Vîrlan Mihnea-Alexandru

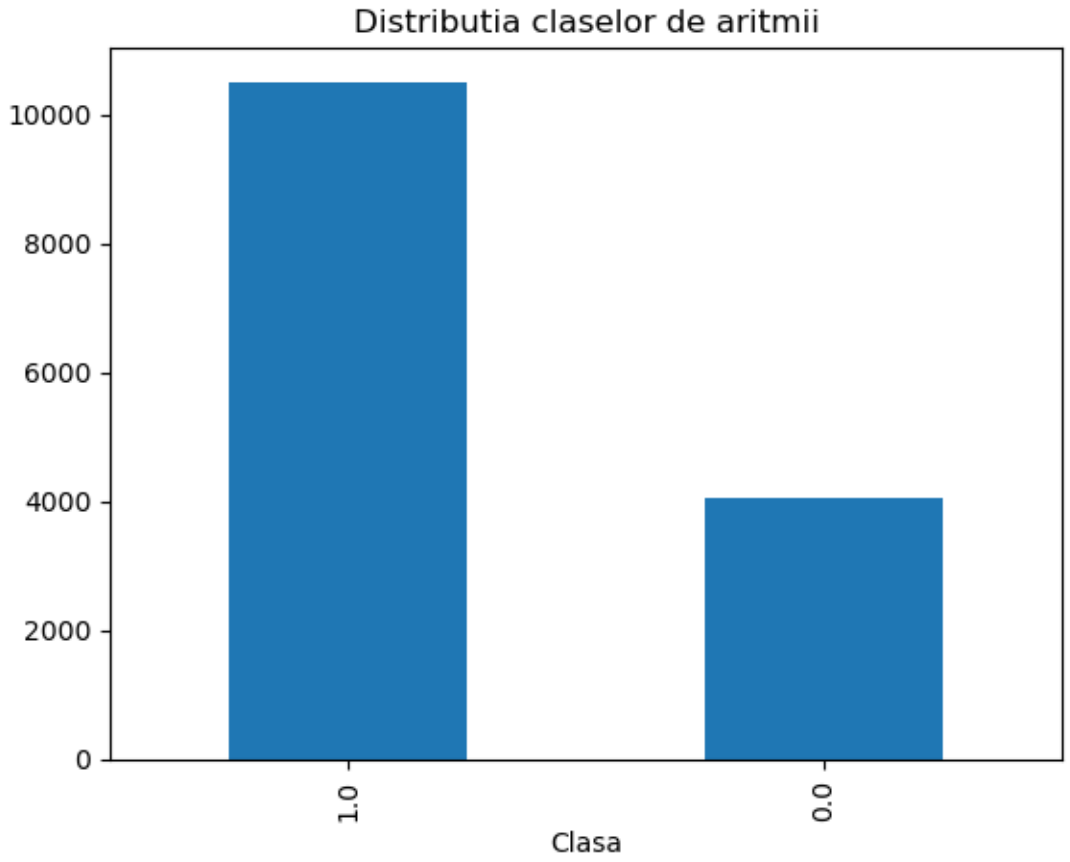
grupa 341C4

# *Raport tema 2 - Învățare automată*

## 1. Analiza exploratorie a datelor

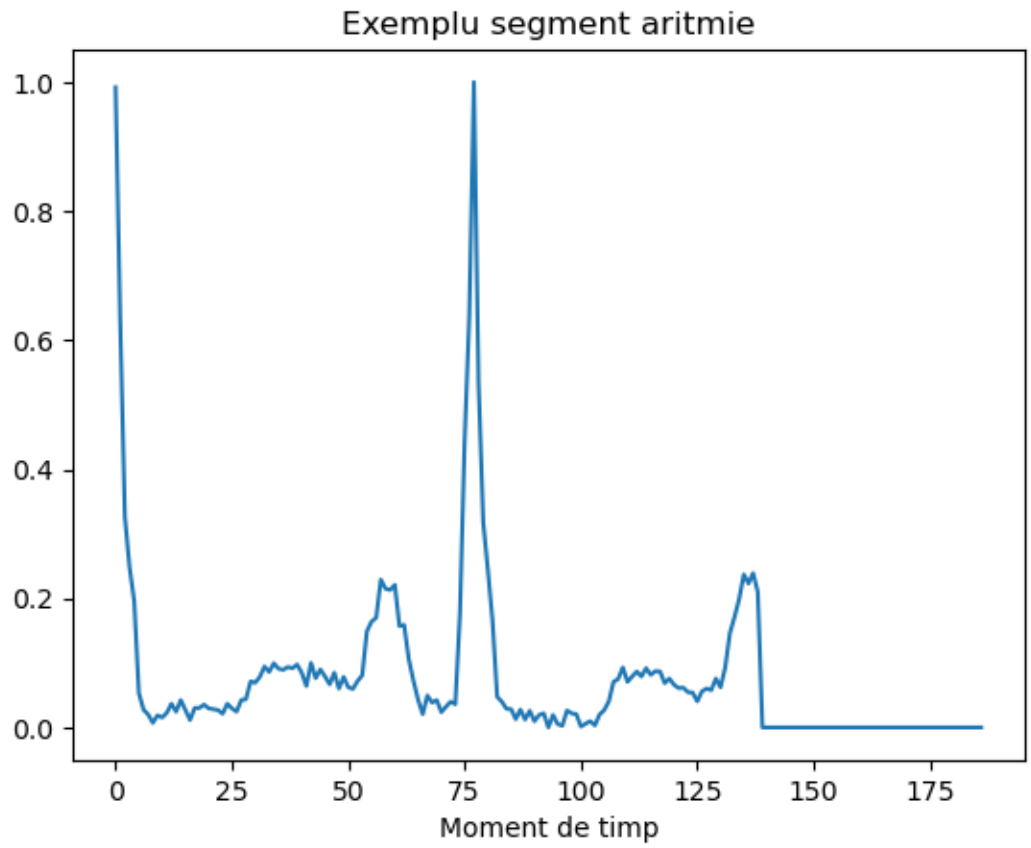
1.1. Echilibrul de clase

Acumulând cele două seturi de date pentru bătăile normale, respectiv anormale de inimă, obținem următoarea dsitribuție a claselor

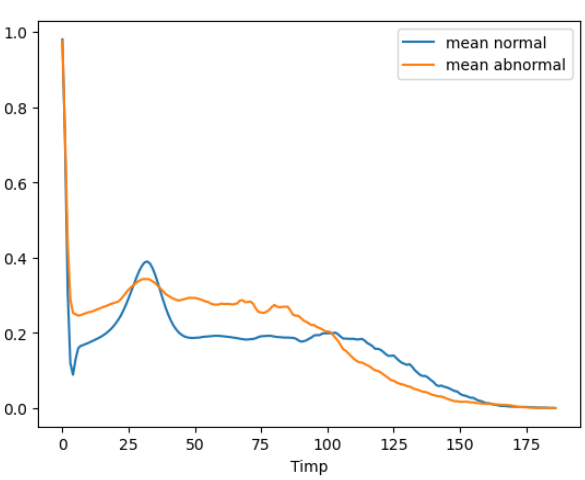


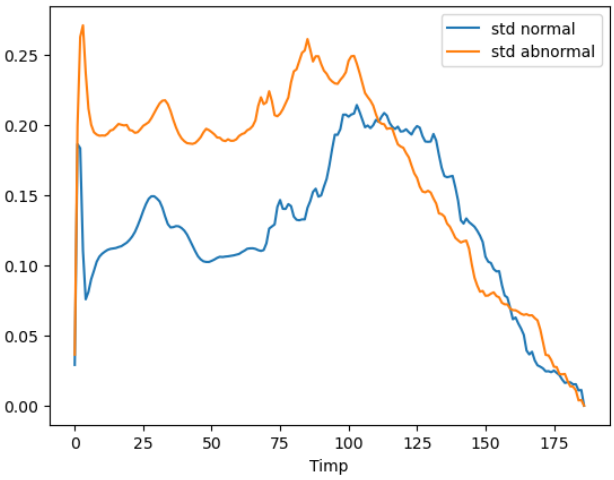

Se constată un dezechilibru major, numărul de bătăi de inimă corespunzător aritmiilor fiind mult mai mare decât numărul bătăilor normale de inimă.

1.2. Afișare de serii de timp




1.3. Grafice medie și deviație standard




Observăm că, atât pentru medie, cât și pentru deviația standard, acestea au un ritm de evoluție similar: pentru aritmii se constată valori mai mari decât cele din clasa opusă până la momentul de timp 100 (aproximativ), urmând ca acelea să devină mai mici, iar apoi să ajungă la un punct comun final.

## 2. Evaluarea arhitecturilor neurale

2.1 Rețeaua de convoluție

Pentru prescurtare voi nota cu conv_{i}, respectiv setup_{i}, arhitectura convoluțională, respectiv setupul care include dimensiunea batch-ului, learning-rate-ul, learning rate scheduler-ul și numărul de epoci. Peste tot se va folosi optimizatorul Adam

conv_1

```
conv_net=nn.Sequential(
    nn.Conv1d(in_channels=1,out_channels=15,kernel_size=3,stride=1), #1,187 ->(15,(187-3)/1+1) ->(5,185)
    nn.ReLU(),
    nn.AvgPool1d(kernel_size=5), # =>(15,37)
    nn.Dropout(p=0.1),
    nn.Conv1d(in_channels=15,out_channels=10,kernel_size=5,stride=1,padding=1), # (15,37) ->(10,35)
    nn.Tanh(),
    nn.Flatten(1),
    nn.Linear(350,100),
    nn.Dropout(p=0.25),
    nn.ReLU(),
    nn.Linear(100,20),
    nn.ReLU(),
    nn.Linear(20,1)
)
```

setup_1 : exponentialLR cu gamma=0.99, batch_size=32,lr=1e-2, număr epoci=100

conv_2 e acelasi cu conv_1

setup_2 schimbam batch_size la 64

conv_3 : schimbăm la conv_1 kernel_size-ul average pooling la 3

setup_3: batch_size=64,num_epochs=100,

conv_4: de la conv_3 schimbăm doar primul dropout la p=0.2

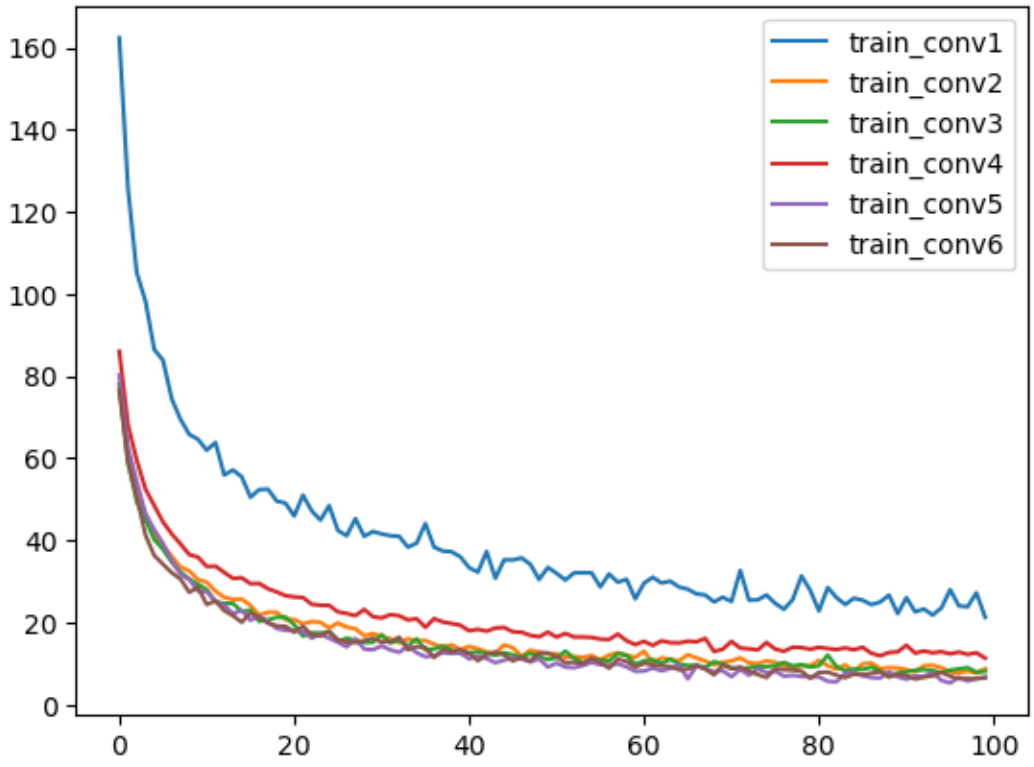
setup_4 la fel ca la setup_3

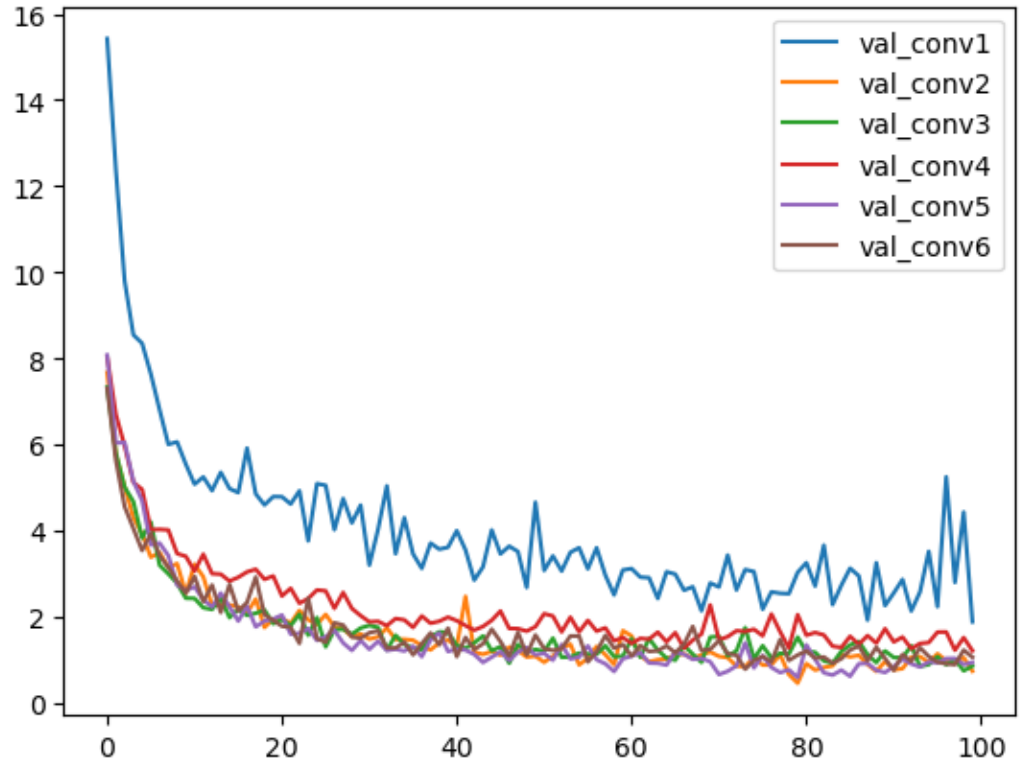
conv_5: punem la loc primul dropout la p=0.1, schimbăm la ultimul strat convoluțional kernel_size la 3 și stride la 2

setup_5 la fel ca la setup_4

conv_6: similar ca la conv_5, dar stride-ul ultimului strat de convoluție este 1

setup_6: la fel ca setup_5





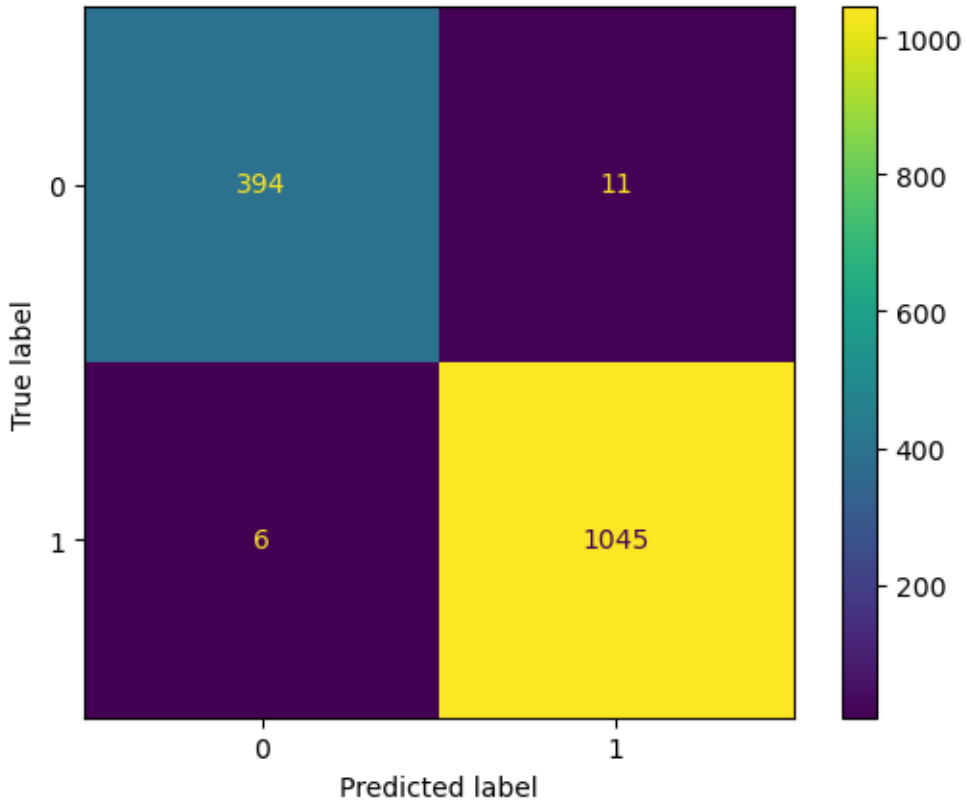

| Arhitectura+ setup | Precizie clasa 0 | Recall clasa 0 | F1 clasa 0 | Precizie clasa 1 | Recall clasa 1 | F1 clasa 1 |
|---|---|---|---|---|---|---|
| conv_1,setup_1 | 0.9509 | 0.9556 | 0.9532 | 0.9828 | 0.9810 | 0.9819 |
| conv_2,setup_2 | 0.9701 | 0.9630 | 0.9665 | 0.9858 | 0.9886 | 0.9872 |
| conv_3,setup_3 | 0.9633 | 0.9728 | 0.9681 | 0.9895 | 0.9857 | 0.9876 |
| conv_4,setup_4 | 0.9531 | 0.9531 | 0.9531 | 0.9819 | 0.9819 | 0.9819 |
| conv_5,setup_5 | **0.9850** | **0.9728** | **0.9789** | **0.9896** | **0.9943** | **0.9919** |
| conv_6,setup_6 | 0.9724 | 0.9580 | 0.9652 | 0.9839 | 0.9895 | 0.9867 |

Analizând tabelul și graficul se poate vedea că:

- un batch size mare înseamnă și o varietate mare a datelor, dar și a împărțirii pe clase, deci e firesc să fie loss-ul mai mic

- dropout-ul de la ultimul layer convoluțional afectează rezultatele, mai exact, dacă e mai mare cresc șansele de overfitting
- impactul kernelul-ui de la average pooling arată că o dimensiune mai mică a acestuia înseamnă că scade șansa de false negative, acest lucru observându-se prin creșterea scorurilor F1
- un stride mai mare a crescut scorurile prin faptul că rețeaua vede mai multe noi pattern-uri ale semnalului, lăsându-le pe cele vechi în urmă

Arătăm matricea de confuzie pentru a cincea arhitectură și setup




2.2. Arhitectura MLP pe setul de date PTB diagnostic ECG

Peste tot se foloseste optimizatorul Adam cu learning rate initial de 1e-2

MLP_{i} denota o configuratie de MLP la experimentul i

setup_{i} reprezinta setup la experimentul i

MLP_{1}:

```
MLP_net=nn.Sequential(
    nn.Linear(187,100),
    nn.ReLU(),
    nn.Linear(100,20),
    nn.ReLU(),
    nn.Linear(20,1)
)
```

setup_{1}:exponentialLR cu gamma=0.99, batch_size=64, numar de epoci 100

MLP_{2}: schimbam ultima functie de activare neliniara la Tanh

setup_{2}: ca la setup_{1}

MLP_{3}:

```
MLP_net=nn.Sequential(
    nn.Linear(187,100),
    nn.ReLU(),
    nn.Linear(100,250),
    nn.ReLU(),
    nn.Linear(250,20),
    nn.Tanh(),
    nn.Linear(20,1)
)
```

setup_{3}, identic ca la {2}

MLP_{4}:

```
MLP_net=nn.Sequential(
    nn.Linear(187,100),
    nn.ReLU(),
    nn.Linear(100,250),
    nn.Dropout(p=0.15),
    nn.ReLU(),
    nn.Linear(250,20),
    nn.Tanh(),
    nn.Linear(20,1)
)
```

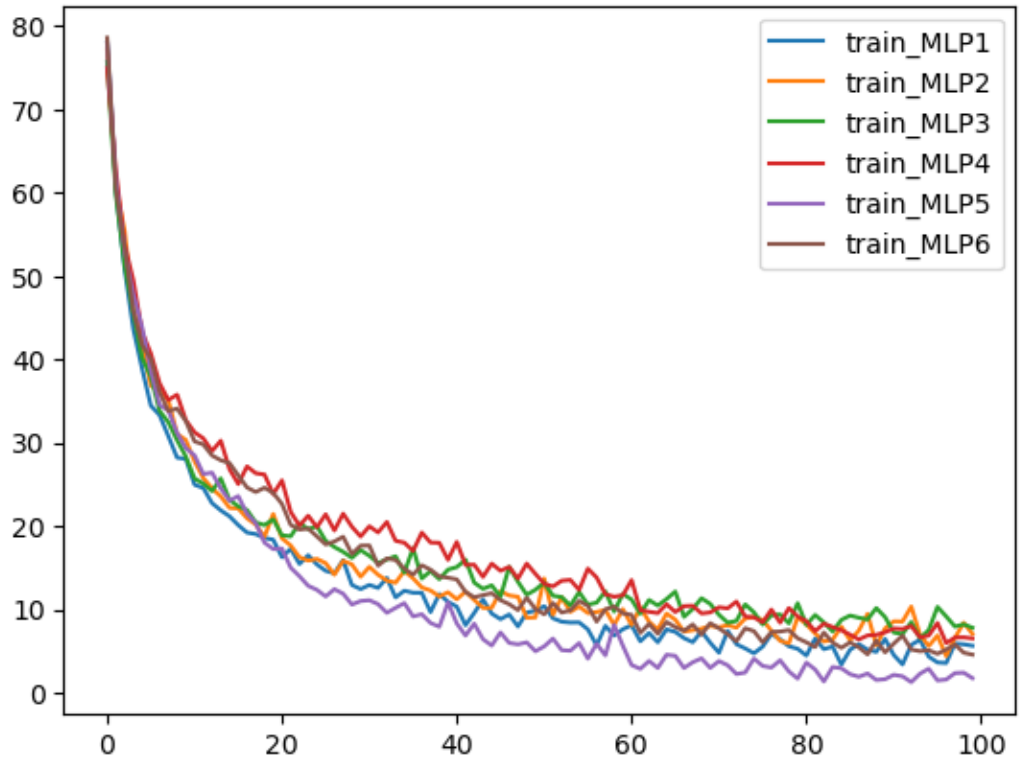
setup_{4}: schimbam gamma la 0.8

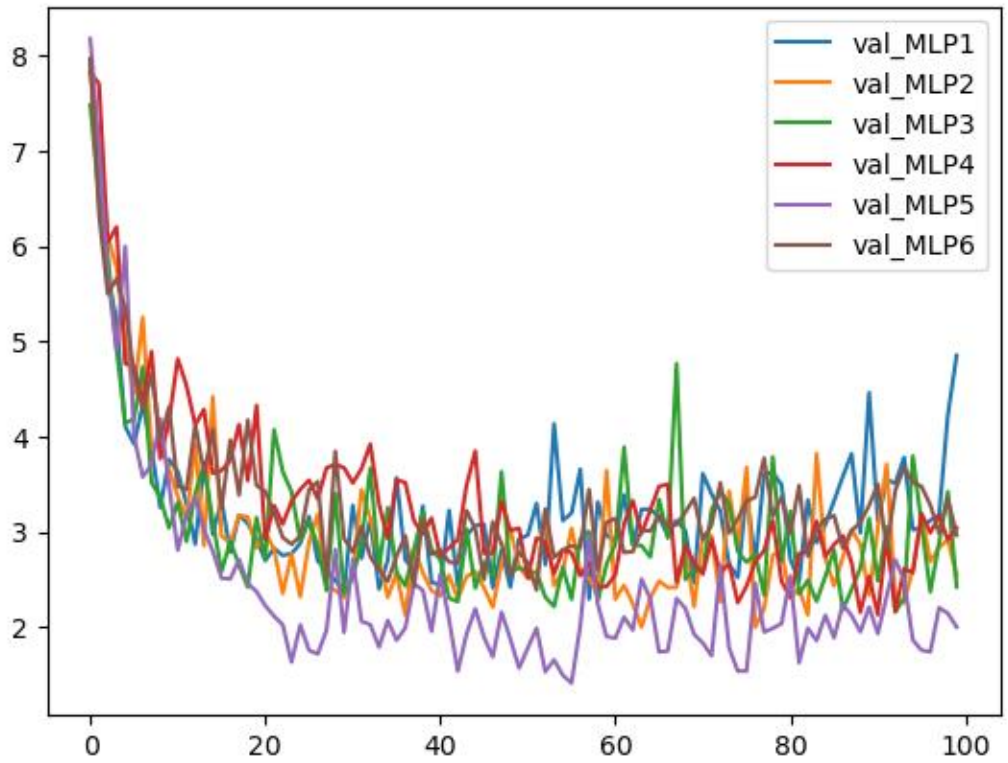
MLP_{5}:

```
MLP_net=nn.Sequential(
    nn.Linear(187,100),
    nn.Tanh(),
    nn.Linear(100,250),
    nn.Dropout(p=0.3),
    nn.ReLU(),
    nn.Linear(250,20),
    nn.ReLU(),
    nn.Linear(20,1)
)
```

setup_{5}: idem {4}

MLP_{6}: Schimbam prima activare neliniara la ReLU






| Arhitectura+ setup | Precizie clasa 0 | Recall clasa 0 | F1 clasa 0 | Precizie clasa 1 | Recall clasa 1 | F1 clasa 1 |
|---|---|---|---|---|---|---|
| MLP_{1},setup_{1} | 0.9497 | 0.8864 | 0.9170 | 0.9573 | 0.9819 | 0.9695 |
| MLP_{2},setup_{2} | 0.9229 | 0.9457 | 0.9341 | 0.9789 | 0.9696 | 0.9742 |
| MLP_{3},setup_{3} | 0.9385 | 0.9037 | 0.9208 | 0.9634 | 0.9772 | 0.9702 |
| MLP_{4},setup_{4} | 0.9444 | 0.9235 | 0.9338 | 0.9708 | 0.9791 | 0.9749 |
| MLP_{5},setup_{5} | **0.9668** | 0.9333 | **0.9497** | 0.9746 | **0.9876** | **0.9811** |
| MLP_{6},setup_{6} | 0.9426 | 0.9333 | 0.9380 | 0.9744 | 0.9781 | 0.9763 |

Observând graficele și tabelul putem spune că:

- un MLP format din mai multe straturi cu mulți neuroni surprinde mai bine datele
- dropout-ul aici trebuie să fie mai mare pentru a reduce overfitt-ul pe cât posibil

- funcții de activare neliniară care nu ignoră date, precum tangenta hiperbolică sunt benefice

Pentru a cincea combinație de arhitectură și setup arătăm matricea de confuzie