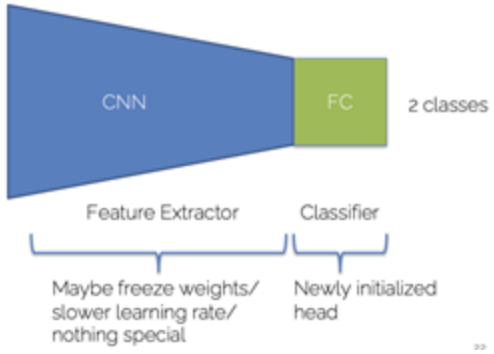
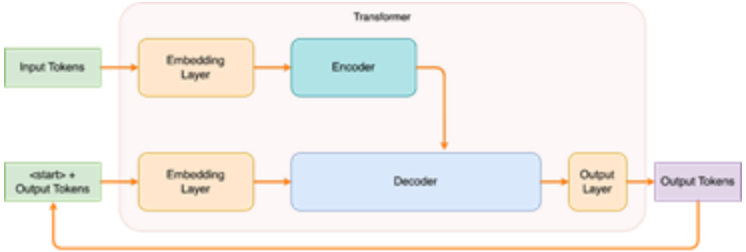
Introduction to Deep Learning (I2DL)

Exercise 11 and 12: NLP

Today's Outline

- Exercise 10 Review
- Natural Language Processing
  - Exercise 11 and 12





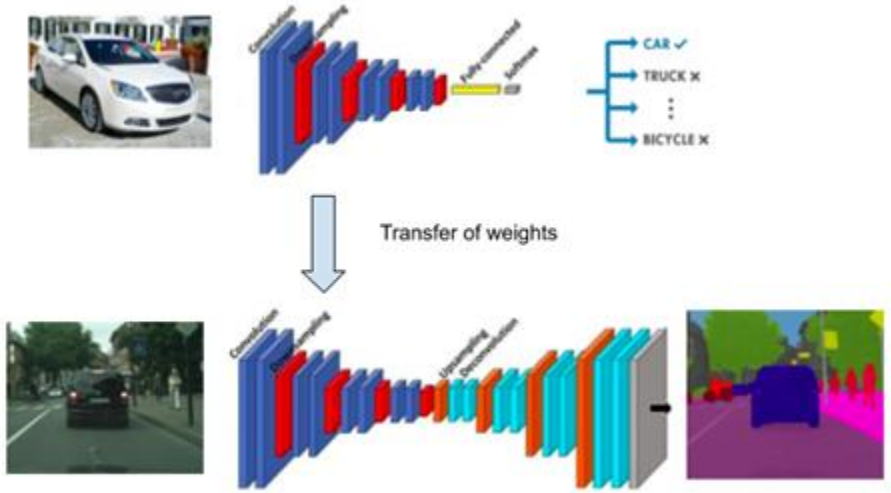
Exercise 10

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

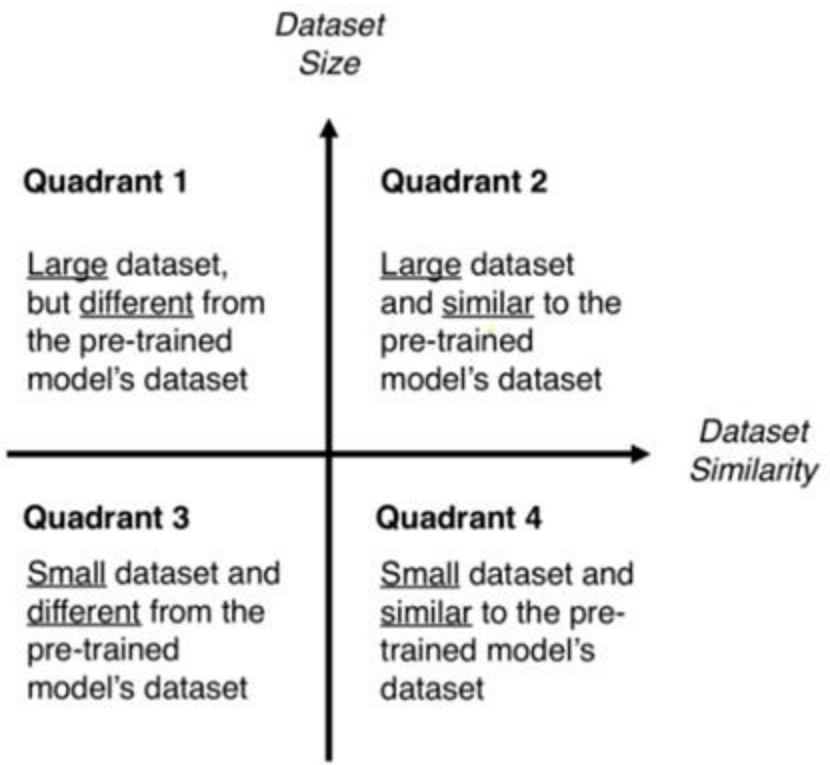


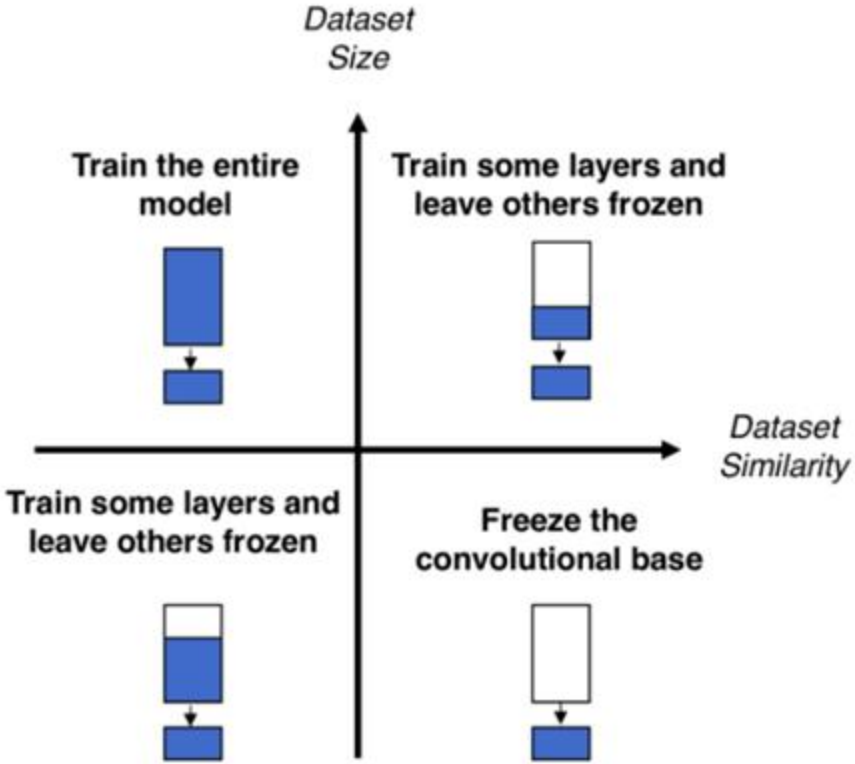
“Default” Approach (93.15)

- 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)
```

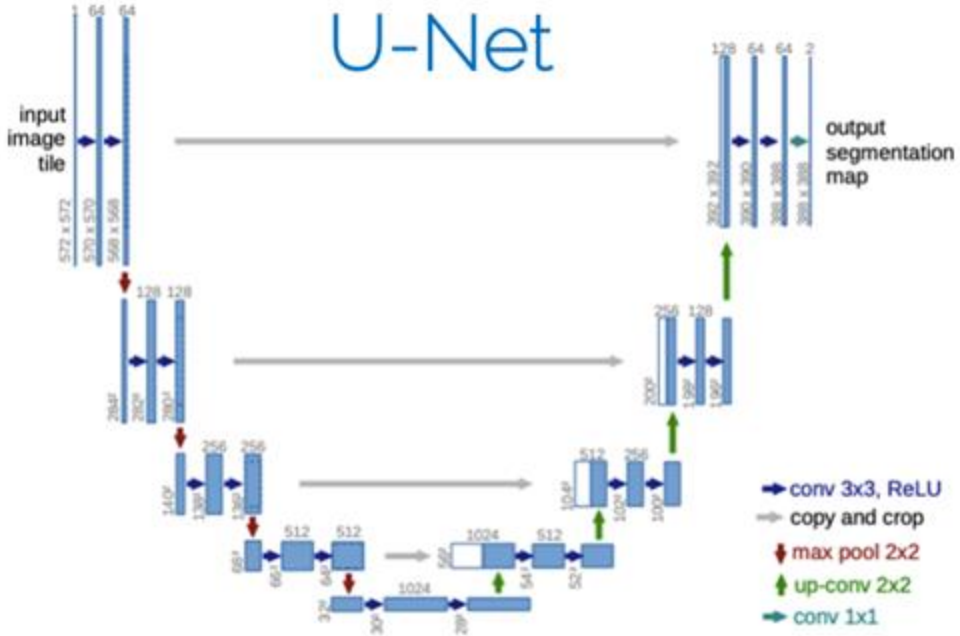
When/what to finetune?





So… something else? (92.38)

- 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]
```

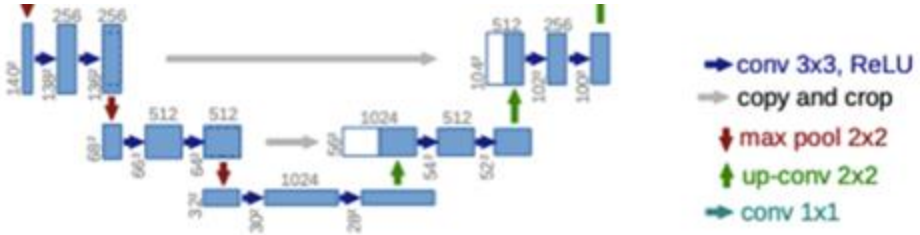
Let's check forward

- 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

- Bottleneck seems to be too tight
  - Make room for multiple non-linearities before upsampling and merging



- Bottleneck is using 8x8 with 1280 features
  - Use 1x1 convolutions to shrink down size first (we are not imagenet with 1000 classes where would need it)

0	feature_extractor	Sequential	2.2 M
1	input_normalization	Normalize	0
2	upsampler	ModuleList	0
3	convs	ModuleList	~~1.6 M~~
4	initialConv	Sequential	410 K
5	lossFcn	CrossEntropyLoss	0

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 ☺

Natural Language Processing

Natural Language Processing

- So far, it has always been clear how to feed data into our model
- Tabular Data -> Load each row as vector and normalize data
- Images -> Convert Image into Matrix for convolution or flatten into vector for linear layers
- Text -> „What should I do with this?“ -> ?

Tokenization

- Split text into individual tokens
- Tokens can be
  - Individual words like „Hi“, „these“, „are“, „tokens“
  - Subwords like “sub“ and „words“
  - Letters like „h“, „i“
  - Punctuations and white spaces like „!“, „,“ or „ “
  - Combination of all

- Assign each token an ID: „Hi“ -> 53, „Bye“ -> 647

Tokenization - Training

- For simple Tokenizer:
  - Loop through dataset and split strings by whitespace
  - Add every new „word“ into dictionary and assign ID
  - That's it!

- Problems:
  - What happens with words that were not in the dataset?

- Possible Solutions:
  - Add unknown token placeholder
  - Use a more sophisticated tokenizer like Byte-Pair Encoder

Exercise 11
Notebook 1

Map Style Datasets

- So far we have always used a map style Dataset:
  - Given an index, return the item at this index

- This approach works great for
  - Small Tables -> simply return the row where row_id = index
  - Images -> load image from disk by indexing a list of paths

Map Style Datasets

- Advantages:
  - Data can be prefetched
  - Data shuffling is easy to implement
  - Parallelization is easy to implement

- Problems:
  - What happens if your data is stored in very large text files of tables? We cannot load everything in memory to properly index it
  - What happens if our data is coming in as a stream? (Think of live audio or video feed!)

Iterable Datasets

- Do we really need to index the data? All we want is to get the next sample!

- Iterable Datasets: Instead of a __getitem__(index) method we implement a __iter__() method

- Read through the file/table line by line and yield the current sample
    - We do not have to load the entire file into memory!

Iterable Datasets

Exercise 12
Notebook 1

- Advantages
  - Less Memory Usage
  - Easy to seamlessly cycle through multiple files

- Disadvantages
  - Shuffling data in data loader not possible
  - Parallel Processing and using multiple workers for data loading not as easy to implement

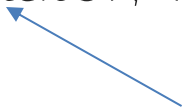
Data Collator

- So far data usually always had the same dimensions
  - We can easily create batches by adding a tensor dimension!

- Problem with natural language: Not every sentence is the same length!

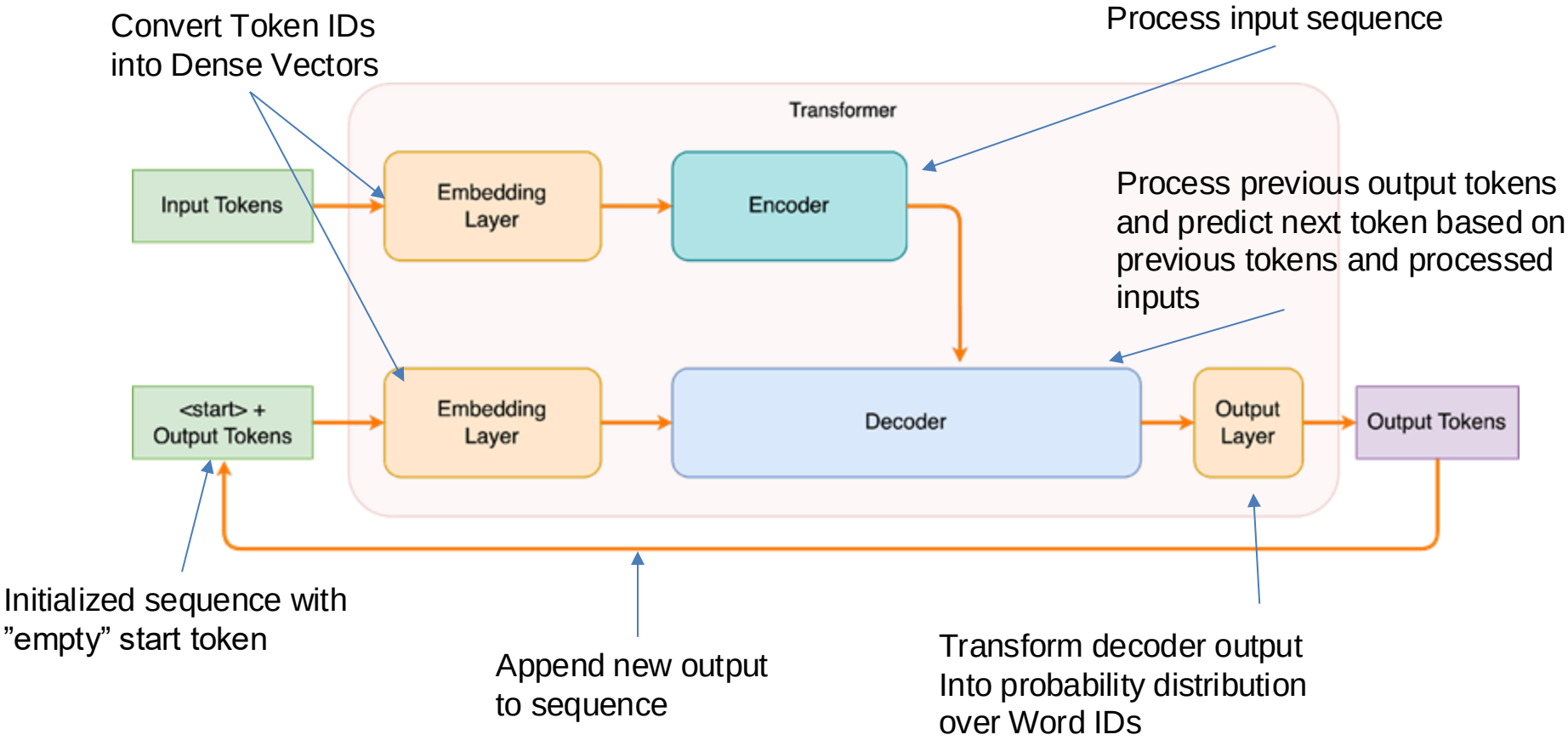
Data Collator

- Solution: Add padding tokens to the shorter sentences in a batch to match the lengths
  - [“Hi” “how” “are” “you” “?”] -> 5 Tokens
  - [“Good” “and” “you” “?” “<pad>”] -> 4 Tokens -> Add pad to match lengths!
- This is implemented in the Data Collator, which is part of the Data Loader!



Exercise 12
Notebook 1

Transformers –High Level



Exercise 11

Exercise 11: Content and Goal

- Notebook 1
  - Introduction to Byte Pair Tokenization
  - Training Algorithm and Examples
  - Goal: Train own tokenizer on a dataset

- Notebook 2
  - Introduction to Embeddings – Convert Tokens into Vectors

Exercise 11: Content and Goal

- Notebook 3
  - Intuition behind Attention Mechanism
  - Build Attention and Multi Head Attention Block from Scratch

- Notebook 4
  - Positional Encodings – Fixing the Attention Block

Exercise 12

Exercise 12: Content and Goal

- Notebook 1
  - Iterable Style Datasets
  - Data Collator for NLP Tasks

- Notebook 2
  - Transformer Model from Scratch

Good Luck with the exercise and exam!