Projekt Dokumentáció: Útjelző Táblák Felismerése OCR és TensorFlow Technológiával

Tartalomjegyzék

---

1. 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 szöveges információk digitalizálására. A modern képfeldolgozási technikák és gépi tanulási modellek alkalmazásával, mint például a TensorFlow, a rendszer elemzi a közlekedési jelzéseket, azonosítja a szimbólumokat és a szöveges utasításokat. Ezáltal támogatja a vezetők tájékozódását és döntéshozatalát, növelve a közlekedésbiztonságot.

Megoldandó Feladat Kifejtése

A projekt fő célja egy olyan modell létrehozása, amely képes felismerni és azonosítani különböző útjelző táblákat, valamint digitalizálni a rajtuk található szöveges információkat. 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.

Optikai Karakterfelismerés (OCR)

Az OCR (Optical Character Recognition) technológia lehetővé teszi a nyomtatott vagy kézzel írott szöveg digitális formába történő átalakítását. Az OCR algoritmusok képesek felismerni és értelmezni a karaktereket és szavakat különböző formátumú dokumentumokból és képekből. A modern OCR rendszerek általában gépi tanulási modelleket, például neurális hálózatokat használnak a karakterek pontos felismerésére.

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.

Konvolúciós Neurális Hálózatok (CNN)

A konvolúciós neurális hálózatok (CNN-ek) a mély tanulás egy speciális típusát képezik, amelyeket kifejezetten a képfeldolgozási feladatokhoz terveztek. A CNN-ek rétegei között megtalálhatók a konvolúciós rétegek, a pooling rétegek és a teljesen összekapcsolt rétegek, amelyek együttműködve képesek a képek jellemzőit hatékonyan kinyerni és feldolgozni.

3. A Megvalósítás Terve és Kivitelezése

Adatgyűjtés és Előkészítés
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 használtunk fel, beleértve a nyilvánosan elérhető adatbázisokat, mint például a GTSRB (German Traffic Sign Recognition Benchmark) és a Chinese Traffic Sign Database. Emellett saját képeket is készítettünk különböző fényviszonyok és perspektívák alatt, hogy biztosítsuk a modell általánosítási képességét.

Az adatgyűjtés során különös figyelmet fordítottunk arra, hogy minden tábla különböző nézőpontokból és környezetben legyen ábrázolva. Ez magában foglalta a nappali és éjszakai képeket, különböző időjárási körülményeket (pl. eső, köd, napsütés), valamint különböző távolságokat és szögeket. Ezek az eltérések segítenek a modellnek abban, hogy robusztusabbá váljon és jobban teljesítsen a valós környezetben.

Adatok Előkészítése
Az adatok előkészítése során a képeket először normalizáltuk, 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 konvertáltuk, 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 alkalmaztunk 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: Képekhez véletlenszerű 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.

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:

Konvolúciós rétegek (Conv2D): A képi jellemzők kinyerésére szolgálnak. Több konvolúciós réteget alkalmaztunk különböző szűrőméretekkel és szűrőszámokkal.
Max Pooling rétegek (MaxPooling2D): A térbeli méretek csökkentésére és a legfontosabb jellemzők kiválasztására szolgálnak.
Dropout rétegek (Dropout): Az overfitting csökkentésére szolgálnak, véletlenszerűen kinullázva a bemenetek egy részét.
Teljesen összekapcsolt rétegek (Dense): A kinyert jellemzők alapján történő osztályozásra szolgálnak. Az utolsó réteg softmax aktivációs függvényt használ az osztály valószínűségének meghatározására.
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.
A projekt első lépése a megfelelő mennyiségű kép összegyűjtése és előkészítése. Minden egyes útjelző táblához 20 képet kell összegyűjteni, amelyek különböző perspektívákból, fényviszonyok mellett és különböző torzításokkal ábrázolják ugyanazt a táblát.

Beállítások Konfigurálása

A `settings.py` fájlban állítsd be az adott útjelző tábla azonosítóját (`class_id`) és a képek számát (`num_images`):

```python
class_id = '64'  # Az adott útjelző tábla azonosítója
num_images = 20  # A szükséges képek száma
```

Adat Argumentálás

Az argumentálás célja, hogy a meglévő képekből további példányokat hozzunk létre különböző transzformációk (eltolás, fényerő, forgatás) alkalmazásával. A `data.py` fájlban a következő paraméterek alapján történik az argumentálás:

```python
csv_data = [
    [28, 25, 5, 5, 23, 20, 20],
    [30, 27, 5, 5, 25, 22, 20],
    ...
    [99, 88, 9, 8, 91, 81, 20]
]
```

Ezek a paraméterek meghatározzák az egyes képek átméretezését és transzformációit.

Képek Előkészítése és Tárolása

Az összegyűjtött képeket elő kell készíteni a trénelési folyamat számára. Ehhez a `trainer.py` szkriptet használjuk, amely elvégzi a képek feldolgozását és a megfelelő helyre történő mentését.

Képek Betöltése és Ellenőrzése

Az alábbi funkciók ellenőrzik, hogy az összes szükséges kép elérhető-e, és előkészítik a képeket a feldolgozásra.

```python
def validate_image_paths(image_paths):
    missing_images = [img for img in image_paths if not os.path.exists(img)]
    if missing_images:
        print("The following images are missing and will be skipped:")
        for missing in missing_images:
            print(missing)
    return [img for img in image_paths if os.path.exists(img)]
```

Képek Feldolgozása és Mentése

A képek feldolgozása és mentése során a szkript átméretezi és elmenti a képeket a megfelelő könyvtárstruktúrában, valamint frissíti a CSV fájlokat a szükséges információkkal.

```python
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'])

    for idx, image_path in enumerate(valid_image_paths):
        input_image = cv2.imread(image_path)
        if input_image is None:
            print(f"Failed to load image: {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 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:  # Reset counter and increase set number after reaching limit
                image_counter = 0
                set_counter += 1
```

`

Trénelés Indítása

Miután az összes képet előkészítettük és elmentettük, a trénelési folyamatot a `trainer.py` szkript

futtatásával indíthatjuk el:

```bash
python trainer.py
```

Modell Felépítése és Trénelése

A modell felépítése az adott képek számától függően körülbelül 15 percet vesz igénybe.

Adatok Betöltése és Előkészítése

Az adatok betöltését és előkészítését a `load_training_data` és `load_test_data` függvények végzik.

```python
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:

```python
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:

```python
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:

```python
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:

```python
model.save("traffic_signs_v10.h5")
```

Teszt Adatok Betöltése és Kiértékelése

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

```python
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:

```python
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()
```

Fő Folyamat

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

```python
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.

A mért eredmények grafikonokon történő megjelenítése segít a modell fejlődésének nyomon követésében és a tréning folyamat kiértékelésében.

5. Felhasználói Leírás

A GUI indításához futtasd a fő Python szkriptet a projekt gyökérkönyvtárában:

```bash
python main.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.

6. Irodalomjegyzék

- [TensorFlow hivatalos weboldala](https://www.tensorflow.org/)
- [OpenCV dokumentáció](https://opencv.org/)
- [PIL (Pillow) dokumentáció](https://pillow.readthedocs.io/)
- [Pytesseract GitHub oldala](https://github.com/madmike/ocr-Template-matching)

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

al/trafficdata/recognition.html)
- [DFG Resources](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.