Video Object Segmentation based on pixel-level annotated dataset

Midpoint Presentation

Chong Hu/ch3467
Yanlin Liu/yl4238

Contents

1. Quick Recap
2. Web InterFace
3. Data Preprocess
4. Model
5. Intermediate Result
6. Future Work

Quick Recap

Video Object Segmentation

Goal: extracting foreground objects from video clips.

Application:

- video summarization/editing
- object tracking
- video action detection[3][4]
- autonomous driving
- etc…

Figure 1. separating foreground object(s) from the background region of a video[5]

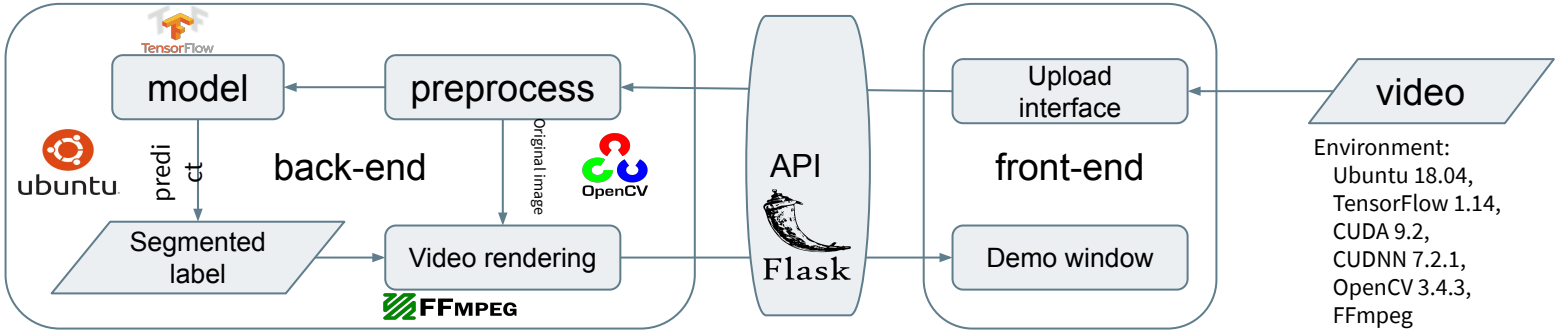
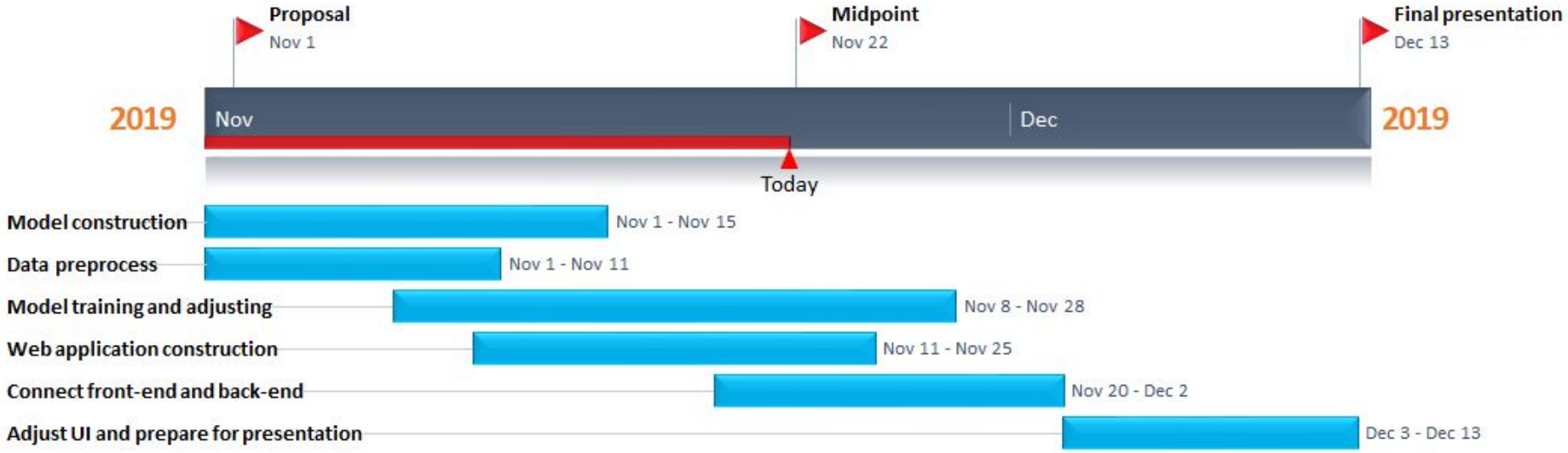


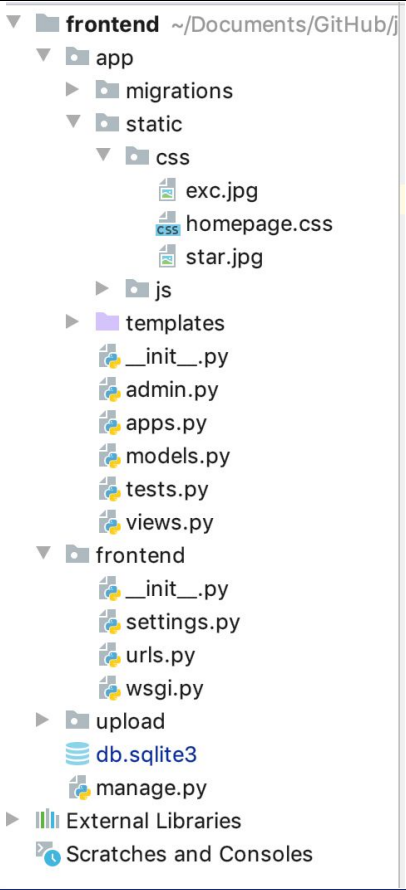
Figure 2. Simple overview diagram of system

Environment:
Ubuntu 18.04,
TensorFlow 1.14,
CUDA 9.2,
CUDNN 7.2.1,
OpenCV 3.4.3,
FFmpeg
API: Flask
Front-end: django

Schedule


Proposal
Nov 1
Midpoint
Nov 22
Final presentation
Dec 13
2019
Nov
Dec
2019
Today
Model construction
Nov 1 - Nov 15
Data preprocess
Nov 1 - Nov 11
Model training and adjusting
Nov 8 - Nov 28
Web application construction
Nov 11 - Nov 25
Connect front-end and back-end
Nov 20 - Dec 2
Adjust UI and prepare for presentation
Dec 3 - Dec 13

Web Interface



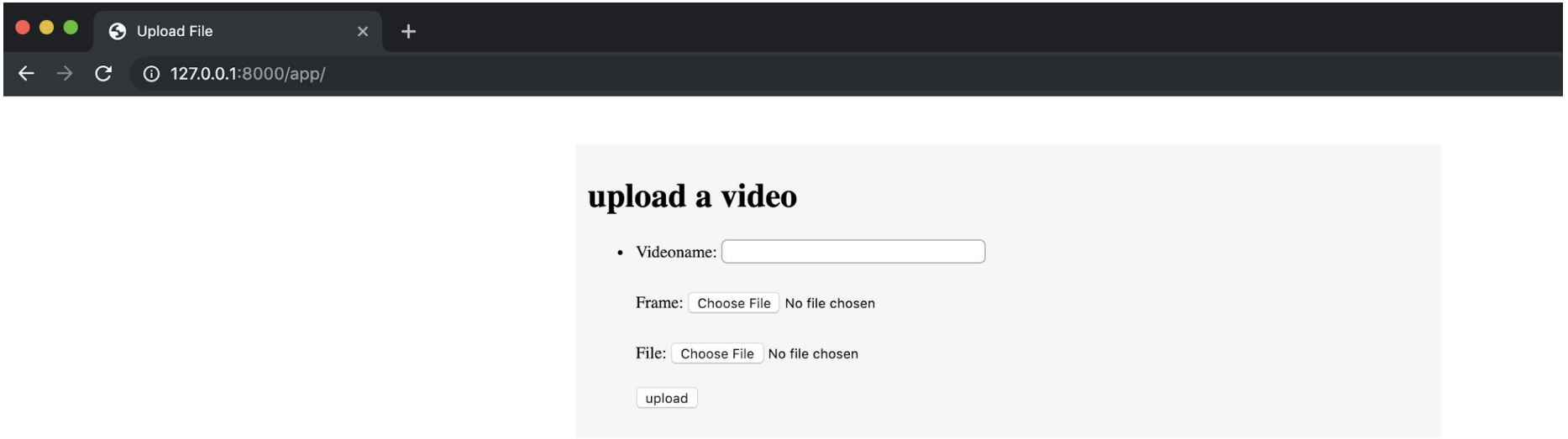
homepage.html | homepage.css | models.py | views.py

```python
from django.db import models

# Create your models here.
class Video(models.Model):
    videoname = models.CharField(max_length = 30)
    Frame = models.FileField(upload_to='./upload/First_Frame')
    File = models.FileField(upload_to= './upload/File')

    def __str__(self):
        return self.videoname

```

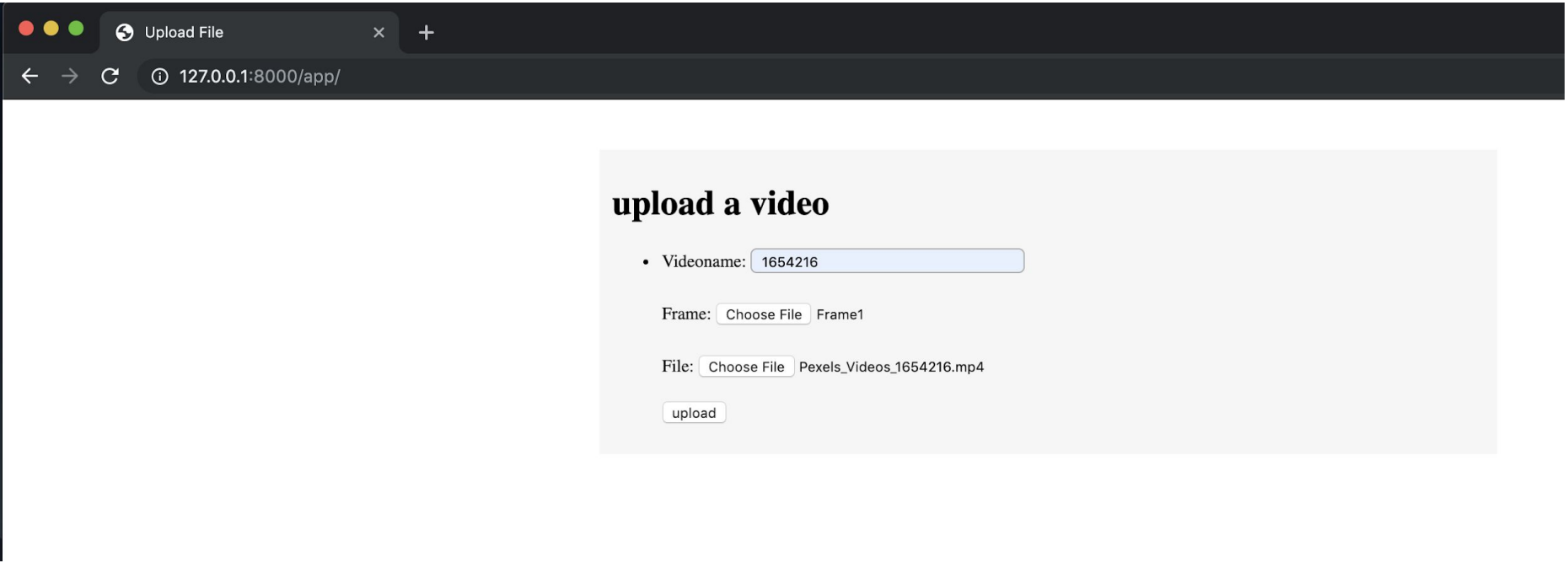
homepage.html | homepage.css | models.py | views.py

```python
from django.shortcuts import render,render_to_response
from django import forms
from django.http import HttpResponse
from app.models import Video
# Create your views here.
class VideoForm(forms.Form):
    #file_field = forms.FileField(widget=forms.ClearableFileInpu
    videoname = forms.CharField()    #string
    Frame = forms.FileField()
    File = forms.FileField()     #file

def homepage(request):
    if request.method == "POST":
        uf = VideoForm(request.POST, request.FILES)
        if uf.is_valid():  # if valid
            videoname = uf.cleaned_data['videoname']
            Frame = uf.cleaned_data['Frame']
            File = uf.cleaned_data['File']
            # save file
            video = Video()
            video.videoname = videoname
            video.Frame = Frame
            video.File = File
            video.save()
            return HttpResponse('upload successfully!')
    else:
        uf = VideoForm()

    return render_to_response('homepage.html',{'uf':uf})
```

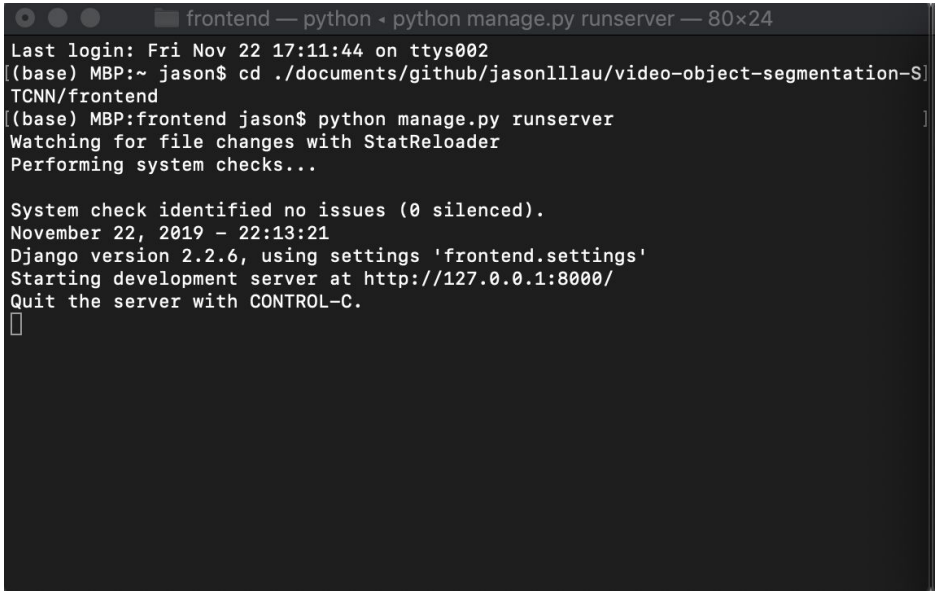
Web Interface

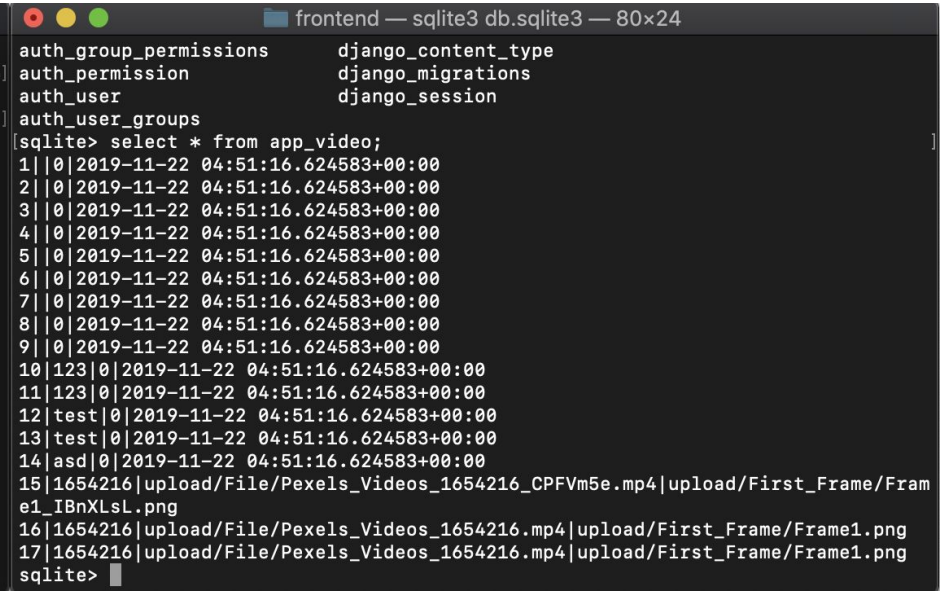


Web Interface



Web Interface

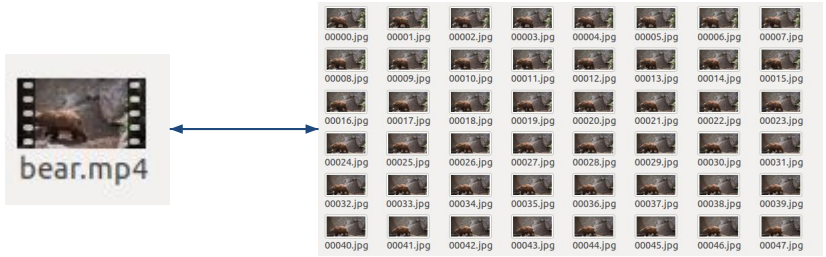




Data preprocess

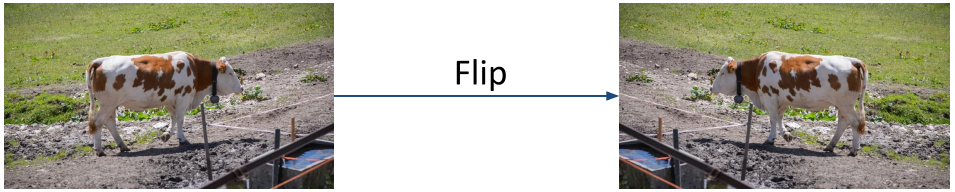
1. video <-> image

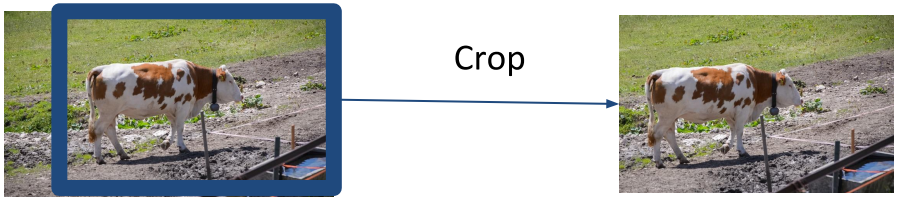
   480p 854x480x3



2. Data augmentation:

   Flip and Crop





3. Train set vs. test set:
   a. Train set 30 video sequences
   b. Test set 20 video sequences

Possible supplement:

YouTube dataset; DAVIS-2017

Model Construction

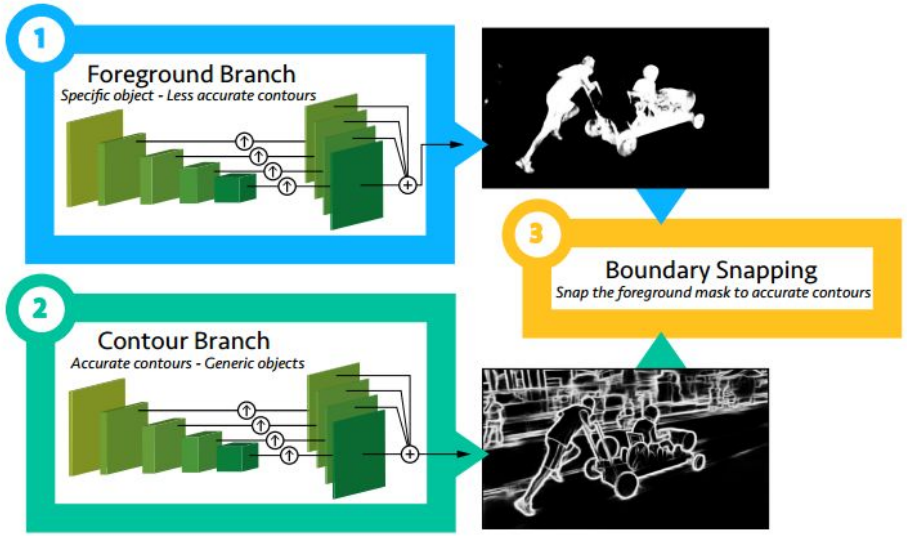
- Model update:

  Spatiotemporal CNN -> One-Shot VOS
  - Reasons:
    - Simpler: GPU limitation & easy to implement
    - Faster: One-Shot
    - Comparable result: ~80% in paper

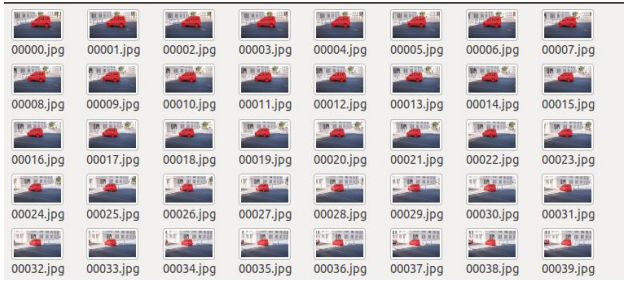
- Model Structure
  - Parent network + Finetune (Transfer learning)
  - Two branch: Foreground & Contour
  - Loss function: imbalanced version of binary pixel-wise cross-entropy

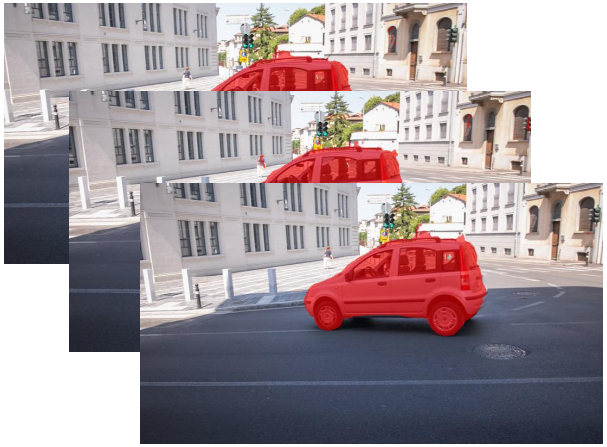
$$\mathcal{L}_{mod} = -\beta \sum_{j \in Y_+} \log P(y_j=1|X) - (1-\beta) \sum_{j \in Y_-} \log P(y_j=0|X) \quad (1)$$

where $\beta = |Y_-|/|Y|$.



Intermediate Result





Model prediction per frame

render

Intermediate video demo with foreground mask

To Do

- Web:
  - Further Web design (data visualization)
  - Connection with back-end (output video demo)
  - UI design (navigation bar; video information)
- Model
  - Better model weight
  - Revise model structure
  - Provide corresponding API
- Evaluation
  - Numerical evaluations
- Application
  - Whole pipeline

Thank You