Scarlat Marius Ștefan
Grupa 252

Realistic Image Classification

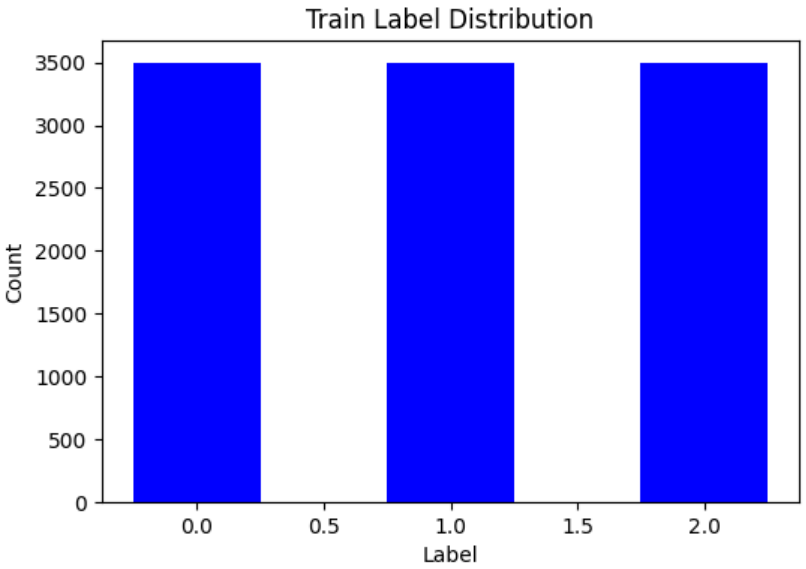
1. Descrierea Proiectului

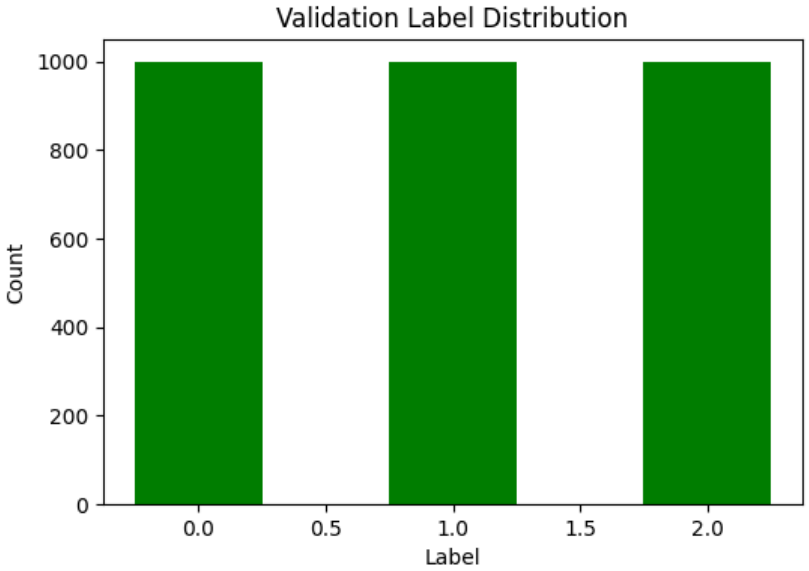
Proiectul presupune clasificarea unor imagini de dimensiuni 80 x 80 generate de modele generative adânci în 3 clase distincte de la 0 la 2.

2. Setul de date

- 10500 imagini de train, împreuna cu label-urile asociate.
- 3000 imagini de validation, împreuna cu label-urile asociate.
- 4500 imagini de test

Distributia label-urilor pe train si validation:





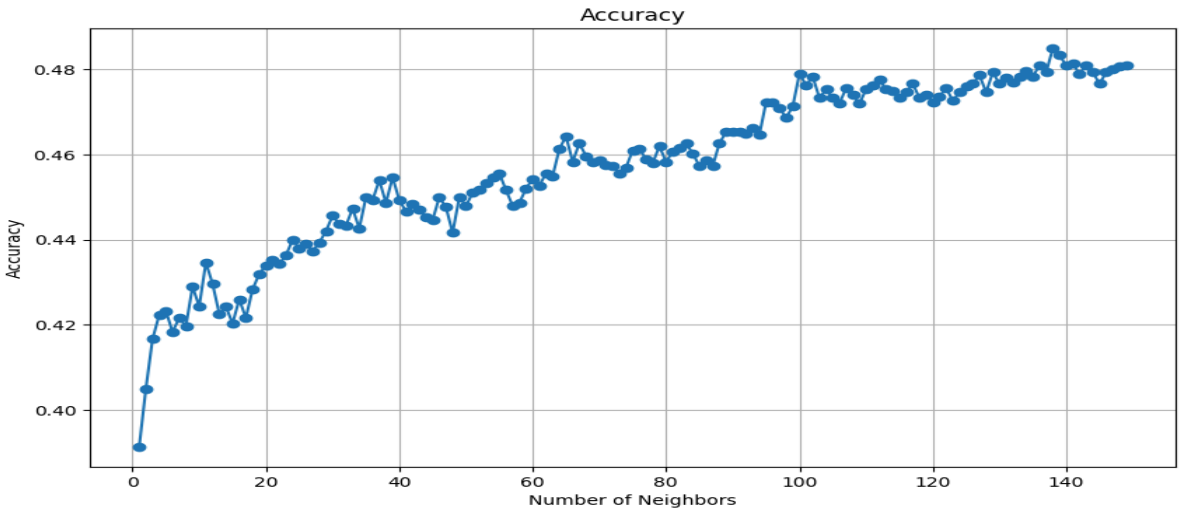
3. Abordări

În rezolvarea problemei de clasificare am încercat mai mulți algoritmi precum:

- KNN
- CNN

A. KNN

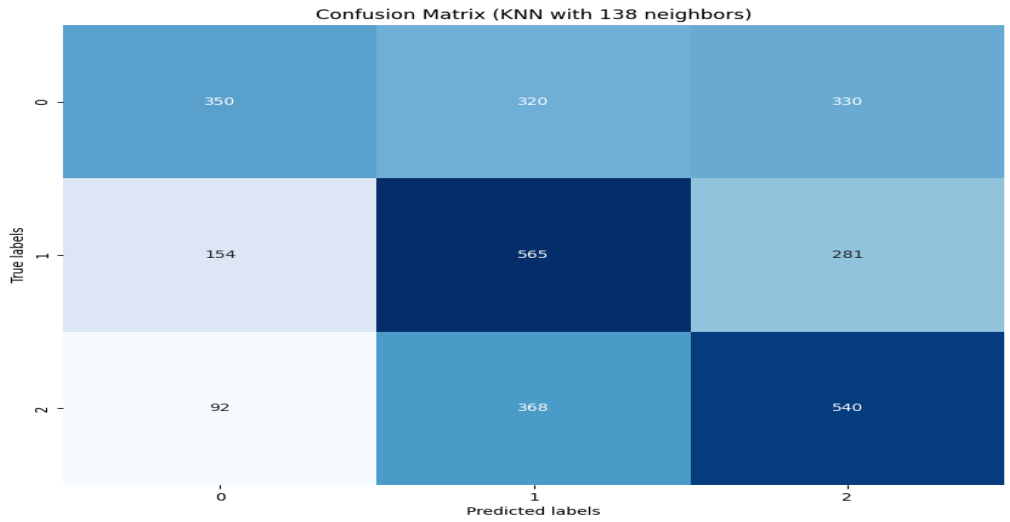
Am ales să încep cu algoritmul KNN (K-Nearest Neighbors) pentru a stabili un punct de referință cât mai solid în vederea dezvoltării unui eventual model de tip CNN (Convolutional Neural Network). În prima faza, am decis sa nu aplic nicio augmentare a datelor si sa pastrez imaginile în forma lor originală pentru a evalua corect performanța brută a modelului. Diagrama de mai jos ilustrează performanța modelului pentru un număr de vecini situat intre 1 si 150, utilizand metrica Minkowski.



Folosind metrica Euclidiana, am reușit să obțin o acuratețe maximă de 0.484. Această metrică calculează distanța „dreaptă” între două puncte în spațiul caracteristicilor, fiind una dintre cele mai utilizate metode datorită simplității sale. Cu toate acestea, am constatat că utilizarea metricii Manhattan, care calculează distanța „pe axă”, a dus la o îmbunătățire a performanței, atingând o acuratețe maximă de 0.50. Toate aceste scoruri maxime fiind obținute pentru un număr de 138 vecini (n_neighbors=138).

Pe lângă alegerea metricii, numărul de vecini (n_neighbors) are un impact semnificativ asupra performanței modelului KNN. Un număr prea mic de vecini poate duce la un model care este prea sensibil la zgomotul din date, în timp ce un număr prea mare poate duce la un model prea generalizat, care pierde detalii specifice ale claselor.

Aceste rezultate obținute subliniază importanța alegerii corecte a metricii de distanța și a numărului de vecini pentru optimizarea performanței. Mai jos, atasez o imagine cu matricea de confuzie. Următorul pas va fi aplicarea tehnicilor de augmentare a datelor și explorarea modelelor mai complexe, cum ar fi CNN, pentru a îmbunătăți acuratețea clasificării.



B. CNN

După ce am evaluat performanța KNN pe datele brute, am decis să încerc și un model CNN (Convolutional Neural Network). CNN urile sunt cunoscute pentru abilitatea lor de a efectua convolutii, ceea ce le permite să extragă automat caracteristici relevante din imagini. Aceste rețele aplică filtre convoluționale pentru detectarea trăsăturilor importante și utilizează straturi de pooling pentru a reduce dimensiunea datelor, păstrând informațiile esențiale. Am construit un model CNN utilizând PyTorch, obținând o performanță inițială mult mai bună decât KNN, cu o acuratețe inițială maximă de 70% pe setul de validare.

1. Procesarea Imaginilor:
    - **Redimensionarea Imaginilor:** Imaginile originale au fost redimensionate la 80x80x3 pixeli pentru a uniformiza dimensiunile datelor de intrare.
    - **Normalizarea Imaginilor:** Valorile pixelilor au fost scalate la intervalul [0, 1] pentru a asigura o convergenta mai rapidă și stabilă a modelului în timpul antrenării.
2. Augmentarea Datelor:

    Transformări Aplicate pentru setul de antrenament: am aplicat o serie de transformări pentru a crește diversitatea datelor, inclusiv:

    - **RandomHorizontalFlip**
    - **RandomRotation(5)**

    Transformări pentru setul de validare și test: imaginile din seturile de validare și test au fost doar normalizate, fără alte augmentări, pentru a asigura o evaluare corectă a performanței modelului.

3. Echilibrarea Overfitting-ului:
    - **Augmentări Minime:** S-au aplicat augmentări minimale până când am observat o mică diferență între acuratețea pe setul de antrenament și cel de validare, sugerând o descreștere în overfitting.
    - **Impactul Augmentărilor:** Am constatat că introducerea unui număr mai mare de augmentări ducea la scăderea performanței modelului, indicând o posibilă supra regularizare și o perturbare excesivă a datelor.
4. Definirea Modelului:

    Layere folosite:

    - Conv2d: Layer care aplică filtre pe imaginea de input pentru a genera o matrice de caracteristici
    - BatchNorm2d: Layer care normalizează datele de input, astfel încât media este 0 și dispersia este 1, ajutând la o învățare mai rapidă.

- ReLU: Layer de activare care aplică funcția ReLU, transformând valorile negative în 0 și lăsând neschimbate valorile pozitive.
- MaxPool2D: Layer care reduce dimensiunile matricei de caracteristici prin glisarea unei ferestre și selectarea valorii maxime din fiecare fereastră.
- Dropout: Layer care reduce overfitting-ul prin setarea aleatorie a unor neuroni la 0 (dezactivare) cu o frecvență dată.
- Flatten: Layer care transformă matricea de input într-un vector unidimensional.
- Linear: Layer dens conectat care face legătura între toți neuronii din layer-ul precedent și toți neuronii din layerul curent.

Hyperparameters:

Pentru **layere convolutionale**:

- numărul de filtre din mulțimea {32, 64, 128, 256}
- am folosit un kernel de (3,3) - fereastră de 3 x 3 glisat pe suprafața pozelor
- ca activator am folosit ReLu

Pentru **layerele dropout**: am setat rata de dropout la 0.1 deoarece valorile mai mari duceau la underfitting, însă am testat modele si fără dropout, acestea ducand la performanțe similare.

Pentru **layere liniare**:

- nr de neuroni din mulțimea {32, 64, 128, 256}
- ca activator am folosit ReLu
- pentru ultimul strat am 3 neuroni pentru a clasifica in cele trei clase, layer care se folosește de funcția de activare softmax pentru a face predicțiile finale.

5. Compilarea modelului:
   - folosesc funcția de loss cross entropy care este utilizata pentru clase multiple si calculeaza loss-ul dintre label-uri si predictii.

- folosesc Adam ca optimizator - care este un superset pentru stochastic gradient descent.
- metrica modelului este acuratețea pe testele de validare.
- folosesc early stopping pentru a salva modelul cu cea mai buna acuratete pe datele de validare si pentru a oprii modelul dacă într-un anumit range fixat (de exemplu 20) de epoci nu am obtinut deloc nicio îmbunătățire semnificativă.

6. Variante de modele încercate:

   Într-o prima etapa am încercat o rețea cu un singur strat dens si cu un singur layer de convolutie înainte de a aplica MaxPool2D.

```
nn.Conv2d(64, 64, kernel_size=3),
nn.BatchNorm2d(64),
nn.ReLU(),
nn.MaxPool2d(2, 2),
```

```
nn.Flatten(),
nn.Linear(256 * 2 * 2, 3),
```

Layer	Hiperparametrii
Conv2D	filters = 32, kernel = (3,3), activation = relu
BatchNormalization	default
MaxPool2D	pool_size = (2,2)
Conv2D	filters = 64, kernel = (3,3), activation = relu
BatchNormalization	default
MaxPool2D	pool_size = (2,2)
Conv2D	filters = 128, kernel = (3,3), activation = relu
BatchNormalization	default
MaxPool2D	pool_size = (2,2)
Conv2D	filters = 256, kernel = (3,3), stride = 1, padding = 0, activation = relu
BatchNormalization	default
Flatten	default
Linear	units = 3

Cu aceasta abordare am obtinut un scor undeva în jur de 72%. Pentru a crește acuratețea modelului am decis sa cresc complexitatea modelului. Inițial, am încercat sa adaug Dropout-uri pe straturile de convolutie si pe cele dense, însă nu am observat o performanta imbunatatita semnificativă. Am observat însă că dublarea straturilor de convoluție înainte de a aplica MaxPool2D si folosirea a 3 layere pentru stratul dens a dus la o îmbunătățire semnificativă a modelului, acesta ajungand la o acuratețe în jur de 76%.

```
nn.Conv2d(64, 128, kernel_size=3),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.Conv2d(128, 128, kernel_size=3),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.MaxPool2d(2, 2),
```

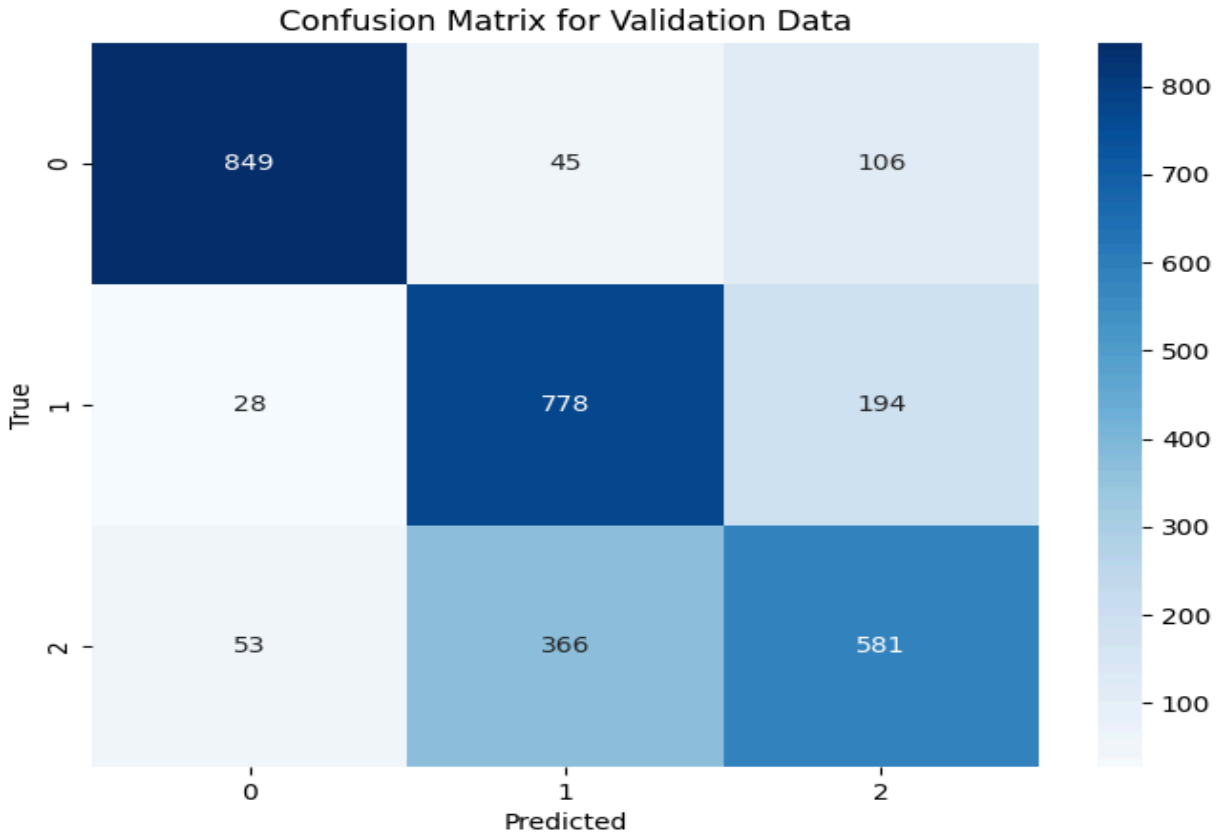
```
nn.Flatten(),
nn.Linear(256 * 2 * 2, 128),
nn.ReLU(),

nn.Linear(128, 256),
nn.ReLU(),

nn.Linear(256, num_classes),
```

Arhitectura finală a modelului:

Layer	Hiperparametrii
Conv2D	filters = 32, kernel = (3,3), activation = relu
BatchNormalization	default
Conv2D	filters = 32, kernel = (3,3), activation = relu
MaxPool2D	pool_size = (2,2)
Conv2D	filters = 64, kernel = (3,3), activation = relu
BatchNormalization	default
Conv2D	filters = 64, kernel = (3,3), activation = relu
MaxPool2D	pool_size = (2,2)
Conv2D	filters = 128, kernel = (3,3), activation = relu
BatchNormalization	default
Conv2D	filters = 128, kernel = (3,3), activation = relu
MaxPool2D	pool_size = (2,2)
Conv2D	filters = 256, kernel = (3,3), stride = 1, padding = 0, activation = relu
BatchNormalization	default
Conv2D	filters = 256, kernel = (3,3), stride = 1, padding = 0, activation = relu
BatchNormalization	default
Flatten	default
Dense	units = 128, activation = relu
Dense	units = 256, activation = relu
Dense	units = 3, activation = softmax



4. Referințe

- https://medium.com/thecyphy/train-cnn-model-with-pytorch-21dafb918f48
- https://gist.github.com/aliwaqas333/5d53e4a85a43f32db9e2a778b7d49fb1#file-imageclassification-py
- https://pytorch.org/docs/stable/
- https://www.youtube.com/watch?v=jztwpsIzEGc
- https://fmi-unibuc-ia.github.io/ia/