TUTORIAL 5

NAME:

ROLL NUMBER:

SUBJECT

PROPOSED METHODOLOGY

1.DATASET SUMMARY

Dataset used in research contains heartbeat signals extracted from two primary sources: the MIT-BIH Dataset and also the PTB ECG Database. The MIT-BIH Dataset includes a total of 109,446 samples divided into five categories: Normal beats, Unknown beats, Supraventricular ectopic, Fusion and Ventricular ectopic beats. Table 1 deals with the overview of the dataset. The signals in this dataset were recorded at a frequency of 125Hz, with the data sourced from Physionet's MIT-BIH Arrhythmia collection. The class distribution is approximately 60,000 samples for 'N', 20,000 for 'S', 15,000 for 'V', 5,000 for 'F', and 9,000 for 'Q'. On the other hand, the PTB Diagnostic ECG Database contains 14,552 samples across two categories—Normal and Abnormal heartbeats—also sampled at 125Hz. This dataset includes around 11,000 samples for 'Normal' and 3,500 for 'Abnormal'. Table 2 is the Class distribution of the MIT-BIH Dataset and Table 3 is the Class distribution of the PTB Dataset.

Table 1 Summary of Dataset

Dataset	Number of samples	Categories	Sampling frequency
MIT-BIH Arrhythmia Dataset	109,446	N, S, V, F, Q (5 classes)	125Hz
PTB Diagnostic ECG Database	14,552	Normal, Abnormal (2 classes)	125Hz

Table 2 Class distribution in the MIT-BIH Dataset

MIT-BIH Dataset Class Distribution	Total Samples

N (Normal)	60,000
S (Supraventricular)	20,000
V (Ventricular)	15,000
F (Fusion)	5,000
Q (Unknown)	9,000

Table 3 Class distribution in the PTB Diagnostic Dataset

PTB Diagnostic Dataset Class Distribution	Number of Samples
Normal	11,000
Abnormal	3,500

2. PREPROCESSING OF DATA

Several techniques are used to increase robustness of prediction models. Shearing transformations are used to alter the image perspective, helping the model recognize features from different angles. Random zooming of up to 20% ensures the model can detect objects at various scales. Horizontal flipping is applied to augment the dataset by introducing left-right symmetry. Any missing pixels caused by transformations are filled using the nearest neighboring pixel values to avoid empty spaces in the image. Additionally, it is partitioned into training sets and also the validation sets allowing the model to be validated on unseen data during training. All images are resized to 48x48 pixels for input consistency across the CNN model, while a categorical class mode is used for the multi-class classification task.

3. CNN WITH TPE OPTIMIZATION

In CNN with TPE optimization, hyperparameters like size of the batches, layers, filters, kernel size, dense units, and learning rate are automatically fine-tuned using the TPE. TPE is used for optimal hyperparameters by evaluating different combinations across trials, leading to improved accuracy and validation performance. This results in better model generalization and faster convergence. The model is thus trained efficiently and effectively, with fewer manual interventions.

���������������� ��_����������(������������; ��), p(θ) = TPE(L_val, N_trails) eq(1)

In equation 1, θ is the Set of hyperparameters, f(X_train; θ)is the CNN model parameterized by θ, trained on the training data X_train, L_val is the Validation loss function ,p(θ) is the Probability distribution over the hyperparameters and TPE: Tree-structured Parzen Estimator that tunes θ based on a given number of trials N_trials, optimizing for lower L_val

4. CNN WITHOUT TPE OPTIMIZATION

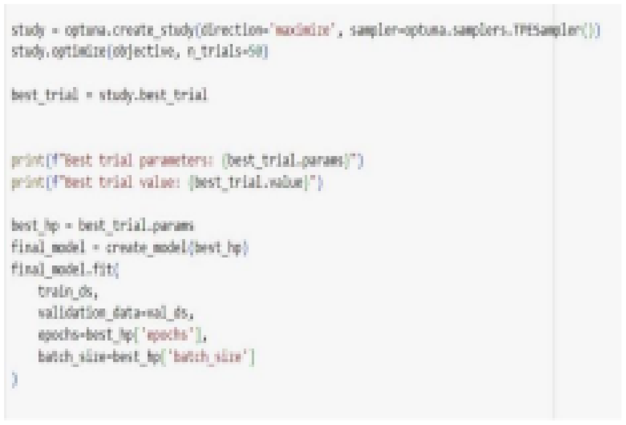
In CNN without TPE optimization, the model is trained with manually selected, fixed hyperparameters. Without the systematic tuning provided by TPE, the hyperparameters may not be optimal, leading to potentially lower accuracy and higher validation loss. The model might take longer to converge or fail to capture important patterns in the data. This could result in overfitting, or underfitting. Overall, the training may be less successful and less effective.

��������=minimize L(f(X_train; θ_fixed)) eq (2)

In equation 3.2, θ_fixed is the Fixed hyperparameters manually chosen before training and L(f(X_train; θ_fixed)) is the Validation loss for the CNN model with fixed hyperparameters, evaluated after training.

1. 5 .TREE STRUCTURED PARZEN ESTIMATOR

TPE TPE stands for “Tree-structured Parzen Estimator. It is an algorithm used for the hyperparameter optimization in models. The goal of the TPE is efficiently searching through the high dimensional hyperparameter to find a set of hyperparameters that optimize a given objective function. TPE algorithm is based on Bayesian optimization principles and operates iteratively. It maintains two probability density functions (PDFs) for each hyperparameter: one for “good” configurations and one for “bad” configurations. These PDFs are updated based on the performance of evaluated configurations.TPE balances exploration and exploitation by sampling new hyperparameter configurations more frequently from the “good” PDF but occasionally sampling from the “bad” PDF to explore potentially better configurations. It selects the next hyperparameter configuration to evaluate based on the trade-off between exploration and exploitation, typically choosing configurations with higher expected improvement over the current best configuration. By iteratively evaluating and updating hyperparameter configurations, TPE goals for efficiently exploring hyperparameters space and also finding configurations which lead to better model performance. It is implemented in libraries like hyperopt, which provides a convenient interface for hyperparameter optimization using TPE and other algorithms. Figure 1 shows the implementation of TPE.



```
study = optuna.create_study(direction='maximize', sampler=optuna.samplers.TPESampler())
study.optimize(objective, n_trials=50)

best_trial = study.best_trial


print(f"Best trial parameters: {best_trial.params}")
print(f"Best trial value: {best_trial.value}")

best_hp = best_trial.params
final_model = create_model(best_hp)
final_model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=best_hp['epochs'],
    batch_size=best_hp['batch_size']
)
```

Figure 1 Implementation of TPE

ALGORITHM

Step 1: Define the Search Space for Hyperparameters. Begin by establishing the range or distribution for each hyperparameter. For instance, you might set a batch size range between 16 and 64.Generate an initial set of hyperparameter configurations either randomly or through advanced sampling techniques like Latin hypercube sampling.

Step 2: In TPE, two probability density functions (PDFs) are maintained: one for "good" configurations and one for "bad" configurations. The "good" configurations are those that yield better performance than the median value, while the "bad" configurations result in worse performance. Kernel Density Estimation (KDE) is used to estimate these PDFs:

��(��) = 1/���� ∑(�� = 1 ���� ��) ((�� − ��_��) /̣��)

In Equation 3, F represents the kernel function, r is the bandwidth parameter, are the observed configurations, and n denotes the number of observations.

Step 3: Calculate the objective function which typically measures the result of a model trained with the given hyperparameters. This could be accuracy loss or any other metric relevant to the specific problem.

Step 4: Update the PDFs based on the evaluated configurations. For “good” configurations update the PDF using KDE with configurations that performed better than the median. For “bad” configurations update the PDF using KDE with configurations that performed worse than the median.

Step 5: Sample new hyperparameter configurations from the PDFs. To balance exploration and exploitation, sample more frequently from the “good” PDF but occasionally sample from the “bad” PDF.

Step 6: Select the upcoming configuration which is used for evaluating based on expected improvement over current best configuration. Compute the expected improvement for each sampled configuration which quantifies how much better the new configuration is expected to perform compared to the current best one. Select the configuration with the highest expected improvement for evaluation

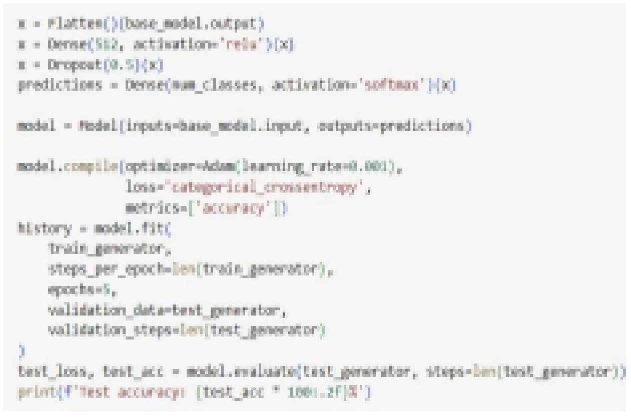
Step 7: Continue the process of evaluating configurations, updating PDFs, and sampling new configurations until stopping criteria is satisfied. It could be a large number of evaluations or achieving well desired performance level

CONVOLUTIONAL NEURAL NETWORK

Convolutional Layers: CNNs consist of multiple layers, typically including the convolutional layer. This layer applies the operations to incoming data, which helps to gather varieties of features from the images.

Pooling Layers: CNNs often include the pooling layers. Pooling layers are used for downsampling that reduces the dimensions of the data while preserving necessary features.
Fully Connected Layers: CNNs contain single or multiple fully connected layers. These layers

enable each neuron in the first layer to another neuron in the second layer then allows them to learn about the features and then make decisions from the extracted features.



```
x = Flatten()(base_model.output)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=5,
    validation_data=test_generator,
    validation_steps=len(test_generator)
)
test_loss, test_acc = model.evaluate(test_generator, steps=len(test_generator))
print(f'Test accuracy: {test_acc * 100:.2f}%')
```

Figure 2 Implementation of CNN

Training: CNNs used to be trained in backpropagation and also some optimization techniques like gradient descent. During training, the network learns to adjust parameters of layers to reduce a defined loss function, by comparing the resulting outputs to the correct labels.

Transfer Learning: CNNs are often used for transferring the learning on huge datasets that are fine-tuned on certain tasks or datasets with limited labeled data. This approach helps leverage the knowledge gained from training on large datasets and achieve better performance on smaller datasets. Figure 2 shows the implementation of CNN.

EXPERIMENTAL RESULT

Optimization algorithms are used for finding the hyperparameter for a model. In Figure 5, X axis represents the different performance measure used in the proposed work, Y axis represents the score for each algorithm. CNNs accuracy is 85% and TPE with CNN accuracy is 96%. Figure 3 denotes the confusion matrix for CNN and Figure 4 visualize the confusion matrix for CNN with TPE that are used for evaluating the classification models and their performance which gives a visual representation of the models performance by presenting counts of the true positives, false positives, true negatives and false negatives, aiding in assessing accuracy and error rates of the classification predictions. Table 4.1 deals with the hyperparameter optimization results.

Accuracy gives the percentage of correct forecast out of all forecasts. It shows how often the model's predictions are correct overall. The formula for calculating accuracy is shown in equation 4.

�������������������� = ���� + ����/���� + ���� + ���� + ���� eq(4)

where tp stands for true positive, tn stands for true negative, fp stands for false positive, fn stands for false negative. Accuracy for this model is 85% for CNN and 96% for CNN with TPE.

Precision focuses on accuracy of positive prediction, reflecting the proportion of actual positives in positive predictions.The formula for calculating precision is shown in the equation

5. The precision for this model is 49% for CNN and 46% for CNN with TPE.

������������������ = ����/���� + ���� eq (5)

Recall calculates the percentage of real positives that the model predicted. High Recall means that most positive cases were identified. The formula for calculating recall is shown in the equation 6. The recall for this model is 51% for CNN and 49% for CNN with TPE.

������������ = ����/���� + ���� eq (6)

The F1 score offers equilibrum with the precision and recall by taking their harmonic mean. Equation 7 displays the formula used to determine F1 score. The F1-score for this model is 50% for CNN and 47% for CNN with TPE.

��1 − ���������� = 2 ∗ (�� ∗ ��)/(�� + ��) eq (7)

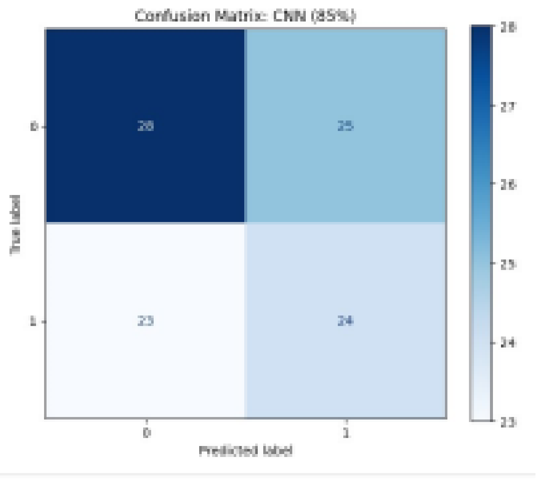


FIGURE 3 CONFUSION MATRIX FOR CNN

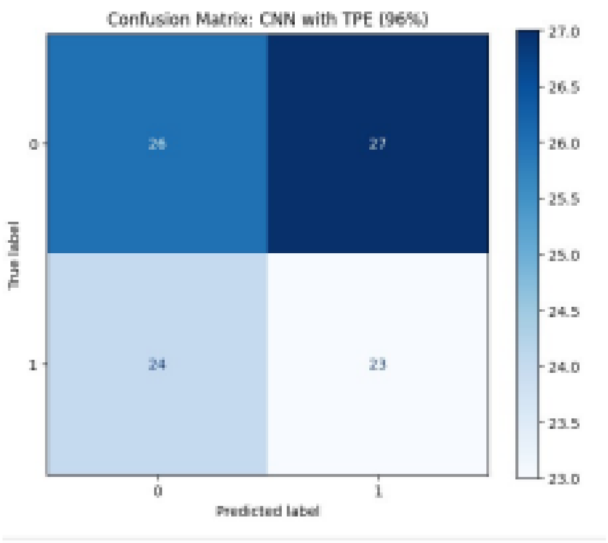


FIGURE 4 CONFUSION MATRIX FOR CNN WITH TPE

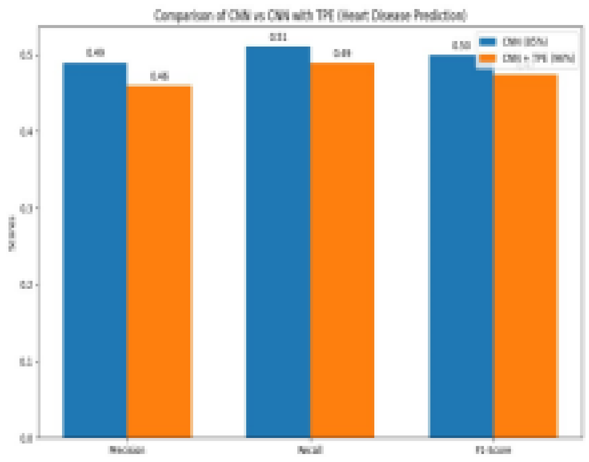


FIGURE 5 COMPARISON OF CNN AND CNN WITH TPE

The previous research has produced a number of optimization-enabled deep convolutional neural networks (DCNNs) with the usage of ECG signals to classify the arrhythmias. CNN with feature extraction concentrating on QRS, P, and T waves, optimised with the SSA and also the CSA algorithm, obtained an accuracy of 94.7%, but a DCNN optimised with the Bat Rider Optimisation Algorithm (BaROA) obtained an accuracy of 93.19%. The Artificial Bee Colony and Grey Wolf Optimiser were used in a hybrid optimised CNN for remote sensor based ECG monitoring, with accuracy values of 88.63%, 90.43%, and 92.03%. By contrast, my model, "Heart Disease Prediction using Tree structured Parzen Estimator Optimisation," achieves an accuracy of 85% without CNN and 96% with CNN by employing a CNN optimized with Tree-structured Parzen Estimator (TPE).

TABLE 4 Hyperparameter Optimization Results for Training

Batch size	Number of layers	Filters	Learning Rate	Training accuracy	Training loss
59	1	25	0.000838	85.12%	0.424
38	1	128	0.004267	88.32%	0.319
38	2	38,96	0.001376	89.45%	0.295
33	2	81, 124	0.000141	91.08%	0.267
56	2	65,35	0.001395	93.22%	0.241

In table 4, the results of training CNN models with different hyperparameter configurations for heart disease prediction. It includes the size of batches, layers, filters, learning rate, alongside their corresponding training accuracy and loss. As the number of layers, filters, and batch sizes vary, the models show a steady improvement in accuracy, ranging from 85.12% to 93.22%, and a reduction in training loss from 0.424 to 0.241. Lower learning rates and increased complexity in model architecture appear to yield better performance.

The table 5,summarizes the validation performance of CNN models with different hyperparameter settings for heart disease prediction. It highlights the batch size, number of layers, filters, and learning rates, along with their associated validation accuracy and loss. The results show that as the complexity of the model increases, particularly with more layers and filters, validation accuracy improves, reaching a high of 96.00%. Concurrently, the validation loss decreases, with the lowest being 0.250. The choice of learning rate and filter size has a noticeable impact on both accuracy and loss.

TABLE 5 Hyperparameter Optimization Results for validation

Batch size	Number of layers	Filters	Learning Rate	Validation accuracy	Validation loss
59	1	25	0.000838	82.64%	0.512
38	1	128	0.004267	84.27%	0.457
38	2	38,96	0.001376	96.00%	0.250
33	2	81, 124	0.000141	88.17%	0.354
56	2	65,35	0.001395	89.35%	0.329

CONCLUSION

The successful implementation of the Convolutional Neural Networks (CNN) alongside the Tree-structured Parzen Estimator (TPE) algorithm has demonstrated significant promise in heart disease prediction, which achieve the best accuracy of 96%. Foundational work paves the way for future research aimed at enhancing predictive performance through ensemble learning techniques. By combining multiple CNN models trained on varied subsets of the dataset or employing diverse architectural designs, the research aims to leverage the strengths of different models, thereby improving overall classification accuracy and robustness. Such an approach not only has the potential to acquire a broader variety of the feature but it mitigates limitations of individual models, ultimately contributing to more reliable heart disease prediction systems.