Sion Kim

1.1 Introduction

Our goal is to build a car license plate object detection model using car images. This is an experimental project to explore Dataiku's ability & limitations. Hence, from labeling to model training, we experiment various tasks on the platform.

Sample image used

1.2 Methodology

Building an object classification model in Dataiku is quite simple. We can generalize processes in 4 steps:

1. Gather input sources to train model with
2. Transform input sources into the form we want using recipes. For example, gray scaling or thresholding images.
3. Label the images. This could be done manually in Dataiku using the image labeling recipe, or the user could upload a dataset containing annotation information and merge source images with the dataset in Dataiku.
4. Train & build object classification model using Dataiku's visual analysis tool

1.3 Data used

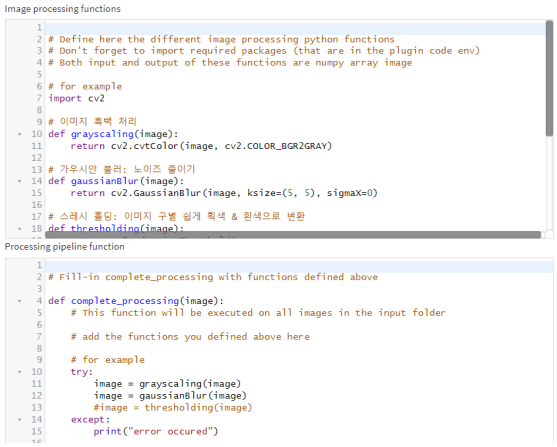
Cars with license plate images totaling 180 are manually extracted from Google. It is divided into training and testing sets in 8:2 ratio.

2.1 Pre-processing image

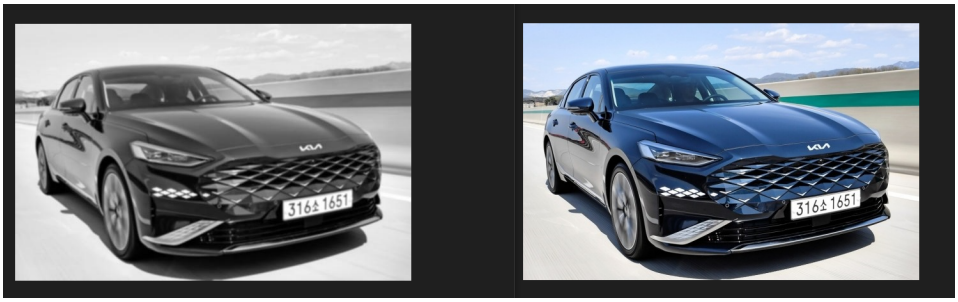
When we initially started this project, the first thought was to use OCR on raw images. However, real-life scenarios aren't ideal; images could include texts other than number plates, such as stop signs or banners. Furthermore, gray scaling simplifies algorithms and as well eliminates the complexities related to computational requirements. This is why we had an image processing step ahead of building models.

Using the Tesseract OCR plugin, it is possible to process input images in our favor. This plugin has multiple components and one of them is image processing recipe. It has image processing functions and pipeline, which is handy to convert images in our favor for input image optimization.

To further explain custom plugins, Dataiku provides custom plugins through the plugin store. There are currently 146 plugins in store for various use cases: cloud, NLP, data enrichment, etc. If the user feels the need to develop one for some reason, they can develop one referencing documentation or by converting a project into an application-as-recipe.



Writing functions and pipeline in image processing recipe



Comparison of raw and processed image

2.2 Image labeling

In Dataiku, we can label images at ease without leaving the platform. The platform provides a good image annotation recipe, which consists of 2 parts: annotating and validating. Given a data preparation tool, companies would hire labelers to annotate the image and reviewers could either accept/reject annotations.

Time cost would depend on the complexity of the project, but in this case it only took about 3~5 seconds each.

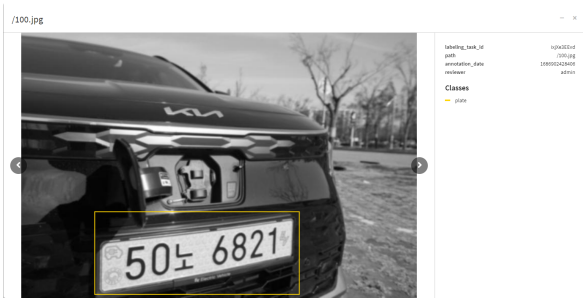


Image annotation process using label recipe. Yellow box indicates label area

2.3 Image labeling limitations

Currently, Dataiku doesn't provide advanced annotation tooling. Annotators can only use square boxes, which can result in low model accuracy (think of instances where we have to label traffic signs, which are circles). In comparison, V7 Labs provides automatic annotation and various annotation shapes, which has advantages of precision in pixels and labeling speed. Also, Dataiku is missing on pre-annotated datasets (data collections that have already been annotated), while Roboflow and V7 Labs provide such feature on common

objects (cars, human, or hand-writing). These downsides directly connect to capital cost and time efficiency, so if the user is mainly focused on building computer vision models, it might be better to use products mentioned above.

2.4 Post labeling

After the labeling step, a new dataset is created. This dataset contains label information on each image, including task ID, reviewer information, label, etc in formatted manner. It's necessary to have this dataset for object detection model training, since it requires raw images and annotation information.

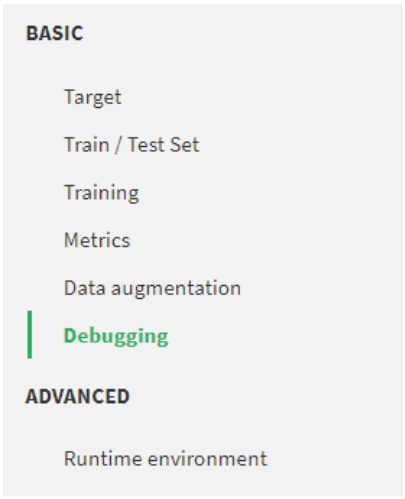
labeling_task_id	path	annotation_date	reviewer	label
string	string	bigint	string	string
Text	Text	Integer	Text	Array
j6KqRxxLjH	/11081.jpg	1686639466252	admin	[{"category":"license_plate","bbox":[630.0,1712.0,275.0,185.0]}]
j6KqRxxLjH	/11083.jpg	1686639466253	admin	[{"category":"license_plate","bbox":[2614.0,1181.0,446.0,330.0]}]
j6KqRxxLjH	/11084.jpg	1686639466255	admin	[{"category":"license_plate","bbox":[2609.0,979.0,154.0,254.0]}]
j6KqRxxLjH	/11086.jpg	1686639466256	admin	[{"category":"license_plate","bbox":[141.0,479.0,95.0,55.0]}]
j6KqRxxLjH	/11088.jpg	1686639466257	admin	[{"category":"license_plate","bbox":[436.0,1265.0,395.0,273.0]}]
j6KqRxxLjH	/11089.jpg	1686639466258	admin	[{"category":"license_plate","bbox":[180.0,655.0,239.0,119.0]}]
j6KqRxxLjH	/11090.jpg	1686639466265	admin	[{"category":"license_plate","bbox":[1297.0,455.0,188.0,171.0]}]
j6KqRxxLjH	/11092.jpg	1686639466266	admin	[{"category":"license_plate","bbox":[289.0,355.0,131.0,71.0]}]
j6KqRxxLjH	/11093.jpg	1686639466271	admin	[{"category":"license_plate","bbox":[2680.0,1399.0,570.0,363.0]}]
j6KqRxxLjH	/11097.jpg	1686639466273	admin	[{"category":"license_plate","bbox":[289.0,966.0,696.0,388.0]}]
j6KqRxxLjH	/11100.jpg	1686639466274	admin	[{"category":"license_plate","bbox":[1159.0,618.0,188.0,140.0]}]

Dataset created after annotation

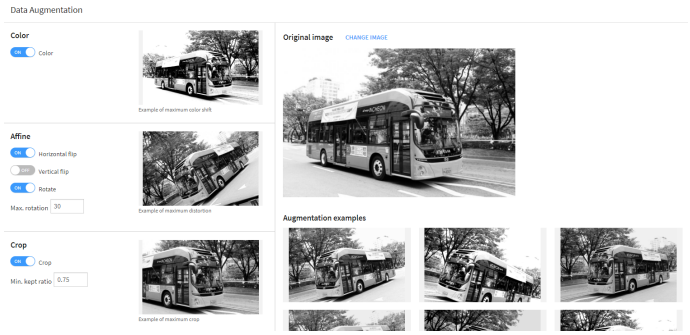
3.1 Training and building object detection model

Dataiku, by default, contains an object detection recipe. It is a no-code tool, which provides users with visual insights and customizable configurations. Users can configure training targets, train/test set, training options (model, optimization, and fine-tuning), metrics, and data augmentation.

Data augmentation comes in handy through adding additional diversity to the training dataset by applying distortions to images that could be expected in the real world.



Configurations available in object detection recipe



Data augmentation

3.2 Limitations on no-code recipe

The only model available for object classification is the Faster R-CNN, and is used by default. This object detection model works well for production use, but it came out in 2015. Since then, better models (in terms of speed and accuracy) have been introduced. Of course, a user could implement models they want using Python recipes, but it is a complicated and time-consuming process, and is not recommended for following reasons: data prediction (score) and evaluation recipes can't be chained with a model built with a Python recipe, so the user would have to write another Python recipe to do so, which leaves the project with less readability and scalability. I expect the Dataiku team will add up-to-date models in visual recipes in the future, based on constant updates in newer versions.

I would like to emphasize that machine learning models built with visual recipes come with a scoring API, which can be utilized in a production environment. For instance, a user can build a real time application to upload images periodically and fetch scored results using Dataiku's API feature, which takes considerable time and complexity just setting up API endpoints in traditional development.

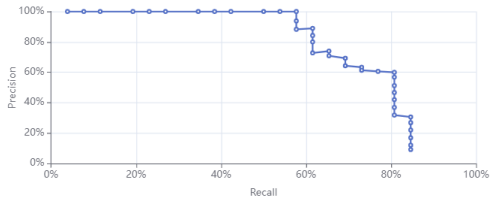
3.3 Need of graphic cards for model training

Training model took 47 hours with the local server's CPU resources. Note that time taken to train could have been significantly improved with the use of graphic cards. Also, it is recommended to use the platform with cloud providers, because integration allows deployment of complex architectures in a fully managed fashion (auto-scaling, security management, etc). Therefore, before the company starts a project, it is a good idea to think about whether they will need to use a DSS instance in a local environment or cloud. Google Cloud has a Dataiku Enterprise Ready AI product for $6,000 per month with no additional charge for usage, which is worth noticing. This article discusses leveraging Dataiku on on-premise and cloud.

3.4 Training result

Despite low model training settings, we were able to achieve decent average precision (0.823; IoU=0.5). As shown below, we can now score images with a trained model, which will provide us with predicted plate location. With this information, we can now extract plate numbers (consisting of Korean letters and Arabic numbers) in later steps.

Object detection model training summary



Precision-Recall matrix (IoU=0.5)

Scored test images

3.5 Limitations on resources

Since this project is a proof-of-concept to demonstrate what users can build in a matter of minutes using Dataiku's capability, rather than focusing on model accuracy, we used minimal input data (180 images) and low training settings. This can easily be improved with larger inputs and the use of graphic cards. Auto-ML tasks with low features can be done in less than an hour, but computer vision tasks or deep learning on Dataiku is resource intensive. Hence, it is necessary to identify what kind of models the project will need and use the appropriate computing resources in order to balance out time on training.

4.1 OCR (Optical character recognition)

Optical character recognition is a method to extract texts from an image. There are various kinds of OCR packages available as open source. In this project, we use Easy OCR, Paddle OCR, and Tesseract OCR through a Python recipe. When writing code recipes, the Jupyter notebook comes in as default, and this allows developers to see the code results in-line and boost productivity. Easy OCR was easiest to use among all, because it required only a few

dependencies and minimal code. All three were configured to detect Korean and numerics on cropped license plates.

```python
# -*- coding: utf-8 -*-
import dataiku
import pandas as pd, numpy as np
from dataiku import pandasutils as pdu
from PIL import Image
import easyocr

# this needs to run only once to load the model into memory
reader = easyocr.Reader(['ko'])

# Read recipe inputs
processed_plate_images = dataiku.Folder("sXkzfqBm")
processed_plate_images_info = processed_plate_images.get_info()

res = []

for img in [item["fullPath"] for item in processed_plate_images.get_path_details()["children"]]:
    with processed_plate_images.get_download_stream(path=img) as stream:
        im = Image.open(stream)
        result = reader.readtext(im, detail = 0)
        res.append({"file": img, "text": "".join(result)})

# Compute recipe outputs
# TODO: Write here your actual code that computes the outputs
# NB: DSS supports several kinds of APIs for reading and writing data. Please see doc.

extracted_texts_easyocr_df = pd.DataFrame(res) # Compute a Pandas dataframe to write into extracted_texts_easyocr


# Write recipe outputs
extracted_texts_easyocr = dataiku.Dataset("extracted_texts_easyocr")
extracted_texts_easyocr.write_with_schema(extracted_texts_easyocr_df)
```

Easy OCR implementation through Python code recipe

4.2 OCR results

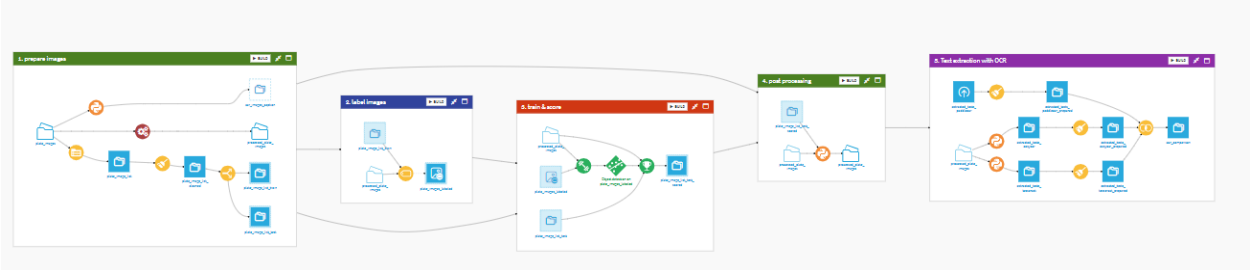
We tested 3 libraries on the same images and the results are shown below. Easy OCR and Paddle OCR had noteworthy accuracy, but not in production level, while Tesseract OCR had the worst character detection among three in our usage. This paper argues that Tesseract OCR's accuracy for English recognition is high, but Hangul has less training data due to its complex structure, so the accuracy is significantly lower. One way to improve recognition accuracy is to train OCR models specifically with plate characters. Or if the operating cost is manageable, paying for CLOVA OCR usage through API would be a great option as it is one of the best performing OCR providers.

file	text_easyocr	text_paddleocr	text_tesseract
string	string	string	string
Text	Text	Text	Text
175.jpg	284조2644	284조2644	2 84조 2 644 .
61.jpg	59부르6739	359부6733	
33.jpg	1405750	14허5750	`「、¶1 4m 5 7...
176.jpg	179우1483	17991483	
154.jpg	176부2816	176부2816	를 1 76부 2 8 1 6탤

OCR results on test images

5.1 Closing notes

In this project, we explored Dataiku by building a license plate recognition model. It is a great platform to perform collaborative visual machine learning at ease. While it has some sorts of limitations, we believe this is a highly utilizable platform for both traditional programmers and non-programmers in various use cases.



Overview of the project

5.2 Pros and Cons

Pros:

- Good for ML OPS transparency and collaboration (with version control)
- Easy to build and deploy machine learning model
- Can be fully managed when integrated with Cloud providers
- User can visualize and orchestrate workflow
- Fast-paced model deployment using visual recipes

Cons:

- Lack of pre-built model options
- Some visual recipes lack advanced usage
- When used in local, code environment configurations, DSS instance update, and writing custom Python recipe could be tedious