Explainability

Hongli Peng

May 2023

1 Introduction

In this project, we focused on the topic of **Explainability**. We would discuss this in three main sections

1. **Tabular Data** applying the Lime method with the dataset Pima Indians Diabetes Database
2. **Predictive Modeling on Animal Images**
3. **Feature Attribution on Animal Images**

Sections 2 and 3 utilized the dataset **'Animal-10N Image Classification Dataset'**. We adopted the linear model, CNN, and fine-tuned CNN models for predicting the animal images and compared the models' loss and accuracy. We also explored the use of **SmoothGrad** and **Lime Image** for the explainability for animal images.

2 Tabular Data

We used 'Pima Indians Diabetes Database', and the goal was to predict if a patient has diabetes. The outcome variable is **'Outcome'** and the shape is (768, 9). There are 8 variables that could be used to predict 'Outcome' that whether or not a patient had diabetes. Since there were no missing values and obvious outliers, we reserved all data.

The percentage of the total outcomes have the patient as diabetic was 34.895833%. The outcome variable of the patient was being diabetic under-sampled in this case. But for predicting if one was diabetic or not, this imbalance case was acceptable. And we split the dataset into a train and a test set with the proportion 8:2, and the random state was 42.

2.1 Logistic Model with an L1 Penalty

We applied a logistic regression model with an L1 penalty to fit the data. From table 1, we could see the coefficients of 8 variables and compare their importance with absolute importance. The first 4 coefficients were Glucose, BMI, Age, and BloodPressure.

The sign of each coefficient indicated the direction of the relationship with the outcome. A positive sign indicated that as the feature value increases, the log odds of the outcome being diabetic increase, while a negative sign indicated the opposite

Lasso regression or L1 penalty had the property of driving the coefficients of less important features to exactly zero, effectively performing feature selection. Therefore, we should only select the features that return non-zero coefficients. 'SkinThickness' had the smallest coefficient 0.026896.

We chose the solver 'liblinear' that used the logistic loss function (the cross-entropy loss) to optimize the parameter because it is the only solver that can handle L1 lasso regularization.

Table 1: Feature Importance for Logistic Regression Model

Feature_LM	Importance_LM	Abs Importance_LM
Glucose	1.060	1.060
BMI	0.785	0.785
Age	0.421	0.421
BloodPressure	-0.242	0.242
DiabetesPedigreeFunction	0.218	0.218
Pregnancies	0.209	0.209
Insulin	-0.183	0.183
SkinThickness	0.027	0.027

2.2 Random Forest for Feature Selection

We used a random forest classifier to perform feature selection. From Table 2, 'Glucose' had the highest importance 0.258864 and 'SkinThickness' had the lowerest importance 0.065646, which was consistent with the previous result using a logistic regressor with L1 penalty.

Feature_RFC	Importance_RFC
Glucose	0.2589
BMI	0.1700
Age	0.1409
DiabetesPedigreeFunction	0.1238
BloodPressure	0.0881
Pregnancies	0.0766
Insulin	0.0761
SkinThickness	0.0656

Table 2: Feature importance from Random Forest Classifier

3 Lime Explainability for the Importance

By utilizing **LIME** (Local Interpretable Model-agnostic Explanations), we can identify the importance of each feature in the selected datapoint's prediction of being diabetic or not. The num_features in **LimeTabularExplainer** were the top k significant features in the explanation generated by LIME. We chose all the features which were 8 in this case. For the first instance in the test set, the predicted probability of being Diabetes is 0.31. All the variables have a positive influence except the variables 'Glucose' and 'Pregnancies'. The importance of features were stored in Table 3.

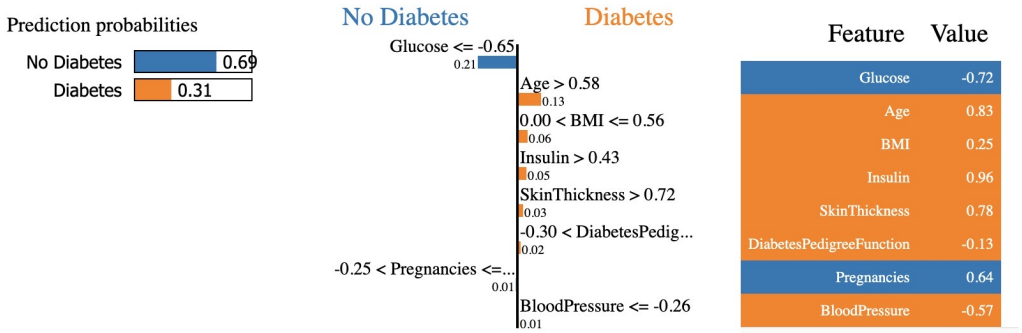


Figure 1: Lime for Random Forest in 1st Instance

Furthermore, we compared the stability across different instances. From the 5th instance that predicted diabetes with 0.54 probability and the variable 'Age', 'BMI', 'Insulin' and 'SkinThickness'had

Table 3: Feature Importance from LIME in 1st Instance

Feature	Importance
Glucose	-0.2411
Age	0.1281
BMI	0.0624
Insulin	0.0554
SkinThickness	0.0489
DiabetesPedigreeFunction	0.0308
Pregnancies	-0.0086
BloodPressure	0.0077

a positive influence on predicting diabetes, and others hold the opposite influence. Even though the probability of predicting diabetes and the importance of features could change across different data points, there was a trend that 'Glucose', 'Age', and 'BMI' were selected as more important features. We also compared the 19th and other instances, these features were also selected first. Hence, it was fair to say that this LIME was stable across different data points.

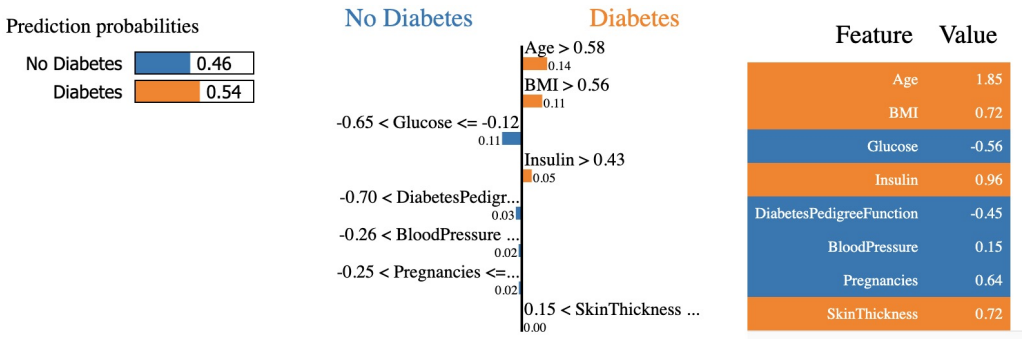


Figure 2: Lime for Random Forest in 5st Instance

By comparing the importance among LIME, Forest, and Linear models across the data points we examined above, 'Glucose', 'Age', and 'BMI' all had the highest importance than other features and 'SkinThickness' had the lowest importance. We could conclude that in our example, the three models selected similar features across different data points.

4 Predictive Modeling on Animal Images

The dataset Animal-10N Image Classification Dataset was used in this section. the ANIMAL-10N dataset consisted of ten classes of animals, comprising 50,000 training images and 5,000 testing images. We split the data into train and test with the proportion 9:1 since we had a large dataset. We adopted the **data_loader.py** by filling in the missing codes. The id_bytes = 4, label_bytes = 4, num_train_files = 1, num_train_images = 50000, width = 64, height = 64, depth = 3, num_classes = 10. We operated the code with **GPU in Kaggle**. The example images of each class were shown in Figure 3.

4.1 Linear Model on the Animal Classification

We adopted PyTorch to run a linear model. We normalized the data image and used the **cuba** device to apply GPU. We chose the epochs = 10, batch_size = 128, learning_rate = 0.001, weight_decay = 0.0002, Adam optimizer and cross-entropy loss function. We added **weight_decay** which was L2 regularization and a technique used to prevent overfitting by adding a penalty term to the loss function. We found that this parameter helped improve the accuracy. The final accuracy for the linear model was 24.2%. Figure 4 showed the comparison between train cross-entropy loss and test cross-entropy

Figure 3: Example Images of Animal-10N data

loss, and the comparison between train accuracy and test accuracy. The accuracy was stable after epoch 5.

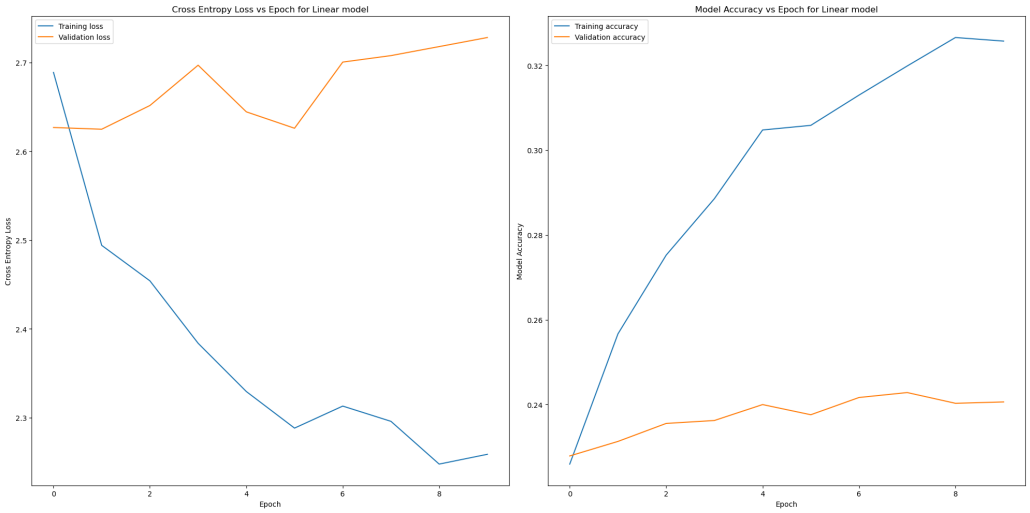


Figure 4: Linear Model's Performance

4.2 Vanilla Deep Convolutions Network

We built a vanilla CNN in this section. We constructed two convolutional layers, each followed by a ReLU activation and max-pooling, and a final fully-connected layer for classification. From Figure 5, we could examine the model performance. The best model was around epoch 8 where the test accuracy was the highest and test loss was the lowerest. The best accuracy is 50.9%.

We tried some methods to improve the model performance like adding more layers, the number of inputs in each layer, the dropout method, and changing the activation functions. The 'Tuned-CNN_drop' model was composed of four convolutional layers interspersed with batch normalization, Leaky ReLU activation that addressed the "dying ReLU" problem, max pooling, and dropout for regularization. After the convolutions, the feature maps are flattened and passed through a fully connected layer with a dropout and another Leaky ReLU activation. We compared the difference between with and without dropout methods in hyperparameter tuning. From Figure 5 and Figure 6, both models performed better than the initial vanilla CNN, and the model with dropping out had a better performance than without dropping out with a lower loss and higher accuracy. The best test accuracy for Tuned VanillaCNN with the dropout method was 67.36% around epoch 15. For this model, epochs = 30, batch_size = 128, learning_rate = 0.001 and weight_decay = 0.0002.

In conclusion, the hyper-parameter tuning was successful to improve the model performance. The

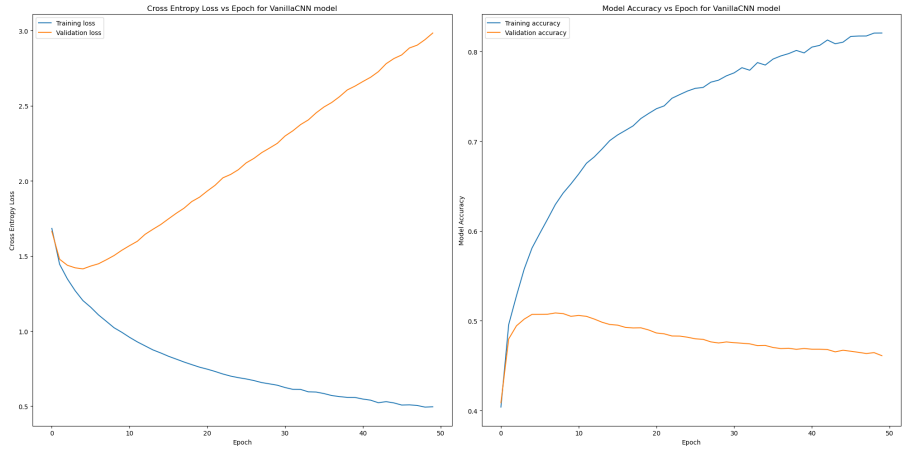


Figure 5: VanillaCNN Model's Performance

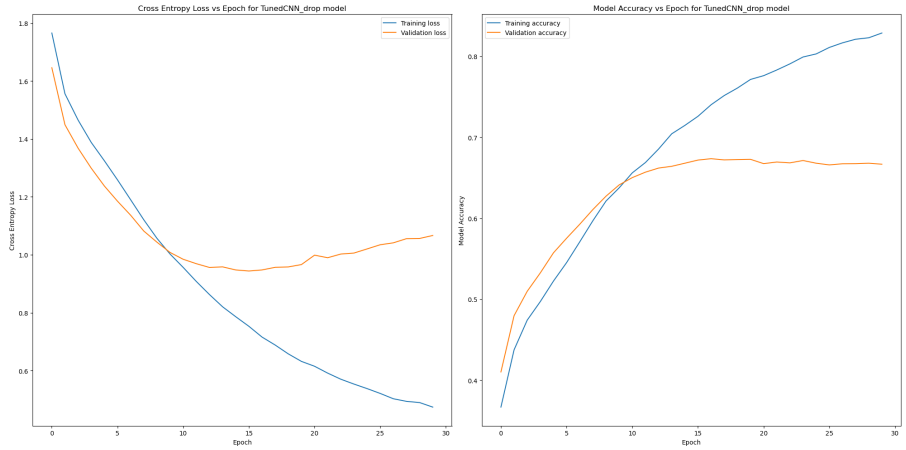


Figure 6: Tuned VanillaCNN Model's Performance with Dropping Out

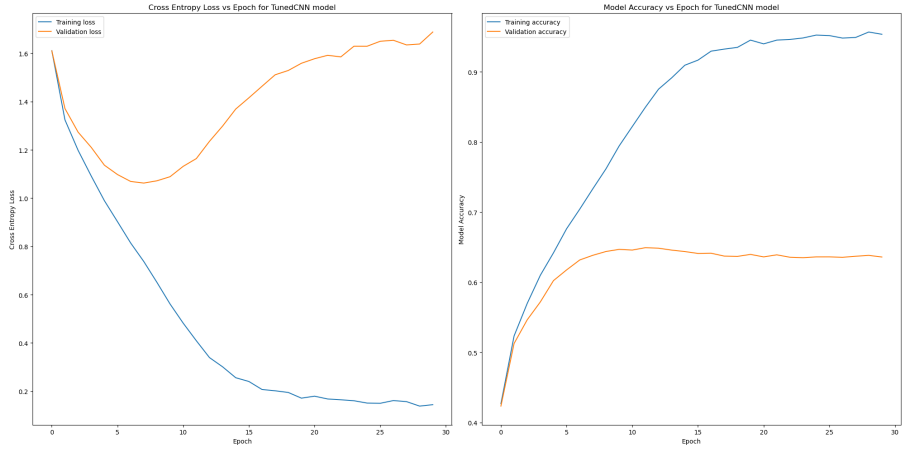


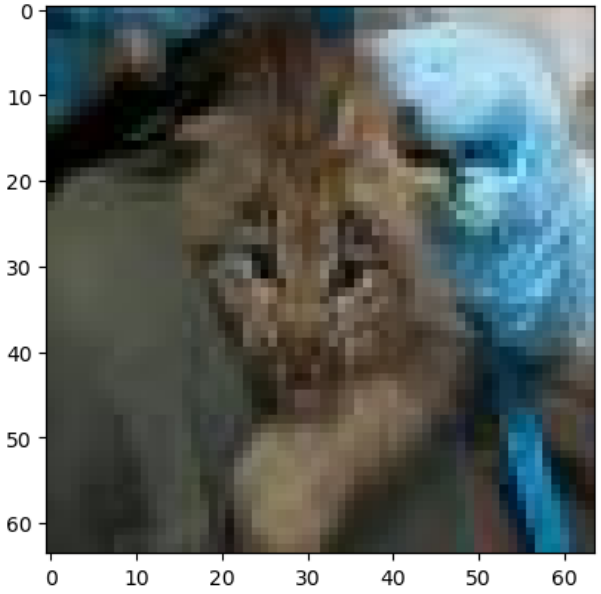
Figure 7: Tuned VanillaCNN Model's Performance without Dropping Out

tuned vanilla CNN with the dropout method performed the best among these models due to the lowerest test loss and the highest accuracy. And I found adding more layers and adding the dropout method were the more important parameters to increase the model performance.

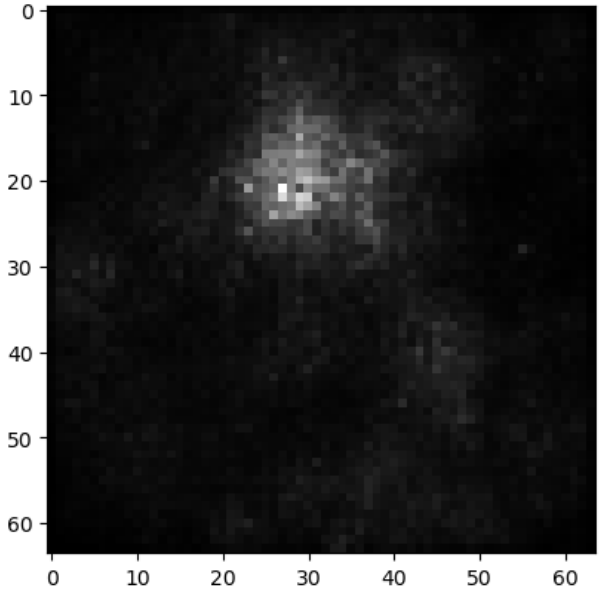
5 Feature Attribution on Animal Images

In this section, we discussed **SmoothGrad** and **Lime images** explainability for the tuned vanilla CNN with the dropout method.

Firstly, we ran SmoothGrad on the tuned vanilla CNN model and examined the model performance by plotting a similar curve as above. Also, we showed a heatmap of the gradient on the image demonstrating what pixels SmoothGrad was selecting. What's more, we plotted the graph in different noise levels 0, 0.05, 0.1, 0.2, 0.3, and 0.5 for better visualization from Figure 9.



(a) Original Image for 817th Instance



(b) Heatmap for 817 Instance

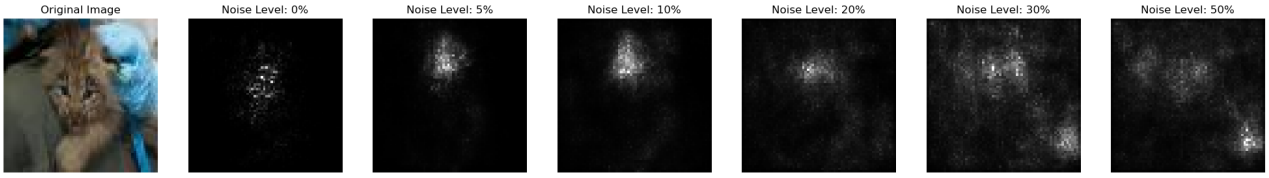


Figure 9: Original Images with Different Noise Levels

Secondly, we used LIME on images to generate an explanation of what features the tuned vanilla CNN model was selecting and compared the features selected on by LIME and SmoothGrad. From Figure 11, we plotted two wolves and one Lynx, and one cat. Both LIME and SmoothGrad give similar features selection. The shape and the head of these animals were the more important common features.

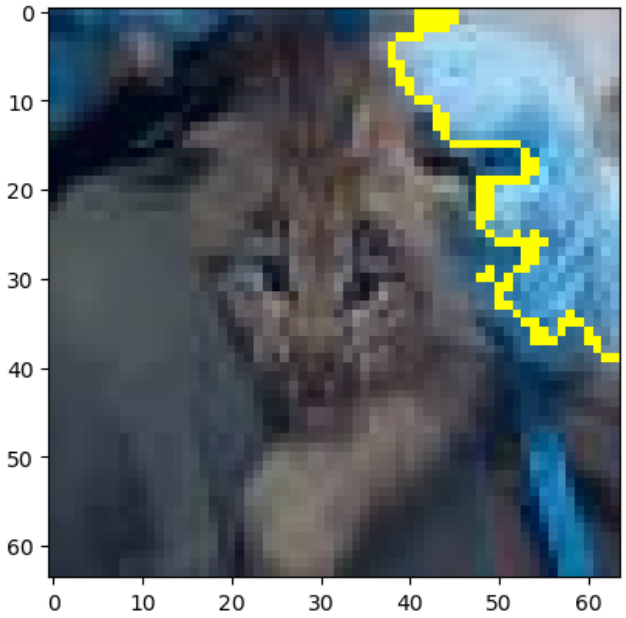
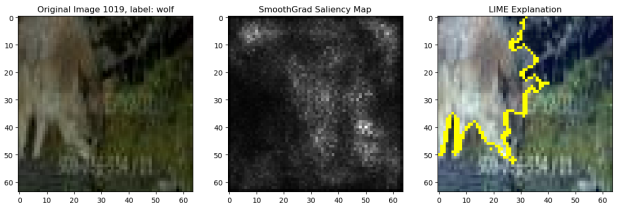
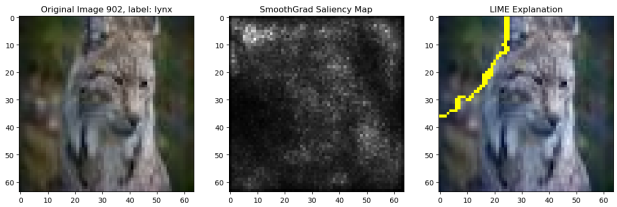


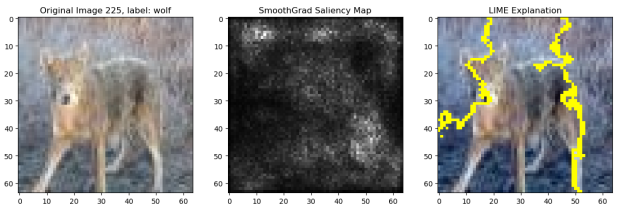
Figure 10: Lime Explanations for the 817th instance



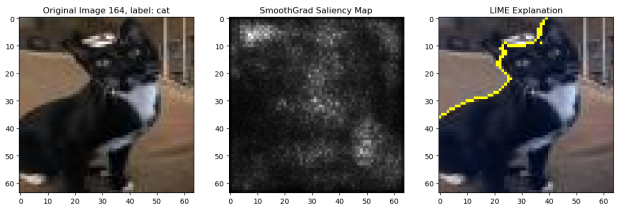
(a) Figure for Wolf



(b) Figure for Lynx



(c) Figure for another Wolf



(d) Figure for Cat

Figure 11: Graphs of Original images, SmoothGrad, and Lime images