Chapter 6
Explainable Deep Learning

Recent advances in deep learning have made tremendous progress in the adoption of neural network models for tasks from resource utilization to autonomous driving. Most deep learning models are opaque black-box models that are not easily explainable. Unlike linear models, the weights of a neural network are not inherently interpretable to humans. The need for explainable deep learning has led to the development of a variety of methods that can help us better understand the decisions and decision-making process of neural network models. We note that many of the general post-hoc model-agnostic methods presented in Chap. 5 are applicable to deep learning models. This chapter presents a collection of explanation approaches that are specifically developed for neural networks by leveraging architecture or learning method.

6.1 Applications

The need for explainable deep learning is being driven by many real-world needs and applications. We discuss three broad categories: model validation, debugging, and exploration.

1. **Model Validation**: Model validation is the task of assessing how well a model behaves as intended in the real world. By doing so, we can assess the effectiveness and accuracy of a model. Explainable AI techniques can help us understand when errors in prediction occur and why they occur.
2. **Model Debugging**: Sometimes, a model may behave as intended but possess hidden biases. In recent years, we have seen real-life consequences of data and model biases. Our understanding of the weaknesses and limitations of deep learning models is vital for their adoption. Model debugging through explainable AI can help us uncover these deficiencies and provide insights into solutions to overcome them.

U. Kamath, J. Liu, *Explainable Artificial Intelligence: An Introduction to Interpretable Machine Learning*, https://doi.org/10.1007/978-3-030-83356-5_6

3. **Model Exploration**: With the exponential growth in complexity of recent deep learning models (for instance, GPT-3 consists of 96 layers and 175 billion parameters), we have arrived at a point where we are unsure of what hidden abilities these models possess. Model exploration is the task of assessing the performance of a model on tasks beyond what it was originally intended. Explainable AI has become essential in order for us to understand the capabilities of these deep models through model exploration.

6.2 Tools and Libraries

Table 6.1 provides details of all the libraries used for various models in the chapter.

Table 6.1 Models and implementations

Model/Algorithm	Library
Attention (NMT)	TensorFlow
Attention (image captioning)	TensorFlow
LIME	Captum (PyTorch)
Occlusion	Captum (PyTorch)
RISE	Keras
Activation Maximization	tf-keras-vis
Saliency map	Captum (PyTorch)
DeepLIFT	Captum (PyTorch)
DeepSHAP	Captum (PyTorch)
Deconvolution	Captum (PyTorch)
Guided Backprop	Captum (PyTorch)
Integrated Gradients	Captum (PyTorch)
Layer-wise relevance propagation	Captum-0.4.0 (PyTorch)
Excitation backpropagation	excitationbp (PyTorch)
GradCAM	Captum (PyTorch)
TCAV	Captum (PyTorch)

6.3 Intrinsic

Intrinsic explainable deep learning methods leverage inherent model architecture to provide explanations of model predictions. Intrinsic methods encompass two types: attention-based and jointly trained multi-task models. One distinction between intrinsic and post-hoc explainable deep learning methods is that they can provide explanations even during training.

6.3.1 Attention

Attention-based neural networks mimic cognitive attention processes and can implicitly provide explanations of their output directly from the weights of their attention mechanism. The weights of the attention layer are learned during training. These weights inform which parts of the input feature space are "attended to" and influence the prediction. They provide a measure of feature importance and can be visualized via heatmaps, such as Fig. 6.1 for a word model or Fig. 6.2 for a visual classifier.

Attention mechanisms can provide useful feature-based explanations, but they are limited by the scope of the input space. Furthermore, they are interpretable only if the inputs are themselves interpretable. For intermediate representations in higher layers of a deep neural network, interpretability may be difficult. Serrato and Smith [SS19] have shown that attention weights may not necessarily correspond to importance or represent optimal explanations.

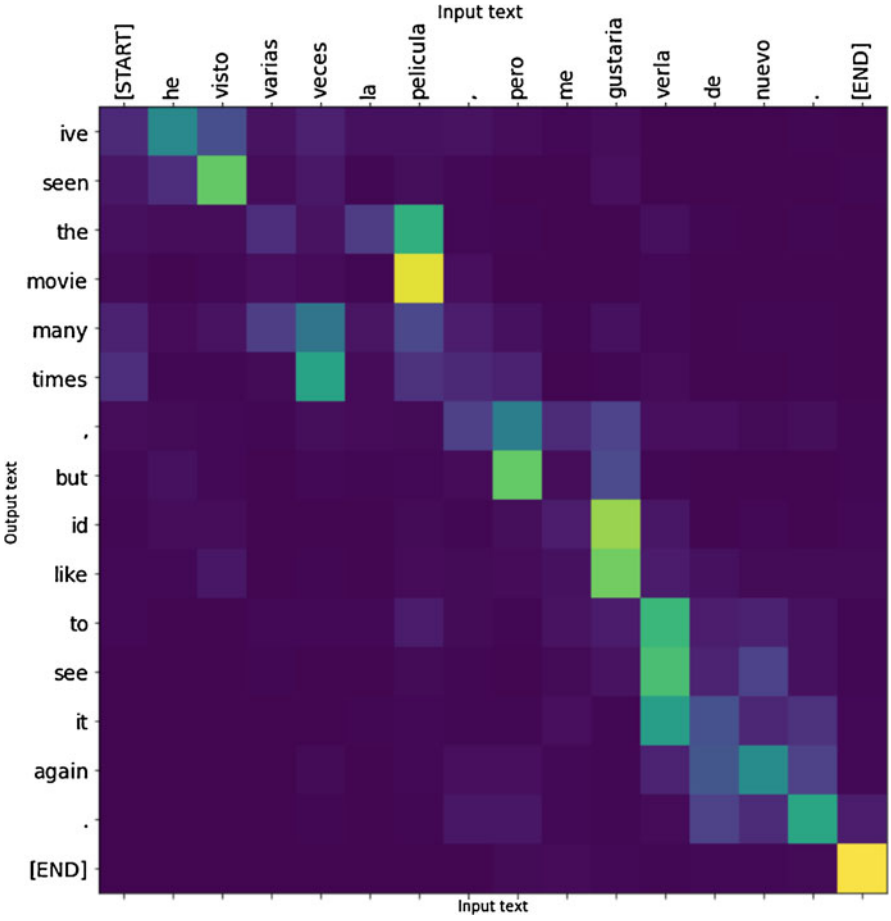

Fig. 6.1 Attention-based heatmap for word modeling

Interpretation of Attention Mechanisms: The attention weights of an attention-based model provide an intrinsic explanation to feature relevance on model predictions, but these weights may not provide optimal or interpretable explanations.

Attention-based neural networks are useful for tasks in NLP (RNNs and LSTMs), computer vision (CNNs), classification, and others.

Explainable properties of Attention-based methods are shown in Table 6.2.

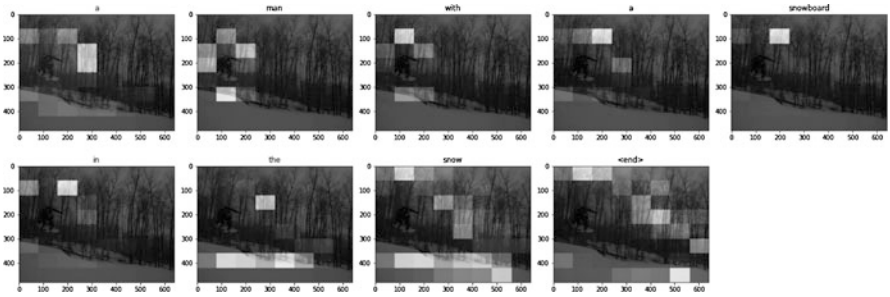

Fig. 6.2 Attention-based heatmap for visual classifier

Observations:

- Fig. 6.1 is a visualization of the attention weights for a TensorFlow Neural Machine Translation (NMT) model based on Bahdanau attention. The bi-LSTM model was trained on the Anki Spanish-to-English dataset for 5 epochs. Lighter colors indicate stronger weights. Word alignment (e.g., *movie* ⇔ *pelicula*, *seen* ⇔ *visto*) explanations are directly observable from the attention weights.
- Fig. 6.2 shows a sequence of blended attention masks for a neural image captioning system with a visual attention mechanism that shows what parts of the image the model focuses on as it generates a caption. The TensorFlow CNN with soft attention model was trained on the MS-COCO dataset for just 10 epochs. Note that the visual attention mechanism correctly attended to "man," but incorrectly attended to "snowboard."

Table 6.2 Explainable properties of attention-based methods

Properties	Values
Local or global	Local
Linear or non-Linear	Non-linear
Monotonic or non-monotonic	Non-monotonic
Feature interactions captured	Yes
Model complexity	Low

6.3.2 Joint Training

Another intrinsic method is to adopt a multi-task approach where an additional task is jointly trained to provide model explanations. This task can be trained to provide text-based justifications [LYW19, Zel+19], heatmaps over the feature [LBJ16, Iye+18] or concept space [AJ18, Don+17], or model prototypes [Li+17, Che+19]. Figure 6.3 illustrates an example of how a DNN architecture can be augmented with an explanation task that is jointly trained.

Joint training is a flexible method that enables high quality explanations at the cost of computational complexity due to changes in model architecture. The explanation task often comes a greater need for more data, and it may require explanation annotations in the training data in order to be trained with supervision.

Explainable properties of Joint-Training methods are shown in Table 6.3.

Interpretation of Joint Training Tasks: Augmenting network architecture to jointly train an explanation task is very flexible and can provide high quality explanations but can impose a heavy computational burden and requires training data with explanation annotations.

Observations:

- High quality explanations can be generated by jointly training a classifier against a gold set of explanations, but it can add significant complexity and computational cost.
- Explanation quality can be evaluated by calculating an explanation factor based on the explanation classifier output and target model output.

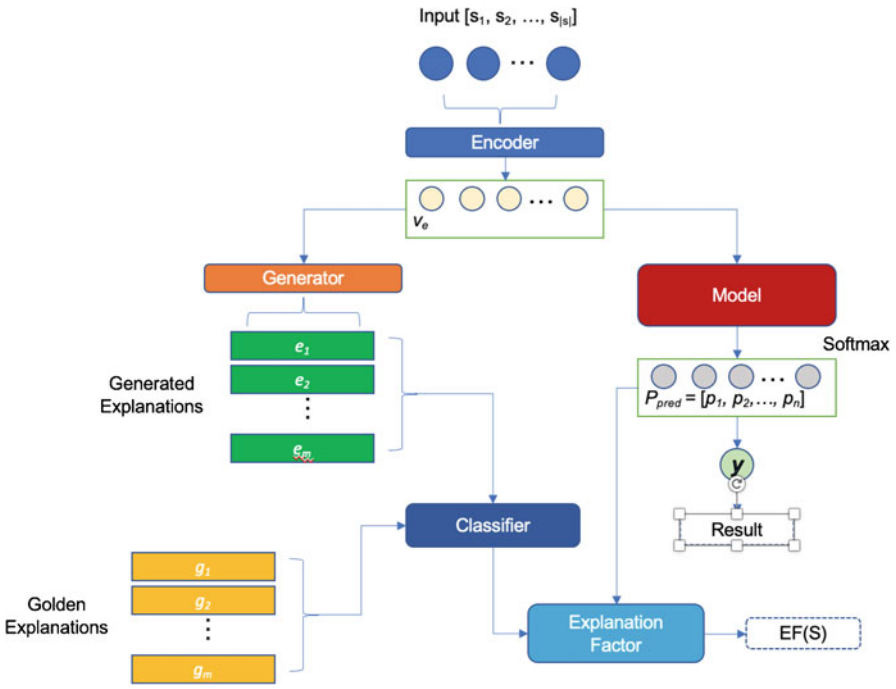

Fig. 6.3 Joint training for explanations

Table 6.3 Explainable properties of joint-training methods

Properties	Values
Local or global	Both
Linear or non-Linear	Both
Monotonic or non-monotonic	Both
Feature interactions captured	Yes
Model complexity	High

6.4 Perturbation

Perturbation methods attempt to explain feature relevance by measuring changes in prediction score as features are altered. Perturbations at a feature level include replacing, omitting individual features or groups, and learning attribution masks that can explain the contributions of features.

6.4.1 LIME

As noted in Chap. 5, surrogate explanation methods replace complex models with simpler models that approximate the predictions of the original model. For neural networks, this process is known as "model distillation" where the knowledge encoded in a neural net is distilled into an interpretable machine learning model that can mimic its behavior [HVD15]. Explanation of the original neural network is provided through this interpretable model.

Local Interpretable Model-agnostic Explanations (LIME) is a useful method for generating local explanations of a model for specific instances. As LIME is model-agnostic, it is applicable to a variety of neural networks. LIME maps input data to an interpretable representation $x \rightarrow z = g(x)$, which is typically a binary vector used to represent the presence or absence of specific features in the input. For images or text, this could be the presence or absence of a patch of pixels or a set of words, respectively. It seeks to learn an interpretable model $h(z)$ by optimizing with the objective

$$\arg\min_{g \in \mathscr{G}} \mathscr{L}(f, g, \pi_x) + \Omega(g) \tag{6.1}$$

where π_x is a distance penalty between samples z and x, $\mathscr{L}$ is a measure of the unfaithfulness of g in imitating f in the local region defined by π_x, and Ω is a complexity penalty that ensures the learned model is not too complex. As an example, we can generate local explanations for a neural network image classifier by applying the following:

$$\begin{aligned} h(z) &= a_g^T x \\ \pi_x(z) &= \exp\left(-||x - z||^2/\sigma^2\right) \\ \mathscr{L}(f, g, \pi_x) &= \sum_{z,z'} \pi_x(z) \left(f(z) - h(z')\right)^2 \\ \Omega(h(z)) &= ||a_g|| \end{aligned}$$

As previously noted, LIME explanations require a large number of randomly perturbed samples to compute accurate local explanations of a complex model. The class of each of these samples must be first predicted by a forward pass of the complex model, which could add computational burden when explaining large neural networks.

Explainable properties of LIME are shown in Table 6.4.

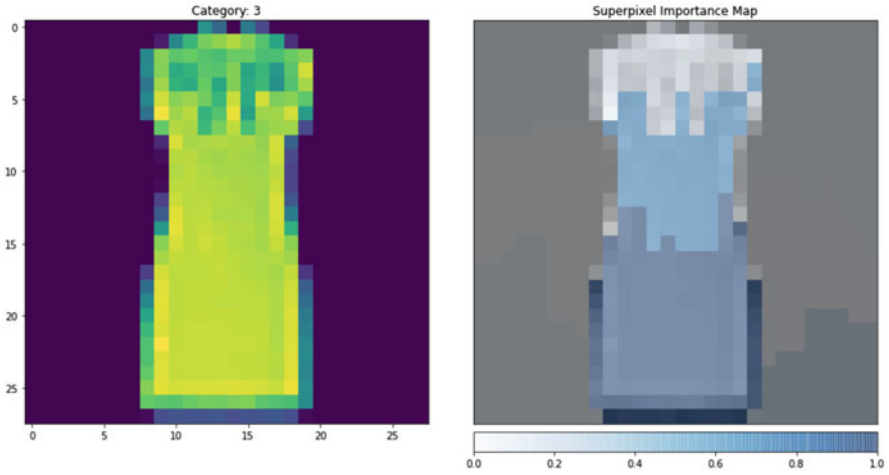

Fig. 6.4 LIME explanations on Fashion MNIST

Observations:

- The left side of Fig. 6.4 shows the model prediction from a PyTorch CNN model with three convolutional layers and two fully connected layers trained on Fashion MNIST for 2 epochs.
- LIME provides local explanations for image classification by computing the effect of the presence or absence of superpixels on the classification.
- The right side shows the importance of the superpixels (of a single color) on the prediction category 3 (Dress).

Table 6.4 Explainable properties of LIME

Properties	Values
Local or global	Local
Linear or non-linear	Both
Monotonic or non-monotonic	Both
Feature interactions captured	Yes
Model complexity	Medium

6.4.2 *Occlusion*

Perhaps one of the easiest ways to perturb an instance, occlusion (also named feature ablation) is a local method by which the input features of an instance are sequentially replaced with a constant (commonly zero). In 2013, [ZF13] proposed occlusion sensitivity as an explanation method for image classification by systematically occluding different portions of the input image with a gray patch sliding window. Feature relevance is measured by the change in prediction accuracy of the correct class or feature activation magnitude of the last neural network layer. This approach is applicable to other machine learning tasks as well. In 2017, [LMJ17] proposed an occlusion method for natural language-related tasks termed representation erasure, where input words are systematically erased to determine their contribution to prediction accuracy.

Interpretation of Occlusion: Occlusion methods are local explanations similar to ablation methods in Chap. 5, where input features are systematically replaced with a constant. The more prediction accuracy drops, the more significant the occluded features. Occlusion methods are computationally efficient but do not capture feature interaction well.

Like the feature ablation methods of Chap. 5, occlusion has limited ability to capture feature interaction effects. If interaction effects are significant, occlusion will likely return incorrect results. Unlike LOCO, occlusion is a local method that is easy to compute and does not require model retraining.

Explainable properties of Occlusion are shown in Table 6.5.

Observations:

- The left side of Fig. 6.5 shows two category 9 (shoe) classifier predictions by a PyTorch 3-convolutional layer CNN model trained on Fashion MNIST.
- The right side of Fig. 6.5 shows the occlusion importance maps generated by sliding a black 3x3 pixel mask across the image and measuring the resulting change in prediction probability.
- For both images, the importance maps indicate the diagonal area which most influence the classifier prediction for category 9.

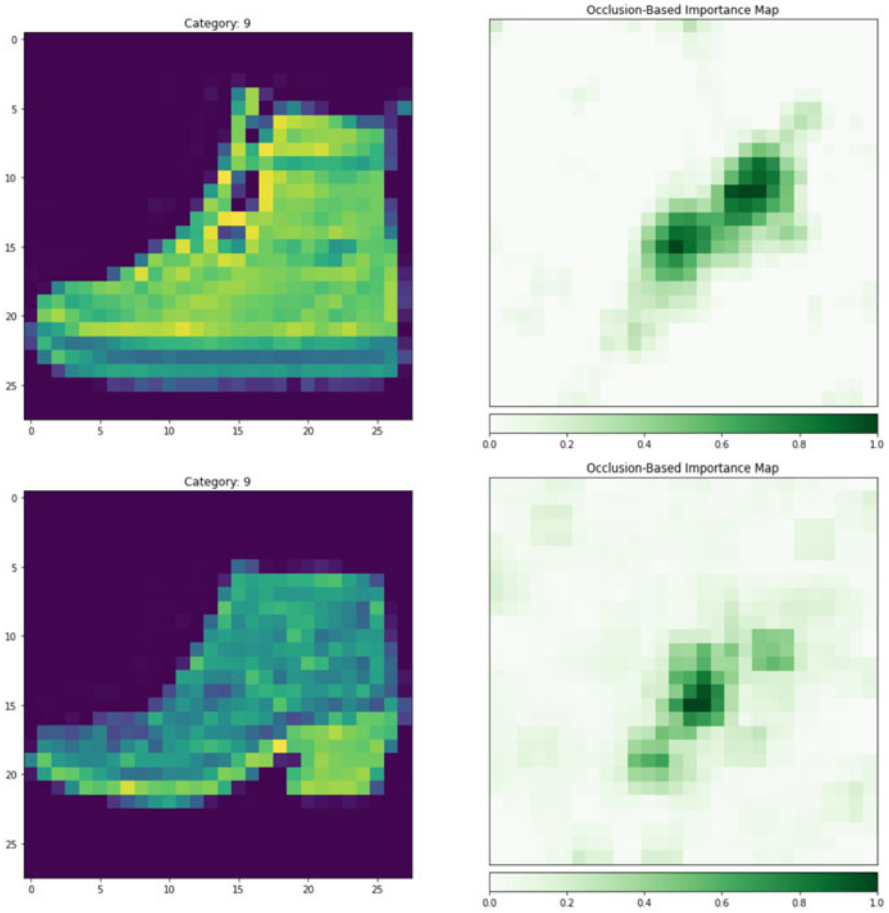

Fig. 6.5 Occlusion-based importance map on Fashion MNIST

Table 6.5 Explainable properties of occlusion

Properties	Values
Local or global	Local
Linear or non-linear	Non-linear
Monotonic or non-monotonic	Non-monotonic
Feature interactions captured	No
Model complexity	Low

6.4.3 RISE

In 2018, [PDS18a] proposed the Random Input Sampling for Explanations (RISE) method as generalized version of occlusion by probing a model with randomly masked portions of the input instance. Given a random mask M, an input instance

x, and a model $f(x)$, the feature importance of x_j(the j-th feature of x) is given by

$$S_f(x_j) = \mathbb{E}_M \left[f(x \star M) | M(x_j) = 1 \right] \tag{6.2}$$

where $\star$ denotes element-wise multiplication. Thus, RISE computes the importance map as the weighted average of random masks.

In practice, Monte Carlo sampling is used to generate random masks to compute RISE. As binary masks suffer when feature interactions exist, [PDS18a] proposed a soft version that up-samples a small binary mask using bilinear interpolation. The resulting mask values are continuous across [0, 1].

Explainable properties of RISE are shown in Table 6.6.

Interpretation of RISE: As a Monte Carlo sampled occlusion method, RISE is useful for generating importance map explanations of specific instances. By incorporating bilinear interpolation of smaller binary patches, RISE can take feature interactions into account.

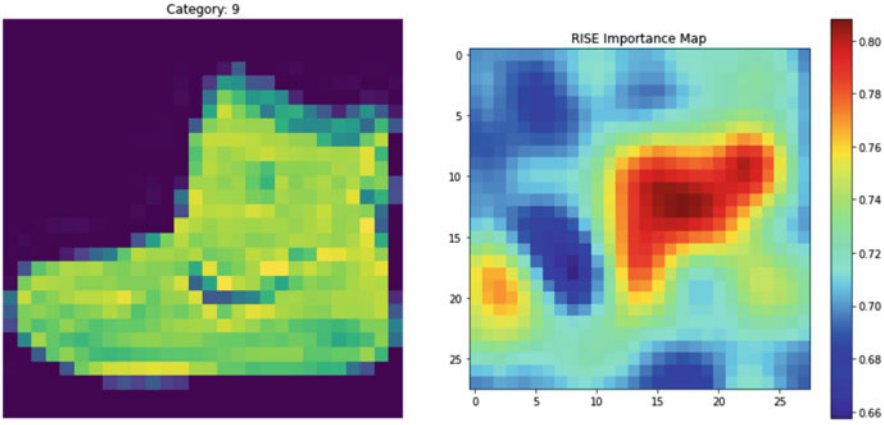

Fig. 6.6 RISE importance map on Fashion MNIST

Observations:

- The left image in Fig. 6.6 was classified as category 9 (shoe) by a 2-layer Keras CNN model trained on Fashion MNIST.
- The RISE importance map on the right side was computed using 2000 Monte Carlo generated random pixel maps.
- The right image shows higher importance around the ankle and toe area of the shoe image, while little to no importance with the rest of the image.
- RISE provides better importance maps than occlusion but at higher computation cost.

Table 6.6 Explainable properties of RISE

Properties	Values
Local or global	Local
Linear or non-linear	Non-linear
Monotonic or non-monotonic	Non-monotonic
Feature interactions captured	Yes
Model complexity	Medium

6.4.4 Prediction Difference Analysis

Prediction Difference Analysis (PDA) was proposed by [Zin+17] as a classifier explanation method to assign a relevance value to each input feature with respect to each class. It estimates this relevance by measuring how prediction changes if a feature value is unknown. Since for neural networks it is impractical to either label a feature as unknown or retrain the model with the feature left out (e.g., LOCO), PDA simulates the absence of the feature by marginalizing over it. Given a class c and an input instance x with j-th feature x_j, the class probability with unknown feature x_j is given by

$$p(c|x_{-j}) = \sum_{x_j} p(x_j|x_{-j})\, p(c|x_{-j}, x_j) \tag{6.3}$$

where x_{-j} is the set of all features in x except the j-th feature and the summation is taken over all possible values of x_j. For large feature spaces, computational efficiency can be gained by assuming feature x_j is uncorrelated with the other features x_{-j}, and the class probability becomes

$$p()c|x_{-j}) \approx \sum_{x_j} p(x_j)p(c|x_{-j}, x_j) \tag{6.4}$$

where $p(x_j)$ can be approximated by its empirical distribution. PDA compares the class probability with all features present $p(c|x)$ with $p(c|x_{-j})$ to determine feature relevance by defining a weight-of-evidence function:

$$WE_j(c|x) = \log_2 (odds(c|x)) - \log_2 \left(odds(c|x_{-j})\right) \tag{6.5}$$

where

$$odds(c|x) = \frac{p(c|x)}{1 - p(c|x)} \quad odds(c|x_{-j}) = \frac{p(c|x_{-j})}{1 - p(c|x_{-j})} \tag{6.6}$$

To account for zero probabilities, [Zin+17] proposed using a Laplace correction to the class probability:

$$p(c|x) \leftarrow \frac{p(c|x)n + 1}{n + k} \tag{6.7}$$

where N is the number of training instances and k is the number of classes. The magnitude of the evidence function WE_j indicates the significance of the j-th feature on class c prediction. A positive value of WE_j implies that feature x_j contributed positively to the evidence for class c, and removing it would reduce confidence in prediction for the class. A negative value implies evidence against the class.

When feature interactions exist, PDA can be adjusted to account for neighbor interactions. Instead of assuming feature x_j is uncorrelated with every other feature and replacing the conditional probability $p(x_j|x_{-j})$ with $p(x_j)$, [Zin+17] proposed using conditional sampling where a neighborhood patch of features around and including x_j is marginalized:

$$p(x_j|x_{-j}) \approx p(x_j|\hat{x}_{-j}) \tag{6.8}$$

Here, $\hat{x}_{-j}$ is the set of all features except for the patch of features around and including x_j.

PDA is also applicable for visualizing neuron contributions to hidden layer activations. Given a hidden layer H with neuron values h and the i-th neuron in the subsequent layer that depends on H with value $z_i(h)$, the activation function g when the j-th neuron value in H is unknown is given by

$$g(z_j|h_{-j}) = \sum_{h_j} p(h_j|h_{-j})z_i(h_{-j}, h_j) \tag{6.9}$$

and the activation difference AD_j is a measure of the contribution of the j-th neuron in hidden layer H to the i-th neuron in the subsequent layer:

$$AD_j(z_i|h) = g(z_i|h) - g(z_i|h_{-j}) \tag{6.10}$$

In practice, PDA is fairly computationally intensive, especially if conditional sampling is used to capture neighbor feature interactions.

Explainable properties of Prediction Difference Analysis are shown in Table 6.7.

Interpretation of Prediction Difference Analysis: PDA is a local method that measures feature relevance by taking the class prediction probability differences while marginalizing over a feature or a patch of features. It can account for feature interactions but comes with a larger computation cost.

Table 6.7 Explainable properties of prediction difference analysis

Properties	Values
Local or global	Local
Linear or non-linear	Non-linear
Monotonic or non-monotonic	Non-monotonic
Feature interactions captured	Yes
Model complexity	Medium

6.4.5 Meaningful Perturbation

Meaningful perturbation (MP) [FV17] is a local explanation method based on a framework of meta-predictors to explain predictions for neural classifiers. These meta-predictors are trained to predict the presence or absence of input features. Their prediction error is a measure of the faithfulness of the explanation.

For a given instance x_0, the method applies a set of meaningful, local perturbations given by

$$[\Phi(x_0, m)] = \begin{cases} m(u)x_0(u) + (1 - m(u))\mu_0, & constant \\ m(u)x_0(u) + (1 - m(u))\eta(u) & noise \\ \int g_{\sigma_0 m(u)}(v - u)x_0(v)dv & blur \end{cases} \tag{6.11}$$

where μ_0 is the average color, $\eta(u)$ are i.i.d. Gaussian noise samples, and σ_0 is the standard deviation of the Gaussian blur kernel g_σ. The method plays a "deletion game," which seeks to find the smallest deletion mask m^* that causes the classifier score f_c for class c to drop $f_c(\Phi(x_0, m)) < fc(x_0)$ by optimizing:

$$m^* = \underset{m \in [0,1]^d}{\arg\min}\ \lambda||1 - m||_1 + f_c(\Phi(x_0, m)) \tag{6.12}$$

where d is the total number of features and ϵ is a hyperparameter. A symmetric "preservation game" can also be played, which seeks to find the smallest subset of the image that must be retained to preserve the classifier score $f_c(\Phi(x_0, m)) \geq fc(x_0)$ by optimizing:

$$m^* = \underset{m \in [0,1]^d}{\arg\min}\ \lambda ||m||_1 + f_c(\Phi(x_0, m)) \tag{6.13}$$

The deletion game tries to remove just enough evidence to prevent the model from recognizing the class, while the preservation game ties to keep just enough evidence. Both of these optimizations can be solved by gradient descent.

To mitigate the effects of artifacts that might exist in the trained neural network, meaningful perturbations propose a modified deletion game where the learned mask is regularized:

$$m^* = \min_{m \in [0,1]^d} \lambda_1 ||1 - m||_1 + \lambda_2 \sum_u ||\nabla m(u)||_\beta^\beta + \mathbb{E}_\tau \left[f_c(\Phi(x_0(\cdot - \tau), m)) \right] \tag{6.14}$$

This optimization can be solved with stochastic gradient descent.

Interpretation of Meaningful Perturbations: MP is a global method that learns where a neural classifier looks by discovering features that most affect its class prediction output score when locally perturbed. It learns a feature mask that explains the classification result as an optimization problem.

In practice, the algorithm learns the smallest, low-resolution, sparse set of masks, which, when up-sampled and added to the input instance, causes the target class prediction to drop.

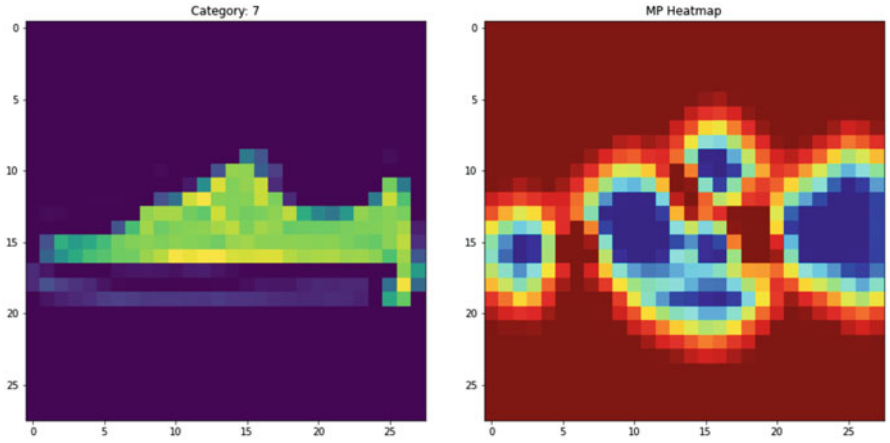

Fig. 6.7 Meaningful perturbations mask on Fashion MNIST

Explainable properties of Meaningful Perturbation are shown in Table 6.8.

Observations:

- The left image in Fig. 6.7 shows the category predictions of a PyTorch 3-layer CNN model trained on Fashion MNIST.
- The right image shows the meaningful perturbations heatmap for the prediction of category 7 (Sneaker).

Table 6.8 Explainable properties of meaningful perturbation

Properties	Values
Local or global	Local
Linear or non-linear	Non-linear
Monotonic or non-monotonic	Non-monotonic
Feature interactions captured	Yes
Model complexity	Medium

6.5 Gradient/Backpropagation

Whereas perturbation-based explanation methods leverage variations in input features to explain feature relevance, gradient methods leverage the flow of information during backpropagation to explain the relationship between input features and network output. Gradient-based methods typically provide visual explanations through heatmaps of neuron or feature attributions.

6.5.1 Activation Maximization

Visual explanations provide an efficient and human-interpretable method to understand deep neural network predictions. One of the earliest global explanation methods is the Activation Maximization method [ECB10], which visually identifies the input features that can create the greatest response in the output of specific neurons.

Given a neural network with parameters θ, an input sample x, and the i-th neuron in the j-th layer with activation $h_{ij}(\theta, x)$, the goal of is to find a hypothetical x^* that can maximize the activation of this neuron. This can be expressed as the optimization

$$x^* = \arg\max_x h_{ij}(\theta, x) \tag{6.15}$$

which can be solved using gradient ascent in the input space. It is similar to the backpropagation method, except instead of adjusting network parameters θ, the

optimization is over the input space while the network parameters are held constant. The synthetic instance x^* can be visualized and will represent the input feature pattern that will maximize the activation of a specific neuron in the network.

> Interpretation of Activation Maximization: AM is a global method that finds the input pattern that can generate the highest activation in the response of a specific neuron in a deep neural network.

Given the non-linear activations, there are no guarantees that gradient ascent will identify a unique global optimum x^*, but in practice using multiple random starting points and either averaging or selecting the maximum activation has been shown to be effective.

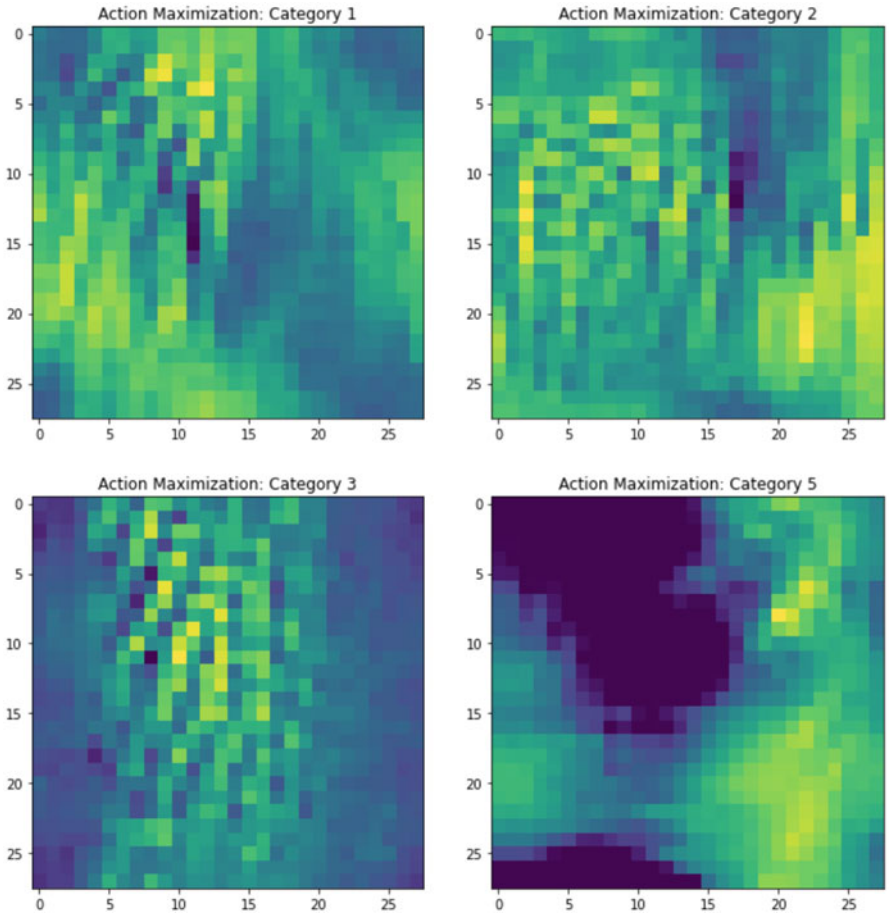

Fig. 6.8 Activation maximization map on Fashion MNIST

Explainable properties of Activation Maximization are shown in Table 6.9.

Observations:

- Fig. 6.8 shows the activation maximization maps for 4 neurons at the output layer of a Keras 2-layer CNN model trained on Fashion MNIST. These neurons correspond to 4 classification categories.
- Note that it is difficult to determine the original classification category from the activation maps (1=Trouser, 2=Pullover, 3=Dress, 5=Sandal).

Table 6.9 Explainable properties of activation maximization

Properties	Values
Local or global	Global
Linear or non-Linear	Non-linear
Monotonic or non-monotonic	Non-monotonic
Feature interactions captured	Yes
Model complexity	Low

6.5.2 Class Model Visualization

Activation maximization is the basis of class model visualizations. Given a neural network classifier with scoring function $S_c(x)$ for an output class c and input x, it is possible to learn an instance x' that is most representative of the class by optimizing the equation

$$x' \leftarrow \arg\max_x S_c(x) - \lambda ||x||_2 \tag{6.16}$$

where λ is a regularization parameter. The generated instances for each class are learned representations by the neural network and can be very visually entertaining.

Interpretation of Class Model Visualization: this global explanation method learns the input patterns that generate the greatest activation for a specific model class. When visualized, these patterns can provide colorful explanations of what the model has learned.

Class model visualizations have recently gained widespread attention for pioneering a new branch of deep learning-generated art called "deep dream" and "Inceptionism" based on the colorful visualizations of model classes.

Explainable properties of Class Model Visualization are shown in Table 6.10.

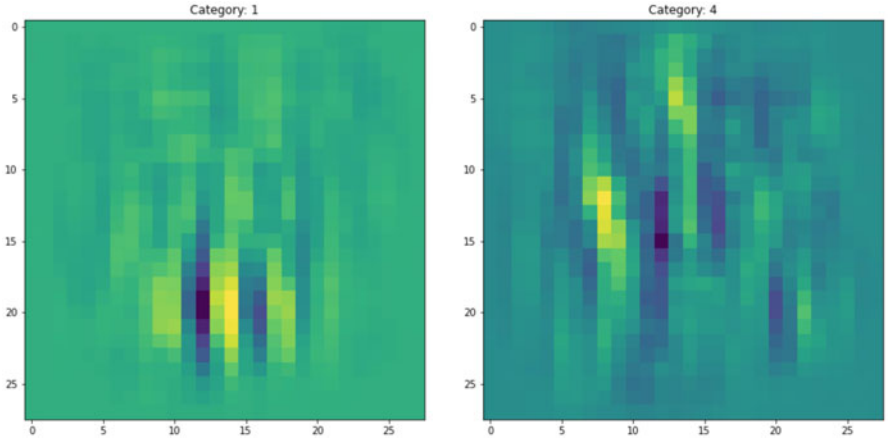

Fig. 6.9 Class model visualization on Fashion MNIST

Observations:

- Fig. 6.9 shows the model visualizations for category 1 (Trouser) and category 4 (Coat) from a PyTorch 3-layer CNN model trained on Fashion MNIST.
- When viewed at a distance, the left image hints at a pair of trousers, while the right image gives the semblance of a coat.

Table 6.10 Explainable properties of class model visualization

Properties	Values
Local or global	Global
Linear or non-linear	Non-linear
Monotonic or non-monotonic	Non-monotonic
Feature interactions captured	Yes
Model complexity	Low

6.5.3 Saliency Maps

Saliency maps [SVZ14a] provide local explanations for specific instances. Given an instance of interest x_0, we can approximate the non-linear scoring function $S_c(x)$ by

a Taylor series expansion around this instance:

$$S_c(x) \approx w^T x + b \tag{6.17}$$

where w are the saliency weights

$$w = \left. \frac{\partial S_c}{\partial x} \right|_{x=x_0} \tag{6.18}$$

The saliency map M_j for the j-th feature is given by

$$M_j = |w_j| \tag{6.19}$$

Interpretation of Instance-Specific Saliency Maps: Instance-specific saliency maps are a local explanation method that takes the partial derivative of the scoring function with respect to each feature as a measure of feature importance. They are very quick to calculate but require the scoring function to be differentiable.

Instance-specific class saliency maps are extremely quick to compute and do not require any additional annotation to provide explanations. They do, however, require the scoring function to be differentiable.

Explainable properties of Saliency Maps are shown in Table 6.11.

Observations:

- The left side of Fig. 6.10 shows the category predictions of a PyTorch 3-layer CNN model trained on Fashion MNIST.
- The right side shows the saliency maps for an image of category 0 (t-shirt) and category 7 (sandal).

Table 6.11 Explainable properties of saliency maps

Properties	Values
Local or global	Local
Linear or non-linear	Linear
Monotonic or non-monotonic	Monotonic
Feature interactions captured	No
Model complexity	Low

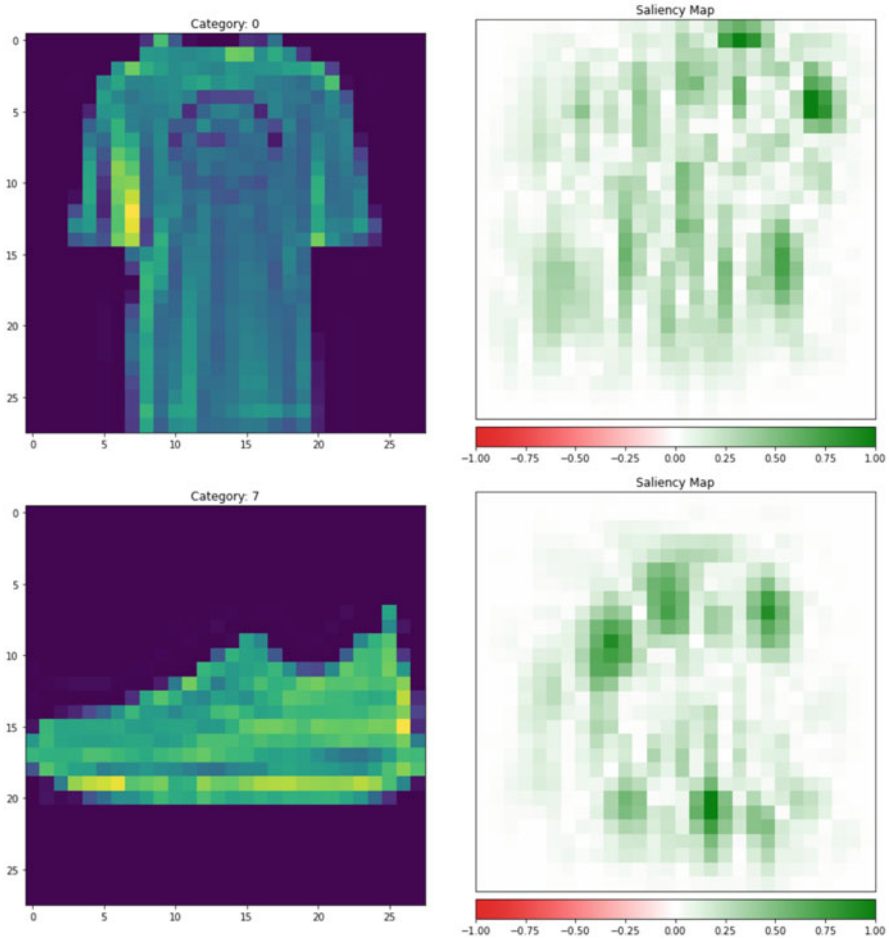

Fig. 6.10 Saliency maps for instances of Fashion MNIST

6.5.4 DeepLIFT

Deep Learning Important Features (DeepLIFT) [SGK19] is a recursive, local explanation method that decomposes a neural network model prediction for a specific instance by backpropagating the contributions of the neurons through the network. DeepLIFT is based on the difference between the activation of each neuron and its "reference activation" in order to compute contribution scores. This reference activation represents a default or a neutral input.

Consider a neuron of interest with activation $f(x)$ and with a set of input neurons $x_1, x_2, \ldots, x_n$. If $f(x')$ represents the reference activation of the neuron of interest

for a reference input x', the difference in neural activation is given by

$$\Delta t = f(x) - f(x') = \sum_{i=1}^{n} C_{\Delta x_i \Delta t} \tag{6.20}$$

where n is the total number of input neurons necessary to compute $f(x)$ and $\Delta x = x - x'$. This is termed the "summation-to-delta" property. The contribution score $C_{\Delta x_i \Delta t}$ relates how changes in input Δx affect changes in neuron activation Δt. If we divide this contribution score by Δx, we can define a multiplier analogous to a partial derivative:

$$m_{\Delta x \Delta t} = \frac{C_{\Delta x \Delta t}}{\Delta x} \tag{6.21}$$

This multiplier follows a useful chain rule:

$$m_{\Delta x_i \Delta t} = \sum_{j} m_{\Delta x_i \Delta y_j} m_{\Delta y_j \Delta t} \tag{6.22}$$

This chain rule allows for the contribution scores to be backpropagated layer-by-layer through the network and is analogous to how gradients are backpropagated. By using the difference from reference approach, DeepLIFT allows contribution scores to propagate even when the gradient is zero.

DeepLIFT proposes three contribution scoring functions: a linear rule applicable to dense and convolutional layers, a rescale rule that can account for saturation and thresholding problems, and a reveal-cancel rule that treats positive and negative contributions separately.

The choice of a reference input x' is an important consideration, as it determines what relevance scores are computed against. For instance, the use of an all zero reference may not be as useful if noise is present in the background.

Interpretation of DeepLIFT: as a local explanation method, DeepLIFT calculates input importance relative to a reference by backpropagating contribution scores through the network. It is very computationally efficient and provides an approximation to Shapley values. The choice of scoring function and reference input should be carefully considered.

DeepLIFT scores can be efficiently computed with a single backward pass. They are connected to Shapely values, which measure the marginal contribution of each feature averaged across the set of all possible coalitions of features. If excluding a feature is equivalent to setting it to its reference value, DeepLIFT can be thought of as a fast approximation of the Shapely values.

Explainable properties of DeepLIFT are shown in Table 6.12.

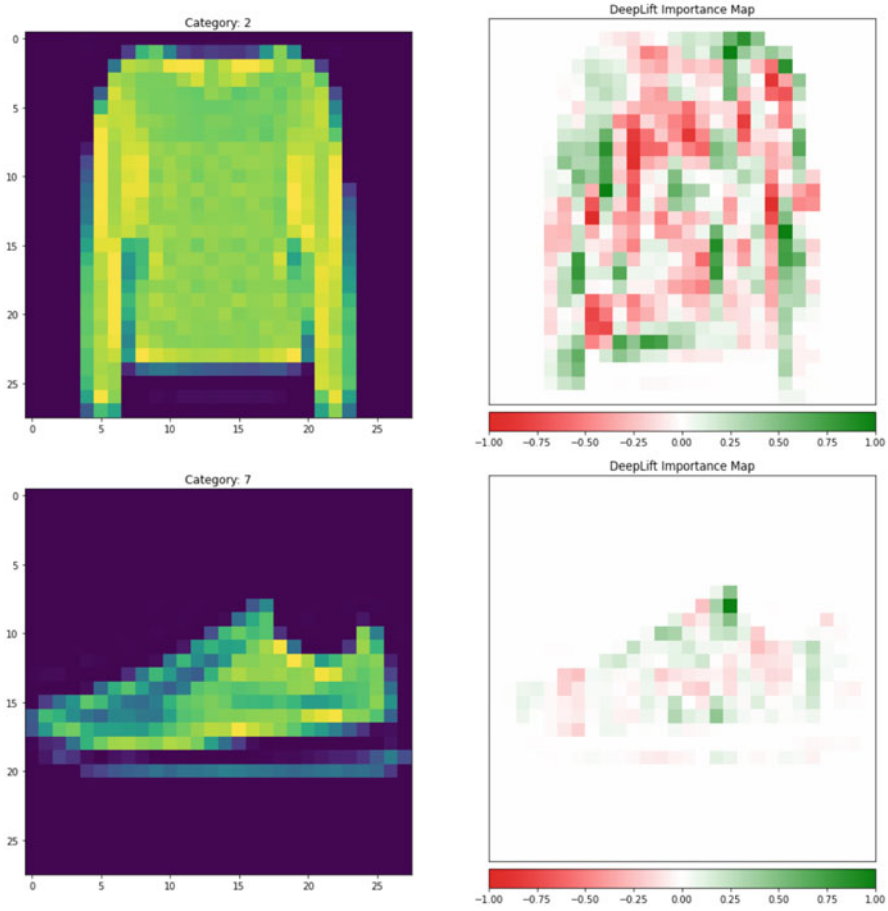

Fig. 6.11 DeepLIFT contribution scores on Fashion MNIST

Observations:

- The left side of Fig. 6.11 shows the category predictions of a PyTorch 3-layer CNN model trained on Fashion MNIST.
- The right side shows the DeepLIFT importance map based on the linear-rule and black baseline reference.
- Note that DeepLIFT provides better visual explanations in comparison to saliency maps.

Table 6.12 Explainable properties of DeepLIFT

Properties	Values
Local or global	Local
Linear or non-linear	Non-linear
Monotonic or non-monotonic	Non-monotonic
Feature interactions captured	Yes
Model complexity	Low

6.5.5 DeepSHAP

DeepSHAP [CLL19] is an extension of the KernelSHAP method of Chap. 5 by leveraging the compositional architecture of deep neural networks to improve computational efficiency. As previously mentioned, the per node attribution rules in DeepLIFT can approximate the Shapley values [SGK19]. DeepSHAP leverages this approximation as well as DeepLIFT's multiplier chain rule. In DeepSHAP, the multipliers are expressed in terms of SHAP values ϕ_i:

$$m_{x_j,f_j} = \frac{\phi_i(f_j, x)}{x_j - \mathbb{E}\left[x_j\right]} \tag{6.23}$$

and follow the chain rule:

$$m_{x_j,f_j} = \sum_j m_{x_j,y_j} m_{y_j,f_j} \tag{6.24}$$

DeepSHAP calculates SHAP values for large networks by starting with Shapley values for simple network components and backpropagating them using this rule.

Rather than setting the reference input x' as in DeepLIFT, DeepSHAP approximates the reference value by averaging over background dataset instances. It can estimate approximate SHAP values such that they sum up to the difference between the expected model output on background instances and the current model output $f(x) - \mathbb{E}[f(x)]$.

> Interpretation of DeepSHAP: as a local explanation method, DeepSHAP is an extension of DeepLIFT to backpropagate Shapley values through the network. DeepSHAP computes an input reference as the expectation over background data instances. It is very computationally efficient and provides a quick approximation to Shapley values, which may be biased when features are strongly correlated.

DeepSHAP is computationally very efficient. Instead of depending on DeepLIFT's contribution rules to linearize each node in the network, DeepSHAP effectively linearizes the network by computing SHAP values using the chain rule.

As such, they are approximations to the true Shapley values and will be biased when strong feature interactions exist.

Explainable properties of DeepSHAP are shown in Table 6.13.

Observations:

- The left side of Fig. 6.12 shows the predictions of a PyTorch 3-layer CNN model trained on Fashion MNIST.
- The right side shows the DeepSHAP importance map based on a baseline reference consisting of a random sample of 10 images from the training set.
- Note that DeepSHAP provides slightly worse visual explanations in comparison to DeepLIFT but is more computationally efficient.

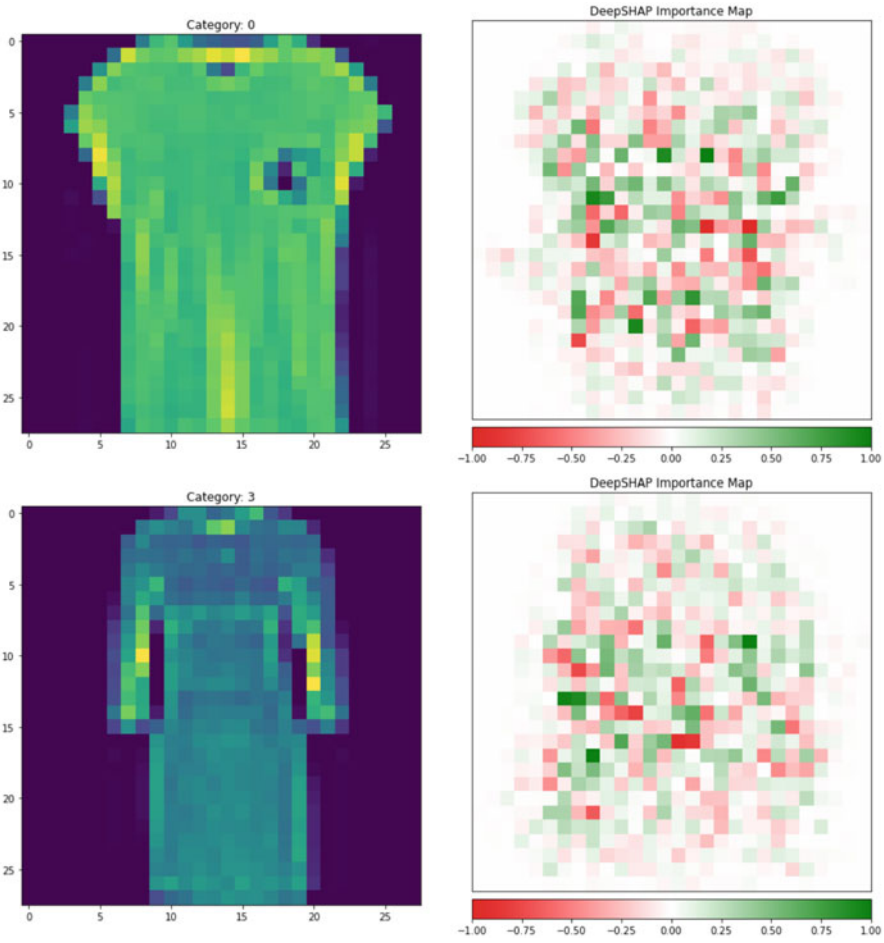

Fig. 6.12 DeepSHAP importance map on Fashion MNIST

Table 6.13 Explainable properties of DeepSHAP

Properties	Values
Local or global	Local
Linear or non-linear	Linear
Monotonic or non-monotonic	Monotonic
Feature interactions captured	No
Model complexity	Low

6.5.6 Deconvolution

Deconvolution [ZF13] is an explanation method proposed for visualizing the feature contributions of CNN architectures (convnets). It takes the output of the CNN and runs the CNN in reverse. The guiding notion is to determine which portions of an input instance are most discriminative for a single neuron.

Given a CNN neural network of J layers, the output of the j-th layer C^j is given by

$$A^j = C^{j-1} * K^j + b^j \tag{6.25}$$

$$B^j = ReLU(A^j) = \max\left(A^j, 0\right) \tag{6.26}$$

$$C^j = \text{maxpool}\left(B^j\right) \tag{6.27}$$

$$s^j = \text{switch}\left(B^j\right) \tag{6.28}$$

where K^j and b^j are the learned filter and bias for the j-th layer, respectively. ReLU is the rectified linear operator, and the switch variable s^j records the indices of the maximum values in the pooling operation for the deconvolution step.

A deconvolution network (deconvnet) is attached to the original convnet to map feature activations back to the input space. Each layer in this deconvnet inverts the corresponding layer of the original convnet. To examine each convnet neuron activation, the activation is set to zero for all other neurons in the layer, and the feature maps are fed as input to the deconvnet layer, which sequentially applies unpooling, rectification, and filtering operations.

$$\hat{C}^j = \text{unpool}\left(C^j, s^j\right) \tag{6.29}$$

$$\hat{B}^j = ReLU(\hat{C}^j) = \max\left(\hat{C}^j, 0\right) \tag{6.30}$$

$$\hat{A}^j = \left(\hat{B}^j - b^j\right) * K^{jT} \tag{6.31}$$

where K^{jT} is the transpose of K^j. Together, the set of deconvolution operations is called transpose convolution. While the max pooling operation is not invertible, unpooling can be performed if switch variables are recorded during the forward propagation of the convnet. The ReLU ensures feature maps are non-negative, and the filtering operation up-weights and up-scales the feature representation in each layer.

Interpretation of Deconvolution: deconvolution explains learned feature maps in CNN-based models by propagating them through an inverted convolutional network called a deconvnet.

The deconvolution method is specific to CNN architectures, though the process is applicable to dense layers as well and other non-linearities beyond ReLU. In order to operate effectively, deconvolution requires a forward pass through the original convnet in order to calculate and store the switch variables to allow the deconvnet to reverse the max pooling operations. As a result, visualizations derived from deconvolution are conditioned on the instance used to calculate the switch variables and do not directly visualize the learned features [Spr+15b].

Explainable properties of Deconvolution are shown in Table 6.14.

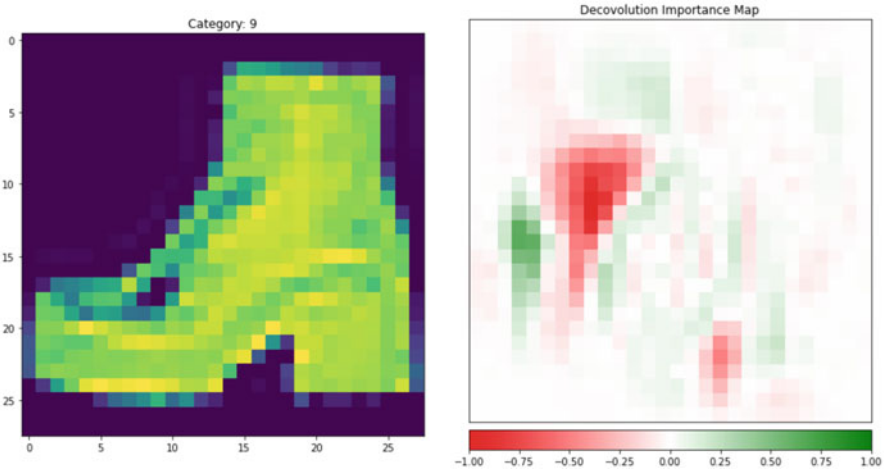

Fig. 6.13 Deconvolution visualizations on Fashion MNIST

Observations:

- The left side of Fig. 6.13 shows the input image and category prediction of a PyTorch 3-layer CNN model trained on Fashion MNIST.
- The right side shows the deconvolution feature importance map conditioned on the input image instance.

6.5.7 Guided Backpropagation

Guided backpropagation [Spr+15b] is another explanation method for feature attribution on CNN-based architectures. It is similar to deconvolution, except it removes the unpooling operation and adopts a modified operation for the ReLU non-linearity.

Table 6.14 Explainable properties of deconvolution

Properties	Values
Local or global	Global
Linear or non-linear	Non-linear
Monotonic or non-monotonic	Non-monotonic
Feature interactions captured	Yes
Model complexity	Medium

In deconvolution, only positive gradients are backpropagated. In vanilla backpropagation, the gradients for only positive inputs are kept for each layer. Guided backpropagation incorporates both of these methods such that only positive gradients associated with positive input values are backpropagated. This stops the backward flow of negative gradients through the inverse network.

Let f_i^j and R_i^j be the i-th neuron input to and feature map of the j-th layer, respectively. Then for guided backpropagation, we have during the backward pass

$$R_i^j = (f_i^j > 0)(R_i^{j+1} > 0)\ R_i^{j+1} \tag{6.32}$$

This is in contrast to deconvolution, where the backward pass applies the operation after unpooling:

$$R_i^j = (R_i^{j+1} > 0)\ R_i^{j+1} \tag{6.33}$$

Because of the additional guidance signal in guided backpropagation, unpooling is unnecessary and does not require an initial forward pass through the convnet to

compute and store switch variables. As a result, it is more computationally efficient. Explanations via guided backpropagation are not conditioned on any single instance as in deconvolution and provide more accurate explanations of feature activations.

Explainable properties of Guided Backpropagation are shown in Table 6.15.

Interpretation of Guided Backpropagation: guided backprop is an improvement upon deconvolution to explain learned feature maps in CNN-based models. It replaces unpooling and ReLU operations with an operation allow only positive gradients associated with positive inputs to be backpropagated.

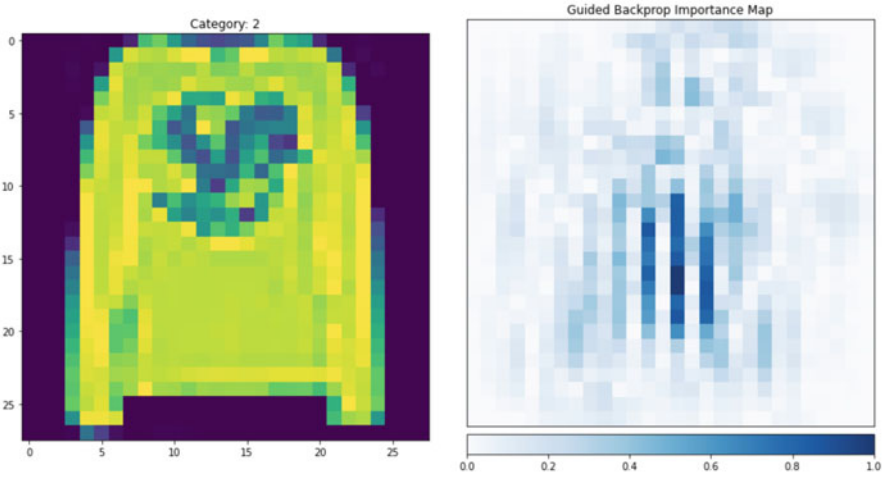

Fig. 6.14 Guided backpropagation visualizations on Fashion MNIST

Observations:

- The left side of Fig. 6.14 shows the category prediction of a PyTorch 3-layer CNN model trained on Fashion MNIST.
- The right side shows the guided backprop feature importance map without the need for a forward pass for conditioning as in deconvolution.
- Note the improvement over deconvolution in visual explanation.

Table 6.15 Explainable properties of guided backpropagation

Properties	Values
Local or global	Global
Linear or non-linear	Non-linear
Monotonic or non-monotonic	Non-monotonic
Feature interactions captured	Yes
Model complexity	Medium

6.5.8 Integrated Gradients

Integrated gradients [SN19] is an explanation method that does not require modification of the original network and takes an axiomatic approach to generate feature attributions for specific instances. Ideally, an attribution method should obey the axiom of sensitivity, which states that if two inputs x and x' differ by a single feature and have different prediction values $f(x) \neq f(x')$, then the feature should be given non-zero attribution. Unfortunately, many gradient-based attribution methods violate this sensitivity axiom, including DeepLIFT, deconvolution, and guided backpropagation.

An attribution method should also obey the axiom of implementation invariance, which states that two neural networks, even with vastly different implementations, are functionally equivalent if they map the same inputs to the same outputs. Attributions are identical across two functionally equivalent neural networks.

The integrated gradients method satisfies both of these axioms. Given a neural network model f, an input instance x, and a baseline reference x', one can traverse along the direct path from reference to input $x' \rightarrow x$ while accumulating gradients. The integrated gradient along the j-th feature is given by

$$IG_j(x) = (x_j - x'_j) \int_{\alpha=0}^{1} \frac{\partial f\left(x' + \alpha\left[x - x'\right]\right)}{\partial x_j} \partial\alpha \tag{6.34}$$

Integrated gradients satisfy the axiom of completeness, which states that the sum of the attributions is equal to $f(x) - f(x')$.

A key consideration with integrated gradients is selecting a baseline, similar to DeepLIFT. Ensuring that the baseline reference has near-zero score is important to ensure the attributions are derived from the input rather than the baseline.

In practice, the path integral is calculated using a Riemann summation approximation:

$$IG_j(x) \approx \frac{(x_j - x'_j)}{M} \sum_{i=1}^{M} \frac{\partial f\left(x' + \frac{i}{M}(x - x')\right)}{\partial x_j} \tag{6.35}$$

where the parameter M is the number of steps in the Riemann summation.

Interpretation of Integrated Gradients: IG takes an axiomatic approach to computing feature attributions by accumulating gradients along the direct path between a baseline reference and an instance of interest. They are simple and quick to compute.

The integrated gradients method is not limited to CNNs and can be applied to a wide variety of deep neural networks. They are computationally efficient and simple to compute.

Explainable properties of Integrated Gradients are shown in Table 6.16.

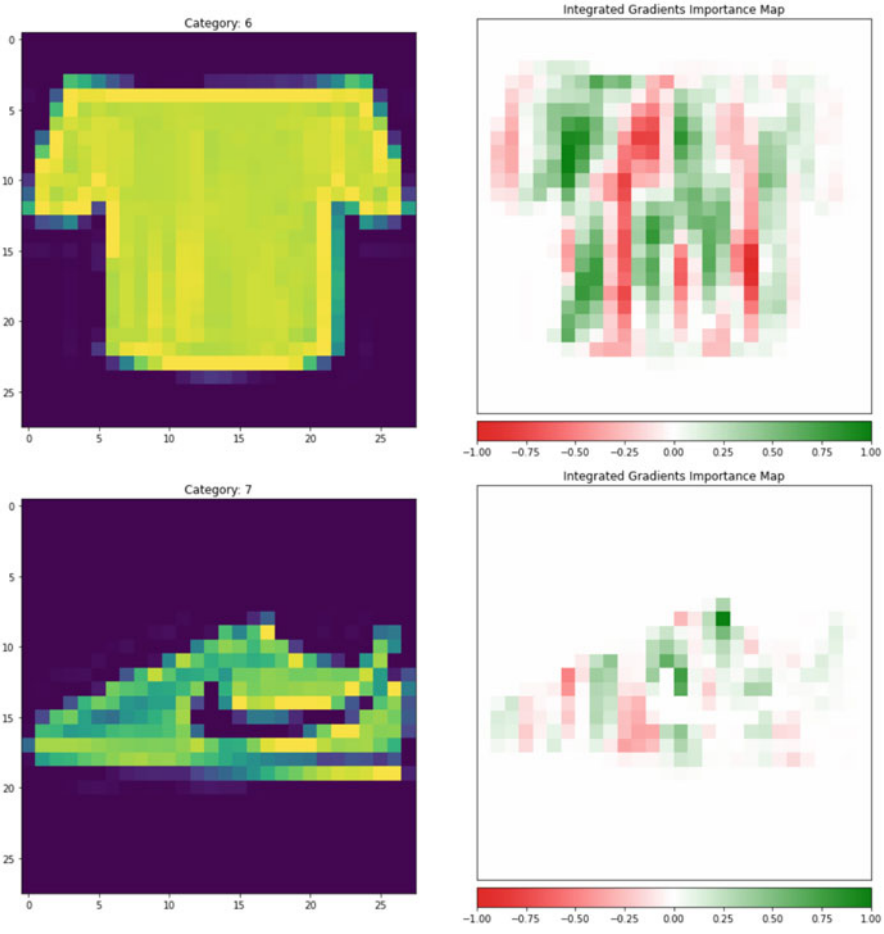

Fig. 6.15 Integrated gradients attributions on Fashion MNIST

Observations:

- The left side of Fig. 6.15 shows the category prediction of a PyTorch 3-layer CNN model trained on Fashion MNIST.
- The right side shows the integrated gradients importance map using a zero baseline and 50-step Riemann approximation.
- Note the quality of the visual explanations in comparison to DeepLIFT.

Table 6.16 Explainable properties of integrated gradients

Properties	Values
Local or global	Local
Linear or non-linear	Non-linear
Monotonic or non-monotonic	Non-monotonic
Feature interactions captured	Yes
Model complexity	Medium

6.5.9 Layer-Wise Relevance Propagation

Layer-wise relevance propagation (LRP) [Bac+15a] is an explanation method that computes feature attributions by backpropagating relevance scores through the network layers from output to input. Relevance scores measure the connection strength between any two neurons.

LRP follows the law of relevance conservation, which states that the relevance of any neuron is equal to the sum of its relevance maps in the previous layer. That is, if the relevance score for the i-th neuron in layer j is given by R_i^j, then conservation states that

$$f(x) = \sum_k R_k^{j+1} = \sum_i R_i^j = \ldots = \sum_m R_m^1 \tag{6.36}$$

where $f(x)$ is the neural network prediction for input x. The total relevance is preserved across layers.

LRP starts with a forward pass on an instance of interest and the class prediction of a single neuron in the top layer with all other neurons at zero value. The relevance is set equal to this class prediction and is backpropagated through the network with the propagation rule:

$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_0^j a_j w_{jk}} R_k \tag{6.37}$$

where neurons j and k are in consecutive layers, a_j is the activation for the neuron in layer j, and w_{jk} is the connection weight between these two neurons. Relevance scores can thus be recursively calculated back to the input and then visualized as a heatmap to explain input feature attribution. Note that other propagation rules can be used for specific applications, and different rules can be used for different layers (Table 6.17) so long as the law of conservation is followed [Mon+19].

Explainable properties of Integrated Gradients are shown in Table 6.18.

Interpretation of Layer-wise Relevance Propagation: LRP provides visual explanations of individual instances by backpropagating relevance scores from the neural network top layer down to the input. By construction, the total relevance is conserved across layers. The choice of propagation rule is a design consideration.

Table 6.17 List of commonly used LRP rules

Rule	Layer
$R_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}} R_k$	Upper
$R_j = \sum_k \frac{a_j w_{jk}}{\epsilon + \sum_{0,j} a_j w_{jk}} R_k$	Middle
$R_j = \sum_k \frac{a_j \left(w_{jk} + \gamma w_{jk}^+\right)}{\sum_{0,j} a_j \left(w_{jk} + \gamma w_{jk}^+\right)} R_k$	Lower
$R_j = \sum_k \left(\alpha \frac{(a_j w_{jk})^+}{\sum_{0,j} (a_j w_{jk})^+} - \beta \frac{(a_j w_{jk})^-}{\sum_{0,j} (a_j w_{jk})^-}\right) R_k$	Lower
$R_j = \sum_k \frac{1}{\sum_j 1} R_k$	Lower
$R_j = \sum_j \frac{w_{ij}^2}{\sum_i w_{ij}^2} R_j$	First
$R_i = \sum_j \frac{x_i w_{ij} - l_i w_{ij}^+ - h_i w_{ij}^-}{\sum_i x_i w_{ij} - l_i w_{ij}^+ - h_i w_{ij}^-} R_j$	First

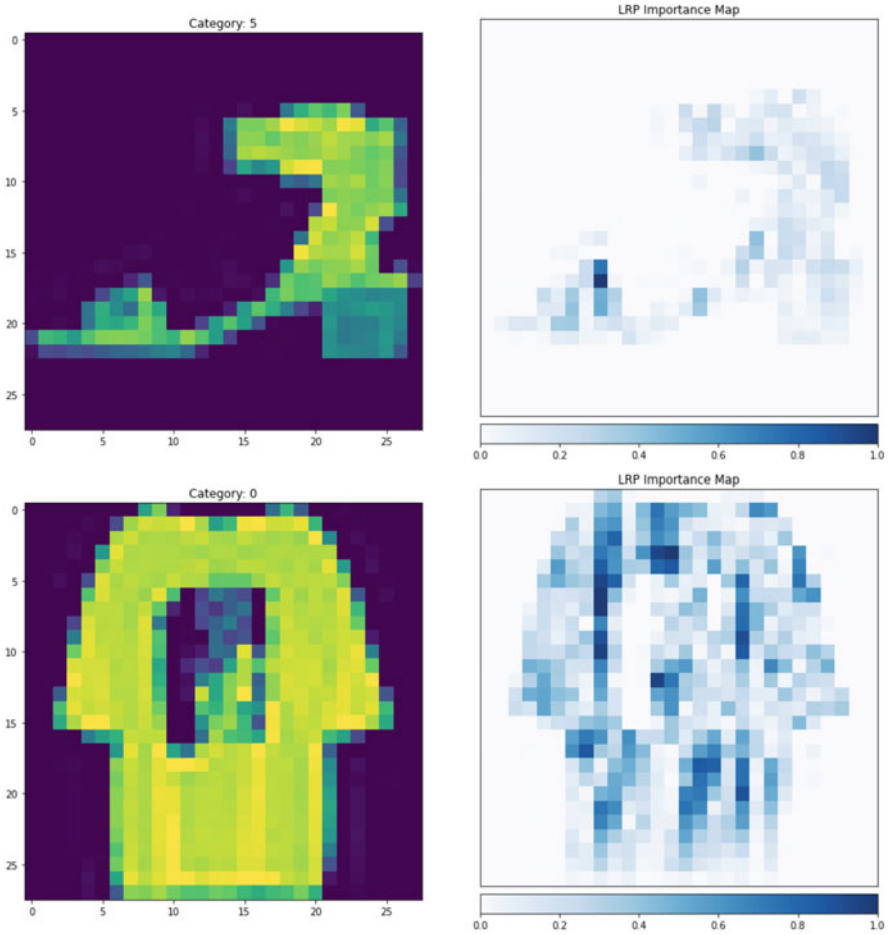

Fig. 6.16 Layer-wise relevance propagation heatmap on Fashion MNIST

Observations:

- Fig. 6.16 shows the category prediction of a PyTorch 3-layer CNN model trained on Fashion MNIST.
- The right side shows the layer-wise relevance propagation heatmap using a zero baseline and 50-step Riemann approximation.
- Note the quality of the visual explanations in comparison to DeepLIFT.

Table 6.18 Explainable properties of layer-wise relevance propagation

Properties	Values
Local or global	Local
Linear or non-linear	Non-linear
Monotonic or non-monotonic	Non-monotonic
Feature interactions captured	Yes
Model complexity	Medium

6.5.10 Excitation Backpropagation

Excitation backpropagation (EBP) [Zha+16] is an explanation method that aims to visualize neuron activations by applying a winner-take-all (WTA) approach to backpropagating through excitatory connections between neurons for classification tasks. It backpropagates only positive weights while keeping gradients normalized.

For a neuron a_i in the i-th layer, the conditional winning probability $P(a_j|a_i)$ of each neuron a_j in the preceding layer connected to it is

$$P(a_j|a_i) = \begin{cases} Z_i \hat{a}_j w_{ji} & w_{ji} \geq 0 \\ 0 & otherwise \end{cases} \tag{6.38}$$

where $\hat{a}_j$ is the input neuron's response and the normalization factor Z_i is given by

$$Z_i = \begin{cases} 0 & \sum_{j:w_{ji}\geq 0} \hat{a}_j w_{ji} = 0 \\ \dfrac{1}{\sum_{j:w_{ji}\geq 0} \hat{a}_j w_{ji}} & otherwise \end{cases} \tag{6.39}$$

In the winner-take-all approach, if a_i is a winning neuron, the next winning neuron will be sampled based on $P(a_j|a_i)$. The weight w_{ji} reflects the top-down feature expectation and $\hat{a}_j$ captures the bottom-up feature strength. Applying this recursively allows EBP to compute marginal winning probability maps which can serve as soft attention maps.

> Interpretation of Excitation Backpropagation: EBP learns soft attention maps by applying a probabilistic winner-take-all process to backpropagate activations top-down through the network. It can also learn contrastive attention maps that improve discriminative ability. EBP is restricted to neural classification tasks.

In practice, EBP is often used to propagate a pair of contrastive top-down signals by backpropagating both a positive and a negative neural activations top-down through the network. As marginal winning probability maps are linear

functions of the top-down signal, the sum of these two activations can be computed simultaneously during a single backward pass. The resulting contrastive marginal winning probability map can amplify discriminative excitations.

Explainable properties of Excitation Backpropagation are shown in Table 6.19.

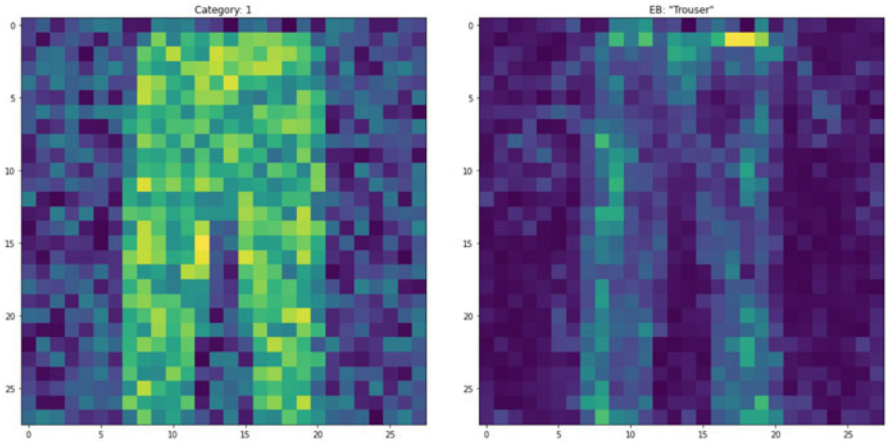

Fig. 6.17 Excitation backpropagation heatmap on Fashion MNIST

Observations:

- Fig. 6.17 shows the input image to a 3-layer MLP model trained on Fashion MNIST.
- The right side shows the excitation backpropagation soft attention map.

Table 6.19 Explainable properties of excitation backpropagation

Properties	Values
Local or global	Local
Linear or non-linear	Non-linear
Monotonic or non-monotonic	Non-monotonic
Feature interactions captured	Yes
Model complexity	Medium

6.5.11 CAM

Class activation maps (CAM) [Zho+15a] is an explanation method applicable to specific CNN architectures. It has been shown that CNNs can be used for object localization if the max pooling operation is replaced with global average pooling [Zho+15b]. By adding a global average pooling operation between the last convolutional layer and the output layer of a CNN, the discriminative image regions associated with a prediction for a particular class can be visualized as a class activation map.

Let $f_k(x, y)$ represent the activation of the k-th neuron in the last convolutional layer of a CNN at spatial location (x, y). Then the output P_c for class c is given by

$$P_c = \frac{\exp(S_c)}{\sum_c \exp(S_c)} \tag{6.40}$$

$$S_c = \sum_k w_k^c F_k \tag{6.41}$$

$$F_k = \sum_{x,y} f_k(x, y) \tag{6.42}$$

where w_k^c is the weight corresponding to class c for the k-th neuron, S_c is the input to the softmax neuron for class c, and F_k is the global average pooled output at the k-th neuron. The weights w_k^c can be interpreted as a measure of the importance of F_k for class c.

The class activation map $M_c(x, y)$ is defined as

$$M_c(x, y) = \sum_k w_k^c f_k(x, y) \tag{6.43}$$

Note that the class c softmax input can be written as

$$S_c = \sum_{x,y} \sum_k w_k^c f_k(x, y) = \sum_{x,y} M_c(x, y) \tag{6.44}$$

and the class activation map can be interpreted as a measure of importance of the activation at spatial location (x, y) for class c prediction. By up-sampling the class activation map to the size of the input image and applying thresholding, a heatmap is generated that identifies the regions of the input image most relevant to class c.

While CAM is computationally efficient as it requires a forward pass and a partial backward pass. Unfortunately, it is restricted to a set of specific CNN architectures that exclude fully connected layers.

Explainable properties of CAM are shown in Table 6.20.

Interpretation of Class Activation Maps: CAMs are useful for a specific set of CNN models by using global max pooling to visualize the regions of an input image most relevant to a class prediction. It exploits the spatial information that is preserved through convolutional layers.

Table 6.20 Explainable properties of CAM

Properties	Values
Local or global	Local
Linear or non-linear	Non-linear
Monotonic or non-monotonic	Non-monotonic
Feature interactions captured	Yes
Model complexity	Medium

6.5.12 Gradient-Weighted CAM

Gradient-weighed CAM (GradCAM) [Sel+19] is a generalization of CAM to allow for more flexible CNN architectures. Instead of relying on a global average pooling after the last convolutional layer, it allows for any architecture as long as layers are differentiable. GradCAM assigns importance values to each neuron by utilizing the gradient information that flows into the last convolutional layer of the CNN.

Let y^c be the score for class c, and let A^k_{xy} be the feature map activations of a convolutional layer neuron at location (x, y). GradCAM calculates the neuron importance weights α^c_k by global average pooling the gradients:

$$\alpha^c_k = \frac{1}{Z} \sum_{x,y} \underbrace{\frac{\partial y^c}{\partial A^k_{xy}}}_{\text{gradient}} \tag{6.45}$$

where Z is a proportionality constant that can be disregarded since it is normalized out during visualization. With these alpha weights, a GradCAM localization heatmap L_c for class c is calculated by

$$L_c = ReLU\left(\sum_k \alpha^c_k A^k\right) \tag{6.46}$$

The ReLU operation is to ensure only positive importance values are emphasized. In effect, GradCAM takes the weighted sum of the feature map activations of the

convolutional layer to generate gradient-weighted class activation maps. These class activation maps are up-sampled to the size of the input image to generate heatmaps of importance values.

> Interpretation of Gradient-Weighted Class Activation Maps: GradCAM is a generalization of CAM to a broader range of CNN architectures. It can efficiently generate localization heatmap explanations for specific instances.

Like CAM, GradCAM is computationally efficient and requires a single forward pass and a partial backward pass. Unlike CAM, it is applicable to a much broader range of CNN-based architectures.

Explainable properties of GradCAM are shown in Table 6.21.

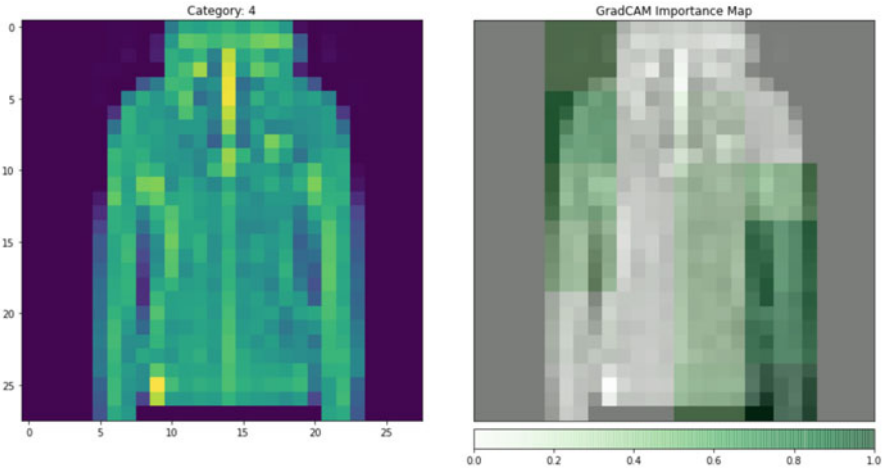

Fig. 6.18 Up-sampled localization heatmap on Fashion MNIST

Observations:

- Fig. 6.18 shows the input image to a PyTorch 3-layer CNN model trained on Fashion MNIST.
- The right side shows the up-sampled GradCAM heatmap for the conv3 layer (the last convolutional layer)
- GradCAM identifies the sleeve areas as important features to the prediction of category 4 (Coat).

Table 6.21 Explainable properties of GradCAM

Properties	Values
Local or global	Local
Linear or non-linear	Non-linear
Monotonic or non-monotonic	Non-monotonic
Feature interactions captured	Yes
Model complexity	Medium

6.5.13 Testing with Concept Activation Vectors

One of the challenges of neural explanation methods is that they may not generate human-interpretable explanations. Feature importance heatmaps may identify regions of the input instance that influence output prediction, but these do not correspond to human-relatable concepts. Furthermore, hidden layer activations are seldom comprehensible. Concept activation vectors (CAVs) [Kim+18] map data and latent representations to human-interpretable concepts.

Let E_m represent the vector space of basis vectors e_m that span the input features and neural activations and E_h represent the vector space of human-interpretable concepts. A concept activation vector is a mapping from E_m to E_h and is learned by training a binary linear classifier f on the layer activations with a set of hand-selected positive examples that contain the concept, as well as a set of random negative instances:

$$f(x) = \begin{cases} 1 & w^T x + b \geq threshold \\ 0 & w^T x + b < threshold \end{cases} \tag{6.47}$$

where w and b are the weights and bias of the binary linear classifier. The CAV v_j^c is the normal vector to the learned hyperplane decision boundary in the direction toward the concept in the j-th layer, $v_j^c = w$.

Testing with Concept Activation Vectors (TCAV) is an explanation method that can quantify the class sensitivity of a trained neural network with respect to a concept in a neural network layer. Let the scoring function $S_{k,j}^c$ be defined as the sensitivity of the neural activation at the j-th layer to class k for a given instance x:

$$S_{k,j}^c = \nabla h_{k,j}(f_j(x)) v_j^c \tag{6.48}$$

where $f_j(x)$ maps the input x to the activation vector of layer j and $h_{k,j}$ maps the activation vector of layer j to the output activation (logit) of class k. The directional derivative is taken toward the concept activation vector v_j^c for layer j. This scalar represents the influence of concept c in influencing the model to predict x as class k. A positive value encourages while a negative value discourages the model toward class k.

TCAV measures the class sensitivity across inputs of an entire class at layer j by computing

$$TCAV_{k,j}^{c} = \frac{\left|x \in X_k : S_{k,j}^{c} > 0\right|}{|X_k|} \tag{6.49}$$

where X_k is the set of input instances labeled as class k. This represents the fraction of instances with activation vectors at layer j positively influenced by concept c. It neither considers the magnitude of the influence nor negative influences.

TCAV has a distinct advantage in that concept activation vectors for user-defined concepts can be learned by providing examples from external datasets. Thus, it is possible to quantify the influence of semantic concepts that are much more human-comprehensible. However, not all concept activation vectors may be meaningful, since even a set of randomly selected instances can still produce a CAV. Furthermore, the CAVs learned for semantically opposing concepts may significantly overlap, resulting in less discriminative ability of the influence of related concepts.

Interpretation of Testing with Concept Activation Vectors: concept activation vectors are effective representations of human-interpretable concepts in the activations of a neural network layer. These vectors can be used to test class sensitivity to particular concepts that generate meaningful and human-understandable explanations.

In practice, learned CAVs should be validated. One method is to retrain the CAV and calculate TCAV on multiple runs using different random negative instances. A meaningful CAV should result in consistent TCAV scores across these iterations, which can be evaluated using a t-test. Another way to validate CAVs is to visualize the patterns that activate each CAV by applying the activation maximization method. A further way to validate CAVs is to visualize the set of instances most and least similar to the CAV in terms of cosine distance.

Explainable properties of TCAV are shown in Table 6.22.

Observations:

- Fig. 6.19 shows the input image (category 9: ankle boot) and individual layer activations of a PyTorch 3-layer CNN model trained on Fashion MNIST.
- TCAV values were calculated using an SGD classifier for a "sneaker" concept trained on 100 positive samples with 100 random negative samples extracted from the test set.
- The TCAV values indicate that class 9 (ankle boot) is fairly sensitive to the concept "sneaker." From a human perspective, they share a semantic relationship as both are types of shoes.

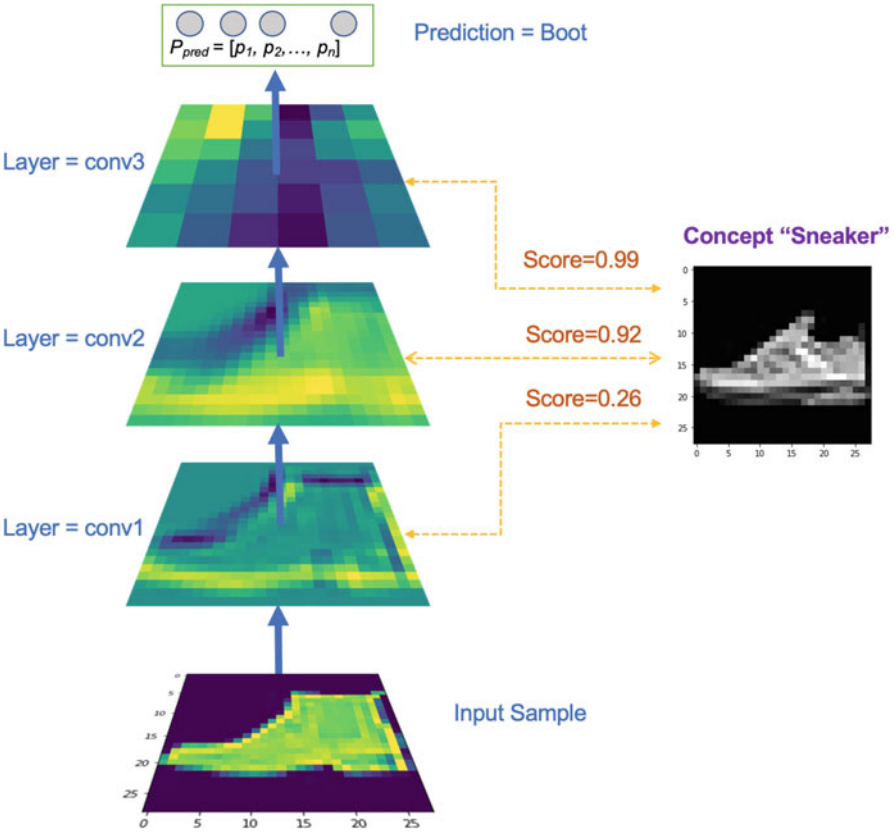

Fig. 6.19 TCAV on Fashion MNIST

Table 6.22 Explainable properties of TCAV

Properties	Values
Local or global	Global
Linear or non-linear	Linear
Monotonic or non-monotonic	Monotonic
Feature interactions captured	Yes
Model complexity	High

References

[AJ18] D. Alvarez-Melis, T.S. Jaakkola, *Towards robust interpretability with self-explaining neural networks* (2018). arXiv:1806.07538 [cs.LG]

[Bac+15a] S. Bach et al., On pixel-wise explanations for non-linear classifier decisions by layerwise relevance propagation. PLOS ONE **10**(7), 1–46 (2015). https://doi.org/10.1371/journal.pone.0130140.9

[Bin+16] A. Binder et al., Layer-wise relevance propagation for deep neural network architectures, in *Information Science and Applications (ICISA) 2016*, ed. by K.J. Kim, N. Joukov, vol. 376. Lecture Notes in Electrical Engineering (Springer Singapore, Singapore, 2016), pp. 913–922. ISBN:978-981-10-0557-2

[Che+19] C. Chen et al., *This looks like that: Deep learning for interpretable image recognition* (2019). arXiv:1806.10574 [cs.LG]

[CLL19] H. Chen, S. Lundberg, S.-I. Lee, *Explaining models by propagating Shapley values of local components* (2019). arXiv:1911.11888 [cs.LG]

[Don+17] Y. Dong et al., *Improving interpretability of deep neural networks with semantic information* (2017). arXiv:1703.04096 [cs.CV]

[ECB10] D. Erhan, A. Courville, Y. Bengio, *Understanding Representations Learned in Deep Architectures*. Tech. rep. 1355. Université de Montréal/DIRO (Oct. 2010)

[FV17] R.C. Fong, A. Vedaldi, Interpretable explanations of black boxes by meaningful perturbation, in *2017 IEEE International Conference on Computer Vision (ICCV)* (Oct. 2017). https://doi.org/10.1109/iccv.2017.371. http://dx.doi.org/10.1109/ICCV.2017.371

[Goy+20] Y. Goyal et al., *Explaining classifiers with causal concept effect (CaCE)* (2020). arXiv:1907.07165 [cs.LG]

[HVD15] G. Hinton, O. Vinyals, J. Dean, *Distilling the knowledge in a neural network* (2015). arXiv:1503.02531 [stat.ML]

[Iye+18] R. Iyer et al., *Transparency and explanation in deep reinforcement learning neural networks* (2018). arXiv:1809.06061 [cs.LG]

[Kim+18] B. Kim et al., Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV), in *ICML*, ed. by J.G. Dy, A. Krause, vol. 80. Proceedings of Machine Learning Research (PMLR, 2018), pp. 2673–2682

[LBJ16] T. Lei, R. Barzilay, T. Jaakkola, Rationalizing neural predictions. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (Association for Computational Linguistics, Austin, Texas, 2016), pp. 107–117. https://doi.org/10.18653/v1/D16-1011. https://www.aclweb.org/anthology/D16-1011

[LMJ17] J. Li, W. Monroe, D. Jurafsky, *Understanding neural networks through representation erasure* (2017). arXiv:1612.08220 [cs.CL]

[Li+17] O. Li et al., *Deep learning for case-based reasoning through prototypes: a neural network that explains its predictions* (2017). arXiv:1710.04806 [cs.AI]

[LYW19] H. Liu, Q. Yin, W.Y. Wang, *Towards explainable NLP: A generative explanation framework for text classification* (2019). arXiv:1811.00196 [cs.CL]

[Mon+19] G. Montavon et al., Layer-wise relevance propagation: An overview. *Explainable AI* (2019)

[PDS18a] V. Petsiuk, A. Das, K. Saenko, *RISE: Randomized input sampling for explanation of black-box models* (2018). arXiv:1806.07421 [cs.CV]

[Sel+19] R.R. Selvaraju et al., Grad-CAM: Visual explanations from deep networks via gradient-based localization. Int. J. Comput. Vis. **128**(2), 336–359 (2019). ISSN: 1573-1405. http://doi.org/10.1007/s11263-019-01228-7

[SS19] S. Serrano, N.A. Smith, *Is attention interpretable?* (2019). arXiv:1906.03731 [cs.CL]

[SGK19] A. Shrikumar, P. Greenside, A. Kundaje, *Learning important features through propagating activation differences* (2019). arXiv:1704.02685 [cs.CV]

[SVZ14a] K. Simonyan, A. Vedaldi, A. Zisserman, *Deep inside convolutional networks: Visualising image classification models and saliency maps* (2014). arXiv:1312.6034 [cs.CV]

[Spr+15b] J.T. Springenberg et al., *Striving for simplicity: The all convolutional net* (2015). arXiv:1412.6806 [cs.LG]

[STY17] M. Sundararajan, A. Taly, Q. Yan, *Axiomatic attribution for deep networks* (2017). arXiv:1703.01365 [cs.LG]

[Tur95] A.M. Turing, *Computers & amp; thought* (MIT Press, 1995), pp. 11–35. Chap. Computing Machinery and Intelligence

[ZTF11] M.D. Zeiler, G.W. Taylor, R. Fergus, Adaptive deconvolutional networks for mid and high level feature learning, in *2011 International Conference on Computer Vision* (2011), pp. 2018–2025. https://doi.org/10.1109/ICCV.2011.6126474

[ZF13] M.D. Zeiler, R. Fergus, *Visualizing and understanding convolutional networks* (2013). arXiv:1311.2901 [cs.CV]

[Zel+19] R. Zellers et al., From recognition to cognition: Visual commonsense reasoning, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019, pp. 6713–6724. https://doi.org/10.1109/CVPR.2019.00688

[Zha+16] J. Zhang et al., *Top-down neural attention by excitation backprop* (2016). arXiv:1608.00507 [cs.CV]

[Zho+15a] B. Zhou et al., *Learning deep features for discriminative localization* (2015). arXiv:1512.04150 [cs.CV]

[Zho+15b] B. Zhou et al., *Object detectors emerge in deep scene CNNs* (2015). arXiv:1412.6856 [cs.CV]

[Zin+17] L.M. Zintgraf et al., *Visualizing deep neural network decisions: Prediction difference analysis* (2017). arXiv:1702.04595 [cs.CV]