**Demonstration report**

**Data generator**

The first step is loading the data. The dataset is split into 6 batches that are loaded using the following function:

Text

Description automatically generated

The images and labels are stored in a pickle file, and the images in the original format have the channels on the first dimension, and we convert them to images with channels on the last dimension.

In order to integrate the data generator with the Keras functionalities, we define the following class:

Text

Description automatically generated

And in the constructor, we read the batches and normalize them in the 0-1 range by dividing with 255:

Text

Description automatically generated

Also, we have the option to shuffle the dataset:

Text

Description automatically generated

This will be called automatically by Keras after each epoch.

The data is read in batches:

Text

Description automatically generated

And the length of the dataset is given by the number of batches:

Text

Description automatically generated

**Models**

In order to create the model architectures, we define the following functions for creating dense and convolutional layers:

Text

Description automatically generated

In order to create the actual models, we use a functional style of defining them. We define the ANN as:

Text

Description automatically generated

The hidden\_layers parameter controls the number of units in each hidden layer. Also, a final dense layer is needed to output the logits for the classes.

In a similar fashion, we define the CNN:

Text

Description automatically generated

But here we also use BatchNormalization layers, and in order to get the desired number of outputs we use a pointwise convolution in order to control the number of channels and followed by a GlobalAvgPool in order to get the vector of logits.

**Training**

The next step is the training for which Keras handles most of the job:

Text

Description automatically generated

The model is either the ANN or the CNN which are created using the functions described in Models.

We use the following optimizer and callbacks:

Text

Description automatically generated

**Evaluation**

In order to perform the evaluation, we first generate a json file containing the ground truth and predicted labels, together with the probabilities for all classes.

This is done on each batch, by loading the corresponding data generator and model as follows:

Text

Description automatically generated

After this we compute the results for each image:

Text

Description automatically generated

And save them as follows:

Text

Description automatically generated

We compute the following metrics: accuracy, precision, recall, f-score and AUC.

In order to compute them, first the confusion matrix must be computed:

Text

Description automatically generated

Both pred and gt are lists containing the labels, but the labels can also be seen as indices in the matrix, therefore at each position described by the pair of indexes (pred, gt) we add 1.

Accuracy:

Text

Description automatically generated

Precision:

Text

Description automatically generated

Recall:

Text

Description automatically generated

F-score:

Text

Description automatically generated

AUC is a binary metric, therefore in order to compute we implement a one vs all approach by taking each class and considering it positive, and the rest negative. First, we implement binary AUC:

Text

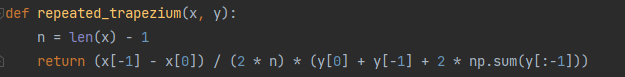
Description automatically generated

Then for all classes:

Text

Description automatically generated

The actual AUC is an integral and we use the repeated trapezium formula to compute it:



In the end, for all metrics, except for accuracy, we record the values for each class and the mean in a json file.

This is done in a cross-validation manner, by computing them on each batch. Finally, we compute the means and confidence intervals batch wise:

Text

Description automatically generated

**Utils**

Useful functions:

Text

Description automatically generated