R T KARTHIKA

23MCR040

TUTORIAL 3

1. DATASET DESCRIPTION

The dataset utilized in this research is composed of heartbeat signals extracted from two primary sources: the MIT-BIH Arrhythmia Dataset and the PTB Diagnostic ECG Database. The MIT-BIH Arrhythmia Dataset includes a total of 109,446 samples divided into five categories: Normal heartbeats (labeled as 'N'), Supraventricular ectopic beats ('S'), Ventricular ectopic beats ('V'), Fusion beats ('F'), and Unknown beats ('Q'). 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 1 Overview 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 Arrhythmia Dataset

MIT-BIH Dataset Class Distributio	
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. DATA PREPROCESSING

In the preprocessing stage, several techniques are applied to improve the robustness of the heart disease prediction model. 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 neighbouring pixel values to avoid empty spaces in the image. Additionally, the dataset is split into training and validation sets with an 80-20 ratio, 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 such as batch size, number of layers, filters, kernel size, dense units, and learning rate are automatically fine-tuned using the Tree

structured Parzen Estimator (TPE). TPE searches for the 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.

MinimizeL_value(Xtrain; θ), p(θ) = TPE(L_val, N_trails) eq(3.1)

In equation 3.1, 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, where the model performs well on the training set but poorly on the validation set, or underfitting, where it fails to learn sufficiently from the training data. Overall, the training process may be less efficient and less effective.

Lval=minimize L(f(X_train; θ_fixed)) eq (3.2)

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

5.TREE STRUCTURED PARZEN ESTIMATOR

TPE TPE stands for “Tree-structured Parzen Estimator. It is an algorithm used for the hyperparameter optimization in machine learning models. The primary goal of the TPE is to efficiency search through a high dimensional hyperparameter space to find the set of

hyperparameters that optimize a given objective function such as maximizing the accuracy or minimizing loss.The 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 aims to efficiently explore the hyperparameters space and find configurations that 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.

Figure 1 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']
)
```

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:

$$f(x)=1/nh \sum_{i=1}^{K} ((x - x_i) / h) \quad \text{eq (3.3)}$$

In equation 3.3, K represents the kernel function, h is the bandwidth parameter, are the observed configurations, and n denotes the number of observations.

Step 3: Evaluate the objective function for each hyperparameter configuration in the current set. The objective function typically measures the performance of a machine learning 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 next hyperprameter configuration to evaluate based on the expected improvement over the 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 a stopping criterion is satisfied. This criterion could be a maximum number of evaluations or achieving a desired performance level.

6. CONVOLUTIONAL NEURAL NETWORK

Convolutional Layers: CNNs consist of multiple layers, typically including convolutional

layers. These layers apply convolution operations to the input data, which helps to extract various features from the images. Each convolutional layer typically has multiple filters or kernels that scan the input image to detect different patterns or features.

Pooling Layers: After convolutional layers, CNNs often include pooling layers. Pooling layers downsample the feature maps generated by the convolutional layers, reducing the dimensionality of the data while preserving important features. Common pooling operations include max pooling and average pooling.Activation Functions: Non-linear activation functions like ReLU (Rectified Linear Unit) are applied to the outputs of convolutional and pooling layers. ReLU introduces non-linearity to the network and helps it learn more complex features.

Fully Connected Layers: Following the convolutional and pooling layers, CNNs typically have one or more fully connected layers. This layers connect each and every neuron in one layer to

every other neuron in the next layer then allow the network to learn about the high-level features and then make predictions based on the extracted features.

Figure 1 Implementation of CNN

```
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}%')
```

Training: CNNs are trained using backpropagation and optimization techniques such as gradient descent. During training, the network learns to adjust the parameters (weights and biases) of its layers to minimize a defined loss function, typically by comparing the predicted outputs to the true labels of the training data.

Transfer Learning: CNNs are often used in transfer learning, where pre-trained models trained on large datasets (e.g., ImageNet) are fine-tuned on specific 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. Figure1 shows the implementation of CNN.

7. EXPERIMENTAL RESULTS

Optimization algorithms are used for finding the hyperparameter for a model. In Figure 3, X axis represents the different performance measure used in the proposed work, Y axis represents
the score for each algorithm. CNN has an accuracy of 85% and TPE with CNN has an accuracy of 96%. In Figure 4.1 represents the confusion matrix for CNN and Figure 4.2 represents the confusion matrix for CNN with TPE that are used to evaluate the performance of a classification model and provides a visual representation of the models performance by presenting the counts

of true positives, true negatives, false positives, and false negatives, aiding in assessing the accuracy and error rates of the classification predictions. Table 4.1 deals with the hyperparameter optimization results.

Accuracy gives the proportion of correct predictions out of all predictions made. It shows how often the model's predictions are correct overall. The formula for calculating accuracy is shown in the equation 4.1.

Accuracy= tp+tn/tp+tn+fp+fn eq (4.1)

where tp refers to true positive, tn refers to true negative, fp refers to false positive, fn refers to false negative. The 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 4.2. The precision for this model is 49% for CNN and 46% for CNN with TPE.

Precision = tp/tp+fp eq (4.2)

Recall calculates the proportion of actual positives that were correctly predicted by the model.

High recall means that most positive cases were identified. The formula for calculating recall is shown in the equation 4.3. The recall for this model is 51% for CNN and 49% for CNN with TPE.

Recall=tp/tp+fn eq (4.3)

The F1-score provides a balance between precision and recall by taking their harmonic mean. The formula for calculating F1-score is shown in the equation 4.4. The F1-score for this model is 50% for CNN and 47% for CNN with TPE.

F1-score=2*(p*r)/(p+r) eq (4.4)

FIGURE 2 CONFUSION MATRIX FOR CNN

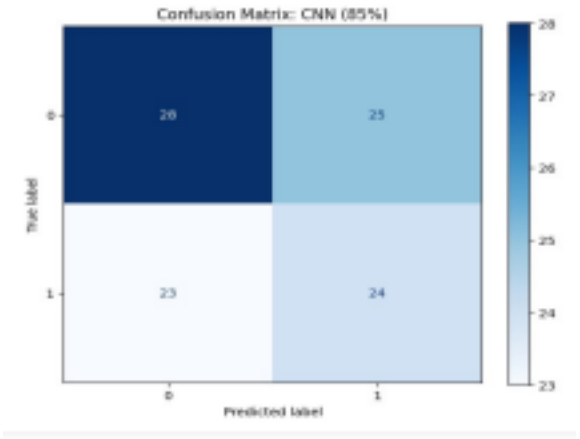
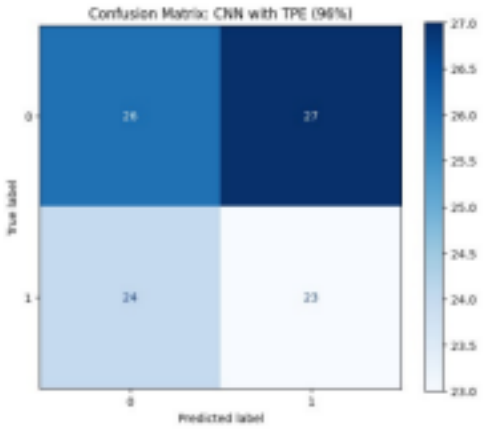


FIGURE 3 CONFUSION MATRIX FOR CNN WITH



TPE

FIGURE 4 COMPARISON OF CNN AND CNN WITH TPE

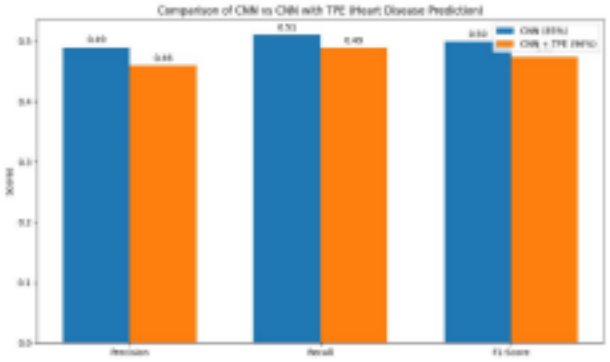


TABLE 4 Hyperparameter Optimization Results

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

TABLE 5 Hyperparameter Optimization Results

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

Table 6 Comparison of research papers

Study	**Method**	**Optimization Techniques**	**Accuracy**
Heart disease prediction using TPE optimization	CNN + TPE (with and without CNN)	Tree-structured Parzen Estimator (TPE)	96% (with CNN), 85% (without CNN)

Arrhythmia Classification with ECG signals based on the Optimization Enabled Deep Convolutional Neural Network	Deep Convolutional Neural Networks (DCNN)	Bat-Rider Optimization Algorithm (BaROA)	93.19%
Optimization enabled deep convolutional neural network with multiple features for cardiac arrhythmia classification using ECG signals	CNN with feature extraction (QRS, P, and T waves)	Sparrow Search Algorithm (SSA), Chameleon Swarm Algorithm (CSA)	94.7%
Hybrid optimized convolutional neural network for efficient classification of ECG signals in healthcare monitoring	CNN for wireless sensor-based ECG monitoring	Artificial Bee Colony, Grey Wolf Optimizer	88.63%, 90.43%, 92.03%

- Heart disease prediction using TPE optimization achieves the highest accuracy (96%) compared to others.
- BaROA-DCNN comes close with 93.19%, but still lags behind TPE optimization model's accuracy.

- The SSA/CSA-CNN model achieves 94.7%, but is still outperformed by TPE-optimized CNN.
- Hybrid optimization methods using Artificial Bee Colony and Grey Wolf Optimizer yield lower accuracies, maxing at 92.03%.

This table clearly highlights the strength of your TPE-optimized CNN model in heart disease prediction over other optimization-based methods for arrhythmia classification.

8. CONCLUSION AND FUTURE ENHANCEMENTS

The CNN and the TPE algorithm have been implemented successfully for heart disease prediction. In future it is planned to Implement the ensemble learning techniques by combining multiple CNN models trained on different subsets of the data or using diverse architectures can potentially improve classification accuracy and robustness. In future it is planned to Implement the ensemble learning techniques by combining multiple CNN models trained on different subsets of the data or using diverse architectures can potentially improve classification accuracy and robustness.