AN APPROACH TO EXPLAINABLE IMAGE CLASSIFICATION

Project Module Winter Semester 2025-2026

Supervisor

Milan Gnyjatovic

OCTOBER 5, 2025

ZEYNEP ÖYKÜ ERDEM

Abstract

This report addresses the problem of **Transparency, Accountability, Traceability, Interpretability and Trustworthiness** of Artificial Intelligence by focusing on Image Recognition prediction with the help of Explainable AI models and deep learning methods. As Artificial Intelligence, chatbots, agents, are becoming increasingly essential for modern applications, there is a growing necessity to move beyond the current “black boxes” by explaining the reasonings behind of these predictions. The main objective of this research is to assess and comprehend the logic underlying deep learning predictions for image recognition tasks. The research used Explainable AI techniques to achieve accuracy in deep learning predictions.

Table of Contents

I. Introduction

As Artificial Intelligence systems, ranging from predictive models to intelligent agents, achieve unprecedented predictive performance, their inherent “black box” nature constraints its deployment in sensitive sectors like healthcare, finance, and autonomous systems. This report addresses this challenge by integrating deep learning methodologies with Explainable AI techniques within the domain of image recognition. The main objective of this work is to evaluate and interpret the reasoning behind deep learning predictions for image recognition tasks. To achieve this, Convolutional Neural Network (CNN) is implemented to predict image classes on the CFIR-10 datasets, while Explainable AI technique, LIME, is used to analyze and visualize the contributions of individual features for model predictions. The research successfully demonstrated that XAI validates the model’s focus on essential features, leading to higher confidence in the system. The overall outcome is showing that deep that deep learning models, when combined with Explainable AI techniques, can provide accurate predictions while remaining interpretable and transparent.

The remainder of this report is structured as follows: Section 2 details the methodology, including the datasets, data pre-processing, hyperparameter tuning, optimization and regularization strategies , and CNN architecture. The training process, and the LIME implementation were given under Section 2. Section 3 presents the experimental results, analyzes the performance metrics, and discusses the interpretability findings derived from the LIME analysis. Finally, Section 4 offers the conclusion.

II. Methodology

Model Training and Evaluation

Environment Structure

Google Colab Pro was essential for this project. Without GPU acceleration specifically NVIDIA A100 , time efficiency and hardware performance could not be able to meet this complex hyperparameter tuning, optimization, and explanation ai technique.

Dataset

For this study, the **CIFAR-10 dataset** was employed, which is a well-known benchmark in computer vision research. The dataset consists of 60,000 images of size 32x32 pixels, equally distributed across ten different classes (airplane, automobile, bird, cat, deer, dog, frog, horse, horse, ship, and truck). The dataset was split into 50,000 training images and 10,000 testing images.

Model Architecture

Convolutional Neural Network Learning Model was used in Image Classification Task, while CNN achieved superior generalization in image classification, by leveraging the spatial structure of images using Convolutional Layers to feature extraction, Pooling Layers to reduce dimensionality, and Fully Connected Layers for classification. For this project, CNN was implemented using TensorFlow and Keras. The Architecture consists of two convolutional blocks with the following Activation Function, Max-Pooling, and Batch Normalization to stabilize training, overcoming overfitting. After the convolutional blocks, Flatten Layer was used to obtain one dimensional layer before the Fully Connected Layers. In Fully Connected Layers, one Hidden Dense Layer was used to train, final output layer with ten units, SoftMax activation function classifies images into the CIFAR-10 classes.

Data Preprocessing

To ensure the neural network receives normalized input, all images rescaled by a factor of 1/255. Additionally, data augmentation techniques of keras such as ImageDataGenerator class were considered to improve model generalization, reducing overfitting problems.

In ImageDataGenerator class provides several powerful methods for preprocessing operations on images. These include resizing, rotation, flipping, cropping, and normalization. These methods can be chained together to create a sequence to be applied to input images. Moreover, data augmentation techniques such as random cropping and flipping increase the size of the training dataset which improves model's performance and generalization.

Best val_accuracy So Far: 0.692300021648407
Total elapsed time: 02h 49m 12s
Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 32, 32, 3)	0
conv2d_2 (Conv2D)	(None, 32, 32, 32)	896
re_lu_2 (ReLU)	(None, 32, 32, 32)	0
batch_normalization_2 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d_2 (MaxPooling2D)	(None, 10, 10, 32)	0
dropout_2 (Dropout)	(None, 10, 10, 32)	0
conv2d_3 (Conv2D)	(None, 10, 10, 256)	73,984
re_lu_3 (ReLU)	(None, 10, 10, 256)	0
batch_normalization_3 (BatchNormalization)	(None, 10, 10, 256)	1,024
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 256)	0
flatten_1 (Flatten)	(None, 6400)	0
dense_2 (Dense)	(None, 64)	409,664
leaky_re_lu_1 (LeakyReLU)	(None, 64)	0
dropout_3 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 10)	650

Total params: 486,346 (1.86 MB)
Trainable params: 485,770 (1.85 MB)
Non-trainable params: 576 (2.25 KB)

Figure 1 Overview of Data Preprocessing Step Model Summary

```
batch_size = 32
epoch = 20
```

Figure 2 Batch and Epoch Size in Bayesian Optimization

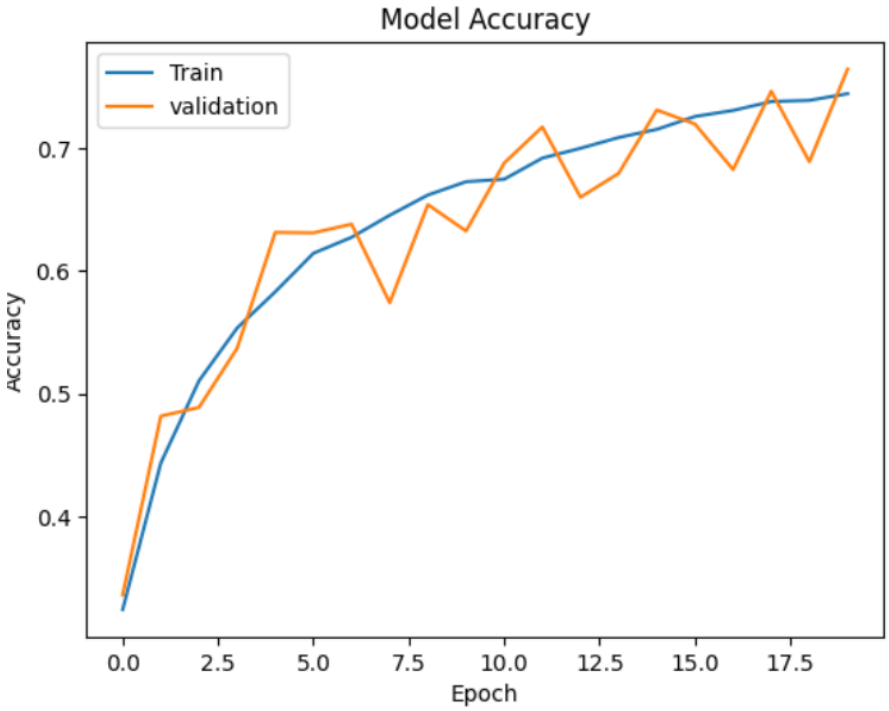


Figure 3 Model Accuracy Comparison of Training and Validation

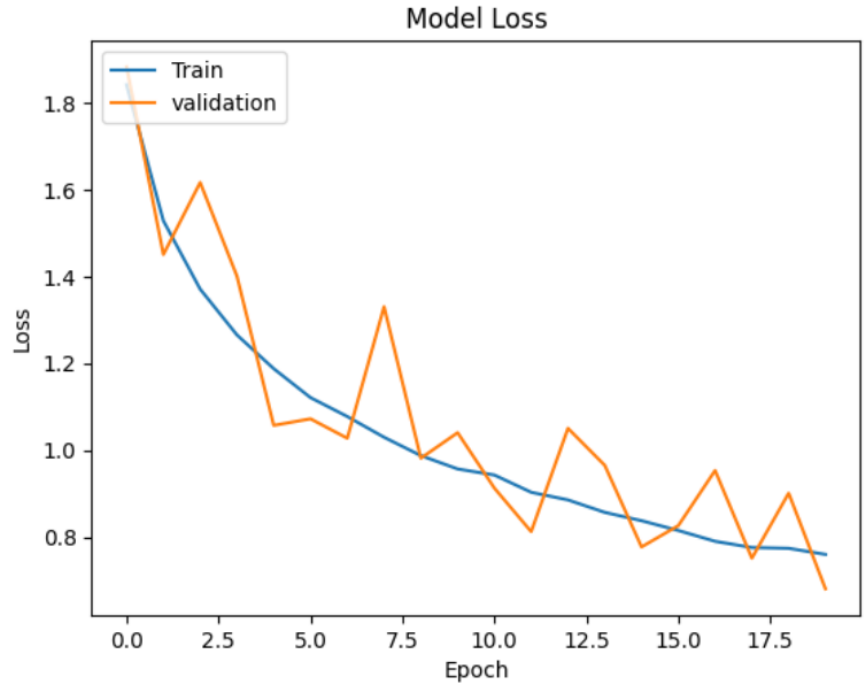


Figure 4 Model Loss Comparison of Training and Validation

The training and validation accuracy curves show a consistent upward trend throughout the epochs, indicating that the model successfully learns the patterns from the training datasets. The Validation accuracy closely follows the training accuracy, meaning that the model generalizes well to unseen data without significant overfitting. Similarly, both training and validation loss values decrease consistently, showing the effectiveness of the optimization process. Even though there are minor fluctuations observed in the validation loss, the overall decreasing implies stable convergence. Therefore, the model demonstrates balanced learning performance; however, to enhance its efficiency and prevent overfitting, **Hyperparameter Tuning, Optimization and Regularization** will be performed in the following stages.

Hyperparameter Tuning

Hyperparameter optimization is the process of selecting optimal values for a machine learning's hyperparameters. The goal is to find hyperparameter settings to minimize the loss function on a given dataset. There are several techniques to optimize hyperparameters, one widely used approach is **Bayesian Optimization.** In this study, I selected several key hyperparameters for the convolutional neural network to optimize:

- **Number of filters per layer**
  Controls the number of feature maps in each convolutional layer.
- **Kernel Size**
  Defines the size of the filters in convolutional layers.
- **Activation function**
  Defines the non-linearity in model.
- **Dropout rate**
  Helps prevent overfitting by randomly dropping units during training.
- **Learning rate**
  Controls the step size during weight updates, important in optimizers.
- **L2 Regularization**
  Penalizes large weights to minimize overfitting.
- **Optimizer**
  Controls weight updates to minimize the loss function.
- **Number of epochs**
  Defines the number of times that learning algorithm goes through the entire training datasets.

```
batch_size = 32
epoch = 20
```

Figure 5 Batch Size and Number of Epochs

Bayesian Optimization

Bayesian Optimization is a probabilistic model-based approach to optimize expensive functions, especially in deep learning hyperparameter tuning. It intelligently selects the next evaluation point, making optimization more efficient, and less over-computing. Bayesian optimization consists of two core components:

1. Surrogate Model
2. Acquisition Function

In Hyperparameter Tuning, Surrogate Model is an approximate model used by the Bayesian Optimizer to estimate the relationship between hyperparameters and performance, allowing it to efficiently explore and identify different promising parameter combinations reducing the need for extensive trial and error. While Acquisition function is used to evaluate the next parameters to be evaluated in the next iterations.

I have defined one function: build_hypermodel () which takes the parameter hp allowing us to use Bayesian Optimization to explore hyperparameters.

```
def build_hypermodel(hp):
    inputs = Input(shape=(32, 32, 3))
    x = inputs
    # First Conv Block
    filters_1 = hp.Choice('conv1_filters', values=[32,
64, 128])
    kernel_size_1 = hp.Choice('conv1_kernel_size',
values=[3, 5, 7])
    activation_func1=hp.Choice('activation_func1',values
=['relu','leaky_relu'])
    if activation_func1=='relu':
     x = Conv2D(filters_1, (kernel_size_1,
kernel_size_1), padding='same',
                kernel_regularizer=regularizers.l2(hp.Flo
at('l2_reg_1', 1e-5, 1e-2, sampling='log')))(x)
     x = tf.keras.layers.ReLU()(x)
     x = BatchNormalization()(x)
     x = MaxPooling2D(pool_size=hp.Choice('pool_size_1',
values=[2, 3]))(x)
    elif activation_func1=='leaky_relu':
     x = Conv2D(filters_1, (kernel_size_1,
kernel_size_1), padding='same',
                kernel_regularizer=regularizers.l2(hp.Flo
at('l2_reg_1', 1e-5, 1e-2, sampling='log')))(x)
     x =
tf.keras.layers.LeakyReLU(alpha=hp.Float('leaky_relu_alp
ha_1', 0.01, 0.3))(x)
     x = BatchNormalization()(x)
     x = MaxPooling2D(pool_size=hp.Choice('pool_size_1',
values=[2, 3]))(x)


    x = Dropout(hp.Float('dropout_1', 0.1, 0.5))(x)
```

Figure 6 build_hypermodel Function

```
    # Second Conv Block
    filters_2 = hp.Choice('conv2_filters', values=[64, 128,
256])
    kernel_size_2 = hp.Choice('conv2_kernel_size',
values=[3, 5, 7])
    activation_func2=hp.Choice('activation_func2',values=['r
elu','leaky_relu'])
    if activation_func2=='relu':
     x = Conv2D(filters_2, (kernel_size_2, kernel_size_2),
padding='same',
                kernel_regularizer=regularizers.l2(hp.Float('
l2_reg_2', 1e-5, 1e-2, sampling='log')))(x)
     x = tf.keras.layers.ReLU()(x)
     x = BatchNormalization()(x)
     x = MaxPooling2D(pool_size=hp.Choice('pool_size_2',
values=[2, 3]))(x)
    elif activation_func2=='leaky_relu':
     x = Conv2D(filters_2, (kernel_size_2, kernel_size_2),
padding='same',
                kernel_regularizer=regularizers.l2(hp.Float('
l2_reg_2', 1e-5, 1e-2, sampling='log')))(x)
     x =
tf.keras.layers.LeakyReLU(alpha=hp.Float('leaky_relu_alpha_2
', 0.01, 0.3))(x)
     x = BatchNormalization()(x)
     x = MaxPooling2D(pool_size=hp.Choice('pool_size_2',
values=[2, 3]))(x)
    # Flatten and Dense layers
    x = Flatten()(x)
```

Figure 7 build_hypermodel Function

```
     # First Dense layer

    dense_units_1 = hp.Choice('dense_units_1', values=[64,
128, 256, 512])
    x = Dense(dense_units_1,
kernel_regularizer=regularizers.l2(hp.Float('dense_l2_reg_1'
, 1e-5, 1e-2, sampling='log')))(x)
    activation_func_dense=hp.Choice('activation_func_dense',
values=['relu','leaky_relu'])
    if activation_func_dense=='relu':
     x = tf.keras.layers.ReLU()(x)
    elif activation_func_dense=='leaky_relu':
     x =
tf.keras.layers.LeakyReLU(alpha=hp.Float('leaky_relu_alpha_3
', 0.01, 0.3))(x)

    x = Dropout(hp.Float('dense_dropout_1', 0.2, 0.7))(x)

    # Output layer
    outputs = Dense(10, activation="softmax")(x)
    model = models.Model(inputs=inputs, outputs=outputs)
```

Figure 8 build_hypermodel Function

```
    # Optimizer selection
    optimizer_choice = hp.Choice('optimizer', values=['adam',
'sgd', 'rmsprop'])

    if optimizer_choice == 'adam':
        optimizer = tf.keras.optimizers.Adam(
            learning_rate=hp.Float('adam_lr', 1e-5, 1e-2,
sampling='log'),
        )
    elif optimizer_choice == 'sgd':
        optimizer = tf.keras.optimizers.SGD(
            learning_rate=hp.Float('sgd_lr', 1e-4, 1e-1,
sampling='log'),
            momentum=hp.Float('sgd_momentum', 0.8, 0.99)
        )
    else:  # rmsprop
        optimizer = tf.keras.optimizers.RMSprop(
            learning_rate=hp.Float('rmsprop_lr', 1e-5, 1e-2,
sampling='log'),
            rho=hp.Float('rmsprop_rho', 0.8, 0.95)
        )

    model.compile(
        loss='sparse_categorical_crossentropy',
        optimizer=optimizer,
        metrics=['accuracy']
    )

    return model
```

Figure 9 build_hypermodel Function

```
tuner=kt.BayesianOptimization(
                        hypermodel=build_model,
                        max_trials=20,
                        objective="val_accuracy",
                        executions_per_trial=1,
                        overwrite=True,
                        directory="hyperparameter_search",
                        project_name="explainable_ai"

            )
```

Figure 10 Creating Bayesian Optimization instance

In creating Bayesian Optimization instance, the given parameters were used to increase the performance of Bayesian search:

- **Hypermodel**
  Defines a callable that takes hyperparameters and returns a model instance.
- **Max_trials**
  Defines the maximum number of hyperparameter configurations to perform.
- **Objective**
  Specifies the metric to optimize.
- **Executions_per_trial**
  Defines the number of times each hyperparameter combinations to train.
- **Overwrite**
  Removes the oldest results if they have the same directory and project name.
- **Directory**
  Path to save the tuner results.
- **Project_name**
  Name of the project for organizing results

```
conv1_filters: 32
conv1_kernel_size: 3
activation_func1: relu
l2_reg_1: 4.732752932786752e-05
pool_size_1: 3
dropout_1: 0.3483260393937059
conv2_filters: 256
conv2_kernel_size: 3
activation_func2: relu
l2_reg_2: 0.003343648880295255
pool_size_2: 2
dense_units_1: 64
dense_l2_reg_1: 0.00013916755655764762
activation_func_dense: leaky_relu
dense_dropout_1: 0.2522645260486421 3
optimizer: adam
adam_lr: 8.038320164239253e-05
leaky_relu_alpha_3: 0.15102871289677247
sgd_lr: 0.0022463817204007573
sgd_momentum: 0.8435734770052908
leaky_relu_alpha_1: 0.0223386465883282
leaky_relu_alpha_2: 0.17330852186863419
rmsprop_lr: 4.027672556735733e-05
rmsprop_rho: 0.8320585123241646
```

Figure 11 Best Hyperparameter Values after Bayesian Optimization

Optimization and Regularization Strategies

To effectively minimize the loss function and improve model's generalization performance on the CIFAR-10 dataset, essential optimization and regularization strategies were implemented: Dropout, L2 Regularization techniques limit model's complexity and prevent overfitting, whereas an appropriate optimizer and learning rate find the set of hyperparameters that maximize objective function. Furthermore, EarlyStopping and ReduceLROnPlateau callbacks were employed to enhance training stability and avoid overfitting. During Hyperparameter Tuning, initialization of tuner with the objective "val_accuracy," search () function finds optimal hyperparameters regarding the highest validation accuracy, meanwhile EarlyStopping callbacks, to reduce overfitting, finding minimum validation loss throughout the trials. After the Hyperparameter Tuning, to optimize learning rate upon detecting a lack of performance in test validation metrics keras. callback. ReduceLROnPlateau callback was utilized by reducing Learning rate of the optimizer regarding the val_loss parameter. The number of epochs also increased to fifty, providing sufficient runtime for ReduceLROnPlateu mechanism and the Early Stopping callbacks to fully identify the optimal point of generalization.

Optimization

Table 1 Optimization Techniques used in Training.

Techniques	Explanation
Optimizer	**Optimizer while compiling the model controls how weights are updated during training**
Learning Rate	**Determines the step size of updates**
Learning Rate Scheduler	**Dynamically adjusting learning rate to improve convergence**

Regularization

Table 2 Regularization Techniques used in Training.

Techniques	Explanation
Dropout	**Randomly disables neurons during training**
L2 Regularization	**Penalizes large weights to avoid overfitting**
Early Stopping	**Stop training when validation loss stops reducing**
Batch Normalization	**Normalizes the inputs of each layer to stabilize and accelerate training**

Evaluation Metrics

Evaluation metrics were used to assess Model performance. The metrics that were used are:

- Train/Validation Accuracy/Loss Plot

Accuracy/Lost Plots are used to evaluate the model's performance whether the model is overfitting or underfitting to the training and validation data.

- Confusion Matrix

Confusion Matrix is an evaluation tool that summarizes the predictions results of a classification model, by displaying the number of correct and incorrect predictions categorized by each class.

- Classification Report

Classification Report allows us to analyze the metrics; precision, recall and f1-score of each member of the output classes.

Explainable AI (XAI)

Explainable AI (XAI) refers to methods and techniques that make the predictions of AI models more transparent and understandable. Local Interpretable Model-agnostic Explanations (LIME) is a technique used in this project that provides local interpretability of the model, offering individual predictions rather than entire dataset. In CNN, image classification models, LIME helps us understand why a model made a specific prediction for a single image. The input image is divided into groups of neighboring pixels with similar characteristics, each group acts like a feature that can be turned off or on during explanation. While creating perturbed versions of original image by randomly hiding or showing diverse groups of neighbor pixels, allowing model to see how predictions change, which groups were most influential in the prediction. In the end, the most important groups of the image contributed to the predicted class, making the model's decision more interpretable.

```
explainer = lime_image.LimeImageExplainer()# initializing explainer

explanation = explainer.explain_instance() # creating explanation
instance to identify the most important regions

temp_img,mask=explanation.get_image_and_mask()#get image and mask for
the top prediction

#Plotting the important regions
```

Figure 12 Explainer AI Implementation and Important Functions

III. Results and Discussion

```
Test accuracy: 0.7236999869346619
Test loss: 0.8845892548561096
Plotting started....
```

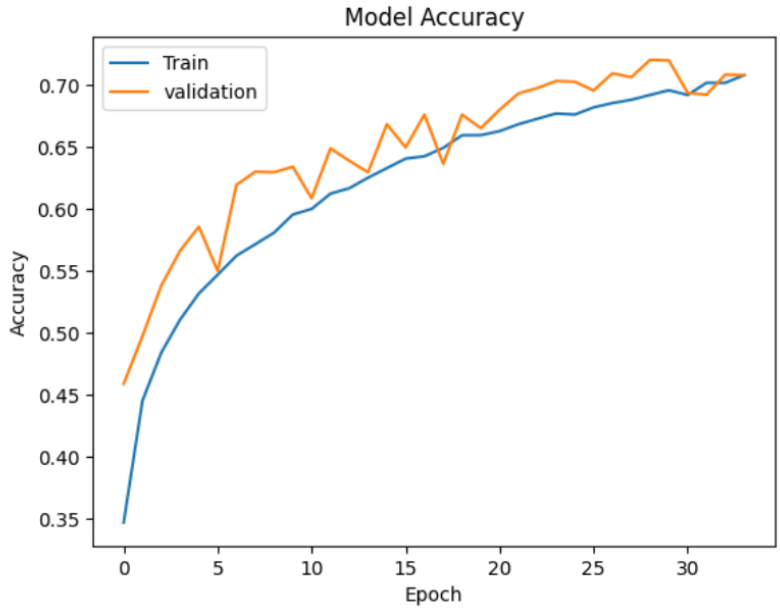


Figure 13 Model Accuracy Plot During the Optimization

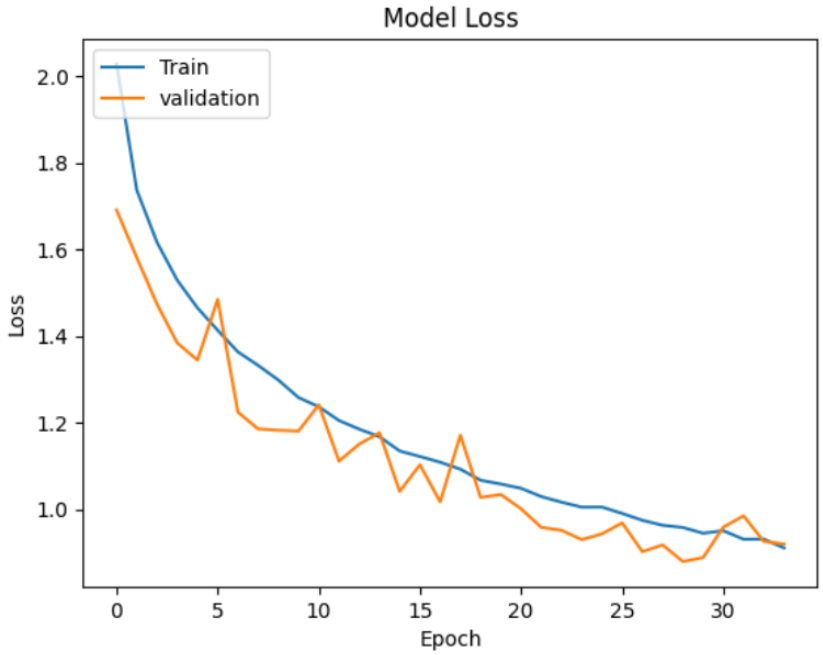


Figure 14 Model Loss Plot During the Optimization

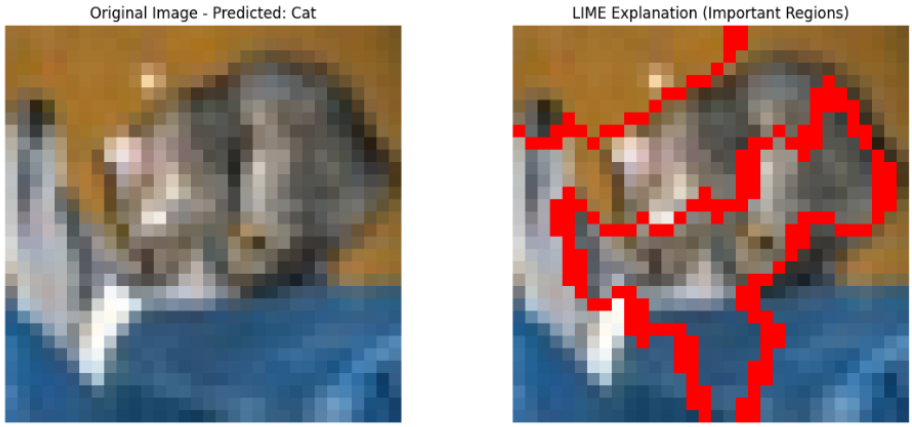
```
Confusion Matrix :
[[664  19  55   4  66  10  27  22  75  58]
 [  2 852   3   3   1   0  20   7  21  91]
 [ 38   1 462  21 130  59 191  70  10  18]
 [  9   8  30 361  83 179 199  76  22  33]
 [  3   1  11  11 713  17 158  75  10   1]
 [  7   4  17  74  61 625  77 110   9  16]
 [  1   2   3  14  21   5 940   5   6   3]
 [  3   2  11  11  46  15  22 872   4  14]
 [ 17  26   9   5  16   3  18   6 871  29]
 [  9  48   2   5   3   3  22  12  19 877]]

Classification Report :
              precision    recall  f1-score   support

    Airplane       0.88      0.66      0.76      1000
  Automobile       0.88      0.85      0.87      1000
        Bird       0.77      0.46      0.58      1000
         Cat       0.71      0.36      0.48      1000
        Deer       0.63      0.71      0.67      1000
         Dog       0.68      0.62      0.65      1000
        Frog       0.56      0.94      0.70      1000
       Horse       0.69      0.87      0.77      1000
        Ship       0.83      0.87      0.85      1000
       Truck       0.77      0.88      0.82      1000

    accuracy                           0.72     10000
   macro avg       0.74      0.72      0.71     10000
weighted avg       0.74      0.72      0.71     10000
```

Figure 15 Confusion Matrix and Classification Report for Image Classification Task



Prediction probabilities for sample 1:
  Airplane: 0.0012
  Automobile: 0.0009
  Bird: 0.0064
  Cat: 0.7430
  Deer: 0.0152
  Dog: 0.1110
  Frog: 0.0602
  Horse: 0.0005
  Ship: 0.0592
  Truck: 0.0023

Figure 16 Lime Image Plot for Prediction Probabilities and Important Regions

Prediction probabilities for sample 2:
  Airplane: 0.0009
  Automobile: 0.1371
  Bird: 0.0000
  Cat: 0.0000
  Deer: 0.0000
  Dog: 0.0000
  Frog: 0.0000
  Horse: 0.0000
  Ship: 0.8612
  Truck: 0.0008

Figure 17 Lime Image Plot for Prediction Probabilities and Important Regions

Figure 18 Lime Image Plot for Prediction Probabilities and Important Regions

Based on the Accuracy and Loss plots , we can confirm that implementation of Learning Rate Reducer caused, the validation and training curves to converge together, resulting stability and better generalization. LIME Explanation plots show that the model's decision-making process is highly localized ; when predicting an object, the model can assign importance to relevant features.

IV. Conclusion

The main purpose of this report is to address the need for **Transparency, Accountability, Traceability, Interpretability and Trustworthiness** of Artificial Intelligence applications. The interpretability and evaluation are the goal of this project in terms of using Explainable AI Techniques on Deep Learning Models. Throughout this goal, CFIR-10 dataset was trained using Convolutional Neural Network (CNN) model. LIME was successfully used to analyze predictions, providing boundaries in mage, by using super pixels to identify which groups are important to reach the prediction result.

V. References

[1] BayesianOptimization Tuner

[2] Interpretable Image Classification Using LIME | by A.I Hub | Medium

[3] Interpretable Machine Learning with LIME for Image Classification

[4] Explainable AI Unleashes the Power of Machine Learning in Banking

[5] Hyperparameter Tuning for CNNs: Best Techniques for Image Classification