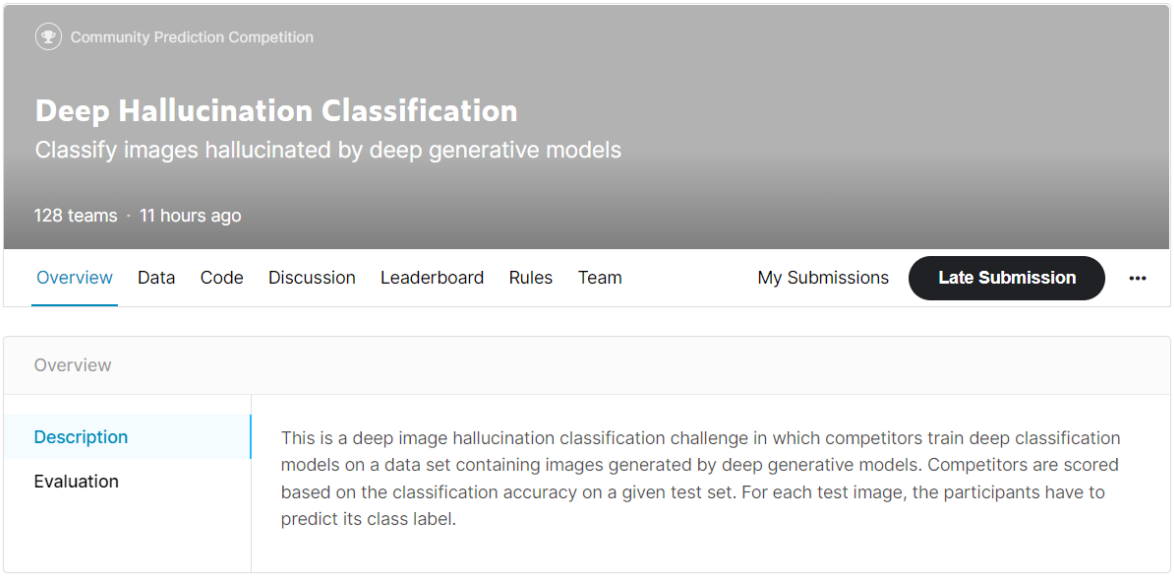
Deep Hallucination Classification - Kaggle Competition

Baciu Daniel - Mihai

Facultatea de Matematica si Informatica, Grupa 243

1 Introducere

Subiectul central al acestei competitii de pe platforma KAGGLE a fost de a clasifica imagini RGB de dimensiune 16 x 16 in 7 clase, numerotate de la 0 la 6. Pentru aceasta clasficare trebuia sa venim cu un model pe care il puteam antrena pe 8000 imagini, valida pe 1173 poze, iar partea de testare se facea pe 2819 imagini de aceeasi dimensiune.



2 Modele folosite :

2.1 KNN :

2.1.1 Descriere generala si implementare :

KNN sau K-Nearest Neighbors este un algoritm ce clasifica o imagine in functie de vecinii acesteia, vecinii fiind calculati dupa o norma (aceasta poate fi L1 - distanta Euclidiana, L2 - distanta Manhattan, s.a.m.d). Pe scurt, daca avem un punct in plan si vrem sa ii atribuim eticheta, luam cei mai apropiati k vecini de acest punct, calculati dupa distanta stabilita si vedem care etichete apar cel mai des printre acesti k vecini, iar acel punct din plan va primi eticheta cu numarul de aparitii maxim. Daca doua etichete au acelasi numar de aparitii, se va alege una random.

La acest model am facut o functie ce clasifica imaginea pe baza datelor de antrenare. Sortez dupa norma L2, ordonez indicii dupa ordonarea pozelor si fac prezicerea.

```
def classifica_imaginea(imagini_train, labeluri_train, pixel_image, no_of_neigh = 1) :

    distante = np.sqrt(np.sum(np.power((imagini_train - pixel_image), 2), axis=1))

    indici_sortati = np.argsort(distante)

    indici_sortati = indici_sortati[ :no_of_neigh]

    preziceri = labeluri_train[indici_sortati]

    return np.argmax(np.bincount(preziceri))
```

2.1.2 Augmentare date :

In primul rand, am facut flatten la imagini, adica din 16 x 16 x 3 am ajuns la un vector de 768 elemente. Apoi am normalizat datele si am aplicat tuturor imaginilor de train o rotire la 90 de grade, urmand ca apoi sa fac preziceri pe acest batch de 16000 date.

```
imagini_pentru_train = np.array([return_pixel_rgb_values(x) for x in fisier_txt_train.id])

imagini_pentru_validation = np.array([return_pixel_rgb_values(x) for x in fisier_txt_validation.id]).reshape((1173, 768)) / 255

imagini_pentru_test = np.array([return_pixel_rgb_values(x, "../data/test/") for x in fisier_txt_test.id]).reshape((2819, 768)) / 255

imagini_pentru_train_rotire_90 = np.rot90(imagini_pentru_train, k=1, axes=(0, 1)).reshape((8000,768)) / 255

imagini_pentru_train = imagini_pentru_train.reshape((8000, 768))/255

#----------------------------------------

imagini_pentru_train_mean = np.mean(imagini_pentru_train, axis=0)

imagini_pentru_train_rotire_90_mean = np.mean(imagini_pentru_train_rotire_90, axis=0)

imagini_pentru_validation_mean = np.mean(imagini_pentru_validation, axis=0)

imagini_pentru_test_mean = np.mean(imagini_pentru_test, axis=0)

#----------------------------------------

imagini_pentru_train_standardizate = imagini_pentru_train - imagini_pentru_train_mean / np.std(imagini_pentru_train)

imagini_pentru_train_rotire_90_standardizate = imagini_pentru_train_rotire_90 - imagini_pentru_train_rotire_90_mean / np.std(imagini_pentru_train_rotire_90)

imagini_pentru_validation_standardizate = imagini_pentru_validation - imagini_pentru_validation_mean/np.std(imagini_pentru_validation)

imagini_pentru_test_standardizate = imagini_pentru_test - imagini_pentru_test_mean / np.std(imagini_pentru_test)
```

2.1.3 Acuratete :

Am facut un for cu numarul de vecini variabil si am afisat acuratetea.

```
for no_of_neig in [10, 11, 12, 15, 20, 23, 25]:
    predictii = []
    for img in imagini_pentru_validation_standardizate:
        predictii.append(classifica_imaginea(img_train, labels_train, img, no_of_neig))
        # fac predictii pe imaginile de train plus cele rotite la 90
    predictii = np.array(predictii)
    print(f"Valoarea de prezicere pentru \tnr vecini = \t{no_of_neig} \teste = \t{(np.array(fisier_txt_validation.label) == predictii).mean()}")
```

In acest fel am observat ca numarul optim de vecini este 12 si am creat fisierul de output folosind numarul de vecini=12.

```
Valoarea de prezicere pentru    nr vecini =     10  este =  0.4356351236146633
Valoarea de prezicere pentru    nr vecini =     11  este =  0.4356351236146633
Valoarea de prezicere pentru    nr vecini =     12  este =  0.4381926683716965
Valoarea de prezicere pentru    nr vecini =     15  este =  0.4339300937766411
Valoarea de prezicere pentru    nr vecini =     20  este =  0.4322250639386189
Valoarea de prezicere pentru    nr vecini =     23  este =  0.4322250639386189
Valoarea de prezicere pentru    nr vecini =     25  este =  0.4339300937766411
```

2.2 SVM :

SVM sau Support Vector Machines este o metoda supervizata de invatare ce este folosita in probleme de clasificare, regresii, etc. Separa clasele calculand o suprafata de decizie aflata la distanta egala si maxima de punctele clasificate.

Aceasta are ca si parametrii mai importanti parametrul de regularizare C si kernelul care poate fi ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’. Kernelul se alege in functie de date, daca acestea sunt linear separabile sau nu.

2.2.1 Implementare :

Pentru aceasta metoda de clasificare am ales sa import sklearn.svm si sa folosesc clasa SVC. Aceasta are ca si parametrii mai importanti valoarea C si kernelul, dar si functii de predict si fit care sunt foarte usor de folosit.

2.2.2 Augmentare date :

Augmentarea datelor este similara metodei folosite si anterior, si anume : dupa ce am facut flatten, am normalizat datele si am aplicat tuturor imaginilor de train o rotire la 90 de grade, urmand ca apoi sa fac preziceri pe acest batch de 16000 date.

```
imagini_pentru_train = np.array([return_pixel_rgb_values(x) for x in fisier_txt_train.id])

imagini_pentru_validation = np.array([return_pixel_rgb_values(x) for x in fisier_txt_validation.id]).reshape((1173, 768)) / 255

imagini_pentru_test = np.array([return_pixel_rgb_values(x, "../data/test/") for x in fisier_txt_test.id]).reshape((2819, 768)) / 255

imagini_pentru_train_rotire_90 = np.rot90(imagini_pentru_train, k=1, axes =(0, 1)).reshape((8000,768)) / 255

imagini_pentru_train = imagini_pentru_train.reshape((8000, 768))/255

#---------------------------------------

imagini_pentru_train_mean = np.mean(imagini_pentru_train, axis=0)

imagini_pentru_train_rotire_90_mean = np.mean(imagini_pentru_train_rotire_90, axis=0)

imagini_pentru_validation_mean = np.mean(imagini_pentru_validation, axis=0)

imagini_pentru_test_mean = np.mean(imagini_pentru_test, axis=0)

#---------------------------------------

imagini_pentru_train_standardizate = imagini_pentru_train - imagini_pentru_train_mean / np.std(imagini_pentru_train)

imagini_pentru_train_rotire_90_standardizate = imagini_pentru_train_rotire_90 - imagini_pentru_train_rotire_90_mean / np.std(imagini_pentru_train_rotire_90)

imagini_pentru_validation_standardizate = imagini_pentru_validation - imagini_pentru_validation_mean/np.std(imagini_pentru_validation)

imagini_pentru_test_standardizate = imagini_pentru_test - imagini_pentru_test_mean / np.std(imagini_pentru_test)
```

2.2.3 Acuratete :

Pentru acest model am facut 2 for-uri, unul pentru valoarea parametrului C si unul pentru valoarea kernelului.

```python
for c in [0.1, 0.5, 1.0, 1.1, 1.8, 2.2, 3.0, 3.3]:

    # for kernel in ["linear", "poly", "rbf", "sigmoid"]:
    for kernel in ["linear", "poly", "rbf"]:

        model_svm = create_model(img_train, labels_train, c_parameter=c, kernel=kernel)

        print(f"Valoarea pt \t kernel =\t{kernel}\t si c =\t{c}\t este = \t{(model_svm.predict(imagini_pentru_validation_standardizate) == np.ar

    print("\n---------------------------------------------------------------\n")
```

Rezultatele sunt urmatoarele :

```
Valoarea pt      kernel =   linear   si c = 0.1  este =     0.45012787723785164
Valoarea pt      kernel =   poly     si c = 0.1  este =     0.49531116794543906
Valoarea pt      kernel =   rbf  si c = 0.1  este =     0.453537936913896
Valoarea pt      kernel =   sigmoid  si c = 0.1  este =     0.10485933503836317
```

```
Valoarea pt      kernel =   linear   si c = 0.5  este =     0.43478260869565216
Valoarea pt      kernel =   poly     si c = 0.5  este =     0.49872122762148335
Valoarea pt      kernel =   rbf  si c = 0.5  este =     0.4961636828644501
Valoarea pt      kernel =   sigmoid  si c = 0.5  este =     0.113384484228474

---------------------------------------------------------------

Valoarea pt      kernel =   linear   si c = 1.0  este =     0.42284739982949704
Valoarea pt      kernel =   poly     si c = 1.0  este =     0.49531116794543906
Valoarea pt      kernel =   rbf  si c = 1.0  este =     0.5157715260017051
Valoarea pt      kernel =   sigmoid  si c = 1.0  este =     0.10059676044330776
```

```
Valoarea pt      kernel =   linear   si c = 1.1  este =     0.42625745950554134
Valoarea pt      kernel =   poly     si c = 1.1  este =     0.49531116794543906
Valoarea pt      kernel =   rbf  si c = 1.1  este =     0.5183290707587382
Valoarea pt      kernel =   sigmoid  si c = 1.1  este =     0.10144927536231885

---------------------------------------------------------------

Valoarea pt      kernel =   linear   si c = 1.8  este =     0.4236999147485081
Valoarea pt      kernel =   poly     si c = 1.8  este =     0.4919011082693947
Valoarea pt      kernel =   rbf  si c = 1.8  este =     0.5362318840579711
Valoarea pt      kernel =   sigmoid  si c = 1.8  este =     0.09207161125319693
```

```
Valoarea pt      kernel =   linear   si c = 2.2  este =     0.4288150042625746
Valoarea pt      kernel =   poly     si c = 2.2  este =     0.49872122762148335
Valoarea pt      kernel =   rbf  si c = 2.2  este =     0.5336743393009378

------------------------------------------------------------

Valoarea pt      kernel =   linear   si c = 3.0  este =     0.42625745950554134
Valoarea pt      kernel =   poly     si c = 3.0  este =     0.4927536231884058
Valoarea pt      kernel =   rbf  si c = 3.0  este =     0.5413469735720375

------------------------------------------------------------

Valoarea pt      kernel =   linear   si c = 3.3  este =     0.4236999147485081
Valoarea pt      kernel =   poly     si c = 3.3  este =     0.48678601875532823
Valoarea pt      kernel =   rbf  si c = 3.3  este =     0.5481670929241261
```

Dupa cum se poate observa, valoarea optima a parametrilor este C = 3.3 si 1.1, iar kernelul este standard, adica "rbf".

Astfel ca vom apela functia de write_file pentru a scrie fisierul de iesire :

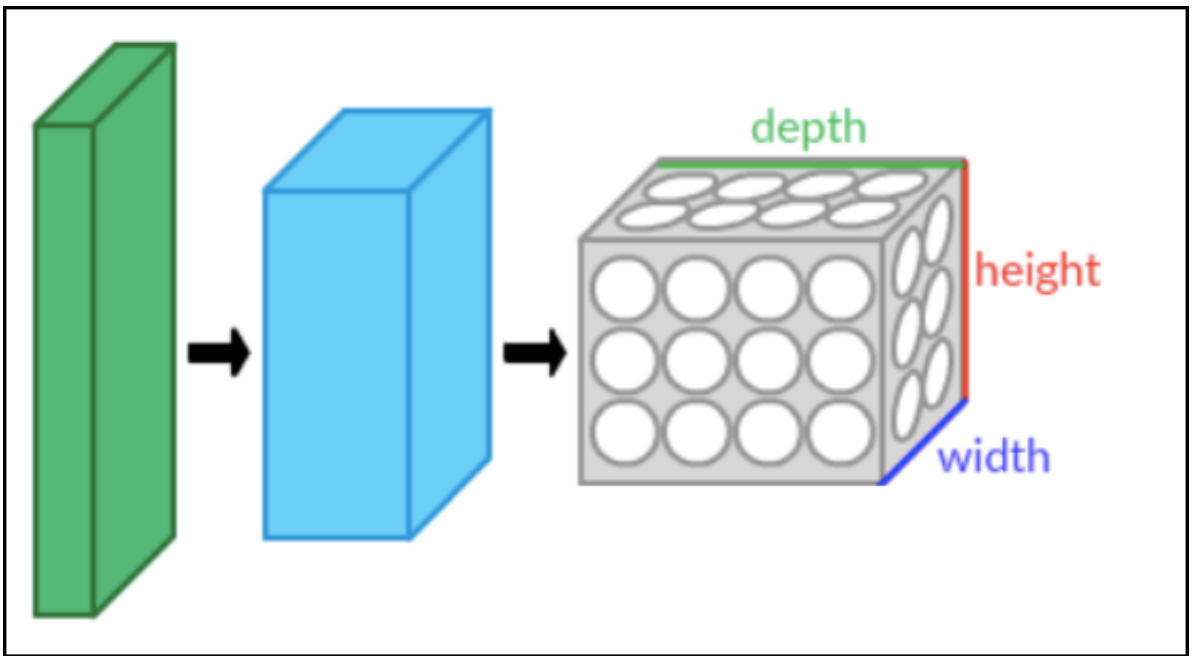
```
# observam ca parametrii optimi pentru c sunt 1.1 si 3.3, iar kernelul este "rbf" asa ca vom face predictii pe ei

pred_img= np.concatenate((img_train, imagini_pentru_validation_standardizate), axis=0)
pred_labels = np.concatenate((labels_train, np.array(fisier_txt_validation.label)), axis=0)

for c in [1.1, 3.3]:

    model_svm = create_model(pred_img, pred_labels, c_parameter=c, kernel = "rbf")

    labels = model_svm.predict(imagini_pentru_test_standardizate)

    write_output_file(labels, "svm" + str(c) + ".txt")
```

2.3 CNN :

CNN sau Retele Neurale Convolutionale sunt retele alcatuite din unul sau mai multe straturi convolutionale (cu sau fara straturi de pooling) si unul sau mai multe straturi complet conectate si foloseste o operatie numita convolutie.

Eroarea se calculeaza la final, iar propagarea inapoi a acesteia se parcurge recursiv de la un

strat la altul, folosind derivatele functiilor si ajustand weight-urile.



Elementul care intra in retea se numeste tensor care este descris de 3 canale : numar de intrari, latimea de intrare si canalele de intrare. Fiecare strat din aceasta retea primeste un input, ii aplica operatia de convolutie si apoi il trimite la urmatorul strat.

2.3.1 Implementare si augmentare date :

Functia de optim folosita este SGD (Stochastic Gradient Descent) care poate fi privita ca o aproximare stocastică a optimizării algoritmului de coborâre pe gradient, deoarece înlocuiește gradientul real cu o estimare a acestuia. Am ales sa nu folosesc momentum.

Intr-o retea neuronala, functia de activare este responsabila pentru transformarea intrarii ponderate insumate de la nod in activarea nodului sau a iesirii pentru acea intrare. Functia de activare folosita de mine este ReLU (Rectified Linear Unit) si este definita ca partea pozitiva a unui numar.

Pentru plotarea datelor am folosit SummaryWriter din biblioteca torch.utils.tensorboard.

Pentru a-mi usura munca, am ales sa rulez aceste modele pe placa video si am folosit torch.device.

```
rulare_device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

print(f" Rulam pe {rulare_device}")
```

Am ales sa imi creez o clasa unde sa definesc layerele si functia de forward. Au fost ajustati numarul de layere, dimensiunea functiei de output si dimensiunea kernelului in functie de teste.

```
class CnnModel_testing1(nn.Module):
    def __init__(self):
        super().__init__()
        self.cnn_layer1_conv = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1),
            nn.BatchNorm2d(num_features=32),
            nn.ReLU(),
        )

        self.cnn_layer1_maxpool_conv = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1),
            nn.BatchNorm2d(num_features=32),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )

        self.cnn_layer2_conv = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1),
            nn.BatchNorm2d(num_features=64),
            nn.ReLU(),
        )

        self.cnn_layer2_maxpool_conv = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),
            nn.BatchNorm2d(num_features=64),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )

        self.cnn_layer3_conv = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
```

```
self.cnn_layer2_maxpool_conv = nn.Sequential(
    nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),
    nn.BatchNorm2d(num_features=64),
    nn.ReLU(),
    nn.MaxPool2d(2, 2)
)

self.cnn_layer3_conv = nn.Sequential(
    nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1),
    nn.BatchNorm2d(num_features=128),
    nn.ReLU(),
)

self.cnn_layer3_maxpool_conv = nn.Sequential(
    nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1),
    nn.BatchNorm2d(num_features=128),
    nn.ReLU(),
    nn.MaxPool2d(2, 2)
)

self.cnn_reduction_function1 = nn.Linear(in_features=512, out_features=7)

# self.cnn_reduction_function1 = nn.Linear(in_features=512, out_features=128)
#
# self.cnn_reduction_function2 = nn.Linear(in_features=128, out_features=7)

self.relu_function = nn.ReLU()
```

```python
def forward(self, x):
    # print(f'in: {x.shape}')
    x = self.cnn_layer1_conv(x)
    # print(f'c1: {x.shape}')

    x = self.cnn_layer1_maxpool_conv(x)
    # print(f'c1 maxpool: {x.shape}')

    x = self.cnn_layer2_conv(x)
    # print(f'c2: {x.shape}')

    x = self.cnn_layer2_maxpool_conv(x)
    # print(f'c2 maxpool: {x.shape}')

    x = self.cnn_layer3_conv(x)
    # print(f'c3: {x.shape}')

    x = self.cnn_layer3_maxpool_conv(x)
    # print(f'c3 maxpool: {x.shape}')

    x = torch.flatten(x, start_dim=1)

    x = self.relu_function(x)

    x = self.cnn_reduction_function1(x)

    # print(f'f1: {x.shape}')

    # x = self.relu_function(x)
```

Pentru a obtine datasetul pe care sa fac antrenarea, validarea si testarea a trebuit sa definesc o alta clasa si o metoda care intoarce datasetul tinand cont de mai mult parametrii.

```python
class Data_Class_For_CNN(Dataset):
    def __init__(
            self,
            files_directory,
            file_txt_with_img_names,
            transformari_de_baza=None,
            augmentari_date=None,
            mode_de_folosire="test"
    ):
        self.files_directory = files_directory
        self.mode_images, self.mode_labels = self.get_imgs_and_labels(file_txt_with_img_names)
        self.transformari_de_baza = transformari_de_baza
        self.augmentari_date = augmentari_date
        self.mode_de_folosire = mode_de_folosire

    @staticmethod
    def get_imgs_and_labels(file_txt_with_img_names):
        file_lines = [x.strip() for x in open(file_txt_with_img_names, 'r').readlines()]

        mode_images = [lne.split(',')[0] for lne in file_lines]
        mode_labels = [int(lne.split(',')[1]) for lne in file_lines]

        return mode_images, mode_labels
```

```python
    def __getitem__(self, index_to_image):
        path_to_image = os.path.join(self.files_directory, self.mode_images[index_to_image])

        this_image = Image.open(path_to_image)

        clasa = self.mode_labels[index_to_image]

        this_image_transformed = self.transformari_de_baza(this_image)

        if self.mode_de_folosire == "train":
            this_image_transformed = self.augmentari_date(this_image_transformed)

        return this_image_transformed, clasa

    def __len__(self):
        return len(self.mode_images)
```

Metoda care ne intoarce datasetul are definite si augmentarile de date. Am ales RandomHorizontalFlip, RandomInvert si RandomAutocontrast.

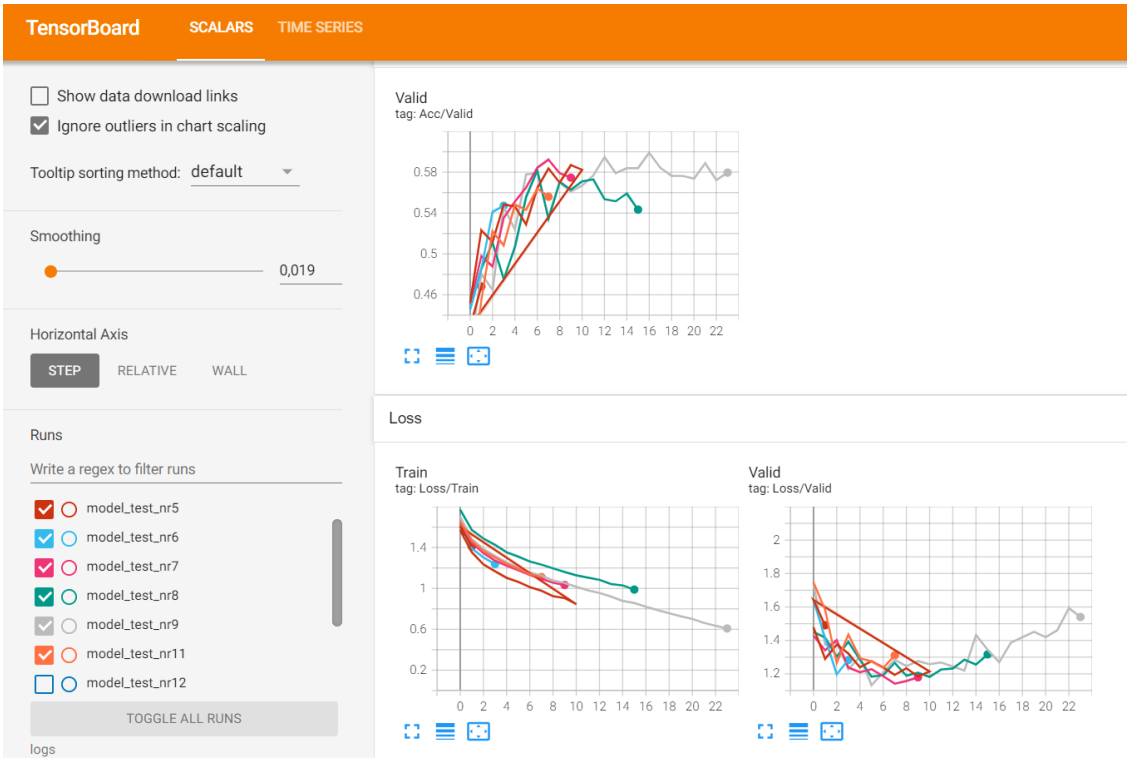
```python
def get_dataloader(files_directory, file_txt_with_img_names, mod_de_folosire, medie, deviatie_standard, dimensiune_batch=2, rearanjari=False):
    transformari_de_baza = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(
            mean=medie,  # (R, G, B)
            std=deviatie_standard  # (R, G, B)
        )
    ])

    augmentari_date = transforms.Compose([
        transforms.RandomHorizontalFlip(p=0.2),
        transforms.RandomInvert(p=0.45),
        transforms.RandomAutocontrast(p=0.45),
    ])

    data_for_return = Data_Class_For_CNN(
        files_directory=files_directory,
        file_txt_with_img_names=file_txt_with_img_names,
        transformari_de_baza=transformari_de_baza,
        augmentari_date=augmentari_date,
        mode_de_folosire=mod_de_folosire
    )

    return DataLoader(
        data_for_return,
        batch_size=dimensiune_batch,
        shuffle=rearanjari
    )
```

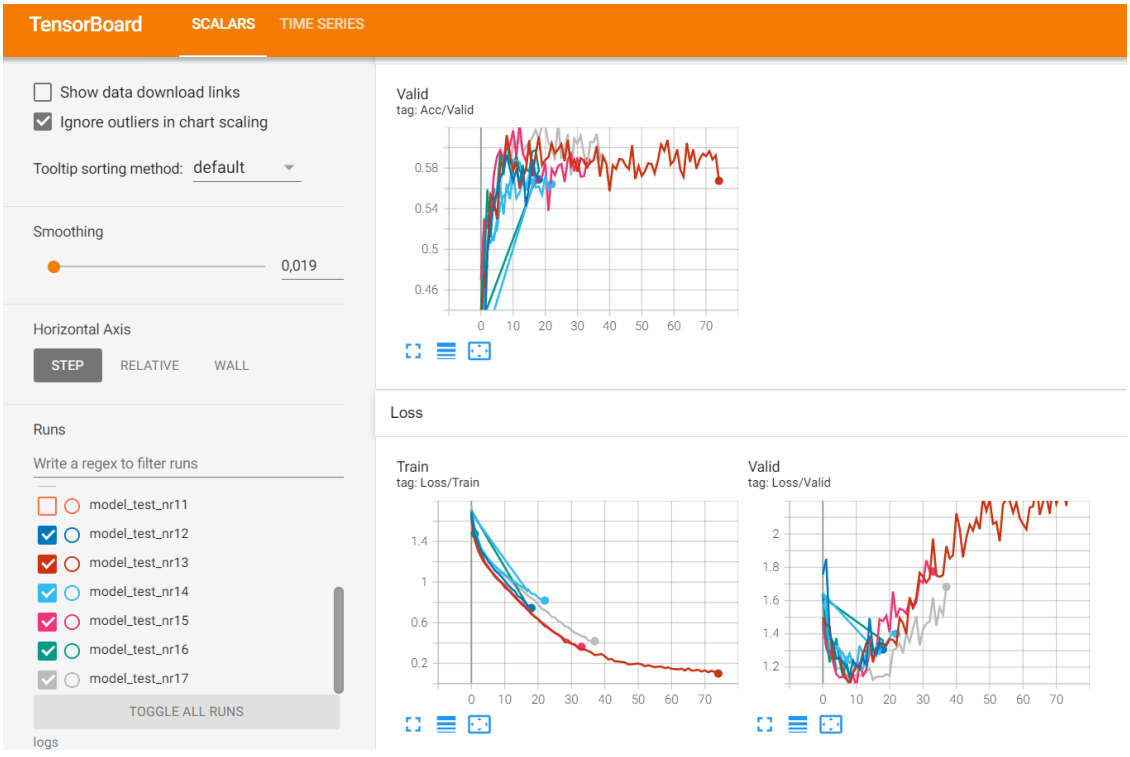
2.3.2 Teste si acuratete :

Am testat mai multe modele, modificand urmatorii hiperparametrii :

1. Numarul de layere
2. Dimensiunea kernelului
3. Probabilitatile functiilor de augmentare
4. Batch size
5. Learning rate

Rezultatele modelelor ajustate si testate :





Am salvat modelul cu acuratetea cea mai buna, dar si pe cel cu loss-ul cel mai bun, astfel scriind fisierul de iesire folosind functia urmatoare :

```python
def write_output_file(running_device, date_de_test, fisier_txt_test, k):
    f = open("cnn_" + str(k) + ".txt", 'w')
    f.write("id,label\n")
    i = 0

    cnn_model = CnnModel_testing1()
    cnn_model.to(running_device)
    cnn_model.load_state_dict(torch.load("../models/model_test_cnn_" + str(k)))

    with torch.no_grad():

        for imgs, _ in date_de_test:

            imgs = imgs.to(running_device)

            output = cnn_model(imgs)

            _, labeluri_prezise = torch.max(output.data, 1)

            for label in labeluri_prezise:
                f.write(f"{fisier_txt_test.id[i]},{label}\n")
                i += 1

    f.close()
```

Submisiile au fost facute folosind mai multe modele testate, dar procentul maxim a folosit urmatorii parametrii : batch_size = 8, probabilitatile augmentarilor au fost : 0.2 pentru RandomHorizontalFlip, 0.45 pentru RandomInvert si 0.45 pentru RandomAutocontrast), kernel=3 pentru toate layerele, learning_rate = 0.01, 6 layere convolutionale (3 cu MaxPool2d) si o functie de reducere a dimensiunii de la 521 la 7.

3 Referinte :

1. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
2. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
3. https://en.wikipedia.org/wiki/Support-vector_machine
4. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

5. https://en.wikipedia.org/wiki/Convolutional_neural_network#Distinguishing_features
6. https://en.wikipedia.org/wiki/Torch_(machine_learning)
7. https://pytorch.org/
8. https://pytorch.org/docs/stable/tensorboard.html
9. https://pytorch.org/vision/stable/transforms.html
10. https://pytorch.org/tutorials/beginner/basics/data_tutorial.html
11. https://pytorch.org/docs/stable/tensorboard.html
12. https://pytorch.org/docs/stable/generated/torch.optim.SGD.html
13. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.htm