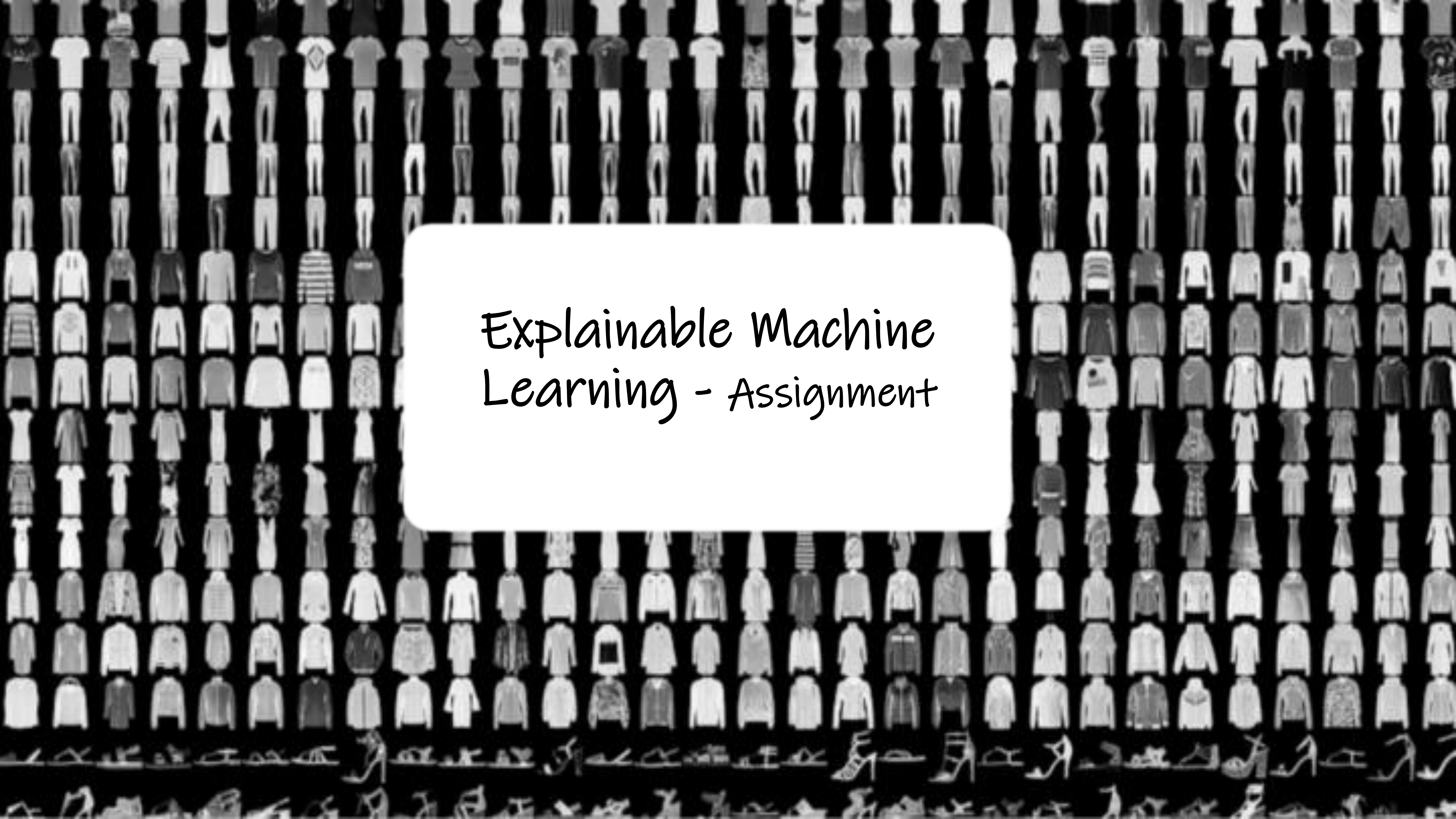


Explainable Machine Learning - Assignment

Summary

1. Intro:
    a. FASHION-MNIST dataset
    b. Network Structure
    c. Network Results
2. Shap
3. Conclusions

Intro – Fashion-MNIST dataset

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. We intend Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

Each training and test example is assigned to one of the labels in table.

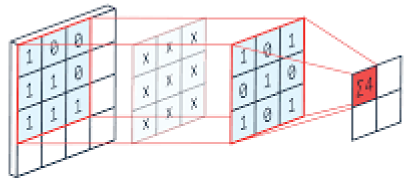
Intro - Network structure 1

```
Model: «sequential_1»
_________________________________________________
Layer (type) Output Shape Param #
=================================================
conv2d (Conv2D) (None, 26, 26, 32) 320

max_pooling2d (MaxPooling2D (None, 13, 13, 32) 0 )

dropout (Dropout) (None, 13, 13, 32) 0

conv2d_1 (Conv2D) (None, 11, 11, 64) 18496

max_pooling2d_1 (MaxPooling (None, 5, 5, 64) 0 2D)

dropout_1 (Dropout) (None, 5, 5, 64) 0

flatten_1 (Flatten) (None, 1600) 0

dense_2 (Dense) (None, 32) 51232

dropout_2 (Dropout) (None, 32) 0

dense_3 (Dense) (None, 10) 330
```

Image Input: An image input layer inputs 2-D images to a network and applies data normalization. Here is specified the size of the input data, as a row vector of integers [h w c], where h, w, and c correspond to the height, width, and number of channels respectively. In our case, is [28, 28, 1].

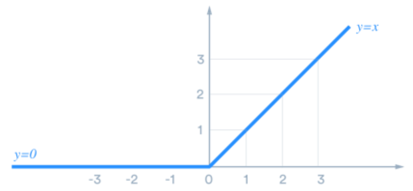
2-D Convolution: this layer applies sliding convolutional filters to 2-D input. The layer convolves the input by moving the filters along the input vertically and horizontally and computing the dot product of the weights and the input, and then adding a bias term.
The parameters used are:

- **filters** - the number of filters (Kernels) used with this layer; here filters = 32;
- **kernel_size** - the dimmension of the Kernel: (3 x 3);
- **activation** - is the activation function used, in this case **relu**;
- **kernel_initializer** - the function used for initializing the kernel;
- **input_shape** - is the shape of the image presented to the CNN: in our case is 28 x 28



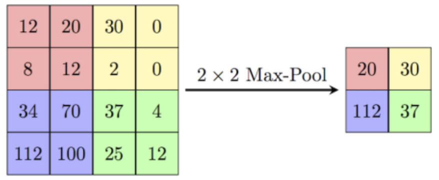
ReLU: this layer performs a threshold operation to each element of the input, where any value less than zero is set to zero.
This operation is equivalent to:

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$



2-D Max Pooling: this layer performs downsampling by dividing the input into rectangular pooling regions, then computing the maximum of each region.
It is usually used after a convolutional layer. Parameters used here are:

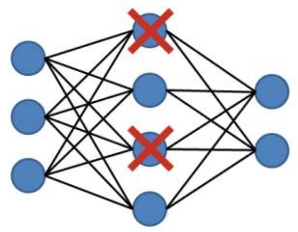
- **pool_size:** in this case (2,2), representing the factors by which to downscale in both directions;



Intro - Network structure 2

```
Model: «sequential_1»
__________________________________________________
Layer (type) Output Shape Param #
==================================================
conv2d (Conv2D) (None, 26, 26, 32) 320

max_pooling2d (MaxPooling2D (None, 13, 13, 32) 0 )

dropout (Dropout) (None, 13, 13, 32) 0

conv2d_1 (Conv2D) (None, 11, 11, 64) 18496

max_pooling2d_1 (MaxPooling (None, 5, 5, 64) 0 2D)

dropout_1 (Dropout) (None, 5, 5, 64) 0

flatten_1 (Flatten) (None, 1600) 0

dense_2 (Dense) (None, 32) 51232

dropout_2 (Dropout) (None, 32) 0

dense_3 (Dense) (None, 10) 330
```

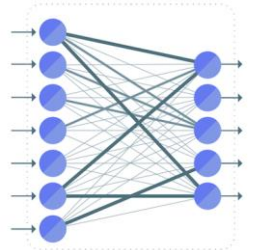
Dropout: is a technique where randomly selected neurons are ignored during training. They are "dropped out" randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass, and any weight updates are not applied to the neuron on the backward pass. The effect is that the network becomes less sensitive to the specific weights of neurons. This results in a network capable of better generalization and less likely to overfit the training data.



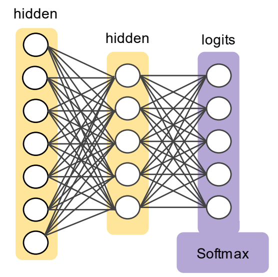
Flatten: it transforms the format of the images from a two-dimensional array (of 28 by 28 pixels) to a one-dimensional array (of 28 * 28 = 784 pixels).

Dense: this layer is a regular fully-connected NN layer. The fully connected layer has neurons connected to all neurons in the preceding layer. This layer multiplies the input by a weight matrix and then adds a bias vector. It combines all the learned features to identify the larger patterns and to classify the images.

In particular, the last Dense layer has:
- **units**: the number of classes (10)
- **activation** : softmax

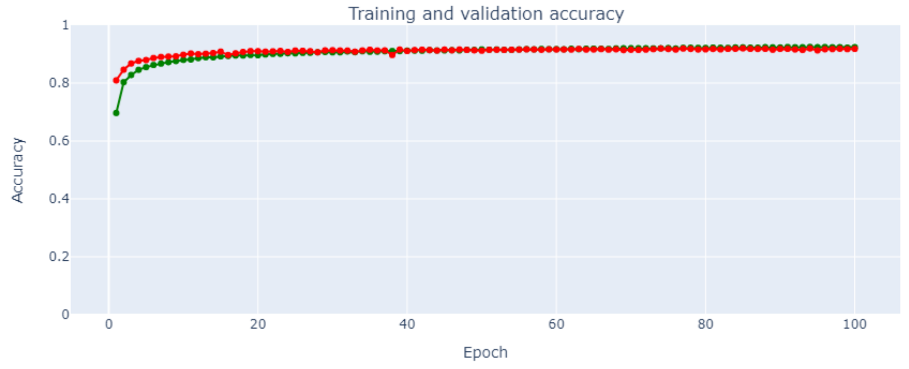


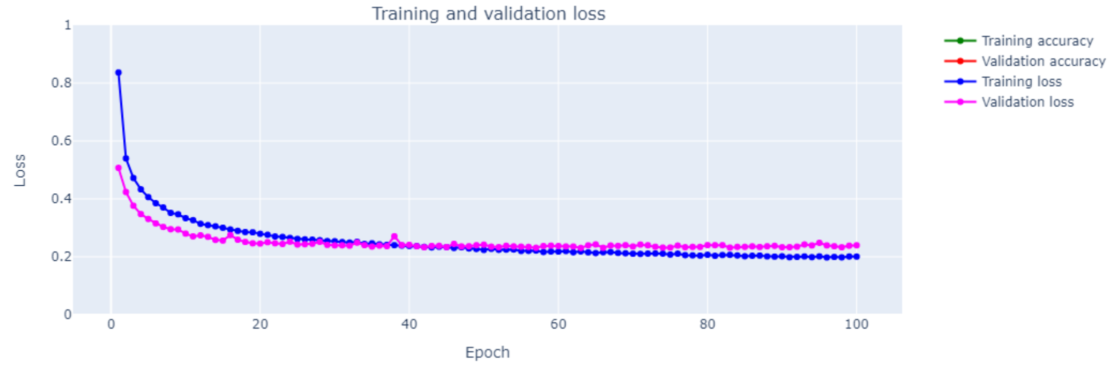
Softmax: softmax activation function normalizes the output of the fully connected layer. The output of the softmax layer consists of positive numbers that sum to one, which can then be used as classification probabilities by the classification layer.



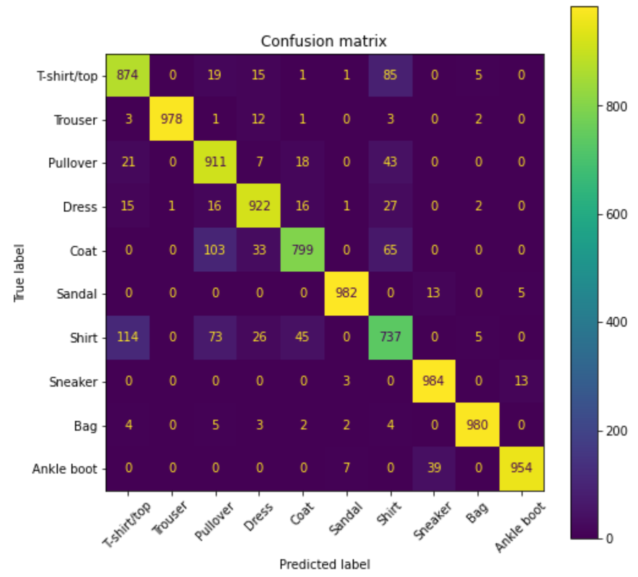
Intro – Network training and evaluation

Model Training:

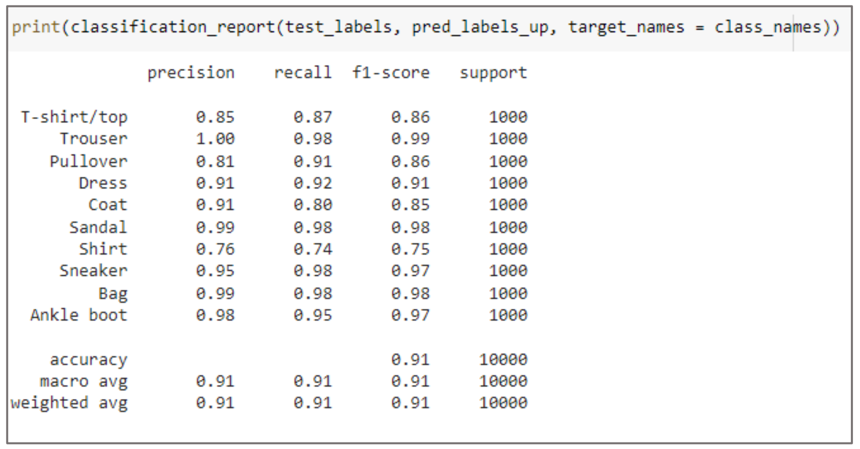




Model Evaluation:



	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	874	0	19	15	1	1	85	0	5	0
Trouser	3	978	1	12	1	0	3	0	2	0
Pullover	21	0	911	7	18	0	43	0	0	0
Dress	15	1	16	922	16	1	27	0	2	0
Coat	0	0	103	33	799	0	65	0	0	0
Sandal	0	0	0	0	0	982	0	13	0	5
Shirt	114	0	73	26	45	0	737	0	5	0
Sneaker	0	0	0	0	0	3	0	984	0	13
Bag	4	0	5	3	2	2	4	0	980	0
Ankle boot	0	0	0	0	0	7	0	39	0	954



```
print(classification_report(test_labels, pred_labels_up, target_names = class_names))

              precision    recall  f1-score   support

 T-shirt/top       0.85      0.87      0.86      1000
     Trouser       1.00      0.98      0.99      1000
    Pullover       0.81      0.91      0.86      1000
       Dress       0.91      0.92      0.91      1000
        Coat       0.91      0.80      0.85      1000
      Sandal       0.99      0.98      0.98      1000
       Shirt       0.76      0.74      0.75      1000
     Sneaker       0.95      0.98      0.97      1000
         Bag       0.99      0.98      0.98      1000
  Ankle boot       0.98      0.95      0.97      1000

    accuracy                           0.91     10000
   macro avg       0.91      0.91      0.91     10000
weighted avg       0.91      0.91      0.91     10000
```

This slide shows the model's training and results

- Above, we show the trend of accuracy and loss on training and validation data throughout the learning phase.
- Below, we show the confusion matrix and the classification report: in both views it can be seen that the model is working reasonably well, managing to correctly classify most of the images. However, it is observed that for some garments, such as coats, shirts and pullovers (those most similar to each other), more prediction errors are made.

Shap

SHAP (SHapley Additive exPlanations) is a game theoretic approach to explain the output of any machine learning model by calculating the contribution of each feature to the prediction.

```
# select a set of background examples to take an expectation over
background = x_train[np.random.choice(x_train.shape[0], 100, replace=False)]
```

The SHAP library has a **deep explainer** module which contains a representation to showcase the positive and negative attributes or contributions towards a classification task in general and specific to each class in particular.

```
# explain predictions of the model on images
new = shap.DeepExplainer(model_upgrade, background)

shap_values_new = new.shap_values(x_test[[0,1,2,4,6,8,9,13,18,19]])

# plot the feature attributions
shap.image_plot(shap_values_new, -x_test[[0,1,2,4,6,8,9,13,18,19]])
```

In this example we apply the Shap method to figure out which ones pixels of the various images are contributing positively or negatively to the classification of the garment.

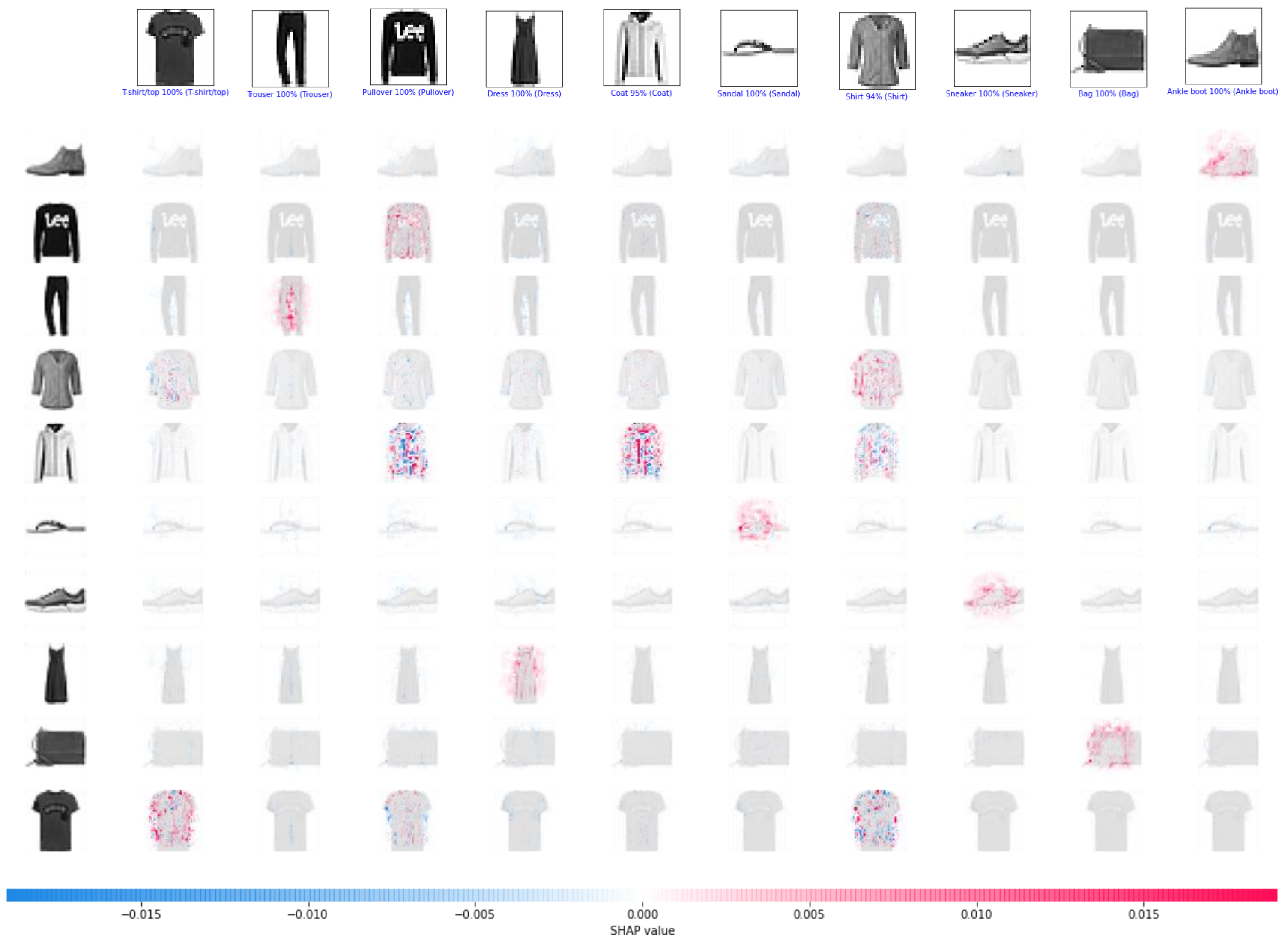
This allows us to explain the numerical results shown in the previous slide.

Conclusions

general results

Predicted class

Shap results



The images on the left shows an example prediction for each of the 10 classes.

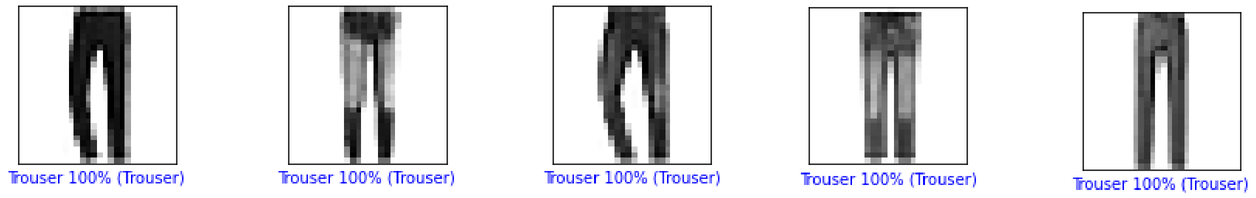
The plot shows the explanations for each class on 10 predictions. Note that the explanations are ordered for the classes 0-9 going left to right along the rows.

In this visualization:

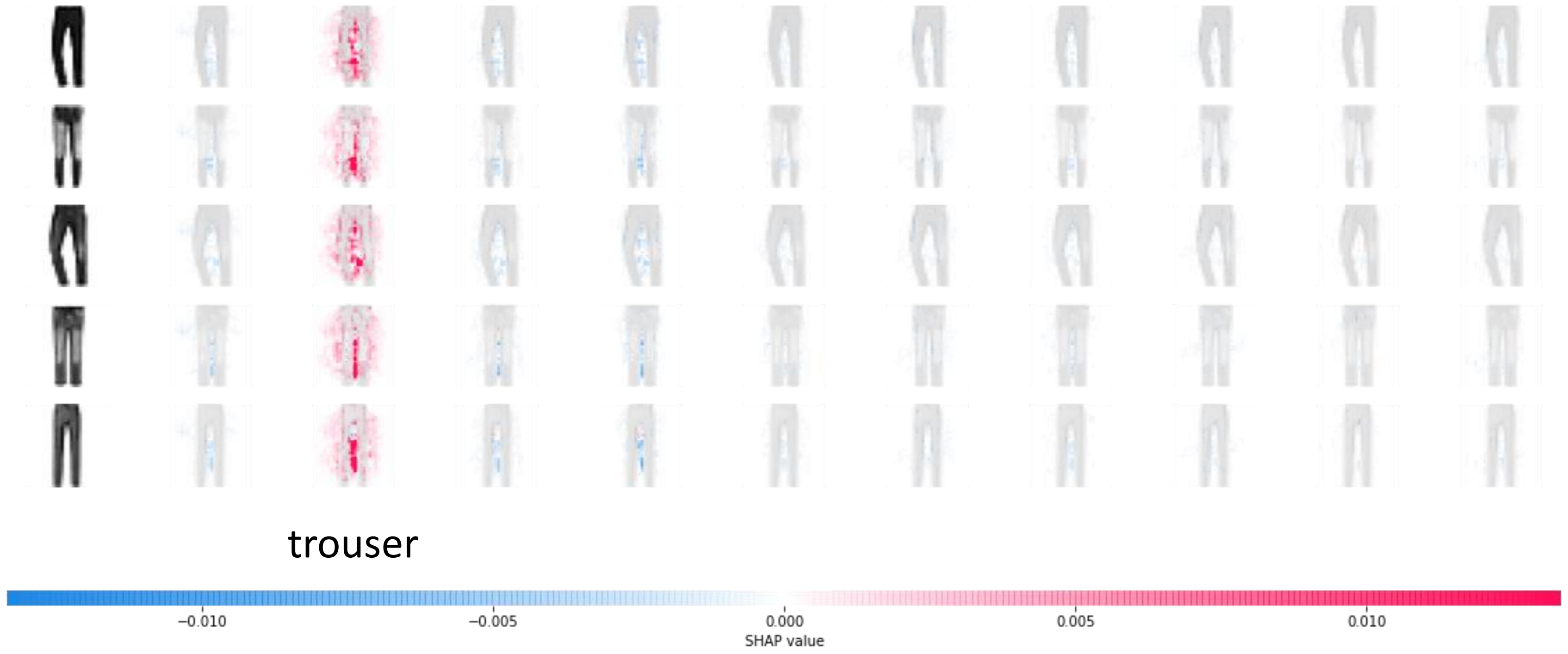
- **Red** pixels represent positive SHAP values that contributed to classifying that image as that particular class.
- **Blue** pixels represent negative SHAP values that contributed to not classifying that image as that particular class.

Conclusions

good results



Shap results



The plot shows the explanations for 5 examples of the Trouser class. All the trousers' examples here considered have been correctly predicted.

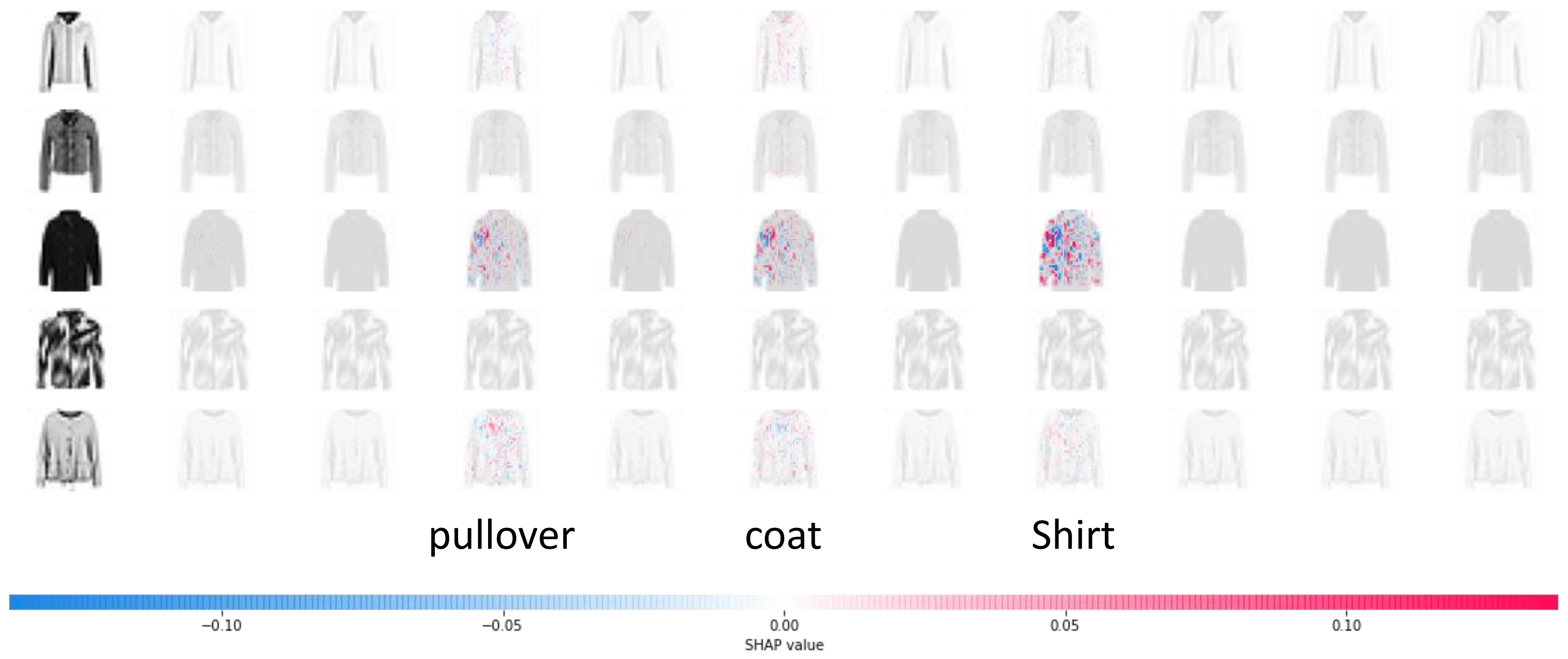
Observing the results of the Shap algorithm it is evident how the pixels contributed to the prediction:

- In the pants class, many **red** pixels are observed in the area between the legs; these pixels have positively contributed to the classification of the image as trousers.
- In the other classes, however, the presence of **blue** pixels is observed in the same area between the legs: this highlights how those points have contributed negatively to the classification.

Conclusions

bad results

Shap results



The plot shows the explanations for 5 examples of the **coat** class. This is one of the more difficult classes to predict.

Observing the results of the Shap algorithm it is not evident, as it was in the trousers' case, how the pixels contributed to the prediction:

- The presence of **red** pixels in different classes shows how the algorithm fails to identify a distinctive feature in some images of the coats.
- It is also observed that the greatest inaccuracy occurs in correspondence with garments that are similar to each other (pullover, coat and shirt)
- We note that in the correct prediction of the first example shown here, with a percentage of 95%, the pixels corresponding to the closure of the jacket gave a positive contribution.