HW5–Explainability

Zihan Zhao

May 2025

1 Introduction

In this assignment, we apply attribution methods—LIME, Shapley Values, and SmoothGrad—to interpret model predictions. We use a CNN trained on the Animal-10B dataset for image classification and some tree-based models to analyze diabetes prediction. These methods help identify which input features most influence each model's decisions.

2 Data Cleaning

2.1 Imbalanced Data

From the outcome of the Diabetic data(figure 1), we can see the data is imbalanced with most people not being diabetic while only a few of them are diabetic. The percentage of two different types of people are:

- Diabetic patients: **34.90%**
- Normal people: **65.10%**

I chose not to resample the minority class, as it represents around 30–40% of the total data. At this level, the class imbalance is moderate and unlikely to significantly impact the model's performance.

2.2 Predictive Modeling

2.2.1 Logistics Regression

To assess feature importance in diabetes prediction, we employed Logistic Regression with an L1 penalty.

- The model was trained using a range of inverse regularization strengths $C \in \{0.001, 0.01, 0.05, 0.1, 0.5, 1, 5, 10\}$.
- The dataset was split into a training set (80%) and a test set (20%). For each C, the model was fitted on the training data.
- We computed accuracy scores for each model and selected the C that yielded the highest test accuracy while avoid overfitting problems. From the figure 2, the best result was achieved at $C = 0.5$, with an accuracy of 0.8247.

$$\text{Accuracy Score} = \frac{N_{\text{correct}}}{N_{\text{total}}}$$

And the feature importance is shown in the table 1. From the table, we can see the DiabetesPedigreeFunction affects the output most, while Insulin and SkinThickness have insignificant influence on the output. As C is large and the penalty is small, all features are selected.

2.2.2 Random Forest

We applied a Random Forest Classifier to the diabetes dataset and analyzed feature importance shown in the table 2. The overall process is described below in three parts:

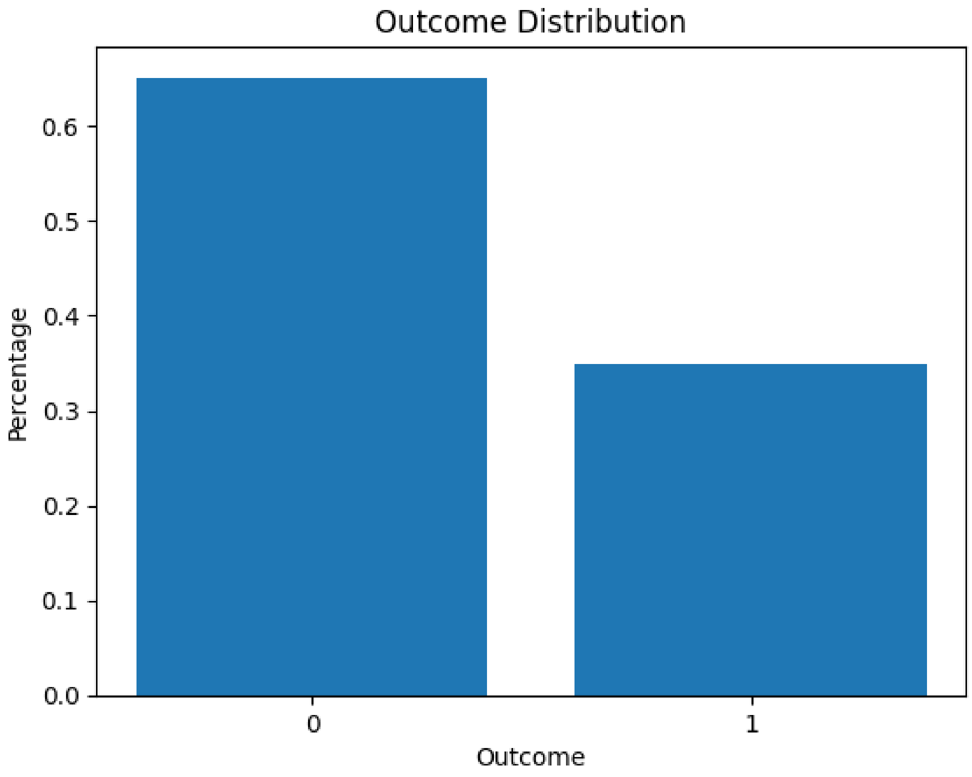

Figure 1: Diabeted imbalanced Data

Table 1: Logistics Coefficients for Selected Features ($C = 0.5$)

Feature	**Coefficient**
Pregnancies	0.0864
Glucose	0.0308
BloodPressure	-0.0152
SkinThickness	0.0046
Insulin	-0.0011
BMI	0.0741
DiabetesPedigreeFunction	0.6006
Age	0.0168

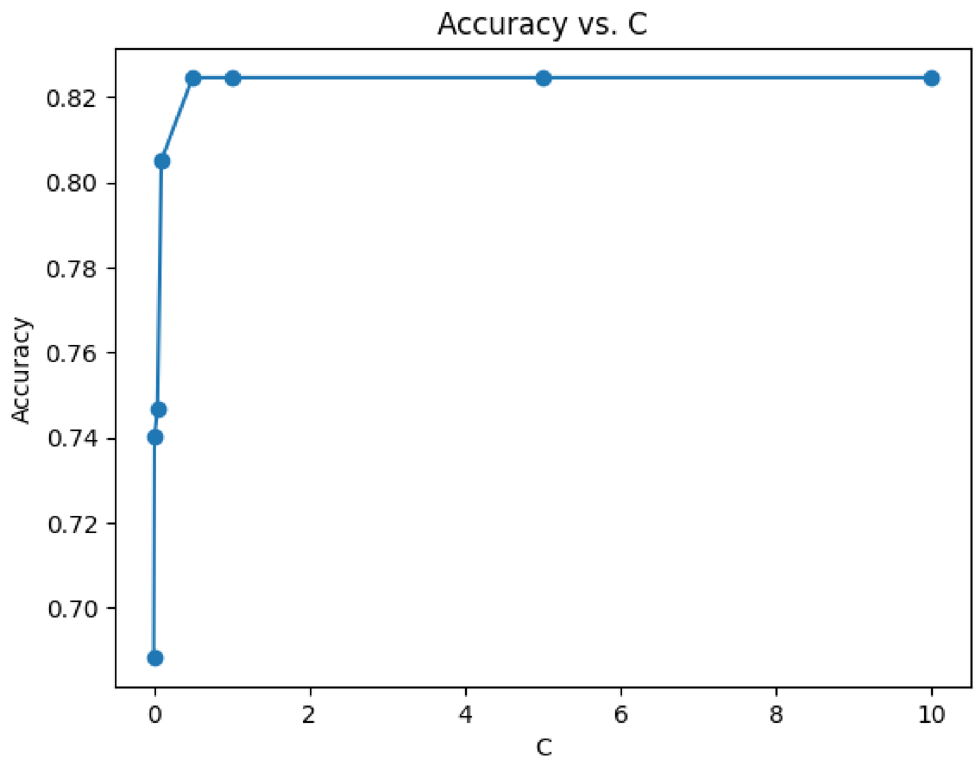

Figure 2: C selection

Model Configuration

- **Model:** We used the `RandomForestClassifier` from `scikit-learn`.
- **Number of Trees (`n_estimators`):** Set to 100 to ensure stable ensemble learning.
- **Maximum Tree Depth (`max_depth`):** Limited to 5 to prevent overfitting.
- **Random Seed (`random_state`):** Set to 0 for reproducibility.

Training and Prediction

- The model was trained on the training dataset using all input features.
- Class labels for the test set were predicted using the `predict()` method.
- Use the 'rfc.feature_names_in_' method and show in the table 2

Feature Importance Interpretation

- **Glucose** is the most important predictor with an importance score of 0.3226.
- **Age** (0.1735) and **BMI** (0.1694) are also strong predictors.
- **DiabetesPedigreeFunction** and **Pregnancies** contribute moderately.
- **BloodPressure** and **SkinThickness** show relatively low importance, indicating limited predictive value under current model settings.

Comparing the two models, we found that the same feature can have different levels of importance depending on the prediction model.

Table 2: Random Forest Feature Importances

Feature	**Importance**
Glucose	0.3226
Age	0.1735
BMI	0.1694
DiabetesPedigreeFunction	0.0952
Pregnancies	0.0684
Insulin	0.0664
SkinThickness	0.0529
BloodPressure	0.0516

2.3 Visualization

2.3.1 LIME method

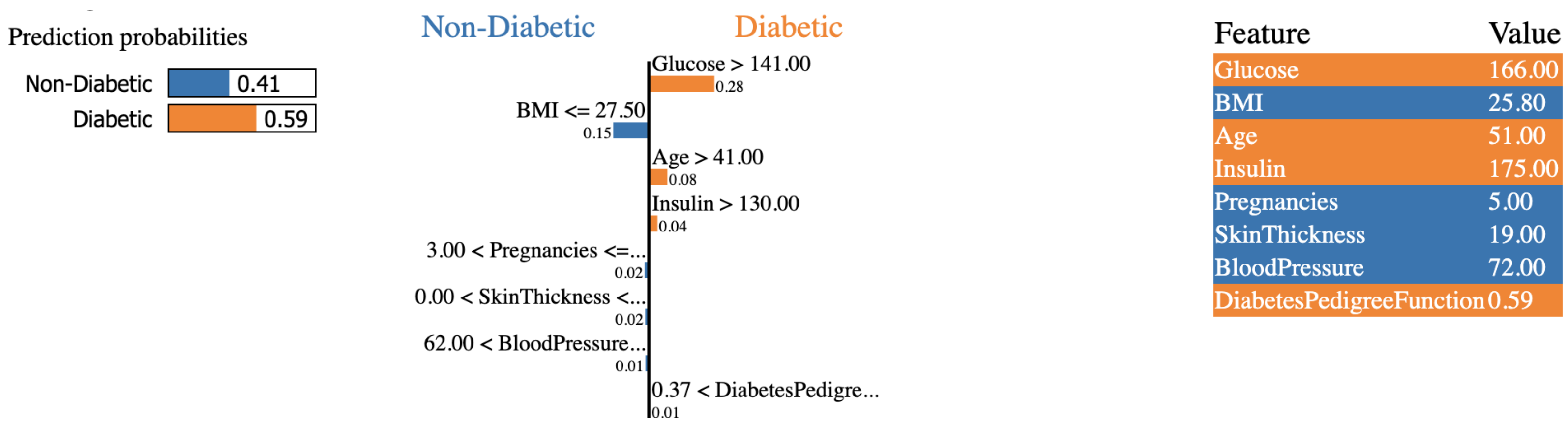

Figure 3: LIME for idx=3

We use LIME (Local Interpretable Model-agnostic Explanations) to explain a prediction made by a **Random Forest** classification model. I chose the data point that has the index=3 and the output is shown in the figure 3. For the selected test instance, the model predicted a 59% probability of the patient being **Diabetic**, and a 41% probability of being **Non-Diabetic**.

LIME breaks down the prediction into locally important features. The following features **pushed the prediction toward Diabetic**:

- **Glucose $>$ 141.00** (weight = 0.28)
- **Age $>$ 41.00** (weight = 0.08)
- **Insulin $>$ 130.00** (weight = 0.04)
- **DiabetesPedigreeFunction $>$ 0.37** (weight = 0.01)

The following features **pulled the prediction toward Non-Diabetic**:

- **BMI $\leq$ 27.50** (weight = -0.15)
- **Pregnancies $\leq$ 3.00** (weight = -0.02)
- **SkinThickness $\leq$ 0.00** (weight = -0.02)
- **BloodPressure $\leq$ 62.00** (weight = -0.01)

LIME is designed to provide **local** interpretability, meaning it generates explanations specific to each individual data point. Because of this locality, **LIME is not guaranteed to be stable across different data points**.

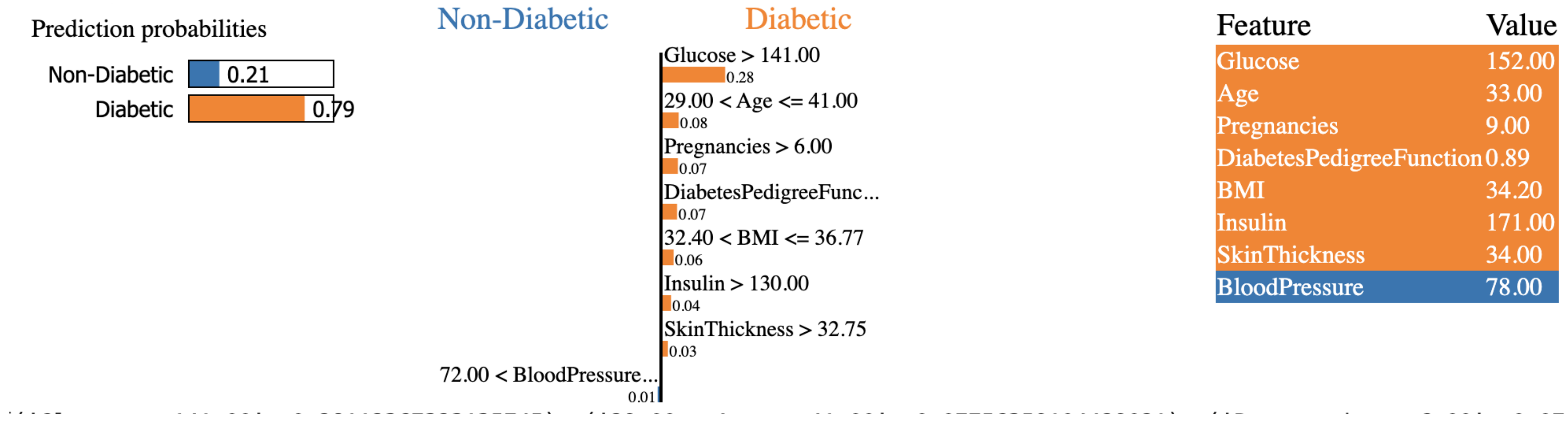

Figure 4: LIME for idx=6

We also chose another data point as the benchmark and compared with the base data point(index=3). You can see from the figure 4, LIME selected the same features but assigned **different importance weights** to the same features. For index=3 point, the top-3 important features are Glucose, BMI and Age, while for the index=6 point, the top-3 important features are Glucose, Age and Pregnancies. But LIME of different data points has similar pattern such as:

- Glucose are the most important features that determine whether the patient is diabetic or not.
- Insulin, SkinThickness and BloodPressure are not important when determining whether the patient is diabetic or not.

2.3.2 SHAP method

Global feature importance The figure 5 conveys both feature importance and feature effect direction in a compact form. The color of each dot represents the actual value of the feature for a given instance (red = high, blue = low), and the position indicates the strength and direction of its effect. We can get an intuitive, model-agnostic understanding of feature influence across the entire dataset.

- **Glucose** is the most influential feature. Higher glucose values (red) strongly increase the predicted probability of diabetes.
- **Age** and **BMI** also have significant effects, where higher values tend to push predictions toward the diabetic class.
- Other features such as **Pregnancies**, **Insulin**, and **DiabetesPedigreeFunction** show moderate contributions.

Local feature importance The figure 6 conveys the feature importance for a specific data point(index = 3). The key components of the plot are:

- The **base value** (0.3746), which represents the average model prediction across the dataset.
- The final prediction $f(x) = 0.59$, which is the model's predicted probability for this individual.
- **Red arrows** show features that increase the prediction probability (toward Diabetic), while **blue arrows** indicate features that decrease the prediction (toward Non-Diabetic).
- The **arrow size** reflects the magnitude of the contribution from each feature.

In this case, the model starts from the base value of 0.3746 and predicts a probability of 0.59 for the Diabetic class. The most influential features pushing the prediction higher were: **Insulin = 175**, **Age = 51**, **Glucose = 166**. One feature had a moderate negative contribution: **BMI = 25.8**.

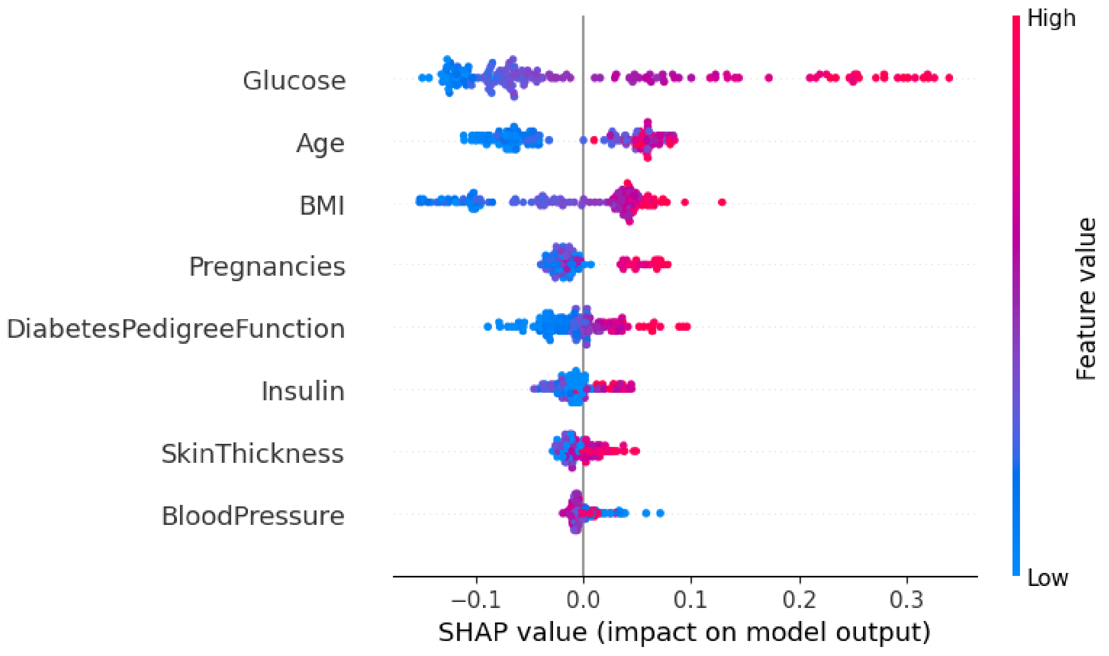

Figure 5: SHAP for global feature importance

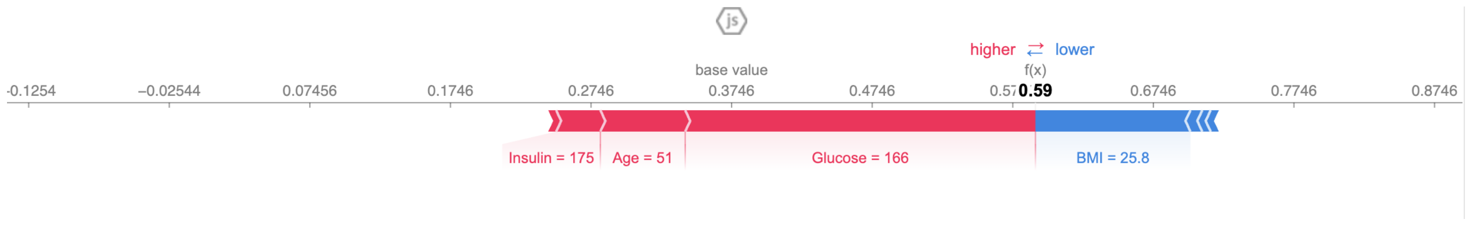

Figure 6: SHAP for the data point(index=3)

2.4 Comparison

We conducted two types of comparisons based on their scope. First, we compared the linear model's feature importance from random forests and global SHAP values. Second, we compared LIME and local SHAP, as both focus on individual data points.

Global Scope Compared to three models, we found that the most important feature for the linear model is DiabetesPedigreeFunction while the most important feature for the SHAP and random forest is Glucose.

This difference likely arises because random forests account for feature interactions and hierarchical splitting. As a result, features that perform well in combination with others—such as Glucose—tend to be assigned higher importance. In contrast, logistic regression does not inherently model interactions unless they are explicitly specified (e.g., through interaction terms), and therefore assigns weights based solely on each feature's individual marginal effect.

Local Scope For a specific data point, I think LIME and SHAP are very similar, because they both consider Glucose to be the most important feature for prediction, while BMI is the negative factor; Age and Insulin are the second and the third ones affecting the output. I think the reasons are the following:

- LIME fits a local surrogate linear model around the data point. SHAP values approximate each feature's marginal contribution, which also resembles a locally linear explanation in many models. So if the model behaves approximately linearly near that data point, both methods will highlight similar features.
- When a small set of features (e.g., Glucose, BMI) clearly drives the output, both methods will assign them high importance.
- When the model is stable and fit the output very well, LIME and SHAP sometimes have similar results.

3 Predictive Modeling on Animal Images

3.1 Visualization Process

The visualization of this problem can be seen from the figure 7. Here I only showed my image from the images sets.

- **Dataset:** Animal-10N is a noisy-label image classification dataset with 10 animal classes (e.g., cat, dog, chicken).
- **Input Size:** All images are resized to $64 \times 64 \times 3$ (RGB).
- **Split:** 45,000 images for training and 5,000 for validation.
- **Image Shown:** A sample input from the dataset after resizing. Despite lower resolution, class-relevant features remain visible.

3.2 Linear Model

We considered a linear model that has the following parameters in table 3. Here B means batch size of each epoch and I let it equal to 128. Then I collected the validation loss and accuracy and drew a plot showing the accuracy in figure 8 and also the loss in figure 9.

3.3 Vanilla Model

We constructed a vanilla model that has the following parameters and drew a plot showing the training accuracy and validation accuracy:

- **Input:** RGB image of size $64 \times 64 \times 3$
- **Convolutional Layers:**
  - Conv ($3 \rightarrow 32$), ReLU, MaxPool $\rightarrow 32 \times 32 \times 32$

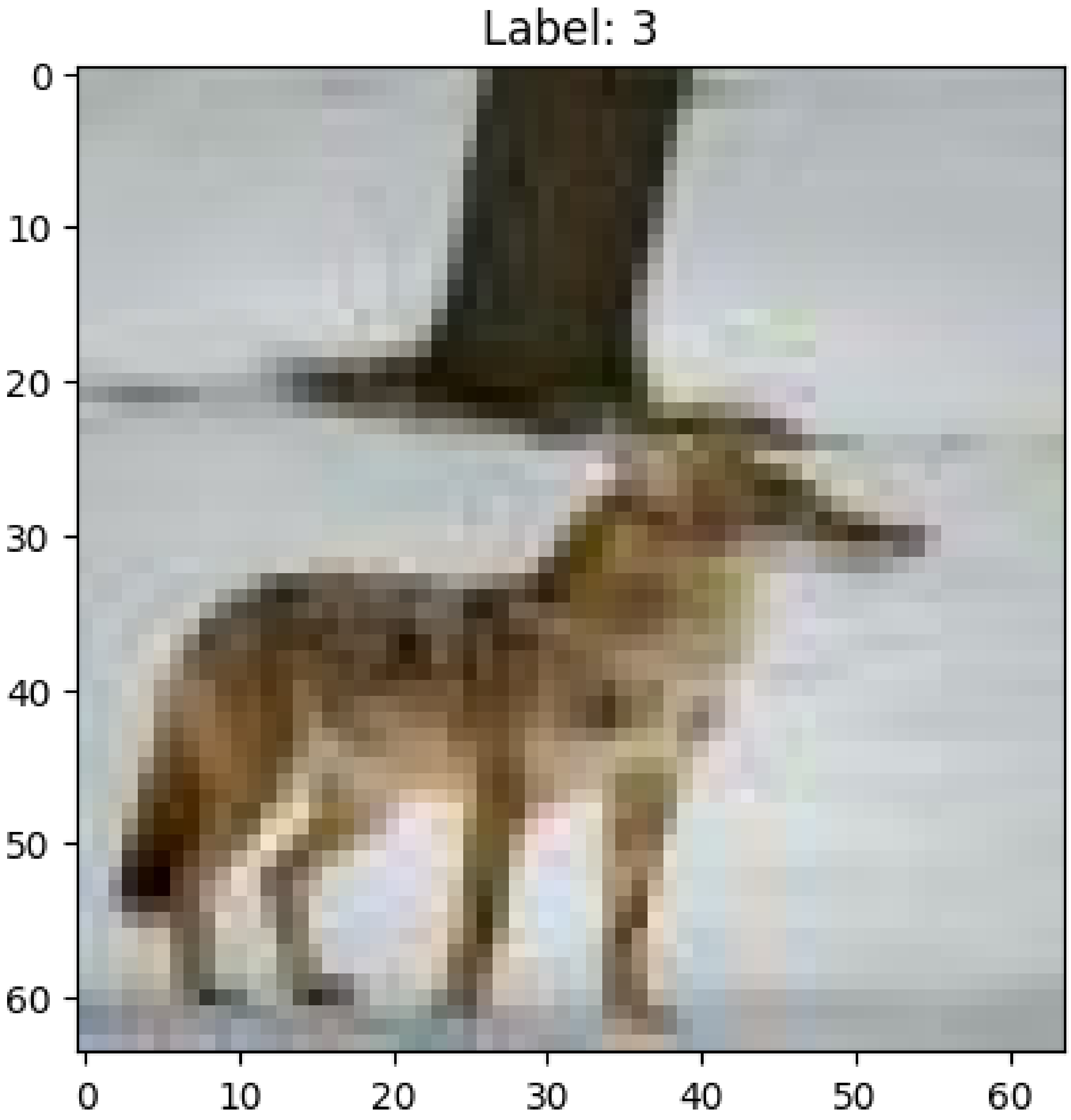

Figure 7: Image

Table 3: Configuration Summary of `LinearModel`

Component	**Details**
Input Shape	$(B, 3, 64, 64)$ (batch of RGB images)
Flattened Input	$(B, 12288)$
Output Shape	$(B, 10)$
Linear Layer	`nn.Linear(12288, 10)`
Activation	None (handled by loss function externally)

  – Conv ($32 \to 64$), ReLU, MaxPool $\to 64 \times 16 \times 16$
  – Conv ($64 \to 128$), ReLU, MaxPool $\to 128 \times 8 \times 8$

- **Fully Connected Layers:**
  – Flatten $\to$ Linear ($8192 \to 256$), ReLU, Dropout(0.5)
  – Linear ($256 \to 10$) for 10-class classification

3.4 Tuning process

I used the two different types of tuning methods. The changes of the first tuned model can be seen in detail from the table 4.

Table 4: Comparison of original CNN and tuned CNN architecture(v1)

Components	**Simple**	**Tuned**	**Reasons**
Activation Function	`ReLU()`	LeakyReLU()	LeakyReLU mitigates the *"dying ReLU"* problem by allowing a small gradient for negative inputs, improving gradient flow and convergence.
Batch Normalization	✗	BatchNorm2d(64)	BatchNorm normalizes feature maps, accelerating training and reducing internal covariate shift. It improves model stability and generalization.

The following table 5 shows the improvement compared with the original model (Vanilla CNN). But actually, our model is constructed on the tuned_v1) model.

Table 5: Comparison of Original CNN and Enhanced CNN Architecture

Component	**Simple**	**Tuned**	**Reasons**
Conv Layer Sizes	$32 \to 64 \to 128$	$64 \to 128 \to 256$	More filters capture more abstract and complex features, improving representational power.
Activation	`ReLU()`	`LeakyReLU()`	LeakyReLU avoids the dying ReLU problem and improves gradient flow, especially in deeper networks.
BatchNorm2d	✗	✓After each conv layer	BatchNorm stabilizes learning by normalizing feature maps and speeds up convergence.
Fully Connected Size	$8192 \to 256 \to 10$	$16384 \to 512 \to 10$	Increased size complements deeper conv layers, allowing for richer representation and learning capacity.
BatchNorm1d	✗	✓Used after FC	Helps normalize dense features, further improving training stability.

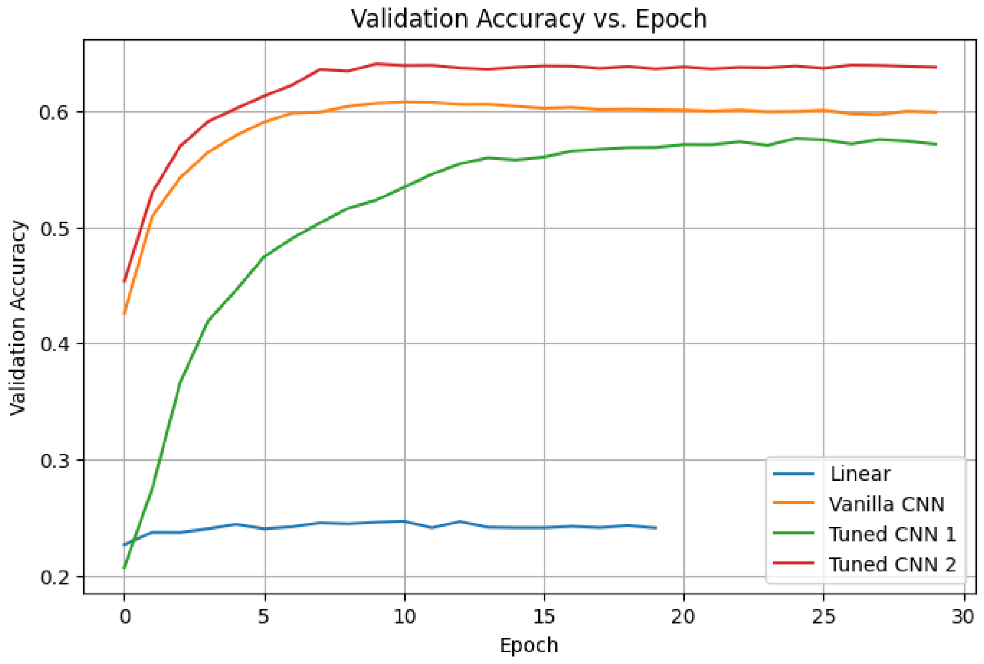

Figure 8: Validation Accuracy vs Epochs

Accuracy Comparison Figure 8 compares the validation accuracy of four models: a linear model, a vanilla CNN, and two tuned CNN architectures.

- **Linear Model**: Achieves minimal accuracy (0.23), indicating poor capacity for image classification. Linear regression is not fit for this problem. I used fewer training epochs and stopped early, as the performance showed no improvement and remained fluctuating around 0.23.
- **Vanilla CNN**: Rapid accuracy increase early on, plateauing near 0.60. Demonstrates strong baseline performance from a basic convolutional structure.
- **Tuned CNN 1**: Incorporates `BatchNorm2d` and replaces `ReLU` with `LeakyReLU`. However, the validation accuracy stagnates below that of the vanilla CNN, suggesting that while these modifications enhance training stability, they do not necessarily boost generalization when network capacity is limited.
- **Tuned CNN 2**: Builds upon Tuned CNN 1 by increasing the number of filters in each convolutional layer and expanding the fully connected layer. This model outperforms all others, reaching validation accuracy around 0.64, confirming that increased model capacity (depth and width) leads to better generalization.

In conclusion, we found that adding BatchNorm and changing the activation functions did not significantly improve model performance. In contrast, increasing model depth and capacity led to better results. However, deeper architectures were not explored due to computational limitations, as training was conducted on a CPU and proved too time-consuming for more complex configurations.

Loss Comparison Figure 9 compares the validation loss of four models: a linear model, a vanilla CNN, and two tuned CNN architectures. We can get the following results:

- **Vanilla CNN**, **Tuned CNN 1**, and **Tuned CNN 2** all achieve significantly lower validation loss than the **Linear model**, indicating better learning capacity for the task.
- The **Linear model** shows consistently high and flat validation loss (∼2.6–2.8) across epochs, suggesting underfitting and limited modeling power.
- **Vanilla CNN** and **Tuned CNN 2** both exhibit overfitting: their validation loss initially decreases but then steadily increases after a few epochs.

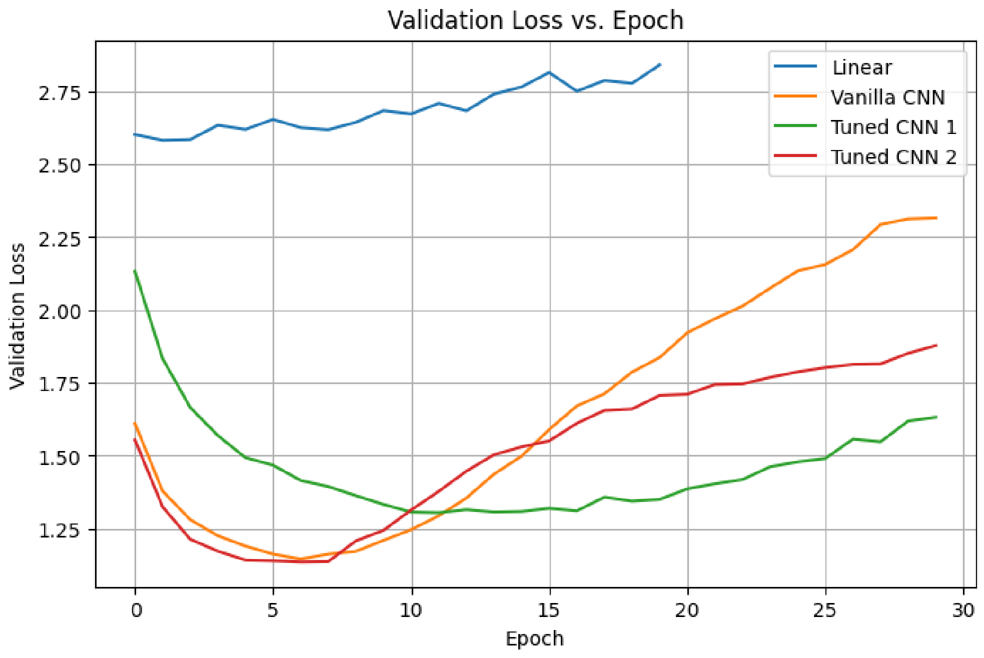

Figure 9: Validation Loss vs Epochs

- **Tuned CNN 1** achieves the lowest and most stable validation loss, indicating effective hyperparameter tuning and better generalization performance.
- Overfitting is a noticeable issue for the non-tuned and partially tuned CNN models, emphasizing the importance of regularization and validation monitoring.

3.5 Smooth Gradient

I ran my SmoothGrad on my model selecting the image that has the ”label=3” on it. I selected the .pt file from epoch 28 of the Vanilla CNN model, as the model had converged by then and achieved stable, high accuracy. The figure 10 shows the result.

3.6 LIME

I also use LIME on the same image and drew the plot in figure 11. Compared with the features selected by smooth grad method, we found the following things:

- They both wrongly capture the trunk in the background.
- They both capture the tail, four legs and back of this wolf.
- LIME also highlights the wolf’s mouth and face, while SmoothGrad captures more noise, such as the bright spot in the middle left corner.

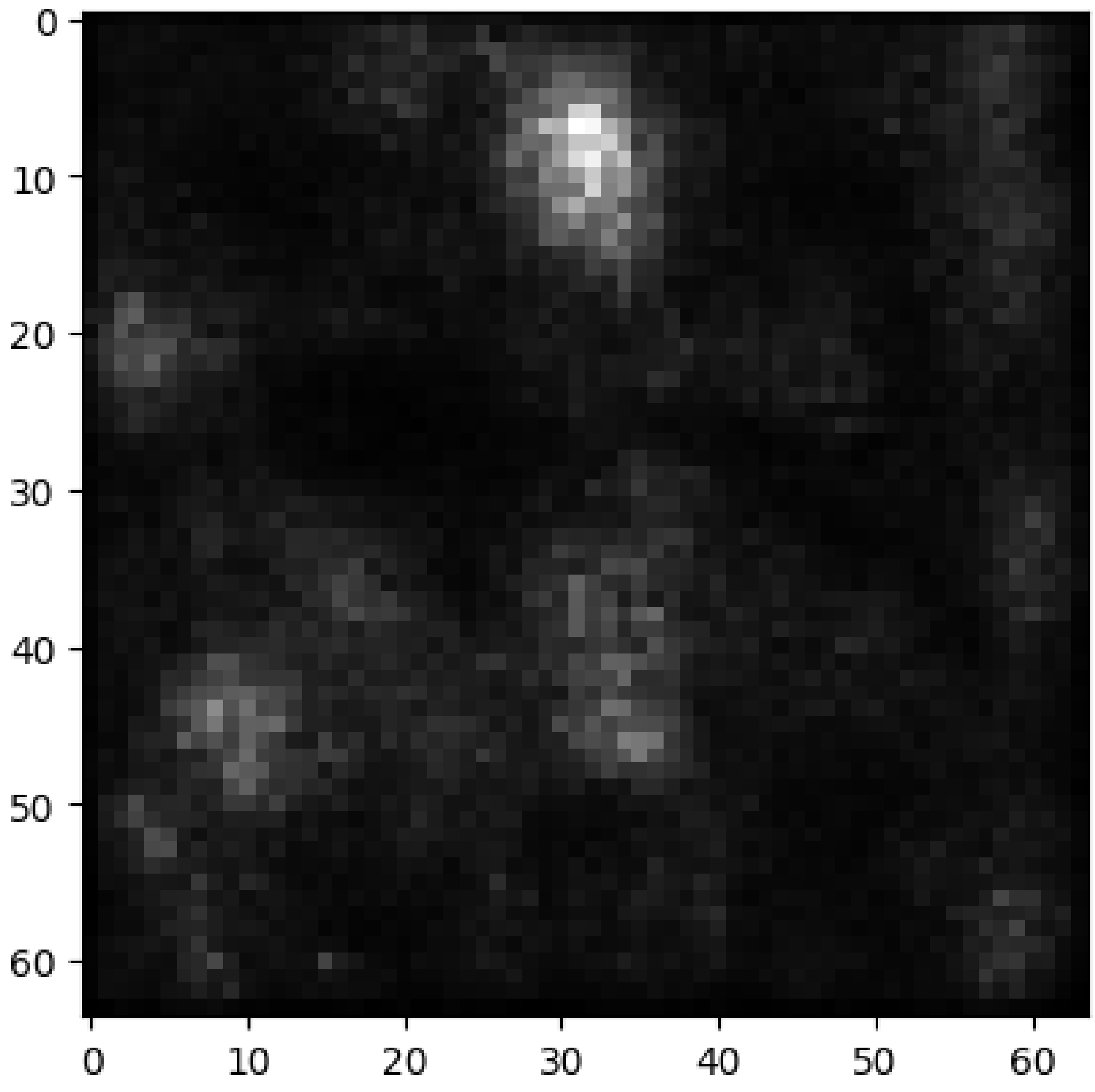

Figure 10: Smooth Gradient Plot

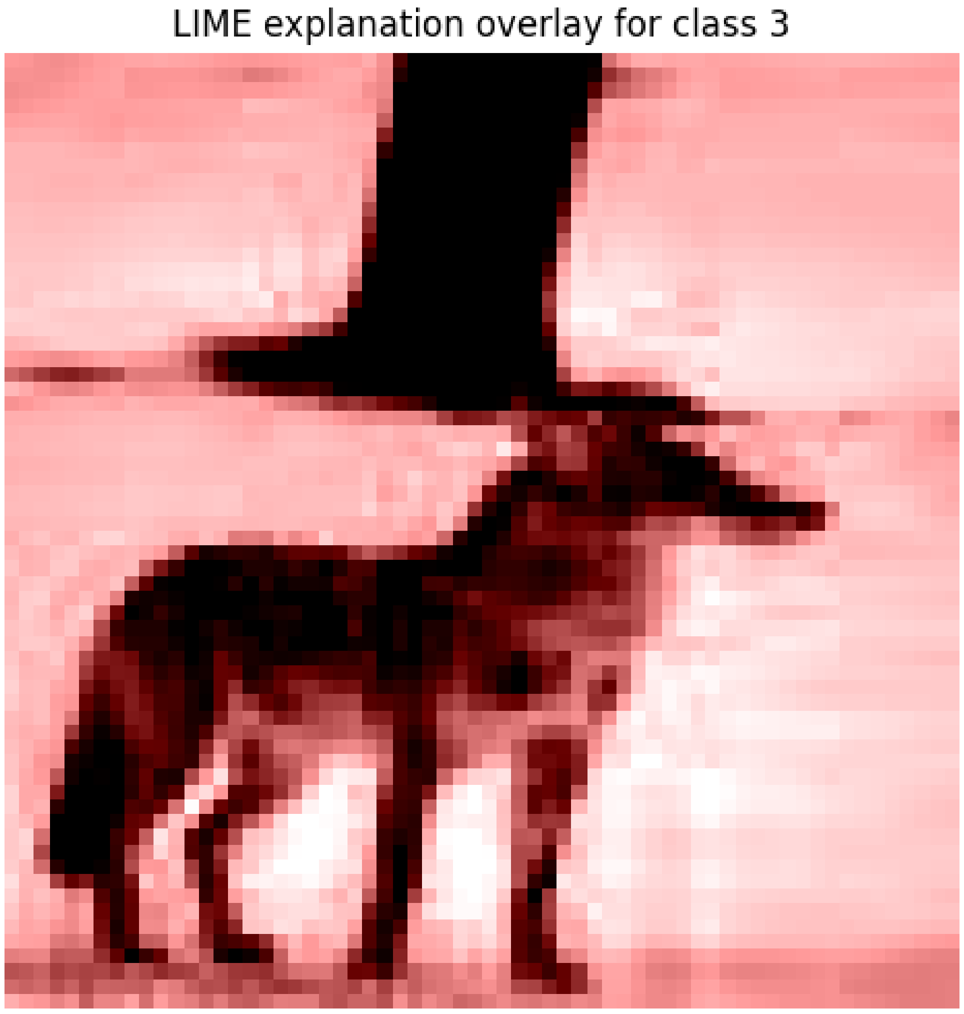

Figure 11: Lime for animals