neurális hálózatok alkalmazása, amelyek több rétegen keresztül képesek a bemenetekből komplex jellemzőket kinyerni és értelmezni.

ábra 1 TrensorFlow logó

A TensorFlow különböző eszközöket és könyvtárakat biztosít a modellek építéséhez, tréningeléséhez és implementálásához. Például a Keras magas szintű API-t használja, amely egyszerűsíti a modellek definiálását és tréningelését, valamint lehetővé teszi a GPU-k és TPU-k kihasználását a számítási feladatok felgyorsításához. A TensorFlow.js segítségével böngészőben is futtathatunk modelleket, míg a TensorFlow Lite lehetővé teszi a modellek futtatását mobil és beágyazott eszközökön.

A TensorFlow támogatja a különböző típusú neurális hálózatokat, mint például a konvolúciós neurális hálózatokat (CNN-ek), amelyek különösen hatékonyak képfelismerési feladatokban, valamint a rekurzív neurális hálózatokat (RNN-ek), amelyek időbeli adatokat dolgoznak fel, mint például a beszédfelismerés és a természetes nyelvi feldolgozás.

A TensorFlow használata egyszerűen elkezdhető a hivatalos TensorFlow weboldalon található tutorialok és útmutatók segítségével, amelyek bemutatják, hogyan lehet adatokat betölteni, modelleket építeni és tréningelni, valamint értékelni a modellek teljesítményét.

Érdemes megtekinteni a következő forrásokat:

- [TensorFlow hivatalos weboldal](<https://www.tensorflow.org>)

- [Coursera TensorFlow kurzus](<https://www.coursera.org/learn/introduction-tensorflow>)

- [Machine Learning Mastery bevezető](<https://machinelearningmastery.com/tensorflow-tutorial-deep-learning/)>

# 3. A Megvalósítás Terve és Kivitelezése

## Adatgyűjtés és Előkészítés

A projekt sikeres megvalósításának egyik alapvető lépése az adatok összegyűjtése és előkészítése. Mivel a modellünk különböző útjelző táblákat fog felismerni, fontos, hogy minden típusú táblához megfelelő mennyiségű és minőségű kép álljon rendelkezésre. Az adatok minősége és mennyisége közvetlenül befolyásolja a modell teljesítményét és pontosságát.

## Adatgyűjtés

Az adatgyűjtés során különböző forrásokból származó képeket használtam fel, beleértve a nyilvánosan elérhető adatbázisokat, mint például a GTSRB (German Traffic Sign Recognition Benchmark) és a Chinese Traffic Sign Database.

Az adatgyűjtés során különös figyelmet fordítottam arra, hogy minden tábla különböző nézőpontokból és környezetben legyen ábrázolva. Ez magában foglalta a nappali és éjszakai képeket, valamint különböző távolságokat és szögeket. Ezek az eltérések segítenek a modellnek abban, hogy robusztusabbá váljon és jobban teljesítsen a valós környezetben a későbbiekben.

## Adatok Előkészítése

Az adatok előkészítése során a képeket először normalizáltam, hogy egységes méretűek és formátumúak legyenek. Ez magában foglalta a képek átméretezését 30x30 pixeles méretre, valamint a színcsatornák standardizálását. Az előkészítés során a képeket szürkeárnyalatos formátumba is konvertáltam, mivel a színek nem minden esetben jelentettek hozzáadott értéket a tábla felismerésében.

## Adat Argumentálás

Az adat argumentálás egy fontos lépés az adatok mennyiségének növelésére és a modell robusztusságának növelésére. Az argumentálás során különböző transzformációkat alkalmaztam a képekre, mint például:

Forgatás: Képek elforgatása különböző szögekkel (pl. -15, 0, +15 fok)

Eltolás: Képek eltolása vízszintes és függőleges irányban

Fényerő és kontraszt módosítása: Képek fényerejének és kontrasztjának növelése vagy csökkentése

## Zaj hozzáadása:

Ezek a technikák segítettek abban, hogy a modell jobban alkalmazkodjon a valós környezet változatosságához, és növelték a trénelési adatok számát anélkül, hogy új képeket kellett volna gyűjteni.

## Modell Felépítése és Trénelése

A modell felépítésének és trénelésének folyamata több lépésből állt, beleértve az adatok betöltését, az adatok előkészítését, a modell architektúrájának meghatározását, a modell trénelését és a modell kiértékelését.

Ez a adatok előkészítésének lépései a következők szerint vannak a kódrészletemben:

**settings.py**

Ebben a fájlban két változó van definiálva: `class\_id` és `num\_images`.

- *`class\_id*` egy karakterlánc, amely egy osztályazonosítót jelöl. Kezdetben `'64'` értékre van állítva, ami azt jelenti, hogy a betanítás ezzel az osztállyal kezdődik.

- *`num\_images`* értéke `20`, ami azt jelzi, hogy a modellt 20 képpel tesztelem.

**trainer.py**

Ez a fájl tartalmazza a betanításhoz szükséges fő logikát, beleértve a képek kezelését és az adatok CSV fájlba írását.

import cv2

import csv

import os

from data import csv\_data

from settings import class\_id, num\_images

- A szükséges modulok importálása

- A `data` modulból importálja a `csv\_data` adatokat, a `settings` modulból pedig az `class\_id` és `num\_images` változókat.

Létező képutak összegyűjtése a Train.csv fájlból

def get\_existing\_image\_paths(csv\_file\_path):

existing\_paths = set()

try:

with open(csv\_file\_path, mode='r', newline='') as file:

reader = csv.DictReader(file)

for row in reader:

existing\_paths.add(row['Path'])

except FileNotFoundError:

pass # Ha a Train.csv nem létezik, csak egy üres halmazt ad vissza

return existing\_paths

- Ez a függvény összegyűjti a már létező képutakat a `Train.csv` fájlból, és egy halmazban tárolja őket.

Utolsó feldolgozott kép keresése

def find\_last\_processed\_image(csv\_file\_path, class\_id):

max\_set\_counter = -1

max\_image\_counter = -1

try:

with open(csv\_file\_path, mode='r', newline='') as file:

reader = csv.DictReader(file)

for row in reader:

if class\_id in row['Path']:

parts = row['Path'].split('\_')

set\_counter, image\_counter = int(parts[-2]), int(parts[-1].split('.')[0])

if set\_counter > max\_set\_counter or (set\_counter == max\_set\_counter and image\_counter > max\_image\_counter):

max\_set\_counter, max\_image\_counter = set\_counter, image\_counter

except FileNotFoundError:

pass # Ha a Train.csv nem létezik, az elejéről indulunk

return max\_set\_counter, max\_image\_counter

- Ez a függvény megkeresi az utolsó feldolgozott képet a `Train.csv` fájlban az adott `class\_id` alapján.

Képutak érvényesítése

def validate\_image\_paths(image\_paths):

missing\_images = [img for img in image\_paths if not os.path.exists(img)]

if missing\_images:

print("A következő képek hiányoznak, és kihagyásra kerülnek:")

for missing in missing\_images:

print(missing)

return [img for img in image\_paths if os.path.exists(img)]

- Ez a függvény ellenőrzi, hogy a megadott képutak léteznek-e, és figyelmeztet, ha hiányzó képeket talál.

Fő kód a képfeldolgozáshoz és CSV fájlba íráshoz

class\_id\_padded = f"{int(class\_id):05d}"

output\_csv\_path = 'output.csv'

train\_csv\_path = 'Train.csv' # Az meglévő Train.csv fájl útvonala

base\_save\_dir = f'Train/{class\_id}'

trainer\_dir = 'Trainer'

image\_paths = [f'{trainer\_dir}/{class\_id}/image{i+1}.png' for i in range(num\_images)]

image\_paths += [f'{trainer\_dir}/{class\_id}/image{i:02d}.png' for i in range(1, num\_images+1)]

valid\_image\_paths = validate\_image\_paths(image\_paths) # Érvényesítés feldolgozás előtt

last\_set\_counter, last\_image\_counter = find\_last\_processed\_image(output\_csv\_path, class\_id)

total\_processed\_images = last\_set\_counter \* 29 + last\_image\_counter + 1

num\_existing\_images = len([img for img in valid\_image\_paths if os.path.exists(img)])

starting\_image\_index = total\_processed\_images - num\_existing\_images

existing\_image\_paths = get\_existing\_image\_paths(train\_csv\_path)

if not os.path.exists(base\_save\_dir):

os.makedirs(base\_save\_dir)

with open(output\_csv\_path, mode='w', newline='') as output\_file, open(train\_csv\_path, mode='a', newline='') as train\_file:

output\_writer = csv.writer(output\_file)

train\_writer = csv.writer(train\_file)

output\_writer.writerow(['Width', 'Height', 'Roi.X1', 'Roi.Y1', 'Roi.X2', 'Roi.Y2', 'ClassId', 'Path'])

set\_counter, image\_counter = (total\_processed\_images // 29, total\_processed\_images % 29) if total\_processed\_images > 0 else (0, 0)

if image\_counter >= 0:

image\_counter += 1

if image\_counter >= 29:

image\_counter = 0

set\_counter += 1

else:

image\_counter = 0

set\_counter = 0 if set\_counter == 0 else set\_counter + 1

for idx, image\_path in enumerate(valid\_image\_paths):

if idx < starting\_image\_index:

continue

input\_image = cv2.imread(image\_path)

if input\_image is None:

print(f"Sikertelen képbetöltés: {image\_path}")

continue

for width, height, roi\_x1, roi\_y1, roi\_x2, roi\_y2, \_ in csv\_data:

resized\_image = cv2.resize(input\_image, (width, height))

filename = f"{class\_id\_padded}\_{set\_counter:05d}\_{image\_counter:05d}.png"

relative\_path = f"Train/{class\_id}/{filename}"

save\_path = os.path.join(base\_save\_dir, filename)

if relative\_path in existing\_image\_paths:

continue

if not os.path.exists(os.path.dirname(save\_path)):

os.makedirs(os.path.dirname(save\_path))

cv2.imwrite(save\_path, resized\_image)

output\_writer.writerow([width, height, roi\_x1, roi\_y1, roi\_x2, roi\_y2, class\_id, relative\_path])

train\_writer.writerow([width, height, roi\_x1, roi\_y1, roi\_x2, roi\_y2, class\_id, relative\_path])

image\_counter += 1

if image\_counter >= 29:

image\_counter = 0

set\_counter += 1

**order.py**

Ez a fájl egy függvényt tartalmaz a képek átnevezésére egy adott könyvtárban. A **SORRENDISÉG** és a **betanítás** miatt fontos ez a lépés!

import os

import glob

def rename\_images(directory, extension="png"):

os.chdir(directory)

image\_files = glob.glob(f"\*.{extension}")

image\_files.sort()

for i, file in enumerate(image\_files, start=1):

new\_name = f"image{str(i).zfill(2)}.{extension}"

os.rename(file, new\_name)

print(f"Átnevezve '{file}' erre: '{new\_name}'")

directory\_path = 'Trainer/49'

rename\_images(directory\_path)

**data.py**

Ez a fájl tartalmazza a CSV-ben megadott képek rotációit és transzformációit:

csv\_data = [

[27, 26, 5, 5, 22, 20, 20],

[28, 27, 5, 6, 23, 22, 20],

[29, 26, 6, 5, 24, 21, 20],

[28, 27, 5, 6, 23, 22, 20],

[28, 26, 5, 5, 23, 21, 20],

[31, 27, 6, 5, 26, 22, 20],

[31, 28, 6, 6, 26, 23, 20],

[31, 28, 6, 6, 26, 23, 20],

[31, 29, 5, 6, 26, 24, 20],

[34, 32, 6, 6, 29, 26, 20],

[36, 33, 5, 6, 31, 28, 20],

[37, 34, 5, 6, 32, 29, 20],

[38, 34, 5, 6, 32, 29, 20],

[40, 34, 6, 6, 34, 29, 20],

[39, 34, 5, 5, 34, 29, 20],

[42, 36, 6, 5, 37, 31, 20],

[45, 39, 6, 5, 40, 34, 20],

[47, 42, 5, 5, 41, 36, 20],

[50, 45, 5, 5, 45, 40, 20],

[55, 49, 6, 5, 49, 43, 20],

[56, 51, 6, 6, 51, 46, 20],

[59, 54, 5, 5, 54, 49, 20],

[64, 57, 6, 5, 59, 52, 20],

[70, 61, 6, 5, 64, 56, 20],

[76, 69, 6, 6, 70, 63, 20],

[86, 75, 8, 6, 79, 69, 20],

[97, 87, 8, 7, 89, 80, 20],

[111, 100, 9, 8, 102, 92, 20],

[131, 119, 12, 11, 120, 109, 20], 1 darab kép 29 fajta transzformációja 29 darabbá

]

## Adatok Betöltése és Előkészítése

Az adatok betöltése és előkészítése során az összegyűjtött és előkészített képeket numpy tömbökbe rendeztük, hogy könnyen felhasználhatók legyenek a TensorFlow által. Az adatokat két fő részre osztottuk: tréning adatokra és teszt adatokra. A tréning adatokkal a modell tanul, míg a teszt adatokkal a modell teljesítményét értékeljük.

## Modell Architektúra

A modell architektúrájának meghatározása során a következő rétegeket használtuk, amelyek részletesen bemutatják az egyes rétegek működését és a kimeneti formákat:

### Konvolúciós Rétegek (Conv2D)

- \*\*conv2d (Conv2D)\*\*:

- \*\*Kimeneti alak\*\*: `(None, 26, 26, 32)`

- \*\*Leírás\*\*: Az első konvolúciós réteg, amely 32 darab, 3x3-as szűrőt használ. Ez a réteg a bemeneti képből (általában 28x28x1 méretű) kinyeri az alapvető jellemzőket.

- \*\*conv2d\_1 (Conv2D)\*\*:

- \*\*Kimeneti alak\*\*: `(None, 22, 22, 32)`

- \*\*Leírás\*\*: A második konvolúciós réteg, amely szintén 32 darab, 3x3-as szűrőt alkalmaz, további jellemzőket kinyerve a képből.

### Max Pooling Rétegek (MaxPooling2D)

- \*\*max\_pooling2d (MaxPooling2D)\*\*:

- \*\*Kimeneti alak\*\*: `(None, 11, 11, 32)`

- \*\*Leírás\*\*: Ez a réteg 2x2-es ablakokat használ, hogy a térbeli méreteket felezze, csökkentve az adatok számát, és kiemelve a legfontosabb jellemzőket.

### Dropout Rétegek (Dropout)

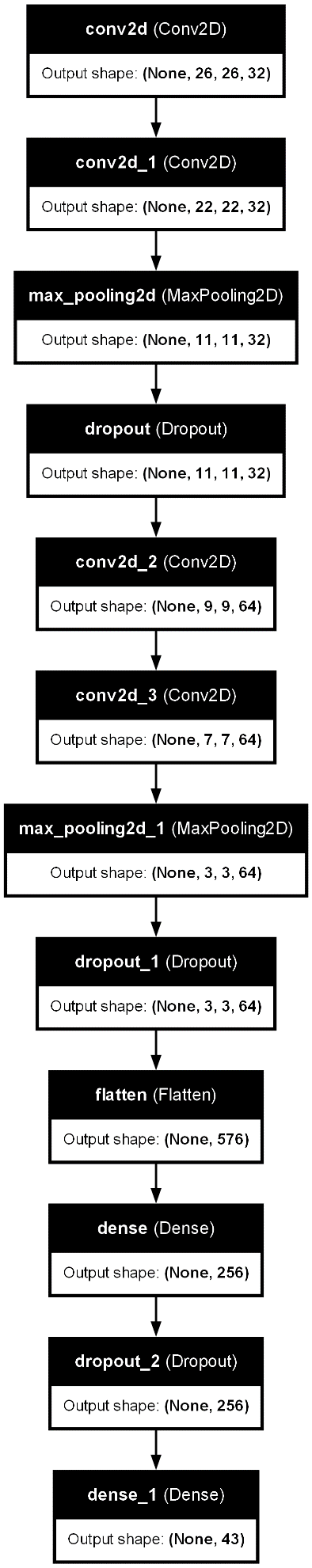
- \*\*dropout (Dropout)\*\*:

- \*\*Kimeneti alak\*\*: `(None, 11, 11, 32)`

- \*\*Leírás\*\*: Ez a réteg az overfitting csökkentésére szolgál, véletlenszerűen kinullázva a bemenetek 25%-át (általában).

### További Konvolúciós Rétegek

- \*\*conv2d\_2 (Conv2D)\*\*:

 - \*\*Kimeneti alak\*\*: `(None, 9, 9, 64)`

- \*\*Leírás\*\*: Harmadik konvolúciós réteg, amely 64 darab, 3x3-as szűrőt használ a további jellemzők kinyerésére.

- \*\*conv2d\_3 (Conv2D)\*\*:

- \*\*Kimeneti alak\*\*: `(None, 7, 7, 64)`

- \*\*Leírás\*\*: Negyedik konvolúciós réteg, amely szintén 64 darab, 3x3-as szűrőt alkalmaz.

### Max Pooling és Dropout Rétegek

- \*\*max\_pooling2d\_1 (MaxPooling2D)\*\*:

- \*\*Kimeneti alak\*\*: `(None, 3, 3, 64)`

- \*\*Leírás\*\*: Újabb max pooling réteg, amely 2x2-es ablakokat használ, hogy tovább csökkentse a térbeli méreteket.

- \*\*dropout\_1 (Dropout)\*\*:

- \*\*Kimeneti alak\*\*: `(None, 3, 3, 64)`

- \*\*Leírás\*\*: További dropout réteg az overfitting csökkentésére.

### Flatten Réteg

- \*\*flatten (Flatten)\*\*:

- \*\*Kimeneti alak\*\*: `(None, 576)`

- \*\*Leírás\*\*: Ez a réteg a 3D-s adatokat (3x3x64) egy hosszú, egy dimenziós vektorrá alakítja át (576), amelyet a teljesen összekapcsolt rétegekhez használunk.

### Teljesen Összekapcsolt Rétegek (Dense)

- \*\*dense (Dense)\*\*:

- \*\*Kimeneti alak\*\*: `(None, 256)`

- \*\*Leírás\*\*: Az első teljesen összekapcsolt réteg, amely 256 neuront tartalmaz, és a kinyert jellemzők alapján előfeldolgozást végez.

ábra 2. Modell Architektúrája grafikon

## Modell Trénelése

A modell trénelésének során a fit függvényt használtuk, amely a tréning adatokon végrehajtja a tanulási folyamatot. A trénelés során figyeltük a modell pontosságát és veszteségét az epochok\* során, valamint validációs adatokat is használtunk a modell teljesítményének kiértékelésére. A trénelési folyamat több epochon keresztül zajlott, hogy a modell megfelelően megtanulja a bemenetek és a kimenetek közötti kapcsolatot.

## ****\*****Egy epoch egy teljes iteráció a teljes tréning adathalmaz felett. Ez azt jelenti, hogy az összes tréning adatpontot egyszer feldolgozza a modell, beleértve a súlyok frissítését és a veszteség kiszámítását minden egyes adatpont után

## Modell Felépítése és Trénelése

A modell felépítése az adott képek számától függően körülbelül 40 percet vesz igénybe.

## Adatok Betöltése és Előkészítése

Az adatok betöltését és előkészítését a `load\_training\_data` és `load\_test\_data` függvények végzik.

def load\_training\_data(image\_directory, num\_classes=images\_sum):

data = []

labels = []

for i in range(num\_classes):

path = os.path.join(image\_directory, str(i))

images = os.listdir(path)

for img in images:

try:

img\_path = os.path.join(path, img)

image = Image.open(img\_path)

image = image.resize((30,30))

image = np.array(image)

data.append(image)

labels.append(i)

except:

print(f"Error loading image: {img\_path}")

data = np.array(data)

labels = np.array(labels)

return data, labels

## Adatok Szétválasztása

Az adatokat tréning és validációs adathalmazokra osztjuk a `train\_test\_split` függvény segítségével:

X\_train, X\_val, y\_train, y\_val = train\_test\_split(data, labels, test\_size=0.2, random\_state=42)

y\_train = to\_categorical(y\_train, images\_sum)

y\_val = to\_categorical(y\_val, images\_sum)

## Modell Felépítése

A modell felépítéséhez a `build\_model` függvényt használjuk, amely egy sor konvolúciós, max pooling és dropout rétegből álló Sequential modellt hoz létre:

def build\_model(input\_shape, num\_classes):

model = Sequential([

Conv2D(32, (5, 5), activation='relu', input\_shape=input\_shape),

Conv2D(32, (5, 5), activation='relu'),

MaxPool2D(pool\_size=(2, 2)),

Dropout(0.25),

Conv2D(64, (3, 3), activation='relu'),

Conv2D(64, (3, 3), activation='relu'),

MaxPool2D(pool\_size=(2, 2)),

Dropout(0.25),

Flatten(),

Dense(256, activation='relu'),

Dropout(0.5),

Dense(num\_classes, activation='softmax')

])

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

return model

## Modell Trénelése

A modell trénelése a `fit` függvény segítségével történik, ahol megadjuk a tréning és validációs adathalmazokat, valamint a batch méretet és az epochok számát:

history = model.fit(X\_train, y\_train, batch\_size=32, epochs=15, validation\_data=(X\_val, y\_val))

## Modell Mentése

A betanított modell mentése a `model.save` függvény segítségével történik:

model.save("traffic\_signs\_v10.h5")

## Teszt Adatok Betöltése és Kiértékelése

A teszt adatok betöltésére és kiértékelésére a következő lépésekben kerül sor:

csv\_path = 'Train.csv'

X\_test, y\_test\_labels = load\_test\_data(csv\_path)

y\_test = to\_categorical(y\_test\_labels, images\_sum)

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=2)

print(f"Test accuracy: {test\_acc}, Test loss: {test\_loss}")

## Eredmények Megjelenítése

A tréning eredményeinek megjelenítése grafikonok segítségével történik, amelyek az accuracy és loss metrikákat ábrázolják az epochok függvényében:

plt.figure(0)

plt.plot(history.history['accuracy'], label='training accuracy')

plt.plot(history.history['val\_accuracy'], label='val accuracy')

plt.title('Accuracy')

plt.xlabel('epochs')

plt.ylabel('accuracy')

plt.legend()

plt.show()

plt.figure(1)

plt.plot(history.history['loss'], label='training loss')

plt.plot(history.history['val\_loss'], label='val loss')

plt.title('Loss')

plt.xlabel('epochs')

plt.ylabel('loss')

plt.legend()

plt.show()

## 

ábra 3 A tréning eredményei

## Fő Folyamat

A fő folyamatot a `main` függvény vezérli, amely meghívja a fentebb definiált funkciókat a teljes trénelési és kiértékelési folyamat elvégzésére:

def main():

# Load training data

image\_directory = 'Train'

data, labels = load\_training\_data(image\_directory)

# Splitting the dataset

X\_train, X\_val, y\_train, y\_val = train\_test\_split(data, labels, test\_size=0.2, random\_state=42)

y\_train = to\_categorical(y\_train, images\_sum)

y\_val = to\_categorical(y\_val, images\_sum)

# Building and training the model

model = build\_model(X\_train.shape[1:], images\_sum)

history = model.fit(X\_train, y\_train, batch\_size=32, epochs=15, validation\_data=(X\_val, y\_val))

# Save the model

model.save("traffic\_signs\_v10.h5")

# Load test data

csv\_path = 'Train.csv'

X\_test, y\_test\_labels = load\_test\_data(csv\_path)

y\_test = to\_categorical(y\_test\_labels, images\_sum)

# Evaluate on test data

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=2)

print(f"Test accuracy: {test\_acc}, Test loss: {test\_loss}")

# Plotting training results

plt.figure(0)

plt.plot(history.history['accuracy'], label='training accuracy')

plt.plot(history.history['val\_accuracy'], label='val accuracy')

plt.title('Accuracy')

plt.xlabel('epochs')

plt.ylabel('accuracy')

plt.legend()

plt.show()

plt.figure(1)

plt.plot(history.history['loss'], label='training loss')

plt.plot(history.history['val\_loss'], label='val loss')

plt.title('Loss')

plt.xlabel('epochs')

plt.ylabel('loss')

plt.legend()

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

main()

# 4. Tesztelés

## Teszt Adatok Betöltése és Kiértékelése

A teszt adatok betöltésére és kiértékelésére a fent említett lépéseket követjük. Az adatok betöltése után a modell tesztelése és az eredmények kiértékelése történik, amely során megkapjuk a teszt pontosságot és veszteséget.

## Tesztelési Eredmények

A tesztelési folyamat során mért pontosság és veszteség értékek fontosak a modell teljesítményének kiértékeléséhez. A következő metrikákat mérjük:

- **Pontosság (Accuracy)**: Az osztályok helyes felismerésének aránya.

- **Veszteség (Loss):** A modell hibájának mértéke.

- **Validációs Pontosság (Val\_accuracy):** A validációs adatokon mért pontosság.

- **Validációs Veszteség (Val\_loss):** A validációs adatokon mért veszteség.

# 5. Felhasználói Leírás

## Felhasználói Leírás

A grafikus felhasználói felület (GUI) indításához futtasd a fő Python szkriptet a projekt gyökérkönyvtárában:

python gui.py

A GUI-n keresztül töltheted fel az útjelző táblák képeit, amelyeket a rendszer elemez, és azonnali visszajelzést ad a felismerés eredményéről. A felhasználói felület intuitív és könnyen használható, lehetővé téve a felhasználók számára a különböző funkciók egyszerű elérését.

## GUI Kód

Az alábbiakban bemutatjuk a GUI kódját, amely a modell betöltését és az útjelző táblák felismerését végzi:

import tkinter as tk

from tkinter import filedialog

from tkinter import \*

from PIL import ImageTk, Image

import numpy as np

from keras.models import load\_model

# Load the trained model to classify signs

model = load\_model('traffic\_signs\_v10.h5')

# Dictionary to label all traffic signs class.

classes = {

1: 'Sebességkorlátozás (20km/h)',

2: 'Sebességkorlátozás (30km/h)',

3: 'Sebességkorlátozás (50km/h)',

4: 'Sebességkorlátozás (60km/h)',

5: 'Sebességkorlátozás (70km/h)',

6: 'Sebességkorlátozás (80km/h)',

7: 'Sebességkorlátozás Vége (80km/h)',

8: 'Sebességkorlátozás (100km/h)',

9: 'Sebességkorlátozás (120km/h)',

10: 'Előzni tilos',

11: '3,5 tonnánál nehezebb járművek előzése tilos',

12: 'Elsőbbségadás kötelező kereszteződésben',

13: 'Főútvonal',

14: 'Adjon elsőbbséget',

15: 'Állj, minden járműnek meg kell állnia',

16: 'Járművek behajtása tilos',

17: '3,5 tonnánál nehezebb járművek behajtása tilos',

18: 'Behajtani tilos',

19: 'Fokozott óvatosság',

20: 'Veszélyes bal kanyar',

21: 'Veszélyes jobb kanyar',

22: 'Kettős kanyar',

23: 'Buckás út',

24: 'Csúszós út',

25: 'Út szűkület jobbról',

26: 'Útépítési munkálatok',

27: 'Közlekedési lámpa',

28: 'Gyalogosok',

29: 'Gyermekátkelő',

30: 'Kerékpár átkelő',

31: 'Jeges/havas út',

32: 'Vadállat-átkelő',

33: 'Sebesség- és előzési korlátozás vége',

34: 'Jobbra kanyarodj',

35: 'Balra kanyarodj',

36: 'Csak egyenesen',

37: 'Egyenesen vagy jobbra',

38: 'Egyenesen vagy balra',

39: 'Tarts jobbra',

40: 'Tarts balra',

41: 'Körforgalom',

42: 'Előzési tilalom vége',

43: '3,5 tonnánál nehezebb járművek előzési tilalmának vége',

44: 'Sebességkorlátozás (90km/h)',

45: 'Gyalogos átkelőhely - Járda',

46: 'Sebességkorlátozás (5km/h)',

47: 'Sebességkorlátozás (15km/h)',

48: 'Sebességkorlátozás (40km/h)',

49: 'Jobbra és előre kanyarodni tilos',

50: 'Gyalogosátkelőhely',

51: 'Egyenesen haladni tilos',

52: 'Jobbra kanyarodni tilos',

53: 'Jobbra és balra kanyarodni tilos',

54: 'Balra kanyarodni tilos',

55: 'Előzni tilos',

56: 'Megfordulni tilos',

57: 'Gépjárművel behajtani tilos',

58: 'Dudálni tilos',

59: 'Sebességkorlátozás Vége (40km/h)',

60: 'Sebességkorlátozás Vége (50km/h)',

61: 'Kötelező haladási irány: balra vagy jobbra',

62: 'Autópálya',

63: 'Kerékpár pálya',

64: 'Megfordulás kötelező',

65: 'Veszélyt jelző tábla vagy Általános veszély'

}

# Initialise GUI

top = tk.Tk()

top.geometry('800x600')

top.title('Traffic sign classification')

top.configure(background='#CDCDCD')

label = Label(top, background='#CDCDCD', font=('arial', 15, 'bold'))

sign\_image = Label(top)

def classify(file\_path):

global label\_packed

image = Image.open(file\_path)

image = image.convert('RGB') # Convert the image to RGB

image = image.resize((30, 30))

image = np.expand\_dims(image, axis=0)

image = np.array(image)

prediction = model.predict([image])

pred = np.argmax(prediction, axis=1)[0] # Get the index of the max value

sign = classes[pred + 1]

print(sign)

label.configure(foreground='#011638', text=sign)

def show\_classify\_button(file\_path):

classify\_b = Button(top, text="Kép szkennelése", command=lambda: classify(file\_path), padx=10, pady=5)

classify\_b.configure(background='#364156', foreground='white', font=('arial', 10, 'bold'))

classify\_b.place(relx=0.79, rely=0.46)

def upload\_image():

try:

file\_path = filedialog.askopenfilename()

uploaded = Image.open(file\_path)

uploaded.thumbnail(((top.winfo\_width()/2.25), (top.winfo\_height()/2.25)))

im = ImageTk.PhotoImage(uploaded)

sign\_image.configure(image=im)

sign\_image.image = im

label.configure(text='')

show\_classify\_button(file\_path)

except:

pass

upload = Button(top, text="Tölts fel egy képet", command=upload\_image, padx=10, pady=5)

upload.configure(background='#364156', foreground='white', font=('arial', 10, 'bold'))

upload.pack(side=BOTTOM, pady=50)

sign\_image.pack(side=BOTTOM, expand=True)

label.pack(side=BOTTOM, expand=True)

heading = Label(top, text="Útjelző tábla jelentése", pady=20, font=('arial', 20, 'bold'))

heading.configure(background='#CDCDCD', foreground='#364156')

heading.pack()

# Force the window to update to calculate dimensions

top.update()

# Add this block where you initialize your GUI components, after defining `sign\_image`

placeholder = Image.open('placeholder.png') # Load the placeholder image

placeholder.thumbnail(((top.winfo\_width()/2.25), (top.winfo\_height()/2.25)))

placeholder\_image = ImageTk.PhotoImage(placeholder)

sign\_image.configure(image=placeholder\_image)

sign\_image.image = placeholder\_image

top.mainloop()

## Részletek a Kódból

1. Modell Betöltése:

model = load\_model('traffic\_signs\_v10.h5')

2. Osztályok Definiálása:

- A `classes` szótár tartalmazza az összes útjelző tábla osztályát és azok magyar nyelvű leírását.

3. Kép Osztályozása:

- A `classify` függvény végzi a kép osztályozását, és frissíti a GUI-t az eredménnyel.

def classify(file\_path):

prediction = model.predict([image])

pred = np.argmax(prediction, axis=1)[0]

sign = classes[pred + 1]

...

label.configure(foreground='#011638', text=sign)

4. Kép Feltöltése:

- Az `upload\_image` függvény lehetővé teszi a felhasználó számára, hogy képet töltsön fel, és

megjeleníti azt a GUI-n.

def upload\_image():

...

file\_path = filedialog.askopenfilename()

...

show\_classify\_button(file\_path)

5. \*\*GUI Beállítások\*\*:

- A GUI beállításait a Tkinter könyvtár segítségével hoztuk létre, ahol gombok, címkék és képek jelennek meg.

A felhasználók könnyedén feltölthetnek képeket, amelyek alapján a rendszer azonnali visszajelzést ad az útjelző táblák felismeréséről. A magyar nyelvű osztályok segítenek abban, hogy a felhasználók pontosan megértsék az eredményeket.

# 6. Jövőbeli lehetőségek és ötletek a modell kapcsán

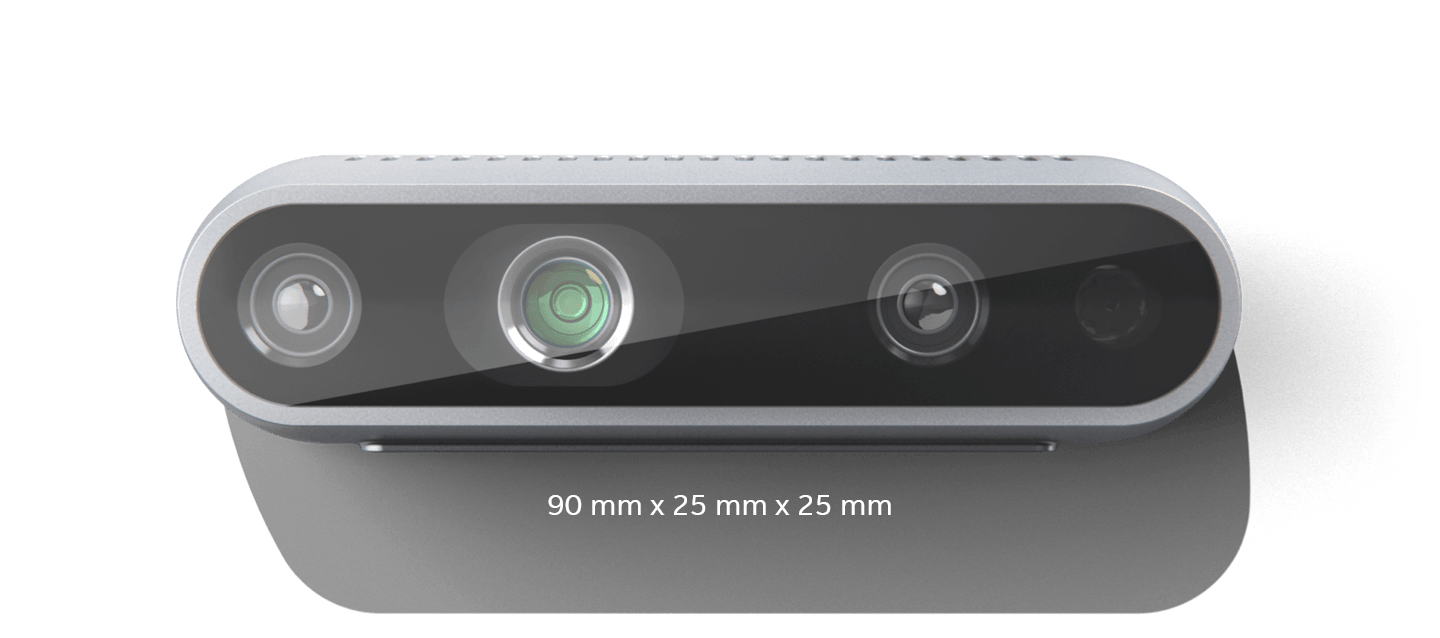
## NVIDIA TensorFlow GPU Használata

Az egyik jövőbeli lehetőség a modell futtatásának optimalizálása az NVIDIA TensorFlow GPU segítségével. A TensorFlow GPU verziójának használata számos előnnyel járhat:

1. Gyorsabb Modell Betanítás: A GPU-k párhuzamos feldolgozási képességei lehetővé teszik a modell gyorsabb betanítását, különösen nagyobb adathalmazok esetén.
2. Valós Idejű Feldolgozás: Az útjelző táblák valós idejű felismerése gyorsabbá válik, ami különösen fontos lehet az élő kamera képek feldolgozásánál.
3. Nagyobb Hatékonyság: Az NVIDIA GPU-k speciálisan optimalizáltak mélytanulási feladatokra, így a számítási hatékonyság növekszik, csökkentve az energiafogyasztást és a feldolgozási időt.

A TensorFlow GPU beállítása során szükség van az NVIDIA CUDA és cuDNN könyvtárak telepítésére, valamint a TensorFlow megfelelő verziójának használatára.

## 3D Kamerák Használata

A jövőbeni fejlesztési tervek között szerepel az Intel RealSense 3D kamerák vagy egyéb 3D kamerák integrálása. Ezek a kamerák lehetővé teszik a mélységi információk és a színes képek együttes használatát, amely számos előnnyel járhat: 

ábra 4 Intel Realsense 3D kamera

1. **Pontosság Növelése**: A 3D információk használata lehetővé teszi a pontosabb tárgyfelismerést és a különböző tárgyak könnyebb megkülönböztetését.

2. **Valós Idejű Felismerés**: Az élő kamera kép alapján történő valós idejű táblafelismerés segíthet a közlekedési helyzetek gyorsabb és pontosabb értelmezésében.

3. **Környezet Felismerése**: A mélységi érzékelés lehetővé teszi a kamera számára, hogy a környezet háromdimenziós képét is figyelembe vegye, ami hasznos lehet például az autonóm járművek navigációjánál.

## GPU Használata az Élő Kamera Kép Feldolgozásához

A 3D kamerák és a GPU-k együttes használata további lehetőségeket nyit meg:

1. Valós Idejű Objektumfelismerés: Az NVIDIA GPU-k párhuzamos feldolgozási képességeinek köszönhetően a valós idejű objektumfelismerés hatékonyabbá válik, ami fontos az élő videó stream feldolgozásánál.

2. Komplex Számítások Gyors Végrehajtása: A mélységi és színes képek együttes feldolgozása jelentős számítási kapacitást igényel, amelyet a GPU-k képesek hatékonyan kezelni.

3. Adaptív Rendszerek: Az élő kamera képek valós idejű feldolgozása lehetővé teszi adaptív rendszerek kialakítását, amelyek valós időben képesek reagálni a változó környezeti feltételekre.

## Összefoglalás

A jövőbeni fejlesztések során az NVIDIA TensorFlow GPU és a 3D kamerák használata jelentős előrelépést jelenthet a modell teljesítményében és funkcionalitásában. Ezek az újítások lehetővé teszik a gyorsabb, pontosabb és hatékonyabb táblafelismerést, különösen valós idejű alkalmazásokban.

# 7. Irodalomjegyzék

- [[TensorFlow hivatalos weboldala](https://www.tensorflow.org)](https://www.tensorflow.org/)

- [[OpenCV dokumentáció](https://opencv.org)](https://opencv.org/)

- [[PIL (Pillow) dokumentáció](https://pillow.readthedocs.io)](https://pillow.readthedocs.io/)

- [[Pytesseract GitHub oldala](https://github.com/madmike/ocr-Template-matching)](https://github.com/madmike/ocr-Template-matching)

- [[GTSRB (German Traffic Sign Recognition Benchmark](https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign)) dataset a Kaggle-on](https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign)

- [[Chinese Traffic Sign Database](https://nlpr.ia.ac.cn/pal/trafficdata/recognition.html)](https://nlpr.ia.ac.cn/pal/trafficdata/recognition.html)

- [[DFG Resources](https://www.vicos.si/resources/dfg)](https://www.vicos.si/resources/dfg/)

Ez a dokumentáció részletesen bemutatja az útjelző táblák felismerésére irányuló projektet, beleértve a szükséges elméleti hátteret, a megvalósítás részleteit, a tesztelési folyamatot és a felhasználói útmutatót. A projekt célja, hogy egy olyan rendszert hozzon létre, amely képes pontosan felismerni és digitalizálni az útjelző táblák információit, támogatva ezzel a közlekedésbiztonságot és a vezetők döntéshozatalát.

Tartalom

[1. Bevezetés 1](#_Toc168852551)

[Megoldandó Feladat Kifejtése 1](#_Toc168852552)

[2. Megoldáshoz Szükséges Elméleti Háttér 1](#_Toc168852553)

[Gépi Látás és Képfeldolgozás 1](#_Toc168852554)

[TensorFlow és Mély Tanulás 1](#_Toc168852555)

[3. A Megvalósítás Terve és Kivitelezése 2](#_Toc168852556)

[Adatgyűjtés és Előkészítés 2](#_Toc168852557)

[Adatgyűjtés 2](#_Toc168852558)

[Adatok Előkészítése 3](#_Toc168852559)

[Adat Argumentálás 3](#_Toc168852560)

[Zaj hozzáadása: 3](#_Toc168852561)

[Modell Felépítése és Trénelése 3](#_Toc168852562)

[Adatok Betöltése és Előkészítése 7](#_Toc168852563)

[Modell Architektúra 8](#_Toc168852564)

[Konvolúciós Rétegek (Conv2D) 8](#_Toc168852565)

[Max Pooling Rétegek (MaxPooling2D) 8](#_Toc168852566)

[Dropout Rétegek (Dropout) 8](#_Toc168852567)

[További Konvolúciós Rétegek 9](#_Toc168852568)

[Max Pooling és Dropout Rétegek 9](#_Toc168852569)

[Flatten Réteg 9](#_Toc168852570)

[Teljesen Összekapcsolt Rétegek (Dense) 9](#_Toc168852571)

[Modell Trénelése 10](#_Toc168852572)

[\*Egy epoch egy teljes iteráció a teljes tréning adathalmaz felett. Ez azt jelenti, hogy az összes tréning adatpontot egyszer feldolgozza a modell, beleértve a súlyok frissítését és a veszteség kiszámítását minden egyes adatpont után 10](#_Toc168852573)

[Modell Felépítése és Trénelése 10](#_Toc168852574)

[Adatok Betöltése és Előkészítése 10](#_Toc168852575)

[Adatok Szétválasztása 10](#_Toc168852576)

[Modell Felépítése 11](#_Toc168852577)

[Modell Trénelése 11](#_Toc168852578)

[Modell Mentése 11](#_Toc168852579)

[Teszt Adatok Betöltése és Kiértékelése 11](#_Toc168852580)

[Eredmények Megjelenítése 12](#_Toc168852581)

[Fő Folyamat 12](#_Toc168852583)

[4. Tesztelés 13](#_Toc168852584)

[Teszt Adatok Betöltése és Kiértékelése 13](#_Toc168852585)

[Tesztelési Eredmények 13](#_Toc168852586)

[5. Felhasználói Leírás 14](#_Toc168852587)

[Felhasználói Leírás 14](#_Toc168852588)

[GUI Kód 14](#_Toc168852589)

[Részletek a Kódból 16](#_Toc168852590)

[6. Jövőbeli lehetőségek és ötletek a modell kapcsán 17](#_Toc168852591)

[NVIDIA TensorFlow GPU Használata 17](#_Toc168852592)

[3D Kamerák Használata 18](#_Toc168852593)

[GPU Használata az Élő Kamera Kép Feldolgozásához 18](#_Toc168852594)

[Összefoglalás 19](#_Toc168852595)

[7. Irodalomjegyzék 19](#_Toc168852596)