# Comparing SHAP Values Based Methods to Interpret Machine Learning Models

**Implementation Track, Prathik Shirolkar, Vasilina Filonova, Zixian Yang**

## Abstract

In this project, we explore explanation methods based on SHAP values which allow to explain contributions of individual features to the overall prediction on a machine learning method. We implement three methods: SHAP values, Kernel SHAP, and Deep SHAP. We apply these methods to explain machine learning models such as linear regression, XGBoost, and neural network for tabular and image datasets. We verify first two methods implementation against 'shap' Python's library. Qualitatively, verification shows excellent agreement between outcomes. Quantitatively, there is a discrepancy with the SHAP values method results, possibly, caused by a difference in background treatment. The Deep SHAP is verified by satisfying theoretical properties of the results. For validation, we apply Deep SHAP to explain the neural network prediction of image classification. The results are excellent and greatly agree with human intuition. The time efficiency test confirms our expectations: the Kernel SHAP is faster than SHAP values. The Deep SHAP is the most time efficient for neural networks, therefore it is applied to image classification. The key learning points during implementations for us are: importance of background choice and model averaging strategies. Also, now we fully appreciate how powerful the SHAP values can be for revealing the individual feature impacts to the model prediction.

## 1 Motivation

Machine learning methods have demonstrated remarkable performances in many real-world applications such as computer vision, natural language processing, traffic prediction, etc. In many tasks, there is a need to use complex machine learning (ML) models with high performance such as ensemble and deep neural networks. For example, deep neural networks can significantly improve image classification compared to other ML models (e.g. AlexNet [4]). However, unlike simple models such as linear regression or decision trees, these 'black-boxes' lack providing the association between cause and effect. We know little about the underlying mechanism, i.e., how and why the model makes a certain decision. This becomes crucial when such methods are used to make decisions for high-stakes applications affecting individuals and social groups in real-life problems, e.g. in ML-driven medical decisions, autonomous driving, and social-policy changes. The effect can be negative and easily overlooked due to limitations in understanding the mechanisms of ML outcomes. Thus, there is a pressing need to explain (or interpret) ML models.

In recent years, there were significant efforts in ML research community to develop methods to explain ML models [3]. In artificial intelligence theory, one would distinguish interpretability and explanability[1]. Roughly, an algorithm is interpretable if its decision can be explained to a non-expert, without necessarily knowing 'why'. Explanability reveals details trying to answer why a particular decision was chosen. As mentioned in [6], correctly interpreting the predictions of a model can also provide insight into how a model may be improved. This also motivates the research in explanability and interpretability.

[1] In this report we use these terms interchangeably

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

To address the need for universal interpreting methods, in 2017 Lundberg and Lee proposed the unified approach for interpreting model predictions, called SHapley Additive exPlanations (SHAP) [6]. The novelty of the paper is in defining the class of additive feature attribution methods that unifies explanation methods that were established previously: LIME [7], DeepLIFT [9; 10], layer-wise relevance propagation [1], and classic Shapley value estimation methods [5; 11; 2]. The paper proposed SHAP values as well as Kernel SHAP, Linear SHAP, Low-order SHAP, Max SHAP, and Deep SHAP for approximating SHAP values. This leads to improvements of the existing methods because the SHAP values provide the unique solution for feature contributions that satisfy the desirable properties of local accuracy, missingness, and consistency. Noteworthy, the Kernel SHAP is an approximation of SHAP values and reduces the computational complexity. Deep SHAP, designed specifically for deep networks, has even lower computational complexity than Kernel SHAP.

For our project, we proposed to implement the SHAP methods from [6]. There were changes to the plan we outlined in the proposal, that can be seen in Appendix A. The main contributions of our final plan of the project are the following:

- We implement three interpreting methods from scratch: SHAP values, Kernel SHAP, and Deep SHAP. We verified the implementation comparing the results against 'shap' Python's package.
- We report, analyze and visualize our results.
- We use tabular and images datasets not used in the original paper.
- We explain the SHAP methods from the implementation standpoint.
- Finally, we obtain impressive results for neural network explanations using Deep SHAP.

## 2 Problem statement

Imagine there is the ML model $f$ that you want to explain in terms of predictability contribution of individual features. The inspiration for solving such problem was found in the game theory where Shapley values were proposed to measure individual contributions in a coalition [8]. This approach was generalized in the paper [6] that introduces the concept of *additive feature attribution methods* and proposes SHAP values as an optimal measure that satisfies the desirable properties.

To explain the original model $f(x)$ based on a single input (row of feature values) $x$, the explanation model $g(z')$ is introduced as a linear additive function:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i \simeq f(h_x(z'))$$

for $z' \approx x'$, where $x'$ are simplified inputs that map the original input via $x = h_x(x')$ with $h_x$ being specific to input $x$. An example of such map $h_x$ can be a 0-1 mask containing all combinations of features inclusion-exclusion (or missingness), respectively. Here, $M$ is the number of (simplified) input features, so $z' \in \{0,1\}^M$. The affect of $i$-th feature is measured by $\phi_i$ values, that can be represented by Shapley or SHAP values.

To solve the stated problem, we implement three methods: 1) SHAP values, 2) Kernel SHAP, and 3) Deep SHAP. These methods are based on other explanability methods: LIME, DeepLIFT, Shapley regression values, and Shapley sampling values. Note, that key insight of the paper [6] is that all these methods are members of the additive feature attribution methods, and that SHAP values provides the unique additive feature importance measure that satisfies all three desirable properties (local accuracy, missingness, and consistency). In the next section we describe SHAP, Kernel SHAP, and Deep SHAP in more details.

### 2.1 Related work

**Shapley regression/sampling values**: Shapley regression values are feature importances for linear models in the presence of multicollinearity. The values are calculated by taking the difference between predictions with a feature withheld, and feature included. Since withholding a feature depends on other features in the model, a weighted difference is calculated for all subsets of features. This approach requires retraining the model on all feature subsets which can be unfeasible task for real-life

applications. This complexity can be simplified by sampling from the training dataset (Shapley sampling values), thus having less number of differences computed and eliminating retraining the model.

**LIME**: LIME randomly generates additional local data points, by selecting appropriate $h_x(x')$. Predictions $y$ are then generated for these newly generated data $x$. A sparse simple linear model $g$ (surrogate model) is then used to regress $y$ on $x$. LIME minimises the objective function similar to what is stated in Kernel SHAP Eq. (5), which helps get good approximation of $f(x)$ using $g(x')$, while keeping $g(x')$ as simple as possible.

**DeepLIFT**: DeepLIFT leverages compositional nature of deep neural networks to explain prediction. Suppose we have a target neuron $t$ with a difference from some reference as $\Delta_t$. Consider some layer (may even be the input layer), which have neurons $x_1, x_2, ..., x_n$. Then DeepLIFT helps to decompose contribution of $\Delta_t$ in terms of $\Delta x_1, \Delta x_2, ..., \Delta x_n$, using backward propagation.

# 3 Methods

## 3.1 SHAP values

The SHAP values are introduced as unified measure for feature importance and defined via:

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'|)!}{M!}[f_x(z') - f_x(z' \backslash i)] \tag{1}$$

$$f_x(z') = f(h_x(z')) = E[f(z)|z_S]) = E_{z_{\bar{S}}|z_S}[f(z)] \tag{2}$$

$$\approx E_{z_{\bar{S}}}[f(z)] \text{ (for independent features)} \tag{3}$$

$$\approx f([z_S, E[z_{\bar{S}}]] \text{ (for linear models)} \tag{4}$$

where $|z'|$ is the number of nonzero entries in $z'$ and $z' \subseteq x'$ are vectors where nonzero vectors are a subset of the nonzero entries in $x'$. Here, $z' \backslash i$ denotes setting $z'_i = 0$ and $S$ is the set of nonzero indexes in $z'$. The input mapping is $h_x(z') = z_S$ and $z_S$ has missing values for feature not in set $S$.

The SHAP values method connects Shapley regression (or sampling) values [5] and LIME [7]. By introducing (2)-(4), SHAP values are estimated via a conditional expectation function of the original model. It allows using Shapley regression values without retraining the model multiple times. Moreover, such SHAP values provide the unique additive feature importance measure that satisfies all three desirable properties proposed in [6]. However, using SHAP values to explain model with large number of parameters, can be unfeasible. For instance, 64-features model leads to billions of subsets where regression problem has to be solved. To reduce subsets space, the paper [6] describes an alternative called Kernel SHAP, which uses Shapley sampling values [11] and assumes that features are independent.

## 3.2 Kernel SHAP

The Kernel SHAP method uses the LIME framework [7] to compute Shapley Values. It proposes to solve the following minimization problem to find $\phi$:

$$\xi = \text{argmin}_{g \in G} \sum_{z' \in Z} [f(h_x(z')) - g(z')]^2 \pi_{x'}(z') \tag{5}$$

with the local weighting kernel:

$$\pi_{x'}(z') = \frac{(M-1)}{(\text{M choose } |z'|)|z'|(M - |z'|)}. \tag{6}$$

Here the SHAP values are approximated assuming linear independence of features (3):$f(h_x(z')) \approx E_{z_{\bar{S}}}[f(z)]$. The method proposes to create a new dataset by excluding one feature at a time and filling it with background data, then take the mean of their predictions as the final prediction for that coalition. With a specific weighting of each sample and a linear regression fit on data, the model prediction coefficients represent the Shapley value approximation [11]. This approach makes Kernel SHAP being much faster than SHAP values and thus a popular explanation method that can be applied to any ML model. For deep neural network explanations, the Deep SHAP method is more efficient than Kernel SHAP.

### 3.3 Deep SHAP

The Deep SHAP method is an approximation of SHAP values for deep neural networks. It combines DeepLIFT [9; 10] and SHAP values, leveraging compositional nature of deep neural networks to improve computational performance of Kernel SHAP. Suppose we have a target neuron $t$, with a difference from some reference as $\Delta_t$. Consider some layer (can be the input layer), which have neurons $x_1, x_2, ..., x_n$. Then DeepLIFT helps to decompose contribution of $\Delta_t$ in terms of $\Delta x_1, \Delta x_2, ..., \Delta x_n$, using backward propagation without calculating gradients. Instead of using a single reference value, DeepLIFT can be made to estimate SHAP values, if we have a distribution of background sample references and assume features independence.

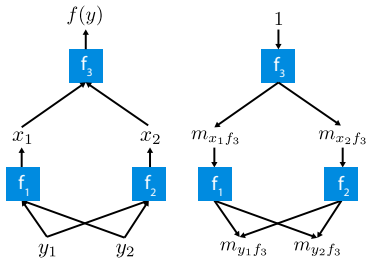



Figure 1: Deep SHAP scheme [6].

In Deep SHAP, we first analytically obtain the SHAP values for simple network components and then pass the SHAP values backwards through the network as shown in Figure 1.There $m_{x_i f_j}$ and $m_{y_i f_j}$ are the multipliers defined in terms of the SHAP values. The back-propagation rules are based on DeepLIFT [10], shown as follows:

$$m_{x_j f_3} = \frac{\phi_j(f_3, x)}{x_j - \mathbb{E}[x_j]}, \quad m_{y_i f_j} = \frac{\phi_i(f_j, y)}{y_i - \mathbb{E}[y_i]}, \quad \forall j \in \{1, 2\}, \ \forall i \in \{1, 2\} \tag{7}$$

$$m_{y_i f_3} = \sum_{j=1}^{2} m_{y_i f_j} m_{x_j f_3}, \quad \phi_i(f_3, y) = m_{y_i f_3}(y_i - \mathbb{E}[y_i]), \ \forall i \in \{1, 2\} \tag{8}$$

where $\phi$ are SHAP values defined in (1).

## 4 Results and evaluation

We implemented the SHAP values, Kernel SHAP, and Deep SHAP methods from scratch (we use some standard Python's library packages, but not 'shap' that is used for verification). We applied the methods to tabular and image datasets. The tabular data is the House Pricing Dataset from Kaggle[2]. The image datasets are MNIST[3] (handwritten digits images) and Fashion-MNIST[4] (cloth images). To evaluate methods implementations we performed verification and analysis, specific to each method.

### 4.1 SHAP values results

For the implementation, we consider the given dataset $X$ with $n$ number of rows and $M$ features, with input (row) $x$ and original model prediction $f(x)$. To compute SHAP values we need to compute $f(h_x(z'))$ for all combinations of features contributions. Also, for each $i$-th feature we need to compute $f(h_x(z' \backslash i))$ where in each combination we substitute this feature by a specifically designed background. The implementation includes several steps, outlined in Appendix (B).

One of the challenges of implementing SHAP values is a generation of a background that mimics the 'exclusion' (missingness or withholding) of the specific feature from the model. Note that there is no actual exclusion of a feature from the dataset. Instead, the feature is replaced by a background. This approach is simpler than computing actual exclusion in Shapley regression, but makes SHAP

[2]https://www.kaggle.com/akash14/house-price-dataset

[3]http://yann.lecun.com/exdb/mnist/

[4]https://www.kaggle.com/zalando-research/fashionmnist

more sensitive to a choice of a background. That said, there is a flexibility of creating background, and SHAP explanations should be understood as values with respect to a baseline background. For our implementation, we generated the dataset with a background using the same approach for both SHAP and Kernel SHAP values methods. Unlike in Kernel SHAP, we run original model prediction for each combination which makes SHAP values method more time consuming. For this reasons, we choose a linear regression model and tabular (housing) dataset with reduced number of features, $M = 5$ and $M = 15$.

Using SHAP values we explain housing price and compare results with those obtained by 'shap' Python's library. Figure 2 shows a water fall chart for SHAP values demonstrating the relative impact. Qualitatively, there is a good agreement between two implementations: the trend of features contributions is the same. However, there is a discrepancy in magnitude of SHAP values. We notice, that the error is accumulating with the number of features. It suggests that the discrepancy can be due to a difference in a background choice and, possibly, due to background duplicates originated by excluding features from masks. Given more time we would investigate these issues in more details. Note, that we already learned, that to reproduce 'shap' we should not incorporate correction in Eq. (1)[5], but use the weights given in the original paper.

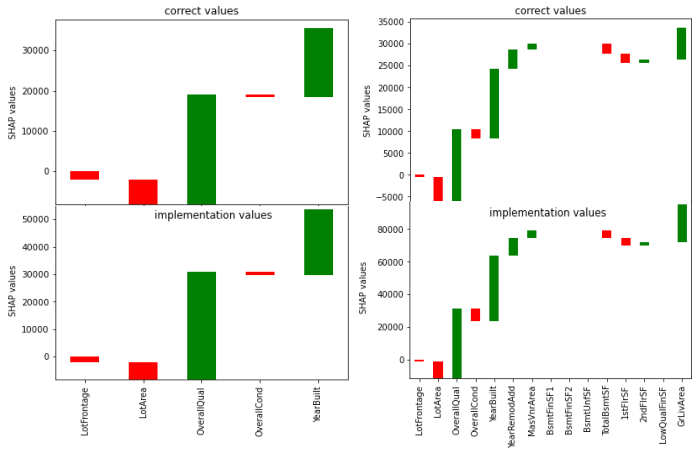



Figure 2: SHAP values results against 'shap' library: features contribution to the housing price for a model with 5 features (left) and 15 (features). Green bar - positive, red - negative contributions, shown relatively to the previous feature.

Computational time for computing SHAP values for 15-features model using our own implementation is 5 minutes by SHAP values method and 44 seconds by Kernel SHAP (run in the same laptop on the same test case). The time complexity difference occurs because SHAP values compare a feature contribution during missingness across all combinations, one feature at a time, while the Kernel SHAP process all features at the same time using the squared loss function.

### 4.2 Kernel SHAP results

We implement Kernel SHAP against two models: linear regression and nonlinear (XGBoost). As Kernel SHAP is model agnostic, it works against any given model. We use housing price tabular data for verification of the method correctness. For a given row of data to be explained, we first create a mask of all possible combination of feature exclusion and inclusion ($2^M$ combinations).As mentioned in the paper, we remove combinations where all features are included or excluded and left with $(2^M - 2)$ combinations. For combination components with 1, we include the feature values from the row that needs to be explained. For the combination components with 0 mask value, we include the background data. For every combination, we include multiple background data, one at a time, find their corresponding prediction using the model provided, and take expectation to get the final prediction for that combination. This way we compute $g(z')$, because the feature missingness is modelled as average effect of that feature over respective background data.

[5]In the original paper, the weight had -1 in the numerator $(M - |z'| - 1)!$, which should be removed as authors announced later in the SHAP GitHub repository

We aim to minimize the squared loss function in Eq. (5). We build a regression model with $y = g(z')$ and the input being the masks (combinations with 1's and 0's). We modify the original regression equation (5) by incorporating weight matrix $W$, Eq. (6), to solve the equation $(X^T W X)^{-1} X^T W y$. The obtained coefficients are essentially the SHAP values for Kernel SHAP.

Figure 3 demonstrates the verification results, in a water fall chart form, for Kernel SHAP obtained for two models and compared against 'shap' results. In both cases, we see the great agreement in trend and magnitude of importance of each feature. In addition we computed the percentage difference between SHAP values in our implementation vs 'shap' and obtained a range 0-5% (for linear regression) and 0-18% (for XGBoost). Some values discrepancy, though small, can be explained by the different background values used in 'shap' library, as compared to that of our implementation.

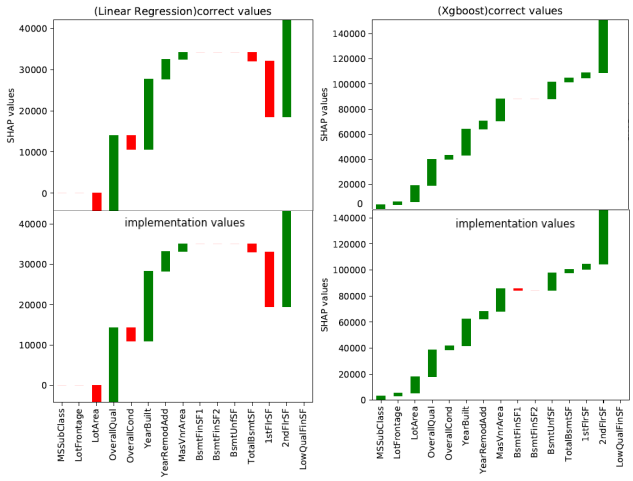



Figure 3: Kernel SHAP: Linear Regression (left) and XGBoost (right).

### 4.3 Deep SHAP results

We implement Deep SHAP for two datasets, MNIST and Fashion-MNIST. MNIST is a dataset with 28x28 grayscale images of handwritten digits. The task for MNIST is to classify these digits which are from 0 to 9. Fashion-MNIST is a dataset with 28x28 grayscale images of fashion products from 10 categories. Each of the two datasets contains 60,000 training images and 10,000 testing images. The task for Fashion-MNIST is to classify these images into the 10 categories. We implement Deep SHAP from scratch for different neural networks that may include fully connected layers, convolutional layers, Sigmoid and ReLU activation functions. We consider three different neural network (NN) models for the classification tasks, NN with Sigmoid, NN with ReLU, and Convolution NN (CNN) with ReLU. The main differences among the three models are whether convolutional layers are used and whether ReLU or Sigmoid activation functions are used. The details of the structures and training parameters can be found in Appendix C.

Though we intended to verify the Deep SHAP implementation against Python's library too, the latter turned out to be challenging to use (due to TensorFlow version issues). Instead, we verify satisfaction of SHAP values properties and elaborate on Deep SHAP results analysis. To evaluate the implementation we provide test accuracy. Also we evaluate Deep SHAP by looking at the heatmap of the SHAP values. For MNIST, the heatmap should have colors on the areas of the handwritten curves. For Fashion-MNIST, the heatmap should have colors on the contour of the fashion product. After that, we will remove 20% of the pixels with smallest SHAP values to change an image of one class to another class. For MNIST, the explanation is better if it is easier to differentiate the changed digit from the original one.

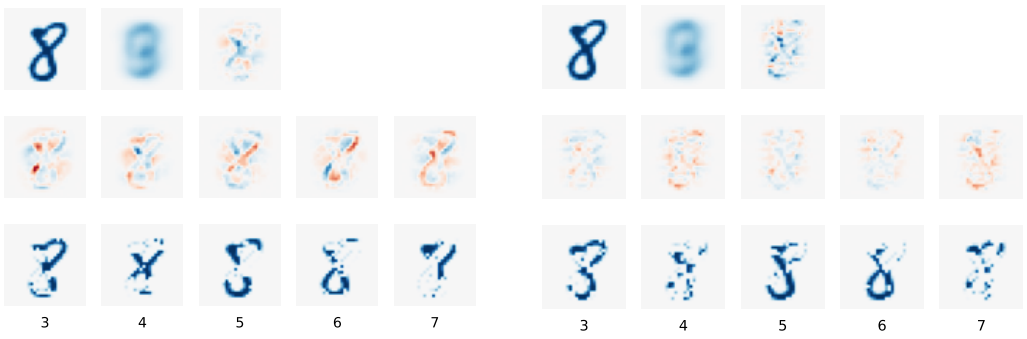
#### 4.3.1 Results: MNIST dataset

We first train three models for MNIST Dataset, NN with Sigmoid, NN with ReLU, and CNN with ReLU. The test accuracy of NN with Sigmoid is 0.9439. The test accuracy of NN with ReLU is

0.9797. The test accuracy of CNN with ReLU is 0.9927. The accuracy increases as the model becomes more complex.

Figure 4a shows the Deep SHAP results for interpreting the classification of a digit 8 in the test set of MNIST with NN with Sigmoid model. In the top row, the first two images are the original image and the background (average) image computed among 1000 test images. The third figure in the top row presents the SHAP values for class 8. Blue areas increase the probability of class 8, and orange areas decrease the probability. The blue areas clearly demonstrate the handwritten curve of digit 8, which means that these pixels have a large contribution to the prediction of class 8. The figures in the middle row are SHAP values for interpreting the contributions to class 3, 4, 5, 6, 7. Blue areas increase the probability of the corresponding class and orange areas decrease the probability. From the middle row of figures, we can see that the blue areas look like handwritten curve of the corresponding digit, and the orange areas are in critical positions that can differentiate the corresponding digit from digit 8. For example, in the first figure in the middle row, there is some deep orange areas which means that those pixels have negative contribution to the class 3. This is obviously true since digit 8 becomes digit 3 after removing those pixels. The images in the bottom row are the masked images after removing 20% pixels with smallest SHAP values from the original image of digit 8. We can see clear patterns of 3, 4, 5, 6, 7 in the masked images.

All of these results verify that Deep SHAP method provides interpretability and explanability for the neural network model. In our experiments, we also verified that the computed SHAP values satisfy the desirable properties of local accuracy and missingness mentioned in [6]. Specifically, for local accuracy, we verified that the summation of SHAP values of all pixels is equal to the difference between model output and the background value of the output. Also, for missingness, since we computed SHAP value by taking the product of the multiplier and the difference between input and the background value of the input, missingness is automatically satisfied.




(a) NN with Sigmoid.

(b) CNN with ReLU.

Figure 4: Deep SHAP: explaining the classification of an image of digit 8 in MNIST.

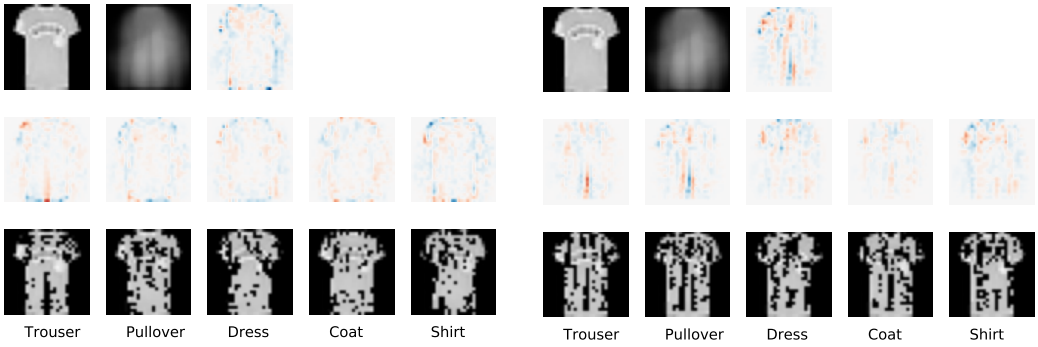
The results using NN with ReLU model are shown in Figure 6 in Appendix D. Figure 4b shows the results using CNN with ReLU model. We also have the same observations except that the pattern is less clear than Figure 4a. More complex models make the explanation and interpretation task more difficult.

#### 4.3.2 Results: Fashion-MNIST dataset

We first train two models for Fashion-MNIST Dataset, NN with ReLU, and CNN with ReLU. The test accuracy of NN with ReLU is 0.8882. The test accuracy of CNN with ReLU is 0.9145. The accuracy of CNN with ReLU is better than NN with ReLU.

Figure 5a shows the Deep SHAP results for interpreting the classification of a T-shirt image in the test set of Fashion-MNIST with NN with ReLU model. The setting of this experiment is the same as Section 4.3.1. In the top row, the blue areas clearly demonstrate the contour of the T-shirt, which means that the contour have a large contribution to the prediction. In the middle row, the patterns of SHAP values are less clear than those in the MNIST experiments. However, these heatmaps can still explain the prediction of the neural network. For example, there is some deep orange areas at the bottom of the first figure of the middle row. These areas reduce the probability that the T-shirt image

would be classified as a trouser because there is always some gap between two legs for trousers. In the bottom row, we can see that the original T-shirt image becomes similar to the corresponding class after removing 20% pixels that have smallest SHAP values, although it may not be as clear as the observation in Figure 4a-4b.




(a) NN with ReLU. (b) CNN with ReLU.

Figure 5: Deep SHAP: explaining the classification of T-shirt in Fashion-MNIST.

Figure 5b shows the results using CNN with ReLU model. In the third figure of the first row, we can see that there are some strip-like orange areas. We think that it is due to the imperfectness of our CNN classification model. There is room to improve the CNN model since these orange areas should not reduce the probability of being classified as a T-shirt. We also notice that there are more orange pixels in the middle row compared with Figure 5a, which explains why CNN model has a better accuracy since CNN can extract local features of an image. For the bottom row, the masked images are less differentiable than Figure 5a when we remove the pixels from the original T-shirt image. It is reasonable since unlike the digit classification task, for Fashion-MNIST task, not only the existence but also the value of an pixel matter, and thus sometimes removing pixels can not change one class to another.

## 5 Discussion and conclusion

In this project, we learned how to implement SHAP methods. We started with basic SHAP values, that proved to be time consuming and sensitive to background choice. We then implemented the Kernel SHAP that proved to be faster and also less sensitive to a background choice than SHAP values method. Finally, we successfully implemented Deep SHAP that was fast enough to run neural network explanations in a laptop and greatly demonstrated SHAP approach on images.

The Kernel SHAP results agree well with the results obtained by 'shap' Python's library. The outcomes for the SHAP values method agrees well with 'shap' only qualitatively giving a discrepancy in magnitude scale which we assume was caused by background specifications. We know little about how background was implemented in 'shap' in [6]. This is a valuable learning point for us for future investigations.

The Deep SHAP was not verified against the 'shap' library as the library function was not working. However, we provided other type of testing and verification of Deep SHAP such as checking properties that SHAP values should satisfy. As the Deep SHAP method was most challenging to implement, we share some additional learning points. Our implementation of Deep SHAP is slightly different from [6]. We first implemented DeepLIFT [10] since the back-propagation rules of Deep SHAP are based on it. Then we added background samples and replaced the multipliers with SHAP values of the components in the network. We found that first taking average of background samples and then passing the average value forward through the network gave us a clearer pattern in the heatmaps of Figure 4a-4b. However, in [6], the order of averaging was opposite, which gives less clear heatmaps in our implementation. We conjecture that the inferior performance can be due to the high variance and some outliers in the background samples.

## Contribution by group members

All authors were contributing equally to the implementation by discussing challenges and taking over parts of the projects. The individual contributions of methods implementations were: Author 1 - Kernel SHAP, Author 2 - SHAP values, Author 3 - Deep SHAP. Author 1 also implemented DeepLIFT for fully connected neural network with Sigmoid activation, which helps the implementation of Deep SHAP. All authors were helping each other equally with different overlapping parts of implementations and analysis of the results. Finally, all authors had equal contributions to writing the report.

We would like to thank our reviewers for noticing weak points of our proposal. In the final report, we tried to addressed all of them. For example, we tried to describe theory in our own words to make it more clear, and we added verification part to evaluation of the implementations.

## References

[1] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[2] A. Datta, S. Sen, and Y. Zick. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In *2016 IEEE symposium on security and privacy (SP)*, pages 598–617. IEEE, 2016.

[3] F. K. Došilović, M. Brčić, and N. Hlupić. Explainable artificial intelligence: A survey. *41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2018*, pages 0210–0215, 2018.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012.

[5] S. Lipovetsky and M. Conklin. Analysis of regression in game theory approach. *Applied Stochastic Models in Business and Industry*, 17(4):319–330, 2001.

[6] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *International Conference on Neural Information Processing Systems*, pages 4768–4777, 2017.

[7] M. T. Ribeiro, S. Singh, and C. Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[8] L. Shapley. A value for n-person games. In *In: Kuhn, H. and Tucker, A., Eds., Contributions to the Theory of Games II*, pages 307–317. Princeton University Press, Princeton, 1953.

[9] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

[10] A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. In *International Conference on Machine Learning*, pages 3145–3153. PMLR, 2017.

[11] E. Štrumbelj and I. Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and information systems*, 41(3):647–665, 2014.

# Appendices

## A Change to the proposed plan

In the final project we made the following changes with respect to the proposed plan:

- We changed the datasets. We omit text data as we were able to process image data successfully. We omitted CIFAR-10, HIGGS, and Rotten Tomatoes to safe time. Instead, we used Kaggle's housing tabular data, MNIST and Fashion-MNIST imaging data.
- We limited application of the SHAP values method only to housing tabular dataset, for linear regression model. We consider it is sufficient.
- We limited the application of Kernel SHAP to housing tabular dataset, for linear regression and XGBoost.
- The authors' contributions were slightly adjusted, e.g. all authors are contributing to the proposal writing equally.
- We extended evaluation part, to address reviewers requests, by elaborating on methods verification.

## B SHAP value implementation details

The SHAP values implementation includes the following main tasks:

- Generate a dataset incorporating all possible combinations maps (or masks) of including-excluding features (e.g. [11],[01],[10],[00] are masks for 2 features). Do it for each row. We fill zero-mask elements with features values. It generates a dataset of size $(2^M n, M)$. (We found that need to exclude all-ones and all-zeros masks from computations to get better results).
- In addition, generate a similar dataset with $i$-th columns being zero in all masks. It creates $M$ datasets each of size $(2^M n, M)$.
- Using the original model $f$, compute $2^M$ average values of model predictions $f(h_x(z')) \approx E_{z_{\bar{S}}}[f(z)]$ (for simplicity, we assume features independence).
- Similarly, compute model prediction averages applied to each specific dataset where $i$-th columns is substituted with a background.

## C Structures of neural networks and parameters used in training

The structures of the three neural networks used in Section 4.3 are as follows:

**NN with Sigmoid**: A small neural networks with two fully-connected layers with Sigmoid activation functions and Softmax output. This model is used for MNIST dataset.

**NN with ReLU**: A neural networks with two fully-connected layers with ReLU activation functions and Softmax output. This model is used for MNIST. For Fashion-MNIST, we add one more fully-connected layers with ReLU activation functions.

**CNN with ReLU**: A neural networks with two convolutional layers followed by two fully connected layers. All the hidden layers use ReLU activation functions and the output is Softmax. We also use dropout during training. This model is used for MNIST and Fashion-MNIST.

More details are shown below:

**NN with Sigmoid**:

- Fully connected layers with 32 neurons
- Sigmoid activation
- Fully connected layers with 10 units
- Softmax output

**NN with ReLU for MNIST**:

- Fully connected layers with 128 neurons
- ReLU activation
- Fully connected layers with 10 units
- Softmax output

**NN with ReLU for Fashion-MNIST**:

- Fully connected layers with 512 neurons
- ReLU activation
- Fully connected layers with 128 neurons
- ReLU activation
- Fully connected layers with 10 units
- Softmax output

**CNN with ReLU**:

- Convolutional layer with 32 filters of kernel size (4,4) and stride (2,2)
- ReLU activation
- Convolutional layer with 64 filters of kernel size (4,4) and stride (2,2)
- ReLU activation
- Dropout with probability 0.25 of leaving out units
- Fully connected layers with 128 neurons
- ReLU activation
- Dropout with probability 0.5 of leaving out units
- Fully connected layers with 10 units
- Softmax output

The training parameters are as follows.

**NN with Sigmoid**:

- Optimizer: SGD
- Learning rate: 0.01
- Batch size: 256
- Epochs: 500

**NN with ReLU**:

- Optimizer: Adam
- Learning rate: 0.001
- Batch size: 256
- Epochs: 100

**CNN with ReLU**:

- Optimizer: Adam
- Learning rate: 0.001
- Batch size: 128
- Epochs: 20 for MNIST; 40 for Fashion-MNIST

## D Additional Deep SHAP results for classifying MNIST digits

Figure 6 shows deep SHAP results using NN with ReLU model. We have the same observations as Figure 4a except that the pattern is a little bit less clear and the curves are thinner. We conjecture that the reason may be that some inactivated ReLU units stop the back-propagation of Deep SHAP multipliers.

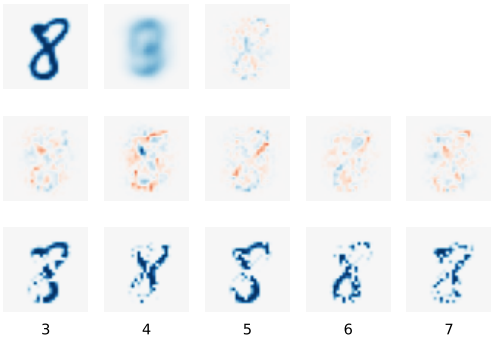



Figure 6: Deep SHAP: explaining NN with ReLU for the classification of an image of digit 8 in MNIST.

Figure 7 compares our Deep SHAP results with those in [6]. Note that in [6] red color is used to indicate positive contributions and blue for negative contributions while we use blue for positive contributions and orange for negatie contributions. Our heatmaps are a little bit inconsistent with those in [6] but the deep color areas are the same. We conjecture that the reason may be that the authors of [6] did some clipping on the SHAP values or erased the SHAP values that are not inside the handwritten curves areas. Actually, their results on GitHub website are also inconsistent with those in their paper.

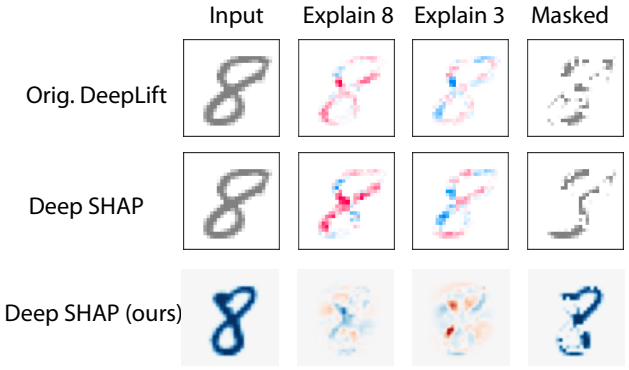



Figure 7: Deep SHAP: comparison