Domain Adaptation a.k.a Transfer Learning, A case example: Finetunning

Alvaro Soto

Computer Science Department, PUC

Transfer Learning

- Learn a model using a rich dataset: Source dataset.
- Transfer the learned knowledge to a second dataset: Target dataset.
- How?, several techniques. Finetunning is the most popular.
- When? transfer learning is highly relevant in cases where one has a rich dataset with lot of annotation and a related but poor dataset with a limited number of annotations.

Transfer Learning

Source Domain

- Lots of labeled data

$$P_S = (X_S, Y_S)$$

Target Domain

- Limited labels

$$P_T = (X_T, Y_T)$$

Hope is that:

$$P_T \approx P_S$$

Transfer Learning

Source Domain

- Lots of labeled data

$$P_S = (X_S, Y_S)$$

Target Domain

- Limited labels

$$P_T = (X_T, Y_T)$$

Hope is that:

$$P_T \approx P_S$$

Or at least

$$P_T(X_T) \approx P_S(X_S)$$

Transfer Learning

Finetunning and Deep Learning

- DL learns features using big models (millions of parameters) and big data (millions of labeled examples).
- Good news is that most DL models learn features that are informative for a wide variety of related tasks.

Example

- In the image domain, it is common to use Imagenet dataset (1.2 million images with 1000 categories) to train a deep learning model (CNN) to recognize objects in images.
- The features learned by the model can then be used to perform related visual tasks such as: scene recognition, face detection, action recognition, people detection, etc.

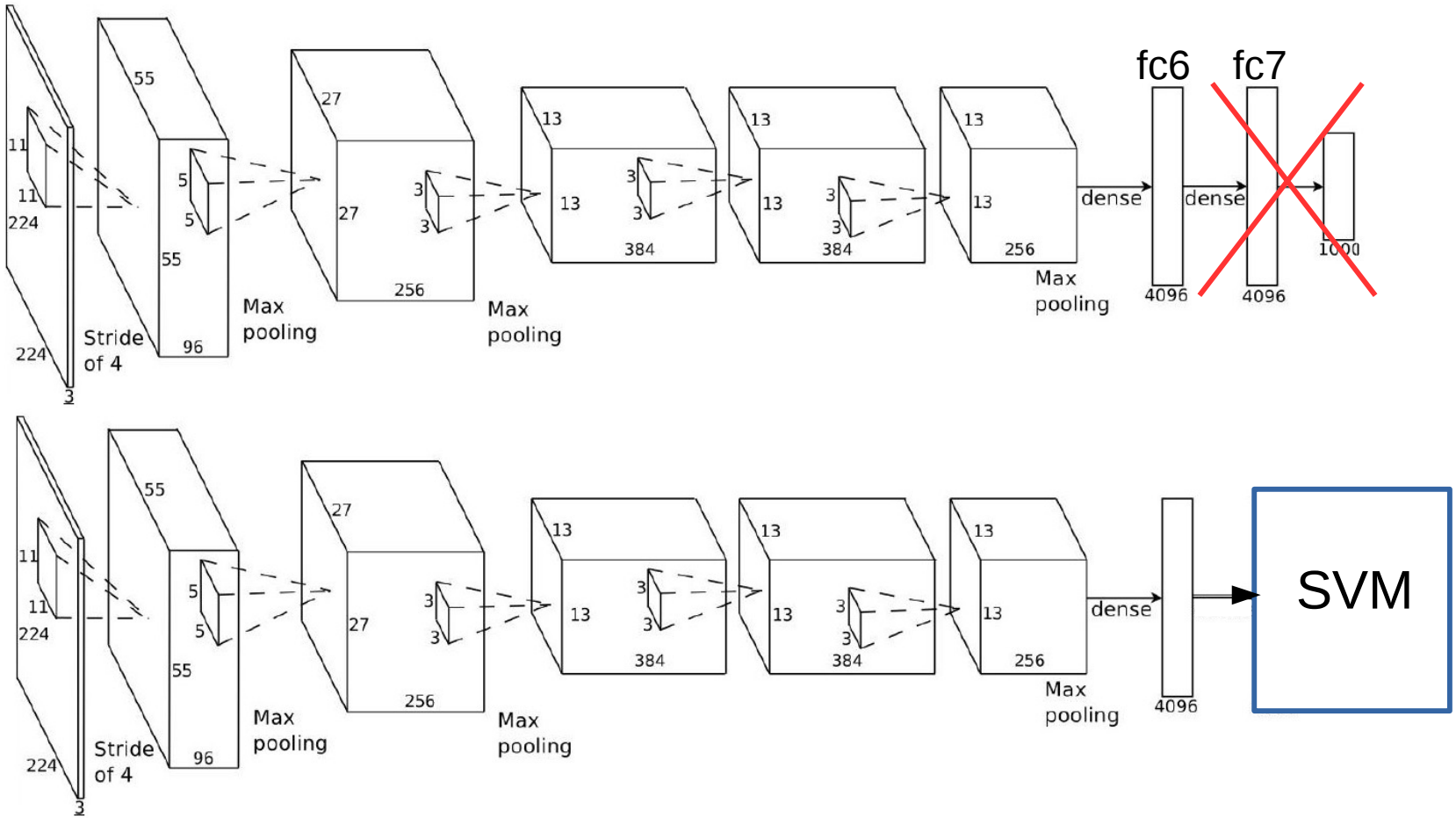
How can we apply transfer learning?

There are 2 common methods:

- Direct transfer learning: Directly transfer CNN's features from one scenario (source domain) to a new one (target domain).
- Finetunning: use weights from initial model (source domain) as a starting point to finetune the model with data from the new scenario (target domain).

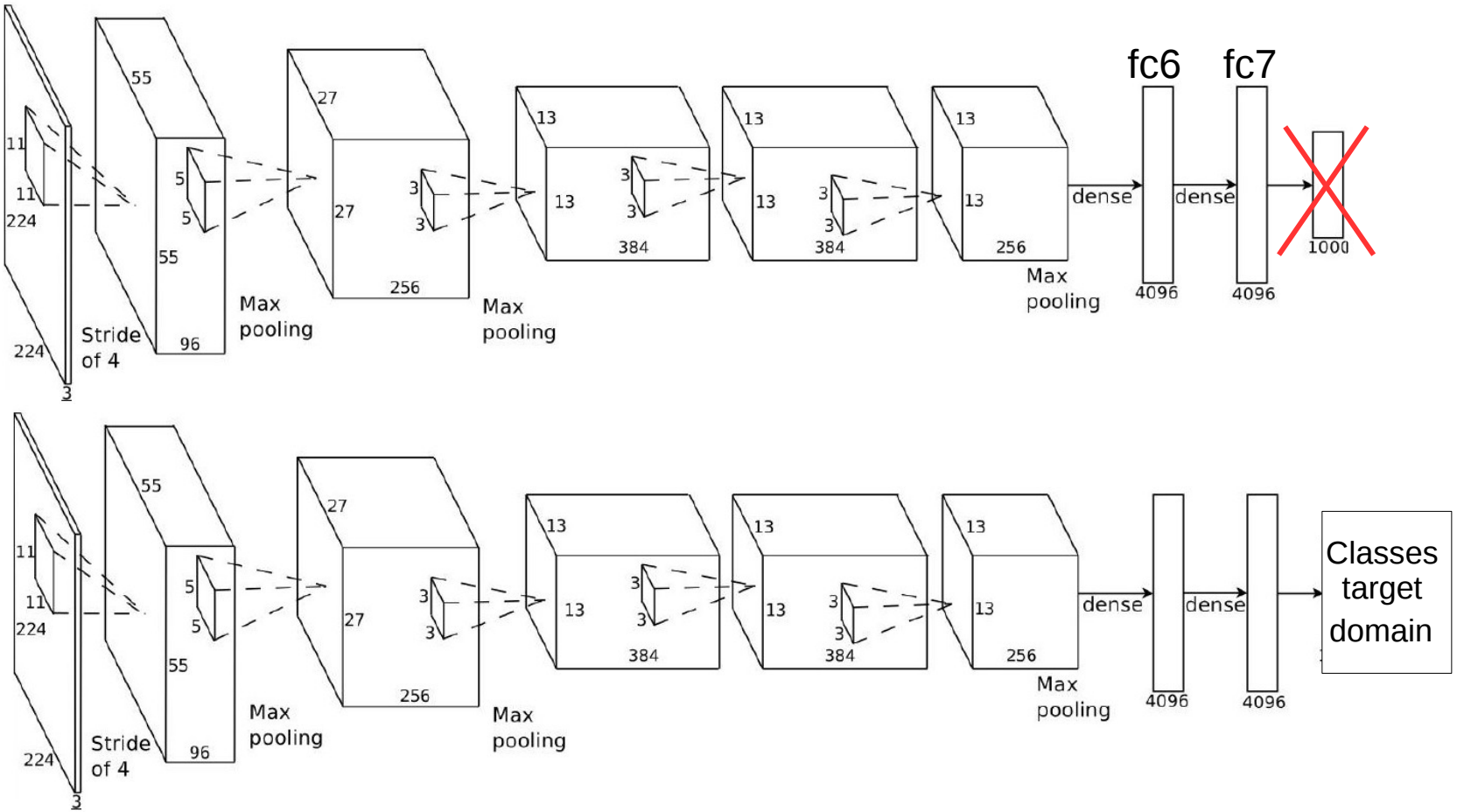
Direct transfer learning:

- Main Idea: Use embedding of last feature layer of a CNN pre-trained on source domain to train a classifier for the target domain.



Finetunning:

- Main Idea: Finetune weights of the pretrained network using data from target domain.

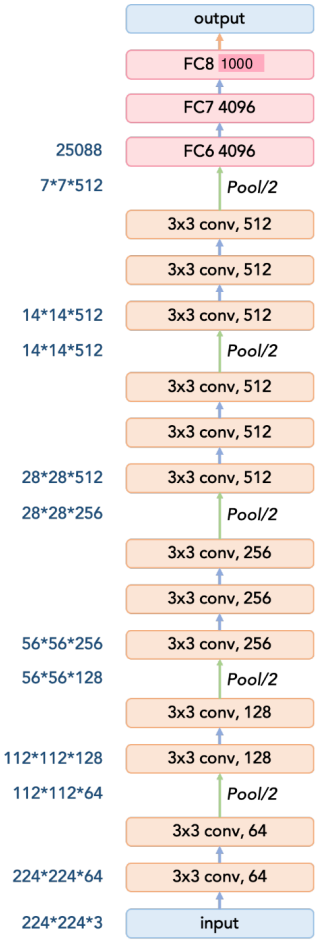


Finetunning:

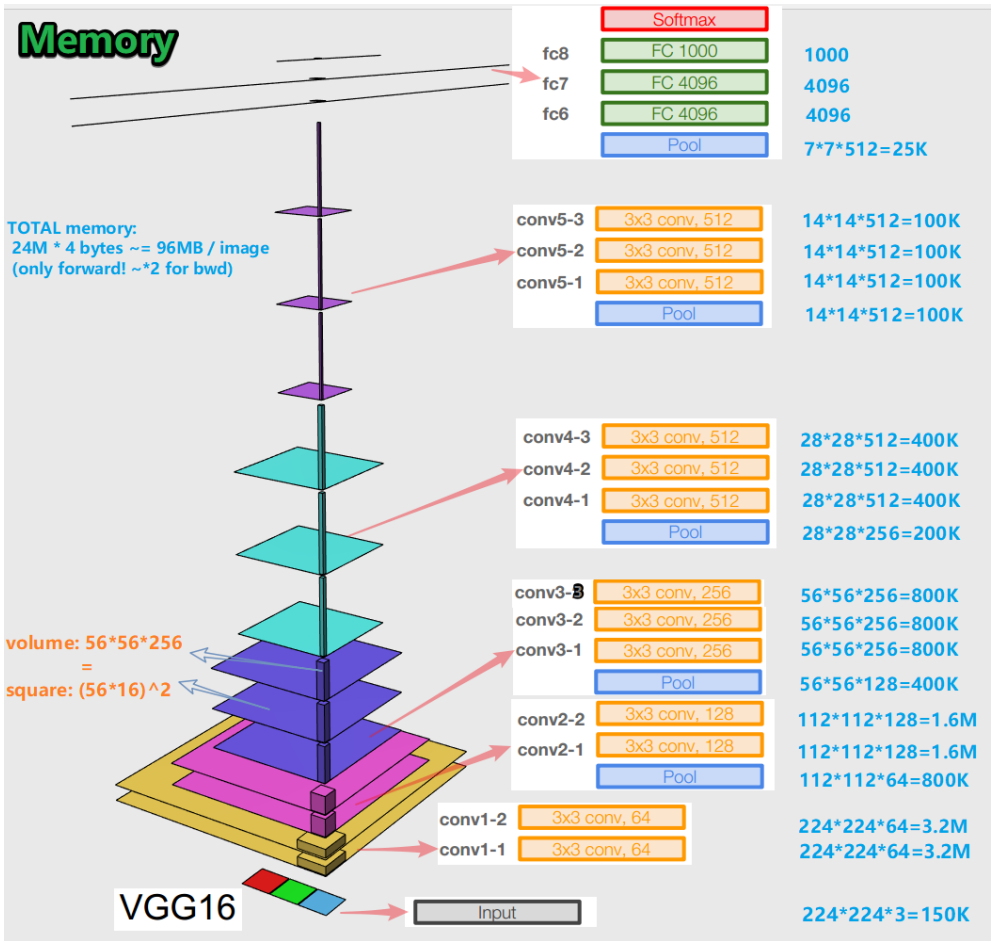
- Main Idea: Finetune weights of the pretrained network using data from target domain.
- Finetune weights on all layers or just in some portions of the network?. What layers should I finetune?
- Finetunning vs training from scratch, when?
- Similarity between source and target domains?.
- Some practical tips:
  - Use a smaller learning rate during finetunning, why?
  - Do not need to resize input data (ex. image resolution). Just need to change the convolution span, why?

Finetunning in Keras

Example: refine VGG-16 network pretrained on ImageNet to classify a new dataset.



(k, s)	(k, s)'	receptive field
7 x 1	404 x 32	404 x 404
2 x 2	212 x 32	212 x 212
3 x 1	196 x 16	196 x 196
3 x 1	164 x 16	164 x 164
3 x 1	132 x 16	132 x 132
2 x 2	100 x 16	100 x 100
3 x 1	82 x 8	92 x 92
3 x 1	76 x 8	76 x 76
3 x 1	60 x 8	60 x 60
2 x 2	44 x 8	44 x 44
3 x 1	40 x 4	40 x 40
3 x 1	32 x 4	32 x 32
3 x 1	24 x 4	24 x 24
2 x 2	16 x 4	16 x 16
3 x 1	14 x 2	14 x 14
3 x 1	10 x 2	10 x 10
2 x 2	6 x 2	6 x 6
3 x 1	5 x 1	5 x 5
3 x 1	3 x 1	3 x 3



Finetunning in Keras

```
# load the VGG16 network, ensuring the head FC layer sets are left off
baseModel = VGG16(weights="imagenet", include_top=False,input_tensor=Input(shape=(224, 224, 3)))

# construct the head of the model that will be placed on top of the base model
headModel = baseModel.output
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(512, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(len(config.CLASSES), activation="softmax")(headModel)

# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)
```