

Computer Vision Models in Production

- Who am I?
- Development Environment
- Production Environment
- Deployment Approaches

Who am I?

Computer science student.

Data engineer at MoonVision.



Computer Vision for automotive and food service industry.

Development Environment



jupyter

Select items to perform actions on them.

Name	Last Modified	File size
eval	3 minutes ago	
explore	3 minutes ago	
export	a minute ago	
import	3 minutes ago	
legacy	3 minutes ago	
SiamFC_Kartine	3 minutes ago	
tests	3 minutes ago	
train	3 minutes ago	
AutogradCheckKernelCutLoss.ipynb	3 minutes ago	2.32 kB
Create-embeddings-Siamese-RGB.ipynb	3 minutes ago	64.1 kB
CRFLossTest.ipynb	3 minutes ago	21.1 kB



- Python
- Conda
- PyTorch
- Cuda

Production Environment

Infrastructure as a Service



-
- Free choice of deployment environment
 - Existing systems
 - Scalability

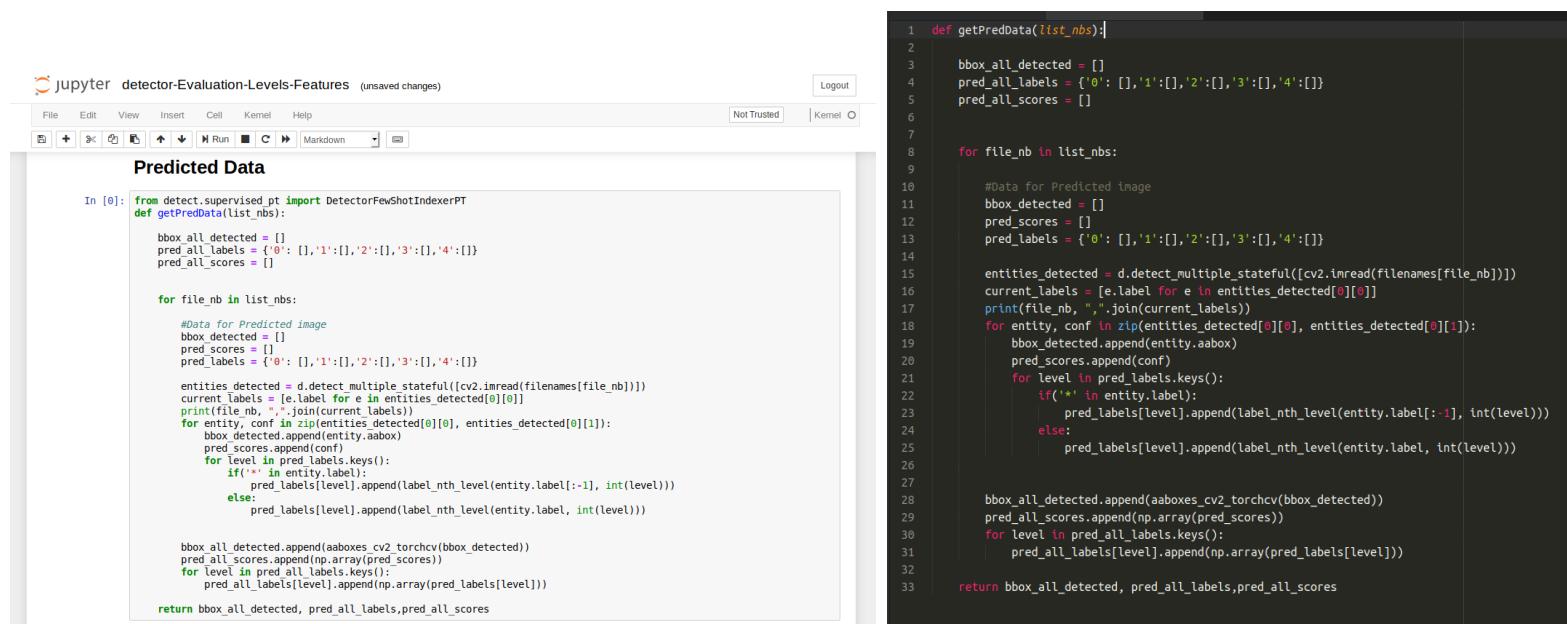
Embedded Systems



-
- Often less powerful
 - Different processor architectures

Deployment Approaches

Replicate Development Environment in Production



The image shows a Jupyter Notebook interface with two panes. The left pane displays the code for a function named `getPredData`. The right pane shows the execution results of the code, which includes variable definitions and a large block of generated Python code.

```
In [0]: from detect.supervised_pt import DetectorFewShotIndexerPT
def getPredData(list_nb):
    bbox_all_detected = []
    pred_all_labels = {'0': [], '1': [], '2': [], '3': [], '4': []}
    pred_all_scores = []

    for file_nb in list_nb:
        #Data for Predicted image
        bbox_detected = []
        pred_scores = []
        pred_labels = {'0': [], '1': [], '2': [], '3': [], '4': []}

        entities_detected = d.detect_multiple_stateful([cv2.imread(filenames[file_nb])])
        current_labels = {e.label for e in entities_detected[0][0]}
        print(file_nb, ":", join(current_labels))
        for entity, conf in zip(entities_detected[0][0], entities_detected[0][1]):
            bbox_detected.append(entity.abbox)
            pred_scores.append(conf)
            for level in pred_labels.keys():
                if('*' in entity.label):
                    pred_labels[level].append(label_nth_level(entity.label[-1], int(level)))
                else:
                    pred_labels[level].append(label_nth_level(entity.label, int(level)))

        bbox_all_detected.append(aaboxes_cv2_torchcv(bbox_detected))
        pred_all_scores.append(np.array(pred_scores))
        for level in pred_all_labels.keys():
            pred_all_labels[level].append(np.array(pred_labels[level]))

    return bbox_all_detected, pred_all_labels,pred_all_scores
```

```
1 def getPredData(list_nb):
2
3     bbox_all_detected = []
4     pred_all_labels = {'0': [], '1': [], '2': [], '3': [], '4': []}
5     pred_all_scores = []
6
7
8     for file_nb in list_nb:
9
10         #Data for Predicted image
11         bbox_detected = []
12         pred_scores = []
13         pred_labels = {'0': [], '1': [], '2': [], '3': [], '4': []}
14
15         entities_detected = d.detect_multiple_stateful([cv2.imread(filenames[file_nb])])
16         current_labels = {e.label for e in entities_detected[0][0]}
17         print(file_nb, ":", join(current_labels))
18         for entity, conf in zip(entities_detected[0][0], entities_detected[0][1]):
19             bbox_detected.append(entity.abbox)
20             pred_scores.append(conf)
21             for level in pred_labels.keys():
22                 if('*' in entity.label):
23                     pred_labels[level].append(label_nth_level(entity.label[-1], int(level)))
24                 else:
25                     pred_labels[level].append(label_nth_level(entity.label, int(level)))
26
27
28         bbox_all_detected.append(aaboxes_cv2_torchcv(bbox_detected))
29         pred_all_scores.append(np.array(pred_scores))
30         for level in pred_all_labels.keys():
31             pred_all_labels[level].append(np.array(pred_labels[level]))
32
33     return bbox_all_detected, pred_all_labels,pred_all_scores
```

- + Sometimes less complicated
- Slower
- Limited amount of target platforms

ONNX

- Open Neural Network Exchange format
- Defines a computation graph model as well as a set of built-in operators and data types.

Supported by a Wide Range of Vendors

Frameworks



Converters



Runtimes



Compilers



Visualizers



Trace PyTorch model

```
# model = ...
torch.onnx.export(
    model,
    args=torch.randn(10, 3, 224, 224),
    f='ptmodel.onnx')
```

Trace Tensorflow model

```
onnx_graph = tf2onnx.tfonnx.process_tf_graph(
    sess.graph,
    input_names=["input:0"],
    output_names=["output:0"])

model_proto = onnx_graph.make_model("test")
with open("tfmodel.onnx", "wb") as f:
    f.write(model_proto.SerializeToString())
```

Inference with Caffe2

```
model = onnx.load("ptmodel.onnx")
prepared_backend = onnx_caffe2.backend.prepare(model)
prepared_backend(data)
```

Inference with ONNX Runtime

```
sess = rt.InferenceSession("rf_iris.onnx")
input_name = sess.get_inputs()[0].name
label_name = sess.get_outputs()[0].name
pred_onx = sess.run(
    [label_name],
    {input_name: data})[0]
```

ONNX Opsets

- Versions the specified operations
- Version number used for addition of new operations, breaking changes and deprecation alike
- Frameworks decide which Opset versions they support
- Can be difficult to find library versions that work together

TorchScript

Tracing

```
def a_function(a, b):
    return a + b

traced_function = torch.jit.trace(
    a_function,
    (torch.tensor(0), torch.tensor(0)))
```

Can also be used to trace entire modules:

```
torch.jit.trace(
    torchvision.models.resnet50().eval(),
    example_inputs=torch.randn(10,3,224,224))
```

Tracing conditions

```
def conditional(x):
    if x > 10:
        x += 10
    return x

trace_a = traced_function = torch.jit.trace(
    conditional, torch.tensor(0))

# returns 5
trace_a(torch.tensor(5))

trace_b = traced_function = torch.jit.trace(
    conditional, torch.tensor(100))

# returns 15
trace_b(torch.tensor(5))
```

Scripting

Subset of Python:

- Statically typed functions
- All return values must be tensors
- Function with `@script` decorator
- `ScriptModule`

Script Function

```
@torch.jit.script
def a_script_function(a, b):
    return a + b
```

Script Module

```
class AScriptModule(torch.jit.ScriptModule):
    def __init__(self):
        pass

    @torch.jit.script_method
    def forward(self, input):
        return input
```

If you want to learn more

ONNX: the long and collaborative road to machine learning portability,

<https://medium.com/moonvision/a6416a96e870>