Assignment - 1

Drithi Davuluri (B21AI055) G Mukund (B21CS092)

Part A: Implementation of LIME and SHAP

Diabetes Classification using Neural Networks:

Built a neural network model for the classification of diabetes. The dataset consists of several health-related features such as pregnancies, glucose levels, blood pressure, etc., and the task is to predict whether an individual has diabetes or not (binary classification).

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

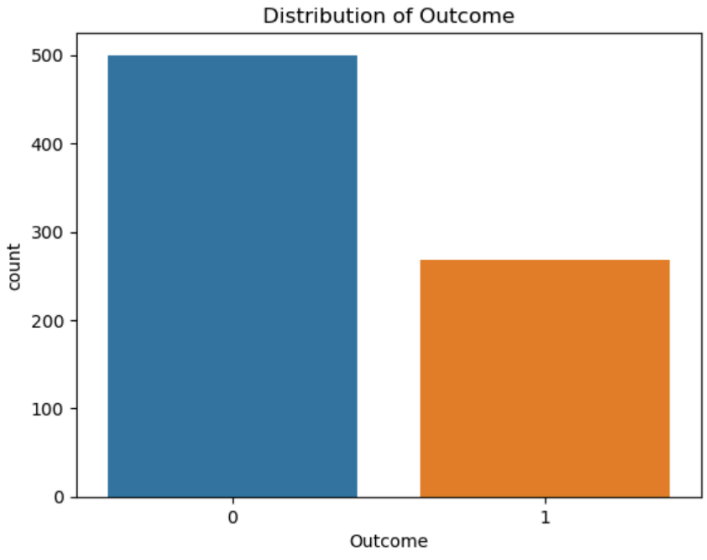
Exploratory Data Analysis (EDA)

A comprehensive EDA was conducted to understand the dataset's characteristics:

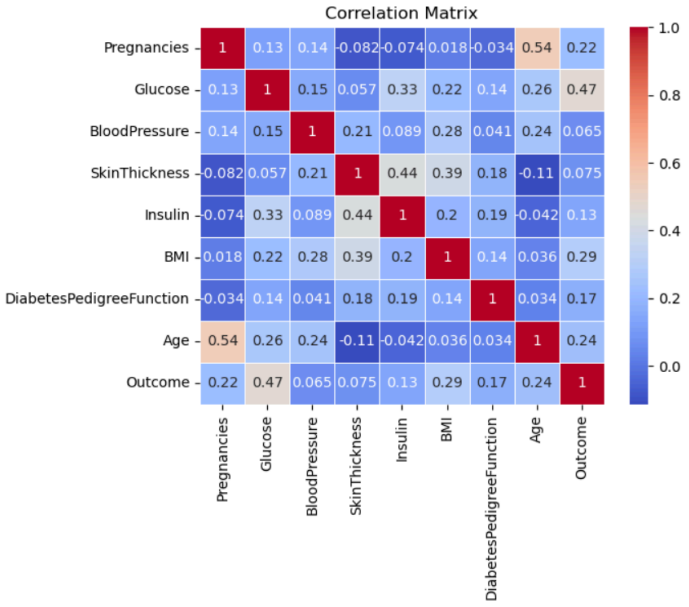
Descriptive statistics were calculated to provide insights into the central tendency and spread of numerical features.

No missing values were found, indicating a clean dataset.

The class distribution of the target variable ("Outcome") was visualized using a countplot.



A correlation matrix and heatmap were used to explore relationships between different features.



Data Preprocessing

The dataset was split into training, validation, and test sets. StandardScaler was applied to normalize the input features, ensuring consistent scaling across the dataset.

Model Architecture

A neural network model was designed with the following architecture:

Input layer with 8 neurons (matching the number of features in the dataset).
Hidden layers with 64, 32, and 16 neurons, respectively, using the ReLU activation function.
Dropout layers (with dropout rates of 0.5 and 0.3) were incorporated to prevent overfitting.
Output layer with 1 neuron and a sigmoid activation function for binary classification.

Model Training

The model was compiled using binary cross entropy loss and the Adam optimizer. Training was performed for 150 epochs with a batch size of 10. Validation data were used to monitor the model's performance on unseen data during training.

Model Evaluation

The trained model was evaluated on the test set to assess its performance. The final accuracy was calculated and reported.

LIME (Local Interpretable Model-Agnostic Explanations):

LIME was implemented to interpret the predictions of the neural network model for a specific instance in the test set. LIME is a method used to explain the predictions of complex machine learning models by approximating them with locally interpretable models.

Procedure:

Instance Selection:
A specific instance from the test set was chosen for explanation. The selected instance was the 2nd point from the test set.

Perturbation of Instances:
A set of perturbed instances was generated by introducing random noise to the selected instance. This step aims to simulate different possible variations (basically locality) of the original instance.

Similarity Calculation:
LIME relies on the concept of similarity between the original instance and the perturbed instances. Two similarity measures were employed:

1) Euclidean Distance: The Euclidean distance between the original instance and each perturbed instance was calculated. Instances closer in Euclidean space were considered more similar.
2) Cosine Similarity: The cosine similarity between the original instance and each perturbed instance was computed. Instances with a higher cosine similarity were regarded as more similar.

Weighting Samples:
Weights were assigned to each perturbed instance based on its similarity to the original instance. Instances that closely resembled the original instance received higher weights, while more dissimilar instances were downweighted.

Interpretable Model Training:
Interpretable linear models, specifically Ridge and Linear Regression, were trained using the perturbed instances. During the training process, weights were considered, allowing the models to approximate the behavior of the complex neural network model locally around the selected instance.

Feature Importance Visualization:

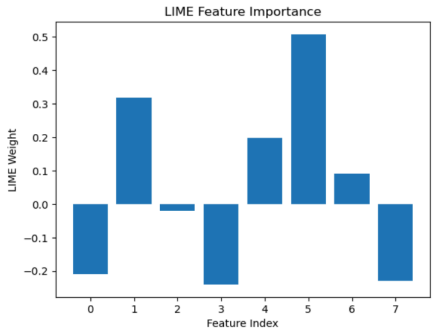
The resulting weights assigned by LIME to each feature in the interpretable models were visualized using bar plots. These visualizations provide insights into the relative importance of each feature for the selected instance's prediction.

Different methods used (hyperparameters)

Method 1: Ridge Regression with Euclidean Distance (LIME_1):

Ridge Regression was applied using the Euclidean distance-based similarity measure to generate feature weights. The resulting weights were visualized to interpret feature importance.

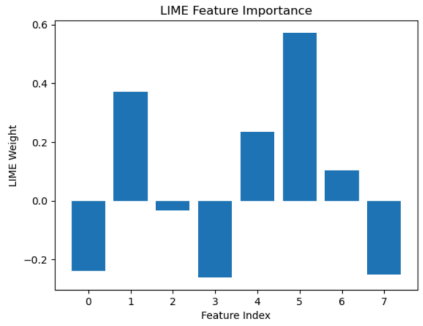
```
32/32 [==============================] - 0s 1ms/step
LIME_1 Feature Weights: [[-0.2085383   0.31814143 -0.02058433 -0.24037817  0.19847317  0.50677894
   0.09170566 -0.22943486]]
```



Method 2: Linear Regression with Euclidean Distance (LIME_2):

Linear Regression, another interpretable model, was employed with Euclidean distance as the similarity measure. The feature weights obtained were visualized.

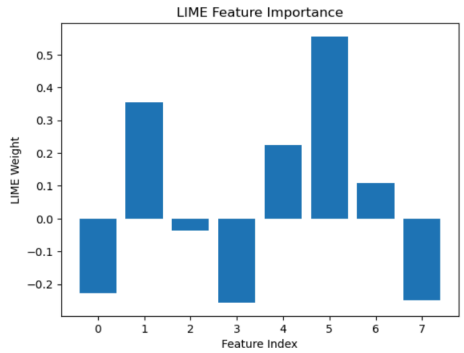
```
32/32 [==============================] - 0s 1ms/step
LIME Feature Weights: [[-0.24013939  0.37088224 -0.03268738 -0.26127317  0.23510212  0.57212438
   0.10420169 -0.25068867]]
```



Method 3: Ridge Regression with Increased Samples (LIME_3):

Ridge Regression was applied again, but this time with an increased number of perturbed samples (10,000). This method explored the impact of sample size on the interpretability of the model.

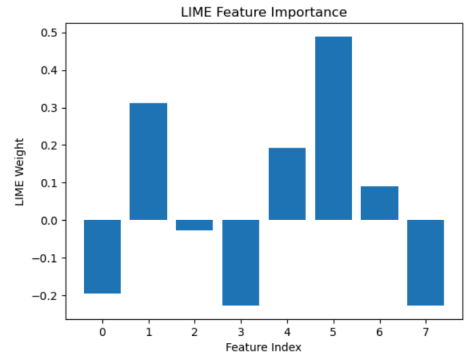
```
313/313 [==============================] - 1s 2ms/step
LIME Feature Weights: [[-0.22804844  0.35567927 -0.03561405 -0.25649934  0.22367757  0.55561293
   0.10930845 -0.2500854 ]]
```



Method 4: Ridge Regression with Cosine Similarity (LIME_4):

Ridge Regression was utilized once more, but with Cosine similarity as the measure of instance similarity. This method provided an alternative perspective on feature importance.

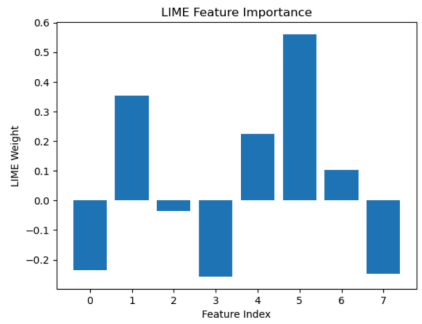
```
32/32 [==============================] - 0s 3ms/step
LIME Feature Weights: [[-0.19566458  0.31251763 -0.02664643 -0.22692088  0.19357023  0.48904837
   0.0906687  -0.22733926]]
```



Method 5: Linear Regression with Cosine Similarity (LIME_5):

Linear Regression was applied using Cosine similarity, offering a different approach to interpretability. The resulting feature weights were visualized.

```
32/32 [==============================] - 0s 2ms/step
LIME Feature Weights: [[-0.23398223  0.35349011 -0.03610874 -0.25682798  0.22445371  0.56035446
   0.10362479 -0.2467442 ]]
```



SHAP (SHapley Additive exPlanations) Implementation Report:

The SHAP (SHapley Additive exPlanations) approach was implemented to provide interpretable explanations for the predictions of a neural network model on a specific instance. SHAP values aim to fairly distribute the contribution of each feature to the model's prediction by leveraging cooperative game theory concepts, specifically Shapley values.

Procedure:

Shapley Value Calculation:

Shapley values represent the fair distribution of the contribution of each feature to the overall prediction of a machine learning model. The Shapley value of a feature is the average marginal contribution of that feature across all possible coalitions of features, i.e,
The Shapley value for feature *i* is calculated as follows:

$$\phi_i(f) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|! \cdot (|N| - |S| - 1)!}{|N|!} \cdot (f(S \cup \{i\}) - f(S))$$

where N is the set of all features, S is a coalition of features, f(S) is the model's prediction for the coalition S, and $|S|$ denotes the size of S.

Perturbation for Marginal Contributions:

Perturbed Instances:
To calculate the marginal contribution, instances are perturbed within each coalition. The perturbation involves adding random noise to features not present in the coalition while keeping the feature of interest unchanged.

Perturbation Scale:
The perturbation scale (ϵ) determines the amount of noise added to features. It is a crucial parameter influencing the stability and interpretability of SHAP values. A higher perturbation scale results in more significant perturbations.

Instance Perturbation Function:
The instance perturbation function $f(S \cup \{i\})$ introduces noise to features not in the coalition and leaves the feature i unchanged:
$f(S \cup \{i\})$=original instance+noise to features not in S

Visualization of SHAP Feature Values:

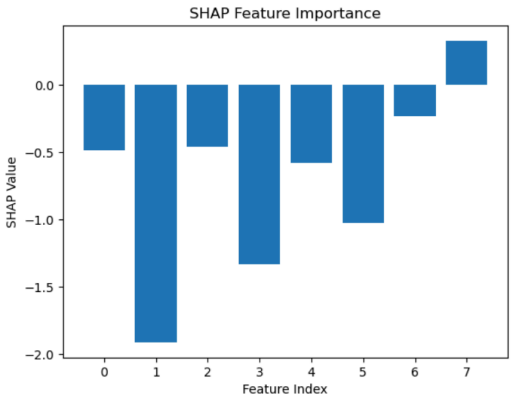
The resulting SHAP values for each feature were visualized using bar plots. This provides a clear understanding of the impact and importance of individual features in influencing the model's prediction for the selected instance.

Sensitivity Analysis with Different Perturbation Scales:

SHAP_1: Default Perturbation Scale:

SHAP values are initially calculated using a default perturbation scale (ϵ=0.1). This represents the baseline SHAP interpretation.

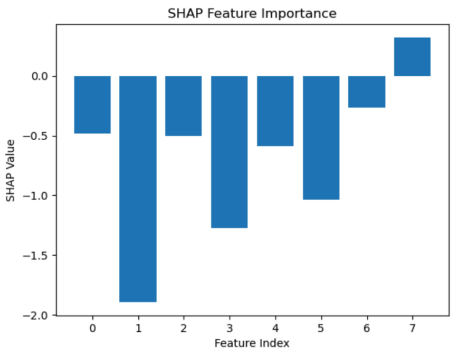
```
[array([-0.48327094], dtype=float32), array([-1.911834], dtype=float32), array([-0.45763573], dtype=float32), array([-1.3350862], dtype=float32),
array([-0.57752204], dtype=float32), array([-1.0237732], dtype=float32), array([-0.229985], dtype=float32), array([0.3278503], dtype=float32)]
```



SHAP_2: Reduced Perturbation Scale:

SHAP values are recalculated using a reduced perturbation scale (ϵ=0.05). This aims to assess how changes in the perturbation scale affect the stability and interpretability of SHAP values.

```
[array([-0.48161572], dtype=float32), array([-1.8929055], dtype=float32), array([-0.5015372], dtype=float32), array([-1.2714599], dtype=float32),
array([-0.5868987], dtype=float32), array([-1.0350531], dtype=float32), array([-0.26518995], dtype=float32), array([0.32138693], dtype=float32)]
```



Conclusion:

Utilizing LIME and SHAP techniques, we delved into interpretability. LIME revealed local feature importance through various interpretable models and perturbation scales. SHAP's Shapley values provided a fair distribution of feature contributions, offering nuanced insights.

Part B: Visualizations of CNN on GradCAM

Grad-CAM is a technique to visualize and explain CNN model predictions. It uses gradient information to identify important regions in the image for a predicted class. Grad-CAM generates a heatmap highlighting influential parts of the image for the model's decision.

Applying Grad-CAM to Explain VGG16 Predictions

The Grad-CAM (Gradient-weighted Class Activation Mapping) technique was implemented to provide visual explanations for predictions made by a pre-trained VGG16 ImageNet model son a sample image (fountain in our case).

Model Architecture and Image Selection

- The VGG16 model is a 16-layer deep convolutional neural network trained on the ImageNet dataset for image classification. It has learned rich feature representations.
- An image of a fountain was chosen from the Fountain dataset to pass through the model and visualize its decision process.

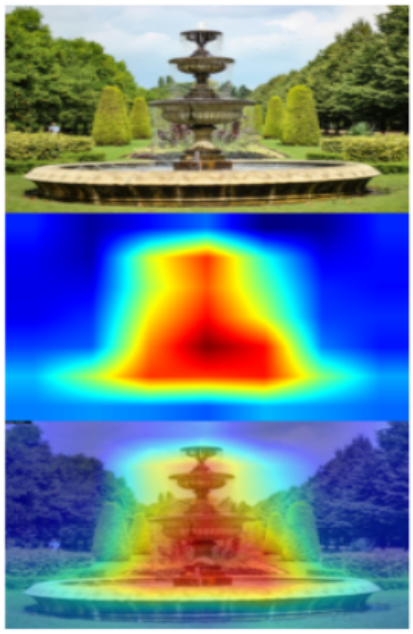
Forward Pass Prediction

- The image was preprocessed by resizing to 224x224 pixels and converting to an array.
- A forward pass was performed to obtain predicted class probabilities. The top prediction was "fountain" class.

```
[INFO]fountain: 100.00%
```

Generating Grad-CAM Heatmap

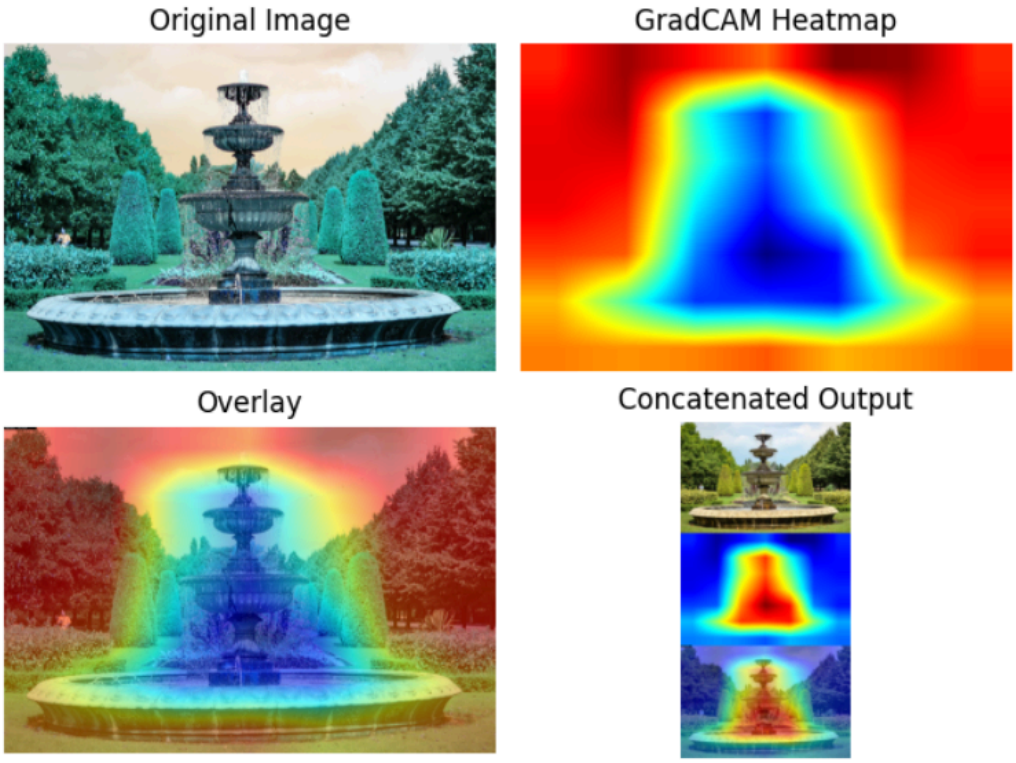
- The target convolutional layer was automatically identified using gradient information.
- Gradients were computed with respect to the loss for the predicted fountain class.
- The gradients were global average pooled to obtain importance weights for each feature map in the layer.
- A weighted combination of feature maps was taken using the weights to generate a coarse heatmap of the same size.
- The heatmap indicates the influential regions for the fountain prediction.



Overlay for Enhanced Visualization

- The heatmap was resized to the input image dimensions.

- It was overlayed on the original image with a heat colormap and transparency to visually augment the explanation.
- This highlighted image regions relevant to the VGG16 model in predicting the fountain class.



Key Highlights

- The Grad-CAM heatmap accurately localized the fountain in the image.
- The technique provided insights into the model's attention areas that led to the prediction.
- No retraining was required, demonstrating the versatility of Grad-CAM.

Conclusion

The Grad-CAM implementation successfully generated visual explanations for the VGG16 model's predictions. The meaningful heatmap localization and intuitive overlay revealed which parts of the image were most influential for the model's decision, enhancing transparency and interpretability.