TeIAS | Tehran Institute for Advanced Studies

بنام خداوند جان و خرد

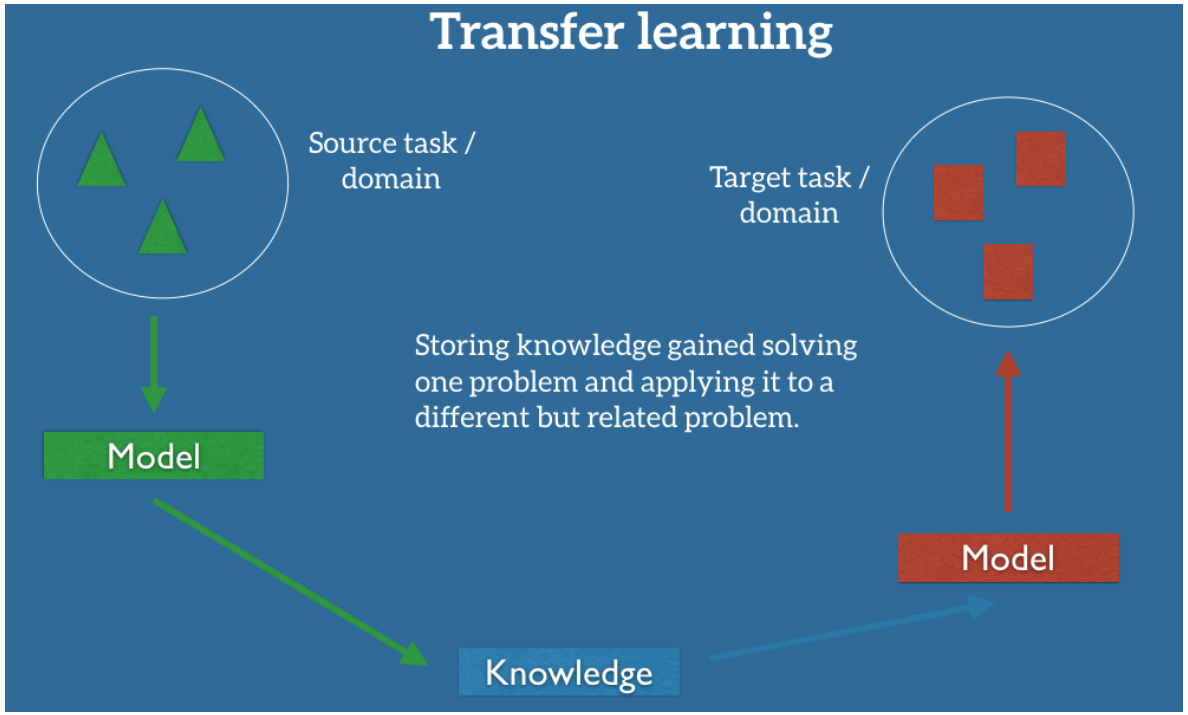
Transfer Learning

Mohammad Taher Pilehvar

Deep Learning
https://teias-courses.github.io/ml01/

Image source

Transfer Learning



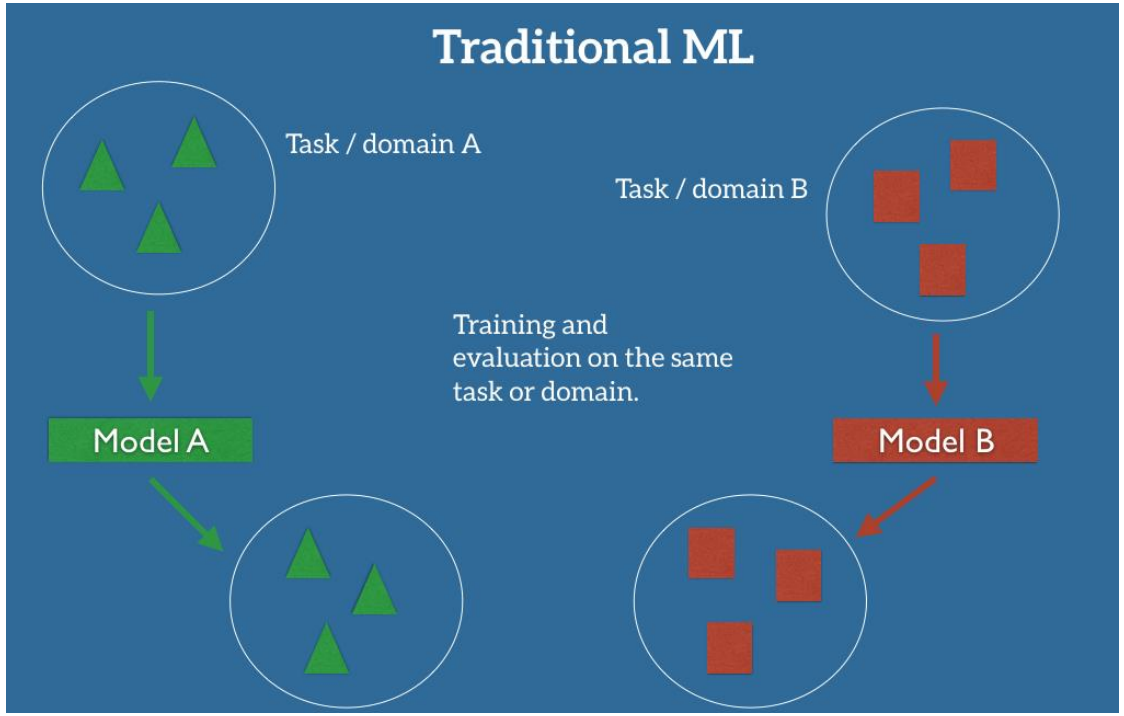
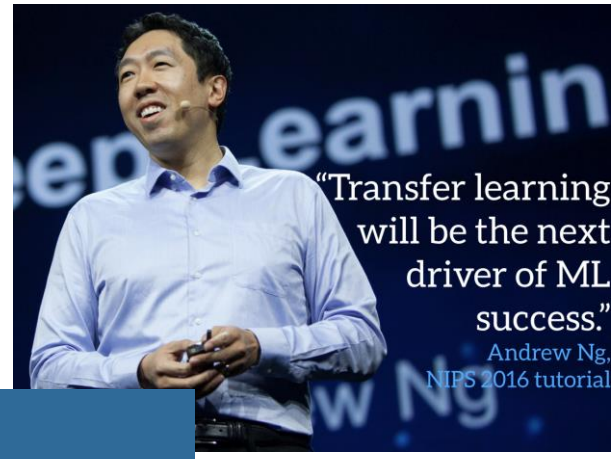


Image from Sebastian Ruder

Transfer Learning



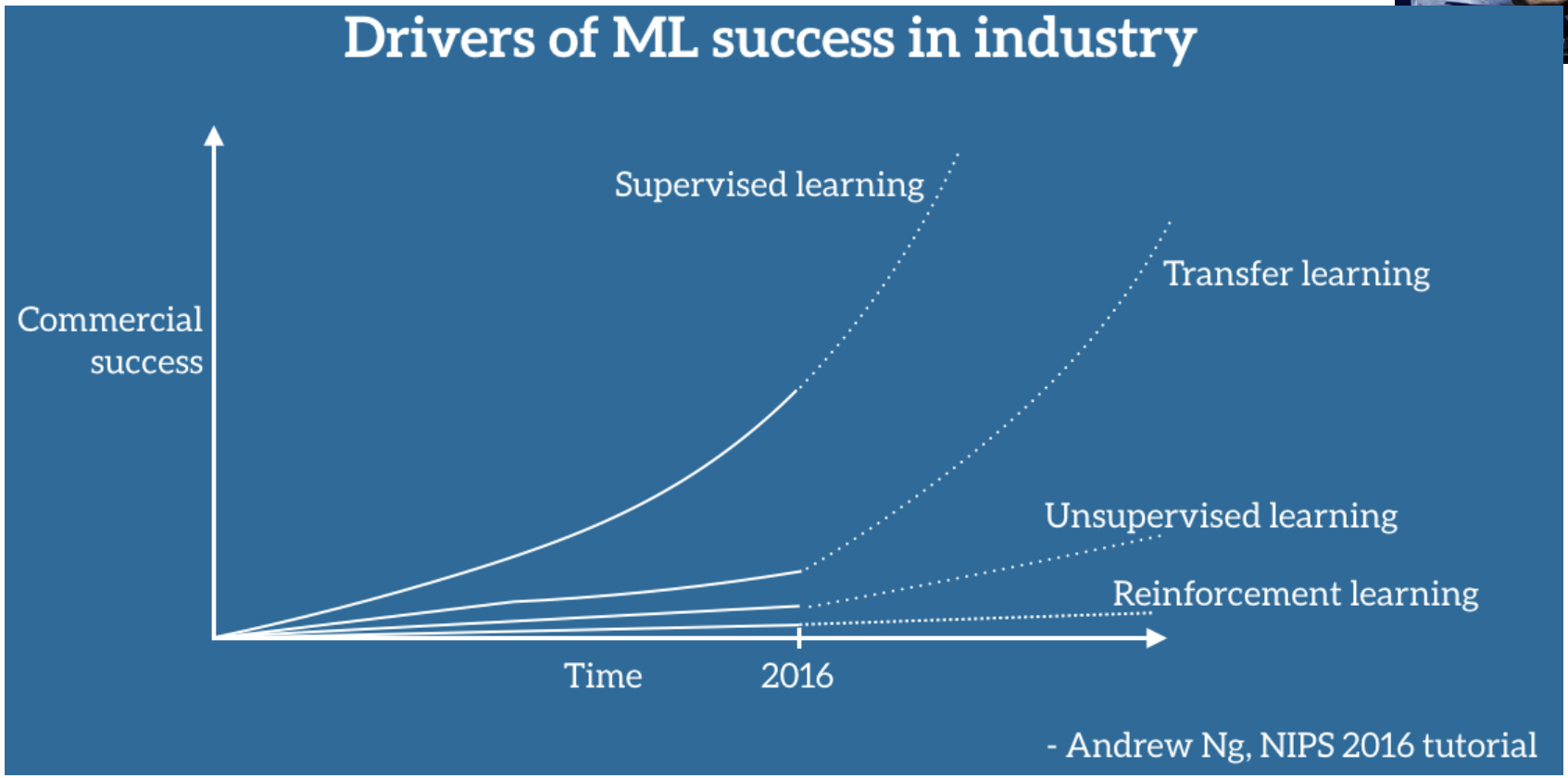


Image from Sebastian Ruder

Transfer Learning

Image from Sebastian Ruder

Transfer Learning

- A *pretrained network* is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task.

- If this original dataset is large enough and general enough, then the spatial hierarchy of features learned by the pretrained network can effectively act as a generic model of the visual world
  - The learned features can prove useful for many different computer vision problems, even though these new problems may involve completely different classes than those of the original task.

Transfer Learning

- For instance:
  - Train a network on ImageNet (where classes are mostly animals and everyday objects)
  - Then repurpose this trained network for something as remote as identifying furniture items in images.

- Such portability of learned features across different problems is a key advantage of deep learning compared to many older, shallow-learning approaches, and it makes deep learning very effective for small-data problems.

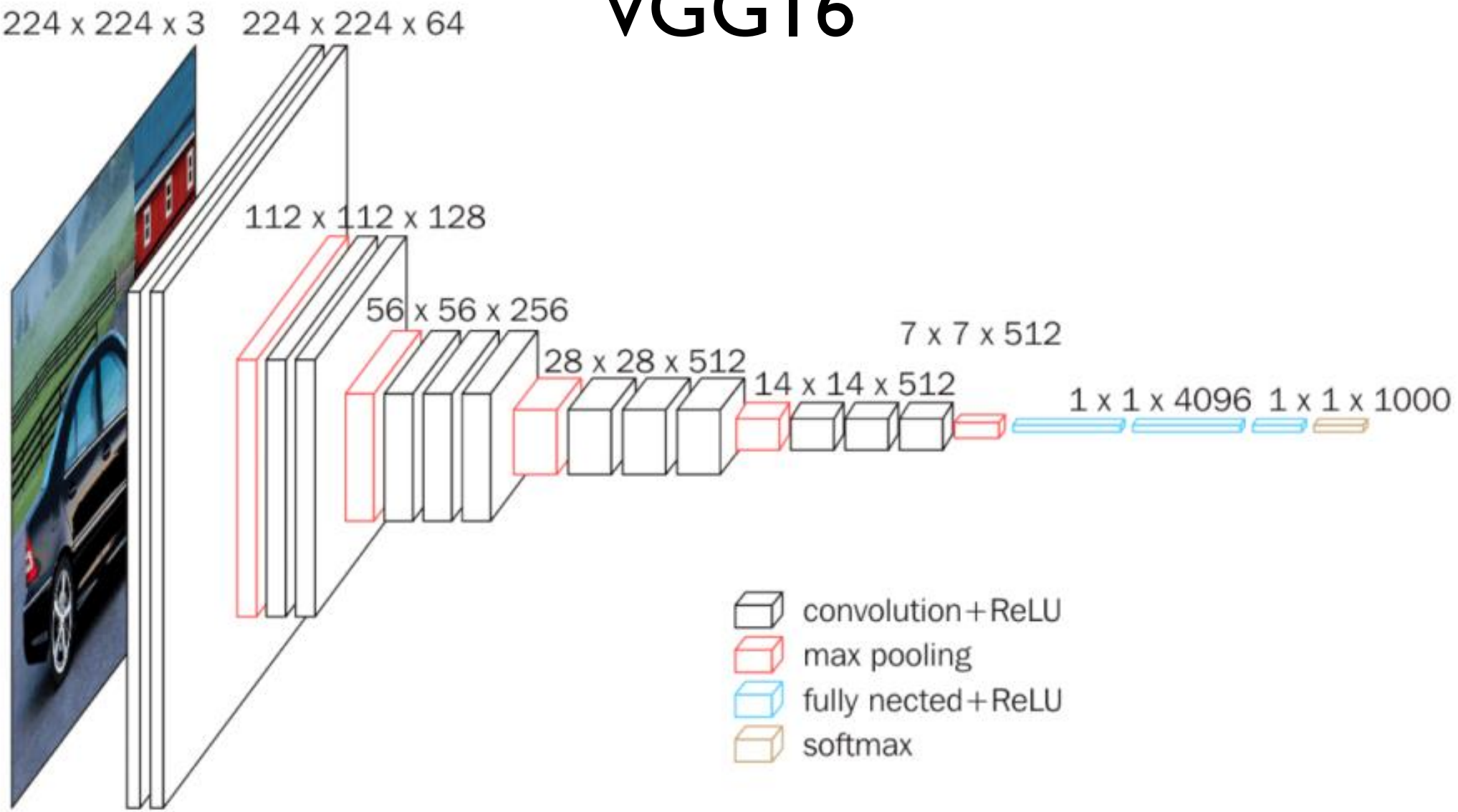
Transfer Learning

- Let's consider a large convnet trained on the ImageNet dataset (1.4 million labelled images and 1,000 different classes)
  - You can thus expect to perform well on the dogs-versus-cats classification problem.

- VGG16 architecture, developed by Karen Simonyan and Andrew Zisserman in 2014
  - a simple and widely used convnet architecture for ImageNet

- Many cutesy models these days:
  - VGG, ResNet, Inception, Inception-ResNet, Xception

VGG16


224 x 224 x 3
224 x 224 x 64
112 x 112 x 128
56 x 56 x 256
28 x 28 x 512
14 x 14 x 512
7 x 7 x 512
1 x 1 x 4096
1 x 1 x 1000
convolution+ReLU
max pooling
fully nected+ReLU
softmax

Transfer Learning

There are two ways to use a pre-trained network:

1. Feature extraction

2. Fine tuning

Feature Extraction

- Using the representations learned by a previous network to extract interesting features from new samples.

- These features are then run through a new classifier, which is trained from scratch.

Feature Extraction

ConvNets for image classification usually comprise two parts:

1. A series of pooling and convolution layers – *convolutional base*
2. A densely connected classifier

Feature Extraction

In the case of ConvNets, feature extraction consists of taking the convolutional base of a previously trained network, running the new data through it, and training a new classifier on top of the output.

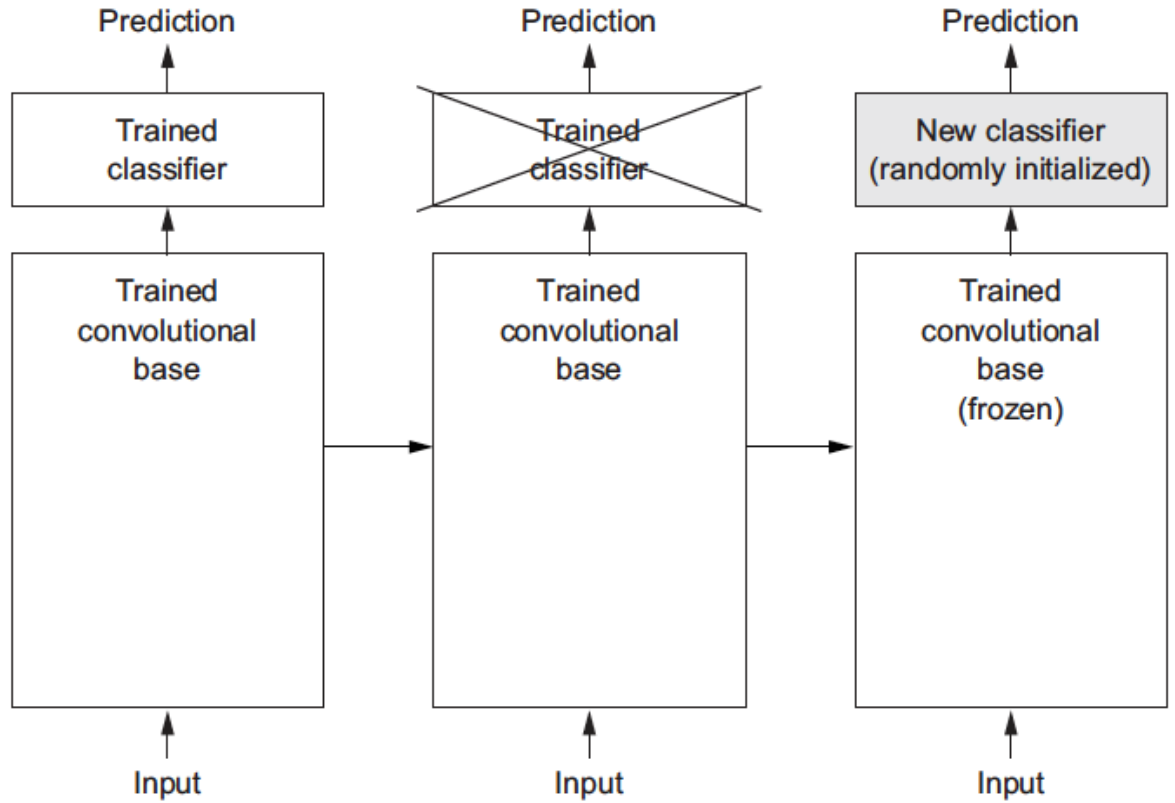


Figure 5.14 Swapping classifiers while keeping the same convolutional base

Feature Extraction

Why only reuse the convolutional base?

Could we reuse the densely connected classifier as well?

- The feature maps of a convnet are presence maps of generic concepts over a picture, which is likely to be useful regardless of the computer-vision problem at hand.

- The representations learned by the dense classifier will be specific to the set of classes on which the model was trained—they will only contain information about the presence probability of this or that class in the entire picture.

Feature Extraction

Level of generality (and therefore reusability) of the representations in ConvNets depends on the depth of the layer in the model.

- Earlier layers extract local, highly generic feature maps (such as visual edges, colors, and textures)
- Later extract more-abstract concepts (such as “cat ear” or “dog eye”).

If the test dataset is substantially different, it is better to use only the first few layers of the model to do feature extraction, rather than using the entire convolutional base.

Feature Extraction

Instantiate the VGG16 model

```
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet`,
                  include_top=False,
                  input_shape=(150, 150, 3))
```

Feature Extraction

```
VGG16(weights='imagenet`, include_top=False, input_shape=(150, 150, 3))
```

- `weights` specifies the weight checkpoint from which to initialize the model.
- `include_top` refers to including (or not) the densely connected classifier on top of the network. By default, this densely connected classifier corresponds to the 1,000 classes from ImageNet.
- `input_shape` is the shape of the image tensors that you'll feed to the network.

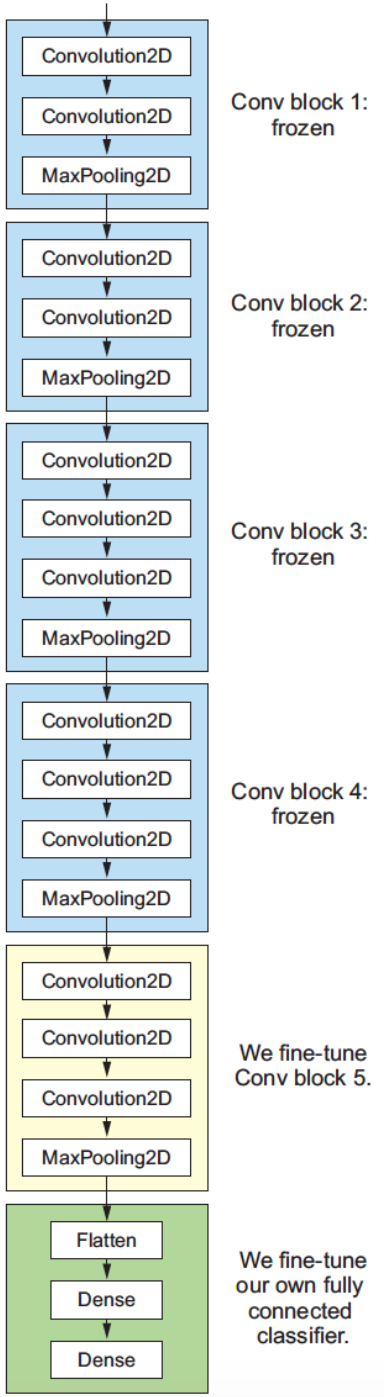
Feature Extraction

```
>>> conv_base.summary()

Layer (type) Output Shape Param #
=================================================================
input_1 (InputLayer) (None, 150, 150, 3) 0
block1_conv1 (Convolution2D) (None, 150, 150, 64) 1792
block1_conv2 (Convolution2D) (None, 150, 150, 64) 36928
block1_pool (MaxPooling2D) (None, 75, 75, 64) 0

block2_conv1 (Convolution2D) (None, 75, 75, 128) 73856
block2_conv2 (Convolution2D) (None, 75, 75, 128) 147584
block2_pool (MaxPooling2D) (None, 37, 37, 128) 0

block3_conv1 (Convolution2D) (None, 37, 37, 256) 295168
block3_conv2 (Convolution2D) (None, 37, 37, 256) 590080
block3_conv3 (Convolution2D) (None, 37, 37, 256) 590080
block3_pool (MaxPooling2D) (None, 18, 18, 256) 0

block4_conv1 (Convolution2D) (None, 18, 18, 512) 1180160
block4_conv2 (Convolution2D) (None, 18, 18, 512) 2359808
block4_conv3 (Convolution2D) (None, 18, 18, 512) 2359808
block4_pool (MaxPooling2D) (None, 9, 9, 512) 0

block5_conv1 (Convolution2D) (None, 9, 9, 512) 2359808
block5_conv2 (Convolution2D) (None, 9, 9, 512) 2359808
block5_conv3 (Convolution2D) (None, 9, 9, 512) 2359808
block5_pool (MaxPooling2D) (None, 4, 4, 512) 0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

Fine Tuning

- Another widely used technique for model reuse

- Unfreeze a few of the top layers of a frozen model base used for feature extraction, and jointly train both the newly added part of the model (in this case, the fully connected classifier) and these top layers.



Fine Tuning

- It is only possible to fine-tune the top layers of the convolutional base once the classifier on top has already been trained.

- If the classifier isn't already trained, then the error signal propagating through the network during training will be too large, and the representations previously learned by the layers being fine-tuned will be destroyed.

Fine Tuning

- Thus the steps for fine-tuning a network are as follow:
  1. Add your custom network on top of an already-trained base network.
  2. Freeze the base network.
  3. Train the part you added.
  4. Unfreeze some layers in the base network.
  5. Jointly train both these layers and the part you added.

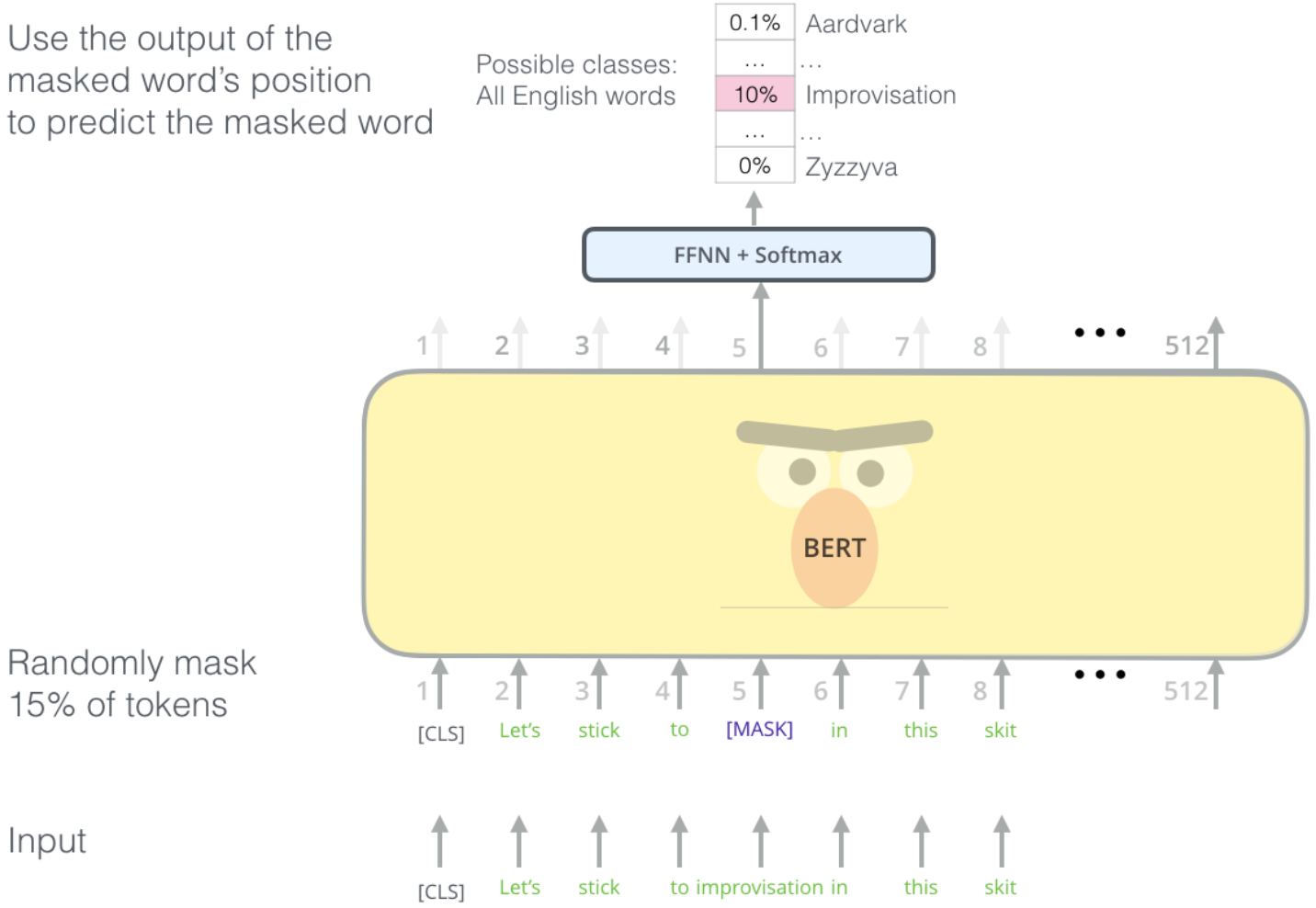
Fine Tuning

Why not fine-tune more layers? Why not fine-tune the entire convolutional base?

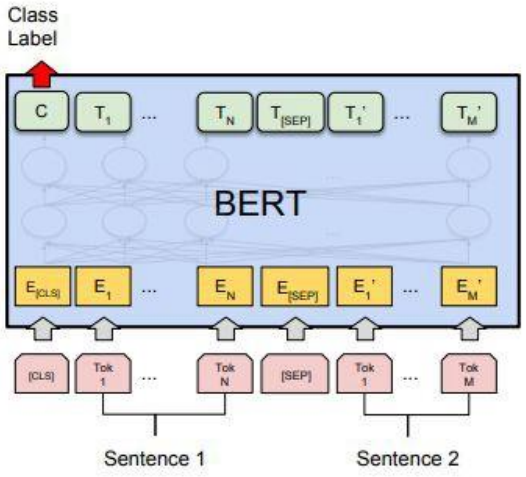
You could. But you need to consider the following:

- Earlier layers in the convolutional base encode more-generic, reusable features, whereas layers higher up encode more-specialized features.
- It's more useful to fine-tune the more specialized features, because these are the ones that need to be repurposed on your new problem.
- The more parameters you're training, the more you're at risk of overfitting. The convolutional base has 15 million parameters, so it would be risky to attempt to train it on your small dataset.

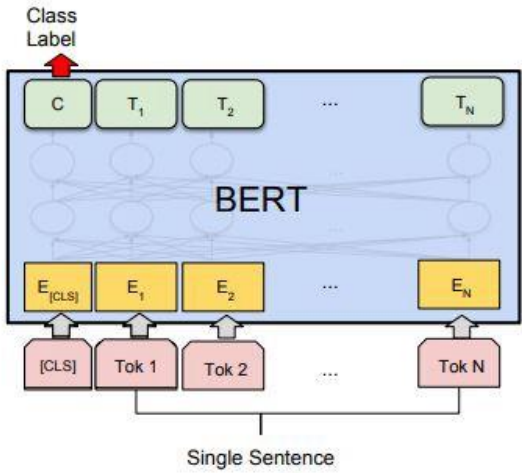
Fine Tuning: Language Models (BERT)



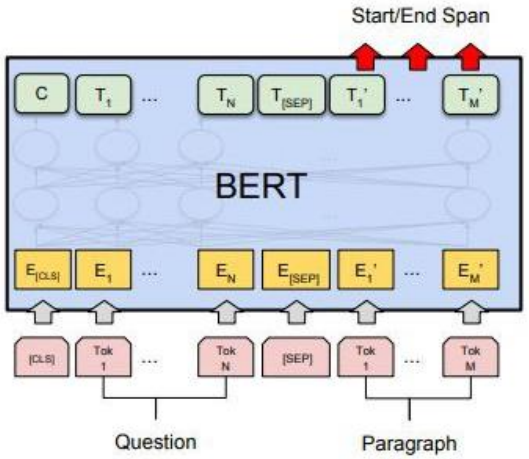
Fine Tuning: Language Models (BERT)



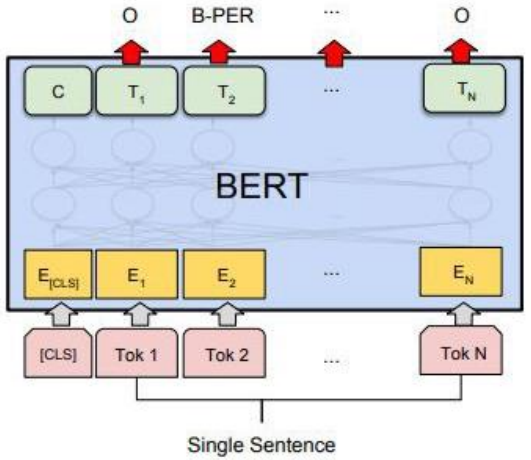
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA



(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Fine Tuning: Language Models (GPT)

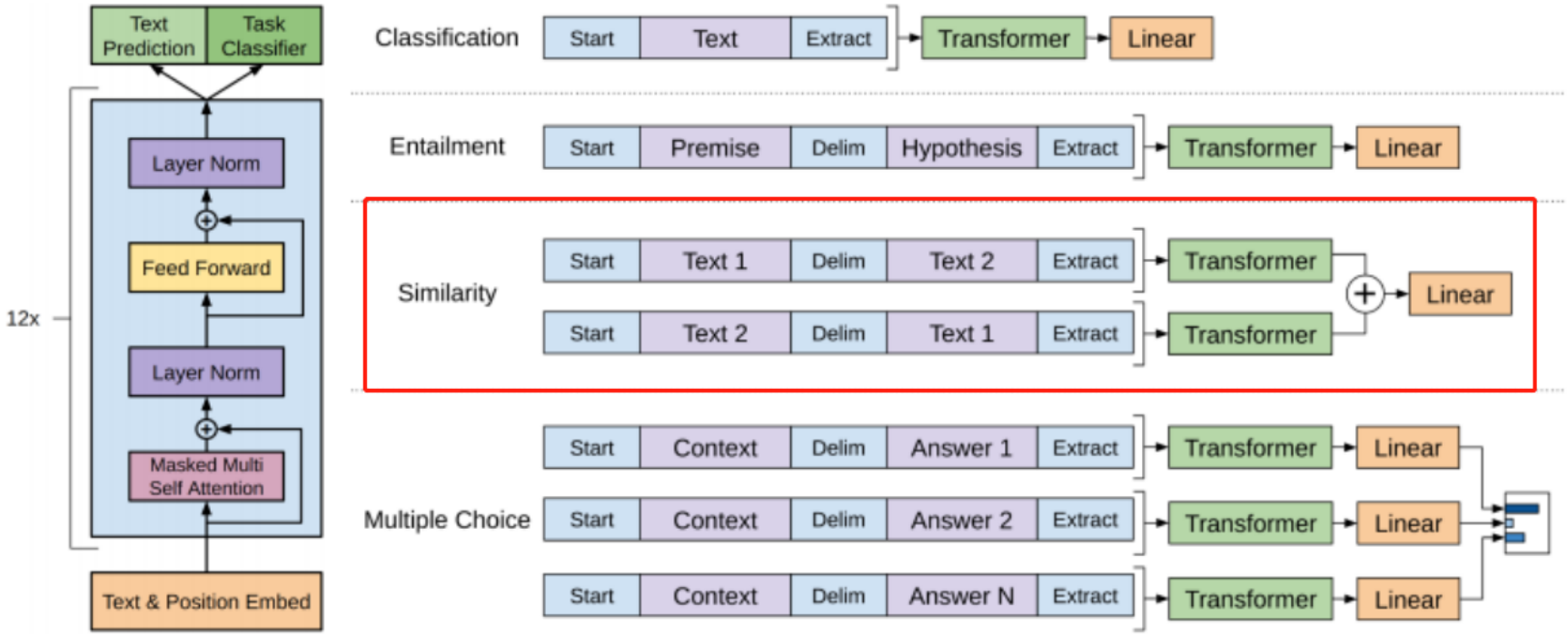
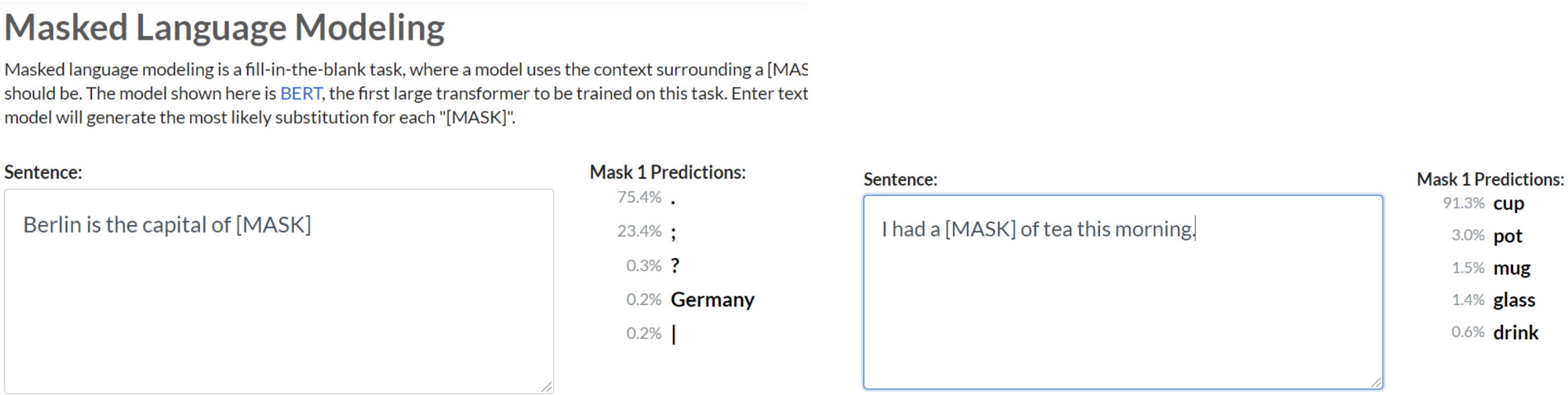


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

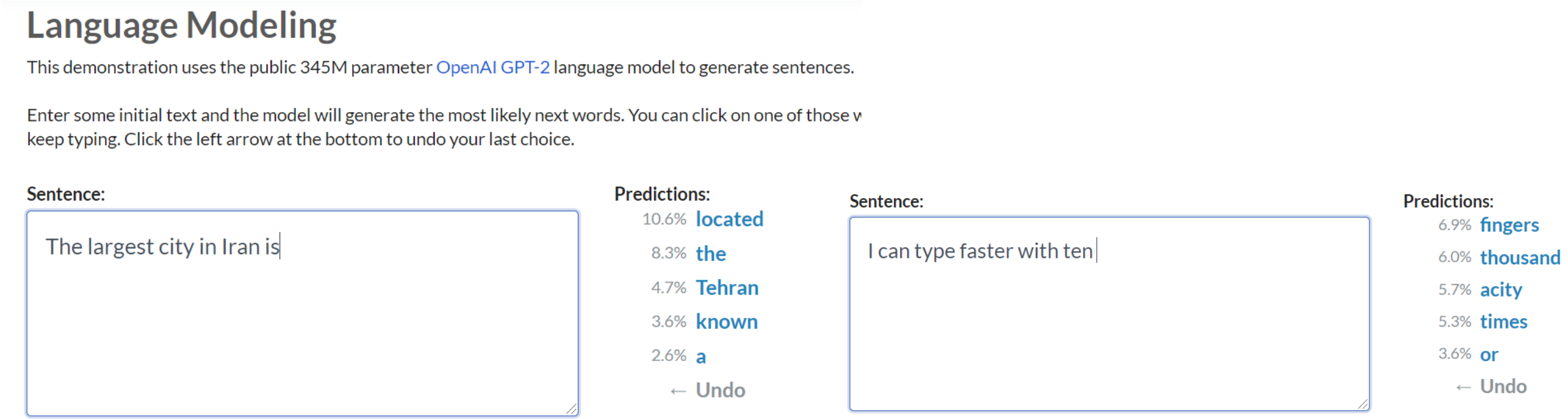
Fine Tuning: Language Models (BERT)

- Allen AI LM demo (demo.allennlp.org)



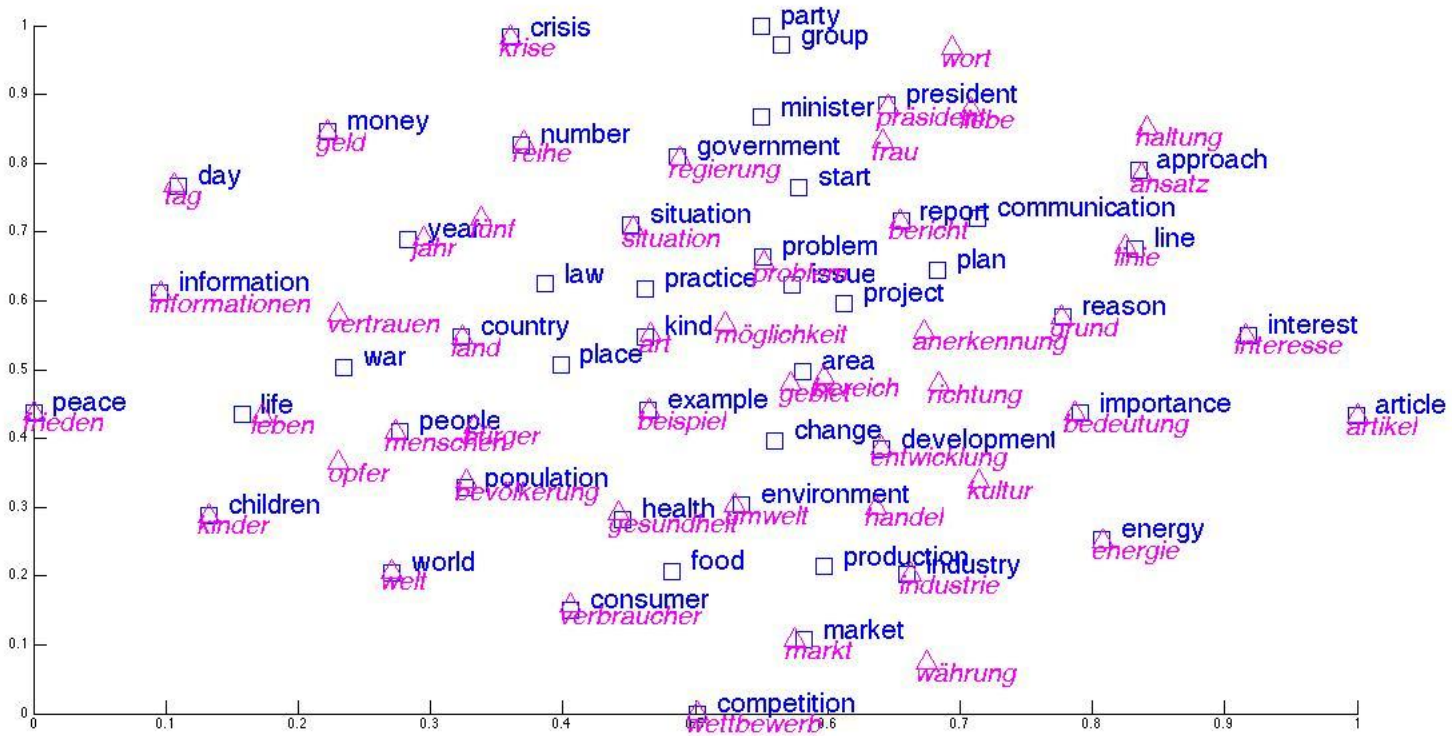
Fine Tuning: Language Models (BERT)

- Allen AI LM demo (demo.allennlp.org)



Transfer learning: Applications

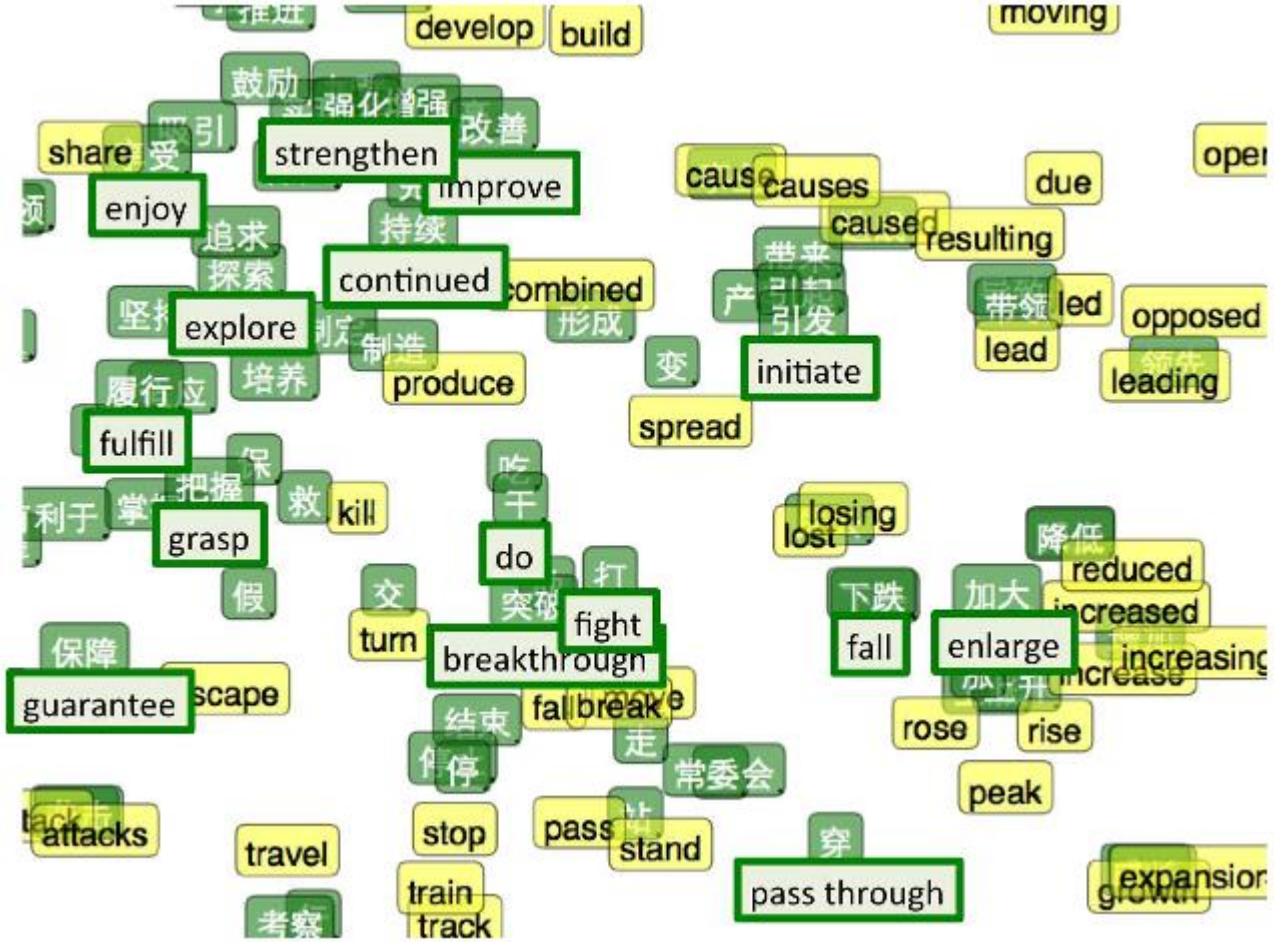
Across languages



https://ruder.io/cross-lingual-embeddings/

Transfer learning: Applications

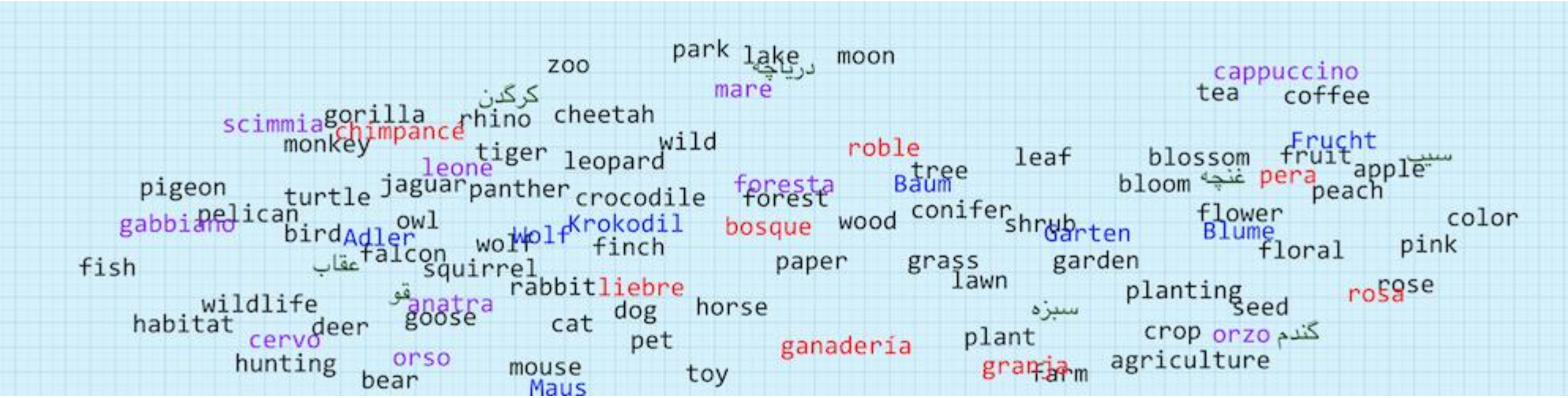
Across languages



https://ruder.io/cross-lingual-embeddings/

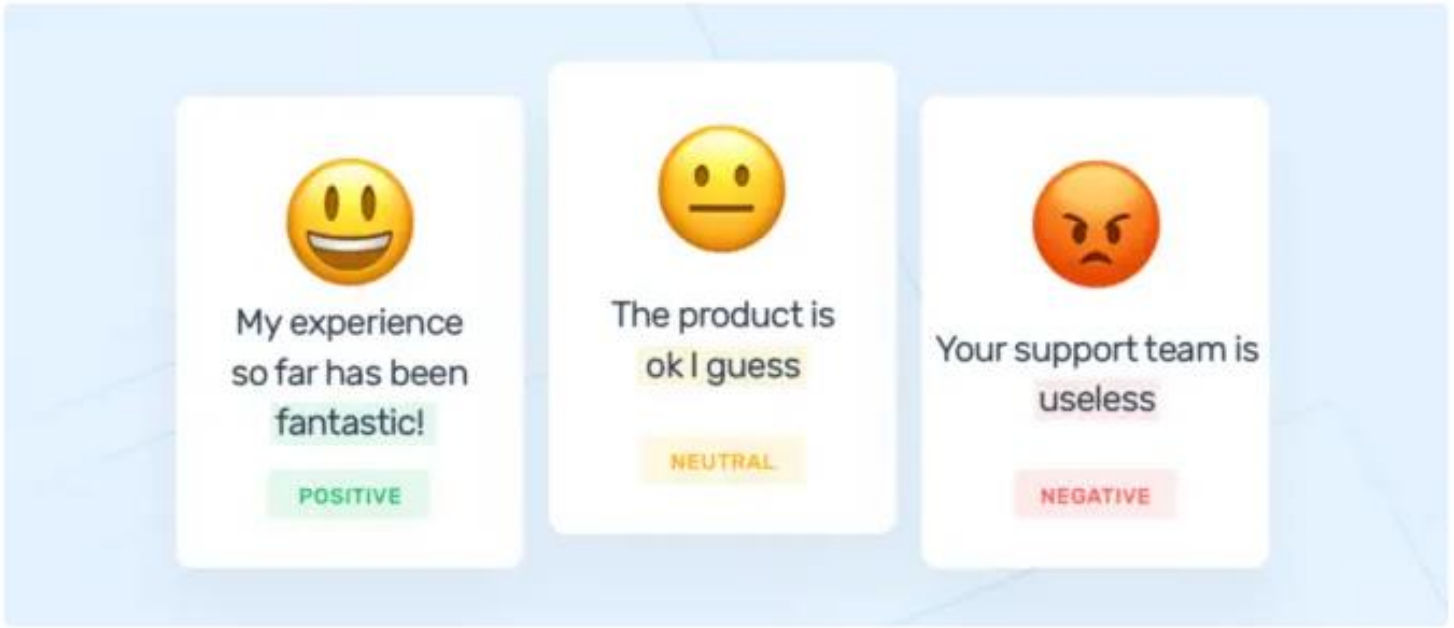
Transfer learning: Applications

Across languages



Transfer learning: Applications

Zero-shot cross-lingual transfer



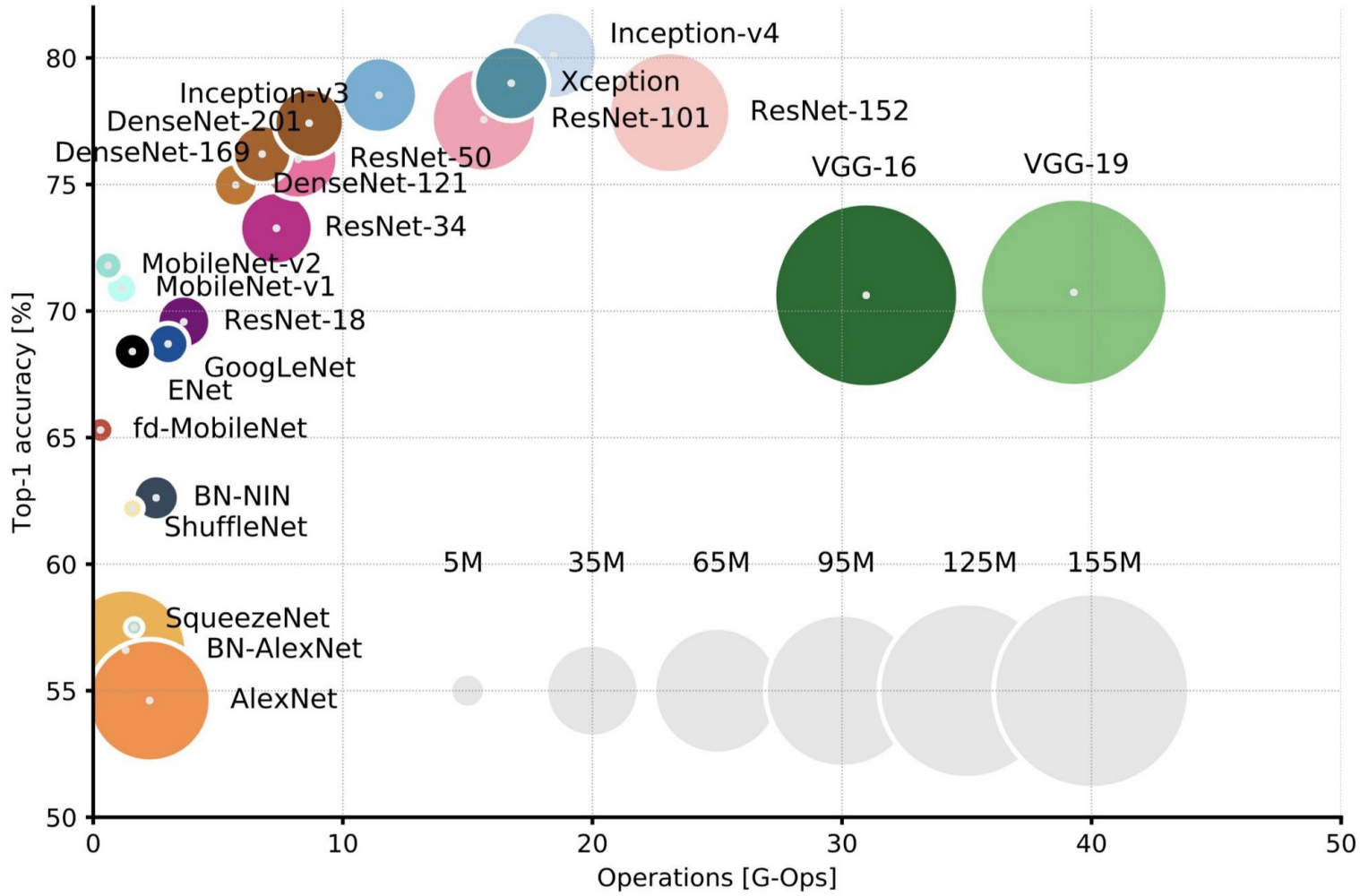
؟

از خدمات دیجیکالا راضی هستم.

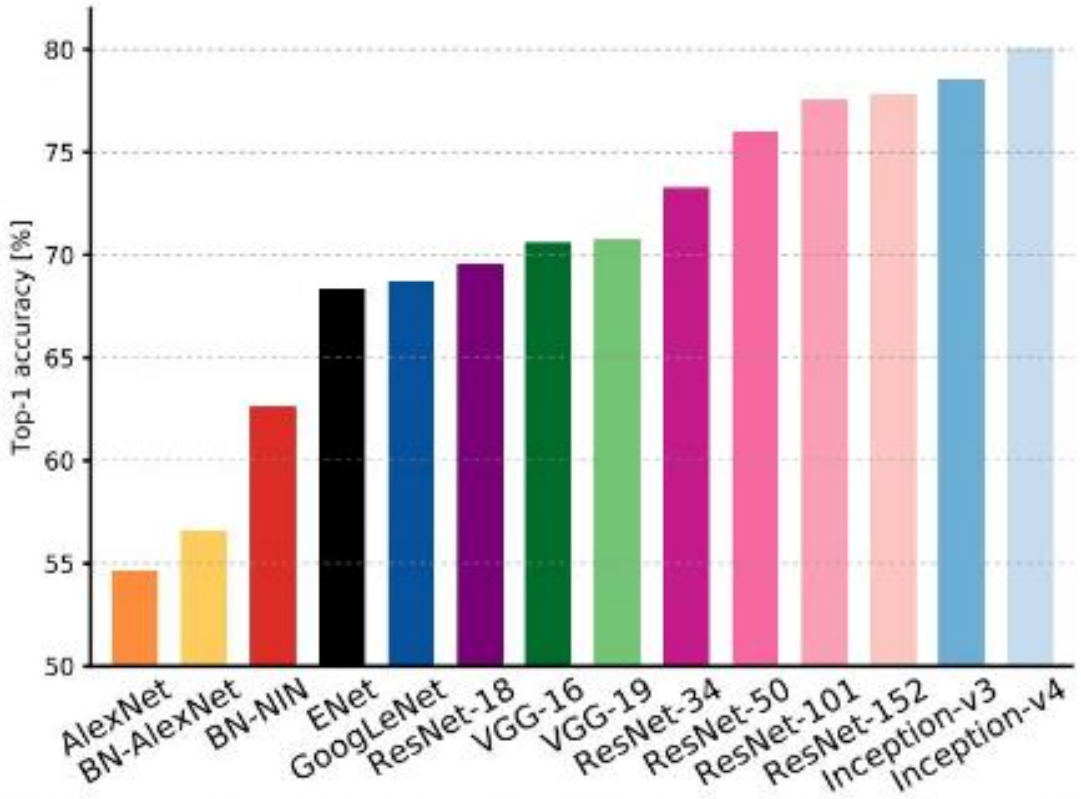
Image source MonkeyLearn

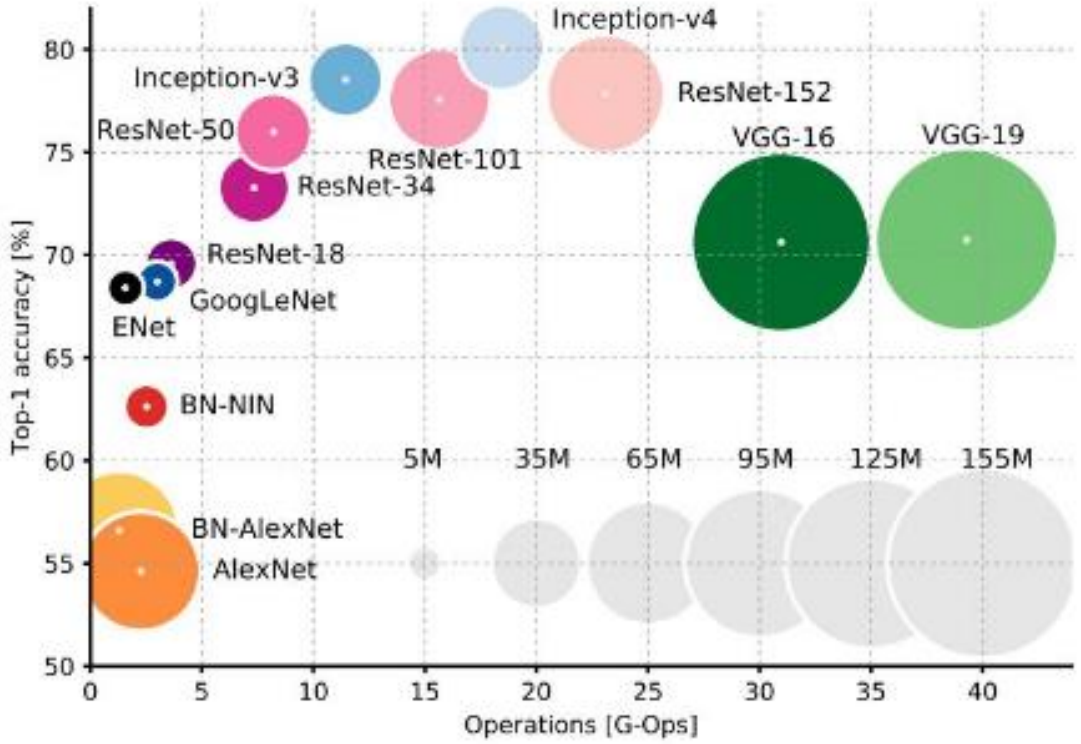
Pre-trained image classification models

- Xception
- Inception V1-V4
- ResNet50
- VGG16
- VGG19
- MobileNet



Pre-trained image classification models



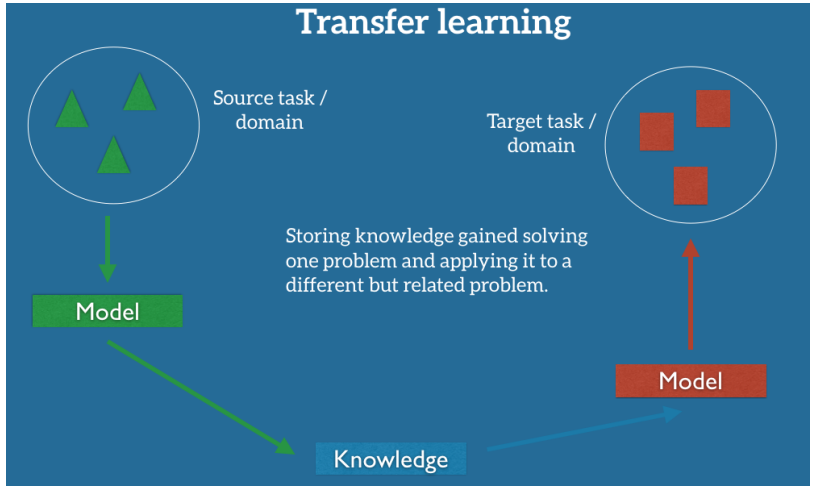


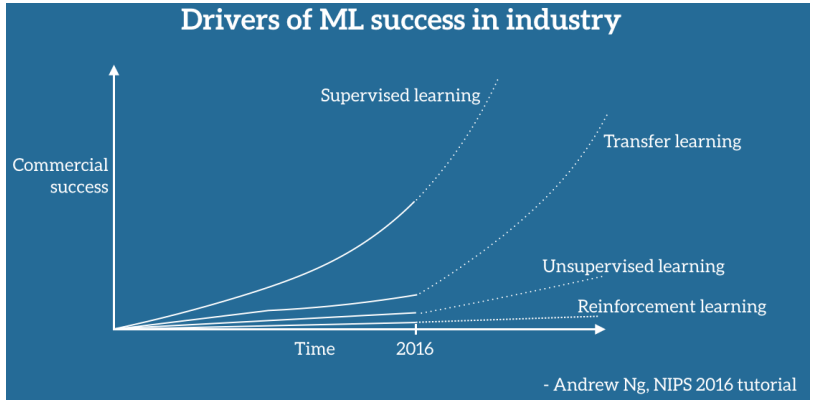
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Transfer Learning: Read more

http://ruder.io/transfer-learning/

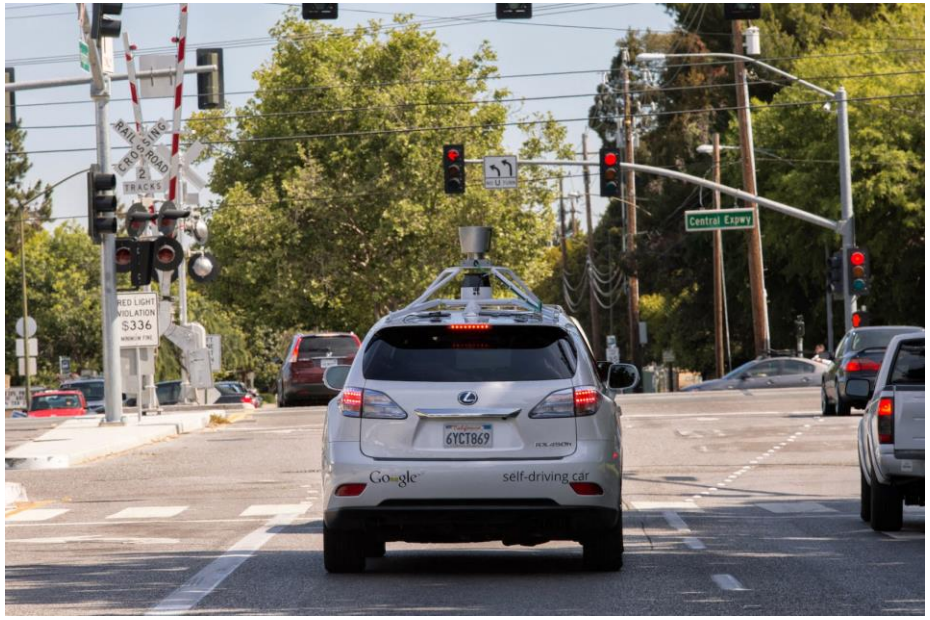
- What is Transfer Learning?
- Why Transfer Learning Now?
- A Definition of Transfer Learning
- Transfer Learning Scenarios
- Applications of Transfer Learning
- Transfer Learning Methods
- Related Research Areas
  - Semi-supervised learning
  - Using available data more effectively
  - Improving models' ability to generalize
  - Making models more robust
  - Multi-task learning
  - Continuous learning
  - Zero-shot learning
- Conclusion





Transfer learning: Applications

Learning from simulations



A Google self-driving car
(source: Google Research blog)

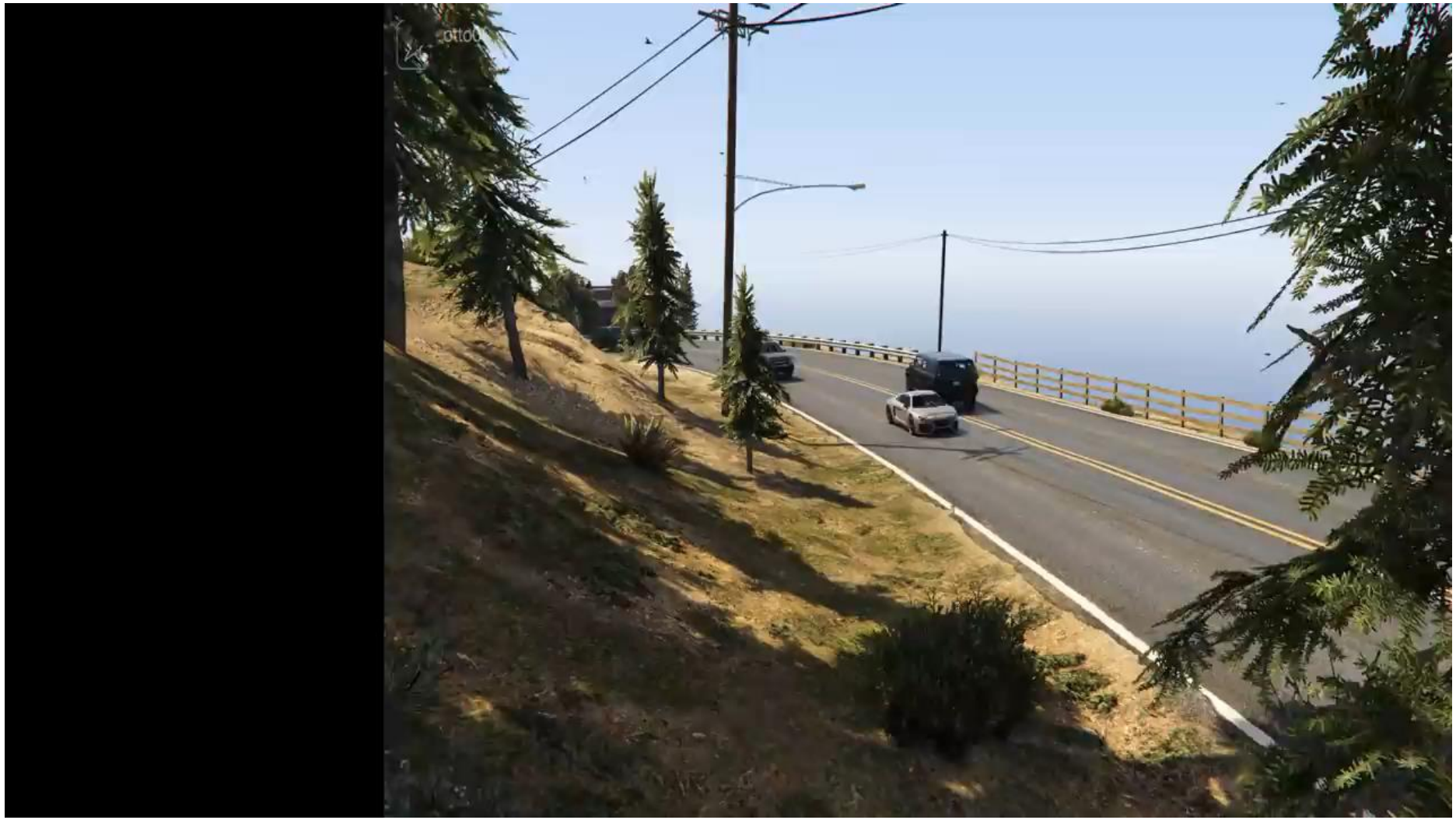
Udacity's self-driving car simulator
(source: TechCrunch)

Transfer learning: Applications

Udacity simulator

Transfer learning: Applications

OpenAI & GTA



Transfer learning: Applications

Learning from simulations

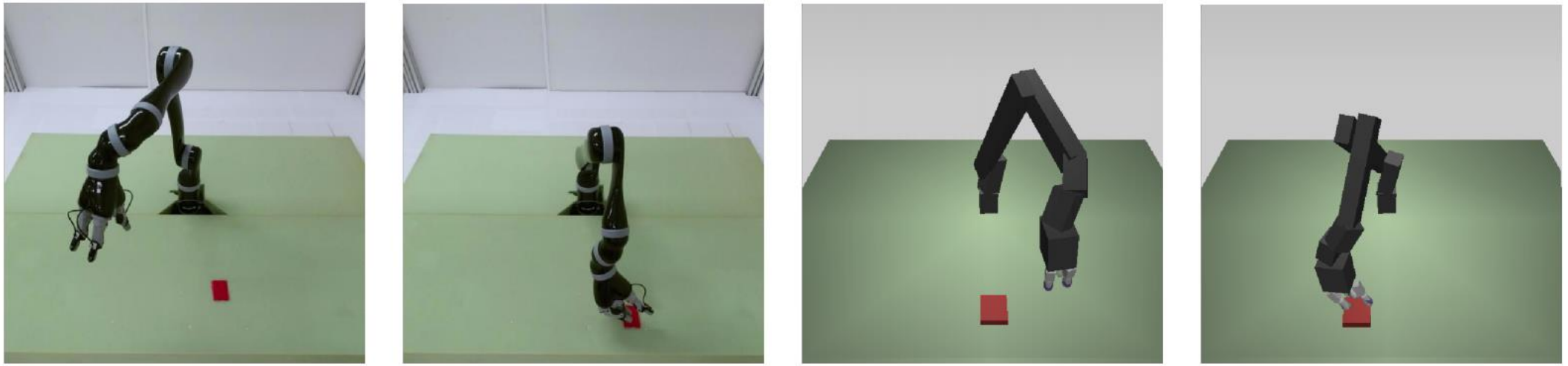


Figure 8: Robot and simulation images (Rusu et al., 2016)

Transfer learning: Applications

Adapting to new domains

Different visual domains (Sun et al., 2016)

Transfer learning: Applications

Adapting to new domains

Transfer learning: Applications

Adapting to new domains