Decision Explanation and Feature Importance for Invertible Networks

Juntang Zhuang, Nicha C. Dvornek, Xiaoxiao Li, Junlin Yang, James S. Duncan
Yale University

Contents

Background

1. Interpretation of model decision is important for real-world applications, such as disease diagnosis and fraud detection.

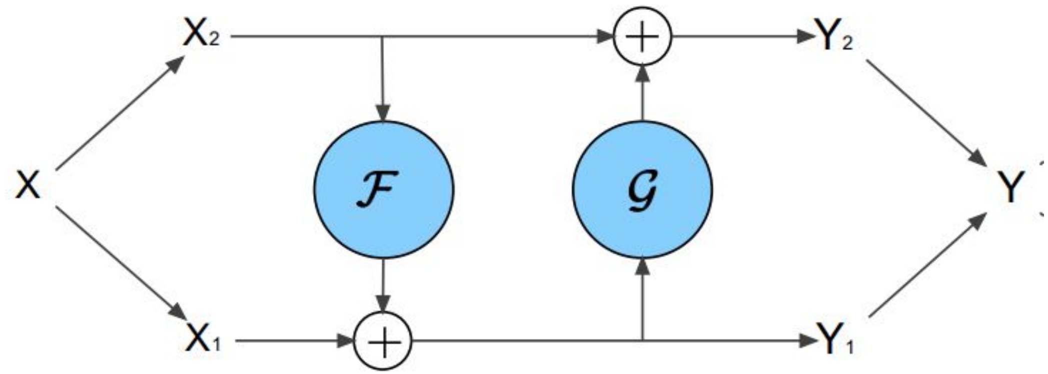
2. Deep networks are typically black-box models, thus hard to interpret.

3. Invertible blocks can accurately reconstruct the input from output, and have the potential to unravel black-box model.

Contents

Structure of Invertible Network

Invertible Block



1) Input is split into two parts (x_1 and x_2) with the same shape. Output is (y_1 and y_2).

The split can be any form, a common example is to split by channel (e.g. split a $H \times W \times C$ tensor into two $H \times W \times \left(\frac{C}{2}\right)$ tensors).

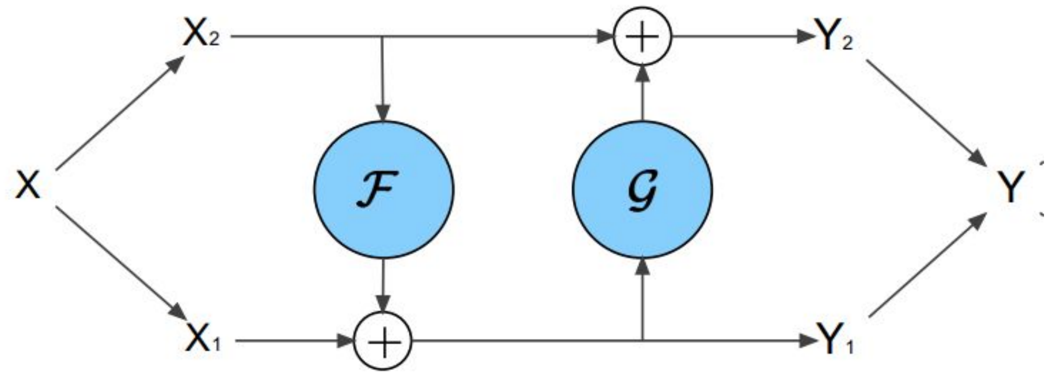
2) ***F*** and ***G*** are two neural networks (e.g. stack of Conv-BN-ReLU layers).

The only requirement is: output of ***F*** and ***G*** must have the same shape as their input.

Ref: MemCNN: a Framework for Developing Memory Efficient Deep Invertible Networks

Structure of Invertible Network

Invertible Block



Forward: ***x*** to ***y***

$$\begin{cases} y_2 = x_2 + F(x_1) \\ y_1 = x_1 + G(y_2) \end{cases}$$

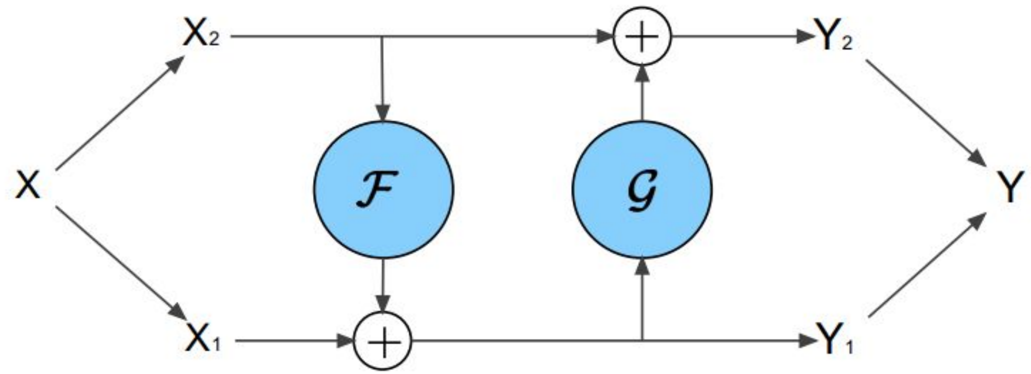
Inverse: ***y*** to ***x***

$$\begin{cases} x_1 = y_1 - G(y_2) \\ x_2 = y_2 - F(x_1) \end{cases}$$

1. Forward and inverse functions define a one-to-one (bijective) mapping between ***x*** and ***y***.
2. The mapping is bijective regardless of forms of ***F*** and ***G***.
   In the extreme case, when parameters of ***F*** and ***G*** are random, the mapping is still bijective.

Structure of Invertible Network

Invertible Block



Forward: ***x*** to ***y***

$$\begin{cases} y_2 = x_2 + F(x_1) \\ y_1 = x_1 + G(y_2) \end{cases}$$

Inverse: ***y*** to ***x***

$$\begin{cases} x_1 = y_1 - G(y_2) \\ x_2 = y_2 - F(x_1) \end{cases}$$

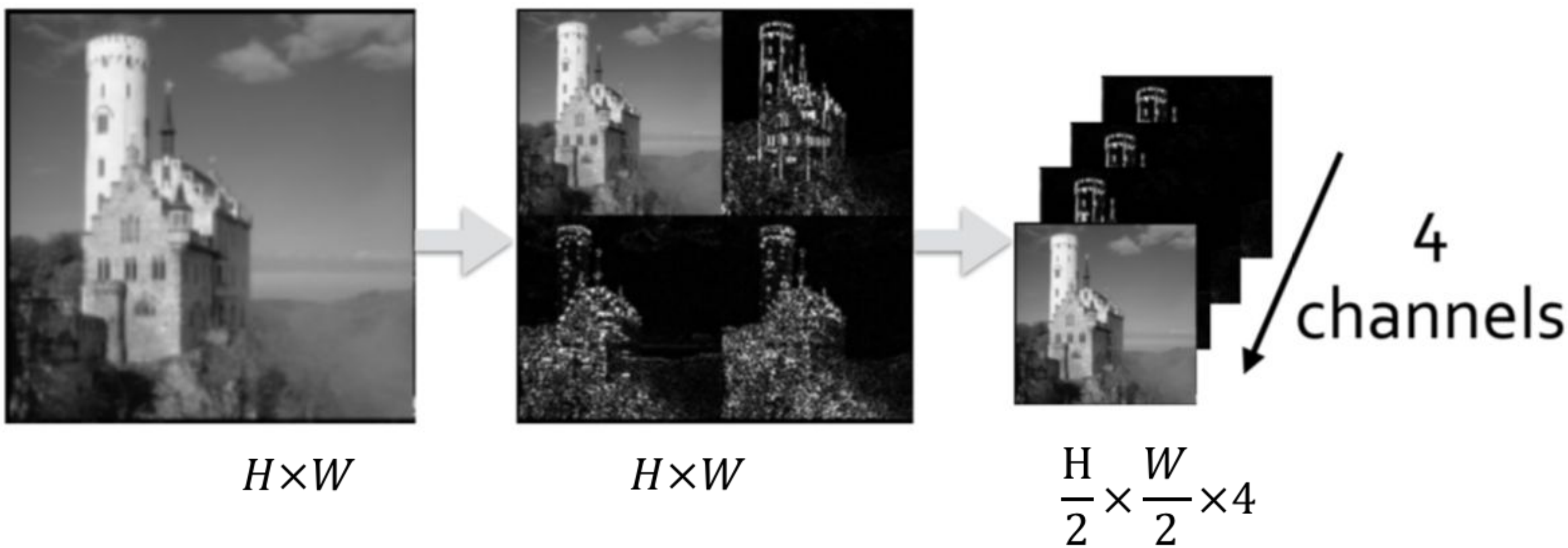
1. "Inversion" is not "Back-propagation".
   Back-prop is used to optimize parameters in ***F*** and ***G***;
   Inversion only reconstructs ***x*** from ***y***, not calculates the gradient of parameters.

2. Inversion does not affect training.
   Invertible network is trained the same way as conventional networks.

3. Training does not affect inversion.

Structure of Invertible Network

Invertible Pooling with 2D Wavelet Transform



1. For an $H \times W \times C$ tensor, each channel is an $H \times W$ 2D image
2. For each 2D image, perform 2D wavelet transform to halve spatial size
3. 2D wavelet transform is invertible

Structure of Invertible Network

Batch-Norm Layer

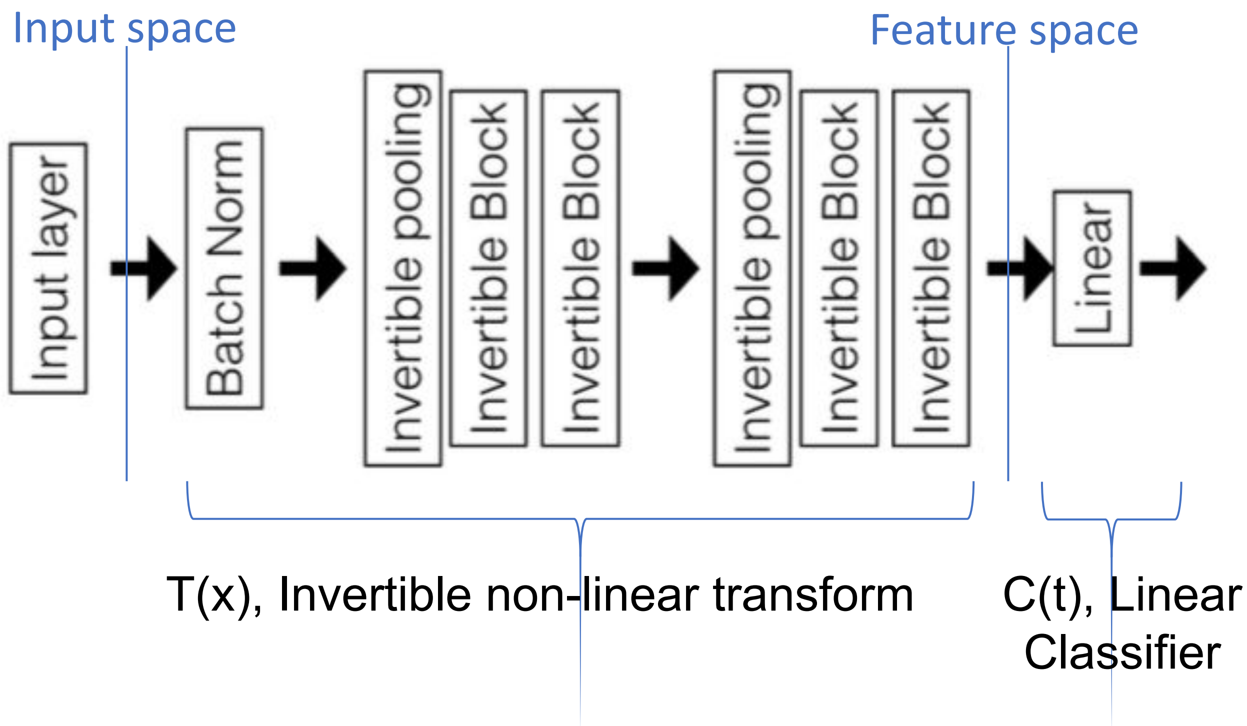
$$y = \frac{x - \mathbf{E}(x)}{\sqrt{\mathbf{Var}(x) + \epsilon}}\gamma + \beta, \quad x = \frac{y - \beta}{\gamma}\sqrt{\mathbf{Var(x)} + \epsilon} + \mathbf{E}(x)$$

Linear Layer

1. Most neural network classifiers end with a channel-wise mean pooling and a FC layer (without activation).

2. The average-pooling and FC can be merged into one linear transform, we call it “Linear Layer”.

Structure of Invertible Network



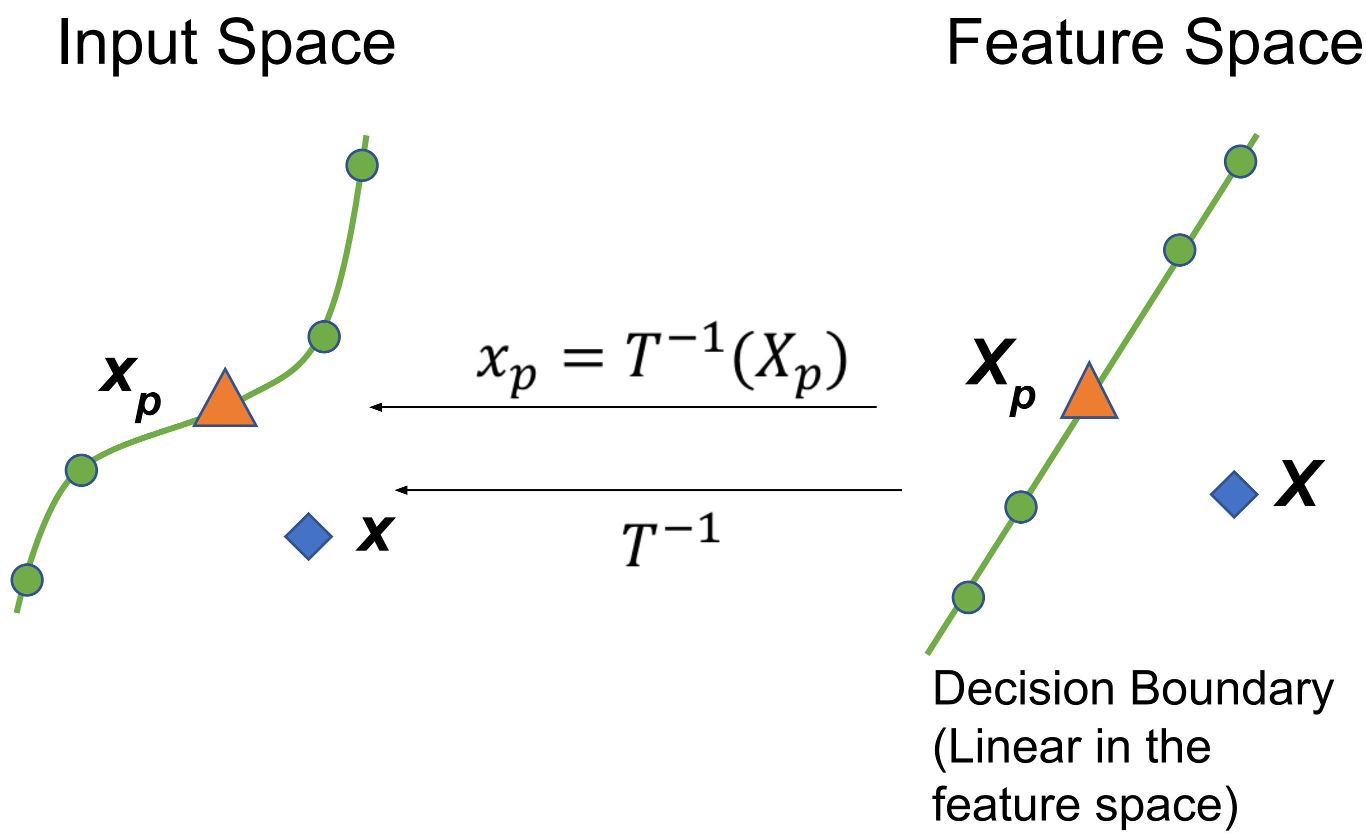
Two-stage model:

$t = T(x)$ Non-linear invertible transform from input space to feature space

$y = C(t)$ Linear classifier in the feature space

Contents

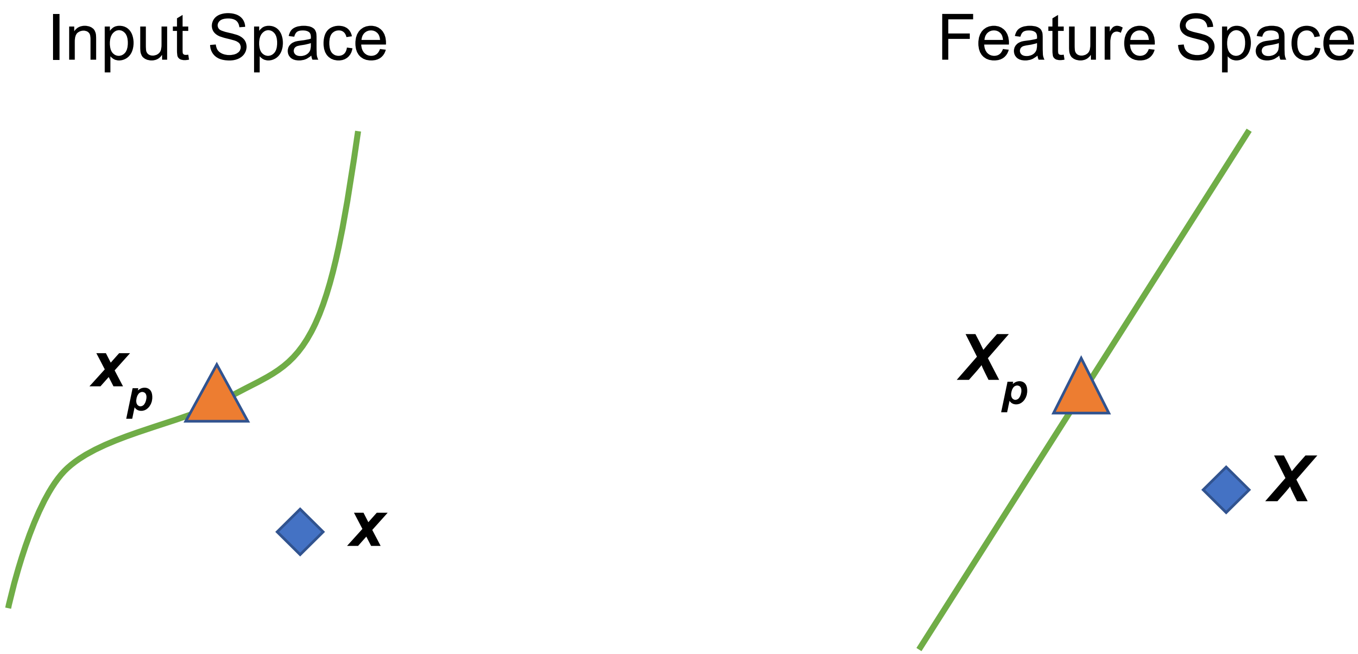
Model Decision Interpretation



1. X_p is the projection of X onto the decision boundary in the feature space
2. X_p is the closest point to X on the boundary in the feature space
3. Explanation for model decision:

$$x - x_p = T^{-1}(X) - T^{-1}(X_p)$$

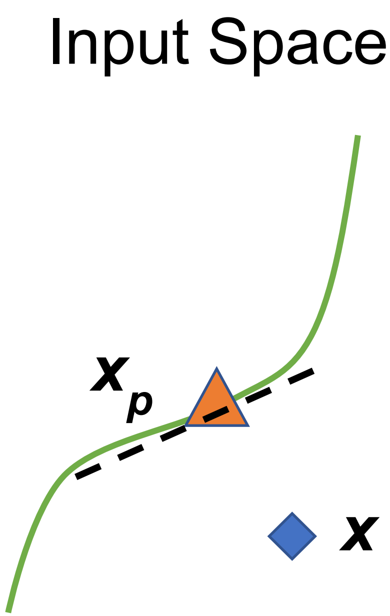
Model Decision Interpretation



Explicitly Determine decision boundary:

- Linear Classifier in the feature space.
- Explicitly determine the decision boundary and projected point in the feature domain.
- Invert boundary points to the input space.

Feature Importance



- Decision boundary is non-linear in the input space, and hard to analyze.
- Locally, we can approximate the non-linear boundary with a linear classifier, by Taylor Expansion

$$f(x) = C \circ T\ (x)$$

$$f(x) = f(x_p) + \nabla f(x_p)^T (x - x_p) + O(||x - x_p||_2^2)$$

Since x_p is on the boundary, $f(x_p) = 0$

$$f(x) = \nabla f(x_p)^T (x - x_p) = \sum_i (x^i - x_p^i)\, w^i$$

Contribution of dimension i is: $(x^i - x_p^i)\ w^i$

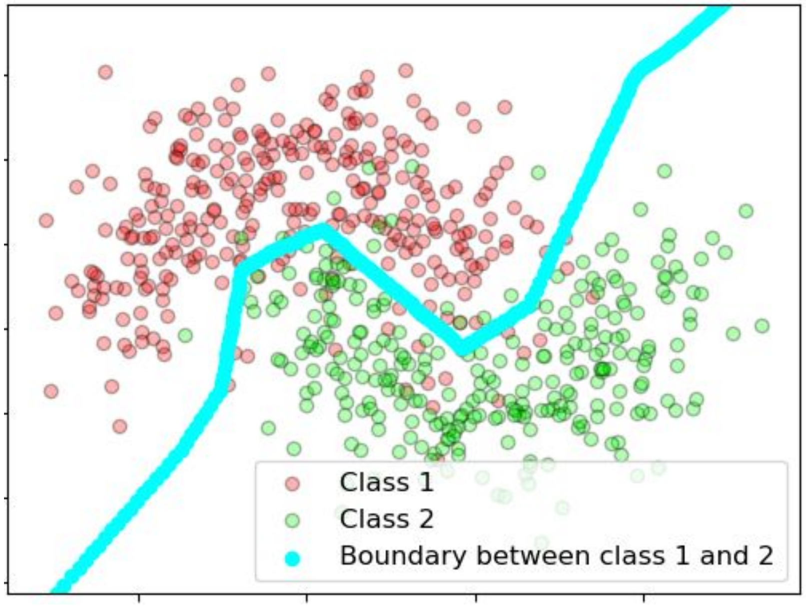
The importance of dimension i is defined as: $|(x^i - x_p^i) w^i|$

Contents

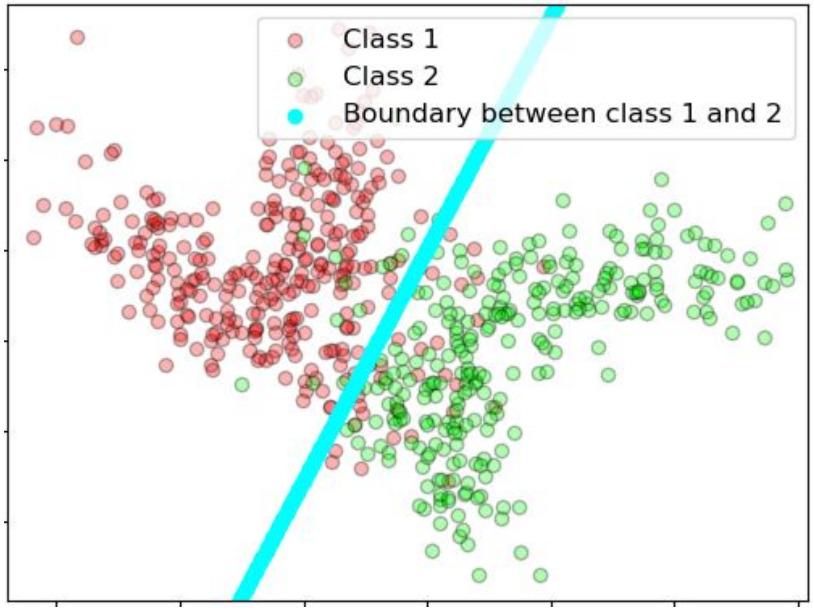
Experiments and Results

Classification results on a 2-dim toy-dataset

Input Space

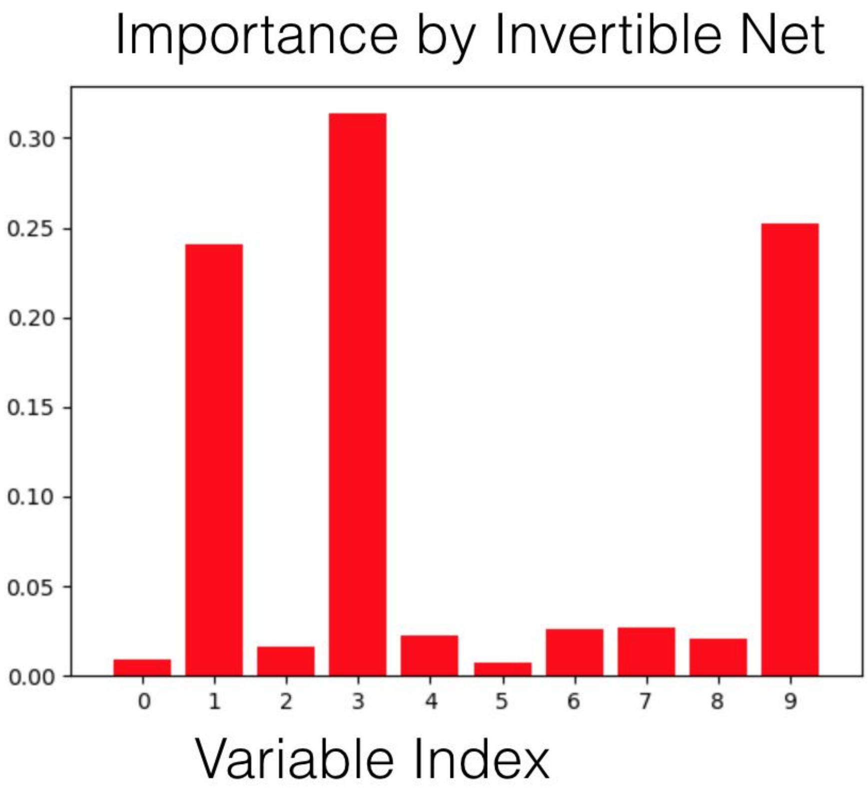


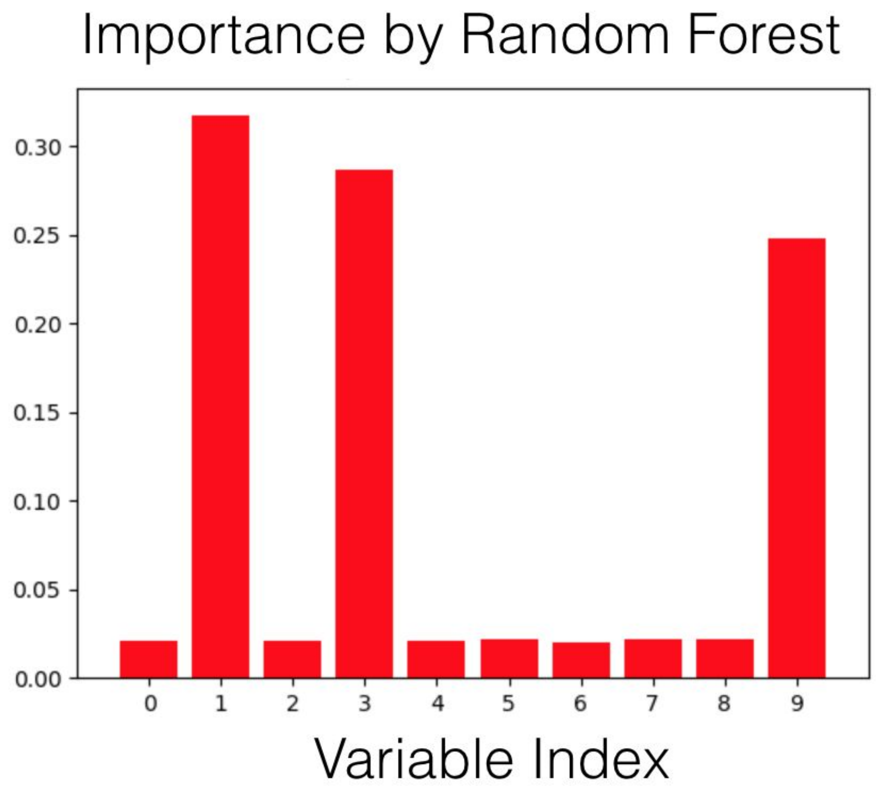
Feature Space



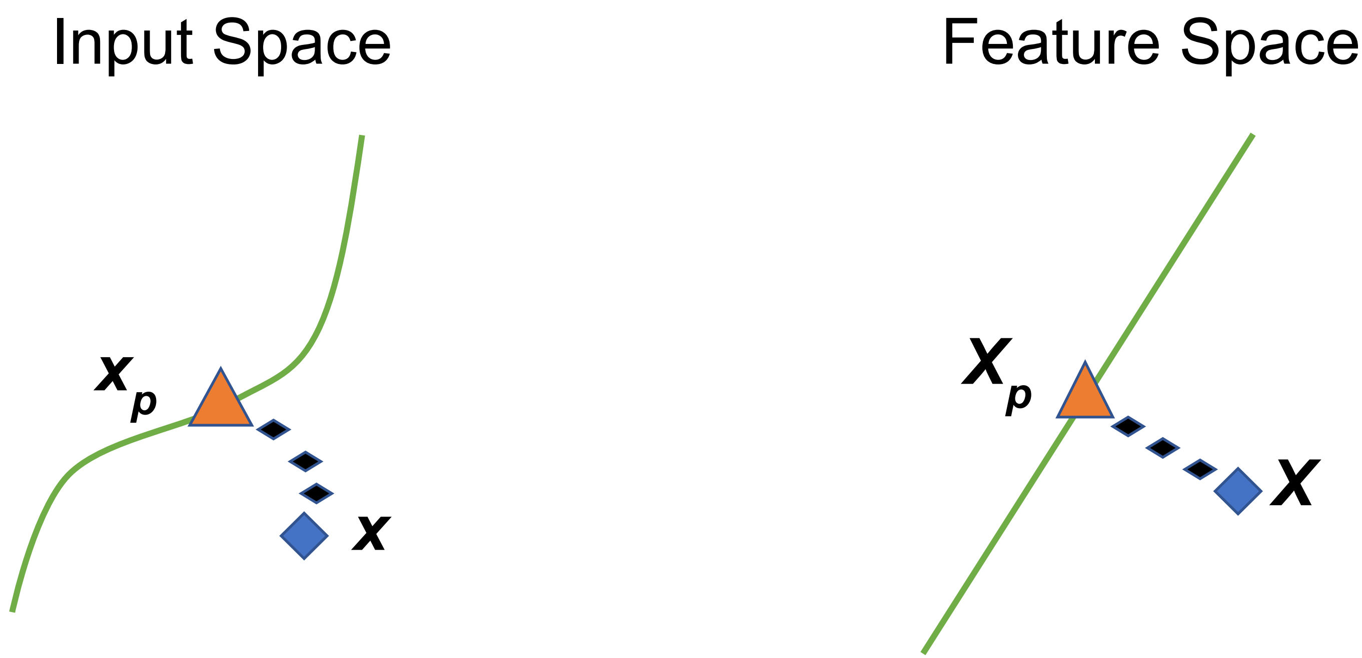
Experiments and Results

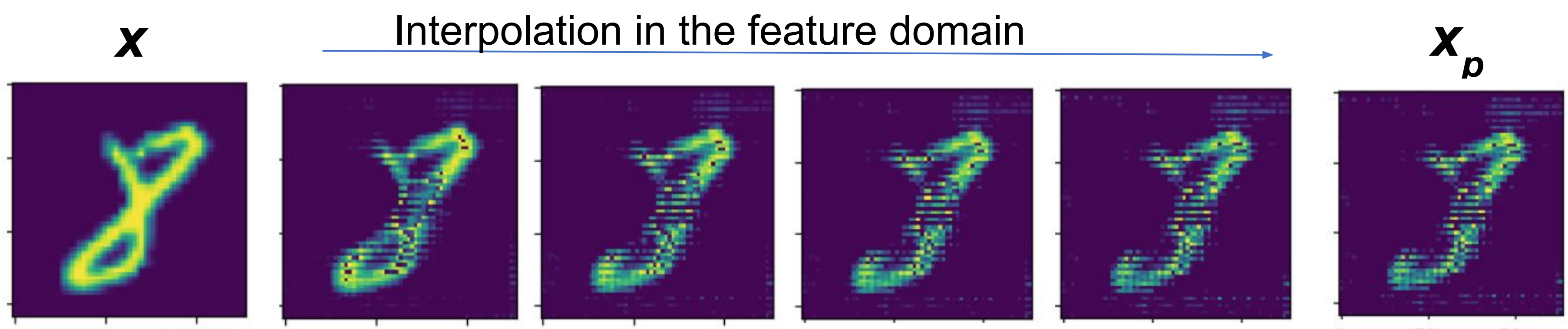
We create a 10-dim dataset with two classes, only variable 1, 3 and 9 are informative features, other dimensions are noise.





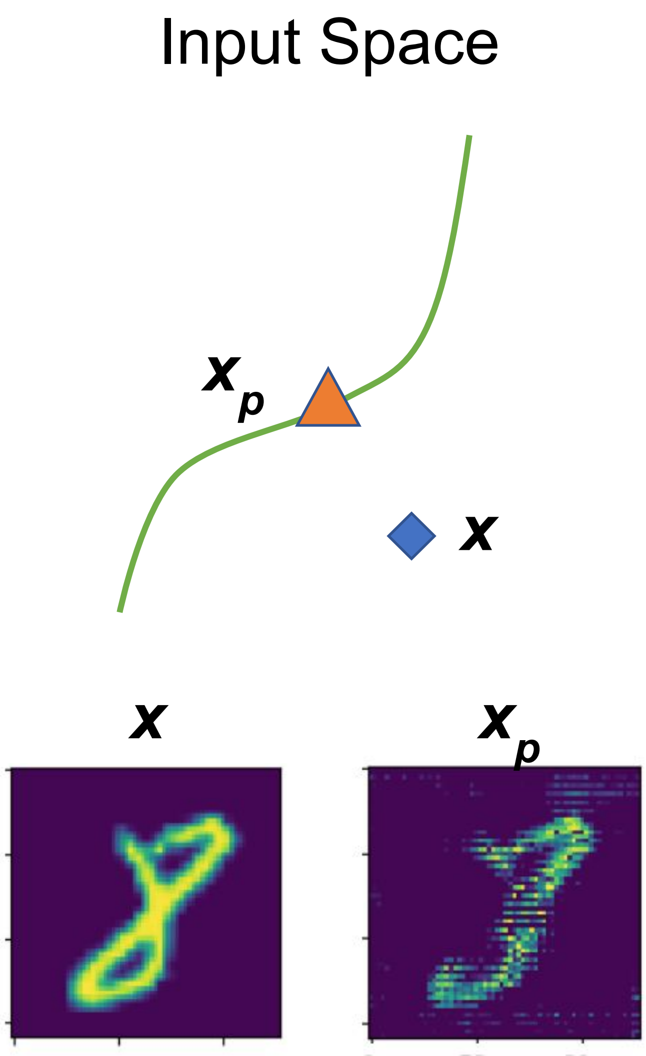
Experiments and Results

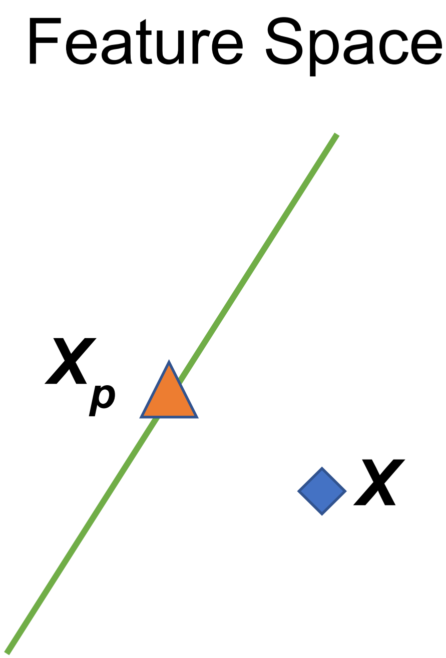




Note that x_p is calculated from x, x_p is NOT in the training set.

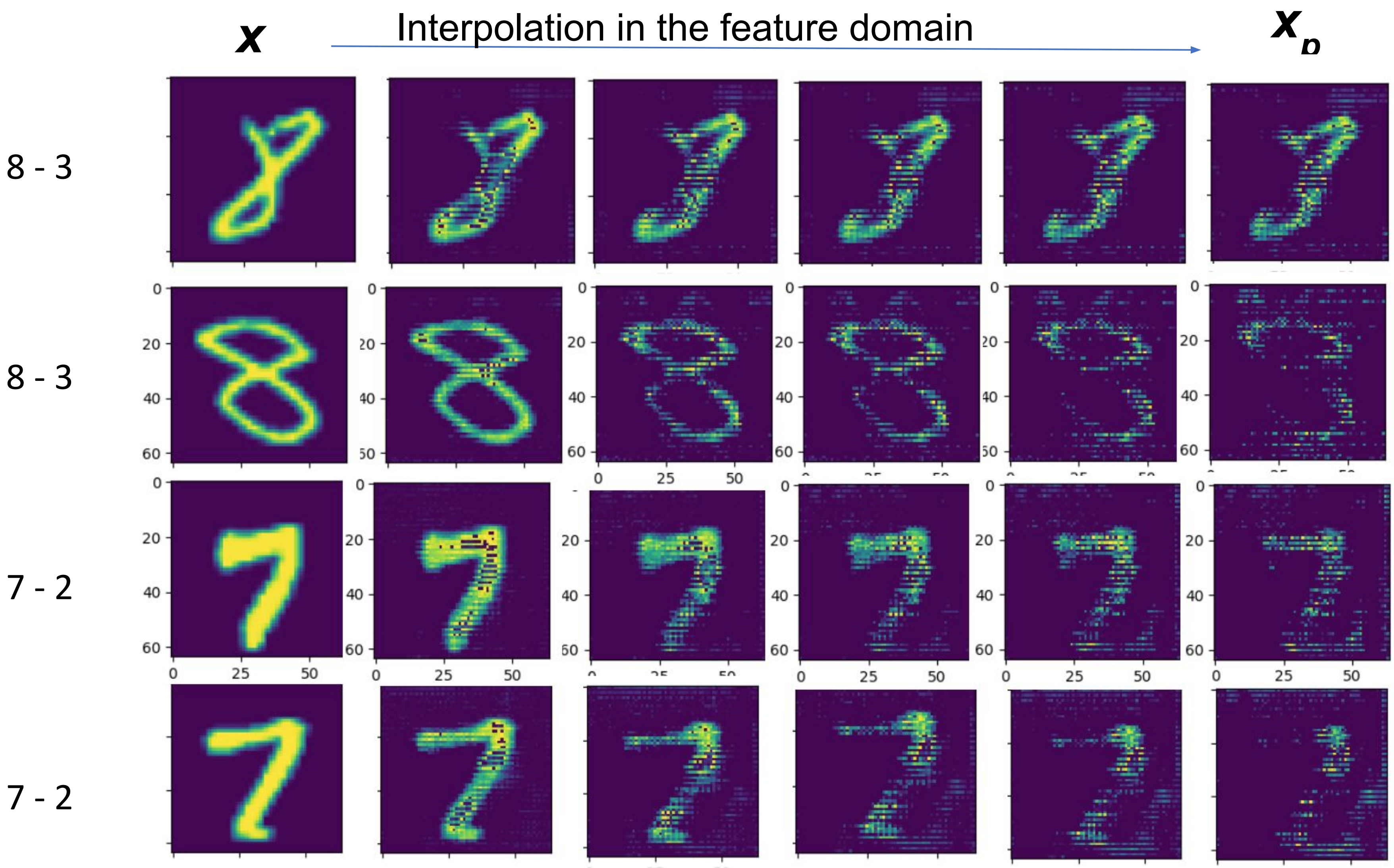
Experiments and Results



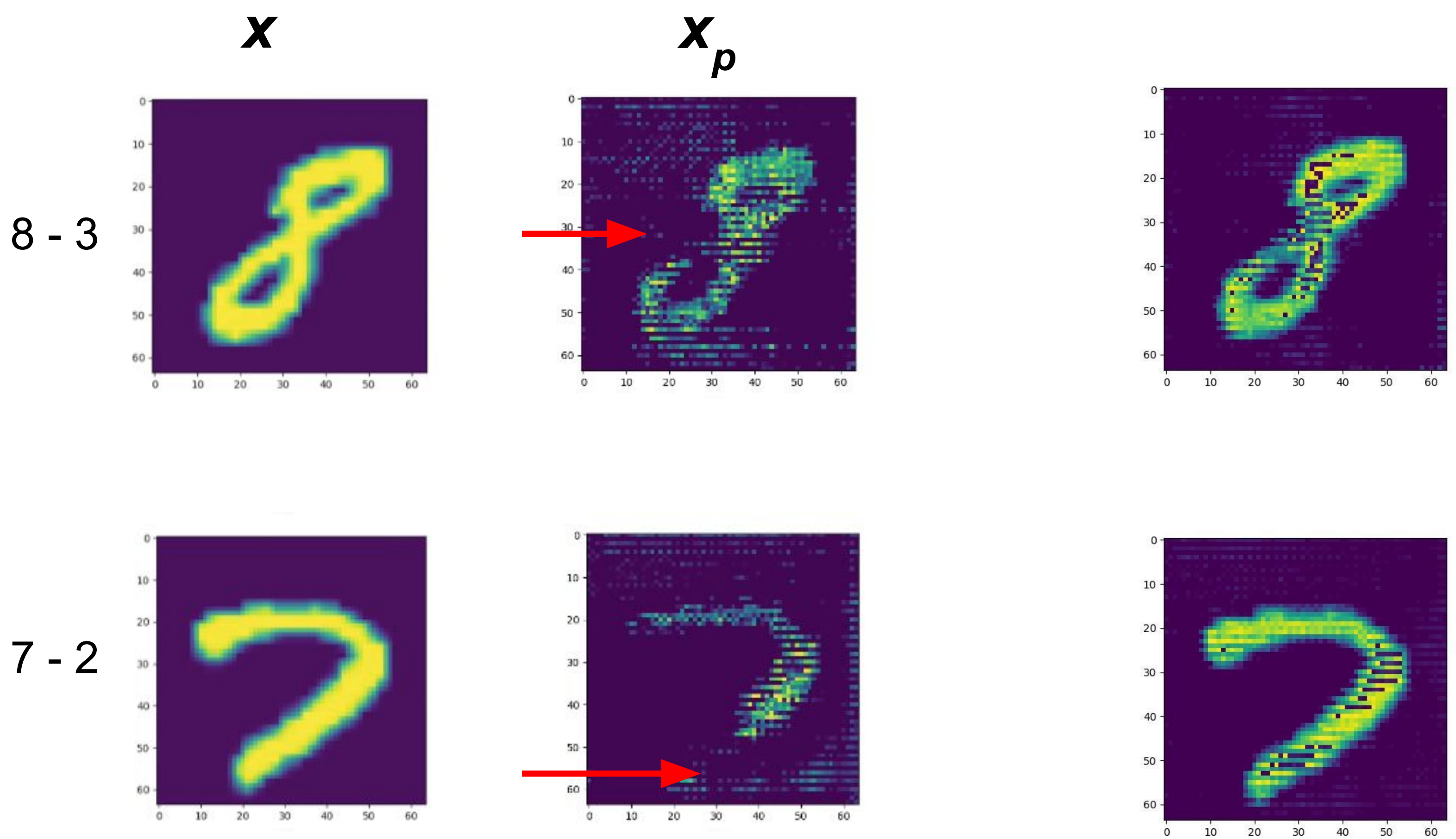


- x_p is calculated from x, x_p is NOT in the training set.
- $x - x_p$ explains model's decision.
- $f\left(x_p\right) = 0$, the model thinks x_p has equal probability of two classes.

Experiments and Results



Experiments and Results



Conclusions and Future work

1. We introduce a family of invertible networks, which can be viewed as a two-stage model:
    - invertible non-linear transform from input space to feature space
    - linear classifier in feature space

2. We can explicitly determine the decision boundary in the feature space, and invert to the input space.

3. We propose to quantify feature importance based on local linear approximation.

4. Demo code available: https://github.com/juntang-zhuang/explain_invertible

5. Future directions:
    - Generalize to a broader family of invertible networks
    - More accurate feature importance quantification
    - Analyze model behavior (such as robustness) with decision boundary

Thank you!