Kézzel írt számjegyek felismerése

# Bevezetés

A kézzel írt számjegyek felismerése egy fontos probléma az informatikában és a gépi tanulásban. Az emberek számára könnyű felismerni a kézzel írt számjegyeket, de a számítógépek számára ez sokkal nehezebb feladat. A kézzel írt számjegyek felismerése az optikai karakterfelismerés (OCR) egyik típusa, amelynek célja a nyomtatott vagy kézzel írott karakterek digitális formára való átalakítása.

A kézzel írt számjegyek felismeréséhez általában a gépi tanulás és a neurális hálózatok alkalmazását használják. A neurális hálózatok olyan algoritmusok, amelyeket az agy működéséből inspirálva fejlesztettek ki. A hálózatok általában több rétegből állnak, amelyek mindegyike előfeldolgozza a bemeneti adatokat. Az utolsó réteg kimenete a végső eredményt adja.

Az első lépés a kézzel írt számjegyek felismerésében a képek előfeldolgozása. Az előfeldolgozás során a képek általában normalizálódnak, a zajt kiszűrik, és a képek mérete is általában normalizálódik. A normalizálás segíti a neurális hálózatot abban, hogy könnyebben felismerje a számjegyeket, míg a zajszűrés csökkenti a hibázás esélyét. A méretnormalizálás lehetővé teszi, hogy az összes bemenet azonos méretű legyen.

A képek előfeldolgozása után a neurális hálózatot betanítják a számjegyek felismerésére. Az adathalmaz általában több ezer vagy több tízezer képet tartalmaz, amelyek mindegyike címkézett, azaz ismerjük azok helyes osztályozását. A képek és az osztályozások összerendeléséből álló adathalmaz felhasználásával a neurális hálózatot betanítják, hogy felismerje a kézzel írt számjegyeket.

A neurális hálózatok betanítása után a rendszer képes lesz felismerni a kézzel írt számjegyeket, amelyeket akár kézírásos beviteli mezőben, akár olyan formában, mint például szkennelt dokumentumokon találhatunk meg. A felismerési folyamat a következő lépéseket tartalmazza: a bemeneti kép előfeldolgozása (amely magában foglalhatja a zajszűrést, normalizálást és élek keresését), majd a bemenet átalakítása olyan formátumra, amelyet a neurális hálózat megért, például tenzorokra.

A hálózat ezután végrehajtja a műveleteket az átalakított bemeneten, amelynek eredményeként meghatározza a bemenetben található számjegy osztályát. Az osztálytól függően a rendszer visszatérhet az osztály címkevel, a valószínűségi értékkel, vagy az osztály által reprezentált számjegy képével.

Fontos megjegyezni, hogy a kézzel írt számjegyek felismerése nem mindig tökéletes. A hálózat teljesítménye függ a bemeneti adatok minőségétől, a hálózat architektúrájától, a betanítási adathalmaz méretétől és minőségétől, valamint a hiperparaméterek beállításától. A hiperparaméterek beállításának finomhangolása segíthet a hálózat teljesítményének javításában.

Összefoglalva, a kézzel írt számjegyek felismerése komplex probléma, amelyhez a gépi tanulás és a neurális hálózatok alkalmazását használják. A felismerési folyamat magában foglalja az előfeldolgozási lépéseket, a neurális hálózat betanítását, valamint a felismerési folyamat végrehajtását. A hálózatok teljesítménye függ a bemeneti adatok minőségétől és a betanítási folyamat beállításaitól. A kézzel írt számjegyek felismerése sok gyakorlati alkalmazásban hasznos lehet, például a pénzügyi dokumentumok, űrlapok vagy az okostelefonokban található kézírásos jegyzetek szkennelése során.

# Történelmi előzmények

A gépi látás története egészen az 1960-as évekig nyúlik vissza. Ekkoriban kezdtek megjelenni az első kísérletek a számítógép által végzett képfeldolgozásra és elemzésre. Az első rendszerek általában olyan egyszerű feladatokat végeztek, mint például az objektumok körvonalának meghatározása.

Az 1970-es években a kutatók elkezdtek dolgozni az objektumok automatikus azonosításával és klasszifikációjával kapcsolatos problémákon. Az általuk alkalmazott módszerek között megtalálhatók az alapszintű geometriai módszerek, például a Hough-transzformáció, amelyek lehetővé tették az egyenesek és körök automatikus detektálását.

Az 1980-as években azonban a neurális hálózatok alkalmazása robbanásszerű növekedést mutatott. Ebben az időszakban a kutatók felfedezték, hogy az összekapcsolt neuronokból álló hálózatok lehetnek képesek tanulni a képfeldolgozási feladatokat. Ezek a rendszerek azonban koruk hardvereinek és szoftvereinek korlátai miatt korlátozott eredményeket produkáltak.

Az 1990-es években azonban a számítógépek és a grafikus kártyák teljesítményének növekedése lehetővé tette a nagyobb és bonyolultabb modellek futtatását. Ebben az időszakban jelentek meg az első Convolutional Neural Network (CNN) alapú gépi látásrendszerek. Ezek a rendszerek lehetővé tették az objektumok pontosabb felismerését és az olyan bonyolultabb feladatok elvégzését, mint például az arcfelismerés.

Az 2000-es években a kutatók széles körűen használták a gépi látást a biometrikus azonosítás, az autonóm járművek, a robotika és a biztonsági rendszerek területén. A gépi látás jelenleg az egyik legdinamikusabban fejlődő területe az AI-nak, és számos ipari és kutatási alkalmazásban használják.

# Megvalósításhoz használható ingyenes programok

Számos ingyenesen hozzáférhető és hatékony program érhető el a kézzel írt számjegyek felismerésére. Néhány ismert program a következő:

1. TensorFlow: Ez egy nyílt forráskódú gépi tanulási keretrendszer, amelyben számos algoritmus található, ideértve a kézzel írt számjegyek felismerésére használt neurális hálózatokat is.
2. Keras: Ez egy magas szintű neurális hálózat API, amely lehetővé teszi a gyors prototípuskészítést és a könnyű átvitelt más keretrendszerekkel. A Keras támogatja a TensorFlow, a Theano és a CNTK háttérként való futtatását is.
3. OpenCV: Ez egy nyílt forráskódú számítógépes látási könyvtár, amely számos előfeldolgozási és képfeldolgozási funkcióval rendelkezik. Az OpenCV-t a képek előfeldolgozására lehet használni, például zajszűrésre, élszűrésre és normalizálásra.
4. Scikit-learn: Ez egy Python alapú gépi tanulási könyvtár, amely számos osztályozó és regressziós algoritmust tartalmaz. A Scikit-learn-t az adatok előfeldolgozására és a gépi tanulási algoritmusok betanítására is használhatjuk.
5. Tesseract: Ez egy nyílt forráskódú OCR (Optical Character Recognition) motor, amely a szövegfelismerési feladatokon túl képes a kézzel írott számjegyek felismerésére is.

Ezek csak néhány példa az ingyenesen hozzáférhető és nagy hatásfokú programokra a kézzel írt számjegyek felismeréséhez. A kiválasztott program attól függ, hogy milyen platformot használunk, milyen célra használjuk, és mennyire ismerjük az adott programozási nyelvet.

1. TensorFlow: TensorFlow egy nyílt forráskódú szoftverkönyvtár, amelyet főként neurális hálózatok építésére és futtatására használnak. A TensorFlow-t 2015-ben fejlesztette ki a Google, és azóta széles körben használják a mélytanulás és a gépi tanulás területén. A TensorFlow rendkívül hatékony a nagy adathalmazokon történő gépi tanulási feladatokban, és számos előre elkészített modult tartalmaz, amelyeket felhasználhatunk a kézzel írt számjegyek felismerésére. Az egyik ilyen modul például a Keras, amely a TensorFlow API-jának magasabb szintű felhasználóbarát változata.
2. Keras: Keras egy magas szintű neurális hálózatok API-ja, amelyet a TensorFlow könyvtár támogat. A Keras nagyon könnyen használható, és lehetővé teszi a neurális hálózatok egyszerű és gyors építését. A Keras támogatja a konvolúciós neurális hálózatokat (CNN), amelyek kiválóan alkalmasak a képfelismerési feladatokra, például a kézzel írt számjegyek felismerésére. A Keras-szal gyorsan és hatékonyan felépíthetjük a kézzel írt számjegyek felismerésére szolgáló neurális hálózatot, és könnyen finomhangolhatjuk a paramétereit az optimális eredmények elérése érdekében.
3. Tesseract: Tesseract egy nyílt forráskódú OCR (Optical Character Recognition) motor, amely a szövegfelismerési feladatokon túl képes a kézzel írott számjegyek felismerésére is. A Tesseract-t 1985-ben fejlesztette ki a Hewlett-Packard Labs, majd 2005-ben átvette a Google. Azóta folyamatosan fejlesztik és karbantartják. Az algoritmus alapvetően egy optikai karakterfelismerő, amely a beolvasott szöveget képes szöveges karakterekre konvertálni. Az OCR szövegfelismerő motorral szemben, amelynek fő feladata a nyomtatott vagy gépelt szöveg digitalizálása, a Tesseract képes az emberi írási stílus különböző változatainak felismerésére is.
4. OpenCV: OpenCV (Open Source Computer Vision Library) egy nyílt forráskódú számítógépes látási könyvtár, amely számos előfeldolgozási és képfeldolgozási funkcióval rendelkezik. Az OpenCV-t a képek előfeldolgozására lehet használni, például zajszűrésre, élszűrésre és normalizálásra. Az OpenCV-nek számos programozási nyelvben (C++, Python, Java stb.) létezik API-ja, így könnyen használható a számítógépes látással kapcsolatos projektekben. Az OpenCV fontos eleme a képfeldolgozó és számítógépes látási rendszereknek, és nagyon hatékony lehet a kézzel írt számjegyek felismerésére.
5. Scikit-learn: Scikit-learn egy Python alapú gépi tanulási könyvtár, amely számos osztályozó és regressziós algoritmust tartalmaz. A Scikit-learn-t az adatok előfeldolgozására és a gépi tanulási algoritmusok betanítására is használhatjuk. A gépi tanulás alapelve az adatok alapján való automatikus döntéshozatal, és a Scikit-learn ebben nyújt hatékony eszközöket. A Scikit-learn-t sokan használják a kézzel írt számjegyek felismerésére, mivel számos osztályozó algoritmust tartalmaz, például a döntési fát, a Random Forest-ot és a neurális hálózatokat is. A döntési fa egy olyan osztályozó algoritmus, amely az adatokat egy fára hasonlító szerkezetben tárolja, és a döntéshozatalt egyszerű szabályok alkalmazásával végzi. A döntési fa általában könnyen értelmezhető, és jól teljesít kisebb adathalmazokon. A Random Forest egy olyan osztályozó algoritmus, amely több döntési fát használ a predikcióhoz, és az eredményeket az összes fa által számított eredmények átlagolásával vagy többségi szavazással állítja elő. A Random Forest hatékonyan kezeli az overfittinget, és kiválóan teljesít nagyobb adathalmazokon. A Scikit-learn további algoritmusai között találhatunk neurális hálózatokat is, amelyeket különféle konfigurációkkal és paraméterekkel finomhangolhatunk az optimális eredmények elérése érdekében. A Scikit-learn használata a kézzel írt számjegyek felismerésére gyakran a feladat előfeldolgozási és a dimenziócsökkentési lépéseket is magában foglalja, például a PCA-t (Principal Component Analysis), amely lehetővé teszi az adatok dimenzióinak csökkentését a hatékonyabb gépi tanulás érdekében. Összességében a Scikit-learn nagyon sokoldalú és hatékony könyvtár a gépi tanulásra és az adatok előfeldolgozására. Az osztályozó és regressziós algoritmusokon túl a Scikit-learn tartalmaz klaszterező és dimenziócsökkentő algoritmusokat is, amelyek fontosak lehetnek a kézzel írt számjegyek felismeréséhez használt adathalmazok előkészítése során.

Összességében tehát többféle ingyenesen hozzáférhető program is rendelkezésre áll a kézzel írt számjegyek felismerésére. A Tesseract-t, az OpenCV-t és a Scikit-learn-t széles körben használják a feladat megoldására, a TensorFlow és a Keras pedig kiváló lehetőséget nyújtanak a mélytanulás és a neurális hálózatok építésére és használatára. A megfelelő program kiválasztása függ az adott feladattól, a rendelkezésre álló adatoktól és a felhasználói preferenciáktól.

# Professzionális megoldások

A kézzel írt számjegyek felismerése az elmúlt években nagy fejlődésen ment keresztül, és ma már számos nagyon hatékony és professzionális megoldás érhető el az adatok felismerésére. Ezek közül néhányat az alábbiakban sorolok fel:

1. Google Cloud Vision API: A Google Cloud Vision API egy nagyon hatékony és megbízható felhőalapú képfeldolgozási szolgáltatás, amely a Google keresőmotorjában is használatos. A Vision API lehetővé teszi a képek automatikus felismerését és azokban található szövegek kiemelését is.
2. Amazon Textract: Az Amazon Textract egy másik felhőalapú szolgáltatás, amely nagyon hatékonyan dolgozik az OCR (optikai karakterfelismerés) területén. Az Amazon Textract képes azonosítani és kiemelni a képekben található szövegeket, és azokat formázva szöveges dokumentummá konvertálni.
3. Microsoft Azure Cognitive Services: A Microsoft Azure Cognitive Services egy nagyon rugalmas és hatékony képfeldolgozási szolgáltatás, amely az OCR mellett számos egyéb funkcióval is rendelkezik. Az Azure Cognitive Services lehetővé teszi a képek automatikus kategorizálását, szövegfeldolgozást, arc- és észlelést, valamint a gépi tanulás funkcióit is támogatja.
4. TensorFlow: A TensorFlow egy nyílt forráskódú gépi tanulási platform, amely lehetővé teszi az adatok felismerését és osztályozását. A TensorFlow különösen hatékony az adatok előfeldolgozása és a neurális hálózatok használata során. A TensorFlow számos előre definiált modellt tartalmaz, amelyek között szerepelnek olyanok, mint például a Convolutional Neural Network (CNN), amely kiválóan alkalmas a kézzel írt számjegyek felismerésére.
5. Keras: A Keras egy magas szintű neurális hálózatokat tartalmazó programozási felület (API), amely a TensorFlow-on és más nyílt forráskódú gépi tanulási könyvtárakon alapul. A Keras lehetővé teszi a felhasználók számára a neurális hálózatok könnyű és intuitív felépítését, és kiválóan alkalmas a kézzel írt számjegyek felismerésére.

# Kidolgozandó feladat

Gépi látás tantárgy keretében választani kellett egy kidolgozandó feladatot, amelynek a megvalósítása, a használt programnyelv teljesen szabadon választható. Önálló munkában, Github-ra feltöltött formában kell a feladatot megoldani. A feladat része a fejlesztői és felhasználói dokumentáció egy dokumentumban, pdf formátumban.

**Kézzel írt karakterek felismerése:** Előre meghatározott karakterek (számok, angol ábécé betűi stb.) felismerése képről. A kép csak egy karaktert tartalmaz homogén háttér előtt.

Általános megoldási módszer egy ehhez hasonló feladat során:

A kézzel írt karakterek felismerése egy összetett folyamat, amely több lépést tartalmaz. A folyamat a karakterek digitalizálásával kezdődik, majd az előfeldolgozás és a szegmentálás lépésein keresztül jutunk el a karakterfelismeréshez. A végleges rendszer hatékonyságát az adatbázis és a gépi tanulási algoritmusok finomhangolása határozza meg.

1. Adatgyűjtés: Az első lépés az adatok beszerzése. A karaktereket tartalmazó képek beszerzése az adatbázisból, vagy saját adatgyűjtés révén történhet.
2. Adatelőkészítés: Az adatokat elő kell készíteni a feldolgozásra. Az előkészítés során a képek méretét egységesíteni kell, és a zajt eltávolítani, hogy javuljon a felismerés hatékonysága.
3. Mintázatkinyerés: A képen szereplő karakter mintáját ki kell nyerni a feldolgozás céljából. Ennek érdekében különböző mintázatkinyerési technikákat lehet alkalmazni, mint például az élek detektálása, a Hough-transzformáció, a szegmentálás és a binarizálás.
4. Jellemzők kiválasztása: A jellemzők kiválasztása az egyik legfontosabb lépés a felismerési folyamatban. A jellemzőknek olyan információkat kell tartalmazniuk, amelyek jellemzőek a karakterre és megkülönböztetik azt a többi karaktertől. A jellemzők lehetnek alak-, szín- vagy textúralapúak.
5. Osztályozó tréningje: A jellemzők alapján az osztályozó tréningjét el kell végezni. Az osztályozó lehet bármilyen gépi tanulási algoritmus, mint például a döntési fa, az SVM, a KNN vagy a neurális hálózat.
6. Tesztelés: A tesztelési folyamat során a tesztadatokon ellenőrizni kell az osztályozó hatékonyságát. Az osztályozó pontosságát, visszahívását és pontosságát az értékelés során kell figyelembe venni.
7. Finomhangolás: A finomhangolási folyamat során az algoritmusok paramétereit finomhangoljuk, hogy javuljon a felismerés hatékonysága.
8. Élesítés: Ha az osztályozó hatékonysága megfelelő, akkor az algoritmus élesítése történik. Az élesítés során a rendszert használni fogják valós időben a felhasználók. Fontos, hogy az élesítés előtt ellenőrizzük a rendszer stabilitását és megbízhatóságát. Ha hibákat találunk, akkor javítanunk kell azokat, és ismét tesztelni kell a rendszert. Ha a rendszer megfelelően működik, akkor bevezethető az élesítés, amely során a felhasználók használhatják a rendszert a kézzel írt karakterek felismerésére. Az élesítés során fontos, hogy a rendszer használata során folyamatosan figyeljük a rendszer működését és javítsunk rajta, ha szükséges.
9. Az utolsó lépés a karakter felismerése és az eredmény visszaadása. A karakter felismeréséhez használhatók például gépi tanulási algoritmusok, mint például a Support Vector Machines (SVM), a k-Nearest Neighbors (k-NN) vagy a neurális hálózatok. Ezen algoritmusok tanítása és finomhangolása az előző lépésekben elkészült adatbázison alapulhat.
10. A végleges rendszer tesztelése. A tesztelés során olyan képek használhatók, amelyek nem szerepelnek az adatbázisban, és a rendszer hatékonyságának ellenőrzésére szolgálnak. Ha a rendszer nem ad megfelelő eredményeket, akkor vissza lehet térni az előző lépésekhez, és finomhangolni lehet a paramétereket.

Egy egyszerűsített megoldási séma, amely még mindig lehet, hogy egyszerűsítésre szorul a megoldás kidolgozása során, hiszen vannak előre generált teszt adatbázisok, amelyek nagy segítséget jelenthetnek a megvalósítás során és megspórolhatják az adatbázis elkészítését.

1. Adatgyűjtés: Első lépésként szükséges gyűjteni a tanítóadatokat, amelyekkel a gépi tanulási algoritmusokat betanítjuk. Ehhez elkészíthetünk egy adatbázist, amelyben különböző emberek által írt számjegyek találhatóak. A felhasználók által írt karaktereket szkennelhetjük be, majd eltárolhatjuk őket egy adatbázisban.
2. Adatfeldolgozás: Az adatfeldolgozás során a szkennelt képeket előfeldolgozzuk és normalizáljuk, hogy a gépi tanulási algoritmusok számára megfelelő formátumot kapjunk. Ehhez használhatunk képfeldolgozási technikákat, például szürkeárnyalatos átalakítást, zajszűrést, élesítést stb.
3. Algoritmusok kiválasztása: Az algoritmusok kiválasztásakor számos lehetőség áll rendelkezésre. A legegyszerűbb megoldás lehet egy k-NN osztályozó, amely az új minta és az adatbázisban található legközelebbi szomszédok közötti távolságot számolja ki. Azonban az SVM, a döntési fa és a Random Forest algoritmusok is hatékonyak lehetnek.
4. Algoritmusok betanítása: Az algoritmusok betanításához használhatjuk az előző lépésben létrehozott adatbázist. Az adatbázisban található karaktereket fel kell osztani tréning- és teszthalmazokra. A tréninghalmaz segítségével betanítjuk az algoritmusokat, míg a teszthalmaz segítségével teszteljük azok hatékonyságát.
5. Élesítés: Ha az algoritmusok sikeresen betanításra kerültek, akkor az élesítésre lehet áttérni. Az élesítés során az elkészített programot valós időben lehet használni a kézzel írt számjegyek felismerésére.

# Program kód

Előzetesen, hosszabb internetes keresgélést követően két program változatot készítettem el, amelyekhez 1-1 nem kézzel irt számot feltöltöttem. A kézzel irt számok adatbázisát majd ezen számokkal elvégzett teszteket követően készítem el.

1. változat OCR

import pytesseract

import cv2

from PIL import Image

# Kép betöltése és előfeldolgozása

img = cv2.imread('image.jpg')

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

gray = cv2.medianBlur(gray, 3)

gray = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)[1]

# Kép mentése ideiglenes fájlként

filename = "{}.png".format("temp")

cv2.imwrite(filename, gray)

# Kézírás felismerése a Tesseract OCR motor segítségével

text = pytesseract.image\_to\_string(Image.open(filename), lang='eng')

# Eredmény kiíratása a képernyőre

print(text)

# Ideiglenes fájl törlése

os.remove(filename)

1. változat modell betanulásos

from sklearn.datasets import fetch\_openml

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

from sklearn.externals import joblib

# Adatok betöltése a MNIST adatkészletből

mnist = fetch\_openml('mnist\_784')

# A bemenetek (X) és a kimenetek (y) elkülönítése

X = mnist.data

y = mnist.target

# Az adatok normalizálása (0-255 skálából 0-1 skálára)

X = X / 255.0

# Az adatok felosztása tanítási és tesztadatra (80%-20% arányban)

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

# SVM modell inicializálása és betanítása

model = SVC(kernel='linear', C=10, gamma=0.01)

model.fit(X\_train, y\_train)

# A modell pontosságának kiszámítása a tesztadaton

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy: {:.2f}%".format(accuracy \* 100))

# A betanított modell elmentése a memóriából

joblib.dump(model, "digits\_cls.pkl")

import cv2

from sklearn.externals import joblib

from skimage.feature import hog

# Betöltjük a betanított modellt

model = joblib.load("digits\_cls.pkl")

# Kézírásos számjegy felismerése

def recognize\_digit(img):

# Kép előkészítése a felismerésre

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

ret, th = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY\_INV+cv2.THRESH\_OTSU)

contours, hierarchy = cv2.findContours(th, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

# Minden kontúr köré írt téglalapba illesztjük az eredeti képen

rects = [cv2.boundingRect(ctr) for ctr in contours]

for rect in rects:

# Kivágjuk a képrészletet, amely a kontúr körül van

x, y, w, h = rect

if w > 8 and h > 8:

roi = gray[y:y+h, x:x+w]

roi = cv2.resize(roi, (28, 28), interpolation=cv2.INTER\_AREA)

# Az adatok átalakítása a modell által elvárt formátumba

roi\_hog\_fd = hog(roi, orientations=9, pixels\_per\_cell=(14, 14), cells\_per\_block=(1, 1), visualize=False)

roi\_hog\_fd = roi\_hog\_fd.reshape(1, -1)

# A modell predikciója

nbr = model.predict(roi\_hog\_fd)

cv2.putText(img, str(int(nbr[0])), (x, y), cv2.FONT\_HERSHEY\_DUPLEX, 1, (0, 255, 255), 2)

return img

# A kép beolvasása

img = cv2.imread("kep.jpg")

# Az előző függvényt használjuk a kép felismerésére

recognized\_img = recognize\_digit(img)

# Az eredmény megjelenítése

cv2.imshow("Recognized Image", recognized\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()