Camera model identification

This project was managed with github kanban, which can be accessed here. There are also mini-milestones.

*All patches generated were **512x512px**, as resizing could interfere with unique footprint characteristics of camera sensors. Validation dataset made out of train was always used for evaluating overfit. (20% of train dataset was used as val dataset)*

v1.0

- First milestone was about:
  - researching the discussion forums and any other source of material that could be help in understanding current progress and underlying problem of how images are formed
  - Creating a basic pipeline dataset ingestion and model_training without any bugs
- **Implemented**:
  - Basic CNN and FCN classifiers were implemented as opposed to pretrained models mentioned in the forum since at first it made sense that shallow models could perform better. Reasoning behind this was that we don't need deep and complex features of RGB images but simple relationships between neighbouring pixels
  - At first only a single center crop per image dataset was created. After this multiple random crops with augmentations (only transformations mentioned on the competittion page) were added

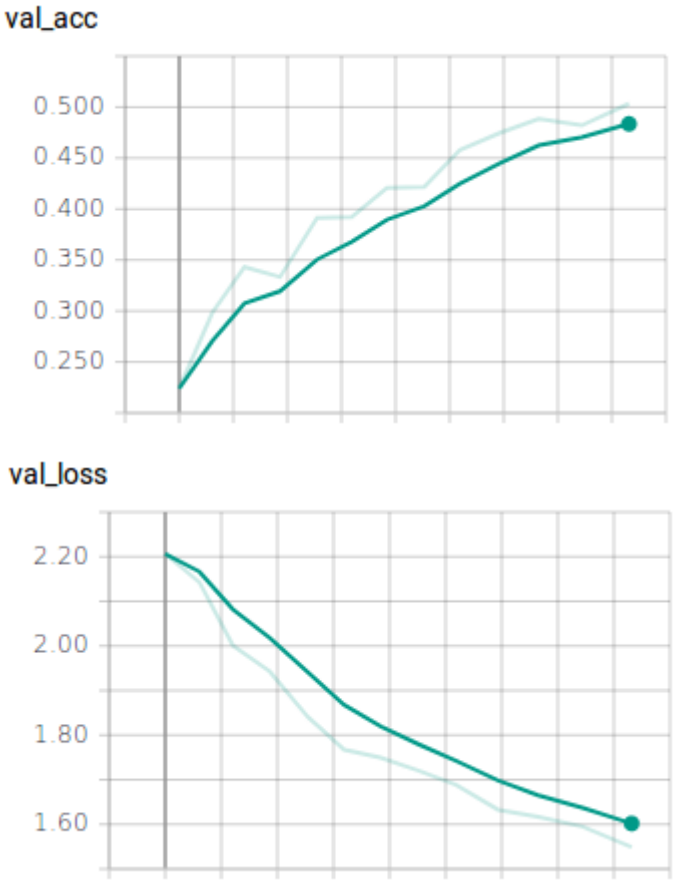
v2.0

- Idea 1: Since shallow models haven't worked, using pretrained models looked promising.
- Idea 2: Use FFT transform instead of raw images, and train on this. Different sensors should have different spectral properties.
- **Implemented**:
  - Basic headless pretrained ResNet, InceptionNet trained with NN on top of it with a large overfit
  - ~~2d FFT dataset generation~~ dropped in favor of v3
  - ~~Train on FFT dataset~~ dropped in favor of v3

v3.0

- Use headless advanced classifier archictectures (ResNet, DenseNet, InceptionNet) for generating high level features from images. Since models heads were chopped, this model was named **Guillotine** :D Use this features as inputs to a smaller FCN/CNN.
- **Implemented**:
  - At first only headless ResNet features were used, this setup learned nicely and I've stopped it when it reached 22% val acc, from where I've continued with down below.
  - High level features were saved in a hdf5 as a dataset and before feeding into a network were concatenated in a single vector of ~600000 features. Simple FCN with 2 layers was trained on that large vector. This model performed the best (0.50 categorical_crossentropy val_accuracy and 0.28

LB score), but unfortunately didn't cope well with augmented images since it was trained only on non-augmented train images.



Highest scoring model: Guillotine 0.28 weighted categorization accuracy (354. LB place woohoo)

Notes:

- All training was manual, no hyperparam search was used, which is very time consuming
- Very limited hardware: Couldn't train v3 with augmentations since it would take too much space (60GB ssd limit)
- Code got a bit bloated towards the deadline
- Weights for the model are around 1GB, at the moment of sending this report I don't have access to the PC were they are stored but will put them on gdrive and send you a link later today

Further ideas:

- Try training on FFT data, if it works at all add it to v3 features
- Train v3.0 longer since it had a nice val_accuracy on the rise
- Try different combinations of high-level features , e.g. instead of ResNet-InceptionNet-DenseNet use just Resnet-InceptionNet, or add new architectures as DenseNet201 Resnet150 etc..
- Introduce more augmentations, flips/rotations/compressions
- Use additional datasets from internet

Folder structure:

- `root`
  - `files` - raw dataset .zip files
  - `data` - unpacked and preprocessed folders
    - `vanilla` - raw unzipped .jpg images
    - `single_patch` - single center 512x512 crop per image, stored as hdf5
    - `aug_patch` - multiple random 512x512 augmented crops
    - `guillotine` - ResNet,DenseNet,InceptionNet features in hdf5
  - `logs` - tensorboard logs
  - **dataset.py** - dataset creation tool
    - *class AugPatchDataset* - Keras sequence class, that reads aug_patch/*.hdf5 files
    - *class SinglePatchDataset* - Keras sequence class, that reads single_patch/*.hdf5 files
    - *class GuillotineDataset* - Keras sequence class, reads high-level features of advanced models
    - *class DerivOutDataset* - Keras sequence class, reads high-level features of ResNet model
    - *class CIDDataset* - Creates all other datasets by creating .hdf5 training and val files
  - **pretrained_resnet.py**
  - **simple_nn.py** - simple neural network models
  - **summary.md** - source of this document