**Szám és betűfelismerés feltanított modellel**

Mesterséges intelligencia és Haladó programozás beadandó

**1. Bevezetés**

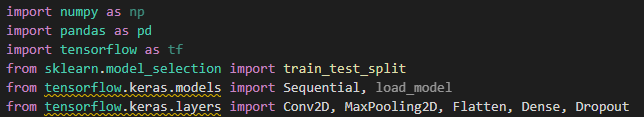
Ez a projekt egy mélytanulási neurális hálózatot használ a karakterek felismerésére az EMNIST adatbázis alapján, amely kézírásos számokat és betűket tartalmaz. A modell célja a karakterek pontos azonosítása, amit egy Streamlit felületen tudunk tesztelni.

**1.2 A projekt fő funkciói**

Az alkalmazásnak három fő funkciója van:

1. **Kézírás-felismerés rajzfelület segítségével** – a felhasználó a Canvasra rajzolhat egy karaktert, amit az alkalmazás felismer.
2. **Kézírás-felismerés feltöltött kép alapján**
3. **Webkamera alapú karakter-felismerés** – webkamera élő képén keresztül történő karakterfelismerés

**1.2. Függőségek**

****

A képen szöveg, képernyőkép, Betűtípus, képernyő látható

Automatikusan generált leírás

A kód működéséhez a következő könyvtárakat kell telepíteni:

* **numpy**: Tömbalapú számításokhoz és adatfeldolgozáshoz
* **pandas**: Adatfeldolgozáshoz, CSV-fájlok kezeléséhez
* **tensorflow és keras**: Mélytanulási modellek építéséhez és betanításához
* **scikit-learn** - Adatok felosztása tréning- és teszthalmazra
* **Streamlit** – felhasználói felület építésére
* **OpenCV** – képfeldolgozás
* **PIL** – képfájlok kezelése

A függőségek telepítése a következő paranccsal történt:

pip install streamlit numpy tensorflow opencv pillow streamlit

**2. A modell feltanítása**

**2.1. Adatelőkészítés**

Az EMNIST adatbázis kézírásos karakterek kép- és címkeadatainak CSV formátumú fájlban van tárolva. A következő lépések történnek az adatok feldolgozása során:

1. **Adatok betöltése**: A CSV fájl betöltéséhez a pandas.read\_csv() függvényt használjuk.
2. **Adatformázás**: A képadatokat 28x28 méretűre alakítjuk át, így egy szürkeárnyalatos, egycsatornás képhez jutunk. Az adatok normalizálása a 0–1 közötti tartományra történik, amit a neurális hálózat könnyebben tud feldolgozni.
3. **Tréning és teszthalmazra bontás**: A teljes adatbázist 80%-ban tréning- és 20%-ban teszthalmazra bontjuk a train\_test\_split függvény használatával, amely segít a modell teljesítményének kiértékelésében.

**2.2. A neurális hálózat architektúrája**

A projekt során egy konvolúciós neurális hálózatot építünk fel, amely a képek felismerésére specializálódott.

A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Automatikusan generált leírás

Rétegei:

1. **Konvolúciós rétegek (Conv2D)**: A hálózat első rétegei konvolúciós rétegek, amelyek a képek jellemzőit nyerik ki, és azokat használják a mintázatok felismeréséhez. Két konvolúciós réteget használunk, amelyek 32, illetve 64 szűrőből állnak.
2. **Max Pooling rétegek**: Minden konvolúciós réteg után egy MaxPooling réteg következik, amely a képek méretének csökkentésére és a legfontosabb jellemzők kiválasztására szolgál.
3. **Flatten réteg**: A képadatokat egydimenziós vektorra alakítja, hogy a következő teljesen összekapcsolt (Dense) réteg feldolgozhassa azokat.
4. **Fully connected (Dense) rétegek**: A Flatten réteg után egy Dense réteget adunk hozzá, és a jellemzők további feldolgozásáért felelős. Az utolsó réteg egy Dense réteg 62 neuronnal, amely az EMNIST adatbázis minden osztályához egy-egy kimeneti neuront biztosít.
5. **Dropout réteg**: A Dropout réteg véletlenszerűen kikapcsolja a neuronok egy részét a betanítás során, ezzel csökkentve a túlillesztést (overfitting).
6. **Aktivációs függvények**: A rejtett rétegekhez ReLU aktivációs függvényt használunk, míg a kimeneti rétegben a softmax aktivációval valósítjuk meg a valószínűségi kimenetet, amely segíti a karakterek közötti kategorizálást.

**2.3. Modell betanítása és értékelése**

A modell betanítása az alábbi módon történik:

1. **Fordítás**: A model.compile() segítségével meghatározzuk a modell betanításához szükséges optimalizálót és veszteségfüggvényt. Itt az Adam optimalizálót és a sparse\_categorical\_crossentropy veszteségfüggvényt használjuk.
2. **Betanítás**: A model.fit() függvénnyel végrehajtjuk a modell betanítását a tréning adathalmazon. A betanítás során a hálózat paraméterei frissülnek, hogy minél jobban illeszkedjenek az adatokhoz. 20 betanítási ciklust (epoch) használtunk jelen esetben, ezzel 85% körüli pontosságot értünk el, és 1% körüli veszteséget.
3. **Értékelés**: A modell teljesítményét a teszthalmazon mérjük. A model.evaluate() függvénnyel kiszámítjuk a veszteséget és a pontosságot. A pontosság megmutatja, hogy a modell milyen pontosan ismeri fel az új, eddig nem látott adatokat.

**2.4. Modell mentése és újrafelhasználása**

A betanított modellt elmentjük egy .keras formátumú fájlba, amely lehetővé teszi, hogy később újra betöltsük és alkalmazzuk. A mentés a model.save() függvénnyel történik, így a jövőben az elmentett modellt betölthetjük a load\_model függvénnyel, és anélkül használhatjuk, hogy újra betanítanánk.

**2.5. Összegzés és jövőbeni fejlesztések**

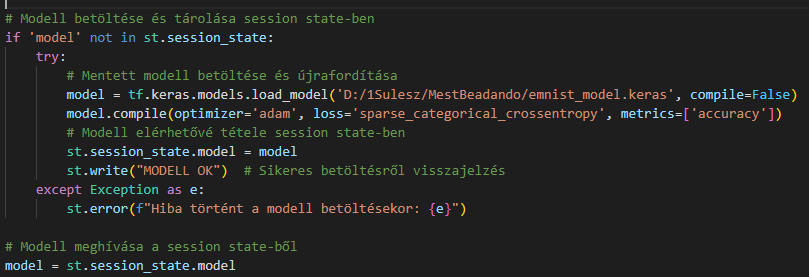
Ezzel elkészült a modellünk amelyet felhasználva meg tudjuk majd állapítani egy rajzolt vagy feltöltött karakterről hogy milyen betű vagy szám.

* **Hálózat bővítése**: További rétegek hozzáadása vagy a hálózat módosítása, hogy még pontosabb felismerést érjünk el.
* **Adatok bővítése és előfeldolgozása**: További adatok és különböző képi zajcsökkentési technikák használata segíthet a modell teljesítményének javításában.
* **Túlillesztés csökkentése**: További Dropout rétegek hozzáadása, vagy az adatok augmentálása (pl. forgatás, nagyítás) a training során.

**3. Streamlit felület**

**3.2 Modell betöltése és tárolása a Session State-ben**

A betöltött modellt a Streamlit session\_state objektumba mentjük, így később újratöltés nélkül hozzáférhetünk.



**3.3 Karakterlista létrehozása**

A characters lista tartalmazza az összes karaktert (számokat és betűket), amelyeket a modell képes felismerni. Ez a lista szükséges az előrejelzés eredményének szöveges megjelenítéséhez.



**3.4 Felhasználói felület és funkciók részletes ismertetése**

**3.4.1 Kézírás-felismerés rajzfelület segítségével**

A Streamlit segítségével egy rajzfelületet hozunk létre, ahol a felhasználó szabadon rajzolhat. A rajzolt kép előfeldolgozásra kerül, majd a modell segítségével megpróbálja azonosítani a karaktert.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

**Előfeldolgozás és felismerés**

A rajzolt kép előkészítése során:

* Szürkeárnyalatúvá alakítjuk
* Binarizáljuk (Otsu módszerrel)
* Átméretezzük 28x28 pixelre és normalizáljuk

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

**3.4.2 Kézírás-felismerés feltöltött kép alapján**

A felhasználó képet tölthet fel, amelyet a program felismerésre feldolgoz. A kép előkészítése során a feltöltött képet (hasonlóan az előzőekben taglaltakkal):

**3.4.3 Webkamera alapú felismerés**

Ha a felhasználó bekapcsolja a webkamerát, az élőképen keresztül történik a karakterfelismerés. Minden egyes képkockát feldolgozunk:

* Átalakítjuk szürkeárnyalatúvá
* Binarizáljuk
* Átméretezzük és normalizáljuk

**4. Kimeneti elemek és vizualizáció**

**4.1 Eredmény megjelenítése**

Az eredményt minden funkció esetében megjelenítjük. A felismerés eredményét a characters listából nyerjük ki, és oszlopdiagramként is ábrázoljuk az előrejelzési valószínűségeket.

**5. Használat**

Az alkalmazás böngészőben fog elindulni, miután kiadtuk a következő parancsot:

**streamlit run \*alkamazás elérési útja\***

**4. Kinézet, használat**

**Canvas:**

**A képen szöveg, képernyőkép, Betűtípus, tervezés látható

Automatikusan generált leírás**

Mindhárom lehetőségnél egy diagram mutatja, hogy mennyire volt biztos egy-egy karakterben.

**A képen szöveg, sor, képernyőkép, szám látható

Automatikusan generált leírás**

**Feltöltés:**

**A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Automatikusan generált leírás**

**Web kamera:**

**A képen szöveg, képernyőkép látható

Automatikusan generált leírás**