Microsoft

mPyPI: A Monadic Way to do Complex Data Processing in Python

Dmitry Soshnikov

Cloud Developer Advocate / Software Engineer, Microsoft

@shwars – http://soshnikov.com

Microsoft

mPyPI: How to Program in Python and Survive

Dmitry Soshnikov

Cloud Developer Advocate / Software Engineer, Microsoft

@shwars – http://soshnikov.com

whoami

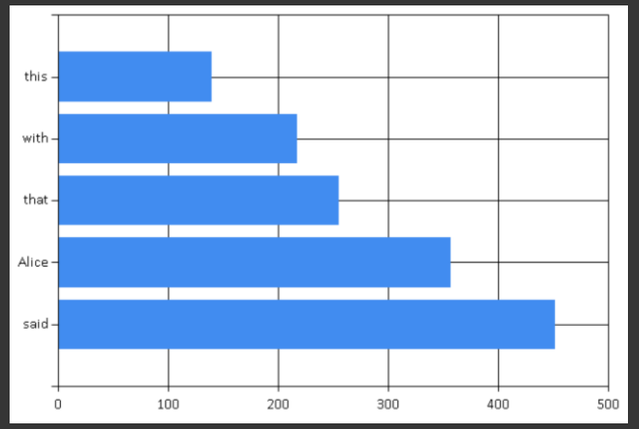
- Software Developer
- Associate Professor teaching Functional Programming & AI
- Technical Evangelist / Cloud Advocate
- AI/ML Developer / Data Scientist
- Interested in Teaching Kids to Program / Science Art / TechnoMagic

Why I (Used to) Hate Python

Motivation

The way I used to develop in .NET / F#

```
File.ReadAllLines @"d:\data\alice.txt"
|> Seq.map (fun x->x.Split())
|> Seq.concat
|> Seq.filter (fun x->x.Length>3)
|> Seq.groupBy (fun x->x)
|> Seq.map (fun (w,s) -> (w,Seq.length s))
|> Seq.sortBy (fun (w,n) -> -n)
|> Seq.take(5)
|> Chart.Bar
```



The way I had to do in Python

```
txt = open('alice.txt','r').readlines()
words = sum(map(lambda x:
          x.lower().strip().split(),txt),[])
filtered = filter(lambda x:len(x)>3, words)
pairs = [ (k,len(list(g)))
      for k,g in groupby(sorted(filtered))]
res = sorted(pairs,key=lambda x:-x[1])[0:5]
plt.bar(list(range(5)),[x[1] for x in res])
plt.xticks(list(range(5)),
          labels=[x[0] for x in res])
```

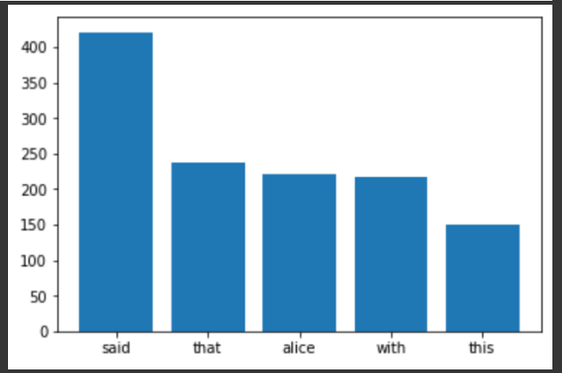


Photo by @tompumford

Learn to Love Python
by Developing Your Own Library

Demo Alert

https://notebooks.azure.com
/shwars/projects/mpypldemo

Azure Notebooks

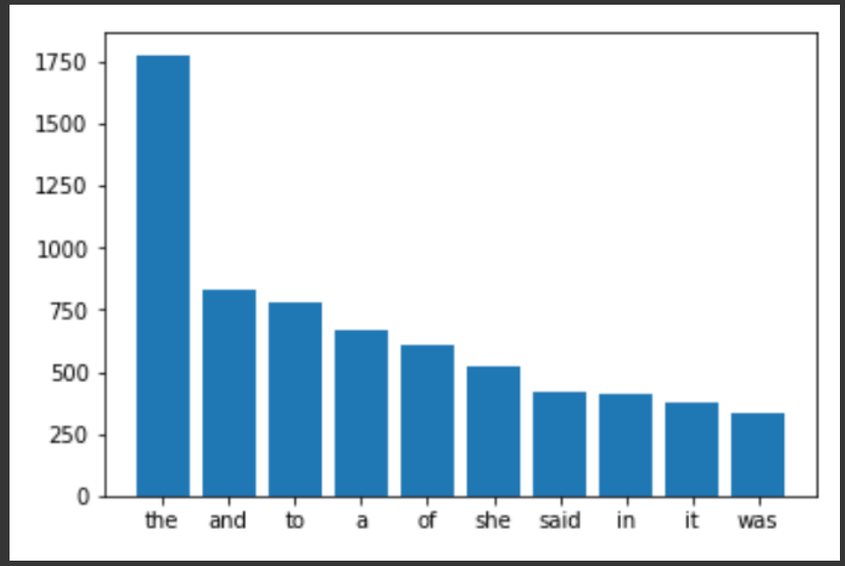
http://aka.ms/aznb

Step 1: Using Pipe

```
( open('alice.txt','r').readlines()
| select(lambda x:
           x.lower().strip().split())
| chain
| groupby(lambda x:x)
| select(lambda x: (x[0],len(list(x[1]))))
| sort(key=lambda x:-x[1])
| take(10)
| bar)
```

```
@Pipe
def bar(seq):
    n,l = zip(*enumerate(list(seq)))
    plt.bar(n,[x[1] for x in l])
    plt.xticks(n,labels=[x[0] for x in l])
    plt.show()
```

- Different options for using pipe operators in Python: https://stackoverflow.com/questions/28252585/functional-pipes-in-python-like-from-rs-magritrr
- Best imho: Pipe by Julian Palard



Problem with Pipeline Operators

- Task: Date-stamp a series of pictures
- Problem: Only one value can float through pipeline
- Solution: Use tuples

```
(mp.get_files('images')
 | take(5)
 | select(lambda x: (x,time.ctime(os.stat(x).st_ctime)[10:19]))
 | select(lambda x: (im_load(x[0],size=(None,200)),x[1]))
 | select(lambda x: imprint(*x))
 | show)
```

[‘Image_1.jpg’, ‘image_2.jpg’]

[(‘Image_1.jpg’,’19:20’), (‘image_2.jpg’,’1930’)]

[(array(..),’19:20’), (array(...),’1930’)]

[array(..),array(...)]

Introducing mPyPl

- Monadic Pipeline Library
- Main idea: use named dicts that flow through pipes
- Special dicts (**mdict**) that support laziness

```
# pip install mPyPl
```

```
import mPyPl as mp
import mPyPl.utils.image as mpui
```

http://shwars.github.io/mPyPl

mPyPl

mPyPl is a library that simplifies all data-processing tasks in Python, by introducing a functional pipeline concept.

View on GitHub | Tutorial | Documentation | PyPi Page

Solution with mPyPl

```
(mp.get_files('images')
 | mp.as_field('fname')
 | mp.apply('fname','time',lambda x: time.ctime(os.stat(x).st_ctime)[10:19])
 | mp.apply('fname','image',lambda x: im_load(x,size=(None,200)))
 | mp.apply(['image','time'],'res',lambda x: imprint(*x))
 | mp.select_field('res')
 | mp.pexec(show_images))
```

['Image_1.jpg', 'image_2.jpg']

[{ 'fname':'Image_1.jpg' }, { 'fname': 'image_2.jpg'}]

[{ 'fname':'Image_1.jpg',
'time': '19:20',
'image': array(),
'res': array() }, ...]

- Main function – apply
  - Takes names of source – dest fields and lambda
  - Need several fields – specify list
- Everything is a generator -> lazy!

- Other functions
  - as_field -> normal stream to named
  - select_field -> names stream to single-valued
  - pexec(f) executes f on the pipeline
  - get_files is a useful util to list all files in a directory

Example with Frequency Dictionary

```
(open('alice.txt','r').readlines()
 | mp.as_field('line')
 | mp.apply('line','word’,
       lambda x: x.lower().strip().split())
 | mp.unroll('word')
 | mp.select_fields(['word'])
 | group_by('word','group')
 | mp.apply('group','count',len)
 | mp.delfield('group')
 | sort(key=lambda x:-x['count'])
 | mp.take(10)
 | mp.select_field(['word','count'])
 | bar)
```

```
@Pipe
def group_by(seq,fld,fld2):
    k = lambda x:x[fld]
    l = sorted(list(seq),key=k)
    for x,xs in itertools.groupby(l,key=k):
        yield mp.mdict({fld:x,fld2:list(xs)})
```

Everything which has beginning has an end…

Sources

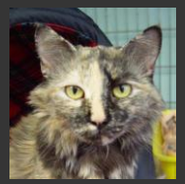
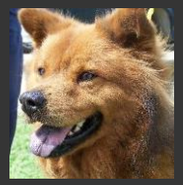
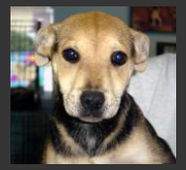
- get_files
- get_datastream [multiclass datasteam]
  - Special source for classification tasks
  - Handles classes as directory names, auto-numbering, train/test split, etc.
- json stream
- xml stream
- csv
- video source (frames)
- range(…)

Sinks

- as_list, as_npy, as_dict
- csv, json, …
- as_batch
- pexec(…)
- collect_video
- imageset (in progress)

Classification in mPyPl

root

- cats
  -  cat01.jpg
  -  cat02.jpg
- dogs
  -  dog01.jpg
  -  dog02.jpg

```
mp.get_datastream('root')
| mp.datasplit
```

```
[ { 'filename' : 'cat01.jpg',
    'class_id' : 0,
    'class_name' : 'cats',
    'split': mp.SplitType.Train },
  { 'filename' : 'dog01.jpg',
    'class_id' : 1,
    'class_name' : 'dogs',
    'split': mp.SplitType.Test },
... ]
```

Classification In mPyPl

```
train, test = (
 mp.get_datastream('d:/temp/imclass')
 | mp.datasplit
 | mp.stratify_sample_tt
 | mp.apply('filename','img',functools.partial(im_load,size=150)))
 | mp.apply('img', 'input', lambda x: x/255.)
 | mp.summary
 | mp.inspect
 | mp.make_train_test_split)
```

```
model.fit(
  train | mp.infshuffle | mp.as_batch('input','class_id', 128),
validation_data =
  test | mp.infshuffle | mp.as_batch('input','class_id', 128))
```

Laziness is the key!

Pipelines are lazy by design

Fields support the following evaluation strategies:

- Value – exact value of the field is stored
- LazyMemoized – value is computed on demand and stored
- OnDemand – value is computed each time the field is accessed

Laziness is super-important because it reduces memory footprint with big data

Classification w/Augmentation

```
train, test = (
 mp.get_datastream('d:/temp/imclass')
 | mp.datasplit
 | mp.stratify_sample_tt
 | mp.apply('filename','img',functools.partial(im_load,size=(150,150)))
 | mp.apply('img','aug',augment, eval_strategy = OnDemand)
 | mp.apply('aug', 'input', lambda x: x/255., eval_strategy = OnDemand)
 | mp.summary
 | mp.inspect
 | mp.make_train_test_split)
```

Fun Example: People Blending

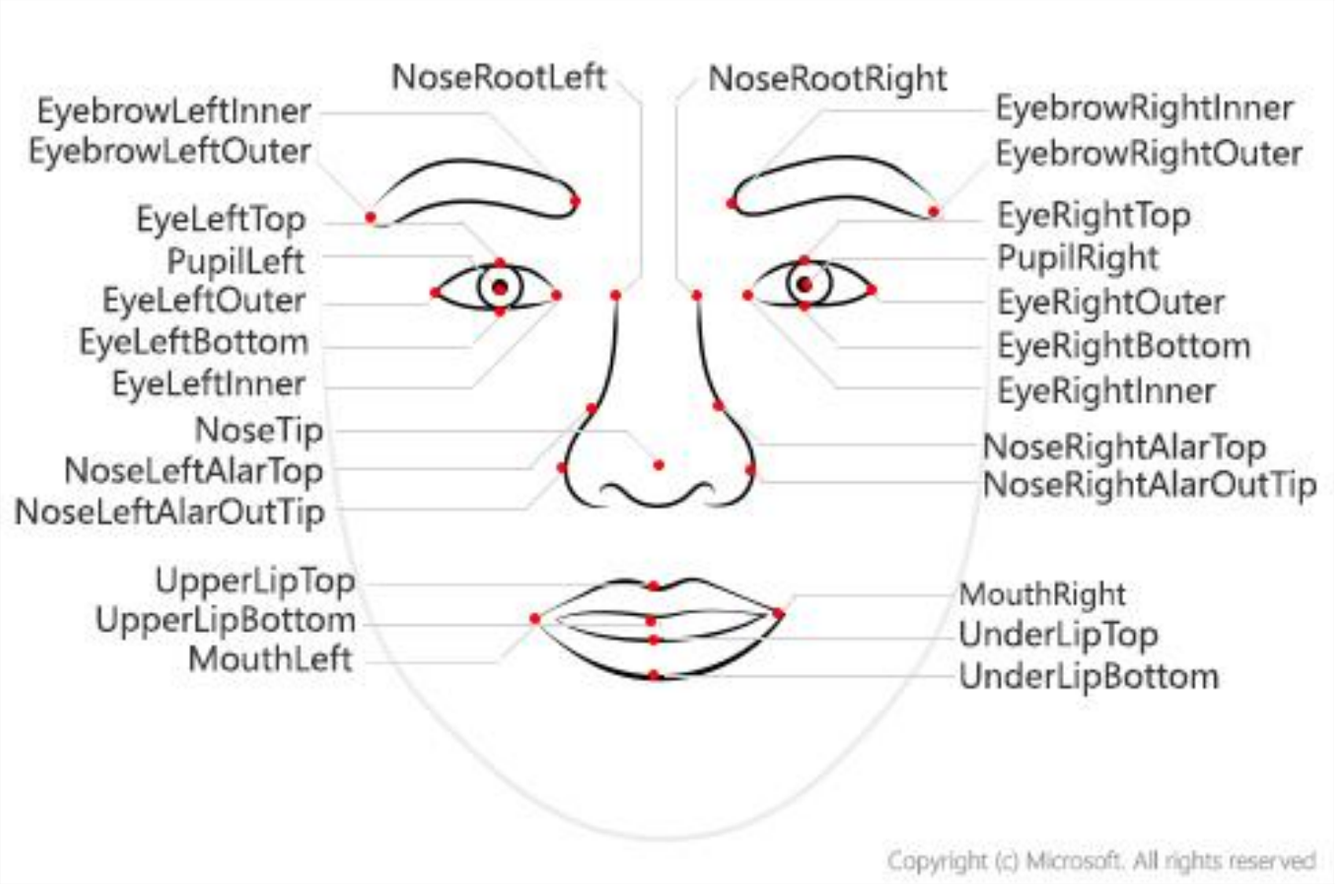
Demo Alert

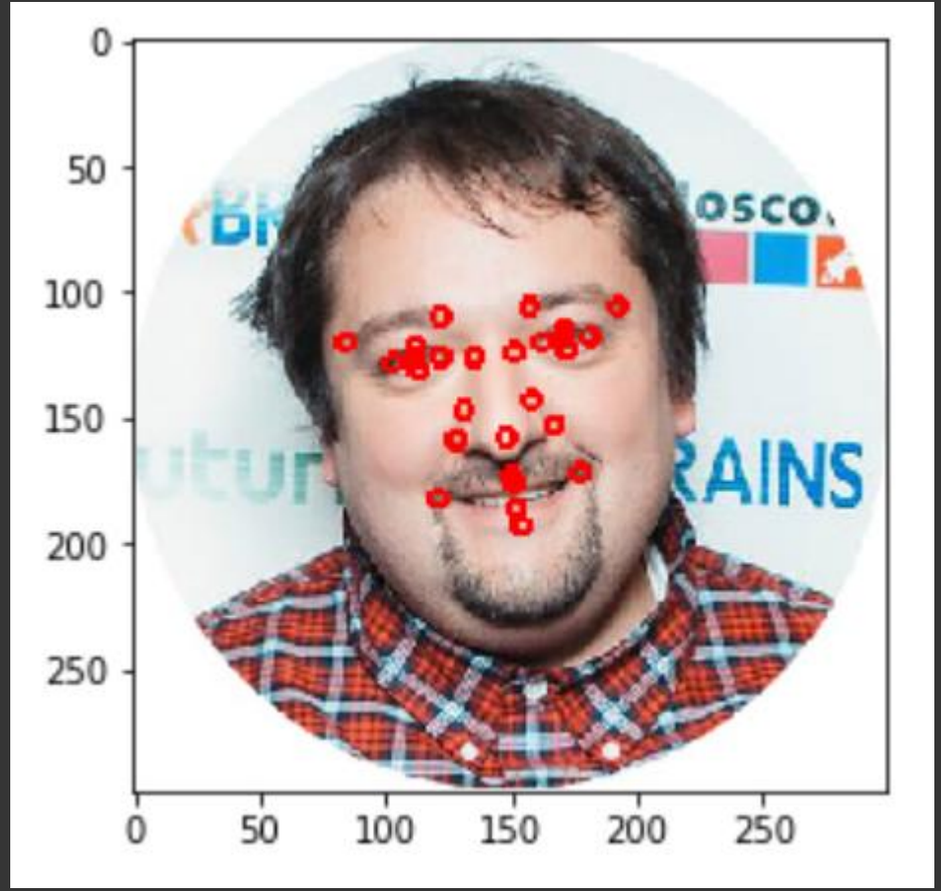
https://github.com/CloudAdvocacy/CreepyFaces

https://notebooks.azure.com/sosh/projects/CreepyFaces

CreepyFaces-Tutorial-mPyPl.ipynb

Main Idea: Use Face API to get face points





How to Use Face API - https://aka.ms/FaceApiDocs

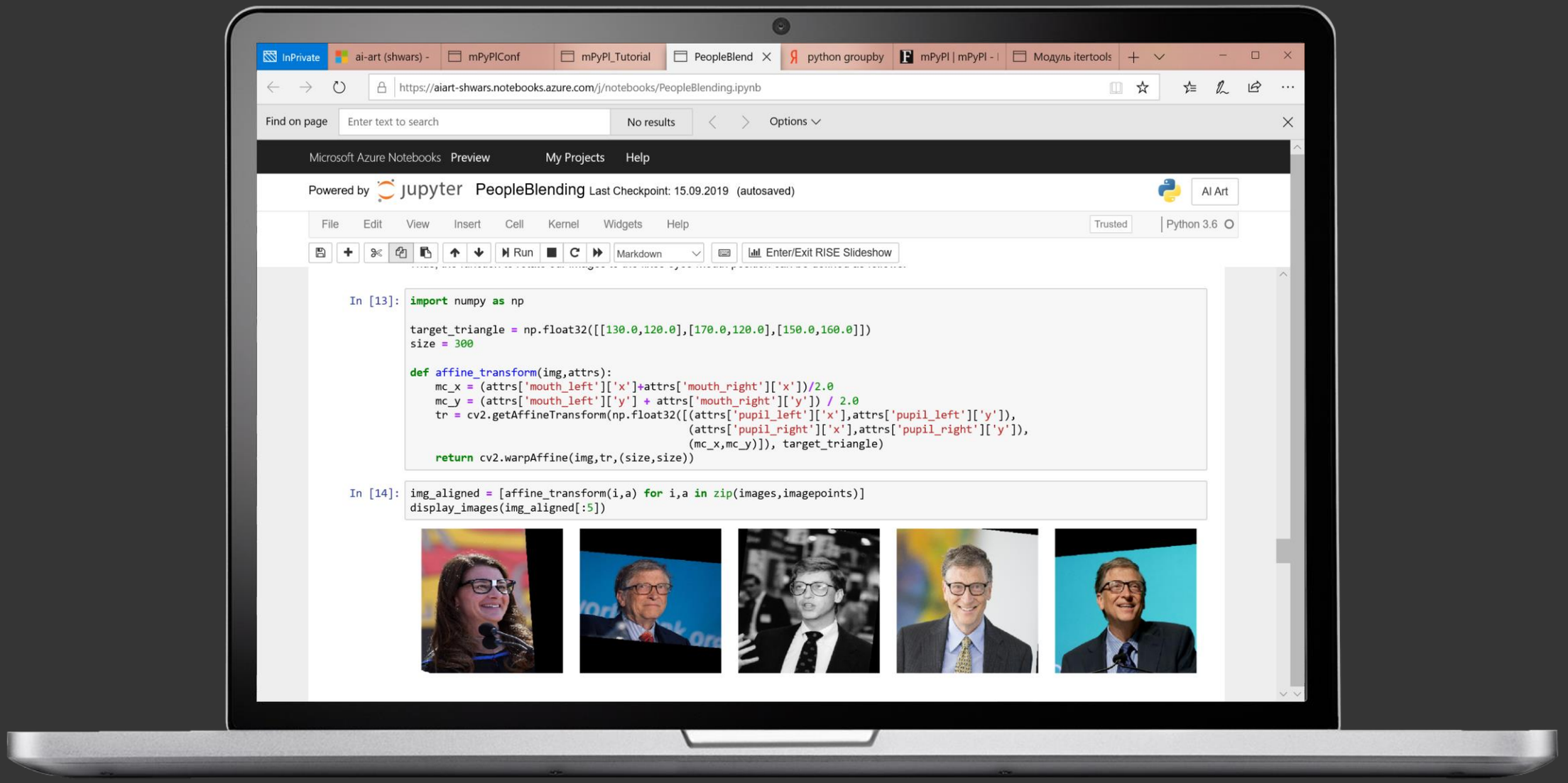
Request Key + EndPoint URL: https://aka.ms/FaceKeyRequest

```
# pip install azure-cognitiveservices-vision-face
```

```
import azure.cognitiveservices.vision.face as cf
from msrest.authentication import CognitiveServicesCredentials
cli = cf.FaceClient(endpoint,CognitiveServicesCredentials(key))
res = cli.face.detect_with_url(...)
```


```
…{'pupil_left': {'x': 668.7, 'y': 672.3}, 'pupil_right': {'x': 832.8, 'y': 682.3},
'nose_tip': {'x': 739.8, 'y': 783.0}, 'mouth_left': {'x': 654.4, 'y': 833.8},
'mouth_right': {'x': 800.9, 'y': 840.8}, 'eyebrow_left_outer': {'x': 602.1, 'y':
646.0}, 'eyebrow_left_inner': {'x': 713.0, 'y': 659.7}, 'eye_left_outer': {'x':
641.1, 'y': 669.7}, 'eye_left_top': {'x': 668.8, 'y': 665.1}, 'eye_left_bottom':
{'x': 663.1, 'y': 687.1}, 'eye_left_inner': {'x': 689.6...
```

Demo Time!


InPrivate
ai-art (shwars) -
mPyPIConf
mPyPI_Tutorial
PeopleBlend
python groupby
mPyPI | mPyPI -
Модуль itertools
https://aiart-shwars.notebooks.azure.com/j/notebooks/PeopleBlending.ipynb
Find on page
Enter text to search
No results
Options
Microsoft Azure Notebooks Preview
My Projects
Help
Powered by jupyter PeopleBlending Last Checkpoint: 15.09.2019 (autosaved)
AI Art
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3.6
Run
Markdown
Enter/Exit RISE Slideshow
In [13]:
import numpy as np
target_triangle = np.float32([[130.0,120.0],[170.0,120.0],[150.0,160.0]])
size = 300
def affine_transform(img,attrs):
mc_x = (attrs['mouth_left']['x']+attrs['mouth_right']['x'])/2.0
mc_y = (attrs['mouth_left']['y'] + attrs['mouth_right']['y']) / 2.0
tr = cv2.getAffineTransform(np.float32([(attrs['pupil_left']['x'],attrs['pupil_left']['y']),
(attrs['pupil_right']['x'],attrs['pupil_right']['y']),
(mc_x,mc_y)]), target_triangle)
return cv2.warpAffine(img,tr,(size,size))
In [14]:
img_aligned = [affine_transform(i,a) for i,a in zip(images,imagepoints)]
display_images(img_aligned[:5])

CreepyFaces Solution

Whole Problem in One Piece of Code

```
res = (mp.get_files("images")
| mp.as_field('filename')
| mp.pshuffle
| mp.take(10)
| mp.apply('filename','image',imread)
| mp.apply('filename','meta',detect)
| mp.iter('filename',lambda x: print("Processing {}".format(x)))
| mp.filter('meta',lambda x: x is not None and len(x)>0)
| mp.apply('meta','faceland’, lambda x: x[0].face_landmarks.as_dict())
| mp.apply(['image','faceland'],'aligned’, lambda x: affine_transform(*x))
| mp.select_field('aligned')
| mp.pexec(merge))
```

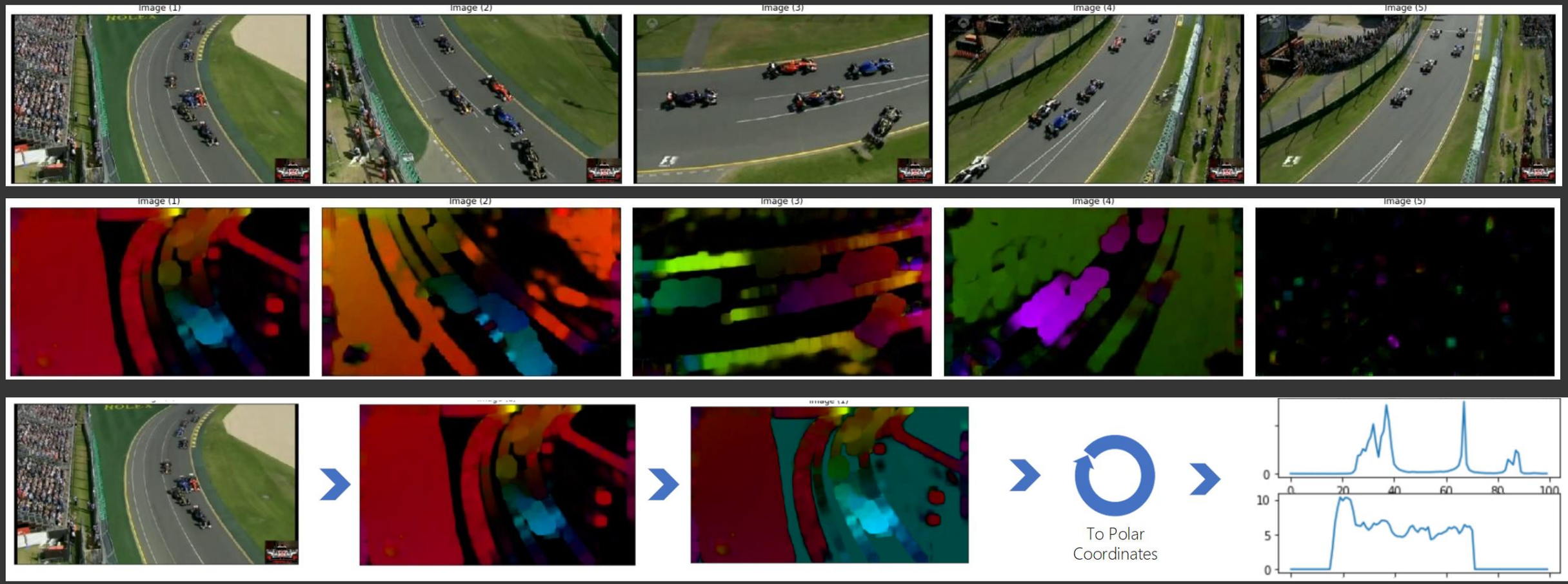
Case Study: Event Detection In Video

https://github.com/vJenny/race-events-recognition

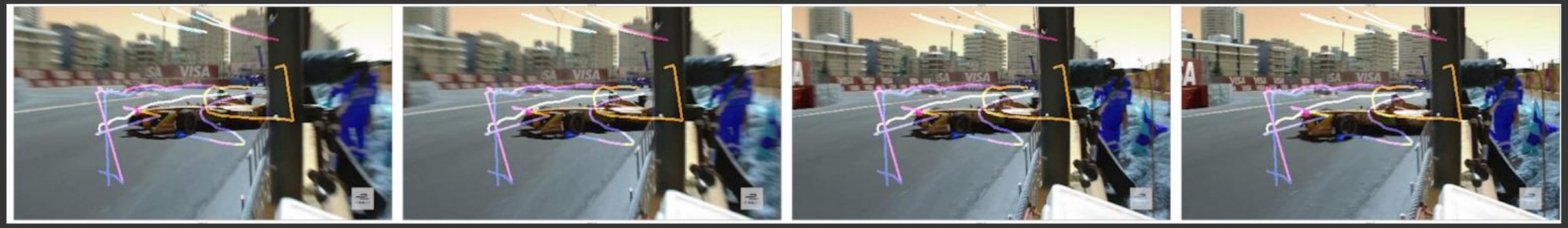
- Problems:
  - Detect car collision in Formula E
  - Detect goal events in soccer
  - Detect thefts in supermarket
- Data:
  - Not a lot of data (100s-1000s sample clips)
  - Complex behavior
  - => need some feature extraction

Video Features: Dense Optical Flow

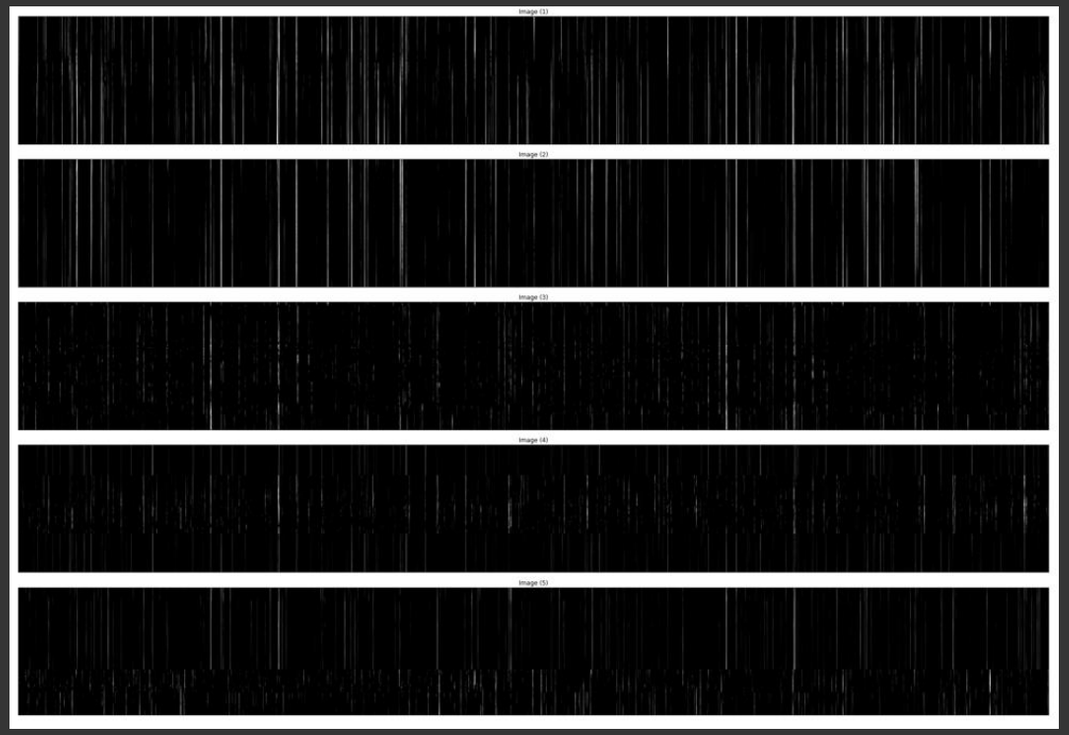
Optical Flow is a vector field of point movements between frames



Video Features: Focused Optical Flow

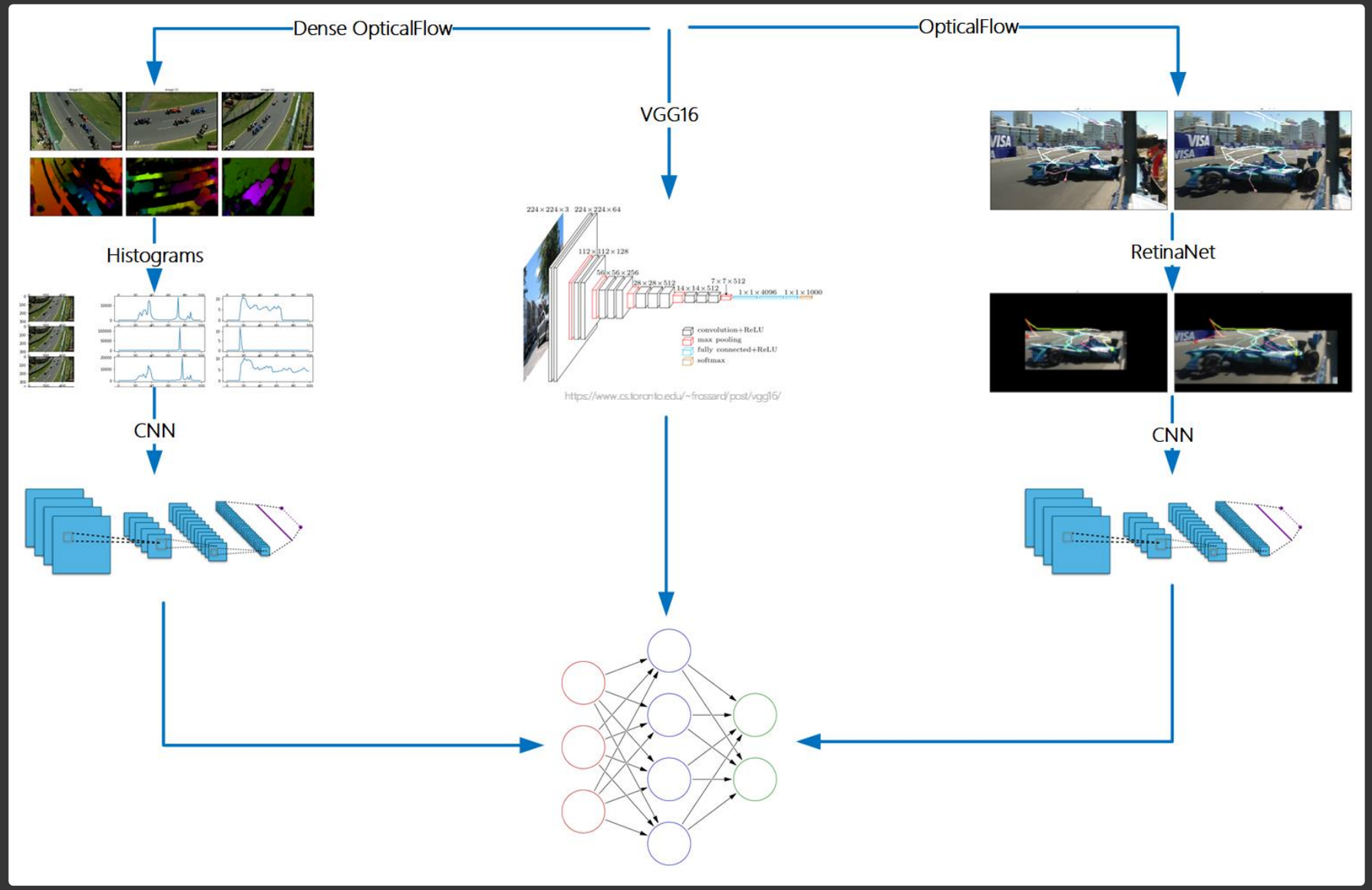


Video Features: VGG Embeddings



- Possible options:
  - 2D CNN – as depicted here
  - 3D CNN – preserve spatial location, slightly better accuracy

Overall Network Architecture



Training Code

```
## Setup combined dataflow

trainstream, valstream = (
    mp.get_datastream(data_dir, 'mp4', classes, split_filename='split2.txt')
    # RetinaFlow pipeline
    | mp.apply('filename', 'raw_rflows', load_optical)
    | mp.apply('raw_rflows', 'gradients', calc_gradients)
    | mp.apply('gradients', 'polar', to_polar)
    | mp.apply('polar', 'histograms', functools.partial(video_to_hist,params=hist_params))
    | mp.apply('histograms', 'res_rflows', functools.partial(zero_pad_hist,shape=retinaflow_shape))
    | mp.delfield(['raw_rflows', 'gradients', 'polar', 'histograms'])
    # VGG16 embeddings pipeline
    | mp.apply('filename', 'raw_vgg', lambda x: np.load(x+'.vgg16.npy'))
    | mp.apply('raw_vgg','prep_vgg', functools.partial(mp.zero_pad,max_frames=125))
    | mp.apply('prep_vgg', 'res_vgg', lambda x: x.reshape(125,-1,1))
    | mp.delfield(['raw_vgg', 'prep_vgg'])
    # DenseFlow pipeline
    | mp.apply('filename','raw_dflows', lambda x: np.load(x+'.flows.npy'))
    | mp.apply('raw_dflows','prep_dflows',functools.partial(mp.zero_pad,max_frames=max_frames))
    | mp.apply('prep_dflows', 'res_dflows', functools.partial(normalize_histograms, ang_diap=ang_diap, mag
_diap=mag_diap))
    | mp.delfield(['raw_dflows', 'prep_dflows'])
    # Split
    | mp.make_train_test_split
)
```

Model Inference on Video

```
(mp.videosource('video.mp4',video_size=video_size)
| mp.as_field('frame')
| mp.apply_batch('frame','vggx',get_vgg,batch_size=16)
| mp.apply('vggx','vgg',lambda x: x.reshape(16384,1))
| mp.silly_progress(elements=2000)
| mp.sliding_window_npy(['frame','vgg'],size=126)
| mp.inspect()
| mp.apply('frame','midframe',lambda x: x[60])
| mp.apply('vgg', 'score', lambda x:
                vgg_model.predict(np.expand_dims(x,axis=0))[0])
| mp.apply(['midframe','score'],'fframe',imprint)
| mp.select_field('fframe')
| mp.collect_video('output.mp4')
)
```

Reading Pascal VOC

- Pascal VOC is XML-based format with pretty complex structure
- In mPyPl, it makes sense to automatically flatten some fields

```
data =
mp.get_xmlstream_from_dir(
    annotation_dir,
    list_fields=['object'],
    flatten_fields=['bndbox','size'],
```

```
data =
mp.get_pascal_annotations(
annotation_dir)
```

```
{'folder': 'HollywoodHeads',
 'filename': 'mov_012_063018.jpeg',
 'size_width': '548', 'size_height': '226', 'size_depth':
'3',
 'segmented': '0',
 'object': [{'name': 'head',
   'bndbox_xmin': '340', 'bndbox_ymin': '20',
   'bndbox_xmax': '397', 'bndbox_ymax': '81',
   'difficult': '0'},
  {'name': 'head',
   'bndbox_xmin': '80', 'bndbox_ymin': '63',
   'bndbox_xmax': '119', 'bndbox_ymax': '112',
   'difficult': '0'}]}
```

Using Pascal VOC

- All numbers obtained from XML are strings, so as_int, etc. can be used

```
def imprint(arg): # arg[0] is image, arg[1] is a list of
`objects`
  for x in arg[1]:
    cv2.rectangle(arg[0],
        (x.as_int('bndbox_xmin'), x.as_int('bndbox_ymin')),
        (x.as_int('bndbox_xmax'), x.as_int('bndbox_ymax')),
        (255,0,255),3)

(data | take(5)
| mp.apply('filename','img',im_load)
| mp.apply(['img','object'],None,imprint)
| mp.select_field('img')
| pexec(show_images) )
```

Takeaway

```
# pip install mPyPI
```

http://shwars.github.io/mPyPI

Functional / monadic way is cool

We have applied it to number of scenarios:

- Complex DNN training
- Video Rendering / Inference
- Object Detection
- Science Art

You can, too! If you do - let us know!

Do the Creepy Faces exercise

http://github.com/CloudAdvocacy/CreepyFaces

Microsoft

mPyPl: A Monadic Way to do Complex Data Processing in Python

Dmitry Soshnikov

Cloud Developer Advocate / Software Engineer, Microsoft

@shwars – http://soshnikov.com