**Are you afraid of the dark?**
Version: 1.1

**Problem**:
Our client is using ML for person identification. Users show their face, an image is taken and used for identification. The identification itself is done in the cloud by a 3rd party provider. The client pays that provider for every identification attempt. That means that sending images that don't contain faces is an issue. Such images will be either rejected or will give unpredictable results (which is very, very bad).

To solve the issue we need to build an identification pipeline to filter out invalid images, those
that don't have faces in them etc. This way only promising images are actually sent.

**Work to be done**:
Your goal will be to investigate part of that pipeline. It was observed that users often are in dark rooms when going over the identification process.
Obviously, that makes identification much harder so making sure "dark" images are dropped is a crucial step.

What you need to do is to write a simple application that, when run, will go over the provided input directory, read all jpg and png files, analyze them and write them back to an output folder with metadata attached. The output and input directories are read from a config.
This metadata is coming from the analysis process and has two pieces of information:
- one of two labels: "dark" or "bright", where "dark" are images that should be rejected according to the algorithm
- a score between 0 and 100 (inclusively) showing how dark the image is. So if a perfectly white image goes into the process a 0 score should be assigned, and when a perfectly dark image goes into the process a 100 score is given

As you probably already noticed those 2 metadata aren't fully independent. There will be a "cut off" point. All images with scores above will be considered "dark" and all images with scores below will be "bright". As programming is often exploration we don't know what value will make the most sense, so make sure this value can be easily changed between runs.

As mentioned above "business" doesn't know beforehand what "dark" precisely means and when images stop being "dark". Instead they did provide us with a set of images divided into "dark" and "bright". Looking at the dark images should give you some intuition on what you are looking for.
(Zip is attached to this task)

When creating your solution make sure that as many images as possible are correctly associated. Obviously, the solutions should also work with images not provided in this set.

The client is operating at web-scale so, when correctness is assured, make sure you use parallelism to achieve good throughput when processing many images.

When submitting the task we expect to receive from you:

- a score showing how many images from the demo set are correctly associated by your solution
- the source code with demo images included ready to be run

**Assumptions**:
To make your task easier here are some assumptions.
Both "in" and "out" directories will exist, so creating them is not needed. The "in" directory will be non-empty and "out" will be empty when the process starts (no name collisions will happen).

The "in" folder is "flat" - doesn't contain any subfolders. It also doesn't hold any files other than jpg and png.
It's good enough if the app is started with a simple sbt run (or similar in your build tool of choice). The example photos should be added to the repository so the app can be started with just one command.

**Example**:
When we run the script given an input folder “in”, output folder “out” and configured “cut off“ point of 55, then the final result should be this directory structure:

```
in/
  perfectly_white.png
  perfectly_black.jpg
  colorful_image.jpg
out/
  perfectly_white_bright_0.png
  perfectly_black_dark_100.jpg
  colorful_image_dark_55.jpg
```

**Criteria**:
Your solution will be judged based on those criteria:
- it should be on time: it’s ok to ask for more time to polish a solution that delivers value (say: works but needs documentation, works slowly or correctly classifies only 75% of images etc.), but it’s undesired if you need the additional time to get a solution to work at all
- code quality: the code should reflect your programming skill
- precision: your solution should strive to correctly classify 100% of the provided images, but doesn’t have to work for ANY image

Obviously, the solution should be your own. It could be inspired by existing research, tools or algorithms, but you should be the author. Dishonest candidates will be removed from the process.

Good luck