**Task 0:-** I worked on three datasets, namely

* GunPoint
* ArrowHead
* ECGFiveDays

**Task 1:-** Average Test Scores:-  
  
**FCN**

* Average Test Accuracy: 0.753
* Average Test Loss: 0.792

**RNN**

* Average Test Accuracy: 0.678
* Average Test Loss: 0.634

**CNN**

* Average Test Accuracy: 0.776
* Average Test Loss: 0.401

**CNN-LSTM**

* Average Test Accuracy: 0.677
* Average Test Loss: 0.571

The CNN model performed best based on the average test scores, with an average test accuracy of 0.776. On the three datasets used, it outperformed the other models.

Let's go through the code for each model and their construction:

1. FCN (Fully Connected Network):

The FCN model consists of a sequential stack of dense layers. Here's the breakdown of its construction:

* The input shape is flattened using the `Flatten` layer.
* It has three dense layers with 128, 64, and 32 units respectively, using the `Dense` layer with 'tanh' activation function and L2 regularization.
* The output layer consists of a single unit with the activation function specified by the `act\_fn` parameter.
* The model is compiled with the specified loss function, optimizer, and metrics.
* The training data is shuffled, and then split into a training set and a validation set.
* The model is trained using the training dataset with early stopping and model checkpointing callbacks.
* Finally, the model is evaluated on the test set, and the test loss and accuracy are printed.

2. RNN (Recurrent Neural Network):

The RNN model utilizes the LSTM layer to capture sequential dependencies. Here's the construction of the RNN model:

* The input shape is reshaped to include a third dimension using the `Reshape` layer.
* It has an LSTM layer with 64 units and the input shape specified by the `ip\_shape` parameter.
* The output layer consists of a single unit with the activation function specified by the `act\_fn` parameter.
* The model is compiled with the specified loss function, optimizer, and metrics.
* The training data is shuffled, and then split into a training set and a validation set.
* The model is trained using the training dataset with early stopping and model checkpointing callbacks.
* Finally, the model is evaluated on the test set, and the test loss and accuracy are printed.

3. CNN (Convolutional Neural Network):

The CNN model uses convolutional and dense layers to extract spatial features. Here's the construction of the CNN model:

* The input shape is reshaped to include a third dimension using the `Reshape` layer.
* It has a 1D convolutional layer with 32 filters, a kernel size of 3, and 'relu' activation.
* The output from the convolutional layer is downsampled using the `MaxPooling1D` layer.
* The output is then flattened using the `Flatten` layer.
* It has a dense layer with 64 units and 'relu' activation.
* The output layer consists of a single unit with the activation function specified by the `act\_fn` parameter.
* The model is compiled with the specified loss function, optimizer, and metrics.
* The training data is shuffled, and then split into a training set and a validation set.
* The model is trained using the training dataset with early stopping and model checkpointing callbacks.
* Finally, the model is evaluated on the test set, and the test loss and accuracy are printed.

4. CNN-LSTM:

The CNN-LSTM model combines the convolutional and LSTM layers to capture spatial and temporal dependencies. Here's the construction of the CNN-LSTM model:

* The input shape is reshaped to include a third dimension using the `Reshape` layer.
* It has a 1D convolutional layer with 32 filters, a kernel size of 3, and 'relu' activation.
* The output from the convolutional layer is downsampled using the `MaxPooling1D` layer.
* The output is then passed through an LSTM layer with 64 units.
* The output layer consists of a single unit with the activation function specified by the `act\_fn` parameter.
* The model is compiled with the specified loss function, optimizer, and metrics.
* The training data is shuffled, and then split into a training set and

**Task 2:-**

Sure! Here's a brief explanation of the two deep learning model structures that were constructed:

**1. Convolutional Neural Network (CNN):**

- The CNN model architecture consists of multiple convolutional layers followed by max-pooling layers to extract spatial features from the input data.

- The first convolutional layer has 128 filters with a kernel size of 5 and uses the ReLU activation function. It operates on the input shape of the data.

- Max-pooling layers with a pool size of 2 are added after each convolutional layer to reduce the spatial dimensions and retain the most important features.

- The final layer of the CNN model is a fully connected layer with 1 unit and uses the sigmoid activation function for binary classification.

- The model is compiled with the Adam optimizer, binary cross-entropy loss function, and accuracy metric.

**2. Multilayer Perceptron (MLP):**

- The MLP model architecture consists of multiple dense (fully connected) layers with dropout regularization to prevent overfitting.

- The model starts with a dropout layer with a rate of 0.1 to randomly deactivate 10% of the input units during training.

- It follows with three dense layers, each with 500 units and the ReLU activation function.

- Dropout layers with dropout rates of 0.2, 0.2, and 0.3 are added between each dense layer to further prevent overfitting.

- The final layer of the MLP model is a dense layer with 1 unit and uses the softmax activation function for binary classification.

- The model is compiled with the Adam optimizer, binary cross-entropy loss function, and accuracy metric.

These model structures were developed and trained on the provided datasets using the `preprocess\_data()` function to prepare the data for training. The `train\_and\_evaluate\_model()` function was used to train the model, evaluate its performance on validation and test data, and print the accuracy scores.

**Task 3:-**

Scores on each dataset with each model (total 9 scores):-

**Training CNN on GunPoint:**

Validation Accuracy: 0.9000000357627869

Test Accuracy: 0.753333330154419

**Training CNN on ArowHead:**

Validation Accuracy: 1.0

Test Accuracy: 1.0

**Training CNN on ECGFiveDays:**

Validation Accuracy: 0.4000000059604645

Test Accuracy: 0.49709638953208923

**Training MLP on GunPoint:**

Validation accuracy: 0.6000000238418579

Test accuracy: 0.4933333396911621

**Training MLP on ArowHead:**

Validation accuracy: 1.0

Test accuracy: 1.0

**Training MLP on ECGFiveDays:**

Validation accuracy: 0.4000000059604645

Test accuracy: 0.5029035806655884

**Training FCN on GunPoint:**

Validation accuracy: 0.5

Test accuracy: 0.4933333396911621

**Training FCN on ArrowHead:**

Validation accuracy: 1.0

Test accuracy: 1.0

**Training FCN on ECGFiveDays:**

Validation accuracy: 0.800000011920929

Test accuracy: 0.5029035806655884

**The Rank of average accuracy scores:** GunPoint Accuracy: 0.68, ArrowHead Accuracy: 1.0, ECGFiveDays Accuracy: 0.746

**Task 3:-**

* I chose a multi-modal architecture combining LSTM and CNN layers to process the three different datasets.
* The model architecture includes LSTM and CNN layers, with modifications such as global max pooling and multiple convolutional layers to capture relevant features.
* The LSTM layer and CNN layer are concatenated to combine their respective representations.
* A fully connected layer with a softmax activation is added to perform classification.
* The model is trained using the Adam optimizer with a learning rate of 0.001 and sparse categorical cross-entropy loss.
* Early stopping with a patience of 3 and model checkpointing are used during training to prevent overfitting and save the best model.
* The model is evaluated on three different datasets, and the test scores are reported as follows:

Dataset1 (Gunpoint):

Test Loss: 0.5232303738594055

Test Accuracy: 0.800000011920929

Dataset2 (Arrowhead):

Test Loss: 0.0

Test Accuracy: 1.0

Dataset3 (ECGFiveDays):

Test Loss: 0.4969989061355591

Test Accuracy: 0.7398374080657959

**Task 4:-**

**Choice of Dataset:**

The ECGFiveDays dataset was chosen for this experiment.

**Choice of Classifier and Transformer Pair:**

The chosen classifier and transformer pair is a Random Forest Classifier with a Rocket transformation followed by an Exponent Transformer. The Rocket transformation is a time series feature extraction method, and the Exponent Transformer applies exponential transformations to the data.

Parameters Optimization:

To optimize the performance of the classifier and transformer pair, a randomized search was conducted. The following parameters were explored:

- `estimator\_\_classifier\_\_max\_depth`: The maximum depth of the Random Forest Classifier.

- `estimator\_\_Rocket\_\_normalise`: A flag indicating whether to normalize the data during the Rocket transformation.

- `estimator\_\_ExponentTransformer\_\_power`: The power parameter for the Exponent Transformer.

The randomized search aimed to find the best combination of these parameters to maximize accuracy.

Best Score and Estimator:

The best score obtained from the randomized search instance was 1.0 (100% accuracy). The corresponding best parameters were:

- `estimator\_\_classifier\_\_max\_depth`: 13

- `estimator\_\_Rocket\_\_normalise`: True

- `estimator\_\_ExponentTransformer\_\_power`: 4

Test Score of the Best Model:

Using the best parameters obtained from the randomized search, a new pipeline was created with the optimized settings. This pipeline achieved a test score of 0.9523809523809523 (95.24% accuracy) on the ECGFiveDays dataset.

**Comparison with Default Setting:**

The default setting of the Random Forest Classifier was used as a baseline for comparison. The pipeline with default settings achieved a test score of 0.9628339140534262 (96.28% accuracy). Comparing this score with the best model's test score, we can observe a slight decrease in accuracy with the optimized parameters.

It's worth noting that while the best model obtained through parameter optimization did not outperform the default model, it's essential to consider the potential trade-offs and benefits of different parameter settings. The default model may have benefited from its default parameter values, while the optimized model may provide more interpretability or robustness in certain scenarios. Further analysis and experimentation could help gain deeper insights into the impact of different parameter settings on the model's performance.

**Task 5:-**

The best model in terms of the test score is the TapNet classifier, which achieved a test score of 0.9710. This is followed by the chosen classifier (RandomForest) with a test score of 0.9635, and finally, the deep learning model with a test score of 0.6667.

For the chosen classifier (RandomForest) and the deep learning model, the multivariate dataset is handled differently:

1. Chosen Classifier (RandomForest): The multivariate dataset is transformed using two transformers: Rocket and ExponentTransformer. The Rocket transformer converts the multivariate time series data into a univariate format, suitable for classification. It extracts features from each time series using random convolutional kernels. The ExponentTransformer then applies an exponential transformation to the data. The transformed data is then used to train a RandomForest classifier using a SklearnClassifierPipeline.

2. Deep Learning Model: The deep learning model uses a TapNet classifier, which is specifically designed for multivariate time series classification. The input shape of the model is determined based on the number of timesteps and features in the input data. The model consists of convolutional layers, dropout layers for regularization, pooling layers, and fully connected layers. The model is trained using the Adam optimizer and sparse categorical cross-entropy loss. The multivariate dataset is prepared as TensorFlow Datasets, where each sample in the dataset consists of a multivariate time series and its corresponding label.

Both approaches handle the multivariate dataset by transforming it into a suitable format for classification. However, the chosen classifier approach uses feature extraction techniques to convert the multivariate data into univariate format, while the deep learning model directly works with the multivariate data using convolutional layers. The choice between these approaches depends on the specific characteristics of the dataset and the desired trade-offs between interpretability, performance, and computational complexity.