Big Data Tank classification

Report

Big Data - ITFactory

JAD Solutions (Team 1):
Axel Van Gestel
David Silva Troya
Jetze Luyten

Academic Year 2022-2023

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

LID VAN ASSOCIATIE KU LEUVEN

THOMAS MORE

TABLE OF CONTENTS

1 A WORLD OF TANKS

There have been a lot of wars on earth, some of them with tanks and some without. Tanks are slow (compared to cars) armored vehicles that shoot a shell that can explode on impact. Tanks move with tracks instead of wheels. They have a couple of different ammunition types (explosive and non-explosive), which results in different barrel diameters and lengths. Tanks can also have different armor types (sloped, vertical, rounded).

We chose tanks as our dataset, because there is a war going on between Russia and Ukraine. But because of the little difference between Russian and Ukrainian tanks and the lack of scrapable images we chose to use different German WW2 tank models.

We have chosen the following 5 German WW2 tanks:

- Hetzer
- Tiger I
- Panzer II
- Panzer IV
- Panther II

2 GETTING THE DATA

2.1 Getting images from the frames of videos.

One of the ways was by taking frames from videos with a python script using the cv2 library. This was giving a big number of pictures, also related with the duration of the video but one of the problems was that the frames or pictures were similar from each other because of the sequence of the video and setting an interval to save fewer frames was also resulting in information lost since sometimes the video could be showing the tanks for a little moment or completely the opposite, for a long time in the same position. The quality of the video, type of video and duration were in fact important parameters to get high quality images. In the end we didn't use this method in the creation of the dataset.

2.2 Scraping images from Google

After experimenting with scraping images from videos and the failure of it we went with a more traditional approach to scraping a dataset together. Using the scraper, we needed to write during the AI Project course we scraped the initial dataset. Some tanks like the Hetzer also had code name, Jagdpanzer 38(t), so we scraped for both terms and later combined them into one category.

2.3 Cleaning the data

Scraping straight from Google results in quite a dirty dataset, so to clean it up a bit we manually went through each image to see if it was the correct tank and crop the image if necessary. This was done in a couple of hours using a tool called Inbac.

The result was a nice and clean dataset of 1598 samples where we knew that each category only contained the correct tank model.

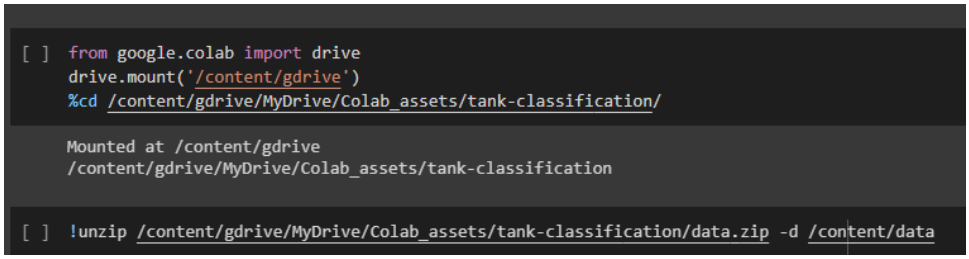
3 FastAI modeling

3.1 Introduction

FastAI is a deep learning library which provides practitioners with high-level components that can quickly and easily provide state-of-the-art results in standard deep learning domains and provides researchers with low-level components that can be mixed and matched to build new approaches. It aims to do both things without substantial compromises in ease of use, flexibility, or performance. This is possible thanks to a carefully layered architecture, which expresses common underlying patterns of many deep learning and data processing techniques in terms of decoupled abstractions. These abstractions can be expressed concisely and clearly by leveraging the dynamism of the underlying Python language and the flexibility of the PyTorch library.

3.2 Modeling

We trained 2 different models, one Quick & dirty model and another using state of the art advanced modeling options. To access our dataset on Google Colab we uploaded it to our Google Drive as a zip file and unzipped it to the session storage on runtime. This way we weren't bottlenecked by Google Drive's poor IO performance.

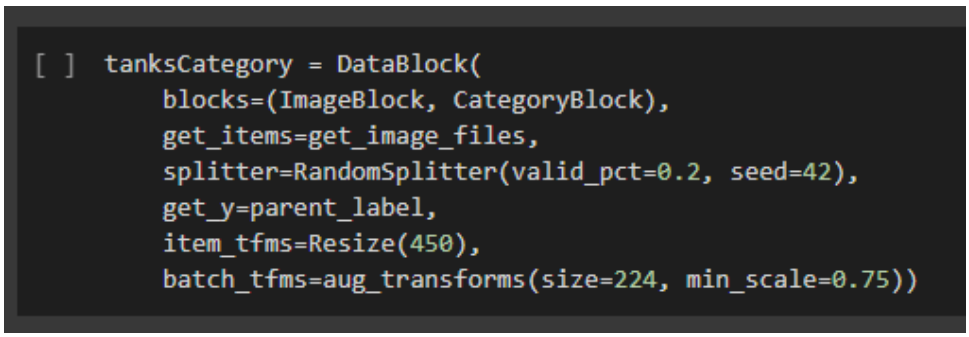


```
[ ]  from google.colab import drive
     drive.mount('/content/gdrive')
     %cd /content/gdrive/MyDrive/Colab_assets/tank-classification/

     Mounted at /content/gdrive
     /content/gdrive/MyDrive/Colab_assets/tank-classification

[ ]  !unzip /content/gdrive/MyDrive/Colab_assets/tank-classification/data.zip -d /content/data
```

In both models use a datablock of the image & category-block kind, both also used presizing to first scale our images to 450px and then do some random cropping with a size of 224px, all on the GPU in one step to keep the number of transforms needed to augment the data as low as possible.



```
[ ]  tanksCategory = DataBlock(
         blocks=(ImageBlock, CategoryBlock),
         get_items=get_image_files,
         splitter=RandomSplitter(valid_pct=0.2, seed=42),
         get_y=parent_label,
         item_tfms=Resize(450),
         batch_tfms=aug_transforms(size=224, min_scale=0.75))
```

3.2.1 Quick & dirty (Showcase on Streamlit)

3.2.1.1 Data block

In the Quick & dirty model we first added another item transformation, namely RandomResizedCrop to randomly crop the images to 128px based on a minimum scale of 30%.

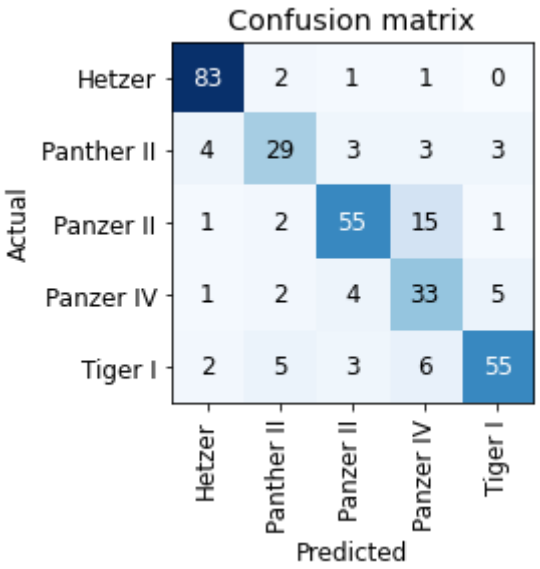
3.2.1.2 Training

After that we started training our model by utilizing resnet50 transfer-learning and just fine-tuning it a little by doing one epoch with the pretrained weights frozen, and then 4 epochs with them unfrozen.

```
our_out_of_the_box_model = vision_learner(dls, resnet50, metrics=error_rate)
our_out_of_the_box_model.fine_tune(4)
```

3.2.1.3 Result

This resulted in an accuracy of about 80% (error rate of 0.200627) and the following confusion matrix.



3.2.2 Advanced options (Google Drive on Streamlit)

3.2.2.1 Data block

For the more advanced model we added an extra batch transformation, namely "Normalize from stats" to normalize our images to a value between 0 and 1, with a mean close to 0 and a standard deviation close to 1. We did this as the resnet152 model we're going to use for transfer-learning has only seen input images in this normalized format.

Next, we omitted the extra item transformation used in the Quick & dirty model as we saw it hurt the performance more than it improved, we believe this is mostly due to the much smaller input size it resulted in (128px vs 224px). On top of that, the presizing option we use does the same thing when the minimum scale is set. This resulted in quite a bump in accuracy and kept the training and validation losses closer together to boot.

```
[ ]  tanksCategory = DataBlock(
         blocks=(ImageBlock, CategoryBlock),
         get_items=get_image_files,
         splitter=RandomSplitter(valid_pct=0.2, seed=42),
         get_y=parent_label,
         item_tfms=Resize(460),
         batch_tfms=[*aug_transforms(size=224, min_scale=0.75),
                                     Normalize.from_stats(*imagenet_stats)]
         )
```

3.2.2.2 Training: learning rate finder

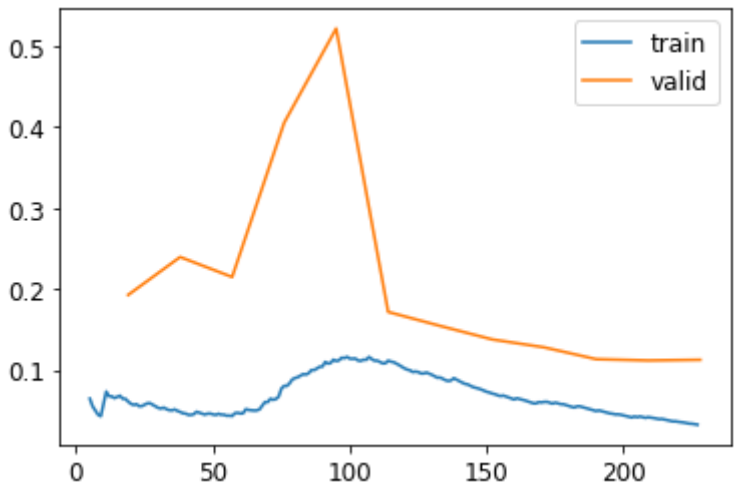
After setting up the data block and learner, we started with learning rate finder. It suggested a learning rate of 10^{-3} we then used in our first fit one cycle of 6 epochs we did, with the pre-trained weights still frozen. This has already resulted in better accuracy than that of the Quick & dirty model (87% vs 80%).

We then unfroze the pre-trained weights and ran the learning rate finder again, this time suggesting a learning rate somewhere between 10^{-5} and 10^{-4}. To play it safe we chose the learning rate of 10^{-5} and ran 6 more epochs using fit one cycle. These unfrozen epochs didn't improve our model as much as we expected (87% --> 90%).

3.2.2.3 Training: mixed precision

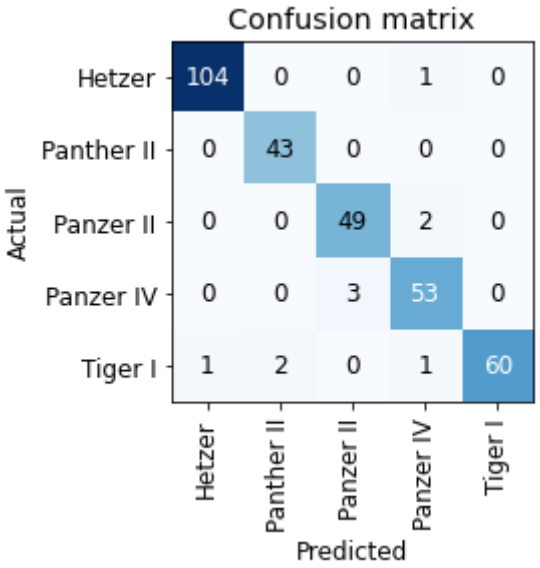
After the initial manual fit one cycle training, we converted the model to use mixed precision (fp16) and fine-tuned it using 3 frozen and 6 unfrozen epochs. This resulted in a quite a performance increase (90% --> 94%).

As we still weren't overfitting, we ran another 8 frozen and 12 unfrozen epochs. The training and validation losses, however, did increase a bit in the first few unfrozen epochs but stabilized back to normal levels in the end.



3.2.2.4 Result

The final accuracy achieved was 97% with the following confusion matrix.



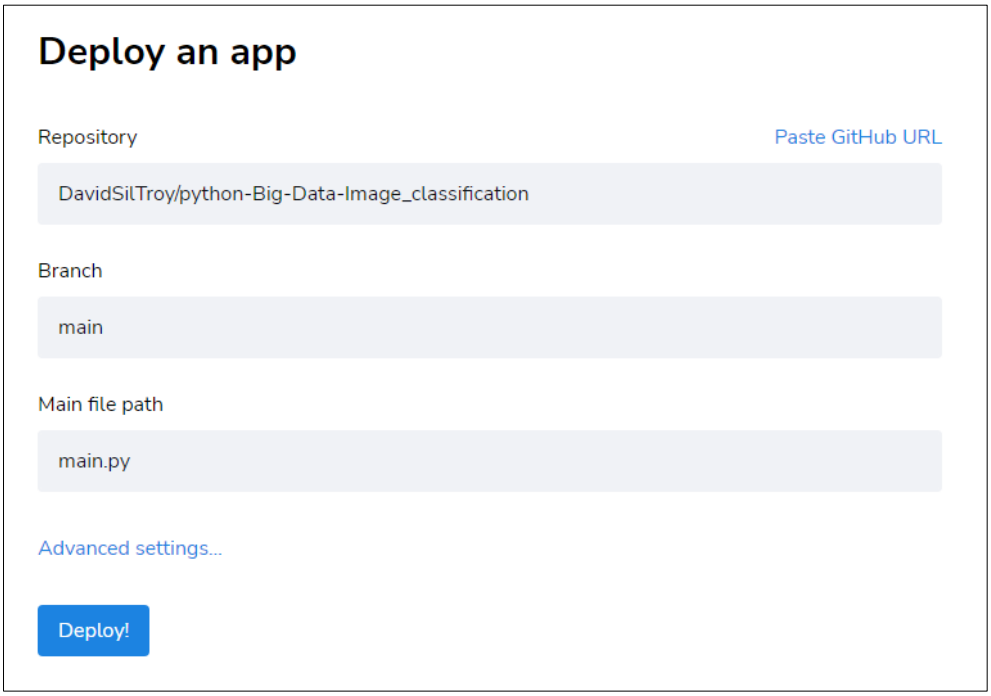
4 A STREAMLIT SOLUTION

Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science. In just a few minutes you can build and deploy powerful data apps.

4.1 Deployment

Streamlit.io allows you to deploy your project from a GitHub repository, 2 important things are to have a requirements.txt file with all the libraries needed for the program and a python file that should start the Streamlit page.

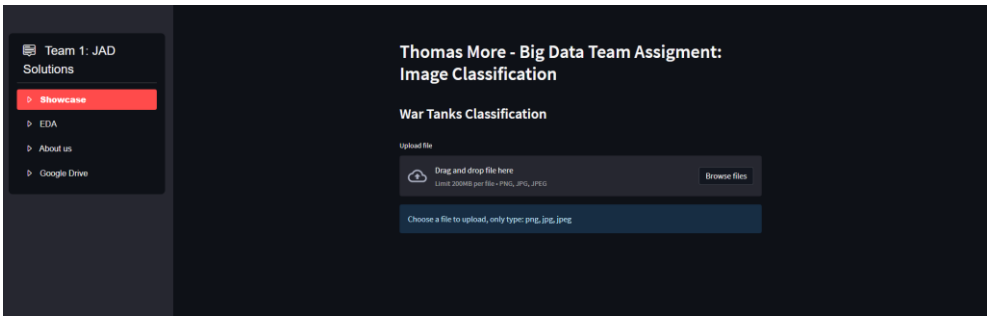
And next just paste the URL of your repository, if you copy the direction of the file where Streamlit start, then all the other fields of the form will be automatically filled. In the Advance setting it is possible to select the Python version needed too.



4.2 Application

Using Streamlit is as simple as calling the library and then using the necessary method to show text in the browser, markdown, sidebar, etc. Next is an example of the code used in our project.

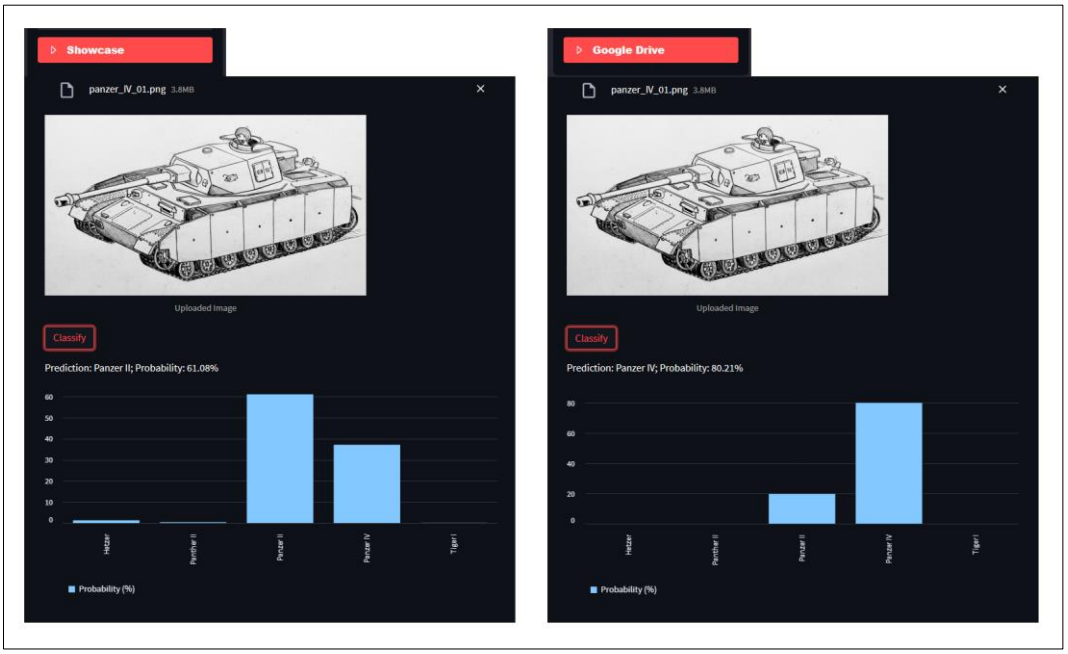
The result is a clean and good-looking User Interface to use the model trained for image classification. Easy to understand for an external user which only goal is to try out the image classification.



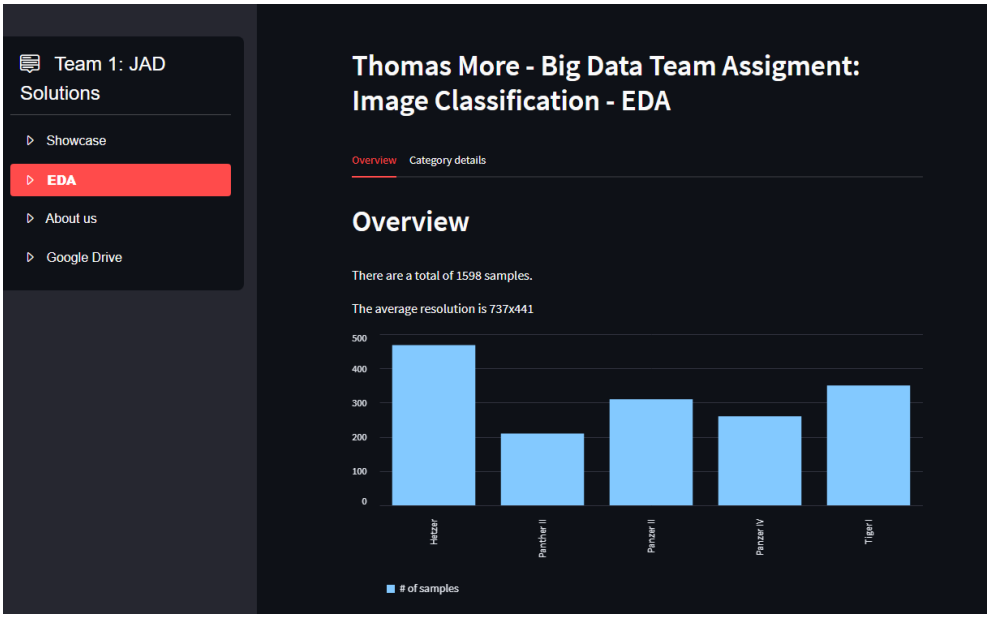
It is important to mention that GitHub has some constraints with the data we upload, being one of those the weight of the files. 100mb is the normal maximum weight for the files, then you may use Git Large Files Storage (LFS) but this one also has constrains about the bandwidth, 1GB/month.

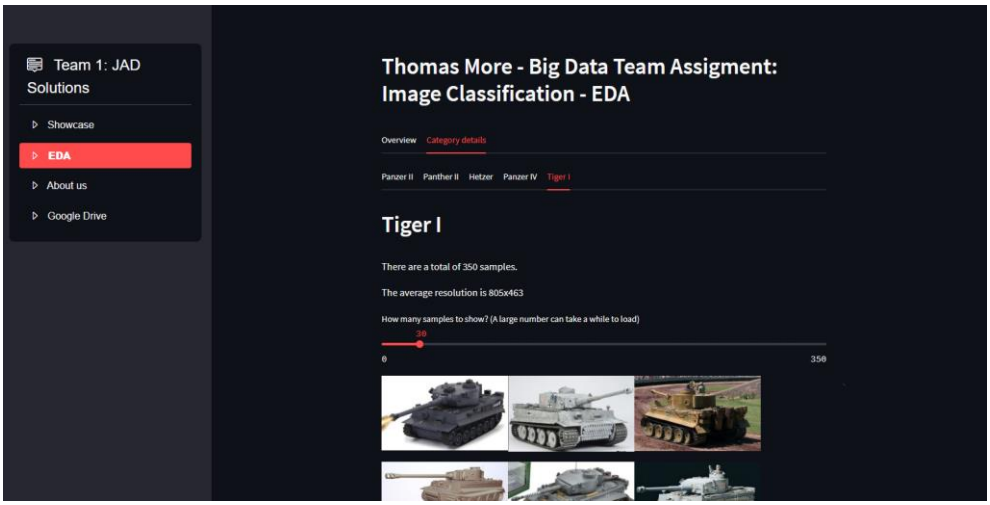
In this project we have 2 models, one trained for a short time and with a light model weight, resulting in a file with less than 100mb and easy to upload in the github repository.

To use a heavier model and after using more than the 1GB/month of bandwidth, it was necessary to use another way, in this case we use Google drive to download the model in Streamlit, this model is around 250mb, and the predictions are usually better than the model in the Showcase tab.

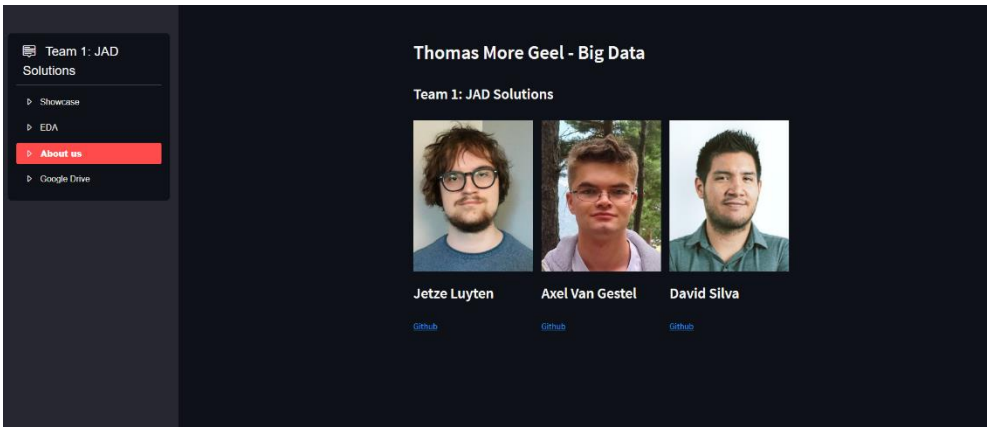


For the Exploratory Data Analysis (EDA) another tab is created to show the dataset of tanks, having also sub-tabs for every type of tank, the same that is used to train the classification model. Here it is possible to see the number of pictures and the pictures themselves.





And finally, the information about the people who made this project possible, a great team with awesome ideas and thirst for knowledge.



4.3 Problems

When we deployed our Streamlit and FastAPI web applications, the model couldn't be accessed. This was because the servers ran Linux compared to Windows we use during the localhost hosting we use during development. To fix this problem the following lines of code were needed.

```
model_name='tanks.pkl' #in case the model is moved to a folder -> 'folder/tanks.pkl'
plt = platform.system()
if plt == 'Windows': pathlib.PosixPath = pathlib.WindowsPath
model = load_learner(pathlib.Path()/model_name)
```

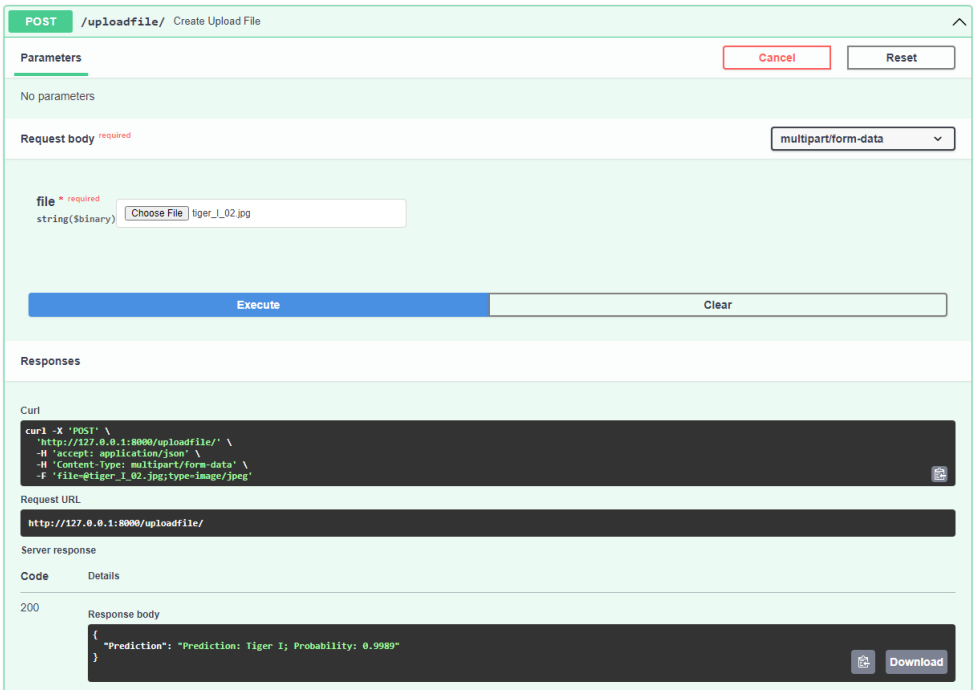
5 FastAPI

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.7+ based on standard Python type hints.

The library FastAPI and uvicorn[standard] are needed for this program. Fastapi is easy to use, just a python file is need, in this case the file is called tanks_api.py and to run it with the command line is necessary to write the next:

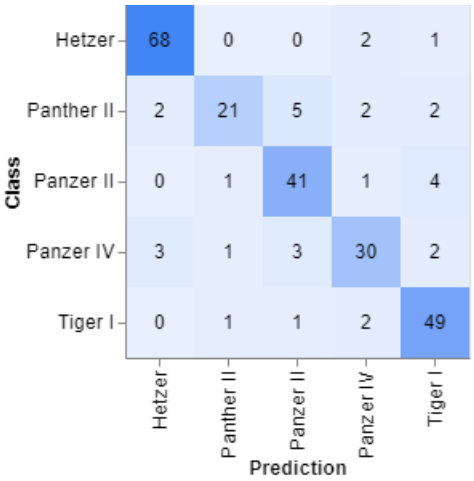
- uvicorn tanks_api:app --reload

The result will be in this case a user interface for an easy use of the API methods, here we are focused in the uploadfile method that we can use once we click in the button "try out". After that it is just choose the file and execute to have a result.



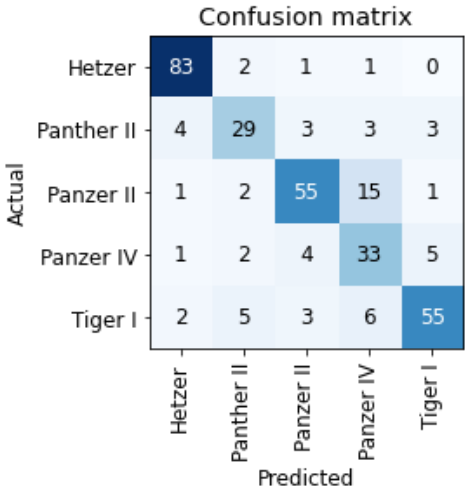
6 We vs Google Teachable machine

6.1 Google Teachable machine

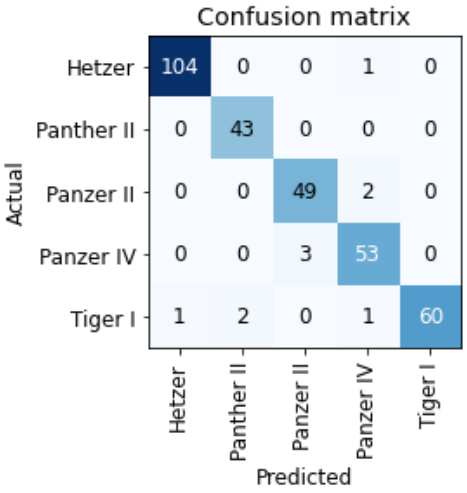


6.2 Our own models

Quick & dirty (Light weight resnet50)



Google Drive (Heavy weight resnet152)



If we compare our own 2 models against the model from Google Teachable machine, we can see the Quick & dirty model performs measurably worse than. Especially the Panzer II category gets wrongly predicted as a Panzer IV.

Compare that with the Google Drive model which measurably outperforms that from Google Teachable machine.

7 Conclusion

The start of the project was a little slow, but the rest was quite quick. Working with both Streamlit and FastAI was fun (especially Streamlit). Training the dataset got increasingly better until we got an error rate of 0.0313 (97% accuracy). Finding pictures of tanks was funny, because some images were a joke.