SEE Hackathon - Maciej Krajewski, Paweł Magnuszewski

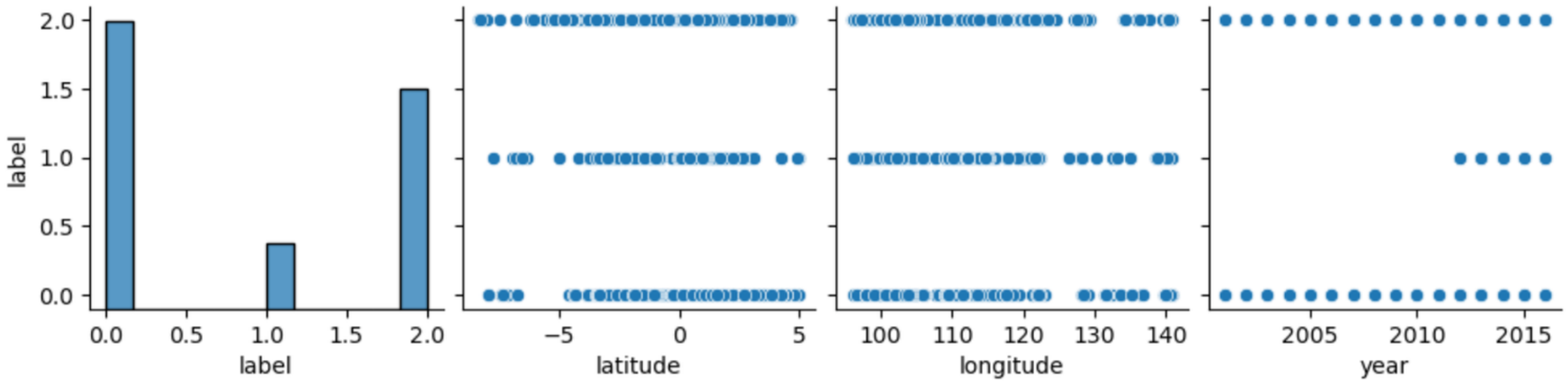
Zero Deforestation Emission

Process

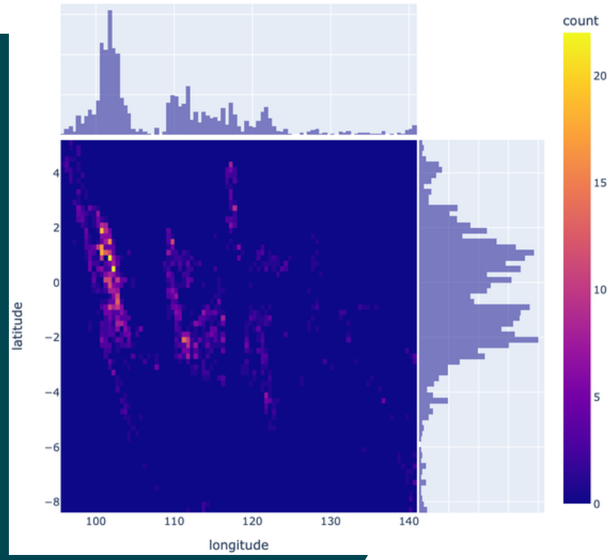
1. EDA of numerical data in .csv files
2. EDA of images image preprocessing
3. Experimentation and implementation
4. Training simple ML models
5. Training and experimenting with CNN models

EDA

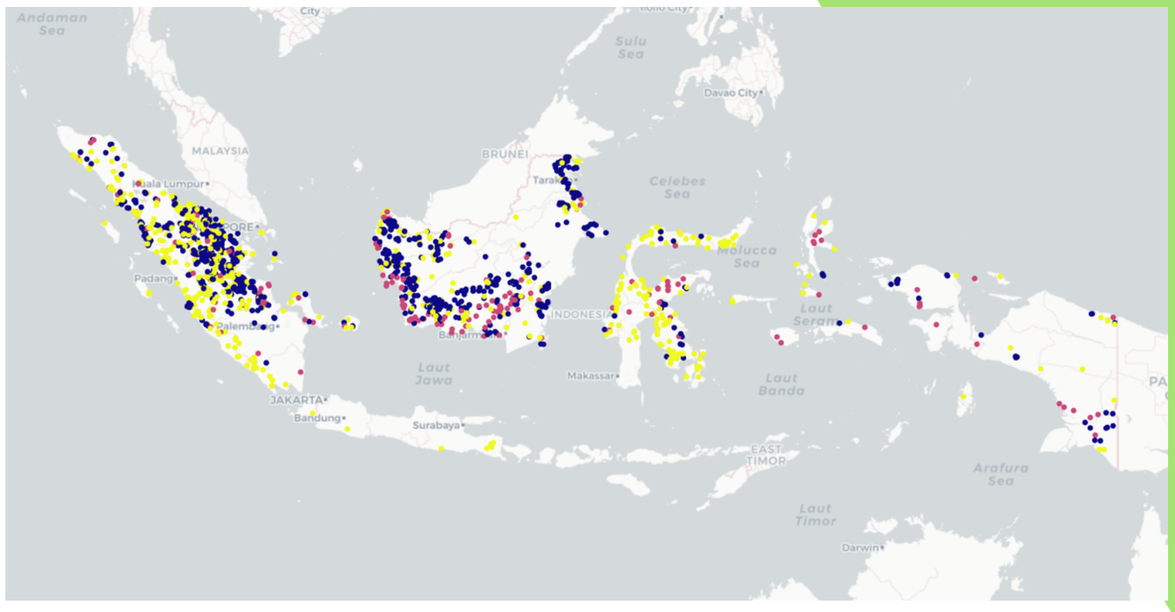
Pair plots using seaborn



Density map using plotly



Scatter map using plotly



Conclusion and further EDA on images

Our conclusions after EDA

- data seems evenly distributed geospatially
- class distribution is not even
- datapoints are all located in Indonesia
- there is no obvious correlation between year, longitude, latitude and label
- label 1 only occurs after year 2012

Images preprocessing

Our images were:

- converted to HSV color space
- the V value histogram was normalized using OpenCV
- converted to BGR color space
- then convoluted using two perpendicular Sobel filters

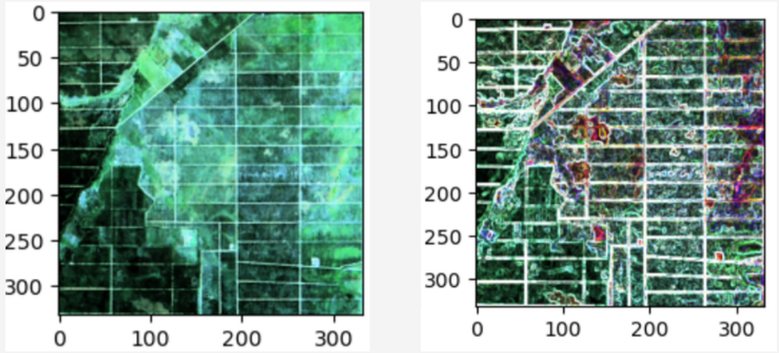
Simple ML models

We trained some simple models using only latitude, longitude and year data from the .csv files. Below are the best results for our classifiers:

- Decision Tree Classifier: val_f1 ~ 0.58
- Random Forest: val_f1 ~ 0.67 (but heavy overfitting)
- Gradient Boosted Classifier: val_f1 ~ 0.62
- Naive Bayes: val_f1 ~ 0.53

We plotted some images from all classes, researched how to approach satellite imagery data for classification and concluded:

- it would be nice to calculate Normalized Difference Vegetation Index but since our images were in .png and not in .tif we were unable to do so
- all images were very dim, so we decided to equalize their histograms both in Y_CrCb and HSV color spaces, but eventually decided to stick with the HSV approach
- we came to the conclusion that edges would be important in our classification task so we ran images through two perpendicular Sobel filters and took combined filtered image, we also tested the Canny filter but we couldn't figure out how to make adaptive thresholds for it



Picture after hist norm and Sobel filters

Experimenting with CNN

We experimented with different CNN architectures but decided to stick with:

Conv2D(16,(3,3))
MaxPool2D((3,3))
Conv2D(16,(3,3))
MaxPool2D((3,3))
Conv2D(16,(5,5))
MaxPool2D((3,3))
Flatten
Dense(256, relu)
Dense(64, leaky_relu)
Dense(8, leaky_relu)
Dense(3, softmax)

On non preprecessed data we achieved val_f1 ~ 0.6
On preprocessed data we achieved val_f1 ~ 0.68