How to implement a medical image classification pipeline with just a few lines of code

A tutorial how to implement a medical image pipeline to classify skin lesions with the AUCMEDI framework

Skin cancer is one of the most common cancer types worldwide but with an early detection the majority of the patients can be cured. This makes it such an promising field to apply deep learning models that can help physicians classify skin lesions.

Implementing deep learning pipelines to classify images can be really complicated and time consuming that is why this blog posts presents the AUCMEDI framework. The AUCMEDI framework (https://github.com/frankkramer-lab/aucmedi) allows to the user to set up medical image classification pipelines with state-of-the-art methods via an intuitive, high-level Python API or via an AutoML deployment through Docker/CLI with just a few lines of code.

The dataset is the ISIC2019 dataset ([1], [2], [3]) that is provided on Kaggle (https://www.kaggle.com/datasets/andrewmvd/isic-2019) and it contains 25,331 images from nine classes: Melanoma (MEL), Melanocytic nevus (NV), Basal cell carcinoma (BCC), Actinic keratosis (AK), Benign keratosis (BKL), Dermatofibroma (DF), Vascular lesion (VASC), Squamous cell carcinoma (SCC), Unknown (UNK). Malignant classes are: Melanoma, Basal cell carcinoma and Squamous cell carcinoma.

The following section presents how to implement an AUCMEDI pipeline to classify skin lesions using the ISIC2019 dataset. The programming language used is python 3.8.

Install AUCMEDI and make necessary imports

First of all AUCMEDI has to be installed.

```
#install aucmedi
!pip install aucmedi
```

```
from aucmedi import *
```

Change csv-file

The ground truth data is stored is a csv-file. The data is one-hot encoded that means that each class has a column and for each image it is encoded in 0 / 1 if the image belongs to this class or not. Unfortunately the Unknown class does not contain any sample which would lead to problems later. Therefore this column is removed from the csv-file.

```
import pandas as pd


#variable for the csv file
csv_file = "data/ISIC/ISIC_2019_Training_GroundTruth.csv"


#Load csv file into a dataframe
data = pd.read_csv(csv_file)


#Drop the Unknown column
data.drop('UNK', axis=1, inplace=True)


#Write the modified dataframe to the csv file
data.to_csv(csv_file, index=False)
```

Obtaining general dataset information

AUCMEDI is based on three pillars:
* Pillar 1: `input_interface()` for obtaining general dataset information
* Pillar 2: `NeuralNetwork()` for the deep learning model
* Pillar 3: `DataGenerator()` for a powerful interface to load any images/volumes into your model

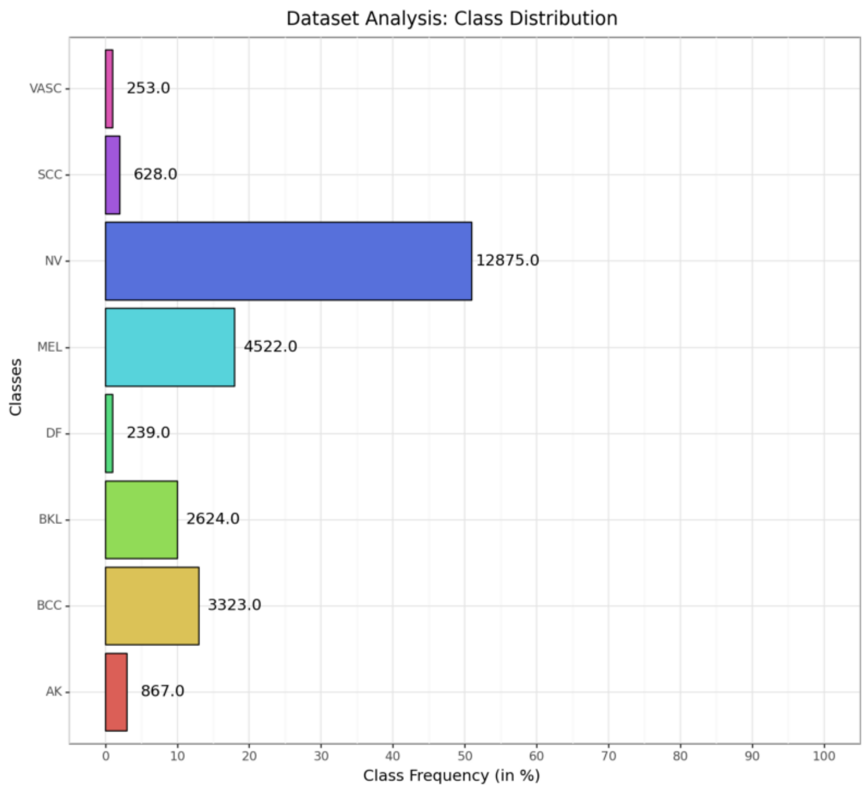
We start with the first pillar which is the `input_interface()` .

```
# Peak data information via the first pillar of AUCMEDI
ds = input_interface(interface="csv", # Interface type
                     path_imagedir=image_directory,
                     path_data=csv_file,
                     ohe=True, #one-hot encoding
                     col_sample="image", #column with
samples
```

```
                    col_class=['MEL', 'NV', 'BCC', 'AK',
'BKL',    'DF', 'VASC', 'SCC'])

(samples, class_ohe, nclasses, class_names, image_format) =
ds
```

One feature of AUCMEDI is that a function to analyse the dataset is provided. Here a barplot for the class distribution is created and saved in the directoy of `out_path` . If wished, a heatmap plot can be generated, too, but in this case the heatmap does not contain additional value.

```
from aucmedi.evaluation.dataset import evaluate_dataset


# Pass information to the evaluation function
evaluate_dataset(samples, class_ohe, out_path="./data
/evaluation", class_names=class_names, show=True,
plot_barplot=True)
```



Class distribution for the ISIC2019 dataset

In general the distribution of the classes is very unbalanced: the NV class contains about the half of the samples whereas the DF and VASC class contain only 1 % of the samples. This is important later for the model.

The model

The next section describes the second pillar of AUCMEDI: the `NeuralNetwork()`. We have seen above that the classes are unbalanced but we can solve this problem by using class weights. The class weights can be computed with the `compute_class_weights` function from AUCMEDI. The return value `cw_loss` is a list of the class weight which can be feeded to a loss function and `cw_fit` is a dictionary with the class weight which can be feeded to `train()` or `keras.model.fit()`.

```
from aucmedi.utils.class_weights import
compute_class_weights


# Compute class weights
cw_loss, cw_fit = compute_class_weights(class_ohe)
```

Now a model is created, in this case the chosen architecture is VGG16 (A list of all available 2D architectures can be found at https://frankkramer-lab.github.io/aucmedi/reference/neural_network/architectures/image/#aucmedi.neural_network.architectures.image). Here the class weights are used for the loss function, in this case the `categorial_focal_loss` function.

```
from aucmedi.neural_network.loss_functions import
categorical_focal_loss
from aucmedi.neural_network.model import NeuralNetwork


# Pillar #2: Initialize a model with ImageNet weights
model = NeuralNetwork(n_labels=nclasses, channels=3,

loss=categorical_focal_loss(cw_loss),
                        architecture="2D.VGG16",
                        pretrained_weights=True)
```

Splitting the dataset

For training and evaluating the model we need three sets: the `train_set` to train the model, the `validation_set` to tune the hyperparameters of the model and the `test_set` to evaluate the model. Here the `train_set` contains 75 % of the samples, the `validation_set` contains 20 % of the samples and 5 % are the `test_set` .

```
from aucmedi.sampling.split import sampling_split


#determine the split ratios
split_ratio = [0.75, 0.20, 0.05]


ds = sampling_split(samples, labels=class_ohe,
sampling=split_ratio)


#Get the sets
train_set = ds[0]
validation_set = ds[1]
test_set = ds[2]
```

Callback functions

In the next section we define callback functions that will help us to optimize our training. `ReduceLROnPlateau` and `EarlyStopping` both monitor the `val_loss` metric and if `val_loss` has not decreased after five epochs (the `patience`) `ReduceLROnPlateau` reduces the learning rate and `EarlyStopping` stops the training.

```
from tensorflow.keras.callbacks import ReduceLROnPlateau,
EarlyStopping

# Define callback reduce learning rate on plateau
cb_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
patience=5, verbose=1,
                          mode='min', min_delta=0.0001,
cooldown=1,
                          min_lr=0.00001)
# Define callback early stopping
cb_es = EarlyStopping(monitor='val_loss', min_delta=0,
patience=5,   verbose=1, mode='min')
```

The DataGenerators

Two `DataGenerators` are created (one for training and one for validation) that are used to train the model later.

```
# Pillar #3: Initialize training Data Generator


train_gen = DataGenerator(samples=train_set[0],
                          path_imagedir=image_directory,
                          labels=train_set[1],
                          image_format=image_format,
                          resize=model.meta_input,

standardize_mode=model.meta_standardize)
```

```
validation_gen = DataGenerator(samples=validation_set[0],
                               path_imagedir=image_directory,
                               labels=validation_set[1],
                               image_format=image_format,
                               resize=model.meta_input,

standardize_mode=model.meta_standardize)
```

Train the model

Finally we can train our model. The number of `epochs` is set to 1000 but likely we will need less `epochs` because of the `callback` -functions.

```
# Run model training with Transfer Learning


history = model.train(train_gen, validation_gen,
epochs=1000, callbacks=[cb_lr, cb_es],
transfer_learning=True)
```

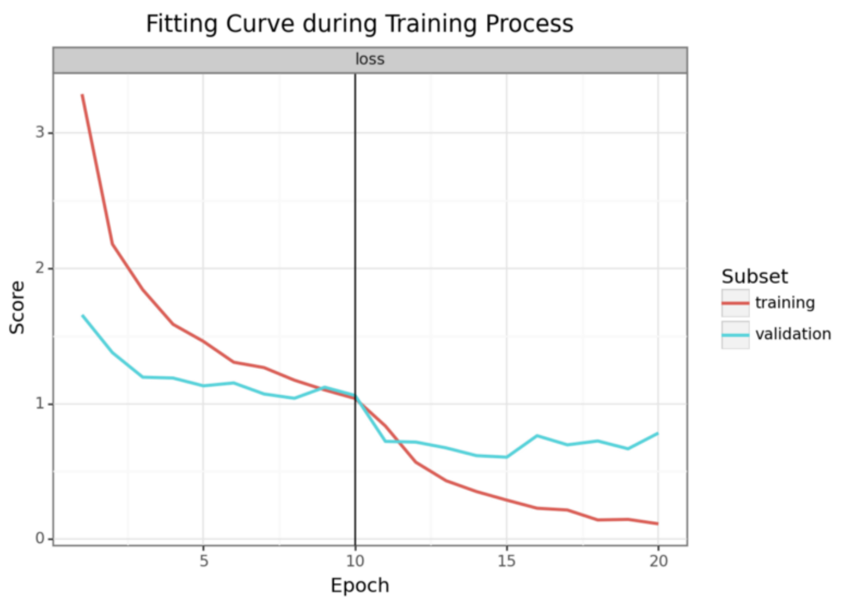
The trained model can now be stored and loaded later to reuse it.

```
# Save the model
model.dump('data/model_VGG16/model')


# Load the model
model.load('data/model_VGG16/model')
```

The training can be evaluated and the plot is stored in the directoy of `out_path` .

```
from aucmedi.evaluation.fitting import evaluate_fitting

# Pass history dict to evaluation function
evaluate_fitting(history, out_path="./data/model_VGG16/")
```

The stored plot has two decreasing graphs for the `train_set` and the `validation_set.`



Evaluation

Now we want to see how the model performs on the `test_set` .

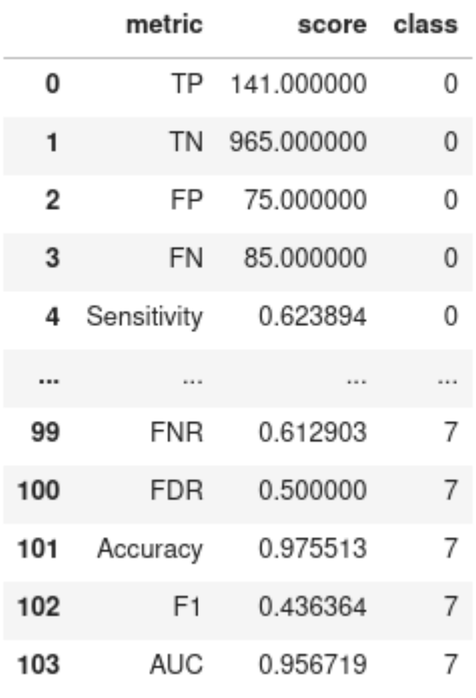
```
# Pillar #3: Initialize testing Data Generator

test_gen = DataGenerator(samples=test_set[0],
                         path_imagedir=image_directory,
                         labels=test_set[1],
                         image_format=image_format,
                         resize=model.meta_input,

standardize_mode=model.meta_standardize)
```

Next we make a prediction on all samples from the `test_gen` set.

```
# Run model inference for unknown samples

preds = model.predict(test_gen)
```

The performance of the model on the `test_set` can be evaluated with functions provided by AUCMEDI. We start with a general overview of the metrics.

```
from aucmedi.evaluation.metrics import compute_metrics

compute_metrics(preds, labels=test_set[1],
n_labels=nclasses, threshold=None
```



	metric	score	class
0	TP	141.000000	0
1	TN	965.000000	0
2	FP	75.000000	0
3	FN	85.000000	0
4	Sensitivity	0.623894	0
...	...	...	...
99	FNR	0.612903	7
100	FDR	0.500000	7
101	Accuracy	0.975513	7
102	F1	0.436364	7
103	AUC	0.956719	7

Next we compute the confusion matrix.

```
from aucmedi.evaluation.metrics import
compute_confusion_matrix

compute_confusion_matrix(preds, labels=test_set[1],
n_labels=nclasses)
```

The output is:

```
array([[141.,  57.,  11.,   4.,  12.,   0.,   1.,   0.],
       [ 55., 542.,  21.,   3.,  23.,   0.,   0.,   0.],
       [  5.,   3., 148.,   4.,   2.,   0.,   1.,   3.],
       [  0.,   1.,  13.,  24.,   4.,   0.,   0.,   1.],
       [ 11.,  21.,  13.,   4.,  75.,   0.,   0.,   7.],
       [  3.,   1.,   0.,   1.,   1.,   5.,   0.,   1.],
       [  0.,   0.,   1.,   0.,   0.,   0.,  12.,   0.],
       [  1.,   1.,  12.,   0.,   4.,   1.,   0.,  12.]])
```

The next possibility to evaluate the performance is the roc curve.

```
from aucmedi.evaluation.metrics import compute_roc


compute_roc(preds, labels=test_set[1], n_labels=nclasses)
```

The output is really long but the beginning looks like this:

```
([array([0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
9.61538462e-04,
         9.61538462e-04, 1.92307692e-03, 1.92307692e-03,
2.88461538e-03,
         2.88461538e-03, 3.84615385e-03, 3.84615385e-03,
4.80769231e-03,
         4.80769231e-03, 5.76923077e-03, 5.76923077e-03,
7.69230769e-03,
         7.69230769e-03, 8.65384615e-03, 8.65384615e-03,
1.05769231e-02,
         1.05769231e-02, 1.15384615e-02, 1.15384615e-02,
1.25000000e-02,
...
```

Finally we use the `evaluate_performance` and the generated plots are stored in `out_path` .

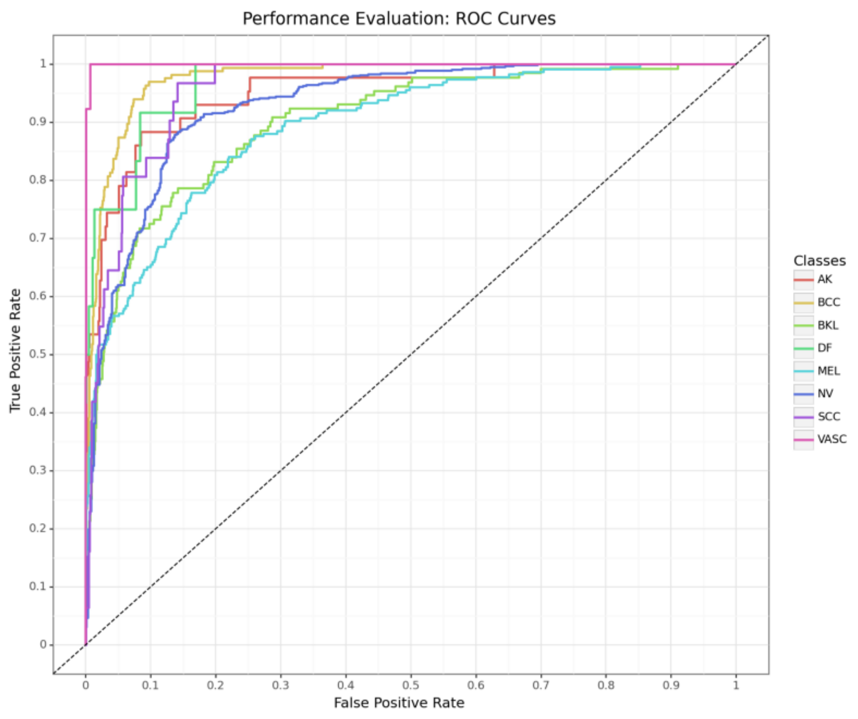
```
from aucmedi.evaluation.performance import
evaluate_performance
```

```
# Pass predictions to evaluation function
evaluate_performance(preds, labels=test_set[1],
out_path="./data/model_VGG16/evaluation",
class_names=class_names)
```

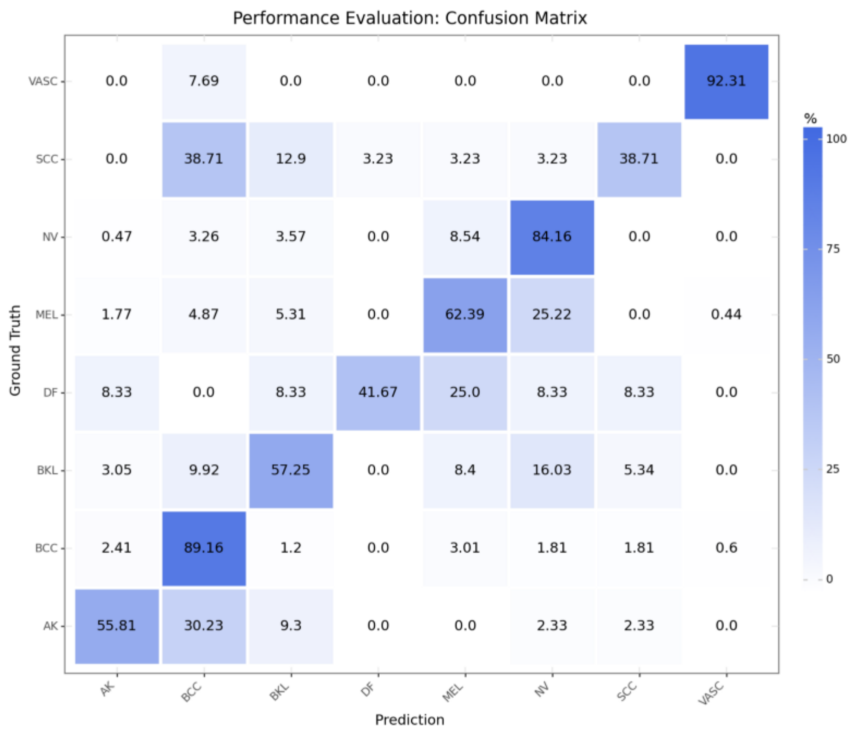
The generated output is:

	metric	score	class
0	TP	141.000000	MEL
1	TN	965.000000	MEL
2	FP	75.000000	MEL
3	FN	85.000000	MEL
4	Sensitivity	0.623894	MEL
...	...	...	...
99	FNR	0.612903	SCC
100	FDR	0.500000	SCC
101	Accuracy	0.975513	SCC
102	F1	0.436364	SCC
103	AUC	0.956719	SCC

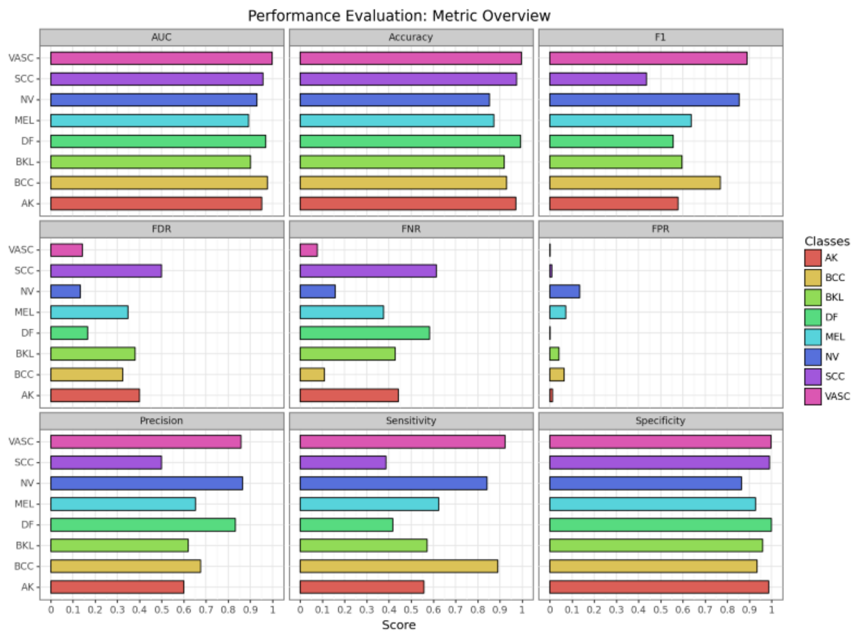
The plot for the roc curve looks like this:



A confusionMatrix is generated, too:



The barplots give an overview over the different metrics:



XAI

For humans it is not possible to understand how deep learning models classify images. Here xai (short for explainiable artificial intelligence) can help to see in which regions the model is interested in. Visit https://frankkramer-lab.github.io/aucmedi/reference/xai/methods/ to get an overview over all implemented xai methods.

```
from aucmedi.xai import xai_decoder

# Compute XAI heatmaps via Grad-CAM (resulting heatmaps are
stored in out_path)
xai_decoder(test_gen, model, preds, method="gradcam",
out_path="./data/model_VGG16/xai")
```

Now we use a sample to check how the prediction works. A sample from the `test_set` is used in this case it is image ISIC_0000002.jpg.

First we check the ground truth of the image:

```
data = pd.read_csv(csv_file)
row = data.loc[data['image'] == 'ISIC_0000002']
print(row)
```

The output is:

```
image  MEL   NV  BCC   AK  BKL   DF  VASC  SCC
2  ISIC_0000002  1.0  0.0  0.0  0.0  0.0  0.0   0.0  0.0
```

The image is from the NV class. Now we check the prediction.

```
prediction_image = model.predict(DataGenerator(samples=
['ISIC_0000002'],

path_imagedir=image_directory,
                                     labels=None,

image_format=image_format,

resize=model.meta_input,

standardize_mode=model.meta_standardize))
prediction_image = prediction_image[0]

prediction_image = prediction_image * 100
prediction_image = np.around(prediction_image)
for i in range(8):
    print(class_names[i] + ": Percentage value " +
str(prediction_image[i]) + "%")
```

The output is:

```
MEL: Percentage value 77.0%
NV: Percentage value 20.0%
BCC: Percentage value 0.0%
AK: Percentage value 0.0%
BKL: Percentage value 3.0%
DF: Percentage value 0.0%
VASC: Percentage value 0.0%
SCC: Percentage value 0.0%
```

We see that the class with the highest percentage value of 77% is the MEL class which is correct.

The image can be shown with:

```
im = Image.open(image_directory+'/ISIC_0000002.jpg')
im.show()
```

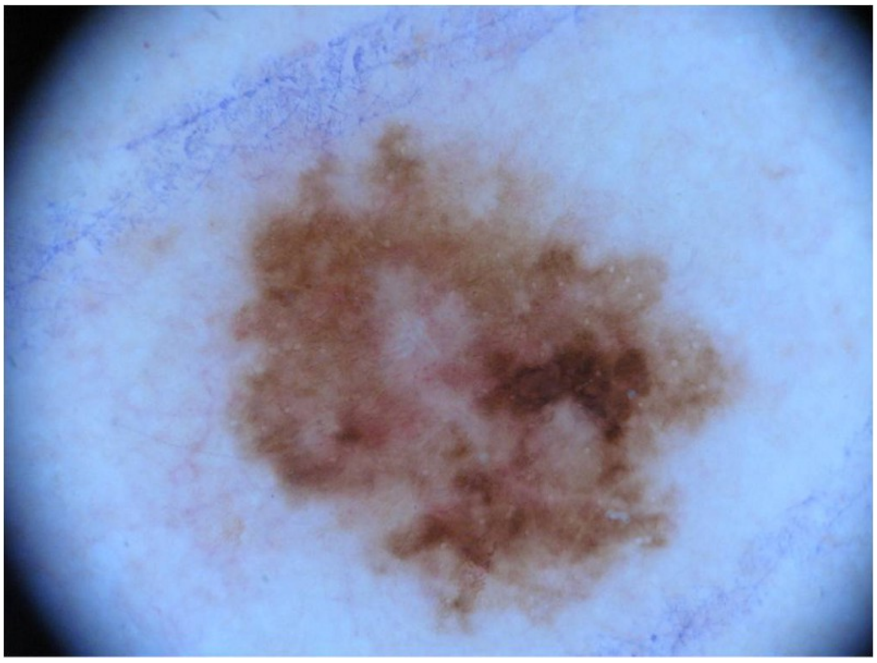
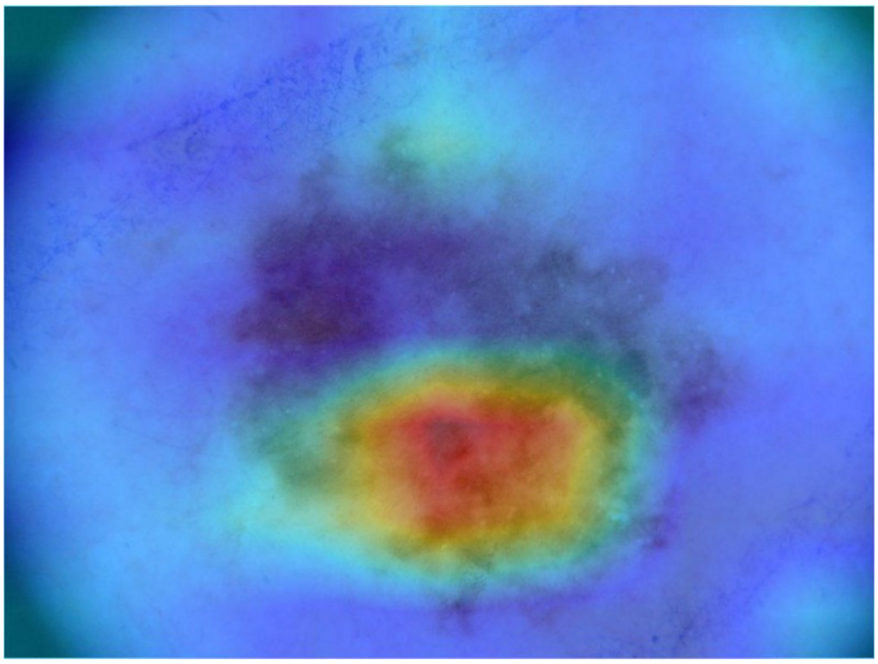


Image ISIC0000002.jpg

Now we show the xai image for this picture:

```
im = Image.open("./data/model_VGG16/xai/ISIC_0000002.jpg")
im.show()
```



XAI image 0000002.jpg

We can see that the majority of the skin lesion is marked red and the prediction seems to work well.

I hope this tutorial gives you a good overview how you can implement a pipeline to classify skin lesions with AUCMEDI

Nadja Bramkamp
Research assistant, Chair of IT Infrastructures for Translational Medical Research
University Augsburg

[1] Tschandl P., Rosendahl C. & Kittler H. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Sci. Data 5, 180161 doi.10.1038/sdata.2018.161 (2018)

[2] Noel C. F. Codella, David Gutman, M. Emre Celebi, Brian Helba, Michael A. Marchetti, Stephen W. Dusza, Aadi Kalloo, Konstantinos Liopyris, Nabin Mishra, Harald Kittler, Allan Halpern: "Skin Lesion Analysis Toward Melanoma Detection: A Challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), Hosted by the International Skin Imaging

Collaboration (ISIC)”, 2017; arXiv:1710.05006.

[3] Marc Combalia, Noel C. F. Codella, Veronica Rotemberg, Brian Helba, Veronica Vilaplana, Ofer Reiter, Allan C. Halpern, Susana Puig, Josep Malvehy: “BCN20000: Dermoscopic Lesions in the Wild”, 2019; arXiv:1908.02288.