General Remarks:

- Feel free to use the *Matplotlib, Tensorflow, Pandas, Numpy & Scikit-learn.*
- Make sure to discuss your results and the selected model configurations.

On *TUWEL*, you can find the publicly available *dogs_vs_cats* dataset. For the project, you will implement a model that correctly classifies unseen images into *dogs* and *cats*. Overall, the project consists of the **classification** and a **transfer learning** part followed by the application of **model explainability & adversarial learning** methods.

First, prepare the data by rescaling it to a shape of (256, 256, 3) and split it into a training and test dataset. For that, use the *tensorflow* function *image_dataset_from_directory* with *crop_to_aspect_ratio=True*. If you are using the provided *JupyterLab*, the dataset is already available under */shared*.

1. Binary Classification

Your first task is to implement a machine learning model classifying images of *cats* and *dogs*.

1.1 Train a Ridge Classifier, a Logistic Regressor, and a SVC from *scikit-learn* on the dataset and evaluate the performance. Briefly analyze the influence of the available regularization hyperparameters and the effects of different kernels. Also, consider using a preprocessing scheme such as standardization or scaling. *Hint: It is sufficient to only select a random subset from the dataset.*

1.2 Use *tensorflow* to implement a neural network that achieves an accuracy above 80% on the test dataset. Plot the learning curves and discuss the model configuration. You can use the *resnet18* block from `https://github.com/jimmyyhwu/resnet18-tf2` as a part of your custom model. *Hint: Also consider using the available data augmentation techniques from tensorflow.*

2. Transfer Learning

In transfer learning, we utilize a pre-trained model from a similar domain to enhance the performance of a given regression or classification scheme.

2.1 Set up a *MobileNetV2*[1] model in *tf.keras* using the pre-trained weights from the *imagenet* dataset. Subsequently, evaluate the network's predictions on the *dogs_vs_cats* dataset and analyze the distribution of the assigned labels for each class.
Hint: Use the provided mobilenet.preprocess_input function.

2.2 Drop the final output layers of the pre-trained *MobileNetV2* and process the *dogs_vs_cats* data through the remaining layers. Retrain the *scikit-learn* models from 1.1 on the processed inputs and compare the performance to the previous results.

2.3 Replace the top layers of the pre-trained *MobileNetV2* with a set of custom *Dense*

[1] See tf.keras.applications

layers suited for the binary *dogs_ vs_ cats* classification task. Train the newly added layers of this composite model to achieve a test accuracy above 95%.
Hint: You can keep the weights of the MobileNetV2 fixed.

3. Adversarial Machine Learning

Most machine learning methods are sensitive to input perturbations, fooling the model into predicting a wrong label. In the following, we will study a straightforward approach for generating such inputs termed the *Fast Gradient Sign Method (FGSM).*

3.1 For *FGSM*, see `https://arxiv.org/abs/1412.6572`, the inputs are perturbated with the signed gradient of the trained model resulting in the adversarial examples:

$$\mathbf{x}_{\text{adv}} = \mathbf{x} + \boldsymbol{\eta} \qquad \boldsymbol{\eta} = \epsilon \cdot \operatorname{sign} \nabla_x J(\boldsymbol{\theta}, \mathbf{x}, y). \tag{1}$$

Implement the code to retrieve the gradients with respect to the inputs and evaluate how sensitive the models from 1.2 and 2.3 are to the obtained adversarial inputs for different values of ϵ. Make sure to include exemplary inputs in your report. *Hint: You can compute gradients by using tf.GradientTape and the corresponding .gradient method.*

4. Model Explainability

Deep neural networks are often considered black-box approaches. Model explainability methods like *Integrated Gradients (IG)* shed some light on the inner workings of the trained models by highlighting the input regions most relevant for a given prediction.

4.1 The *IG* method from `https://arxiv.org/abs/1703.01365` achieves this by integrating the gradients with respect to the input $\mathbf{x}$ along the direct path from a suitable baseline $\mathbf{x}^{'}$. Using a Riemann approximation with M steps, the attribution vector $\mathbf{a}$ is then given by:

$$a_i = (x_i - x_i^{'}) \times \sum_{k=1}^{M} \frac{\partial F(x_i^{'} + \frac{k}{m} \times (x_i - x_i^{'}))}{\partial x_i}, \tag{2}$$

where $F(\cdot)$ denotes the trained neural network. Study the above paper and implement *IG* in *tensorflow. Hint: The procedure to obtain the gradients with respect to the inputs is equivalent to 3.1.*

4.2 Select exemplary inputs and apply *IG* to the models from 1.2 and 2.3 respectively. Elaborate on the role of the baseline for the considered classification problem. Then use your *IG* implementation to study the adversarial inputs from 3.1, i.e., assess how the feature importance changes when *FGSM* is applied.