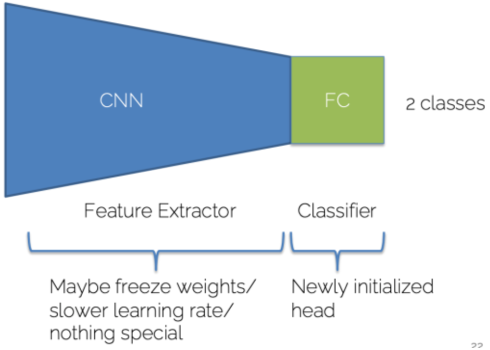
Introduction to Deep Learning (I2DL)

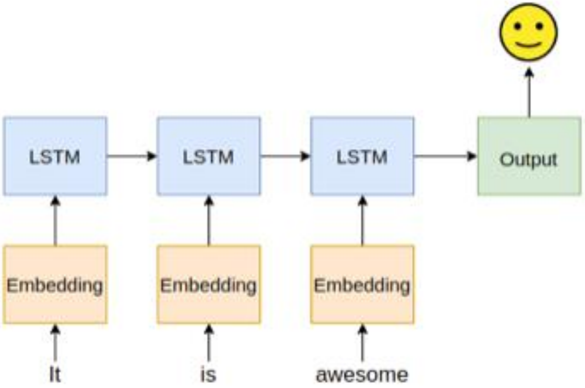
Exercise 11: RNNs

Today's Outline

- Exercise 10 Review
  - Semantic Segmentation

- Recurrent Neural Networks
  - Exercise 11

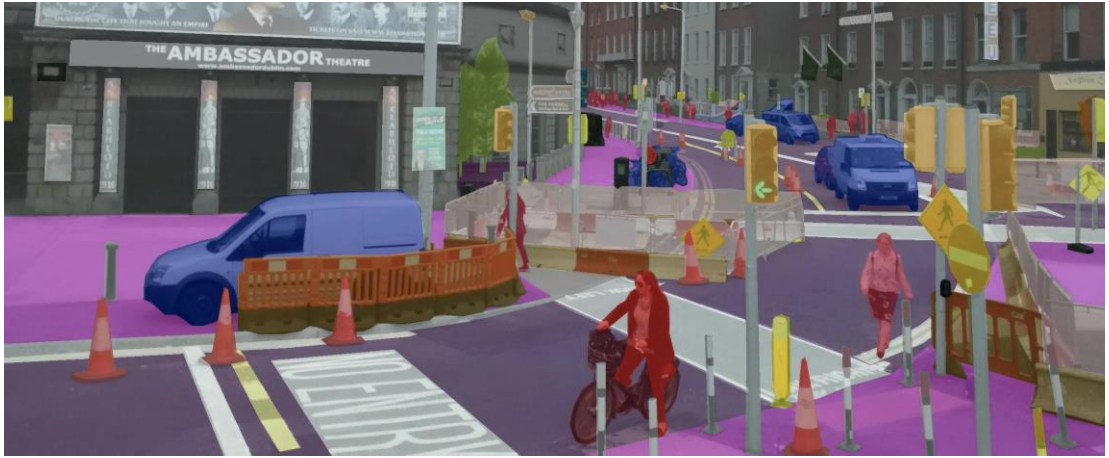




Exercise 10

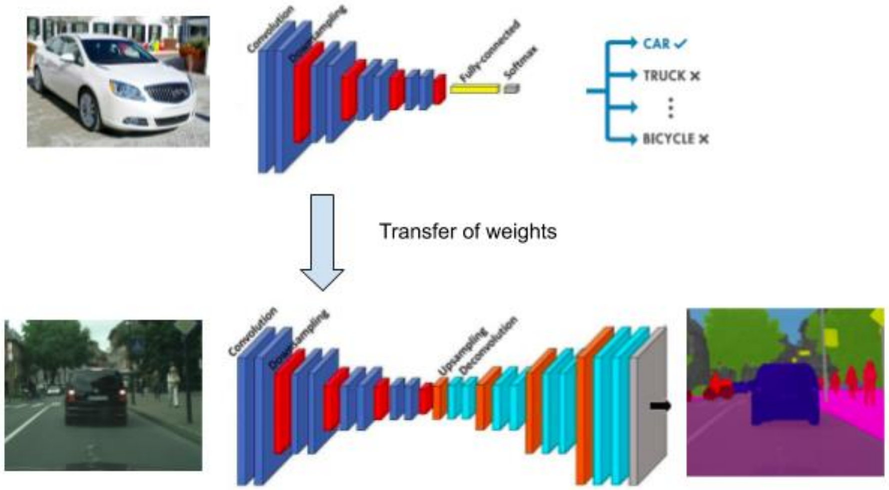
Exercise 10: Semantic Segmentation

- **Goal:** Assign a label to each pixel of the image
- **Output of the network:** Segmentation mask with same shape as input image
- **Dataset:** MSRC v2 dataset, 23 object classes, contains 591 images with “accurate” pixel-wise labeled images



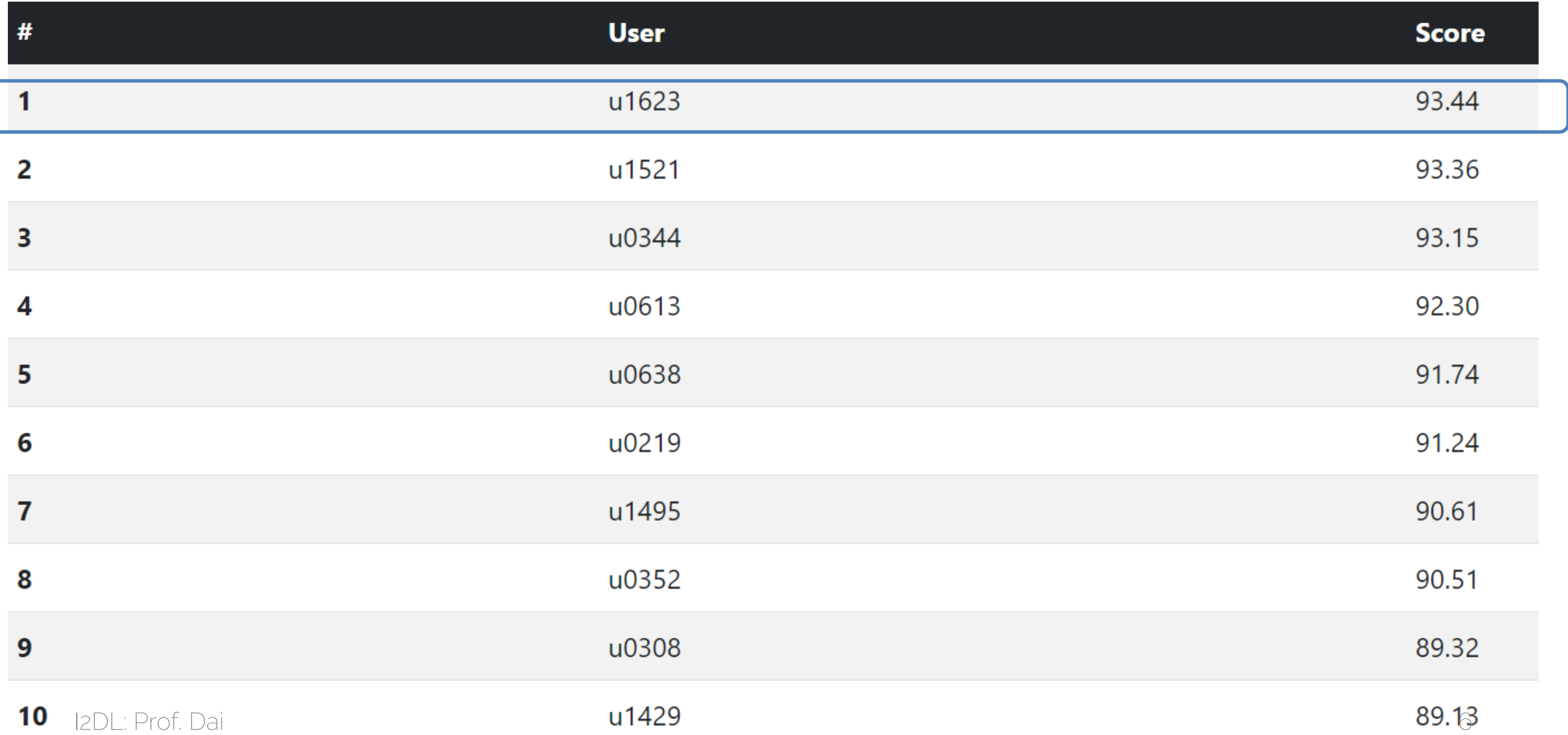
Suggested Approach

- Idea: Encoder-Decoder Architecture
- Transfer Learning: CNNs trained for image classification contain meaningful information that can be used for segmentation -> Encoder
- Check out: pre-trained networks like MobileNets



Transfer of weights

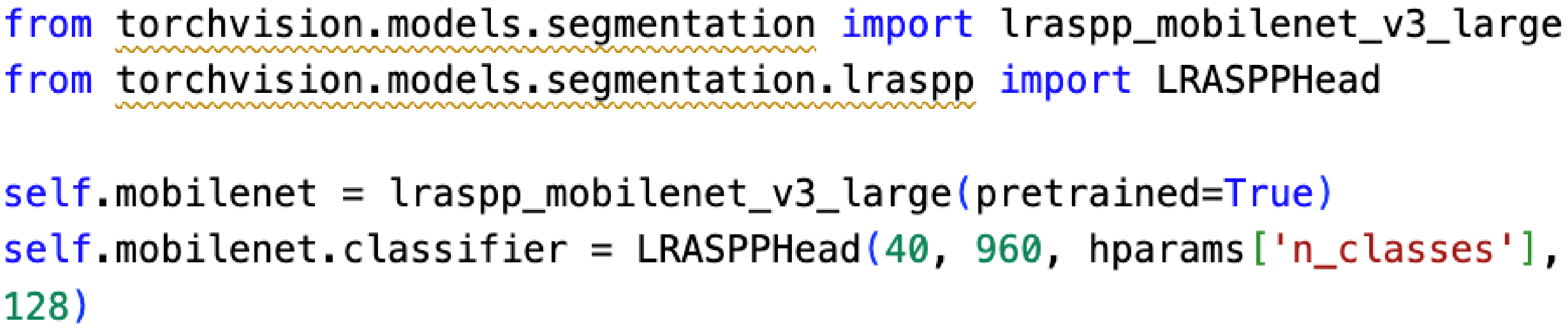
Leaderboard



#	User	Score
1	u1623	93.44
2	u1521	93.36
3	u0344	93.15
4	u0613	92.30
5	u0638	91.74
6	u0219	91.24
7	u1495	90.61
8	u0352	90.51
9	u0308	89.32
10	u1429	89.13

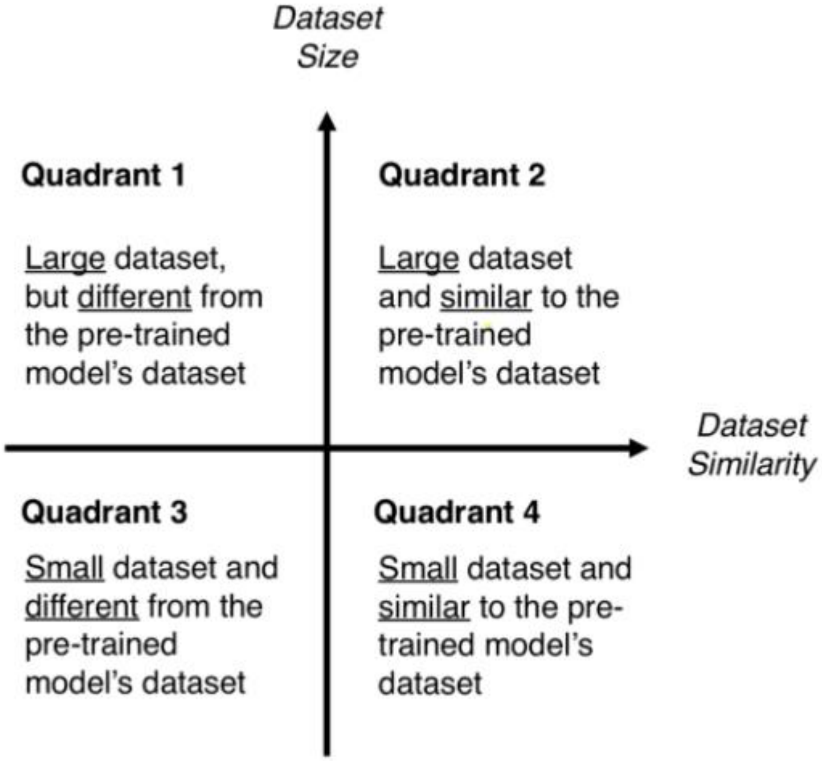
"Default" Approach (93.15)

- Take an already pretrained segmentation network
- Change the output layer to our number of classes
- Success!



```
from torchvision.models.segmentation import lraspp_mobilenet_v3_large
from torchvision.models.segmentation.lraspp import LRASPPHead

self.mobilenet = lraspp_mobilenet_v3_large(pretrained=True)
self.mobilenet.classifier = LRASPPHead(40, 960, hparams['n_classes'],
128)
```

When/what to finetune?



Dataset Size

Dataset Similarity

Quadrant 1

Large dataset, but different from the pre-trained model's dataset

Quadrant 2

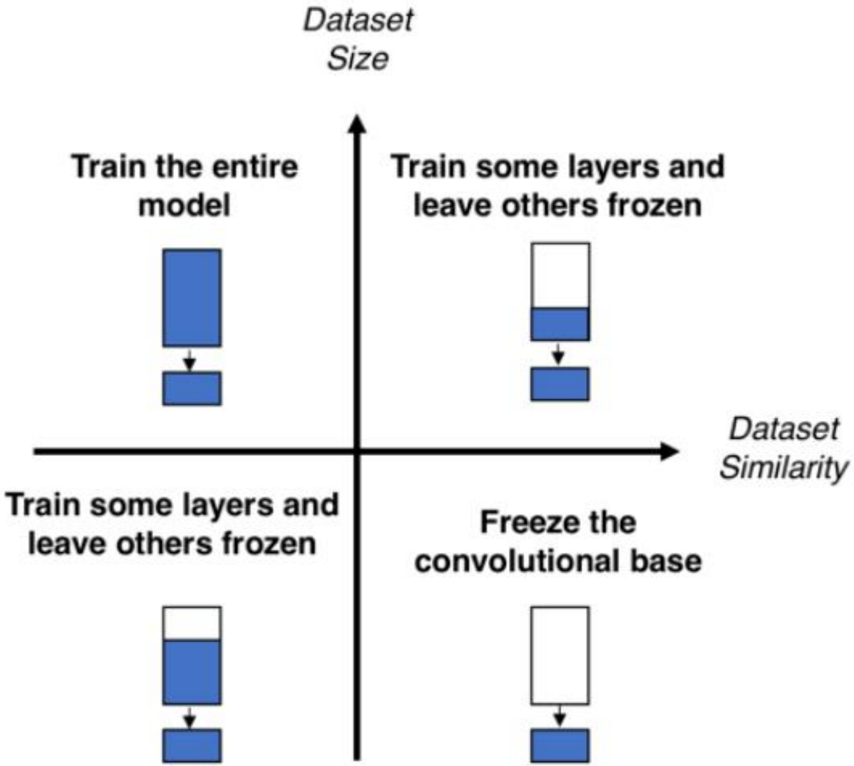
Large dataset and similar to the pre-trained model's dataset

Quadrant 3

Small dataset and different from the pre-trained model's dataset

Quadrant 4

Small dataset and similar to the pre-trained model's dataset



Dataset Size

Dataset Similarity

Train the entire model

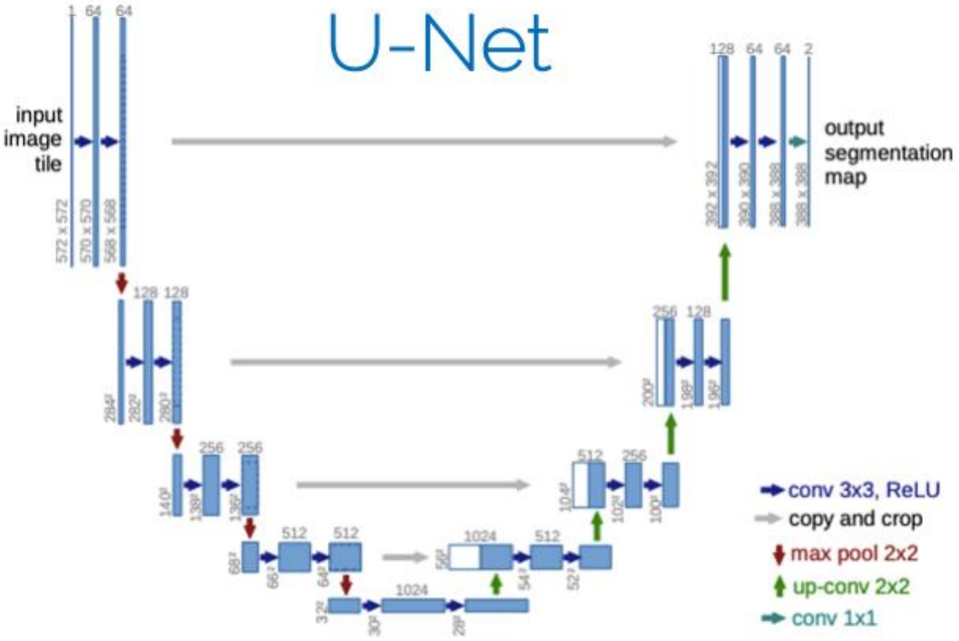
Train some layers and leave others frozen

Train some layers and leave others frozen

Freeze the convolutional base

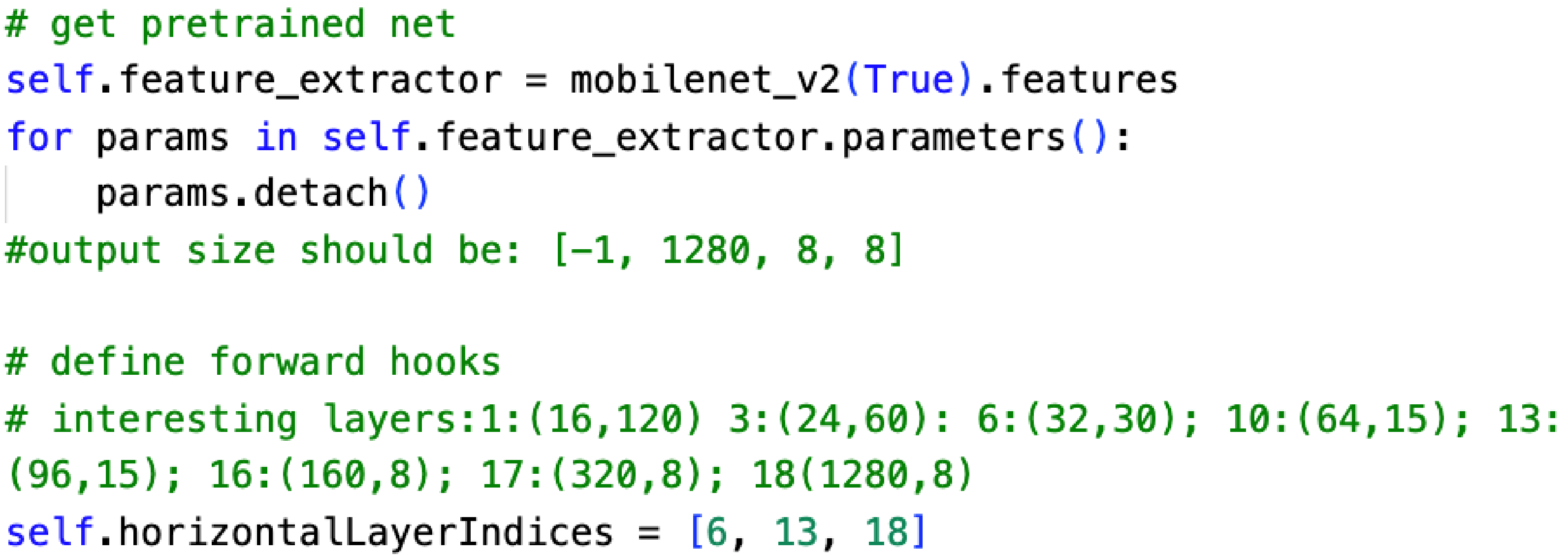
“Default” Approach

- Idea: Let’s build a UNet



Let's start with pretrained backbone

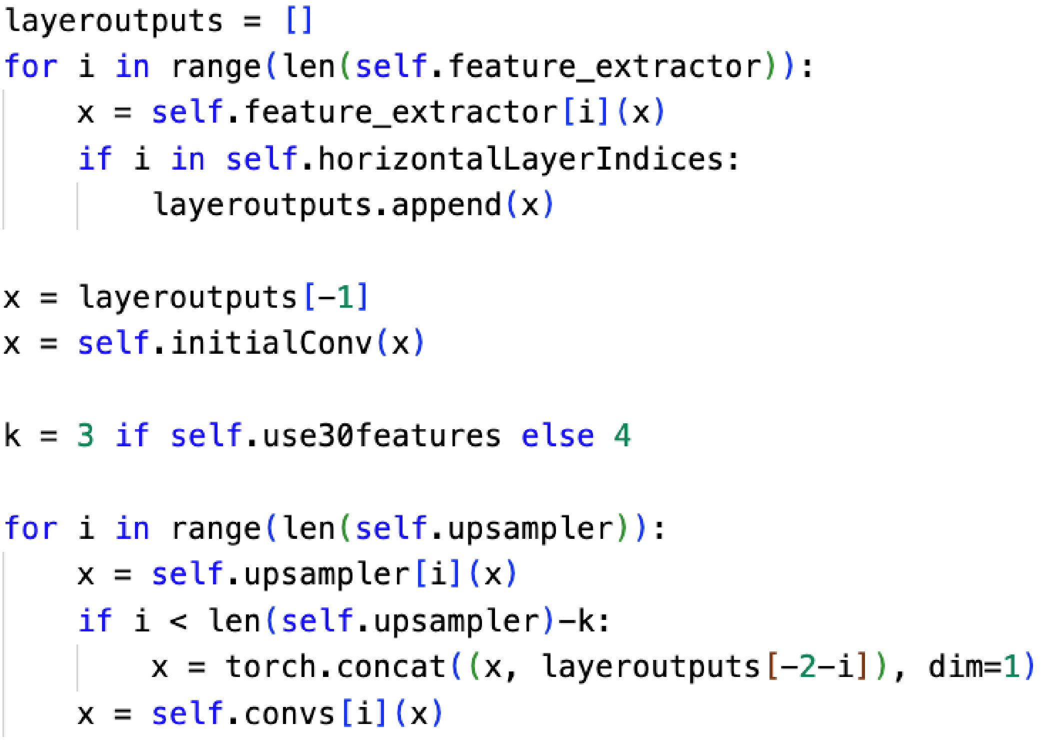
- Get pretrained network and identify skip connection candidates



```
# get pretrained net
self.feature_extractor = mobilenet_v2(True).features
for params in self.feature_extractor.parameters():
    params.detach()
#output size should be: [-1, 1280, 8, 8]

# define forward hooks
# interesting layers:1:(16,120) 3:(24,60): 6:(32,30); 10:(64,15); 13:
(96,15); 16:(160,8); 17:(320,8); 18(1280,8)
self.horizontalLayerIndices = [6, 13, 18]
```

Let's check forward

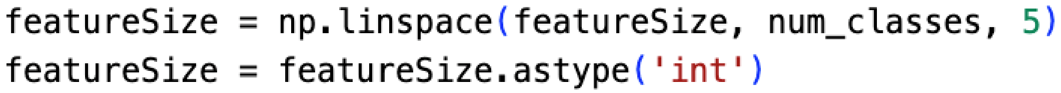
- Forward backbone but keep track of skips
- "Bottleneck"
- Upsampling and filling in skips



```
layeroutputs = []
for i in range(len(self.feature_extractor)):
    x = self.feature_extractor[i](x)
    if i in self.horizontalLayerIndices:
        layeroutputs.append(x)

x = layeroutputs[-1]
x = self.initialConv(x)

k = 3 if self.use30features else 4

for i in range(len(self.upsampler)):
    x = self.upsampler[i](x)
    if i < len(self.upsampler)-k:
        x = torch.concat((x, layeroutputs[-2-i]), dim=1)
    x = self.convs[i](x)
```

Some comments

- Good: usage of variables for filter/network size
  - Just don't hardcore numbers in your init unless you really want to keep them



```
featureSize = np.linspace(featureSize, num_classes, 5)
featureSize = featureSize.astype('int')
```

- Don't forget to use data augmentation even when transfering weights
  - Could have been done outside of notebook, just a reminder ☺

Recurrent Neural Networks

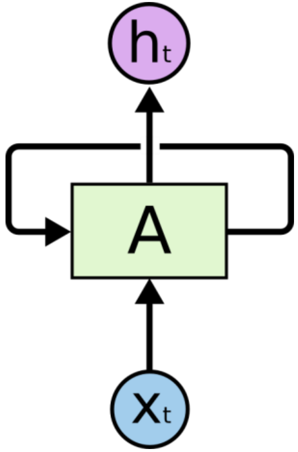
Recurrent Neural Networks

- **Idea:** Network that can capture the relationship between the inputs
- **RNNs:** Learning process is not independent
  - Remember things from processing trainings data
  - Remember things learnt from prior inputs, prior inputs influences decision
- **In other words:** RNNs produce different outputs for same input depending on previous outputs in the series.

$$\boldsymbol{A}_t = \boldsymbol{\theta}_c \boldsymbol{A}_{t-1} + \boldsymbol{\theta}_x \boldsymbol{x}_t$$

Previous hidden state

input



Output / Hidden State

Input

RNN Concepts

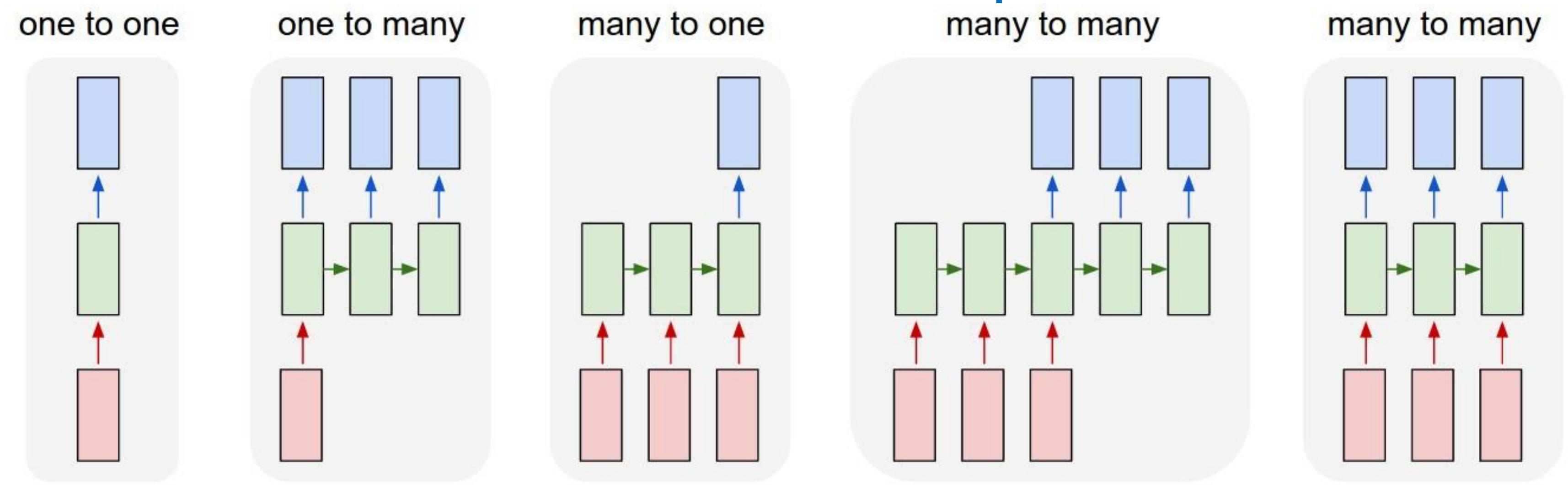


Image Classification

Image Captioning (image -> seq of words)

Sentiment Analysis (seq of words –> sentiment)

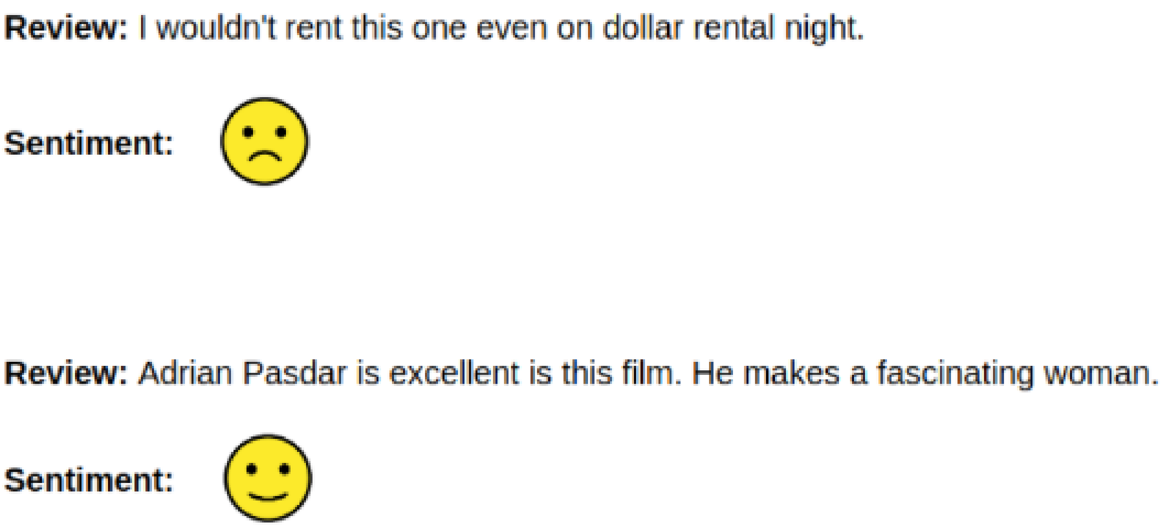
Machine Translation (seq of words -> seq of words)

Video Classification on frame level (seq of frames -> seq of class.)

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Exercise 11

Exercise 11: Goal



Review: I wouldn't rent this one even on dollar rental night.

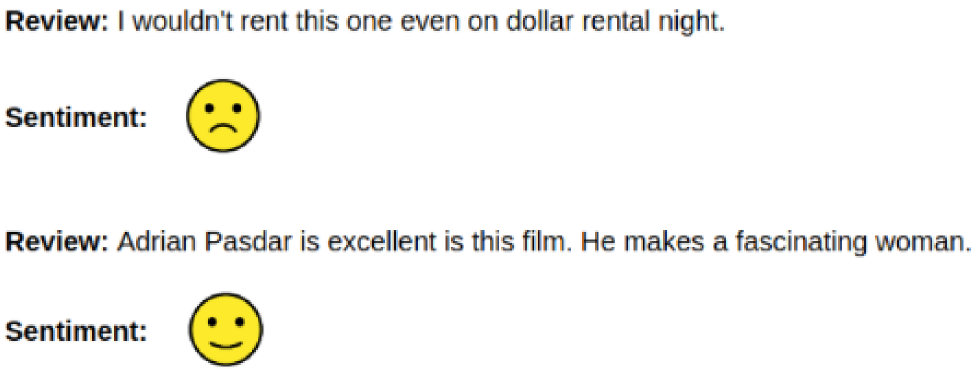
Sentiment: ☹

Review: Adrian Pasdar is excellent is this film. He makes a fascinating woman.

Sentiment: ☺

Exercise 11: Content

- Optional Notebook: RNNs and LSTMs
- Notebook 1: Text Preprocessing and Embedding
- Notebook 2: Sentiment Analysis



Review: I wouldn't rent this one even on dollar rental night.

Sentiment: ☹

Review: Adrian Pasdar is excellent is this film. He makes a fascinating woman.

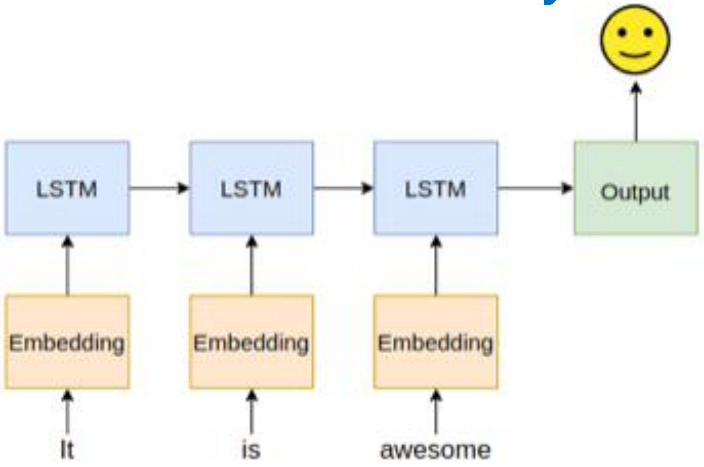
Sentiment: ☺

Notebook 1: Text Preprocessing and Embedding

- **Sequential Data:** from image data to text data
- **Dataset:** IMDb sentiment analysis dataset
- **Goal of the notebook:**
  - Data preparation
  - Implementation of Embedding layer

Notebook 2: Sentiment Analysis

- Network Architecture:
  - Embedding layer
  - RNN
  - Output layer,
    e.g. fully-connected layer
- Loss: Cross-Entropy Loss
- Performance measure: Accuracy
- Goal of the notebook: Implement and train a recurrent neural network for sentiment analysis



Good Luck with the exercise and exam!

☺