Deploying Neural Networks on the Web

Vladimir Haltakov

Notebook:
Sam Ray

neuromatch academy

About me

Vladimir Haltakov

- Self-driving car engineer at BMW (localization)
- PhD in Computer Vision and Machine Learning from TU Munich
- Helping people on Twitter to get started with Machine Learning
- Traveling and photography

@haltakov

Deploying Neural Networks on the Web

- Designing and training a neural network is only a part of the game
- From a pytorch model to a web application
- The Web - the easiest way to reach millions of people

- Tutorial by Sam Ray and Vladimir Haltakov
- Other contributors: Konrad Kording, Spiros Chavlis

What you are going to learn?

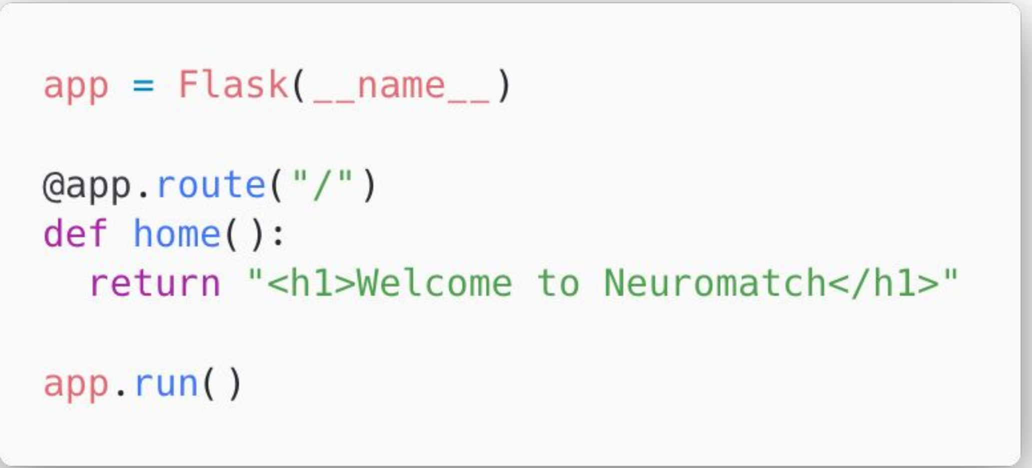
- How to use Flask for serving web pages?
- How to apply the MVVM design pattern to write maintainable code?
- How to build a REST API?
- How to create an interactive UI for your service?
- How to integrate your deep learning model?
- How to deploy your service on Heroku?

Flask

- Python web application microframework
- Lightweight, easy to use, scalable
- Big community and many extensions
- Easier to learn than Django
- Examples: Pinterest and LinkedIn

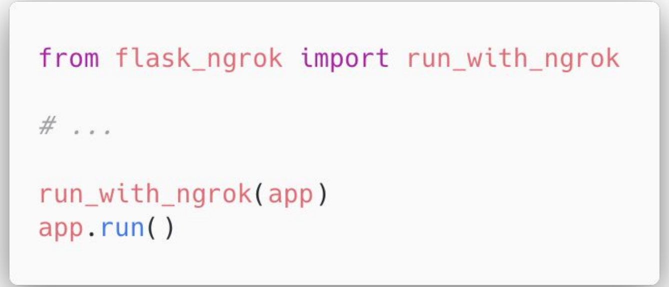
web development,
one drop at a time

Simple Flask App



```
app = Flask(__name__)

@app.route("/")
def home():
  return "<h1>Welcome to Neuromatch</h1>"

app.run()
```

Using ngrok



```python
from flask_ngrok import run_with_ngrok

# ...

run_with_ngrok(app)
app.run()
```

- The problem with running Flask in a notebook
- URL http://127.0.0.1:5000/ not accessible from the Internet
- Create a tunnel from your notebook to the Internet with ngrok
- URL like http://33ca1c4cb1f9.ngrok.io

Alternative web frameworks

- FastAPI - focused on speed
- Bottle - another framework coming with the standard library
- Tornado - an asynchronous web framework

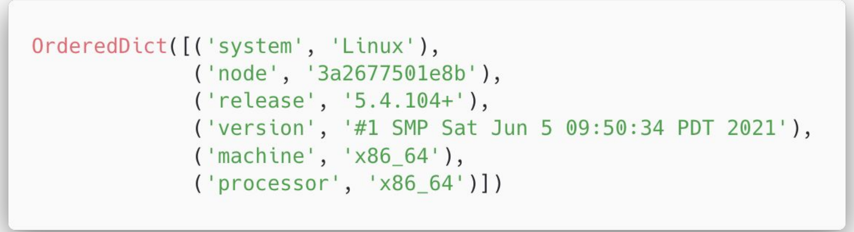
Jinja2 Templates

- Fast, expressive, extensible templating engine
- Separate HTML code and data
- Main features:

Variables	`{{ value }}`
If statements	`{% if value > 0 %} ... {% else %}`
Loops	`{% for value in list %}`
Inheritance	`{% extends "layout.html" %}`
Modules	`{% include helper %}`

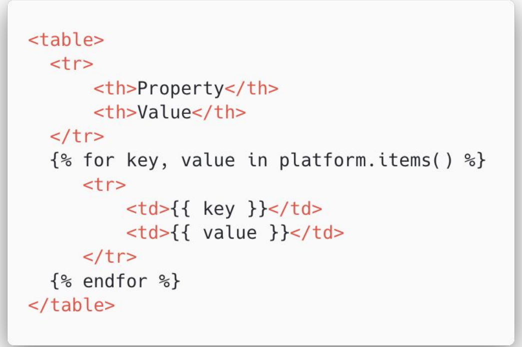
Jinja2 Templates - example

Data

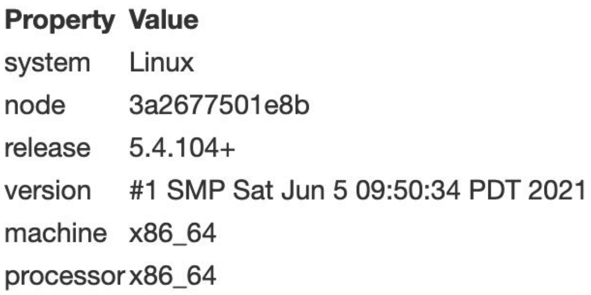


```
OrderedDict([('system', 'Linux'),
             ('node', '3a2677501e8b'),
             ('release', '5.4.104+'),
             ('version', '#1 SMP Sat Jun 5 09:50:34 PDT 2021'),
             ('machine', 'x86_64'),
             ('processor', 'x86_64')])
```

Template



```
<table>
  <tr>
      <th>Property</th>
      <th>Value</th>
  </tr>
  {% for key, value in platform.items() %}
     <tr>
         <td>{{ key }}</td>
         <td>{{ value }}</td>
     </tr>
  {% endfor %}
</table>
```

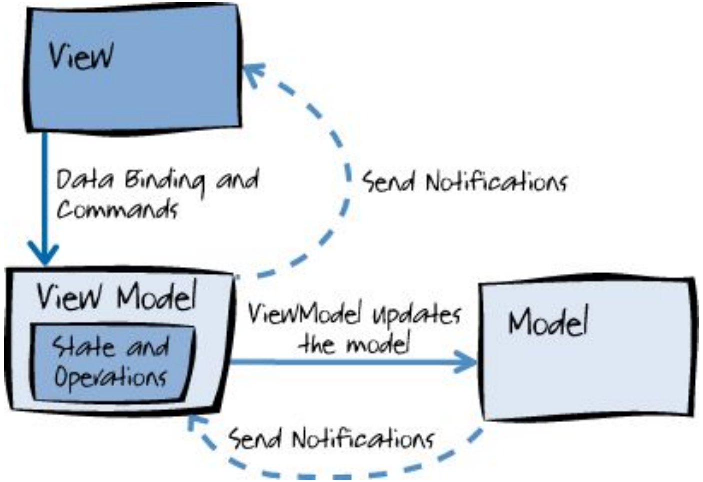


Property	Value
system	Linux
node	3a2677501e8b
release	5.4.104+
version	#1 SMP Sat Jun 5 09:50:34 PDT 2021
machine	x86_64
processor	x86_64

Rendered HTML page

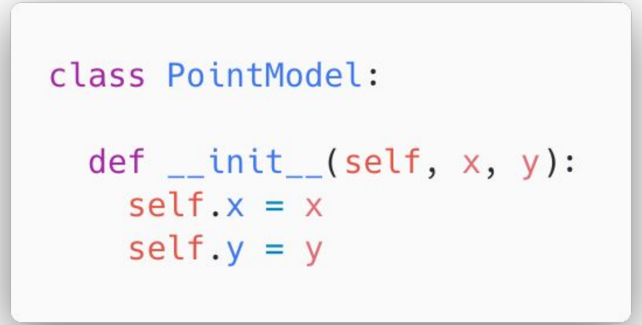
The MVVM Design Pattern

- Model View View-Model (MVVM)
- Powerful design pattern for web apps
  - View - the user interacts
  - Model - the data
  - View-Model - binding between the View and the application state



MVVM - the Model

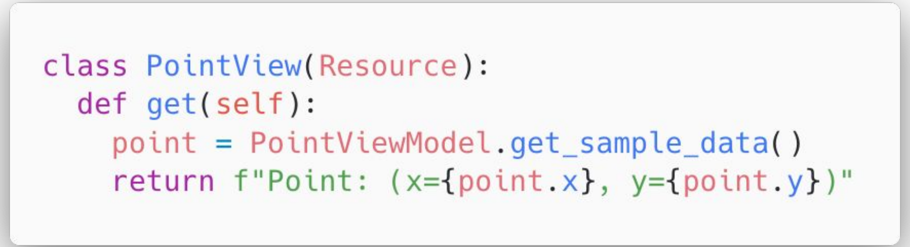
- The model stores your data
- Implements the access to the data (for example in a database)
- ORM (Object Relational Mapper)
- SQLAlchemy



```python
class PointModel:

  def __init__(self, x, y):
    self.x = x
    self.y = y
```

MVVM - the View

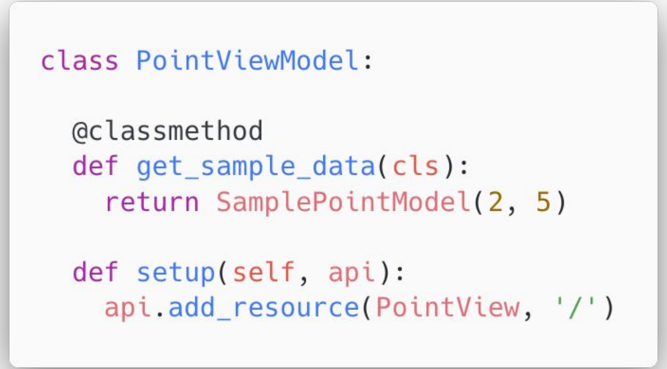
- Structure, layout, and appearance
- Implements the interaction with the user (for example input)
- Implements the rendering HTML page



```
class PointView(Resource):
  def get(self):
    point = PointViewModel.get_sample_data()
    return f"Point: (x={point.x}, y={point.y})"
```

MVVM - the View-Model

- Contains the state of the application
- Implements an automatic binding from the View to the state
- Handles the communication between the view and the state.



```python
class PointViewModel:

  @classmethod
  def get_sample_data(cls):
    return SamplePointModel(2, 5)

  def setup(self, api):
    api.add_resource(PointView, '/')
```

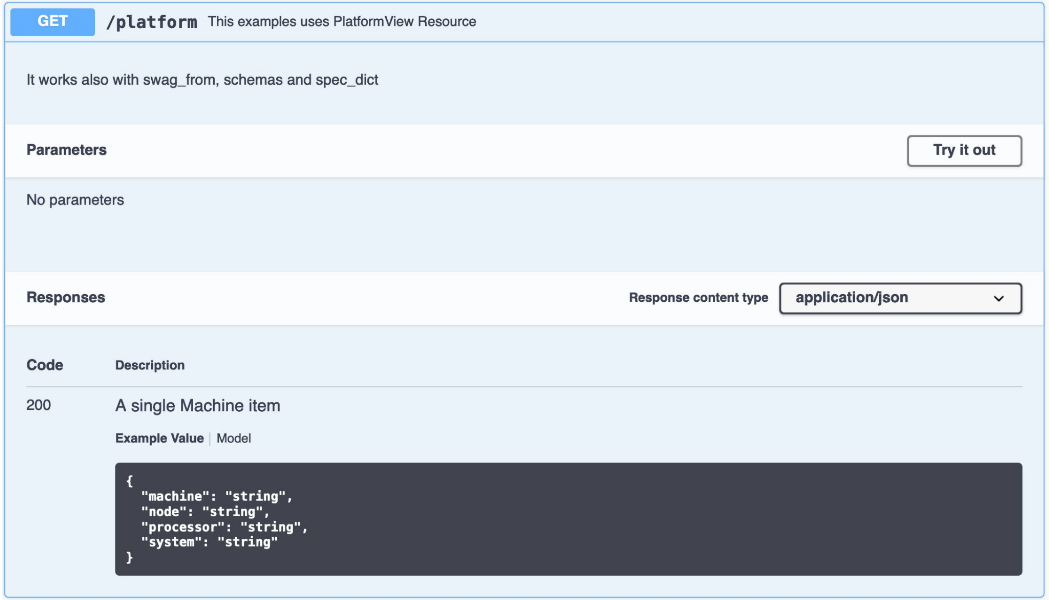
REST API

- Representational State Transfer (REST)
- Enable communication and interaction with other web services
- Rules and constraints for designing APIs
  - Uniform interface
  - Statelessness
  - Cacheability
  - Layered system
  - Code on demand

Example - ML classifier REST API

- `GET /info` - information about your model
  - `{“model”: “ResNet”, “parameters”: 1000000}`
- `GET /classes` - list of the supported classes
  - `[“dog”, “cat”, “car”]`
- `POST /classify` - classify an image
  - `{“dog”: 0.93, “cat”: 0.05, “car”: 0.02}`

REST API - documentation



/platform This examples uses PlatformView Resource

It works also with swag_from, schemas and spec_dict

Parameters

No parameters

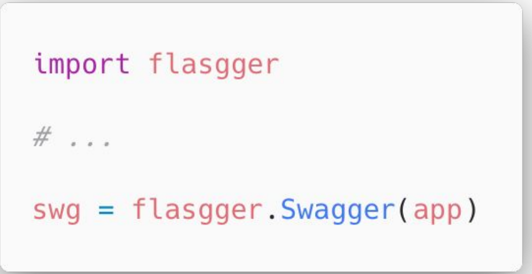
Responses

Response content type: application/json

Code	Description
200	A single Machine item

Example Value | Model

```
{
  "machine": "string",
  "node": "string",
  "processor": "string",
  "system": "string"
}
```



```python
import flasgger

# ...

swg = flasgger.Swagger(app)
```

Docs URL: http://xxxxxxxx.ngrok.io/apidocs/

Interactive UI - Vue.js

- Create interactive UI with Vue.js
- MVVM front end JavaScript framework
- Building UIs and single page web apps
- Alternatives
  - React
  - Angular

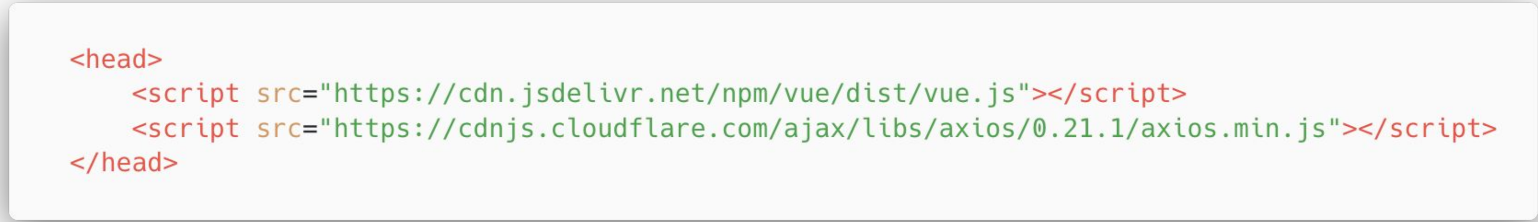
Vue.js - app overview

- Back end (Flask)
  - Serve REST API at `/platform`
  - Serve Vue.js template at /

- Front end (Vue.js)
  - Define HTML template of the webpage
  - Fetch and display data from `/platform` when initialized

Vue.js - include libraries

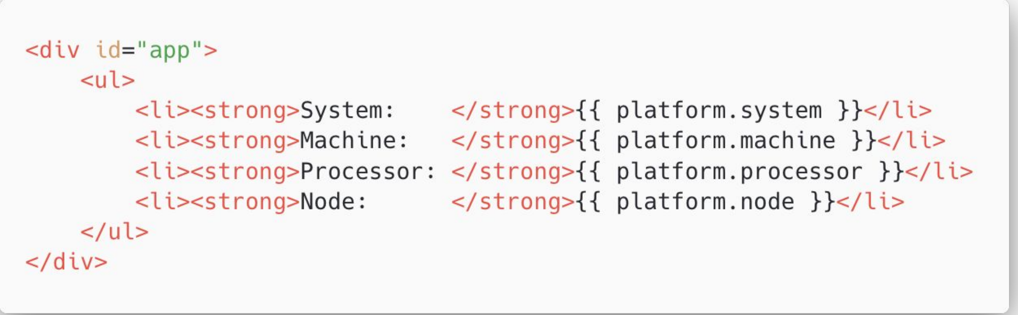
- We first need to include the vue.js and the axios.js libraries.
- We are going to use axios to fetch data from our API



```
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/axios/0.21.1/axios.min.js"></script>
</head>
```

Vue.js - define the template

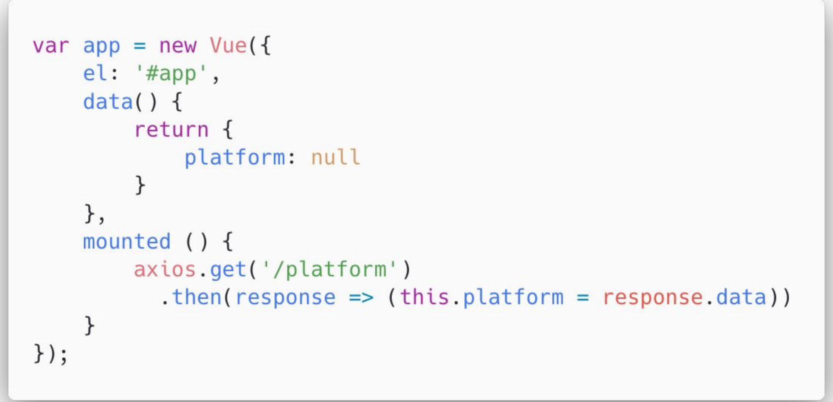
- We can define how the data is displayed
- We can display variables using `{{ value }}`



```
<div id="app">
    <ul>
        <li><strong>System:    </strong>{{ platform.system }}</li>
        <li><strong>Machine:   </strong>{{ platform.machine }}</li>
        <li><strong>Processor: </strong>{{ platform.processor }}</li>
        <li><strong>Node:      </strong>{{ platform.node }}</li>
    </ul>
</div>
```

Vue.js - define the template

- Initialize the Vue application
- Define the data function
- When initialized (`mounted()`), fetch data from the platform API (`/platform`) using axios



```
var app = new Vue({
    el: '#app',
    data() {
        return {
            platform: null
        }
    },
    mounted () {
        axios.get('/platform')
          .then(response => (this.platform = response.data))
    }
});
```

Vue.js - why not use Jinja?

- We did the same with Jinja, why do we need Vue.js?
  - Jinja templates are rendered in the back end
  - Vue templates are rendered dynamically
  - We will add dynamic functionality in the next section

Model Presentation

- Flask web application that classifies images
- Two entry points
  - `/` serve an interactive UI for uploading images
  - `/predict` classify the input image
- Use a pre-trained DenseNet model from torchvision

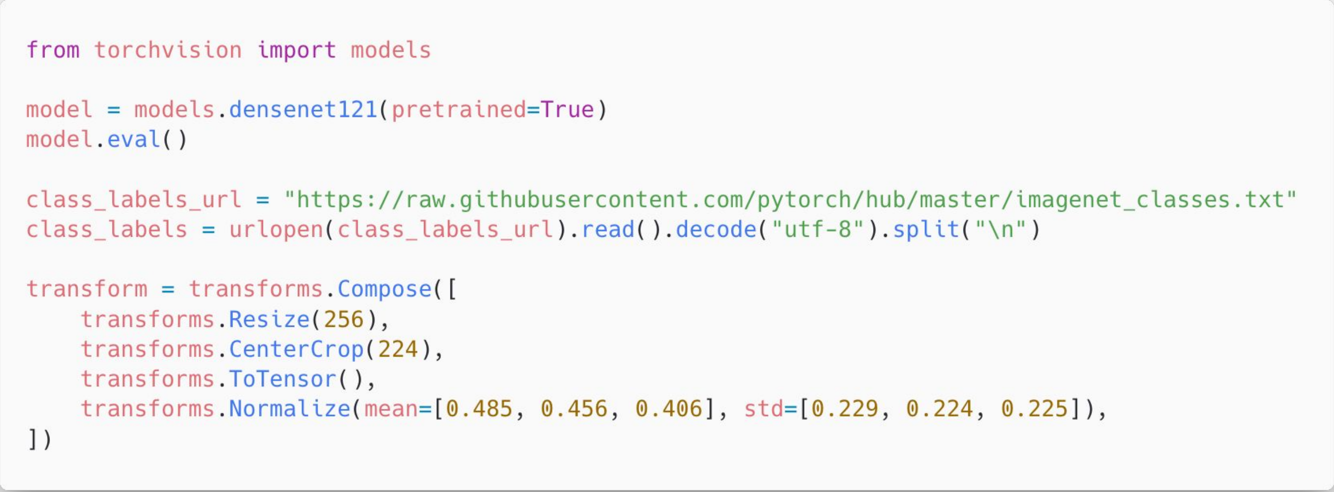
Torchvision

- Official PyTorch package
    - Datasets (ImageNet, MNIST, COCO, ...)
    - Models (AlexNet, ResNet, DenseNet, ...)
    - Image Transformations
    - Other computer vision operators and utilities for working with images

Alternative: TorchServe

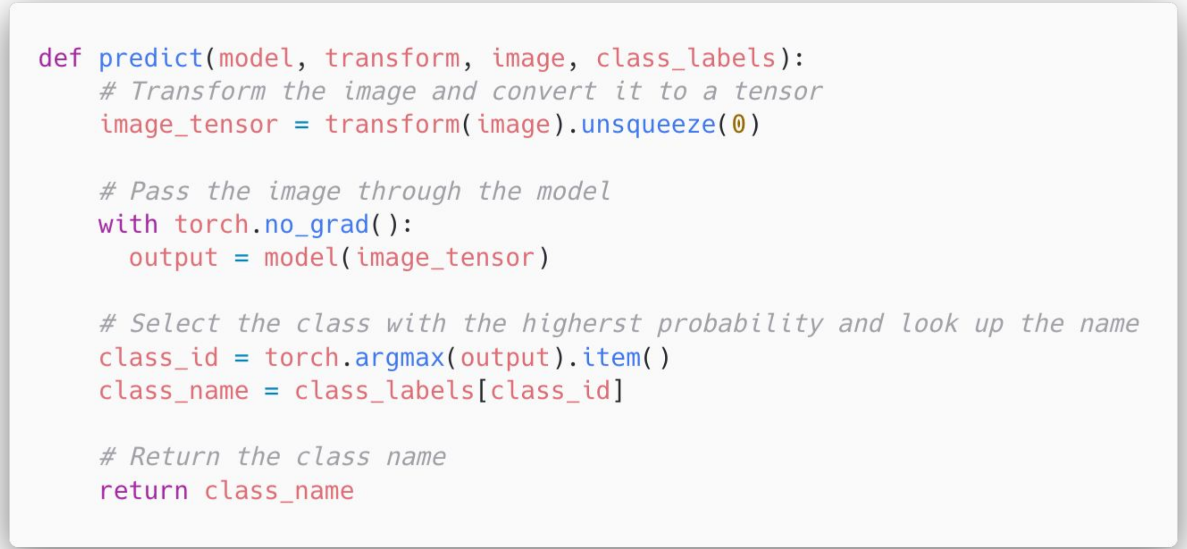
- Official PyTorch package to serve models as web servers
- REST API for classifying images
- Requires a separate deployment of the API

Loading a pre-trained model



```
from torchvision import models

model = models.densenet121(pretrained=True)
model.eval()

class_labels_url = "https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.txt"
class_labels = urlopen(class_labels_url).read().decode("utf-8").split("\n")

transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

Classifying an image



```
def predict(model, transform, image, class_labels):
    # Transform the image and convert it to a tensor
    image_tensor = transform(image).unsqueeze(0)

    # Pass the image through the model
    with torch.no_grad():
      output = model(image_tensor)

    # Select the class with the higherst probability and look up the name
    class_id = torch.argmax(output).item()
    class_name = class_labels[class_id]

    # Return the class name
    return class_name
```

Classifying an image

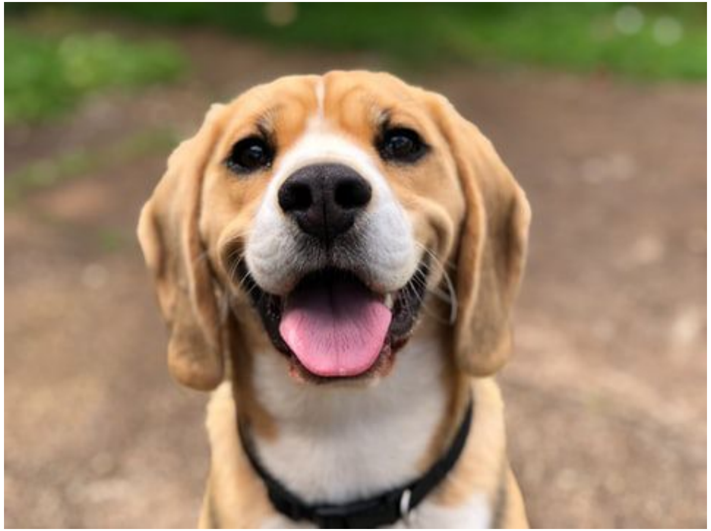
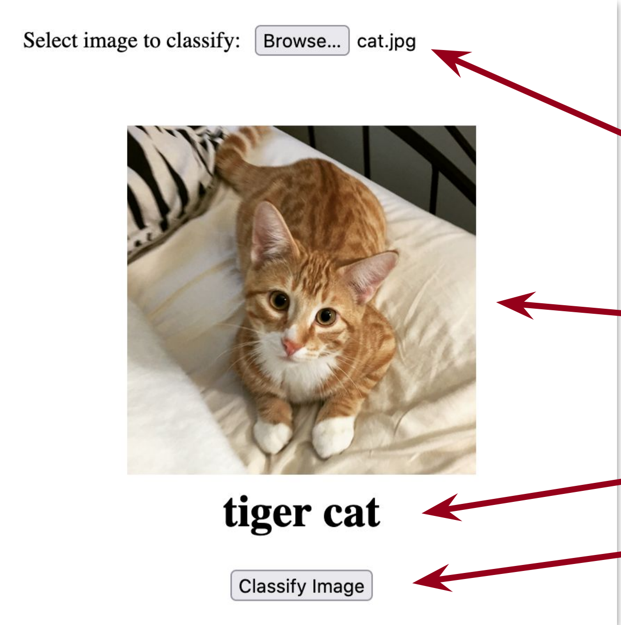


Photo by Marliese Streefland on Unsplash

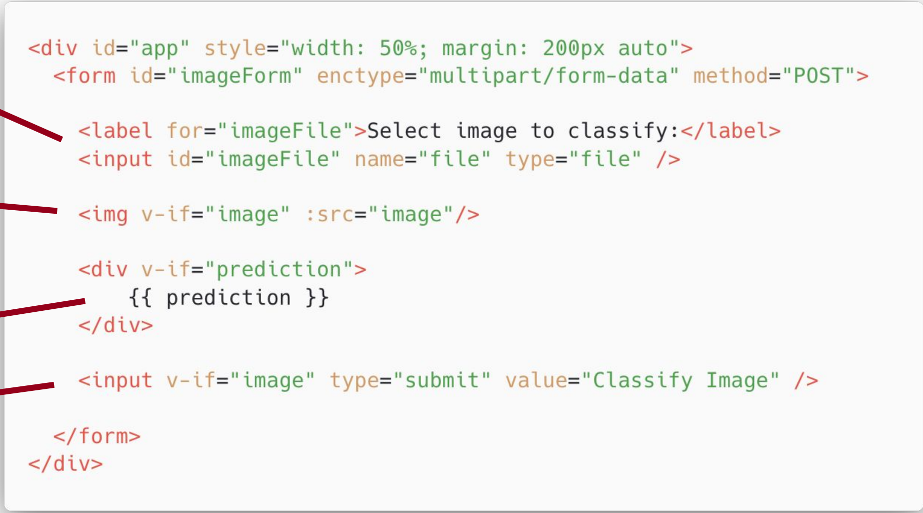
→ English foxhound

Creating the UI



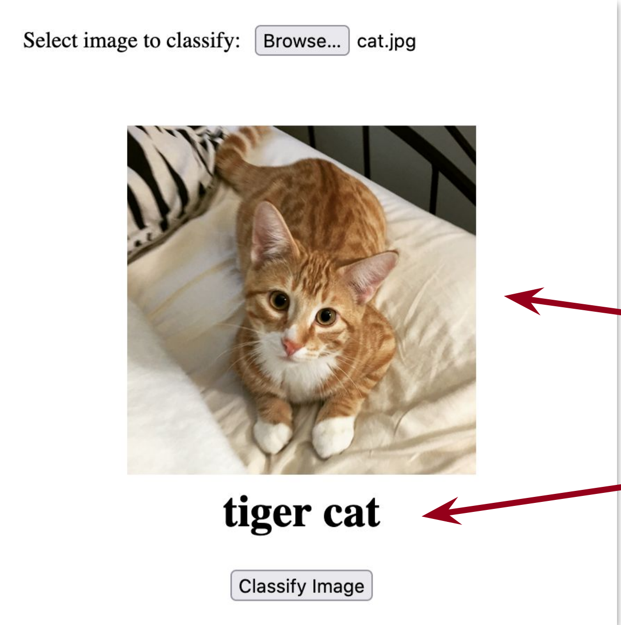
Select image to classify: Browse... cat.jpg

tiger cat



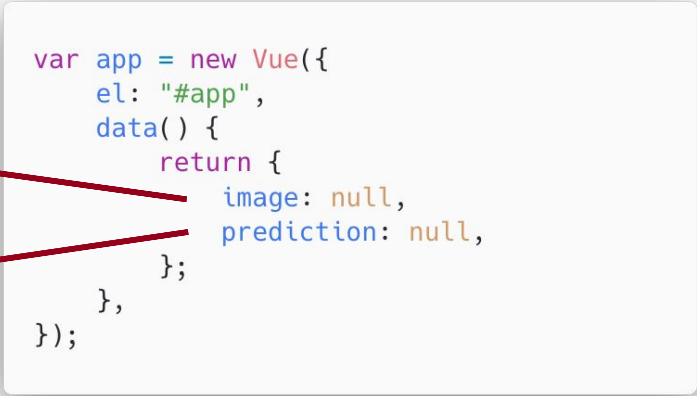
```
<div id="app" style="width: 50%; margin: 200px auto">
  <form id="imageForm" enctype="multipart/form-data" method="POST">

    <label for="imageFile">Select image to classify:</label>
    <input id="imageFile" name="file" type="file" />

    <img v-if="image" :src="image"/>

    <div v-if="prediction">
        {{ prediction }}
    </div>

    <input v-if="image" type="submit" value="Classify Image" />

  </form>
</div>
```

The Vue.js Application



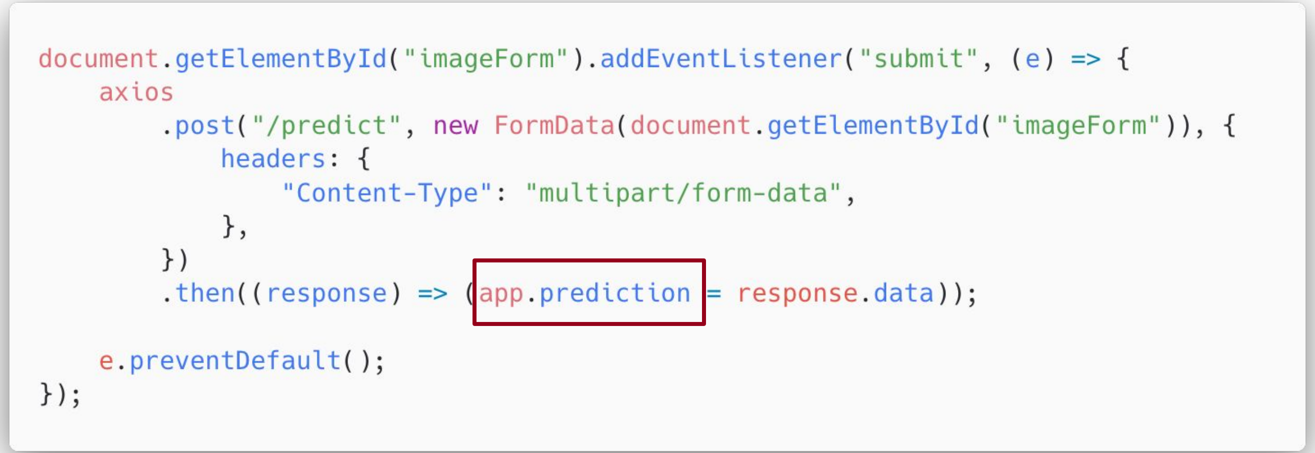
Select image to classify:

tiger cat



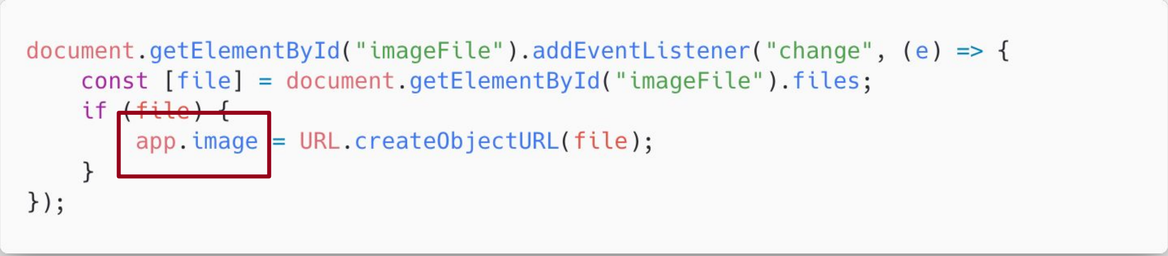
```
var app = new Vue({
    el: "#app",
    data() {
        return {
            image: null,
            prediction: null,
        };
    },
});
```

Using the Classification API



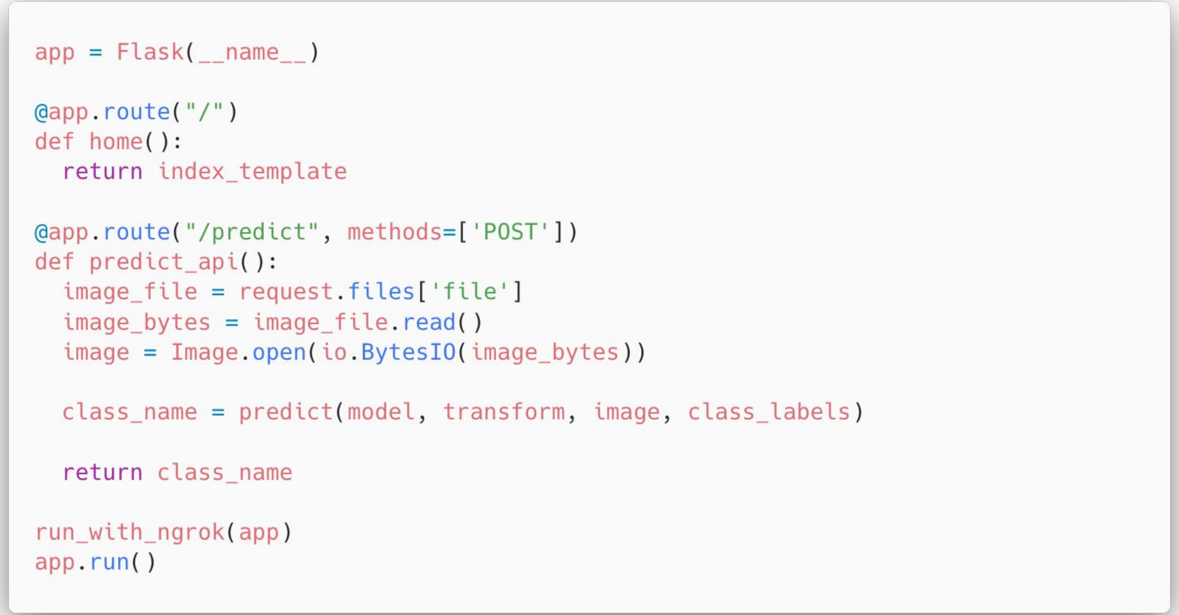
```
document.getElementById("imageForm").addEventListener("submit", (e) => {
    axios
        .post("/predict", new FormData(document.getElementById("imageForm")), {
            headers: {
                "Content-Type": "multipart/form-data",
            },
        })
        .then((response) => (app.prediction = response.data));

    e.preventDefault();
});
```

Using the Classification API



```
document.getElementById("imageFile").addEventListener("change", (e) => {
    const [file] = document.getElementById("imageFile").files;
    if (file) {
        app.image = URL.createObjectURL(file);
    }
});
```

Using the Classification API



```python
app = Flask(__name__)

@app.route("/")
def home():
  return index_template

@app.route("/predict", methods=['POST'])
def predict_api():
  image_file = request.files['file']
  image_bytes = image_file.read()
  image = Image.open(io.BytesIO(image_bytes))

  class_name = predict(model, transform, image, class_labels)

  return class_name

run_with_ngrok(app)
app.run()
```

Deploying on Heroku

- What is Heroku?
  - Public cloud provider
  - Platforms-as-a-Service
  - Pre-configured environments
  - Very easy deployment
  - Scalable and flexible
  - Free tier

Deploying on Heroku - Steps

1. Create a local application (outside of Colab)
2. Create a Heroku account
3. Install the Heroku CLI
4. Create a new Heroku application
5. Initialize a Git repository
6. Push to deploy

Prepare Python Environment

1. Create a Python environment

   ```
   python -m venv .venv
   ```

2. Activate the environment

   `source .venv/bin/activate`or `.venv\Scripts\activate.bat`

3. Install dependencies

   ```
   pip install flask Pillow gunicorn
   ```

4. Install PyTorch (without `torchaudio`)

   ```
   pip install torch torchvision
   ```

Prepare Python Environment

1. Create a Python environment

   ```
   python -m venv .venv
   ```

2. Activate the environment

   `source .venv/bin/activate`or `.venv\Scripts\activate.bat`

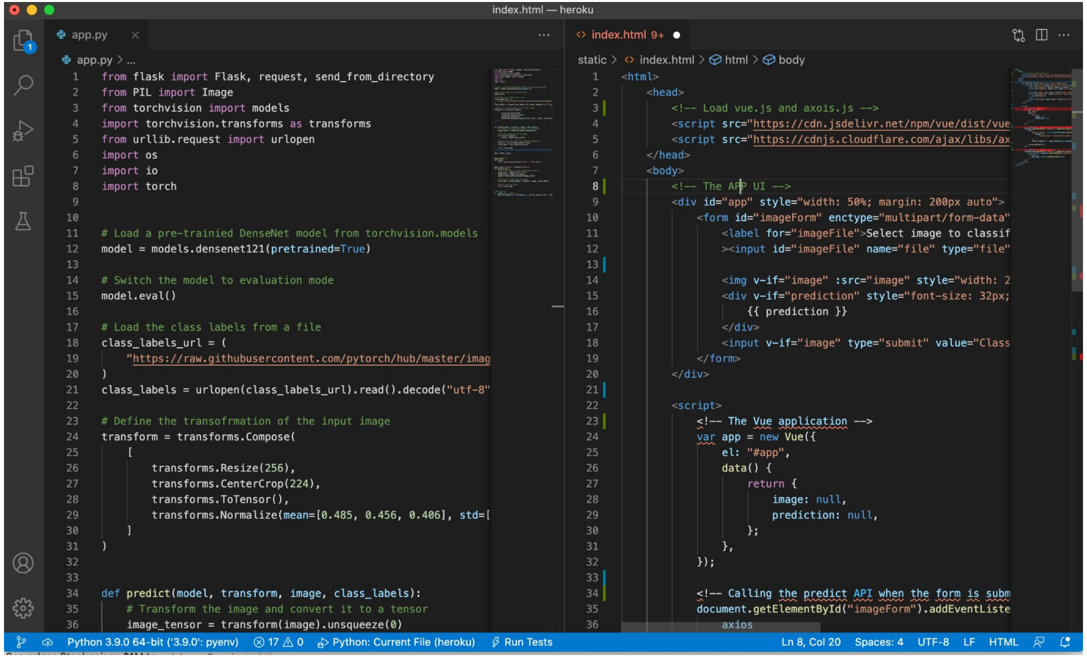
3. Install dependencies

   ```
   pip install flask Pillow gunicorn
   ```

4. Install PyTorch (without `torchaudio`)

   ```
   pip install torch torchvision
   ```

Create a Local Application


index.html — heroku
app.py
index.html 9+
Python 3.9.0 64-bit ('3.9.0': pyenv)   ⊗ 17 ⚠ 0   Python: Current File (heroku)   Run Tests   Ln 8, Col 20   Spaces: 4   UTF-8   LF   HTML

app.py > ...

```python
from flask import Flask, request, send_from_directory
from PIL import Image
from torchvision import models
import torchvision.transforms as transforms
from urllib.request import urlopen
import os
import io
import torch


# Load a pre-trained DenseNet model from torchvision.models
model = models.densenet121(pretrained=True)

# Switch the model to evaluation mode
model.eval()

# Load the class labels from a file
class_labels_url = (
    "https://raw.githubusercontent.com/pytorch/hub/master/imag
)
class_labels = urlopen(class_labels_url).read().decode("utf-8"

# Define the transofrmation of the input image
transform = transforms.Compose(
    [
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[
    ]
)


def predict(model, transform, image, class_labels):
    # Transform the image and convert it to a tensor
    image_tensor = transform(image).unsqueeze(0)
```

static > index.html > html > body

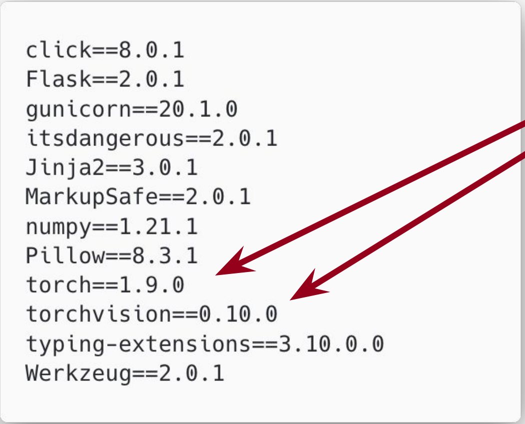
```html
<html>
    <head>
        <!-- Load vue.js and axois.js -->
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue
        <script src="https://cdnjs.cloudflare.com/ajax/libs/ax
    </head>
    <body>
        <!-- The APP UI -->
        <div id="app" style="width: 50%; margin: 200px auto">
            <form id="imageForm" enctype="multipart/form-data"
                <label for="imageFile">Select image to classif
                ><input id="imageFile" name="file" type="file"

                <img v-if="image" :src="image" style="width: 2
                <div v-if="prediction" style="font-size: 32px;
                    {{ prediction }}
                </div>
                <input v-if="image" type="submit" value="Class
            </form>
        </div>

        <script>
            <!-- The Vue application -->
            var app = new Vue({
                el: "#app",
                data() {
                    return {
                        image: null,
                        prediction: null,
                    };
                },
            });

            <!-- Calling the predict API when the form is subm
            document.getElementById("imageForm").addEventListe
                axios
```

Test it locally:

```
python app.py
```

Preparing for Heroku

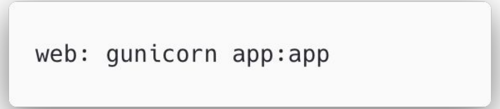
requirements.txt



```
click==8.0.1
Flask==2.0.1
gunicorn==20.1.0
itsdangerous==2.0.1
Jinja2==3.0.1
MarkupSafe==2.0.1
numpy==1.21.1
Pillow==8.3.1
torch==1.9.0
torchvision==0.10.0
typing-extensions==3.10.0.0
Werkzeug==2.0.1
```

Too large because they include both GPU and CPU code

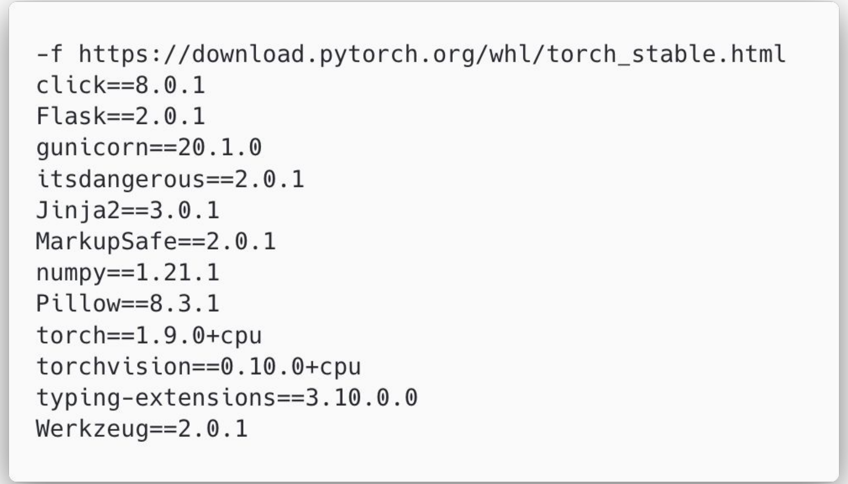
Procfile



```
web: gunicorn app:app
```

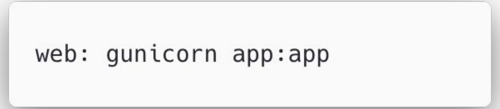
Preparing for Heroku

requirements.txt



```
-f https://download.pytorch.org/whl/torch_stable.html
click==8.0.1
Flask==2.0.1
gunicorn==20.1.0
itsdangerous==2.0.1
Jinja2==3.0.1
MarkupSafe==2.0.1
numpy==1.21.1
Pillow==8.3.1
torch==1.9.0+cpu
torchvision==0.10.0+cpu
typing-extensions==3.10.0.0
Werkzeug==2.0.1
```

Procfile



```
web: gunicorn app:app
```

Deploy Your App

1. Create a Heroku account
2. Install the Heroku CLI
3. Login to Heroku

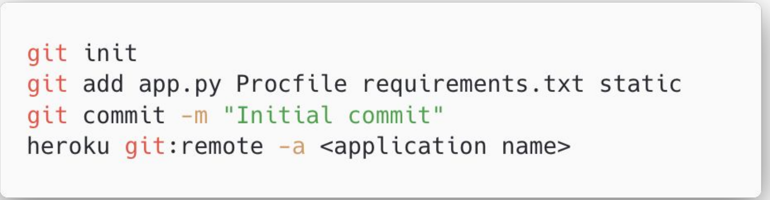
   ```
   heroku login
   ```

4. Create a new Heroku application

   ```
   heroku create <application name>
   ```

5. Initialize a git repository
6. Push to deploy

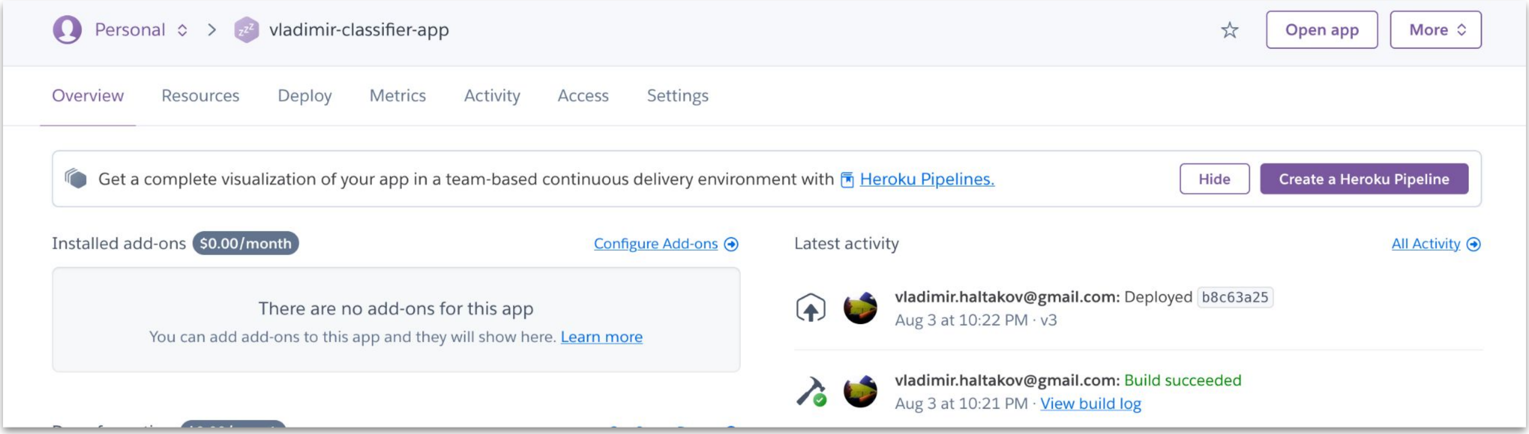
   ```
   git push heroku master
   ```



```
git init
git add app.py Procfile requirements.txt static
git commit -m "Initial commit"
heroku git:remote -a <application name>
```

Heroku Dashboard

https://dashboard.heroku.com/apps



Get a complete visualization of your app in a team-based continuous delivery environment with Heroku Pipelines.

Installed add-ons $0.00/month

There are no add-ons for this app

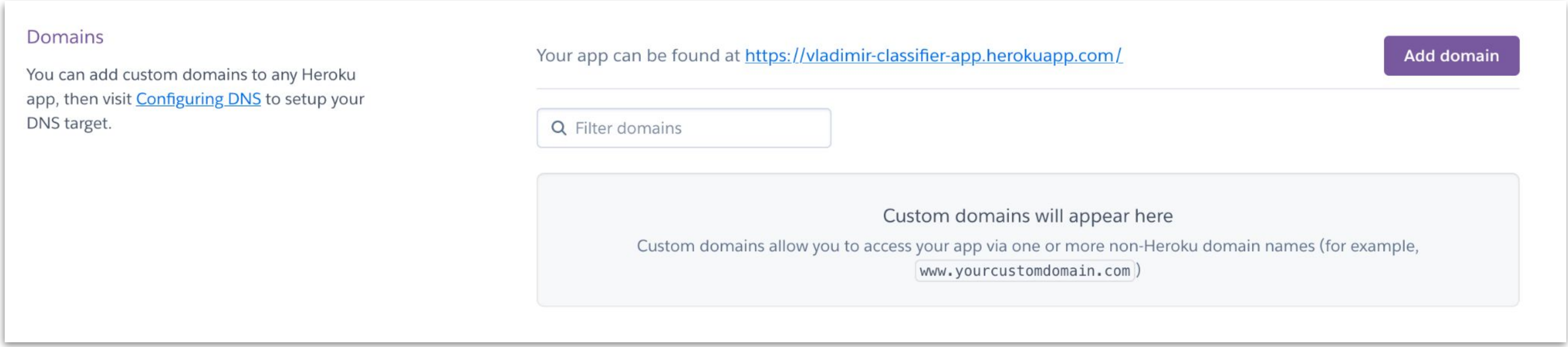
You can add add-ons to this app and they will show here. Learn more

Latest activity

vladimir.haltakov@gmail.com: Deployed b8c63a25
Aug 3 at 10:22 PM · v3

vladimir.haltakov@gmail.com: Build succeeded
Aug 3 at 10:21 PM · View build log

Change the Domain Name?



Domains

You can add custom domains to any Heroku app, then visit Configuring DNS to setup your DNS target.

Your app can be found at https://vladimir-classifier-app.herokuapp.com/

Custom domains will appear here

Custom domains allow you to access your app via one or more non-Heroku domain names (for example, `www.yourcustomdomain.com`)

Summary

- Create Flask applications
- Apply the MVVM design pattern
- Create REST APIs
- Interactive UI with Vue.js
- Integrate a PyTorch model into Flask
- Deploy on Heroku