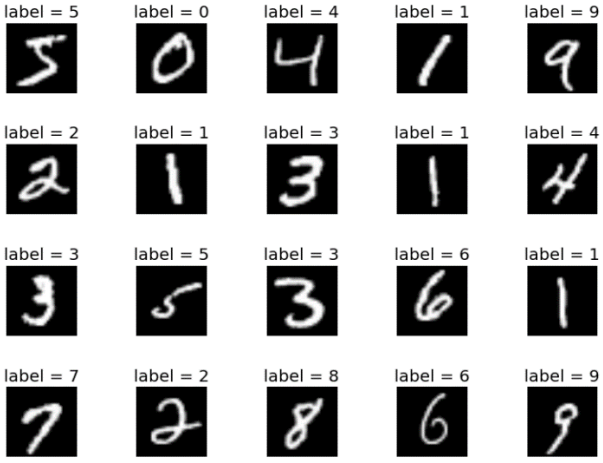
Sárgacsekk digitalizálása

Az elsődleges cél a projekt során az volt, hogy a sárga csekket szkenner nélkül beolvasni képes kódot állítsunk össze. Pontosan erre vonatkozóan nem találtam kész kódot, vagy módszertant, így kénytelen voltam magam megkonstruálni a feladatmegoldás stratégiáját.

1.Számfelismerés

A program központi szerepét a számfelismerés tölti be. Ezt a képességet a mélytanulás módszerével tudtam ráruházni a kódra.
Erre a célra a neurális hálót a *Keras* modulban szerkesztettem, és az mnist adatbázist használtam a tanításhoz. Az mnist egy írott számjegyeket és értéküket tartalmazó, 70 ezer elemű adatbázis, amiben 60 ezer tanító és 10 ezer teszt adatpár található. Ennek megfelelően bontottam szét az adatbázist a kódban is.

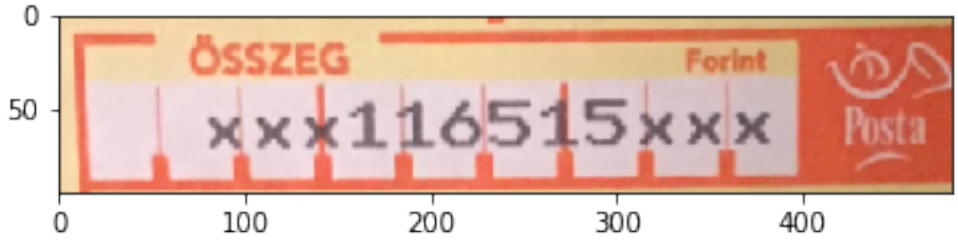


1.ábra az mnist adatbázis szemléltetése
(https://mc.ai/tutorial-1-mnist-the-hello-world-of-deep-learning/)

A tanítás során *adam*-mel optimalizáltam, a és kategorikus keresztentrópiával számoltattam a veszteséget. Az egész adatbázison végig menve, három epoch alatt 98%-os pontosságot sikerült elérni a validációs hiba monoton csökkenésével, ami meglepően jó eredmény. Tehát a számfelismerést elsajátította a hálózat.

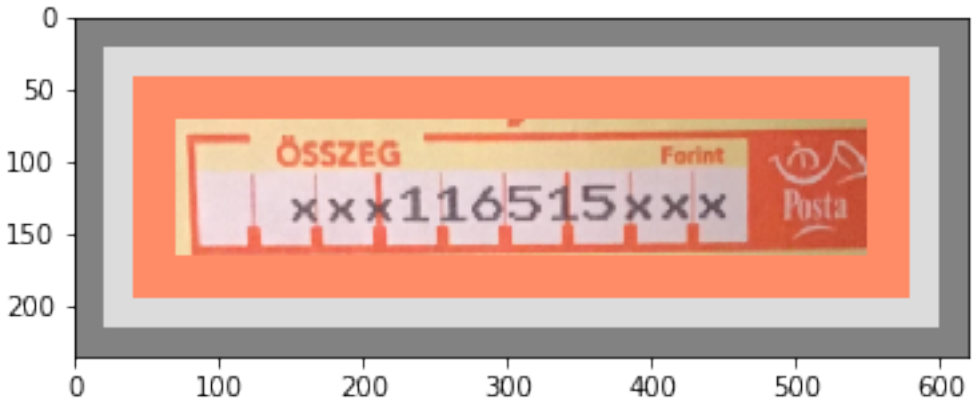
2.Képszegmentáció

A következő probléma az, hogy miként fogja beolvasni az összeget. Ezt onnan közelítettem meg, hogy mivel a sárgacsekk formátuma „szabványos”, ezért a felismerendő számok minden csekken ugyanott helyezkednek el, erre a helyre koordinátákkal hivatkozni. Tehát az első teendő a kép kiegyenesítése és kivágása.
Ezt az *openCV* modul segítségével oldottam meg. A képet szürkeárnyalatúra állítottam, majd a megadtam egy küszöbértéket, ami feletti pixelérték az aktuális pixel fehérségét, az alatti viszont a pixel feketeségét okozta. Ez a szegmentáció akkor hatékony, ha a csekkről sötét, lehetőleg homogén háttérrel készítjük a képet, jó megvilágításban. A sikeres szegmentáció eredménye egy fehér téglalap fekete háttéren. Ezen fehér téglalap ferdeségét a kontúrra illeszthető legkisebb területű téglalap dőlése alapján határoztam meg, majd ennek megfelelően visszaforgattam az eredeti képet, majd ugyanezzel a módszerrel eltávolítottam a sötét hátteret. Ezek után a kép méretét pixelekben újra definiáltam. Így bármilyen méretű bemenő képre ugyanolyan méretű kimenet érkezik. A befizetett összeget tartalmazó cella helyének koordinátája ezek után pixelben megadható. Az ezen kívüli tartományt megint csak eltávolítottam.



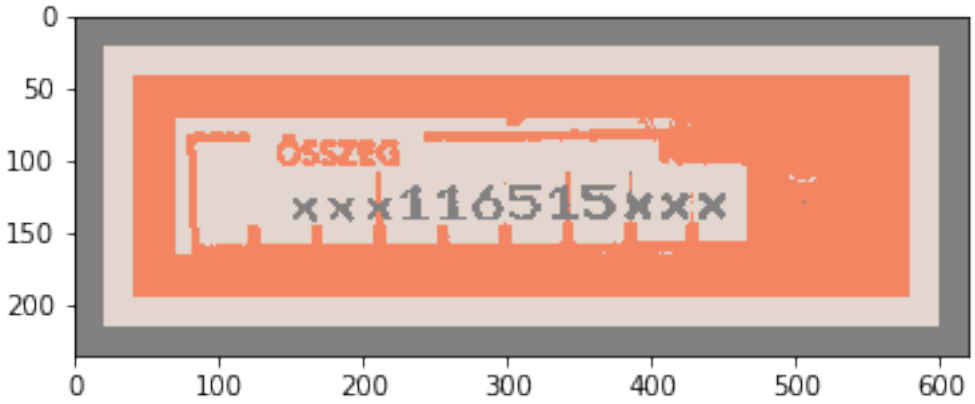
2.ábra a számunkra fontos képrészlet.

A megmaradt képrészlet hordoz minden számunkra fontos információt. Megfigyelhetjük, hogy három jellemző színből épül fel: a piros keret, a sárga háttér és a papír fehérje. Ahhoz, hogy a program felismerje a számjegyeket, ki kell szűrnünk őket ebből az elrendezésből.
Ezt a *k-közép* módszer segítségével értem el. Ezáltal három színre bontottam a képet, mégpedig három olyan színre, amit automatikusan ismer fel az eljárás. Ez nem vezetett jó eredményre, emiatt segíteni kellett az algoritmusnak, hogy észrevegye, milyen színeket használjon: a három színhez közeli árnyalatú kereteket toldottam a képhez.



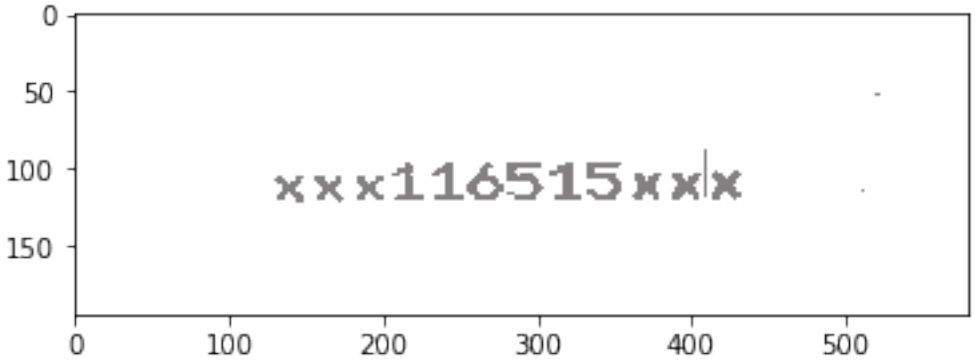
3.ábra a keretezett részlet

Erre a k-közepet lefuttatva a következő eredmény született:



4.ábra a három árnyalattal újra rajzolt kép

Ezek után egy-egy *for*-ciklussal fehérré tettem a piros és világosszürke színeket, a külső, sütét szürke keretet pedig kivágtam. Az eredmény:



5.ábra a képszegmentáció eredménye

3.Számok detektálása

Az utolsó lépés, hogy létrehozzuk a betanult háló bemenetét, azaz a 28 x 28 pixeles számjegyeket. Ehhez újabb szegmentációra volt szükség. Itt nagy jelentősége volt a Gauss-homályosításnak, ugyanis ahogy az 5. ábrán is látszik, olyan pontok/vonalak is megjelennek, amik nem a számjegyek részei. Ezektől próbálunk megszabadulni az említett eljárás segítségével. Ezek után kontúrokat detektálunk az *openCV* modul segítségével, ezeket kivágjuk és kipárnázzuk úgy, hogy 28 x 28-as méretűek legyenek.

4.Pénzösszeg beolvasása és felismerése, összefoglalás

A program helyesen képes detektálni az objektumokat. Ideális esetben az ismeretlen x-jeleket ugyanannak a számnak vélné, és ekkor a prediktált összeg elejéről és végéről három-három karakter levágásával helyes eredményt kapnánk, ám ez sajnos nem teljesül. A program a számokat ezen kívül nem sorban írja ki a hibás hierarchia észlelés következtében, és ezen felül még hajlamos elvéteni is a számokat. A fenti példa kimenete: 411242022515, ami kétségkívül használhatatlan eredmény. Kézírásos csekkek esetében jobb eredményt kapunk, ott a nem használt helyiértékek kihúzása okoz problémát, az *mnist*-en tanított háló azonban pontosabb becsléseket szolgáltat.

Ezen problémák megoldhatók lehetnek nyomtatott számok, betűk és különleges karakterek bevonásával a tanításba.

Források:

https://gist.github.com/bigsnarfdude/d811e31ee17495f82f10db12651ae82d

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_ml/py_kmeans/py_kmeans_opencv/py_kmeans_opencv.html#kmeans-opencv

https://stackoverflow.com/questions/11627362/how-to-straighten-a-rotated-rectangle-area-of-an-image-using-opencv-in-python/48553593#48553593

https://medium.com/@yash.kukreja.98/recognizing-handwritten-digits-in-real-life-images-using-cnn-3b48a9ae5e3