Explaining Neural Models for Image Classification
MTQ315 Report

GitHub: `https://github.com/rishabh-ranjan/mtq315-xai`

Rishabh Ranjan – 2018CS10416

April 2, 2022

1 Introduction

Deep neural architectures such as Convolutional Neural Networks (CNNs) have achieved remarkable performance on image classification tasks in recent years. However, such models tend to be brittle, relying on spurious correlations in the data to make predictions, making it essential to audit them before deploying in the wild, where they could have undesirable societal impact. Consider, for example, the below figure reproduced from [2].

The figure shows saliency maps for an image captioning model. Saliency maps are obtained by taking the gradient of the output with respect to the input. Higher gradients indicate higher importance of that input. In the figure red regions denote the pixels which affect the model's output the most. Saliency map is an XAI technique in itself, but was not explored in this project. In this particular example, XAI exposes the gender bias of the model, which would otherwise have gone unnoticed.

Hence, it is important to analyze image classification models with respect to the pixels being looked at by the model to make predictions.

2 XAI Techniques

This section describes the XAI techniques explored in this project.

2.1 SHAP explanations

SHAP stands for Shapley Additive Explanations. It was introduced in [4]. The idea is to provide additive feature attribution for simplified features on a local approximation of the model using Shapley Values.

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

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

Here, $z' \in \{0, 1\}^M$ are simplified inputs, g is a local approximation of the model, and ϕ_i's are Shapley values.

Computing the exact SHAP values is intractable. Hence, [4] introduces several ways to approximate them. For this project the relevant ones are:

1. **KernelSHAP:** A model-agnostic approximation. Subsumes LinearLIME [5].
2. **DeepSHAP:** Optimized for deep neural networks, leveraging the compositionality of neural components. Subsumes DeepLIFT [6].

2.2 Counterfactuals

Given an input which gives an undesirable prediction, a counterfactual is another hypothetical input which is close to the original input and is classified into a desired class. For images, asking for a counterfactual is intuitively equivalent to asking for the minimal change (under some measure) to the pixel values of the image to achieve a target class. We explore the following methods to generate image counterfactuals:

1. **The First Approach:** Introduced by the seminal paper [7], this was the first method to generate counterfactuals. Counterfactuals are obtained by optimizing the loss function

   $$\mathcal{L}(x'|x) = (f_t(x') - p_t)^2 + \lambda \|x' - x\|_1$$

   where x is the input, x' is the counterfactual, t is the target class, p_t is the desired probability of the target class, f_t is the model prediction at the target class, λ is a hyperparameter and $\|.\|_1$ is the L_1 norm. The use of L_1 distance between the input and the counterfactual enforces sparsity, i.e. the generated counterfactual should differ from the original image only in a few pixels.

2. **Guided by Prototypes:** This method generates in-distribution counterfactuals (i.e. counterfactual image looks like images in the dataset) by following a prototype of the target class to guide the search. This method was introduced in [3].

3. **Via Reinforcement Learning:** This method does not assume differentiability and uses the RL technique of Deep Deterministic Policy Gradients (DDPG) to find counterfactuals. The black-box treatment of the model is helpful in introducing arbitrary flexible constraints, such as immutable protected features like gender or race. However, for this project, there doesn't seem to be any particular benefit of this method.

3 Experimental Setup

This section describes the technical details concerning the implementation and experiments.

3.1 Task

We consider the multi-class image classification task on MNIST-like datasets [1] with 28x28 grayscale images. There are 10 classes.

3.2 Datasets

We use the following datasets:

1. MNIST: a dataset of handwritten digits. Classes are digits 0 to 9.
2. FMNIST (FashionMNIST): a dataset of clothing items. This is significantly harder than the MNIST dataset. The classes are: top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, boot.

Both datasets have $60,000$ training examples and $10,000$ test examples.

3.3 Frameworks

For explaination experiments we use the `shap` library, and use `PyTorch` for deep learning. KernelSHAP is available as `KernelExplainer`, and DeepSHAP is available as `DeepExplainer`. Even though in principle KernelSHAP can be applied to deep neural networks, the `shap` library does not implement KernelSHAP with `PyTorch` support. This makes sense because for neural networks DeepSHAP is known to work better than KernelSHAP and hence must always be used in preference.

For counterfactual experiments we use the `alibi` library, and use `TensorFlow` for deep learning. The first method is available as `Counterfactual` class, the prototype method is available as `CounterfactualProto` class and the RL method is available as `CounterfactualRL` class. Due to a bug in the library, `CounterfactualRL` didn't train sufficiently well, so the corresponding results are omitted.

3.4 Models

We use Convolutional Neural Networks (CNNs), as they achieve state-of-the-art results on these datasets.

The exact architecture details are given below:

1. **CNN for explanation experiments in PyTorch:**

```
self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 10, kernel_size=5),
            nn.MaxPool2d(2),
            nn.ReLU(),
            nn.Conv2d(10, 20, kernel_size=5),
            nn.Dropout(),
            nn.MaxPool2d(2),
            nn.ReLU(),
        )
        self.fc_layers = nn.Sequential(
            nn.Linear(320, 50),
            nn.ReLU(),
            nn.Dropout(),
            nn.Linear(50, 10),
            nn.Softmax(dim=1)
        )
```

2. **CNN for counterfactual experiments in TensorFlow:**

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
```

```
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 64)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 32)        8224
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 1568)              0
_________________________________________________________________
dense (Dense)                (None, 256)               401664
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570
=================================================================
Total params: 412,778
Trainable params: 412,778
Non-trainable params: 0
```

These models attain **98.7 %** test accuracy on MNIST and **92.5 %** test accuracy on FMNIST. This confirms that the models were properly trained, and the results thus obtained are valid.

4 Results

Each row is an input image (one from each class) and each column is a target class. To make comparison possible across the different explainability methods, the exact same input images have been used in all experiments for a dataset. We give detailed analysis for the MNIST results. The FMNIST results are also included, in the same format, and can be interpreted similarly.

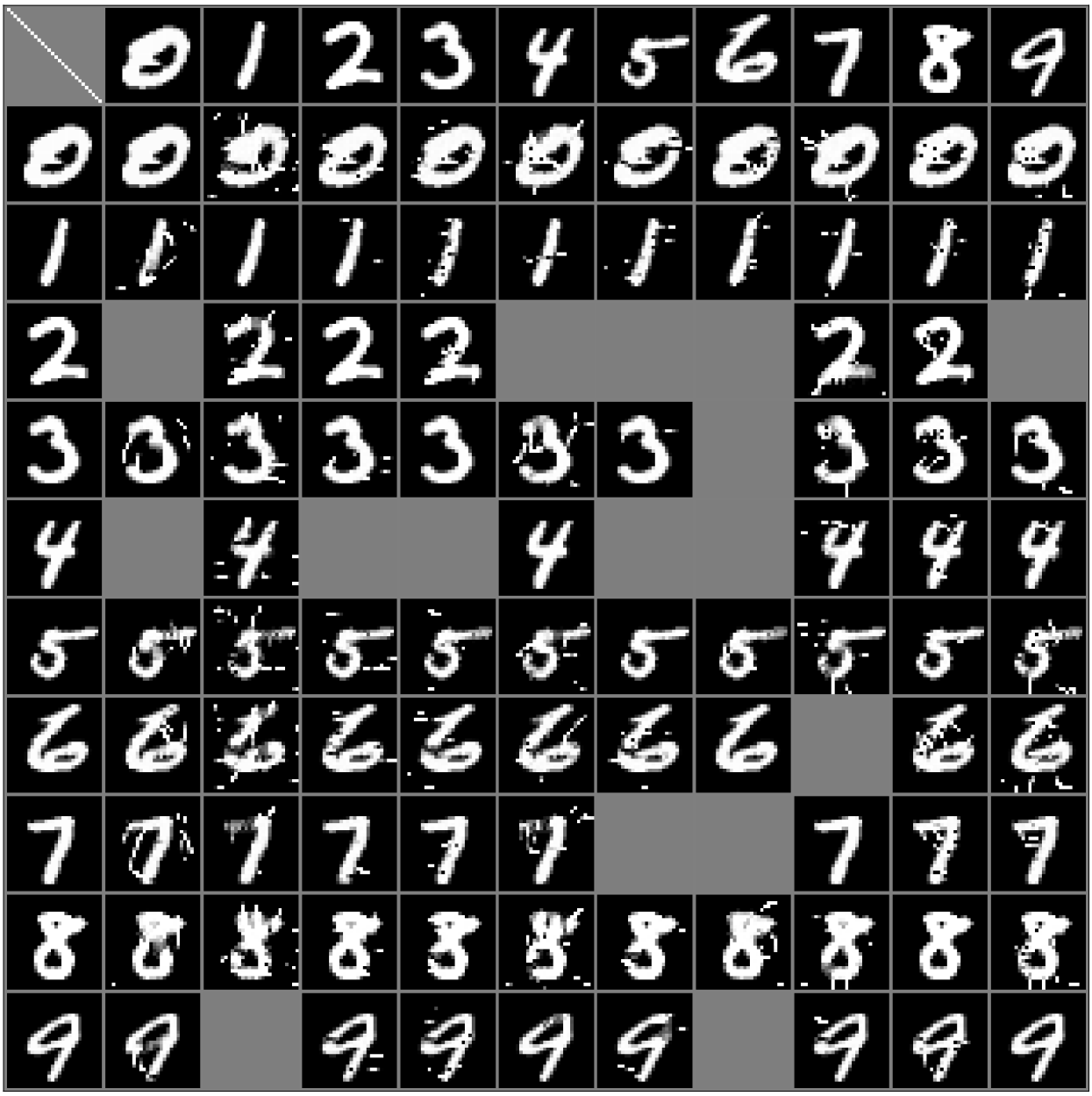
4.1 MNIST

4.1.1 SHAP Explanations

Observations:

1. The original visualization produced by the `shap` library have extremely faint red and blue regions, making the visualization hard to interpret. To alleviate this issue, I have removed the transparency from the visualization. One negative impact of this is that the magnitude of the shap values is harder to distinguish. However, the resulting images are much more interpretable and serve the purpose of XAI.
2. **RED:** this pixel in the input image encourages classification into the class given by the column.
3. **BLUE:** this pixel in the input image discourages classification into the class given by the column.

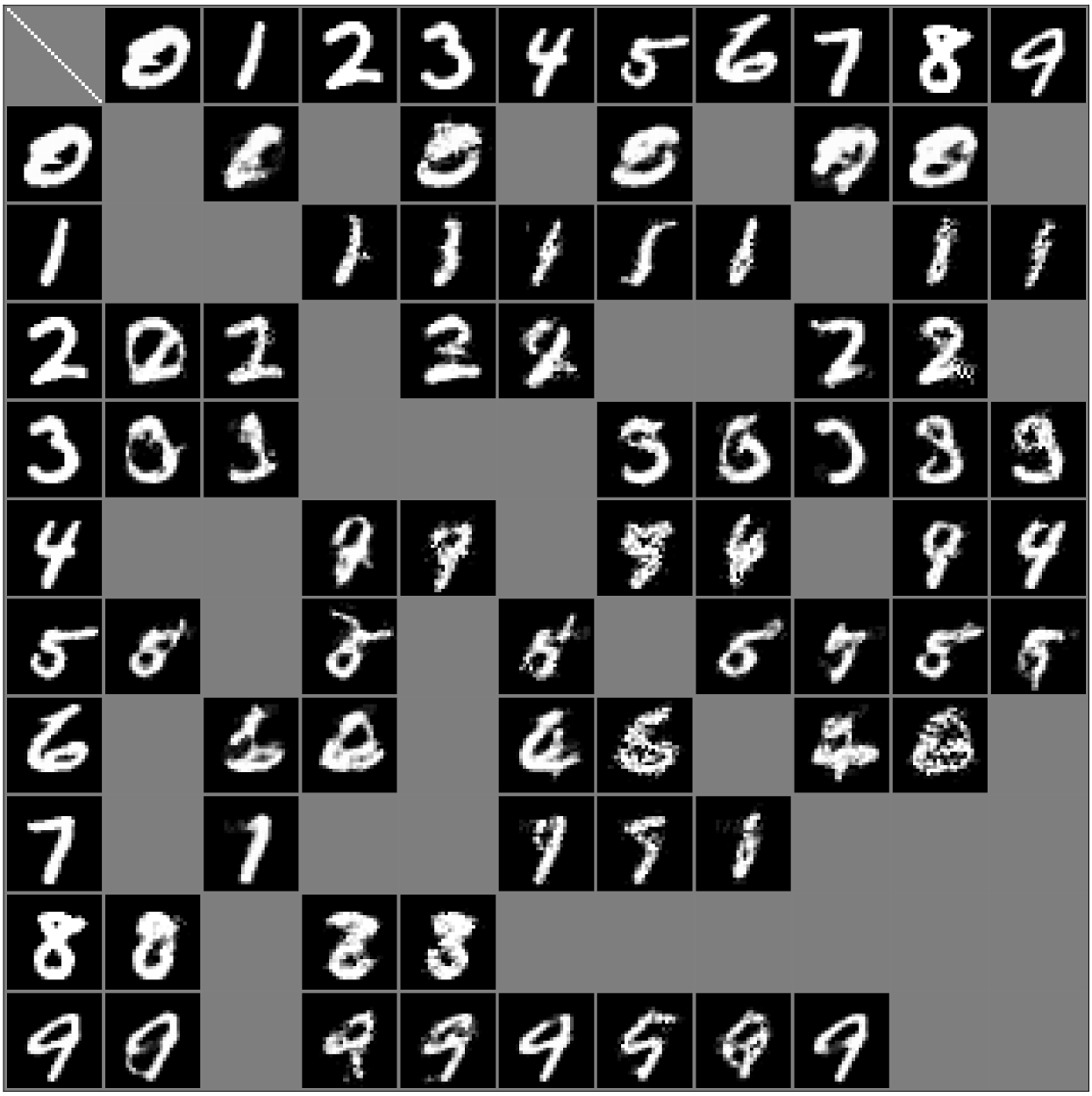
4.1.2 Simple Counterfactuals



Observations:

1. Counterfactuals are not in-distribution, i.e. they don't look like other images in the dataset.
2. The L_1 term in the loss is responsible for sparsity. Most of the image remains the same, but only a few pixels are changed to get the target class. This exposes the brittleness of neural models. Adding a few pixels at strange places is sufficient to make the model predict a desired target class, even though the resulting image does not look like the target class to humans. This is related to adversarial attacks, which is a thriving research field in ML.

4.1.3 Prototype Guided Counterfactuals



Observations

1. The generated counterfactuals are in-distribution: they look at least somewhat like the dataset images. This is due to the use of prototypes.

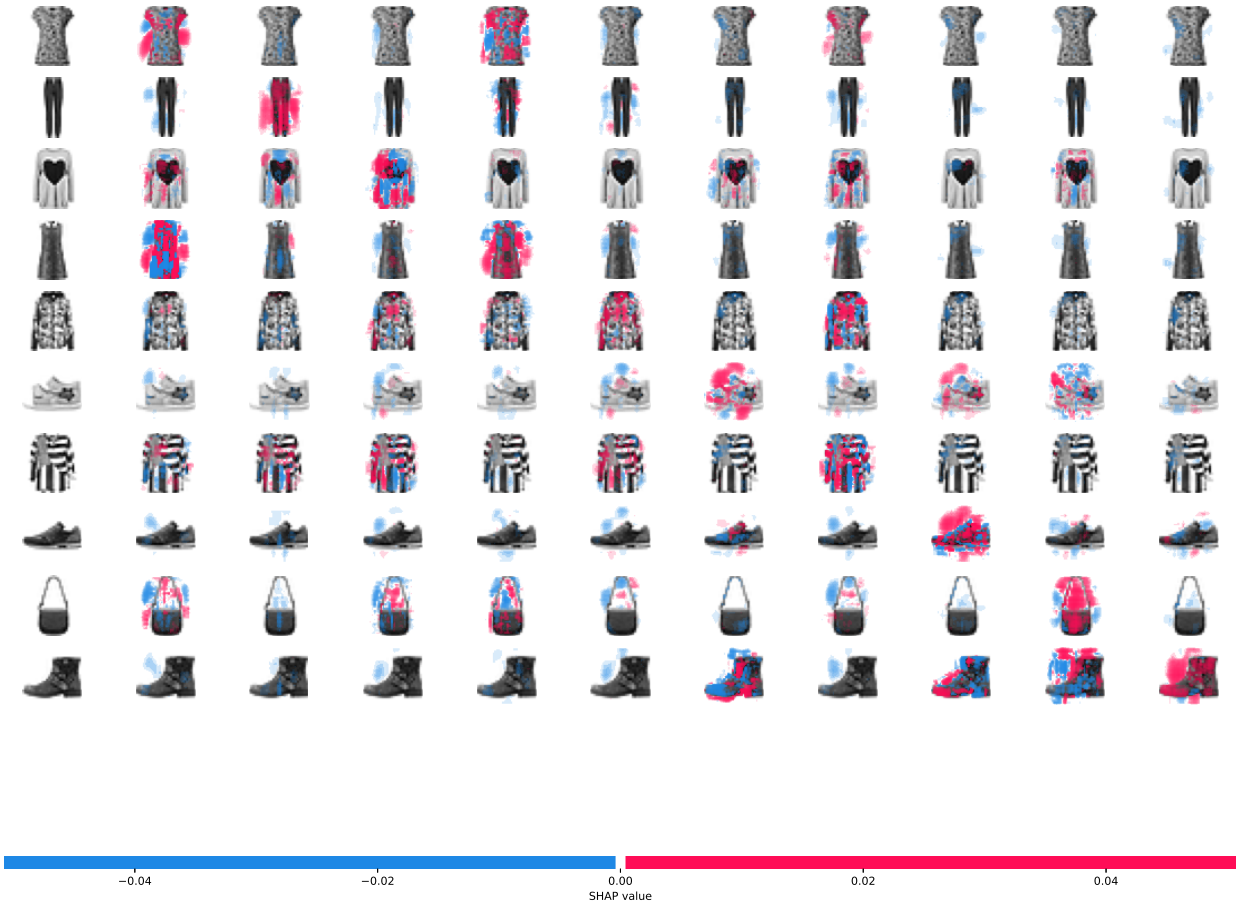
2. The failure frequency of finding counterfactuals is higher. This is because the additional constraint of following the prototype makes the optimization problem harder.

3. Comparing with SHAP explanations, we see that the changed pixels in the counterfactual are correlated with the pixels highlighted by SHAP: roughly, **blue** pixels are flipped.

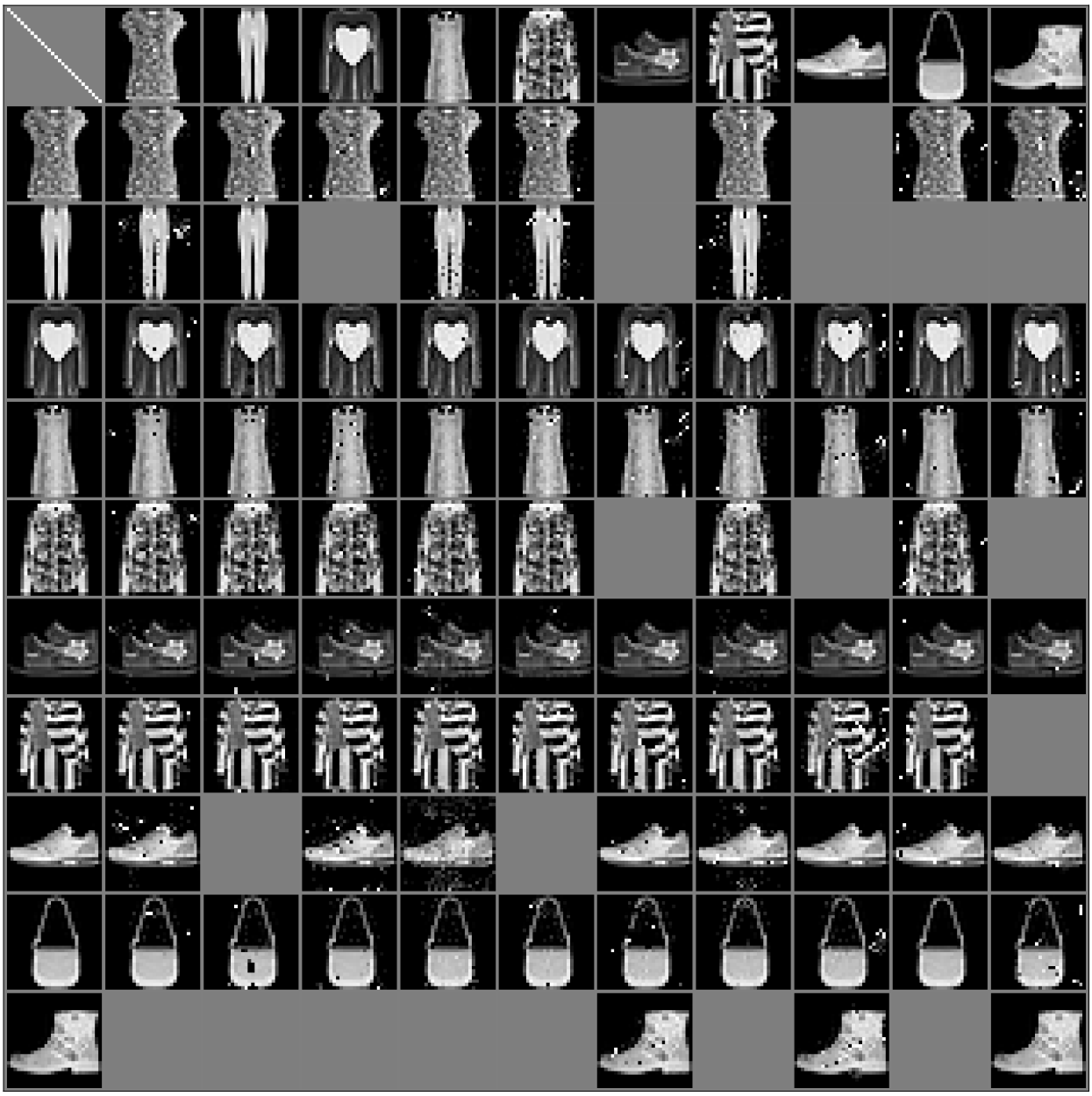
4. In the above, all blue pixels need not be flipped as only some may be sufficient to produce a counterfactual and we want the counterfactual to still be close to the input image.

4.2 FMNIST

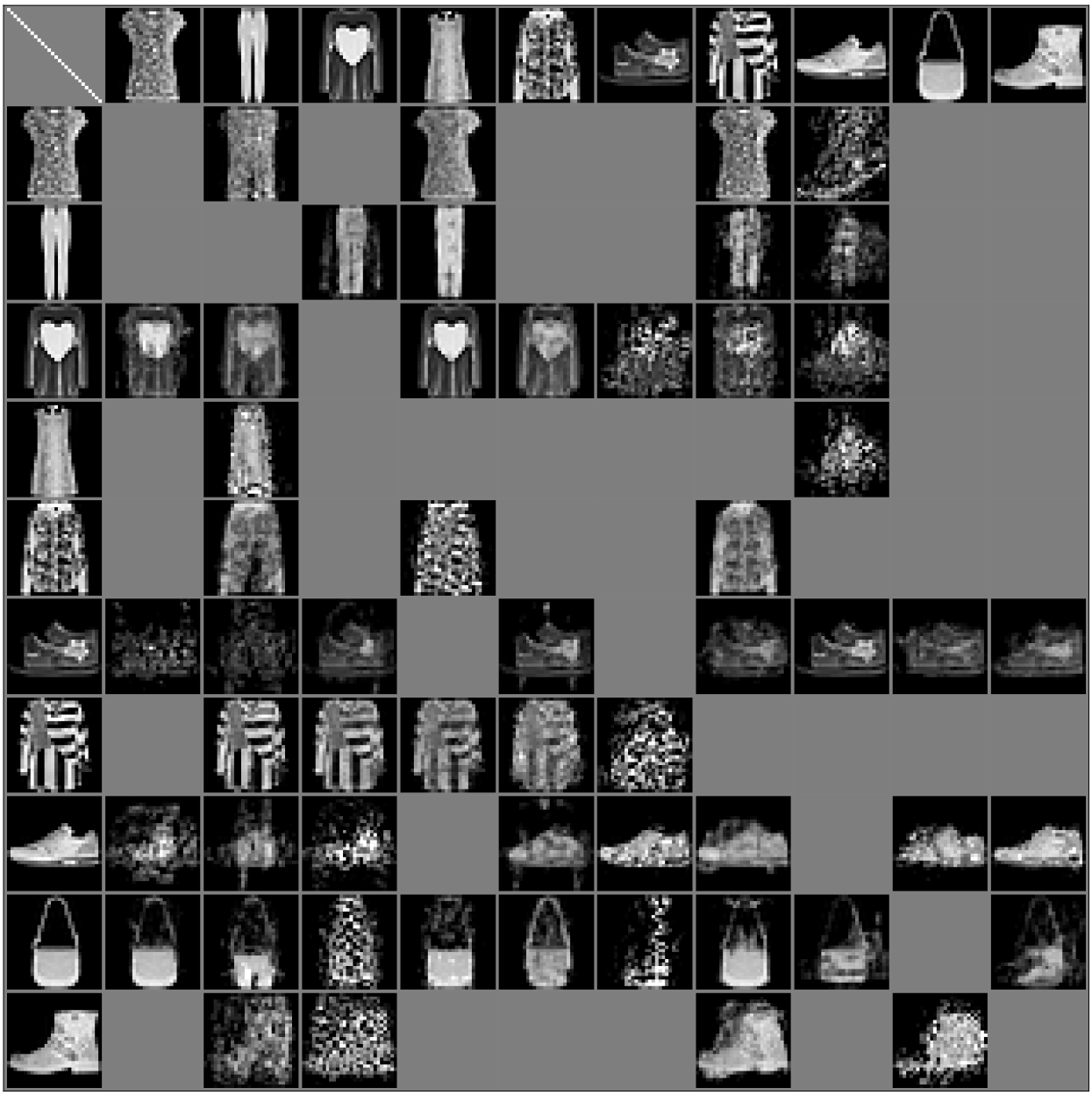
4.2.1 SHAP Explanations



4.2.2 Simple Counterfactuals



4.2.3 Prototype Guided Counterfactuals



References

[1] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[2] Lisa Anne Hendricks, Kaylee Burns, Kate Saenko, Trevor Darrell, and Anna Rohrbach. Women also snowboard: Overcoming bias in captioning models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 771–787, 2018.

[3] Arnaud Van Looveren and Janis Klaise. Interpretable counterfactual explanations guided by prototypes. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 650–665. Springer, 2021.

[4] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[5] Marco Tulio Ribeiro, Sameer Singh, and Carlos 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.

[6] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International conference on machine learning*, pages 3145–3153. PMLR, 2017.

[7] Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harv. JL & Tech.*, 31:841, 2017.