Универзитет у Нишу

Електронски факултет Ниш

Дубоко учење

Детекција аномалија на *ECG5000* скупу података

Студент:

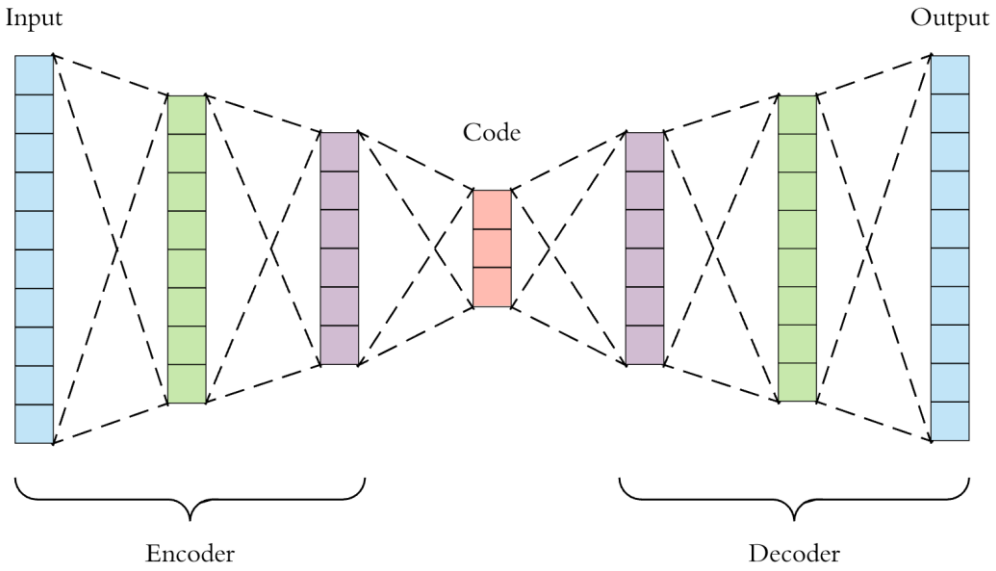
Димитрије Јовић, 928

Ниш, август 2020. год.

С А Д Р Ж А Ј

1. УВОД

Сведоци смо последњих године све веће примене вештачких неуронских мрежа у рачунарству. Вештачка неуронска мрежа представља систем састављен од одређеног броја међусобно повезаних јединица, који се називају вештачким неуронима. Данас постоји велики број комплексних вештачких неуронских мрежа чија се примена прилагођава различитим проблемима. Једна врста проблема којој се прилагођавају вештачке неуронске мреже је и детекција аномалија, односно откривање података који значајно одступају од осталих. Најчешће примењивана врста вештачких неуронских мрежа су аутоенкодери. Аутоенкодери имају за циљ да што прецизније реконструишу улазне податке. Аутоенкодери се састоје из два дела: енкодера и декодера. Енкодери служе да број атрибута улазних података смањи, односно врши редукцију димензионалности улазних података. Декодер покушава да реконструише излаз на основу излаза из енкодера (редукован број атрибута улазних података).



Слика 1: Архитектура аутоенкодера

Циљ овог пројекта је проучавање различитих архитектура аутоенкодерских мрежа и поређење њихових резултата са већ постојећим аутоенкодерским мрежама.

2. СКУП ПОДАТАКА

Скуп података који ће се користи је модификација скупа података *BIDMC Congestive Heart Failure Database(chfdb)* који се налази у *Physionet*-овој бази података. Оригинални подаци представљају 20-часовни ЕКГ. Подаци су модификовани у два корака: у првом кораку је извршено издвајање сваког откуцаја срца, у другом кораку је извршена интерполација откуцаја срца како би се сваки откуцај имао исту дужину. Након тога је случајним узорком изабрано 5000 откуцаја срца. Треба напоменути да овај скуп података садржи информације о класама и да је скуп података подељен на тренинг и тест скуп. Постоје укупно пет класа од којих једна представља нормалан рад срца, док остале представљају рад срца проузрокован неким срчаним обољењем.

3. АПЛИКАЦИЈА

У оквиру апликације коришћене су следеће библиотеке:

- *pandas* – библиотека за манипулацију и анализу података, нуди структуре за операције за рад табелама и временским серијама
- *matplotlib* – библиотека која омогућава визуелизацију података
- *seaborn* – библиотека за визуелизацију заснована на *matplotlib*-у
- *arff2pandas* – библиотека за учитавање *arff* фајлова
- *numpy* – библиотека која омогућава рад и математичке операције над вишедимензионалним низовима и матрицама
- *tensorflow* – библиотека која омогућава рад са неуронским мрежама

3.1. Компоненте апликације

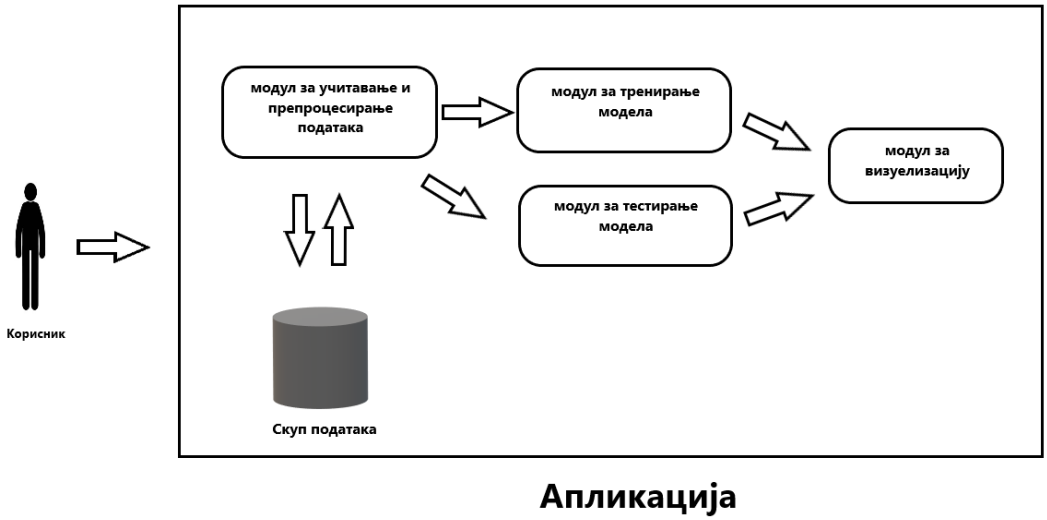
Апликација је имплементирана коришћењем програмског језика *Python* и *Tensorflow* библиотеке. Идеја је да се у скупу података детектују аномалије и да се одреди која конфигурација аутоенкодера даје најбоље резултате, и да се ти резултати пореде са постојећим решењима.

Сама апликација је састављена из неколико главних модула: модул за тренирање, модул за тестирање и модул за визуелизацију. Такође постоје и помоћни модули који врше учитавање и препроцесирање података.

Модул за тренирање служи за обучавање различитих конфигурација аутоенкодера. Саме конфигурације ће бити приказане касније. Након учитавања и препроцесирања података, модул врши обучавање свих конфигурација аутоенкодера над скупом за обучавање. Обучавање се врши на начин да се увек памти најбоље стање, уколико не долази до побољшања стања након, одређеног броја епоха, врши се смањење коефицијента учења да би се пронашло боље стања, уколико оно постоје. Након завршеног обучавања, најбоље пронађено стање модела се чува на диску.

Модул за тестирање служи за тестирање обучених конфигурација аутоенкодера над скупом података за тестирање. Након учитавања и препроцесирања података модул врши учитавање обучених конфигурација које су сачуване на диску. Након завршетка тестирања врши се визуелизација статистичких података као и чување метрика.

Модул за визуелизацију служи за графичко представљање статистичких података скупа података као и за графичко представљање како статистичких података самих модела тако и места где је детектована аномалија као и изглед реконструисаних података.

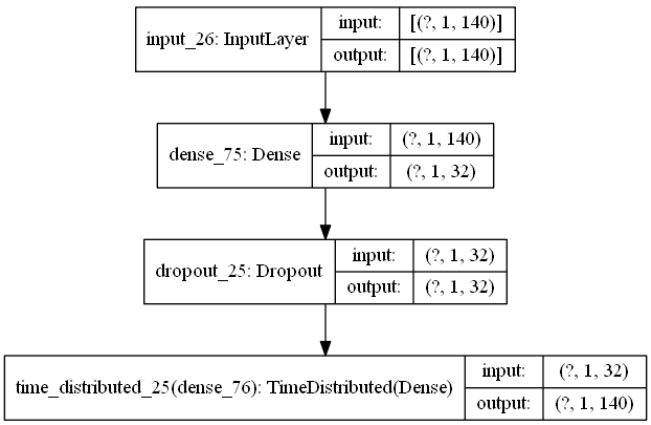


Слика 2: Компоненте апликације

3.2. Модели аутоенкодера

У оквиру саме апликације креирано је неколико модела како би се упоредиле перформансе истих. На сликама испод биће приказан графички приказ модела као и *Python* код за њихово креирање.

Први модел који је креиран је крајње једноставан, састоји се од једног потпуно повезаног слоја и једног *dropout* слоја.

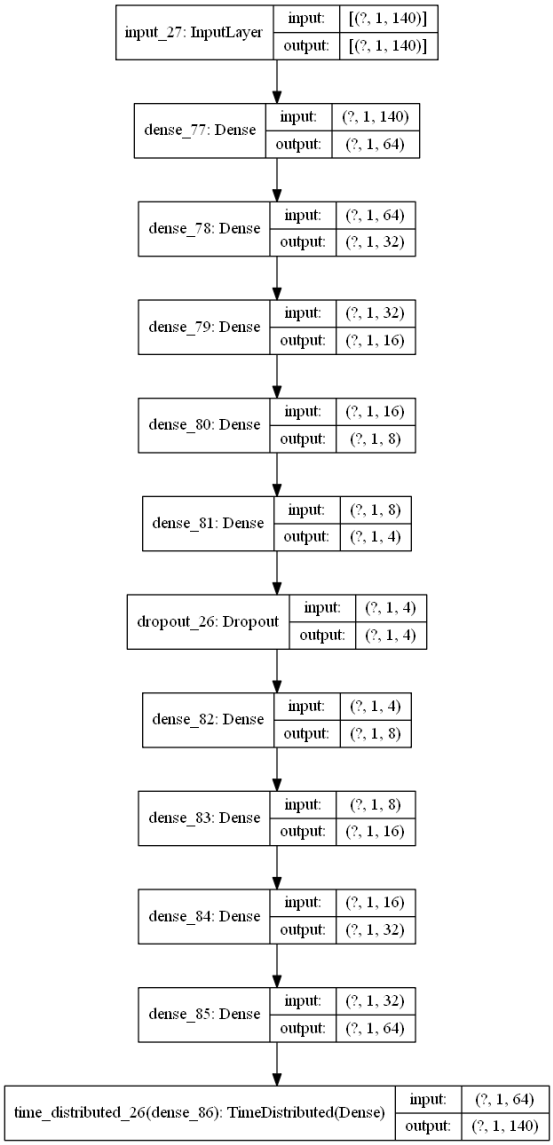


Слика 3: Најједноставнији модел аутоенкодера

```
def simple_autoencoder_model(data, hidden=128, dropout_rate=0.1):
    inputs = tf.keras.Input(shape=(data.shape[1], data.shape[2]))
    L1 = tf.keras.layers.Dense(int(hidden / 4), activation='relu')(inputs)
    L2 = tf.keras.layers.Dropout(dropout_rate)(L1)
    outputs = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(data.shape[2]))(L2)
    model = tf.keras.Model(inputs=inputs, outputs=outputs)
    return model
```

Слика 4: *Python* код за креирање најједноставнијег модела

Други модел који је креиран састоји се само из потпуно повезаних слојева и једног *dropout* слоја.

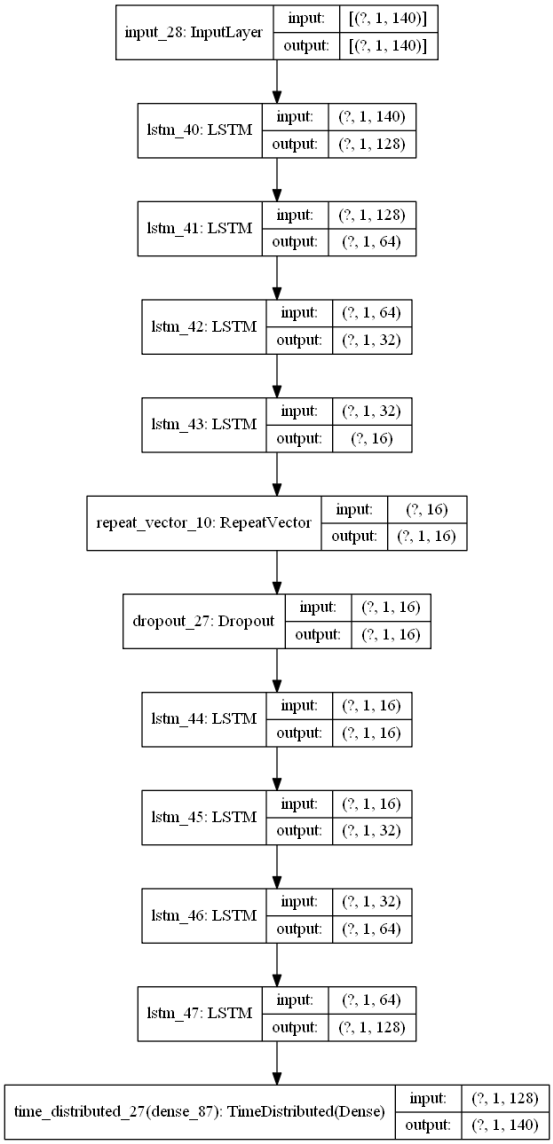


Слика 5: Модел са свим потпуно повезаним слојевима

```
def deep_autoencoder_model(data, hidden=128, dropout_rate=0.1):
    inputs = tf.keras.Input(shape=(data.shape[1], data.shape[2]))
    L1 = tf.keras.layers.Dense(int(hidden / 2), activation='relu')(inputs)
    L2 = tf.keras.layers.Dense(int(hidden / 4), activation='relu')(L1)
    L3 = tf.keras.layers.Dense(int(hidden / 8), activation='relu')(L2)
    L4 = tf.keras.layers.Dense(int(hidden / 16), activation='relu')(L3)
    L5 = tf.keras.layers.Dense(int(hidden / 32), activation='relu')(L4)
    L6 = tf.keras.layers.Dropout(dropout_rate)(L5)
    L7 = tf.keras.layers.Dense(int(hidden / 16), activation='relu')(L6)
    L8 = tf.keras.layers.Dense(int(hidden / 8), activation='relu')(L7)
    L9 = tf.keras.layers.Dense(int(hidden / 4), activation='relu')(L8)
    L10 = tf.keras.layers.Dense(int(hidden / 2), activation='relu')(L9)
    outputs = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(data.shape[2]))(L10)
    model = tf.keras.Model(inputs=inputs, outputs=outputs)
    return model
```

Слика 6: *Python* код за креирање модела са свим потпуно повезаним слојевима

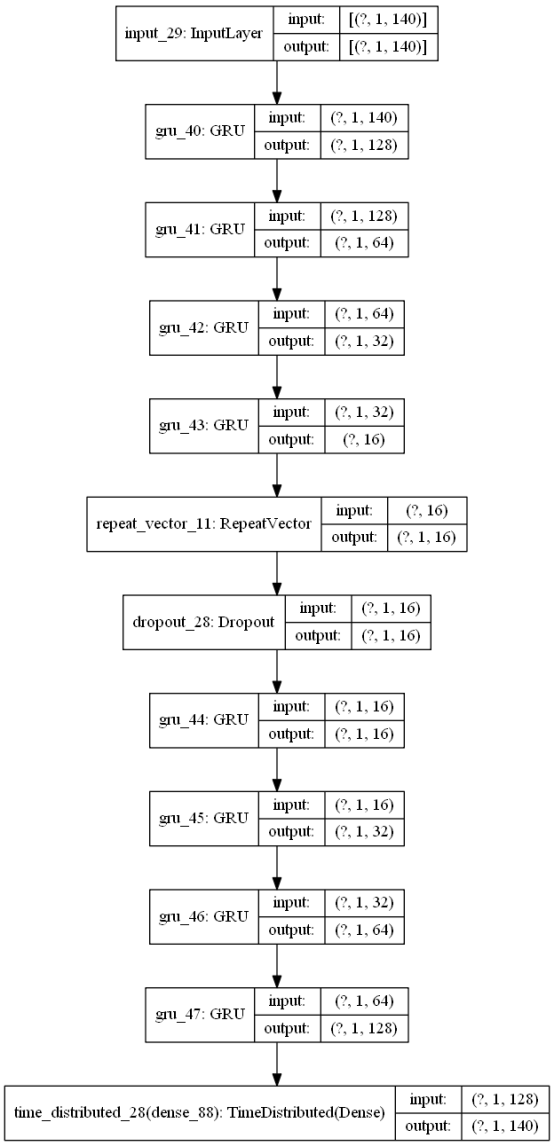
Трећи и четврти модел који су креиран су модели са *LSTM* и *GRU* неуронским ћелијама, односно рекурентним неуронским ћелијама, и једним *dropout* слојем.



Слика 7: *LSTM* модел

```
def LSTM_autoencoder_model(data, hidden=128, dropout_rate=0.1):
    inputs = tf.keras.Input(shape=(data.shape[1], data.shape[2]))
    L1 = tf.keras.layers.LSTM(hidden, return_sequences=True)(inputs)
    L2 = tf.keras.layers.LSTM(int(hidden / 2), activation='relu', return_sequences=True)(L1)
    L3 = tf.keras.layers.LSTM(int(hidden / 4), activation='relu', return_sequences=True)(L2)
    L4 = tf.keras.layers.LSTM(int(hidden / 8), activation='relu', return_sequences=False)(L3)
    L5 = tf.keras.layers.RepeatVector(data.shape[1])(L4)
    L6 = tf.keras.layers.Dropout(dropout_rate)(L5)
    L7 = tf.keras.layers.LSTM(int(hidden / 8), activation='relu', return_sequences=True)(L6)
    L8 = tf.keras.layers.LSTM(int(hidden / 4), activation='relu', return_sequences=True)(L7)
    L9 = tf.keras.layers.LSTM(int(hidden / 2), activation='relu', return_sequences=True)(L8)
    L10 = tf.keras.layers.LSTM(hidden, activation='relu', return_sequences=True)(L9)
    output = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(data.shape[2]))(L10)
    model = tf.keras.Model(inputs=inputs, outputs=output)
    return model
```

Слика 8: *Python* код за креирање *LSTM* модела

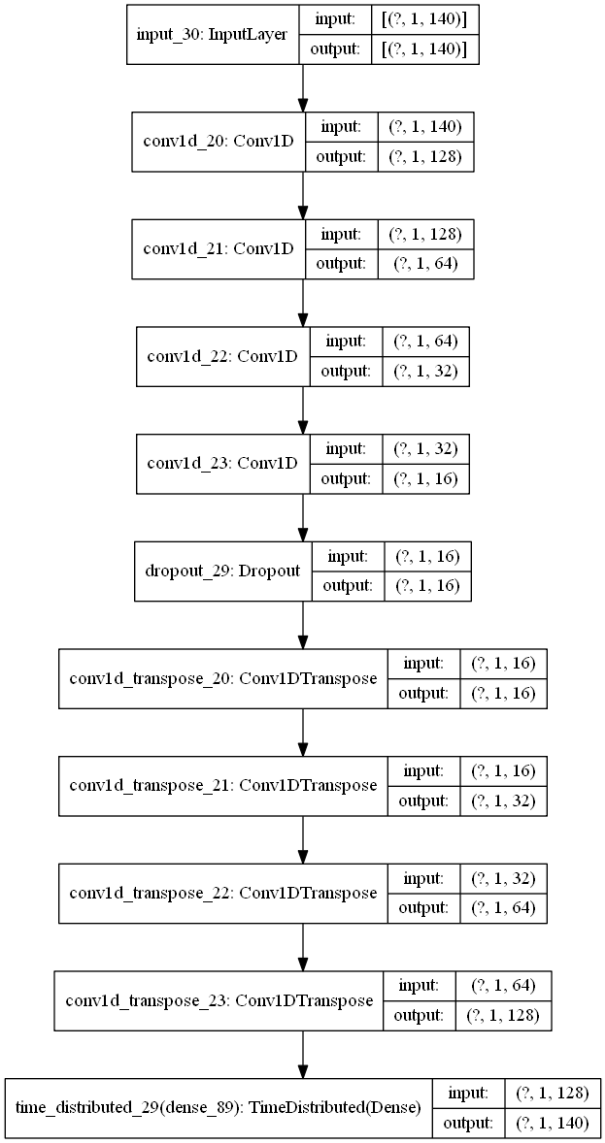


Слика 9: *GRU* модел

```
def GRU_autoencoder_model(data, hidden=128, dropout_rate=0.1):
    inputs = tf.keras.Input(shape=(data.shape[1], data.shape[2]))
    L1 = tf.keras.layers.GRU(hidden, activation='relu', return_sequences=True)(inputs)
    L2 = tf.keras.layers.GRU(int(hidden / 2), activation='relu', return_sequences=True)(L1)
    L3 = tf.keras.layers.GRU(int(hidden / 4), activation='relu', return_sequences=True)(L2)
    L4 = tf.keras.layers.GRU(int(hidden / 8), activation='relu', return_sequences=False)(L3)
    L5 = tf.keras.layers.RepeatVector(data.shape[1])(L4)
    L6 = tf.keras.layers.Dropout(dropout_rate)(L5)
    L7 = tf.keras.layers.GRU(int(hidden / 8), activation='relu', return_sequences=True)(L6)
    L8 = tf.keras.layers.GRU(int(hidden / 4), activation='relu', return_sequences=True)(L7)
    L9 = tf.keras.layers.GRU(int(hidden / 2), activation='relu', return_sequences=True)(L8)
    L10 = tf.keras.layers.GRU(hidden, activation='relu', return_sequences=True)(L9)
    output = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(data.shape[2]))(L10)
    model = tf.keras.Model(inputs=inputs, outputs=output)
    return model
```

Слика 10: *Python* код за креирање *GRU* модела

Пети и последњи модел је креиран коришћењем конволуционих слојева и једног *dropout* слоја.



Слика 11: Модел аутоенкодера са конволуционим слојевима

```python
def convolution_autoencoder_model(data, kernel_size=1, filters=128, dropout_rate=0.1):
    inputs = tf.keras.Input(shape=(data.shape[1], data.shape[2]))
    L1 = tf.keras.layers.Conv1D(filters=filters, kernel_size=kernel_size)(inputs)
    L2 = tf.keras.layers.Conv1D(filters=int(filters / 2), kernel_size=kernel_size)(L1)
    L3 = tf.keras.layers.Conv1D(filters=int(filters / 4), kernel_size=kernel_size)(L2)
    L4 = tf.keras.layers.Conv1D(filters=int(filters / 8), kernel_size=kernel_size)(L3)
    L5 = tf.keras.layers.Dropout(dropout_rate)(L4)
    L6 = tf.keras.layers.Conv1DTranspose(filters=int(filters / 8), kernel_size=kernel_size)(L5)
    L7 = tf.keras.layers.Conv1DTranspose(filters=int(filters / 4), kernel_size=kernel_size)(L6)
    L8 = tf.keras.layers.Conv1DTranspose(filters=int(filters / 2), kernel_size=kernel_size)(L7)
    L9 = tf.keras.layers.Conv1DTranspose(filters=filters, kernel_size=kernel_size)(L8)
    output = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(data.shape[2]))(L9)
    model = tf.keras.Model(inputs=inputs, outputs=output)
    return model
```

Слика 12: *Python* код за креирање модела са конволуционим слојевима

3.3. Модул за учитавање и препроцесирање података

Као што је раније речено, скуп података је подељен на два дела, скуп за тренирање и скуп за тестирање. Оба овај фајла сачувана су у *.arff* формату (формат који се користи у *Weka* окружењу). С тога је потребно учитати податке и пребацити их у структуру погодну за рад. Подаци се чувају у оквиру *pandas* оквира, а библиотека која омогућава директно претварање *.arff* фајла у *pandas* оквир је *arff2pandas*. Након учитавања фајлова, врши се њихово надовезивање у оквиру једног *pandas* оквира при чему се врши замена имена класног атрибута.

```
def load_data(file_name):
    with open(file_name) as f:
        data = a2p.load(f)
    return data


def concatenate_data(train_data, test_data, class_column='target'):
    data = train_data.append(test_data)
    data = data.sample(frac=1.0)
    data = rename_class_column(data, class_column)
    return data


def rename_class_column(data, name):
    new_columns = list(data.columns)
    new_columns[-1] = name
    data.columns = new_columns
    return data
```

Слика 13: Функције за учитавање, конкатенацију и замену имена класног атрибута

Како ћемо за тренирање модела користити само податке које припадају класи која означава нормалан рад срца, а приликом учитавања се сви подаци налазе у оквиру истог оквира, потребно је одвојити податке које представљају нормалан рад срца од оних које то не представљају. Такође, с обзиром да рекурентне неуронске ћелије примају податке у формату (број узорака, временска ознака, број атрибута), потребно је и променити облик подацима.

```
def split_data(data):
    normal_data = data[data.target == str(CLASS_NORMAL)].drop(labels='target', axis=1)
    anomaly_data = data[data.target != str(CLASS_NORMAL)].drop(labels='target', axis=1)

    normal_data_np = normal_data.values.reshape(normal_data.shape[0], 1, normal_data.shape[1])
    anomaly_data_np = anomaly_data.values.reshape(anomaly_data.shape[0], 1, anomaly_data.shape[1])

    return normal_data_np, anomaly_data_np, normal_data, anomaly_data
```

Слика 14: Функција која дели податке и врши промену облика подацима

3.4. Модул за тренирање модела

За тренирање самог модела користиће се подаци који представљају нормалан рад срца. Као што је речено, приликом тренирања модела, чува се увек најбољи конфигурација, смањује се коефицијент учења уколико се након одређеног броја епоха не види напредак, такође је и омогућено заустављање тренирања уколико се утврди да се даљим тренирањем не постиже никакав напредак. Што се подешавања ових техника, као величина која се прати изабран је валидациони губитак, односно губитак над валидационим скупом података, гледа се да је он што мањи, број епоха након којих ће се смањити коефицијент учења је 10, а коефицијент смањења је 0.1, док ће се заустављање извршити уколико се напредак не уочи у 30 епоха. Приликом обучавања

укупан број епоха је 1000, док је величина *batch*-а 64. Валидациони скуп представља ¼ улазних података, док је почетна вредност укупног броја скривених слојева по нивоу 128. Такође, величина кернела код конволуционог модела је 1, док је вероватноћа одбацивања скривеног слоја код свих модела 10%.

```
def training(model, train_data, optimizer='adam', loss='mae', model_name='simple_autoencoder'):
    if loss == 'mse' and optimizer == 'adagrad':
        optimizers = tf.keras.optimizers.Adagrad(s.SGD_ADAGRAD_LEARNING_RATE_MSE)
    elif loss == 'mse' and optimizer == 'sgd':
        optimizers = tf.keras.optimizers.SGD(s.SGD_ADAGRAD_LEARNING_RATE_MSE)
    elif loss == 'mae' and optimizer == 'adagrad':
        optimizers = tf.keras.optimizers.Adagrad(s.SGD_ADAGRAD_LEARNING_RATE_MAE)
    elif loss == 'mae' and optimizer == 'sgd':
        optimizers = tf.keras.optimizers.SGD(s.SGD_ADAGRAD_LEARNING_RATE_MAE)
    else:
        optimizers = optimizer

    model.compile(optimizer=optimizers, loss=loss)

    path = 'models/' + model_name + '/' + loss + '/' + optimizer + '/'
    if not os.path.isdir(path):
        os.makedirs(path)

    save_model = tf.keras.callbacks.ModelCheckpoint(path + 'model.h5', monitor='val_loss',
                                                    mode='min', verbose=s.VERBOSE, save_best_only=True)
    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', patience=s.PATIENCE
                                                      restore_best_weights=True, verbose=s.VERBOSE)
    reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', mode='min', factor=s.REDUCE_FACT
                                                     patience=int(s.PATIENCE / 3), verbose=s.VERBOSE)
    history = model.fit(train_data, train_data,
                        validation_split=s.VALIDATION_SPLIT,
                        epochs=s.EPOCHS,
                        batch_size=s.BATCH_SIZE,
                        callbacks=[save_model,
                                   early_stopping,
                                   reduce_lr
                                   ],
                        verbose=s.VERBOSE + 1,
                        )
    visualize_training_losses(history.history, path, loss)
```

Слика 15: Функција за тренирање модела

3.5. Модул за тестирање модела

Након завршеног тренирања, сви истренирани модели су сачувани и налазе се на диску. С тога је први корак приликом тестирања модела учитавање истих. Након учитавања, врши се предикција над улазним скупом. Треба напоменути да се предикција врши и над подацима који представљају нормалан рад срца, како би се увиделе перформансе система над целим скупом података.

```
def show_results(model, data_np, data, model_name, optimizer, loss, data_type='normal', loss_visualization=True,
                 prediction_visualization=False, anomaly_visualization=False):
    prediction = predict(model, data_np, data)
    threshold = THRESHOLDS[loss][model_name]

    path = 'models/' + model_name + '/' + loss + '/' + optimizer + '/'
    make_directory(path)

    scored = visualize_loss_distribution(data, prediction, path, loss, data_type, loss_visualization)

    path = 'models/' + model_name + '/' + loss + '/' + optimizer + '/reconstructed/' + data_type + '/'
    make_directory(path)

    if prediction_visualization:
        visualize_predictions(data, prediction, scored, path, loss)
    if anomaly_visualization:
        visualize_anomaly(data.columns, data.values, prediction.values, threshold[optimizer], model_name, optimizer,
                          data_type, loss)

    i = count_correct_detection(scored, data_type, threshold[optimizer], loss)

    return i, len(scored[loss]) - i


def predict(model, data, train_data):
    prediction = model.predict(data)

    prediction = prediction.reshape(prediction.shape[0], prediction.shape[2])
    prediction = pd.DataFrame(prediction, columns=train_data.columns)
    prediction.index = train_data.index

    return prediction
```

Слика 16: Функције за тестирање модела

3.6. Модул за визуелизацију

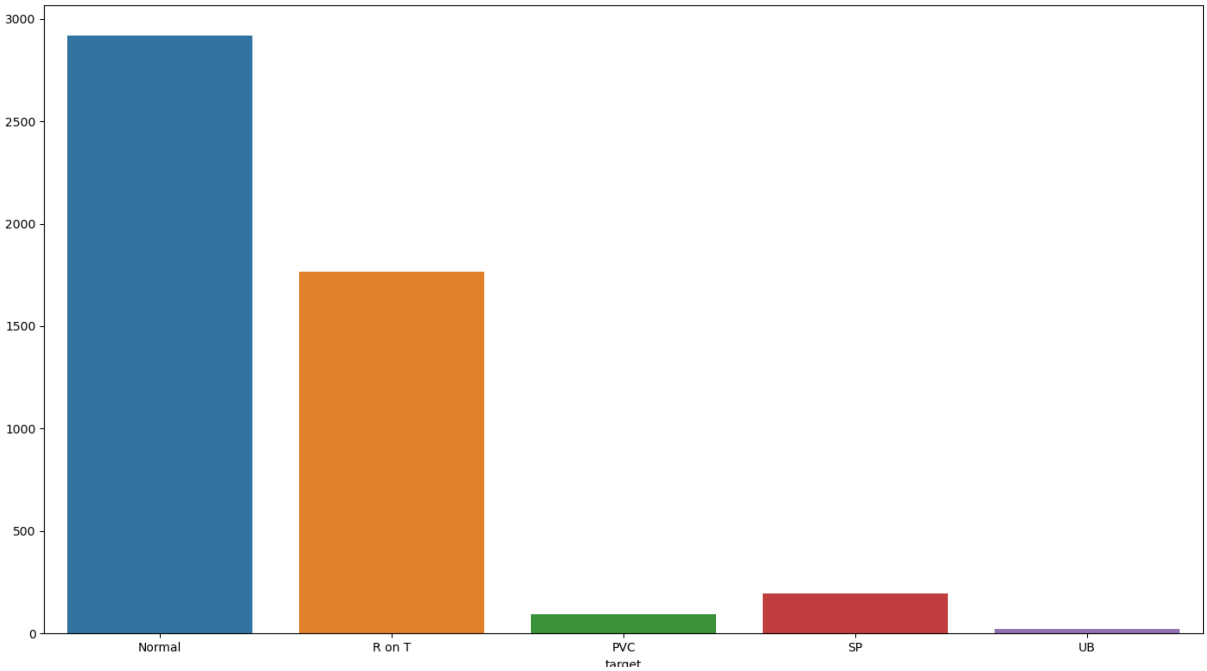
Модул за визуелизацију служи за приказивање статистичких података скупа података и за приказ статистичких података самих модела. Неке од функција које овај модул нуди су:

- Визуелизација дистрибуције класа – број инстанци које свака класа садржи
- Визуелизација шаблона класа – изглед рада срца по класама
- Визуелизација тренинг губитака – график губитака приликом обучавања
- Визуелизација дистрибуције губитака – дистрибуција губитака по класама у односу на истренирани модел
- Визуелизација предвиђања – изглед рада срца који предвиђа истренирани модел
- Визуелизација аномалија – обележава место где се десила аномалија у односу на истренирани модел

4. РЕЗУЛТАТИ

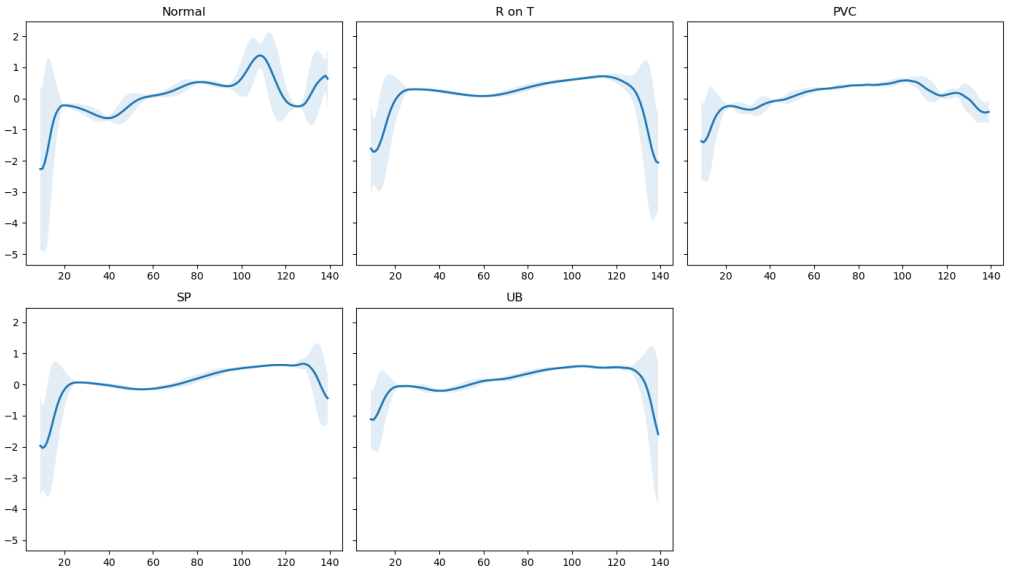
Пре експериментисања, одлучено је да ће се за тестирање модела користи две функције губитака: средња апсолутна грешка и средње квадратна грешка. Такође је одлучено и који ће се оптимизатори користи: *adam*, *nadam*, *adagrad*, *rmsprop* и *sgd*. Након чега се кренуло са експериментисањем. Експериментално је утврђено да модели дају добре резултате за уобичајне вредности коефицијента учења осим модели који као оптимизаторе користе *adagrad* оптимизатор и *sgd* оптимизатор. Стога се код ових модела подешава коефицијент учења који се разликује од осталих модела. Код средње апсолутне грешке коефицијент је 0.1, док је код средње квадратне 0.01.

Пре почетка тренирања модела извршена је статистичка визуелизација параметара како би се утврдио укупан број истанци по класама. Са слике испод се види да је расподела по класама приближно једнака јер четири класе које не карактеришу нормалан рад срце сврставамо у једну групу.



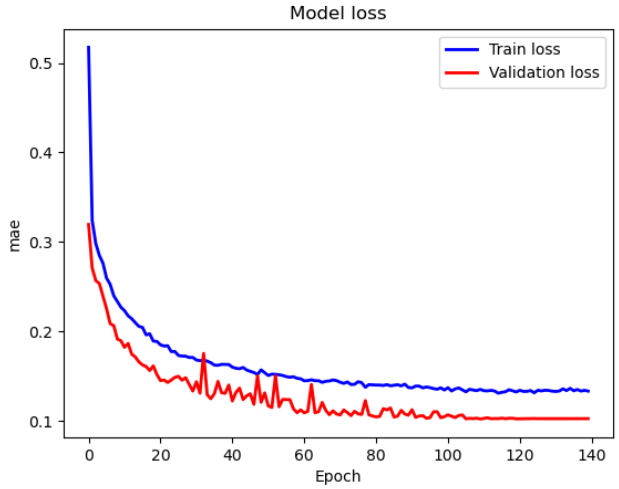
Слика 17: Дистрибуција класа

Такође је извршена и визуелизација шаблона класа, како би се увидело да ли је могућа детекција аномалија, јер ако класе имају сличан шаблон са класом која представља нормалан рад срца детекција би била веома тешка. Са слике се јасно види да постоји разлика у шаблонима нормалног рада срца са свим осталим класама.

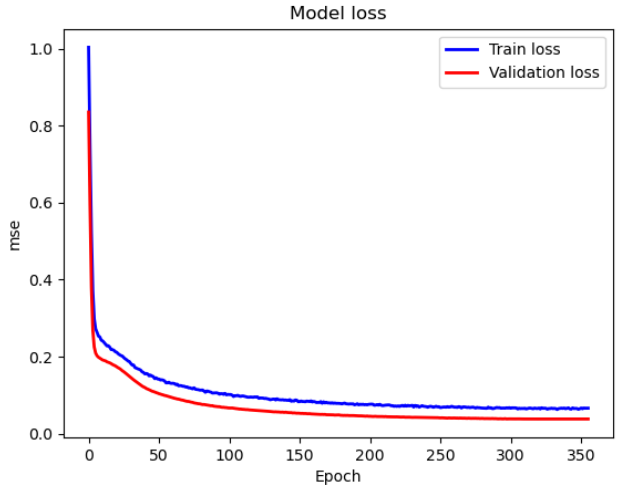


Слика 18: Шаблон класа скупа података

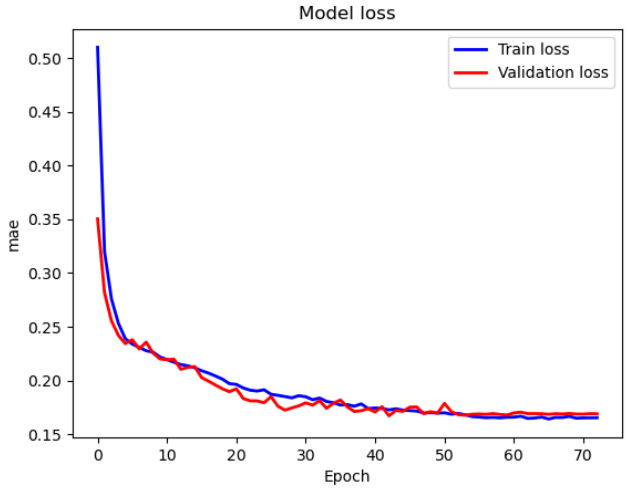
Након што смо утврдили да је сама дистрибуција класа добра и да се шаблони разликују, могуће је кренути са обучавањем модела. Приликом обучавања модела циљ је био што више смањити валидационе губитке, како би модел што боље реконструисао улазне податке. На сликама испод приказани су губици за неке конфигурације различитих модела.



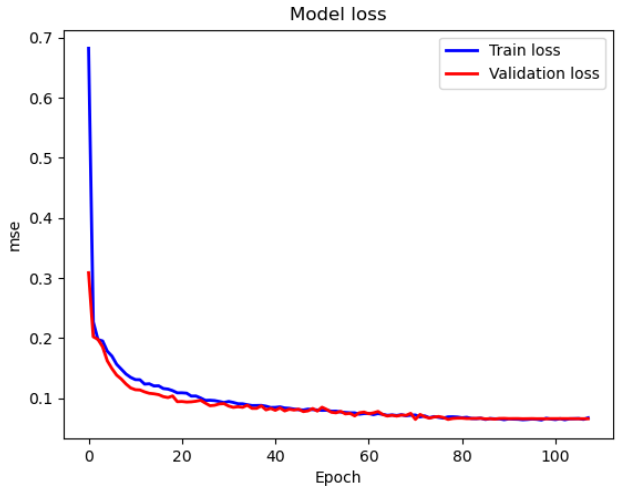
Слика 19: Губици најједноставнијег модела (*adagrad*, средње апсолутна грешка)



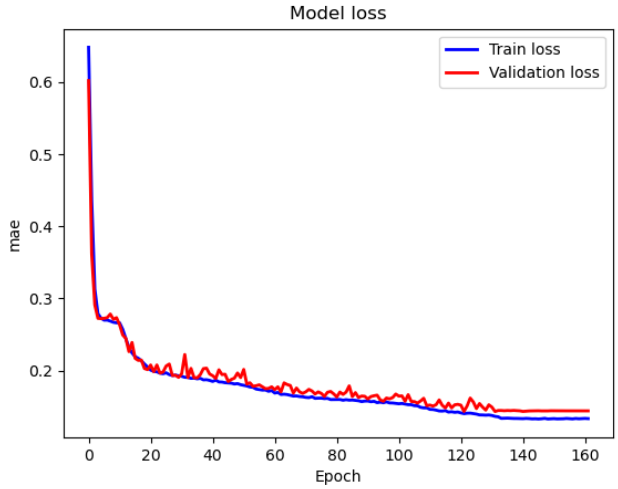
Слика 20: Губици најједноставнијег модела (*adagrad*, средње квадратна грешка)



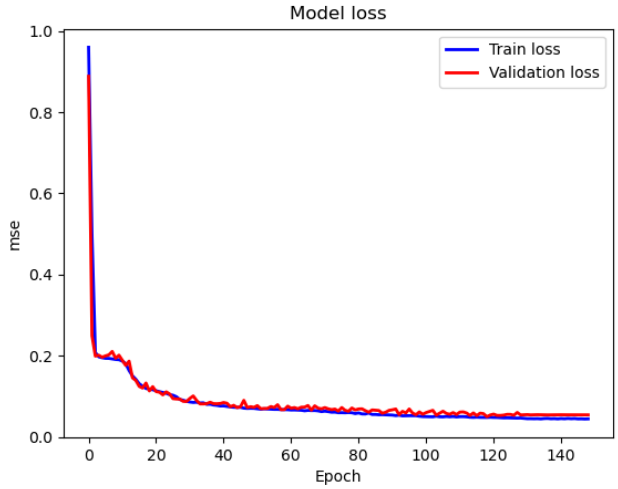
Слика 21: Губици модела са поптуно повезаним слојевима (*adam*, средње апсолутна грешка)



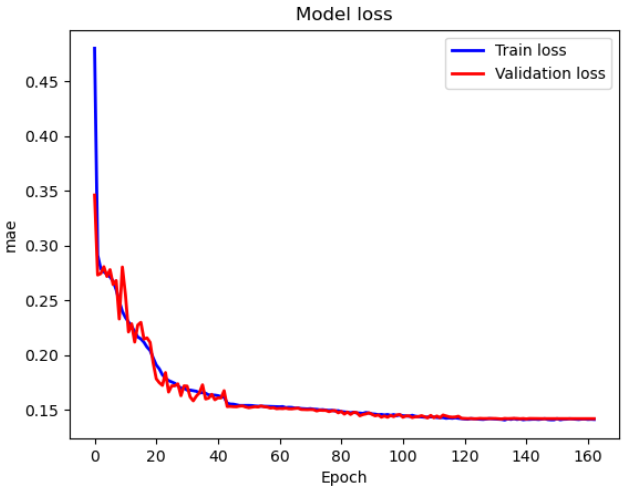
Слика 22: Губици модела са поптуно повезаним слојевима (*adam*, средње квадратна грешка)



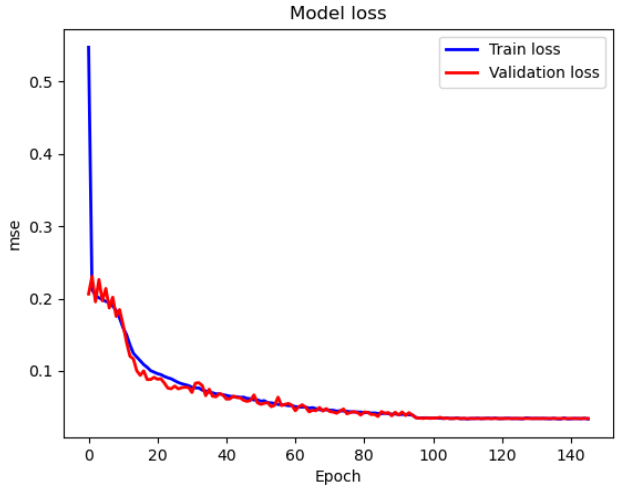
Слика 23: Губици *LSTM* модела (*nadam*, средње апсолутна грешка)



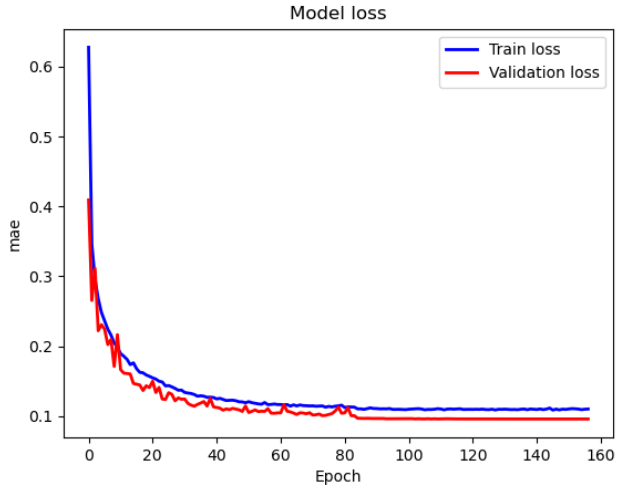
Слика 24: Губици *LSTM* модела (*nadam*, средње квадратна грешка)



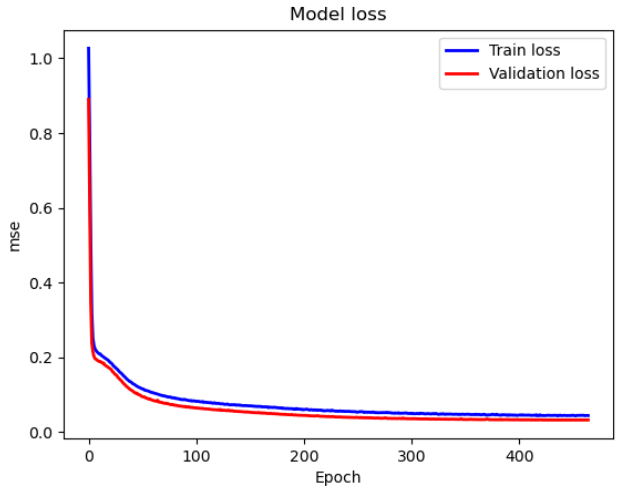
Слика 25: Губици *GRU* модела (*rmsprop*, средње апсолутна грешка)



Слика 26: Губици *GRU* модела (*rmsprop*, средње квадратна грешка)



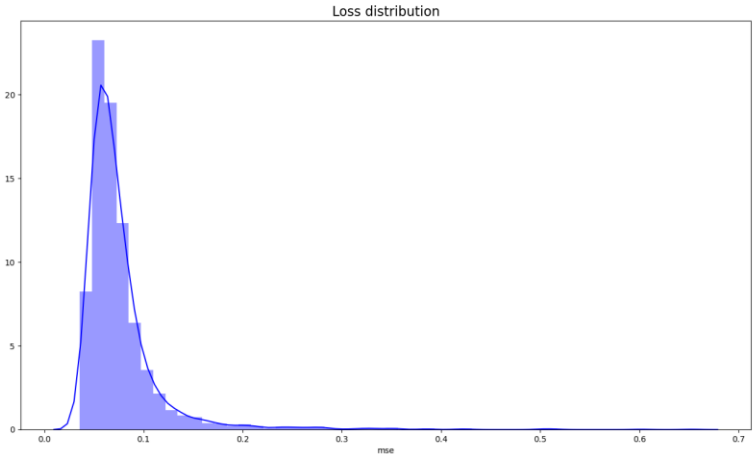
Слика 27: Губици конволуционог модела (*sgd*, средње апсолутна грешка)



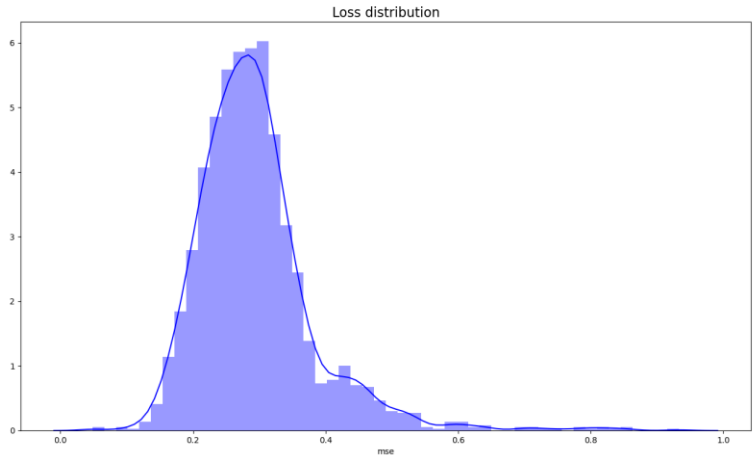
Слика 28: Губици конволуционог модела (*sgd*, средње квадратна грешка)

Након завршеног обучавања потребно је одредити праг на основу кога ће се одредити која је инстанца аномалија а која је нормална. Овај праг одређујемо на основу дистрибуције губитака. Дистрибуција губитака за неки од модела биће приказана испод. Праг се одређује тако да што већи број инстанци буде иза задатог прага уколико је реч о нормалном раду срца, док је за остале класе тај услов обрнут. У конкретном

случају, праг је одређен комбинацијом ова два услова. Такође, све вредности прагова биће приказани на слици испод.



Слика 29: Дистрибуција губитака најједноставнијег модела (нормалан рад срца, *adam*, средње квадратна грешка)



Слика 30: Дистрибуција губитака најједноставнијег модела (аномаличан рад срца, *adam*, средње квадратна грешка)

```
THRESHOLDS = {
    'mae': {'simple_autoencoder': {'adam': 0.17, 'nadam': 0.17, 'adagrad': 0.2, 'rmsprop': 0.17, 'sgd': 0.28},
            'deep_autoencoder': {'adam': 0.33, 'nadam': 0.33, 'adagrad': 0.37, 'rmsprop': 0.38, 'sgd': 0.48},
            'LSTM_autoencoder': {'adam': 0.33, 'nadam': 0.33, 'adagrad': 0.5, 'rmsprop': 0.33, 'sgd': 0.5},
            'GRU_autoencoder': {'adam': 0.3, 'nadam': 0.3, 'adagrad': 0.5, 'rmsprop': 0.3, 'sgd': 0.5},
            'convolution_autoencoder': {'adam': 0.13, 'nadam': 0.15, 'adagrad': 0.17, 'rmsprop': 0.16, 'sgd': 0.21},
            },
    'mse': {'simple_autoencoder': {'adam': 0.15, 'nadam': 0.17, 'adagrad': 0.25, 'rmsprop': 0.16, 'sgd': 0.28},
            'deep_autoencoder': {'adam': 0.35, 'nadam': 0.35, 'adagrad': 0.44, 'rmsprop': 0.35, 'sgd': 0.44},
            'LSTM_autoencoder': {'adam': 0.37, 'nadam': 0.3, 'adagrad': 0.5, 'rmsprop': 0.3, 'sgd': 0.5},
            'GRU_autoencoder': {'adam': 0.25, 'nadam': 0.3, 'adagrad': 0.55, 'rmsprop': 0.25, 'sgd': 0.5},
            'convolution_autoencoder': {'adam': 0.13, 'nadam': 0.15, 'adagrad': 0.2, 'rmsprop': 0.15, 'sgd': 0.21},
            }
```

Слика 31: Вредности прага детекције за сваку конфигурацију модела

Након што смо утврдили прагове детекције аномалија, могуће је утврдити перформансе модела. У табелама испод биће приказане перформансе сваког модела, са свим могућим конфигурацијама понаособ. За мерење перформанси користиће се следеће мере изведене из матрице конфузије: прецизност, одзив, тачност и *F1* мера. Наведене мере се рачунају по следећим формулама:

		Predicted class	
		positive	negative
Actual class	positive	TP	FN
	negative	FP	TN

Слика 32: Матрица конфузије

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

Перформансе модела дате су у табелама испод:

	тачност	прецизност	одзив	*F1*
adam	**0.9804**	0.993354	**0.972936**	**0.983039**
nadam	0.9786	0.995419	0.967797	0.981414
adagrad	0.9734	**0.996436**	0.957862	0.976769
rmsprop	0.975	0.995039	0.961973	0.978227
sgd	0.9744	0.99224	0.963686	0.977755

Табела 1: Перформансе најједноставнијег модела (средње апсолутна грешка)

	тачност	прецизност	одзив	*F1*
adam	0.972	0.993606	0.958205	0.975584
nadam	0.9742	**0.996088**	0.959575	0.977491
adagrad	0.9726	0.993612	0.959233	0.97612
rmsprop	0.9742	0.995032	0.960603	0.977514
sgd	**0.9746**	0.992938	**0.963344**	**0.977917**

Табела 2: Перформансе најједноставнијег модела (средње квадратна грешка)

	тачност	прецизност	одзив	*F1*
adam	0.9798	**0.994386**	0.97088	0.982493
nadam	0.9792	0.993686	0.970538	0.981976
adagrad	0.9676	0.991795	0.952381	0.971688
rmsprop	**0.9844**	0.992374	**0.980815**	**0.986561**
sgd	0.9312	0.959002	0.921548	0.939902

Табела 3: Перформансе потпуно повезаног модела (средње апсолутна грешка)

	тачност	прецизност	одзив	*F1*
adam	0.977	0.981456	0.979102	0.980278
nadam	**0.9856**	**0.99513**	0.98013	**0.987573**
adagrad	0.9326	0.982797	0.900308	0.939746
rmsprop	0.9828	0.988617	**0.981843**	0.985218
sgd	0.9304	0.978415	0.900651	0.937924

Табела 4: Перформансе потпуно повезаног модела (средње квадратна грешка)

	тачност	прецизност	одзив	*F1*
adam	**0.9842**	0.99101	**0.981843**	**0.986405**
nadam	0.9822	0.991319	0.978075	0.984653
adagrad	0.945	0.980727	0.923947	0.951491
rmsprop	0.9688	**0.993217**	0.953066	0.972727
sgd	0.9448	0.98072	0.923604	0.951306

Табела 5: Перформансе *LSTM* модела (средње апсолутна грешка)

	тачност	прецизност	одзив	*F1*
adam	0.9828	0.985602	**0.984926**	0.985264
nadam	**0.983**	0.994763	0.976019	**0.985302**
adagrad	0.9424	0.970662	0.929428	0.949597
rmsprop	0.9816	**0.995443**	0.972936	0.984061
sgd	0.9404	0.966204	0.930456	0.947993

Табела 6: Перформансе *LSTM* модела (средње квадратна грешка)

	тачност	прецизност	одзив	*F1*
adam	**0.9822**	0.987259	0.982186	**0.984716**
nadam	0.979	0.98152	**0.982528**	0.982024
adagrad	0.9448	0.98072	0.923604	0.951306
rmsprop	0.98	**0.993005**	0.972593	0.982693
sgd	0.9446	0.980713	0.923261	0.951121

Табела 7: Перформансе *GRU* модела (средње апсолутна грешка)

	тачност	прецизност	одзив	*F1*
adam	0.9724	0.983316	0.969168	0.97619
nadam	**0.9818**	0.99234	**0.976362**	**0.984286**
adagrad	0.929	0.936946	0.941761	0.939347
rmsprop	0.9814	**0.995442**	0.972593	0.983885
sgd	0.9424	0.970662	0.929428	0.949597

Табела 8: Перформансе *GRU* модела (средње квадратна грешка)

	тачност	прецизност	одзив	*F1*
adam	0.9658	0.99212	0.948955	0.970058
nadam	0.9602	0.973538	0.957862	0.965636
adagrad	0.966	**0.993891**	0.947585	0.970186
rmsprop	0.967	0.991435	0.951696	0.971159
sgd	**0.9692**	0.987999	**0.95889**	**0.973227**

Табела 9: Перформансе конволуционог модела (средње апсолутна грешка)

	тачност	прецизност	одзив	*F1*
adam	0.9684	0.990757	0.954779	0.972435
nadam	0.9706	0.992888	0.956492	0.97435
adagrad	0.9702	**0.993585**	0.955122	0.973974
rmsprop	**0.9708**	0.991841	**0.957862**	**0.974556**
sgd	0.9622	0.993136	0.941761	0.966766

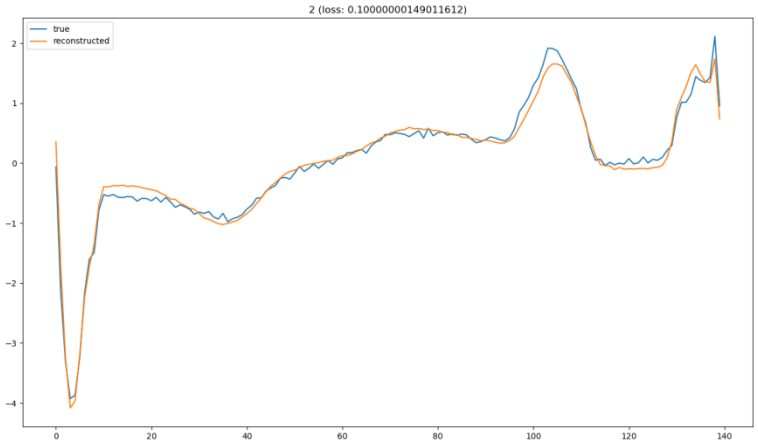
Табела 10: Перформансе конволуционог модела (средње квадратна грешка)

Мера која ће се користи за поређење перформанси модела је *F1* мера јер она комбинује све остале метрике заједно, по формули која је дата изнад. Из табела се види да сви модели дају задовољавајуће резултате и да разлика између модела огледа у

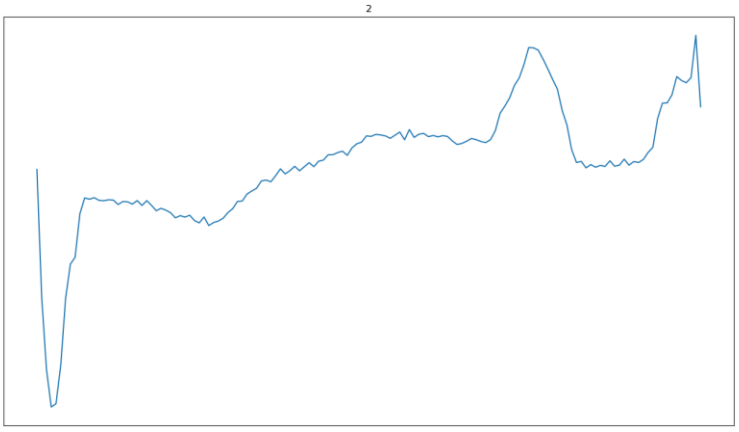
неколико процената, код неких и испод једног процента. Интересантно је приметити да *sgd* і *adagrad* дају нешто лошије резултате у односу на остале оптимизаторе, код већине модела. Разлог лошијих резултата *sgd* і *adagrad* лежи у имплементацији самих оптимизатора. Наиме, што се тиче *sgd* оптимизатора коришћен је класични *sgd* оптимизатор без моментума, док се код *adagrad* јавља проблем различитих промена по димензијама, димензије са већом вредношћу ће се мењати спорије, док ће код димензија са мањом вредношћу промена бити бржа.

С обзиром да постоји мали број радова који врши детекцију аномалија над овим скупом података, свега један док се остали ослањају на проблем класификације, његова тачност је 98.3%, може се закључити да модели креирани у оквиру овог пројекта дају задовољавајуће резултате, шта више постоје модели који имају бољу тачност од наведеног модела.

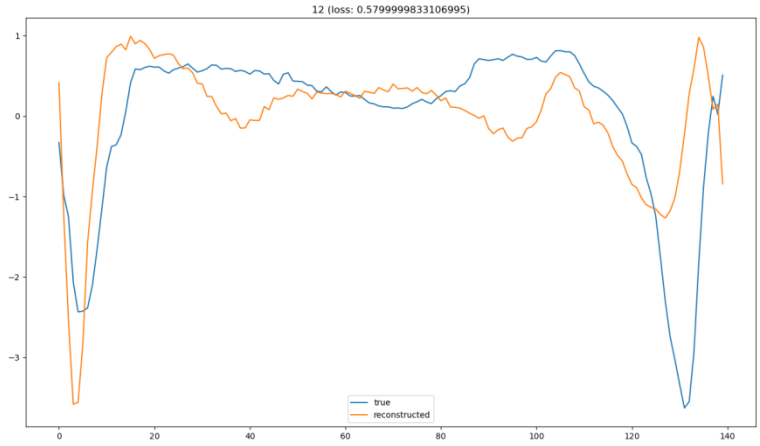
У оквиру пројекта је омогућена и визуелизација реконструисаних вредности улаза, као и детекција аномалије у односу на реконстрисану вредност, односно означавање места где је детектована аномалија. Као пример изгледа ових визуелизација биће узет модел који даје најбоље перформансе (потпуно повезани модел (*nadam*, средње квадратна грешка)).



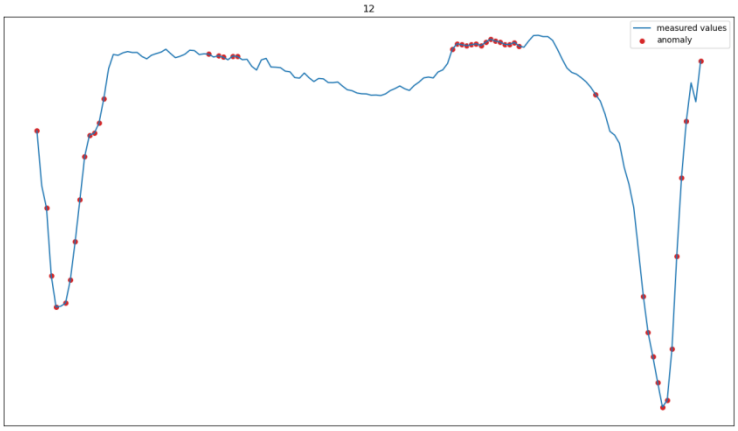
Слика 32: Реконструкција нормалног рада срца



Слика 33: Детекција аномалија нормалног рада срца



Слика 34: Реконструкција рада срца са аномалијом



Слика 35: Детекција аномалија рада срца са аномалијом

5. ЗАКЉУЧАК

Интересовање за употребу техника дубоког учења у медицини с годинама је све веће, стога све више истраживача свој рад фокусира управо ка овом сегменту. Детекција аномалија у медицини игра важну улогу у откривању разних болести. Идеја која се крије иза пројекта је могућност примене техника дубоког учења у детекцији аномалија рада срца. Из приложених резултата се види да су технике дубоког учења имају добре перформансе, могу да раде са великим скуповима података, могу да раде са комплексним скуповима података итд. У будућности, могу се применити други типови вештачких неуронских мрежа како би се упоредиле који типови вештачких мрежа дају најбоље резултате.