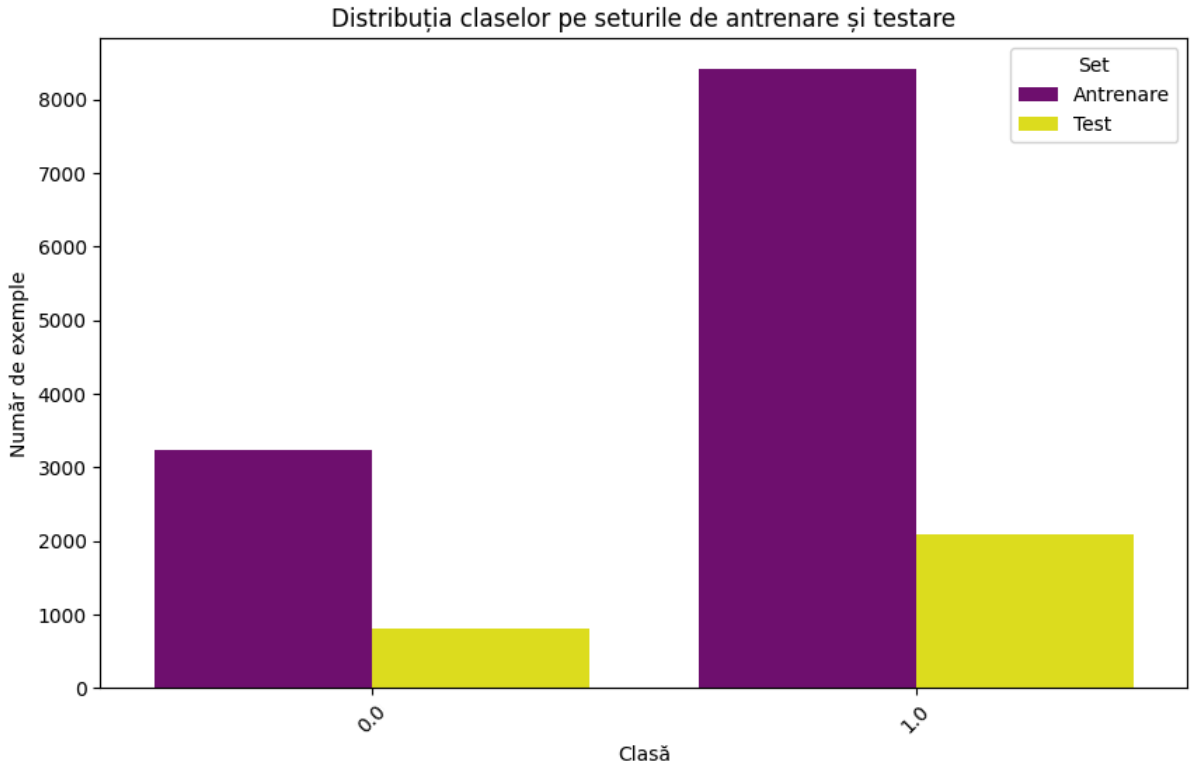
Cotelin Maria-Diana
342C4

Învățare Automată
Tema 2 - 2024

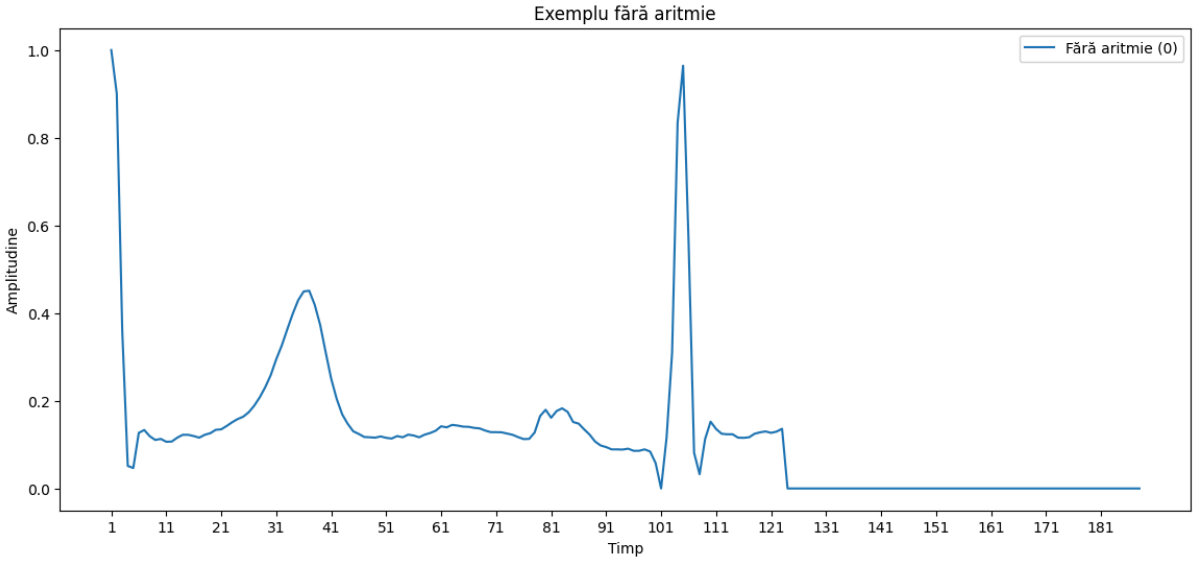
1. Explorarea Datelor

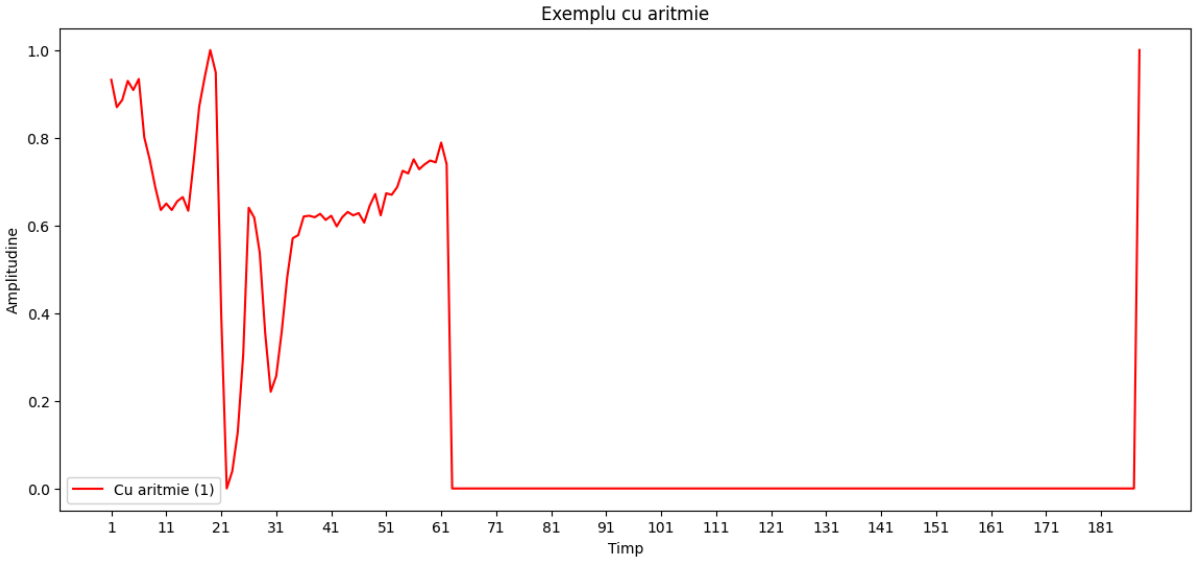
a. Analiza echilibrului de clase

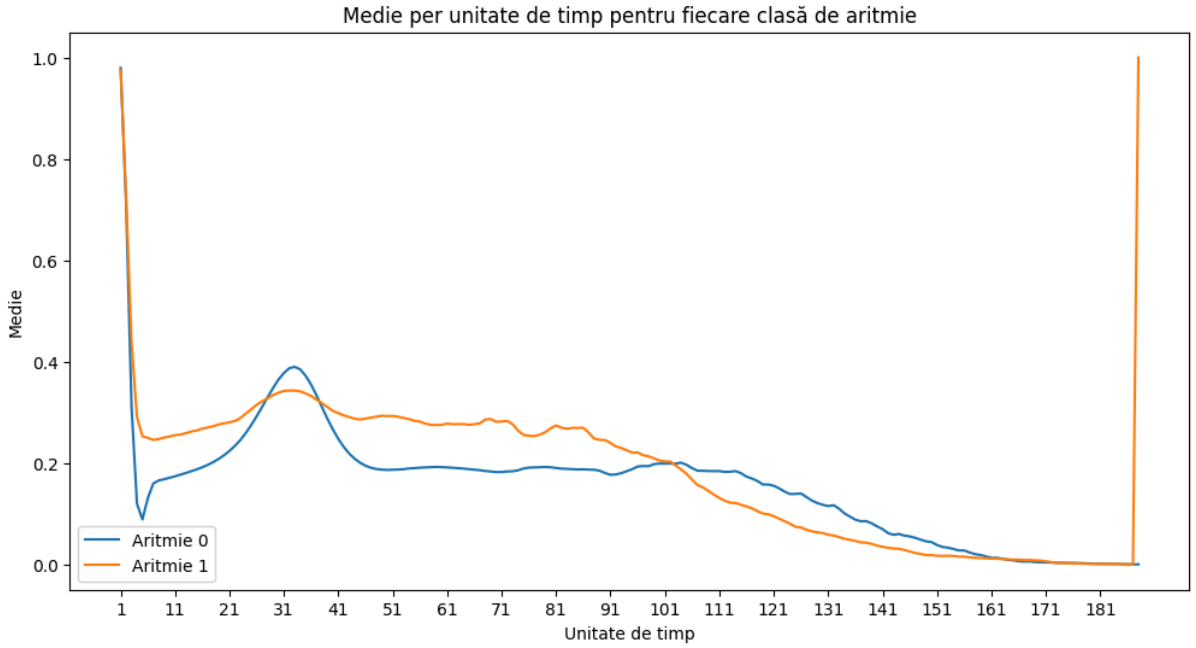


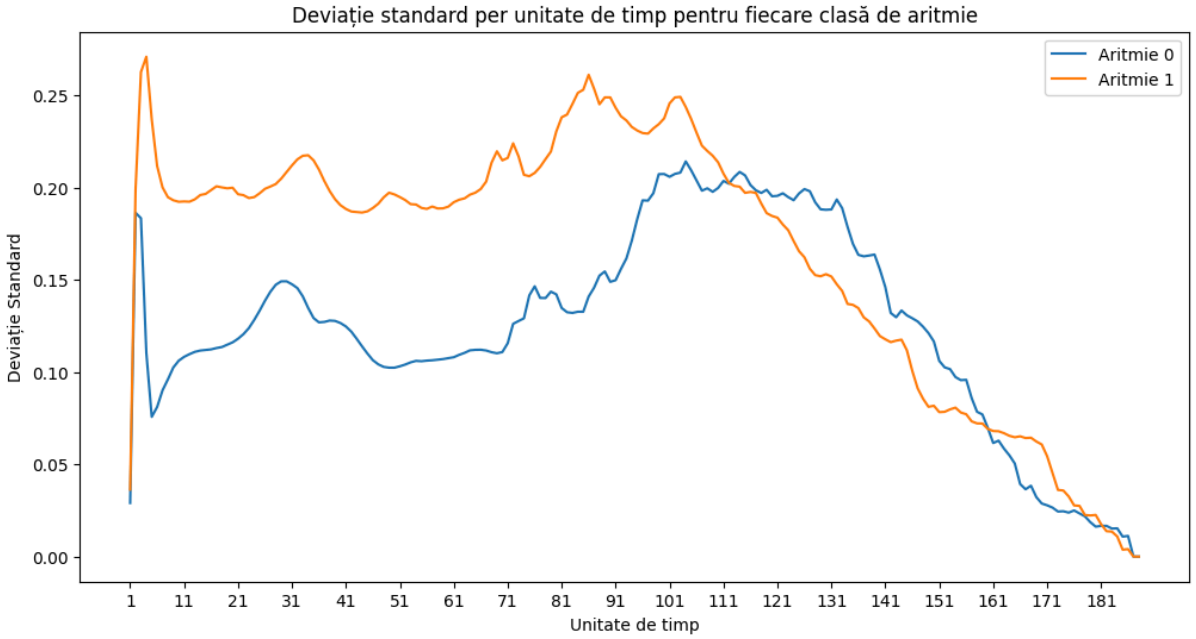
Clasele sunt distribuite proporțional pe setul de test fata de cel de antrenare.

b. Vizualizarea seriilor de timp




Exemplu cu aritmie
Amplitudine
Timp
Cu aritmie (1)
0.0
0.2
0.4
0.6
0.8
1.0
1
11
21
31
41
51
61
71
81
91
101
111
121
131
141
151
161
171
181


Medie per unitate de timp pentru fiecare clasă de aritmie
Medie
Unitate de timp
Aritmie 0
Aritmie 1
0.0
0.2
0.4
0.6
0.8
1.0
1
11
21
31
41
51
61
71
81
91
101
111
121
131
141
151
161
171
181



2. Modele de Retele

a. MLP

i. Patients

```
def create_mlp_model(input_dim, layers, units, activation='relu',
learning_rate=0.001):
    model = Sequential()
    model.add(Dense(units=units, activation=activation,
input_dim=input_dim))

    for _ in range(layers):
        model.add(Dense(units=units, activation=activation))

    model.add(Dense(units=7, activation='softmax'))
    model.compile(optimizer=Adam(learning_rate=learning_rate),
loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

Model:

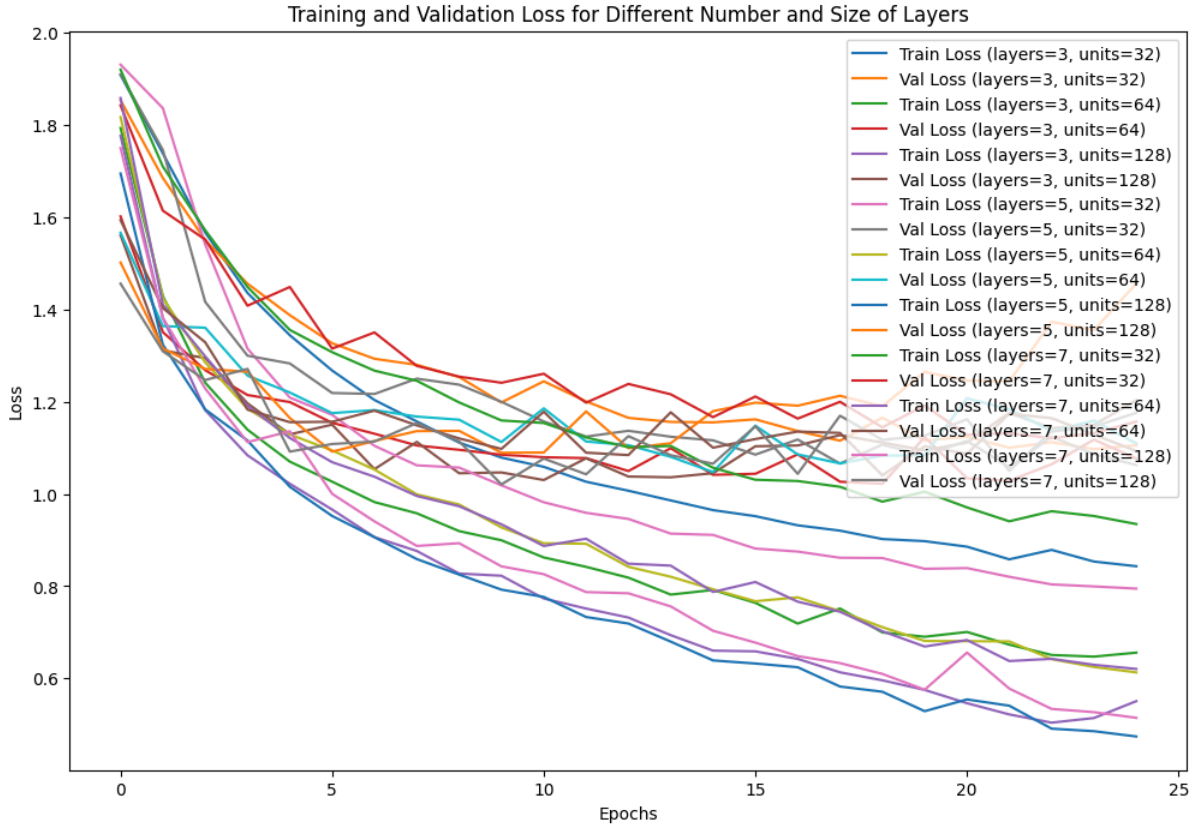
1. Input Layer:
   - Dense(units=units, activation=activation, input_dim=input_dim): Un strat dens (fully connected) cu numărul de neuroni specificat prin units, funcția de activare specificată prin activation, și dimensiunea intrării determinată de input_dim.
2. Hidden Layers:

- Dense(units=units, activation=activation): În fiecare iterare a buclei, se adaugă un strat dens cu numărul de neuroni specificat prin units și funcția de activare specificată prin activation.

3. Output Layer:
    - Dense(units=7, activation='softmax'): Un strat dens cu 7 neuroni (presupunând că avem 7 clase de ieșire), cu activare softmax pentru clasificare multiclasă.

Parametrii selectati:
Optimizator: adam
Batch size: 24
Epochs: 25
loss: categorical_crossentropy
Learning Rate: 0.001
Activation: relu

Rezultate:

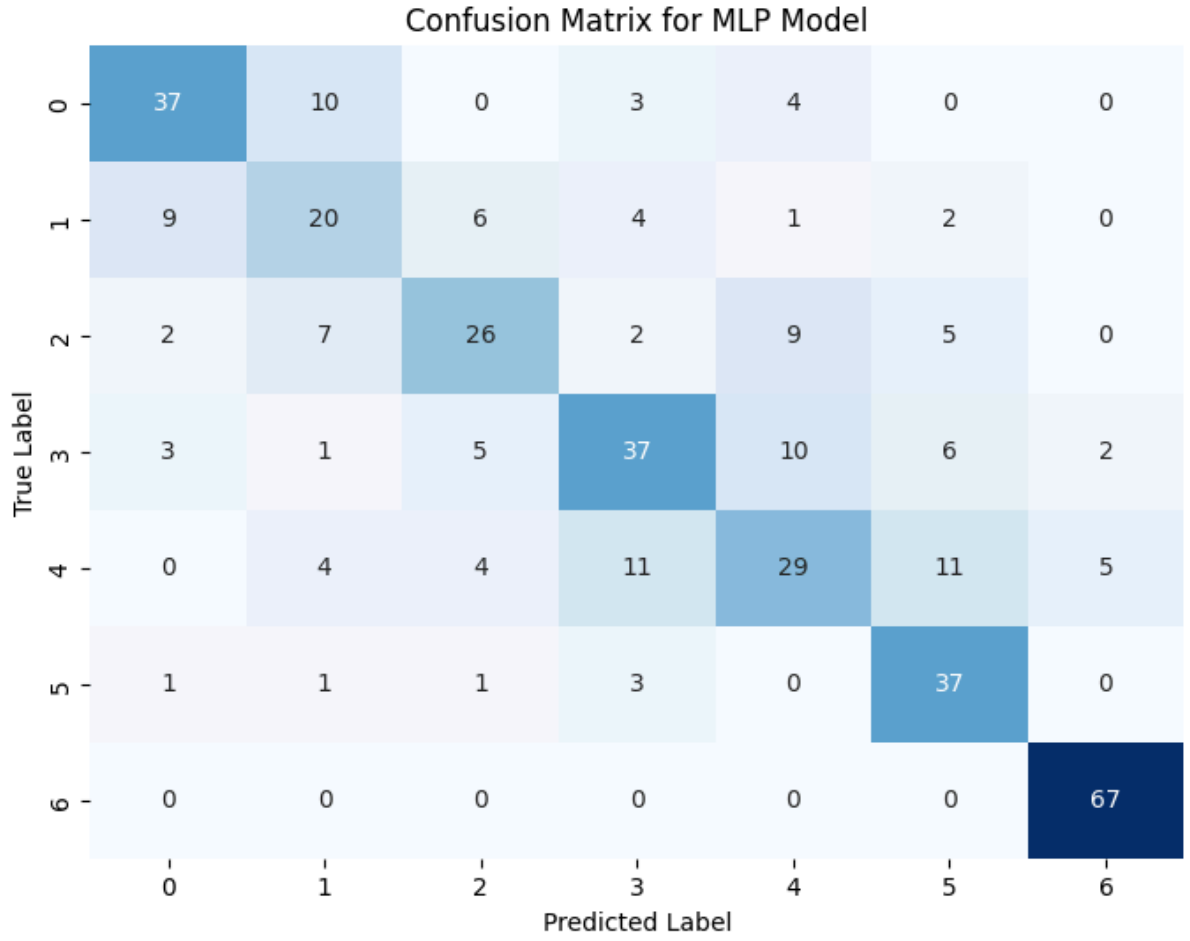


Cel mai mic loss este pentru 7 layere si 32 units pentru setul de validare, iar pentru train pentru 5 layers si 128 units. Se observa ca au aceeasi tendinta de scadere toate.

	layers	units	loss	accuracy	precision_class_0	recall_class_0	f1_class_0	precision_class_1	recall_class_1	f1_class_1	precision_class_2	recall_class_2	f1_class_2	precision_class_3
0	3	32	1.128247	0.620779	0.641791	0.796296	0.710744	0.433333	0.309524	0.361111	0.564103	0.431373	0.488889	0.638889
1	3	64	1.107930	0.672727	0.750000	0.777778	0.763636	0.571429	0.571429	0.571429	0.600000	0.529412	0.562500	0.593220
2	3	128	1.251986	0.683117	0.681159	0.870370	0.764228	0.676471	0.547619	0.605263	0.446154	0.568627	0.500000	0.769231
3	5	32	1.026120	0.657143	0.812500	0.722222	0.764706	0.448980	0.523810	0.483516	0.543478	0.490196	0.515464	0.605263
4	5	64	1.146200	0.680519	0.736842	0.777778	0.756757	0.472727	0.619048	0.536082	0.758621	0.431373	0.550000	0.636364
5	5	128	1.249394	0.659740	0.891304	0.759259	0.820000	0.479167	0.547619	0.511111	0.566038	0.588235	0.576923	0.539474
6	7	32	1.131752	0.610390	0.636364	0.777778	0.700000	0.362069	0.500000	0.420000	0.611111	0.215686	0.318841	0.523810
7	7	64	1.107266	0.651948	0.750000	0.722222	0.735849	0.387097	0.571429	0.461538	0.545455	0.470588	0.505263	0.741935
8	7	128	1.204360	0.664935	0.775862	0.833333	0.803571	0.478261	0.523810	0.500000	0.617647	0.411765	0.494118	0.627451

precision_class_3	recall_class_3	f1_class_3	precision_class_4	recall_class_4	f1_class_4	precision_class_5	recall_class_5	f1_class_5	precision_class_6	recall_class_6	f1_class_6
0.638889	0.359375	0.460000	0.515625	0.515625	0.515625	0.527027	0.906977	0.666667	0.880000	0.985075	0.929577
0.593220	0.546875	0.569106	0.586957	0.421875	0.490909	0.596774	0.860465	0.704762	0.893333	1.000000	0.943662
0.769231	0.468750	0.582524	0.590164	0.562500	0.576000	0.720930	0.720930	0.720930	0.905405	1.000000	0.950355
0.605263	0.359375	0.450980	0.584615	0.593750	0.589147	0.629032	0.906977	0.742857	0.870130	1.000000	0.930556
0.636364	0.546875	0.588235	0.627451	0.500000	0.556522	0.622951	0.883721	0.730769	0.870130	1.000000	0.930556
0.539474	0.640625	0.585714	0.594595	0.343750	0.435644	0.581818	0.744186	0.653061	0.928571	0.970149	0.948905
0.523810	0.515625	0.519685	0.604651	0.406250	0.485981	0.600000	0.837209	0.699029	0.857143	0.985075	0.916667
0.741935	0.359375	0.484211	0.596774	0.578125	0.587302	0.616667	0.860465	0.718447	0.905405	1.000000	0.950355
0.627451	0.500000	0.556522	0.542373	0.500000	0.520325	0.587302	0.860465	0.698113	0.905405	1.000000	0.950355

Cele mai bune rezultate pentru 5 layere si 64 units pentru acuratete. Am realizat si matricea de confuzie.



Clasa 6 a fost prezisa cel mai bine, asa cum reiese si din tabel.

ii. PTB Diagnostic ECG

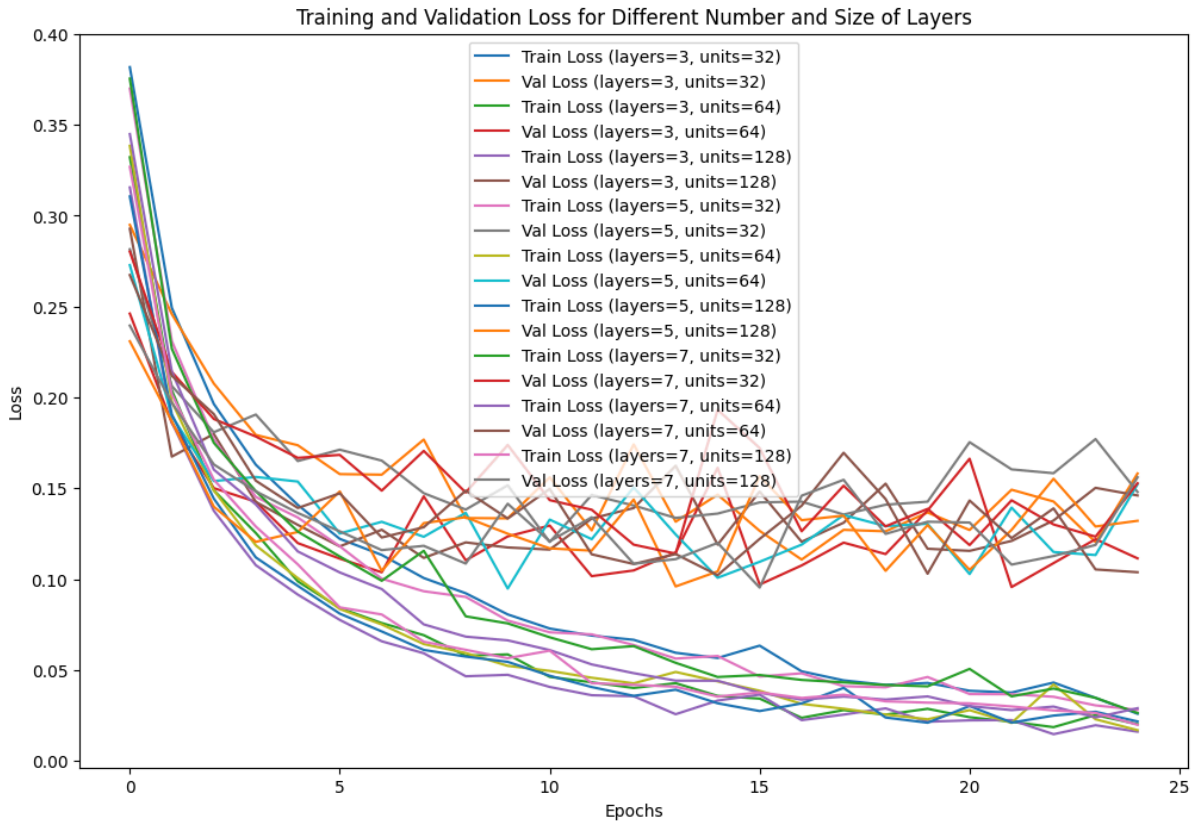
```
def create_mlp_model_2(input_dim, layers, units, activation='relu',
learning_rate=0.001):
   model = Sequential()
   model.add(Dense(units=units, activation=activation,
input_dim=input_dim))
```

```
    for _ in range(layers - 1):
        model.add(Dense(units=units, activation=activation))

    model.add(Dense(units=1, activation='sigmoid'))
    model.compile(optimizer=Adam(learning_rate=learning_rate),
loss='binary_crossentropy', metrics=['accuracy'])

    return model
```

Model:
ca mai sus

Parametrii:
Optimizator: adam
Batch size: 24
Epochs: 15
loss: binary_crossentropy
Learning Rate: 0.001
Activation: relu

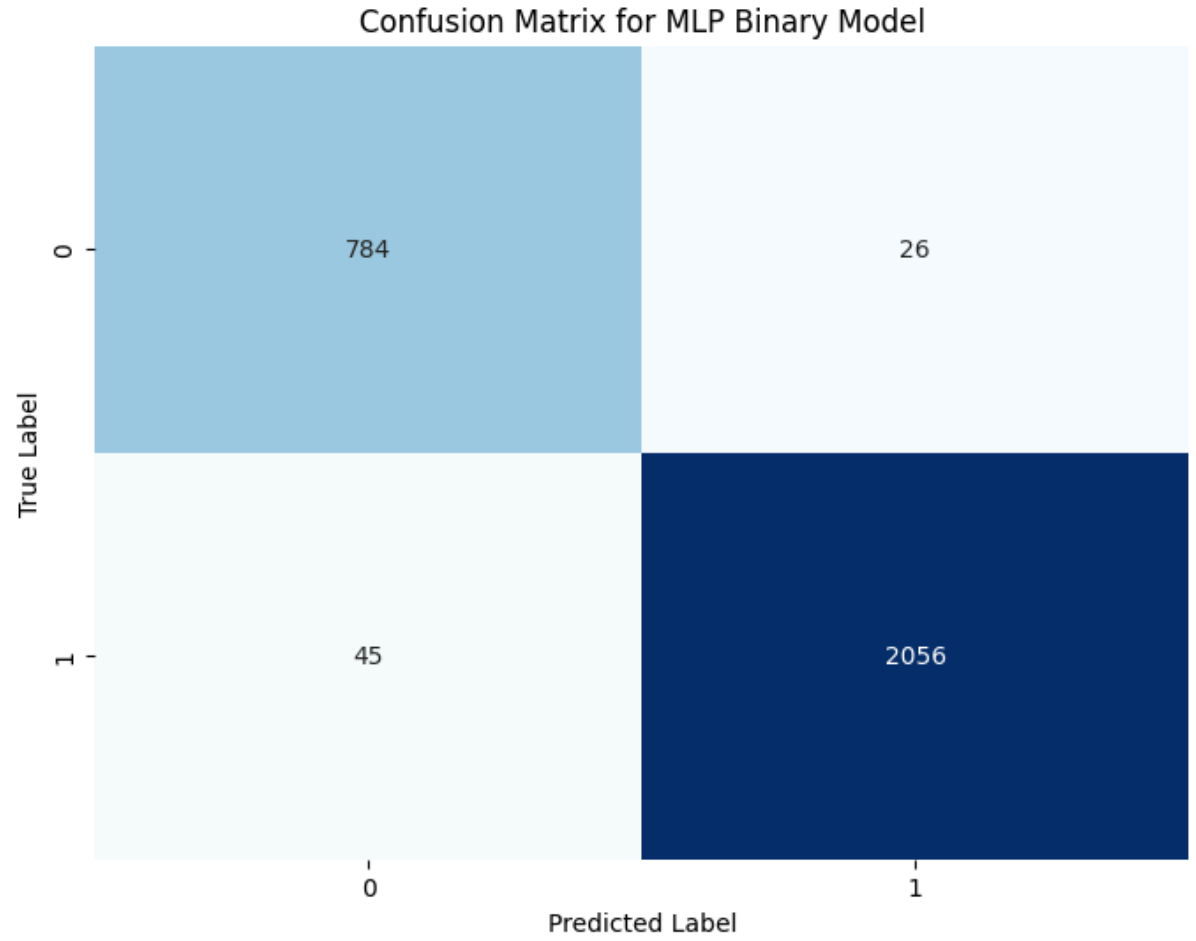
Rezultate:



Valori relativ asemănătoare pe partea de Train Loss, la partea de validation e greu de spus ce configuratie este cea mai buna.

	layers	units	loss	accuracy	precision_class_0.0	recall_class_0.0	f1_class_0.0	precision_class_1.0	recall_class_1.0	f1_class_1.0
0	2	64	0.099340	0.969770	0.946782	0.944444	0.945612	0.978602	0.979534	0.979068
1	2	128	0.122943	0.958434	0.923739	0.927160	0.925447	0.971878	0.970490	0.971184
2	2	256	0.121373	0.969426	0.922626	0.971605	0.946482	0.988824	0.968586	0.978601
3	3	64	0.122236	0.967709	0.947500	0.935802	0.941615	0.975367	0.980010	0.977683
4	3	128	0.133894	0.963586	0.976965	0.890123	0.931525	0.959043	0.991909	0.975199
5	3	256	0.102980	0.974923	0.962359	0.946914	0.954574	0.979659	0.985721	0.982681
6	4	64	0.114866	0.971144	0.935252	0.962963	0.948905	0.985556	0.974298	0.979895
7	4	128	0.125339	0.959808	0.960159	0.892593	0.925144	0.959685	0.985721	0.972529
8	4	256	0.110848	0.973205	0.937799	0.967901	0.952612	0.987470	0.975250	0.981322

Cele mai bune rezultate pentru 3 layers cu 256 units, am realizat matricea de confuzie pentru acea configuratie



b. Convolutional

i. PTB Diagnostic ECG

```
def create_conv_model(kernel_size):
   model = Sequential()
   model.add(Conv1D(32, kernel_size, activation='relu',
input_shape=(x_train_ptb.shape[1], 1)))
   model.add(BatchNormalization())
   model.add(Conv1D(64, kernel_size, activation='relu',
kernel_regularizer=l2(0.001)))
```

```
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Conv1D(128, kernel_size, activation='relu',
kernel_regularizer=l2(0.001)))
    model.add(BatchNormalization())
    model.add(GlobalAveragePooling1D())
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer="adam", loss='binary_crossentropy',
metrics=['accuracy'])
    return model
```

Model:

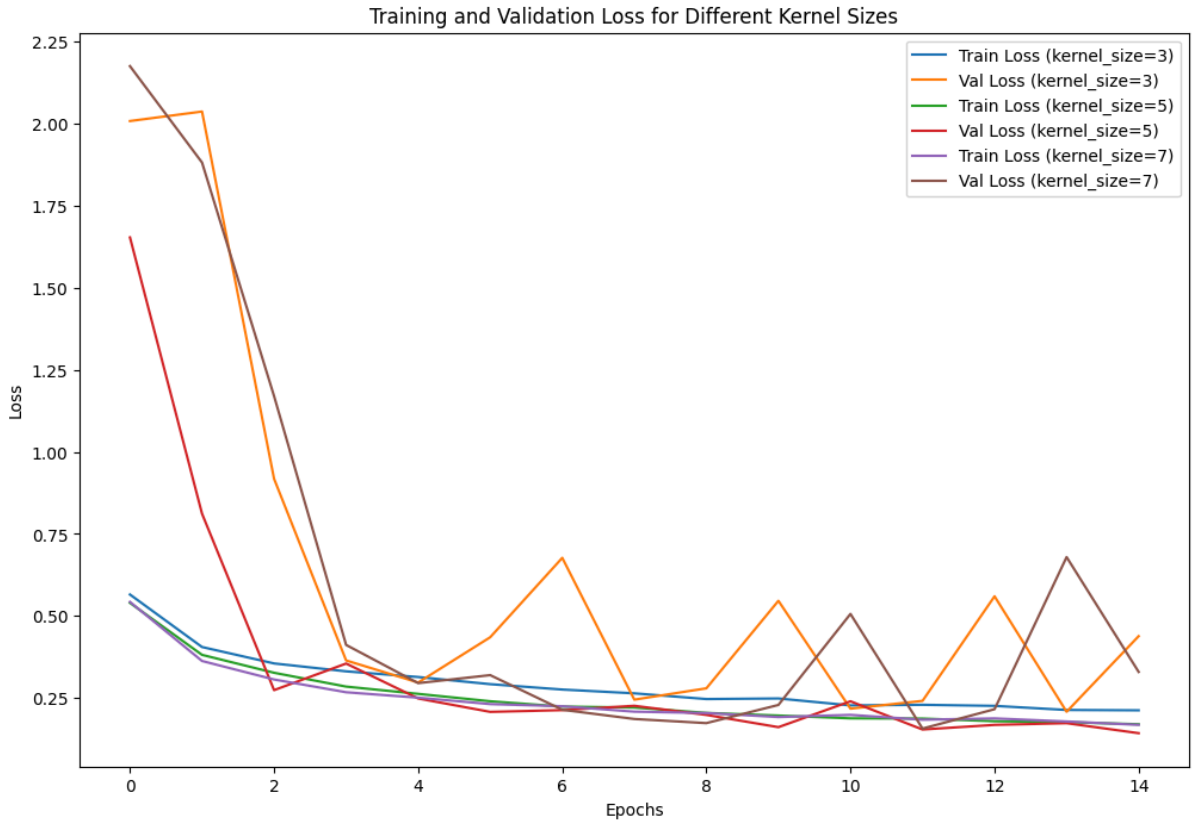
4. Input Layer:
    - Conv1D(32, kernel_size, activation='relu', input_shape=(x_train_ptb.shape[1], 1)): Un strat convoluțional 1D cu 32 de filtre și dimensiunea nucleului specificată prin kernel_size. Funcția de activare este ReLU, iar forma de intrare este determinată de dimensiunea datelor de antrenament x_train_ptb.
5. Hidden Layers:
    - BatchNormalization(): Normalizare batch pentru stabilizarea și accelerarea procesului de antrenare.
    - Conv1D(64, kernel_size, activation='relu', kernel_regularizer=l2(0.001)): Un al doilea strat convoluțional 1D cu 64 de filtre, activare ReLU și regularizare L2 pentru a preveni overfitting-ul.
    - BatchNormalization(): Normalizare batch.
    - Dropout(0.5): Dropout cu rata de 0.5 pentru regularizare și reducerea overfitting-ului.
    - Conv1D(128, kernel_size, activation='relu', kernel_regularizer=l2(0.001)): Un al treilea strat convoluțional 1D cu 128 de filtre, activare ReLU și regularizare L2.
    - BatchNormalization(): Normalizare batch.
    - GlobalAveragePooling1D(): Pooling global pentru reducerea dimensionalității și extragerea caracteristicilor globale.
6. Output Layers:
    - Dense(64, activation='relu'): Un strat dens cu 64 de neuroni și activare ReLU.
    - Dropout(0.5): Dropout cu rata de 0.5.

- Dense(1, activation='sigmoid'): Un strat dens cu un singur neuron și activare sigmoid pentru a produce o probabilitate binară (folosit pentru clasificarea binară).

Parametrii:

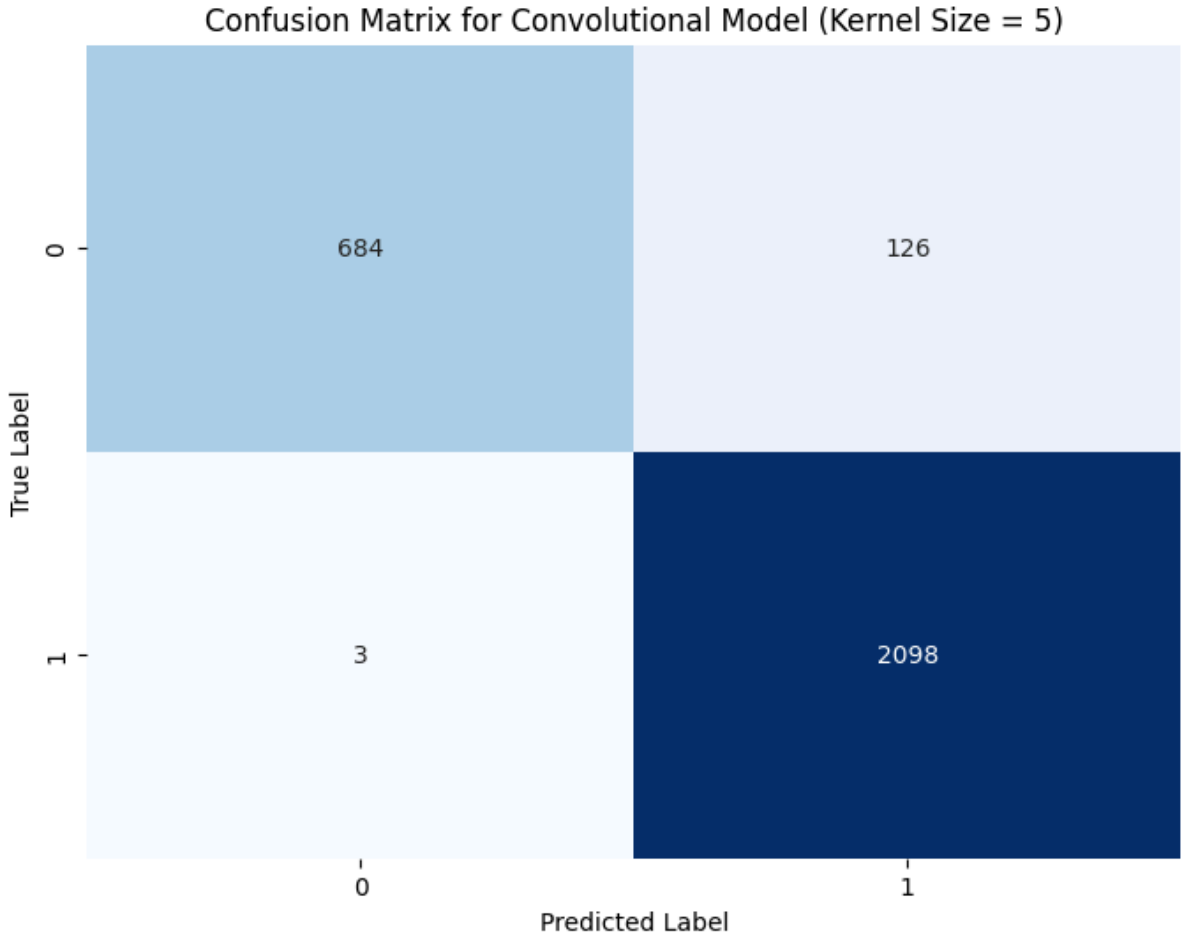
- Optimizator: adam
- Batch size: 32
- Epochs: 25
- loss: binary_crossentropy
- Learning Rate: 0.001
- Activation: relu
- Weight decay: kernel_regularizer=l2(0.001)

Rezultate:



	kernel_size	loss	accuracy	precision_class_0.0	recall_class_0.0	f1_class_0.0	precision_class_1.0	recall_class_1.0	f1_class_1.0
0	3	0.173081	0.949845	0.905868	0.914815	0.910319	0.967033	0.963351	0.965188
1	5	0.186505	0.956029	0.925187	0.916049	0.920596	0.967757	0.971442	0.969596
2	7	0.219615	0.917554	0.839286	0.870370	0.854545	0.949300	0.935745	0.942474

Se observa ca cel mai bun rezultat este pentru kernel_size = 5 pentru toate metricile


Confusion Matrix for Convolutional Model (Kernel Size = 5)
684
126
3
2098
0
1
True Label
0
1
Predicted Label