Advanced Section 1
Transfer Learning

Marios Mattheakis

CS109B Data Science 2

Pavlos Protopapas, Mark Glickman and Chris Tanner

Deep learning & Nature

Very often in deep learning we are inspired by the Nature

Example: The convolution Networks were inspired by the neurons in the visual cortex of animals

Consider a scenario that there is someone who knows how to ride a bike and someone else does not know.
They both now want to learn how to drive a motorbike.

Does the former have any advantage in the learning task?
Why?

Outline

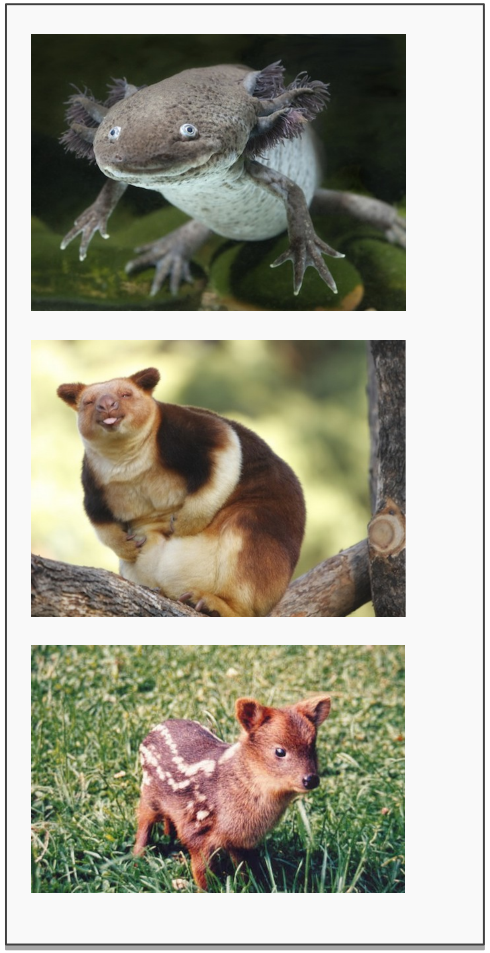
Motivation for Transfer Learning

The Basic idea for Transfer Learning

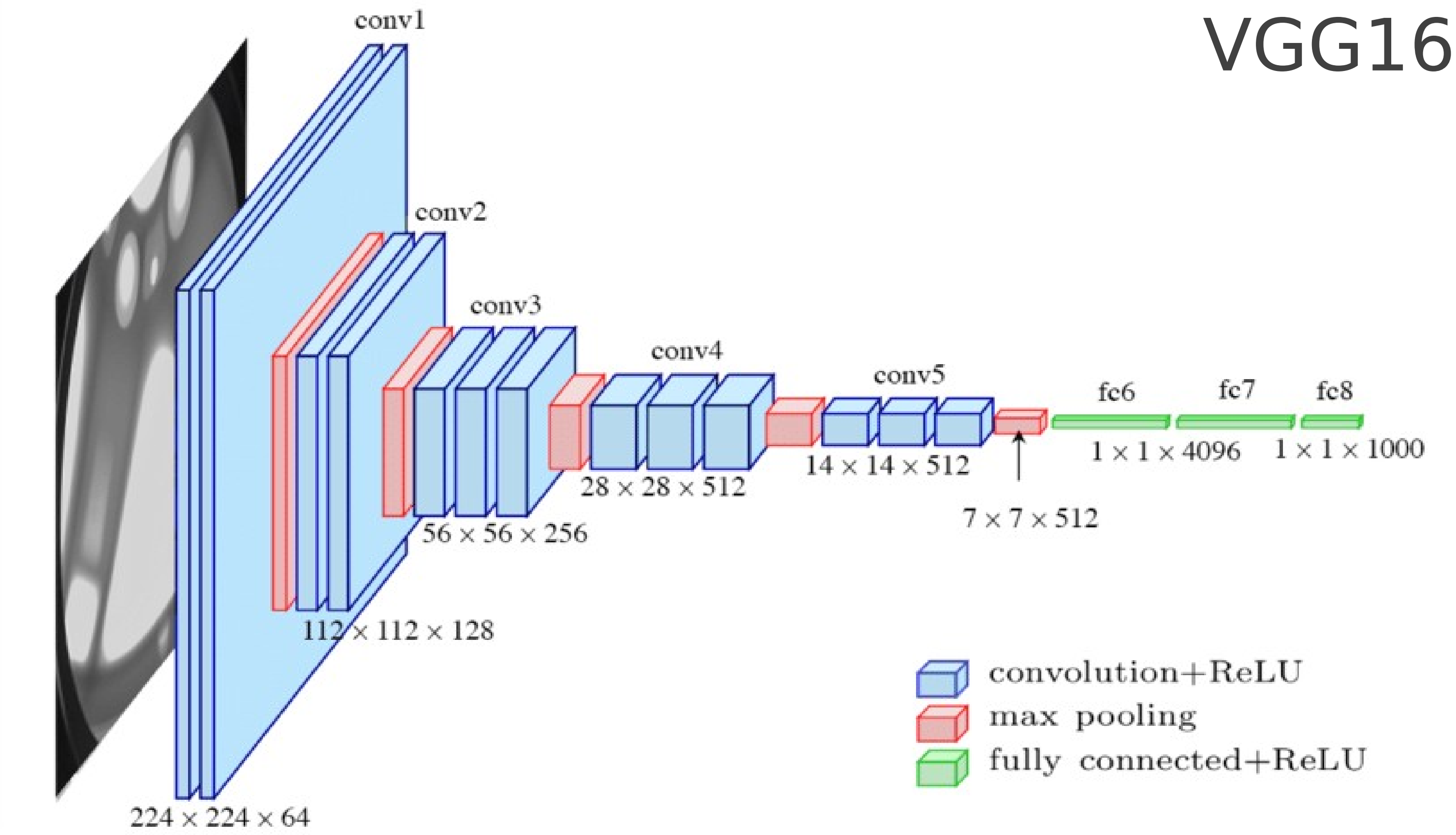
MobileNet. A light weight model

Some Coding

Classify Rarest Animals

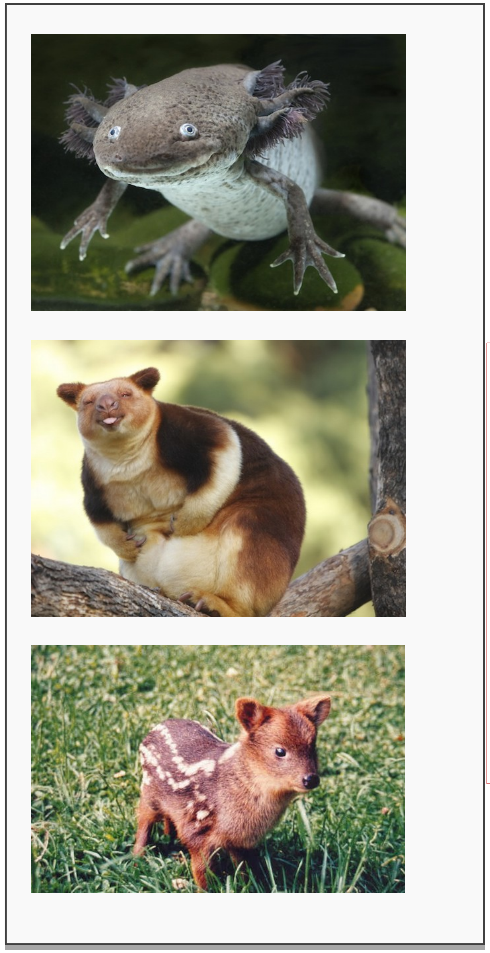


VGG16

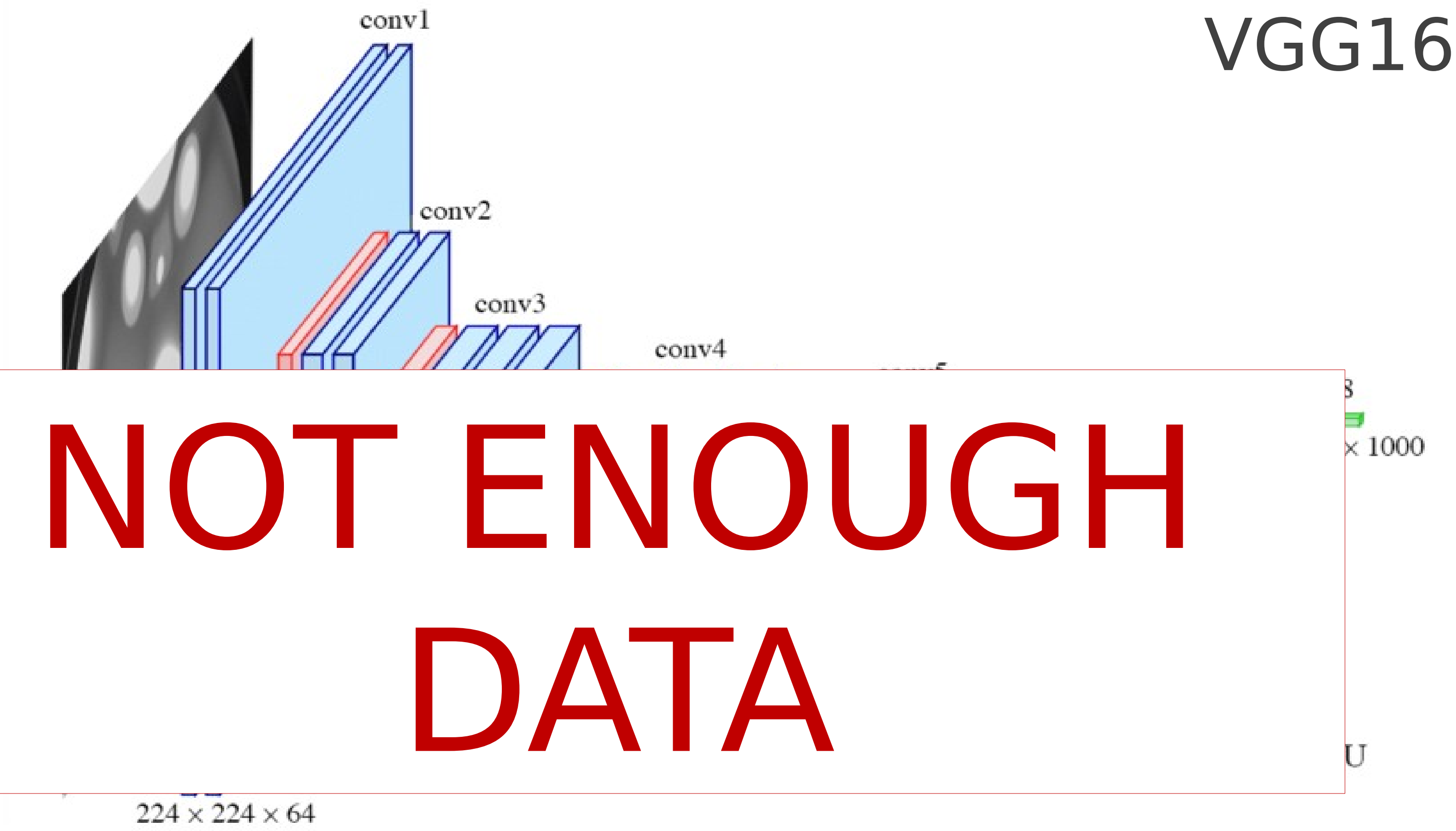


Number of parameters: 134,268,737
Data Set: Few hundred images

Classify Rarest Animals



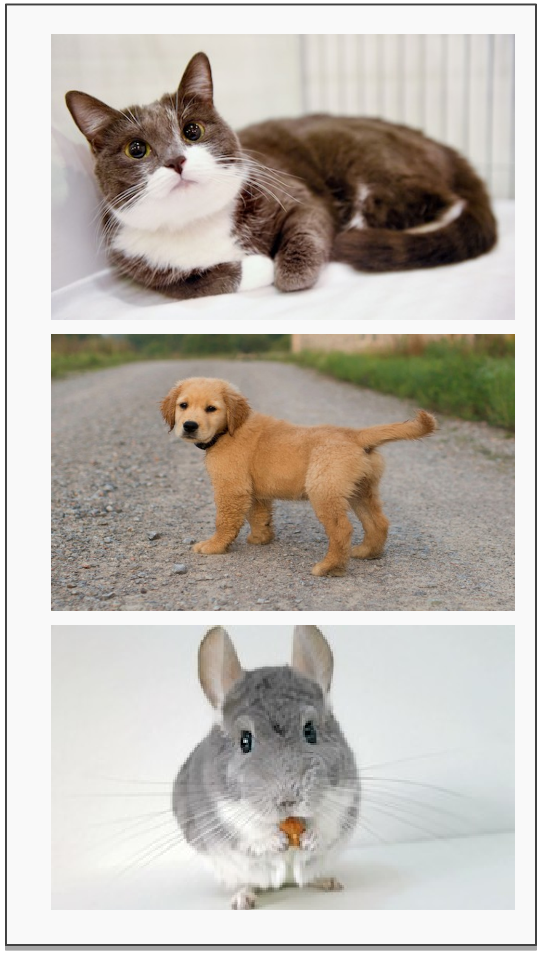
VGG16



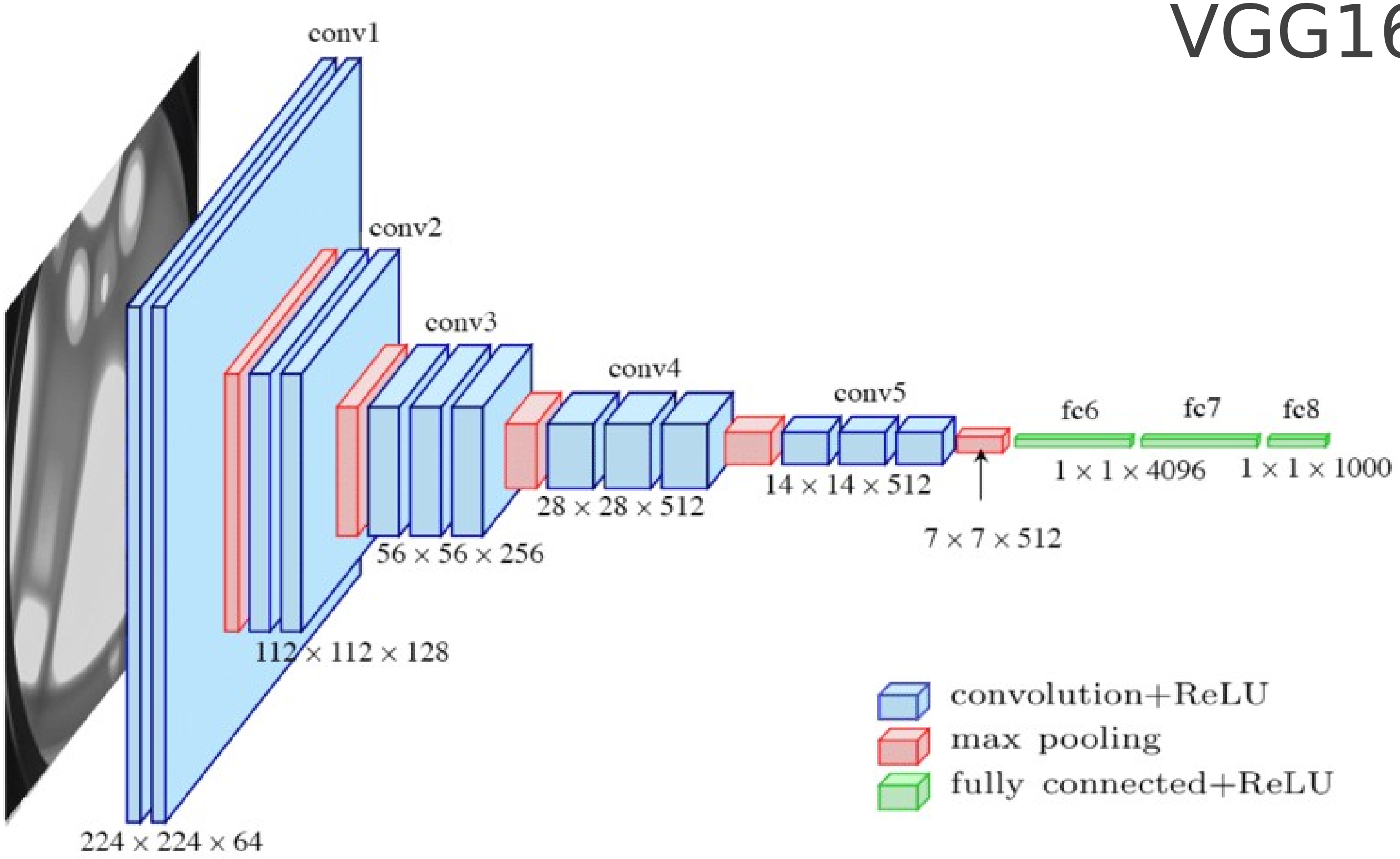
NOT ENOUGH DATA

Number of parameters: 134,268,737
Data Set: Few hundred images

Classify Cats, Dogs, Chinchillas etc



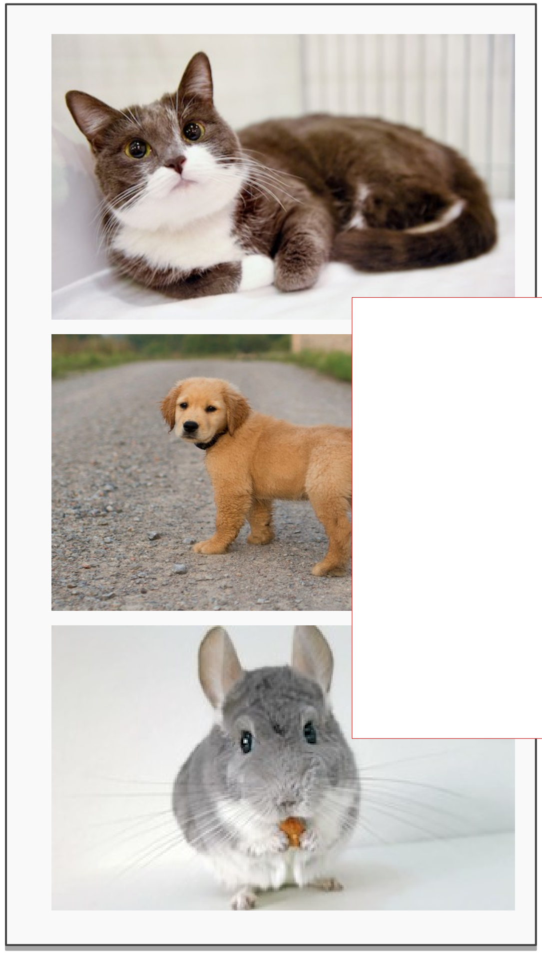
VGG16

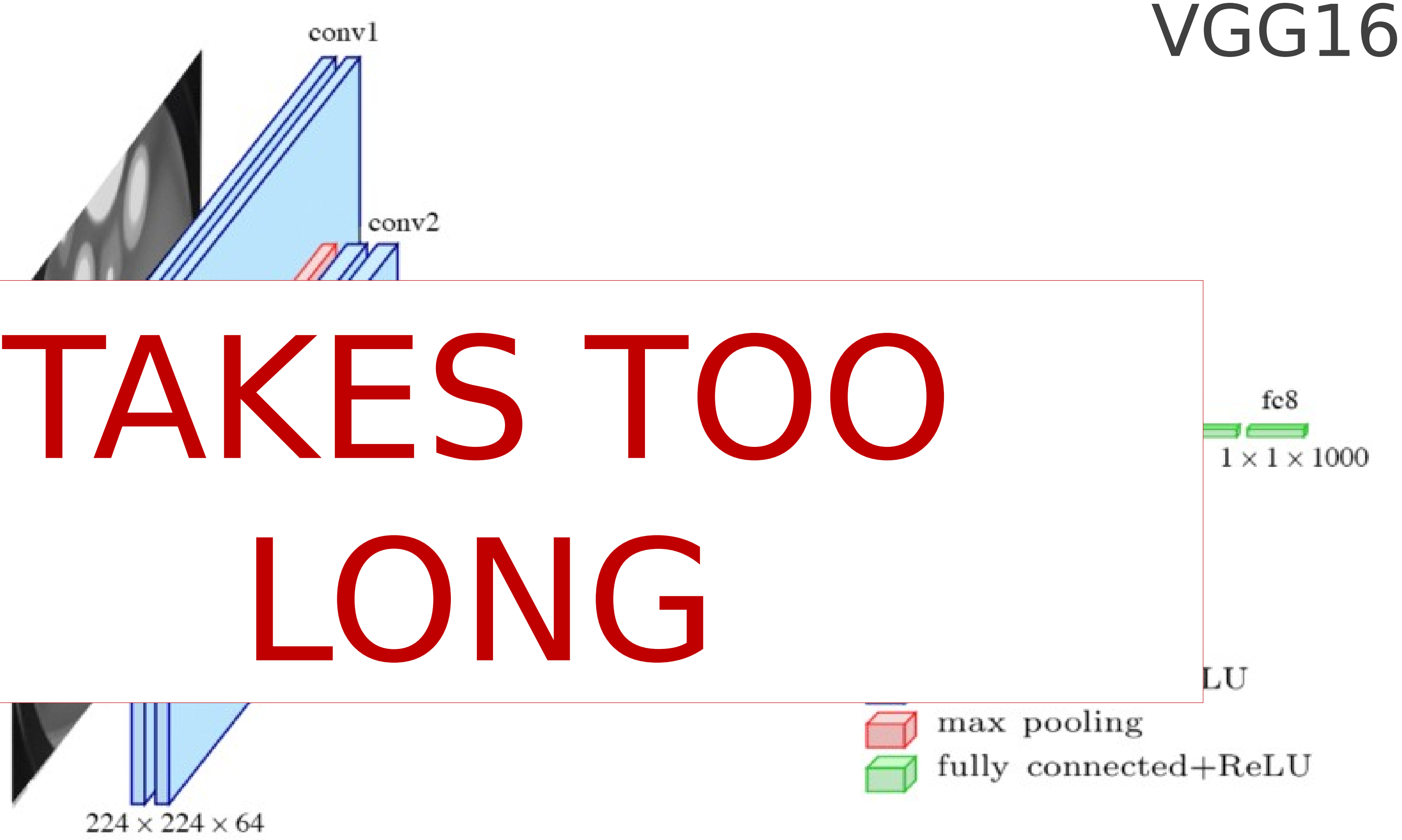


Number of parameters: 134,268,737
Enough training data. ImageNet approximate 1.2M

Classify Cats, Dogs, Chinchillas etc

VGG16





TAKES TOO LONG

Number of parameters: 134,268,737
Enough training data. ImageNet approximate 1.2M

Transfer Learning To The Rescue

How do you build an image classifier that can be trained in a few minutes on a GPU with very little data?

Basic idea of Transfer Learning

Train a ML model M for a task T using a dataset D_s

Use M on a new dataset D_T for the same task T

Use part of M on original dataset D_s for a new task T_n

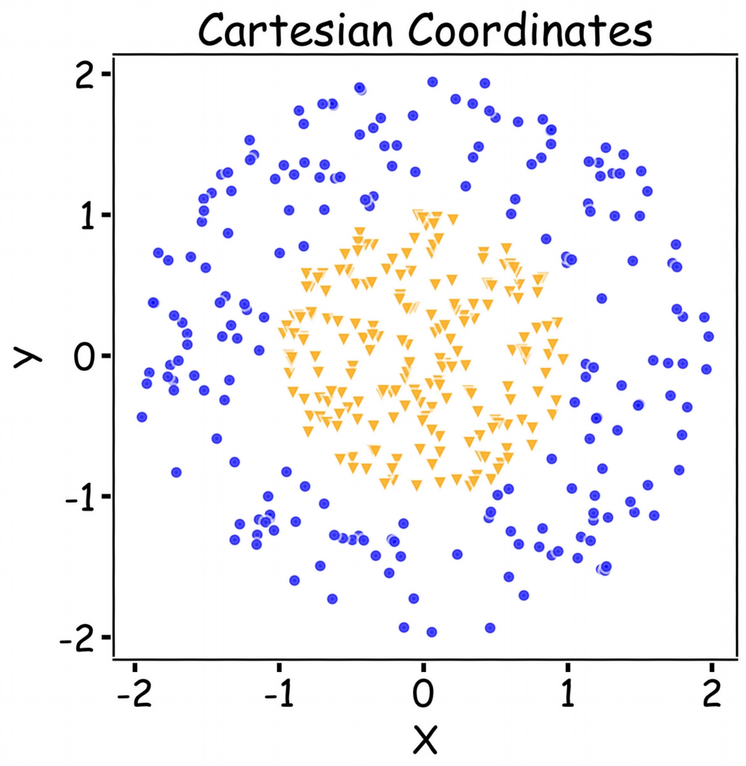
Use part of M on a new dataset D_T for a new task T_n

Wikipedia:
Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.[1]

Key Idea: Representation Learning

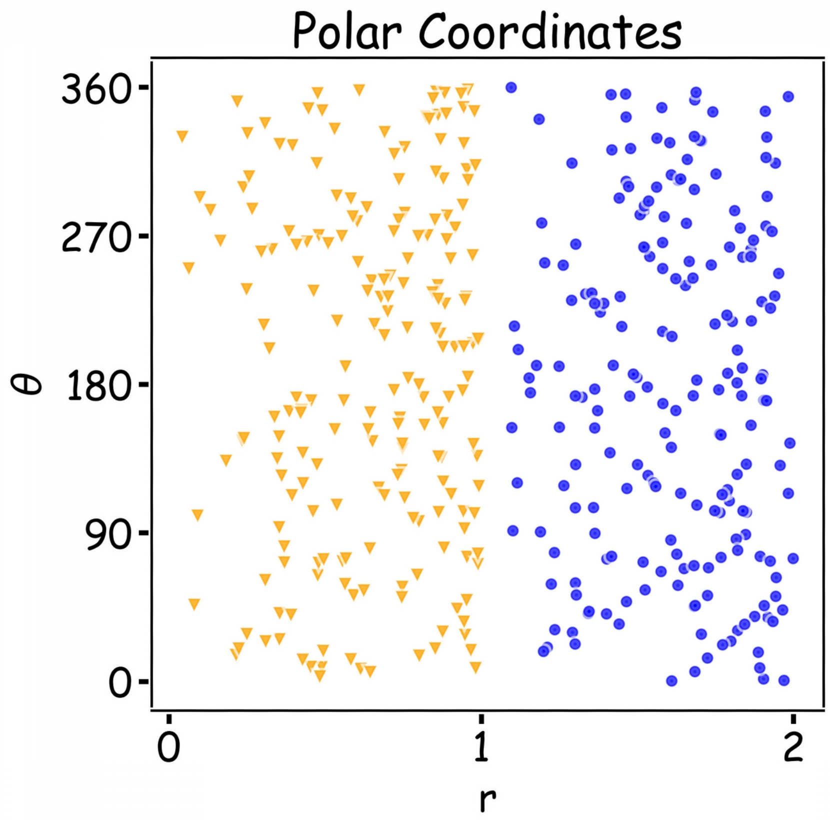
Relatively difficult task

Easier task



Transform

$(X, Y) \rightarrow (r, \theta)$

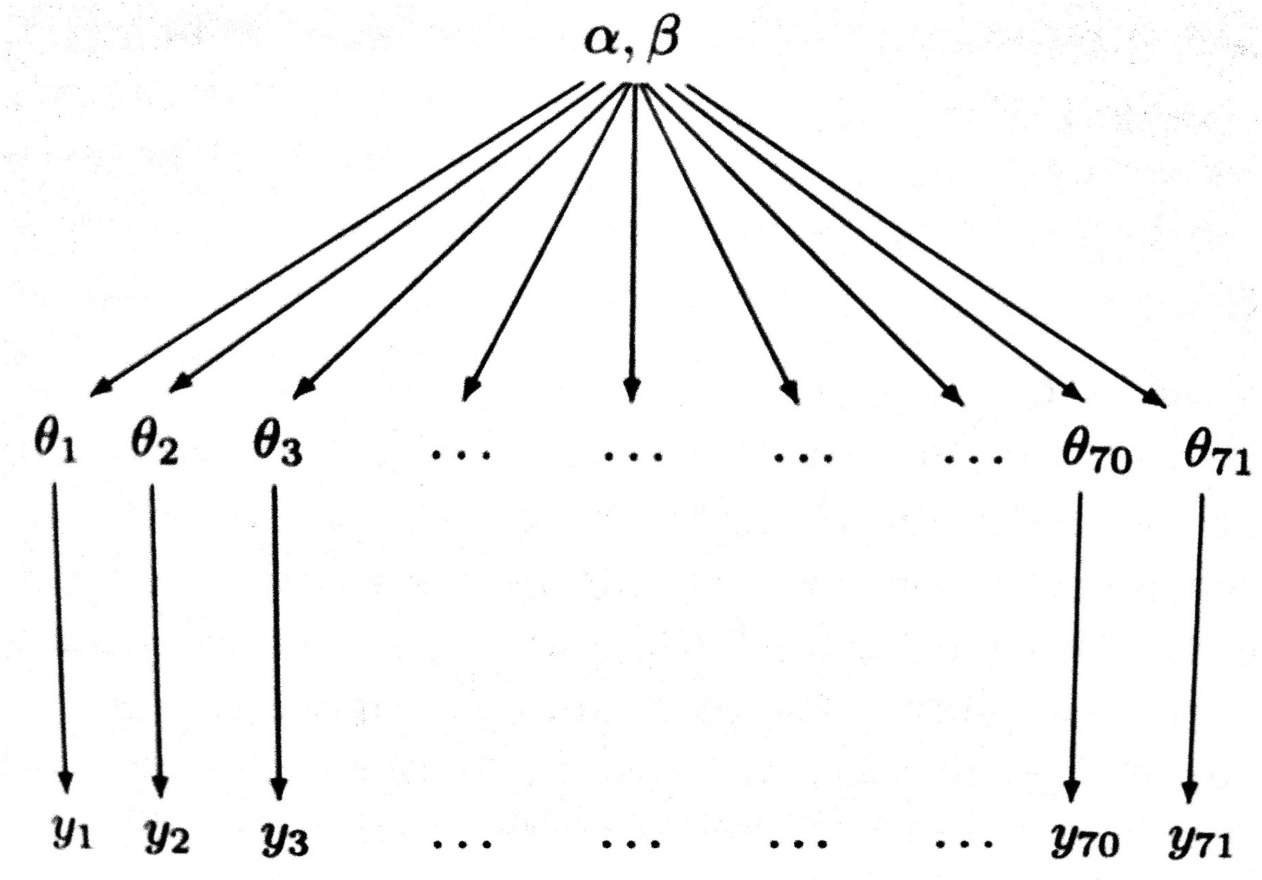


Transfer Learning

Not a new idea!
It has been there in the ML and stats literature for a while.

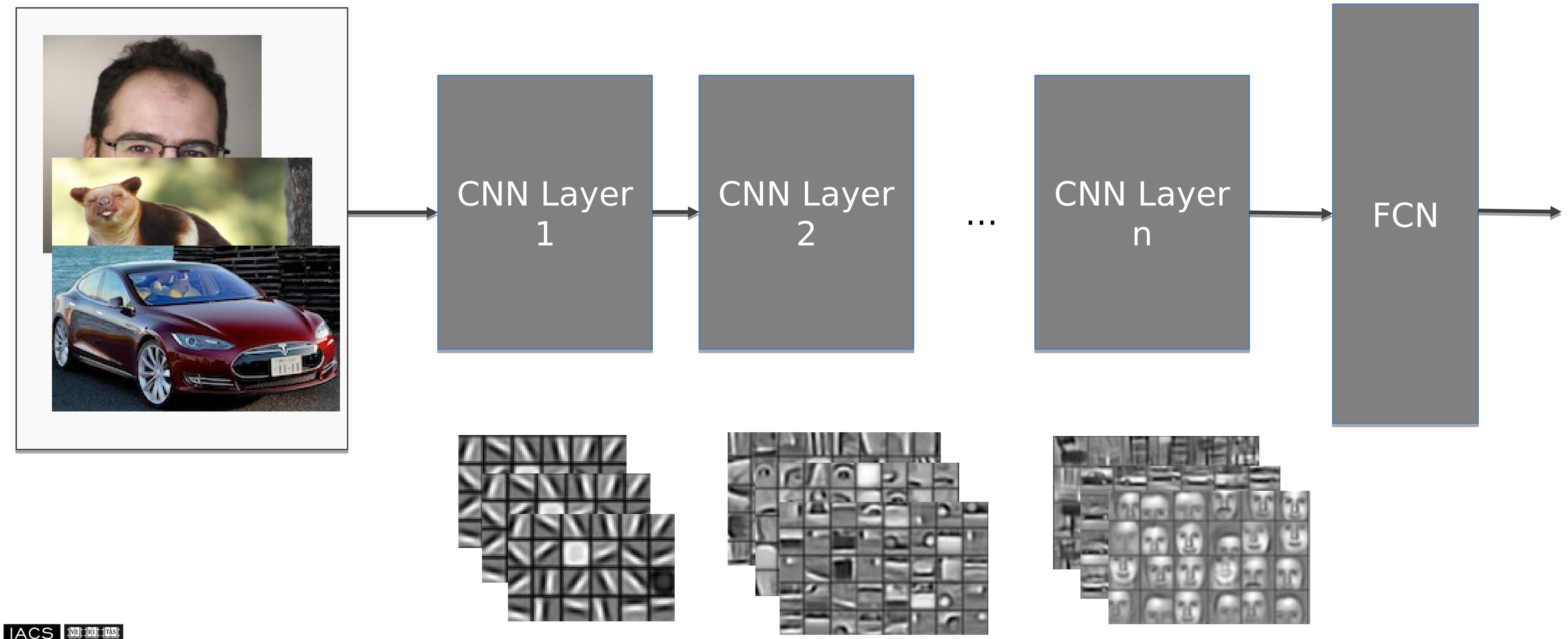
An example is hierarchical **GLM models** in stats, where information flows from higher data units to the lower data units.

Neural networks **learn hierarchical representations** and thus are suited to this kind of learning.



Representation Learning

Task: classify cars, people, animals and objects



Transfer Learning To The Rescue

How do you make **an image classifier** that can be **trained in** a few minutes on a GPU with very little data?

Use pre-trained models, i.e., models with known weights.

Main Idea: Earlier convolution layers learn low level features, which can be adapted to new domains by changing weights at later and fully-connected layers.

Example: Use ImageNet to train a huge deep network. Then retrain it on a few images

Transfer Learning To The Rescue

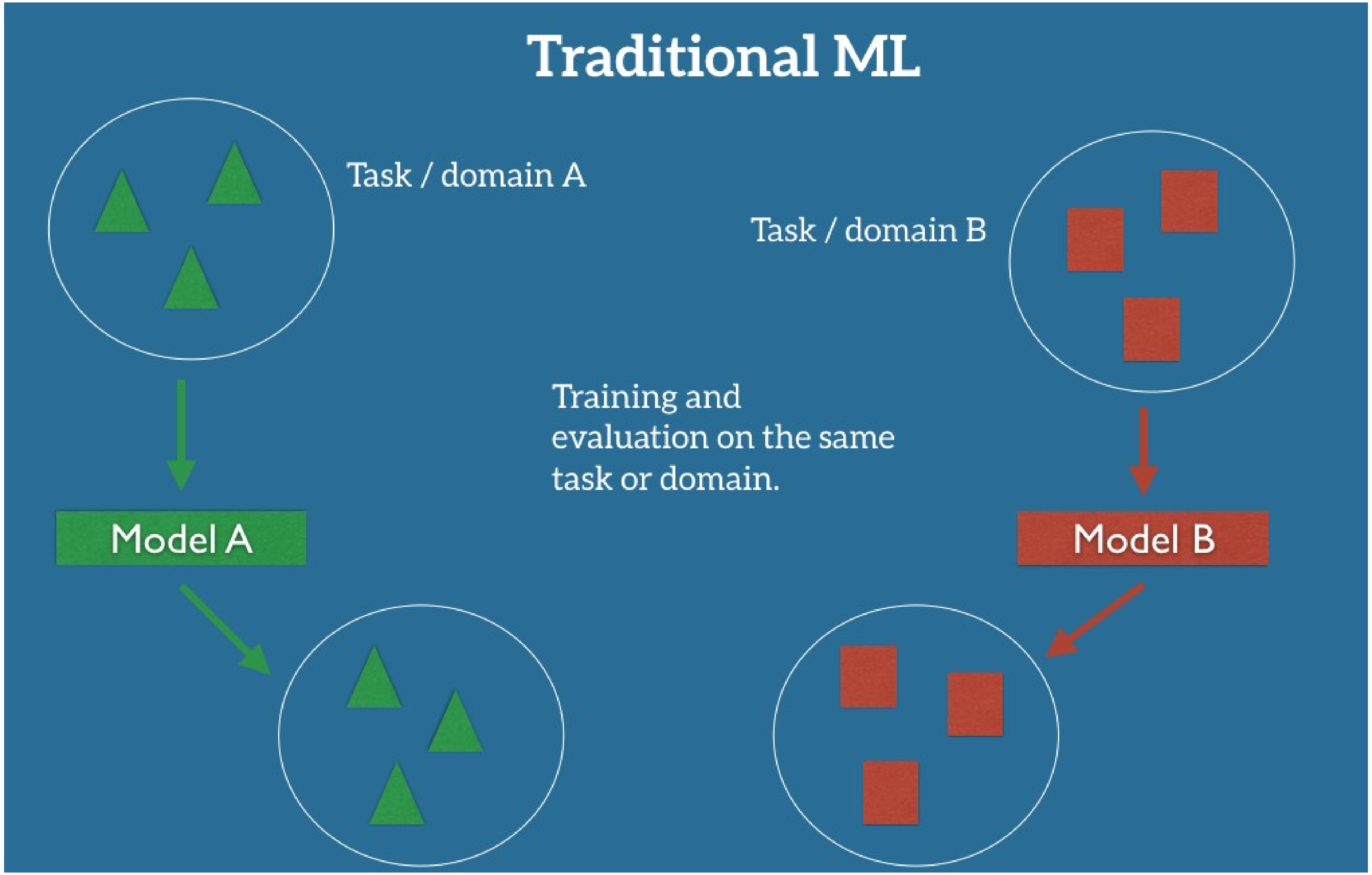
Train on a big "**source**" data set, with a big model, on one particular downstream tasks and save the parameters. This is called a **pre-trained model**.

Use these parameters for other smaller "**target** " datasets (possibly different **domain**, or training distribution).

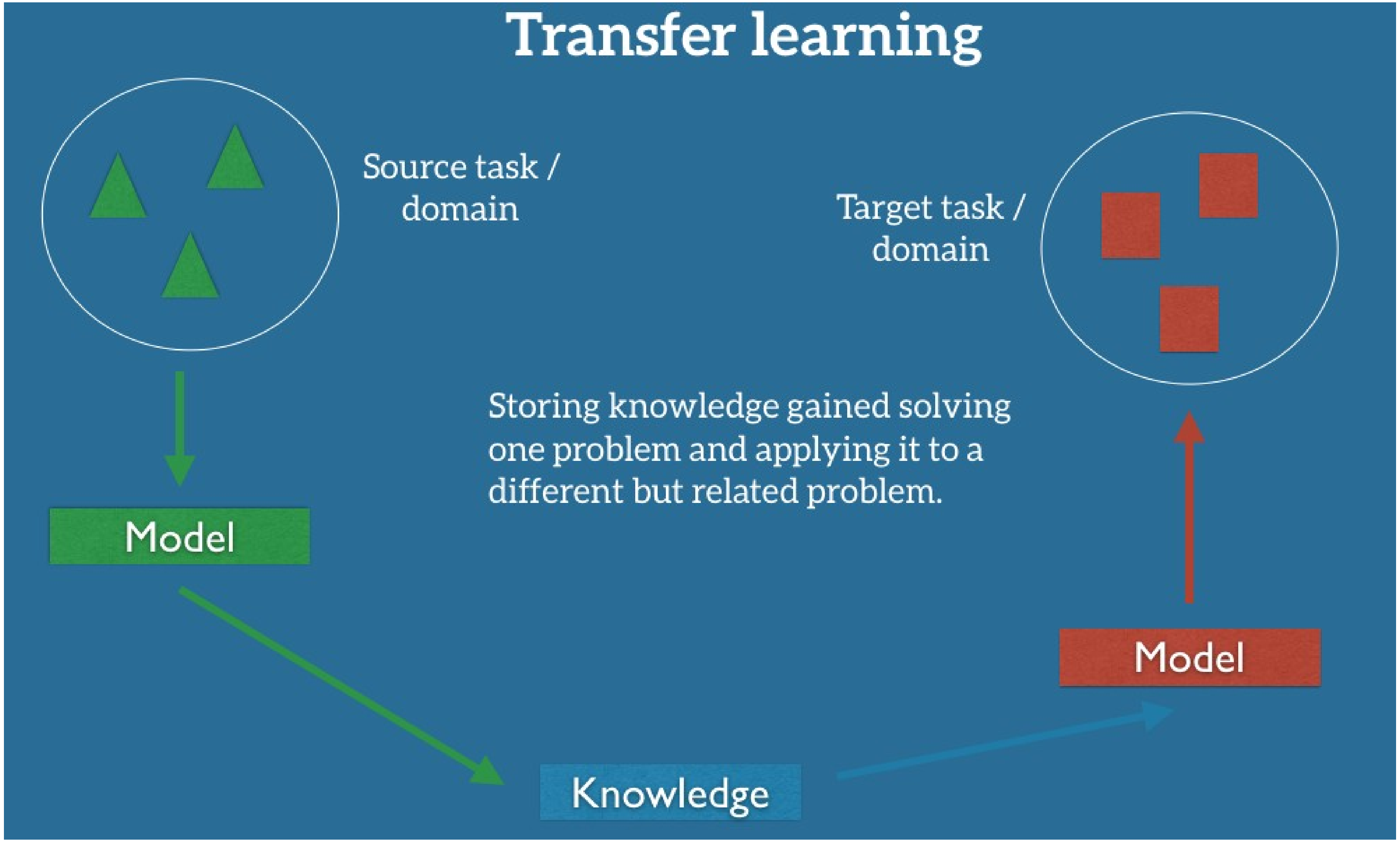
Less helpful if you have a large target dataset with many labels.

It will fail if the source domain has nothing in common with target domain.

Machine Learning



Transfer Learning



Applications

Learning from simulations (self driving cars, games)

Domain adaptation:
Bikes to bikes with backgrounds, bikes at night, etc

Speech recognition.
Classify speakers with minimal training such that only a few words or sentences are needed to achieve high levels of accuracy.

Cross-lingual adaptation for few shot learning of resource poor languages (english->nepali for example)

Using a pre-trained net

Create a classifier to distinguish dogs and flowers

Use MobileNet previously trained on Imagenet with 1.4 M images and 1000 classes. Very expensive training

Replace the head (classifier) FC layers. Freeze the base (convolution) layers. Train the Network

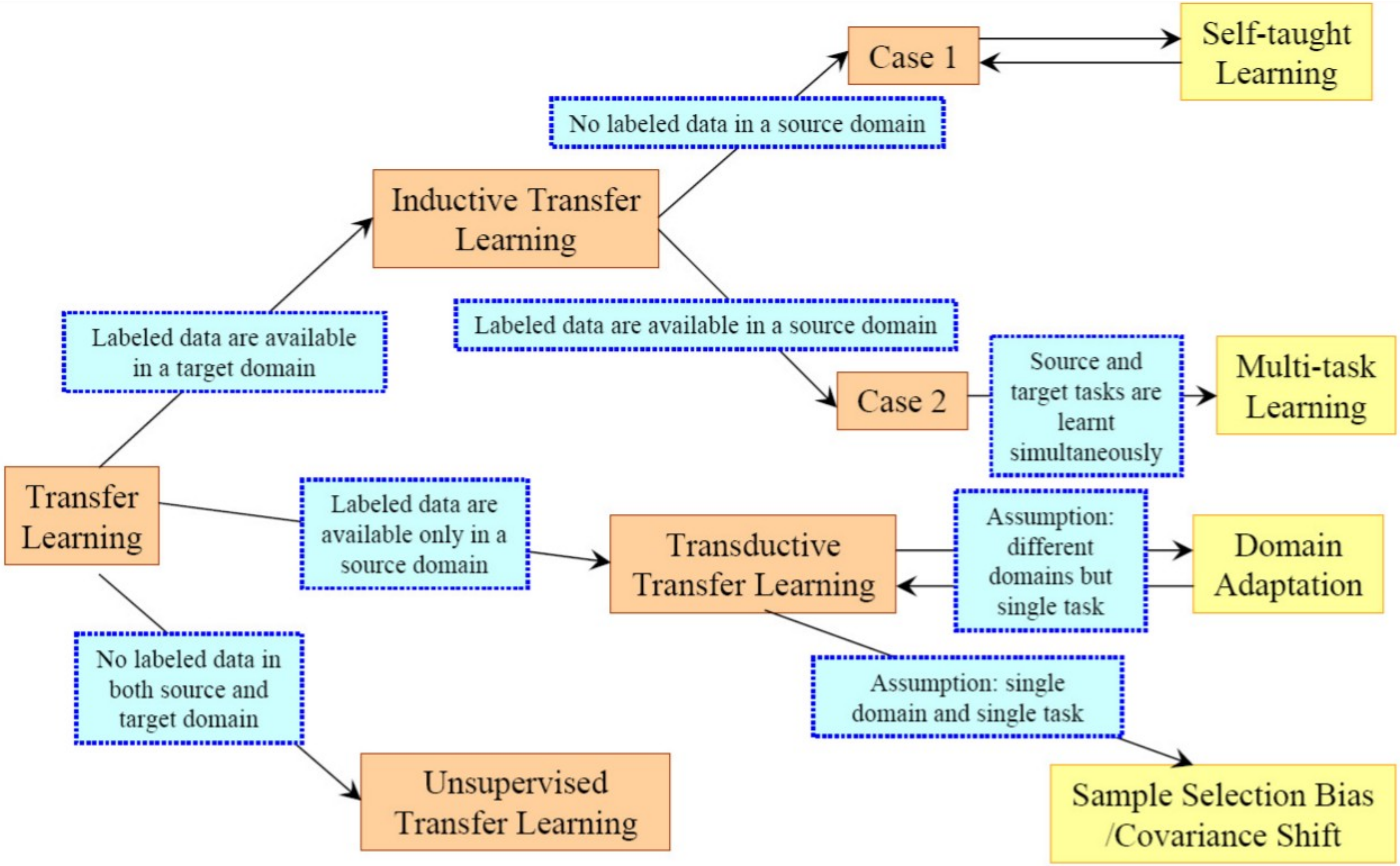
Fine-Tuning. Unfreeze the convolution layers and train the entire network

Key Takeaways

During the process of transfer learning, the following three important questions must be answered:

- **What to transfer:** Identify which portion of knowledge is source-specific and what is common between the source and the target.
- **When to transfer:** We need to be careful about when to transfer and when not to. Aim at utilizing transfer learning to improve target task performance/results and not degrade them (negative transfer).
- **How to transfer:** Identify ways of transferring the knowledge across domains/tasks.

Transfer Learning Strategies



Pan and Yang, A Survey on Transfer Learning

Transfer Learning for Deep Learning

What people thinks

- you can't do deep learning unless you have a million labeled examples.

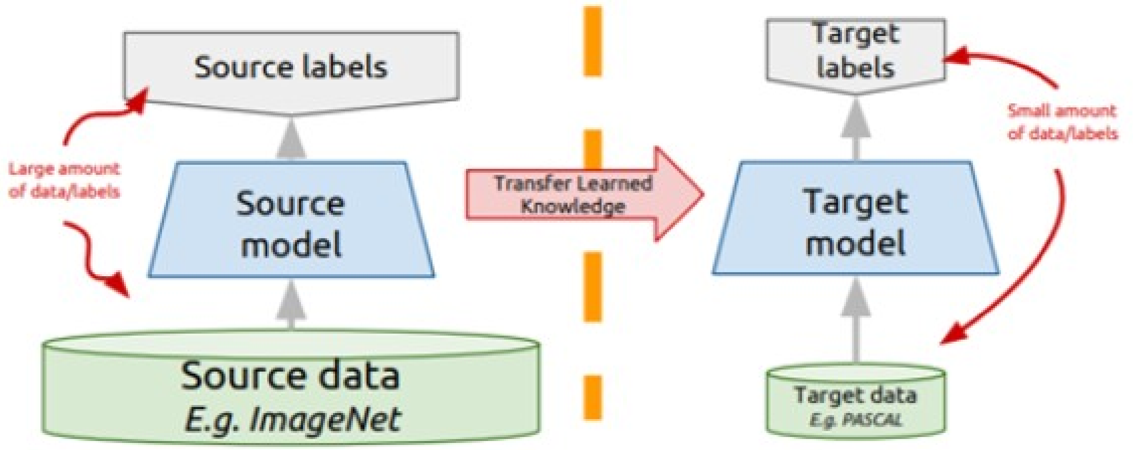
What people can do, instead

- Can learn representations from unlabeled data
- Can transfer learned representations from a relate task.

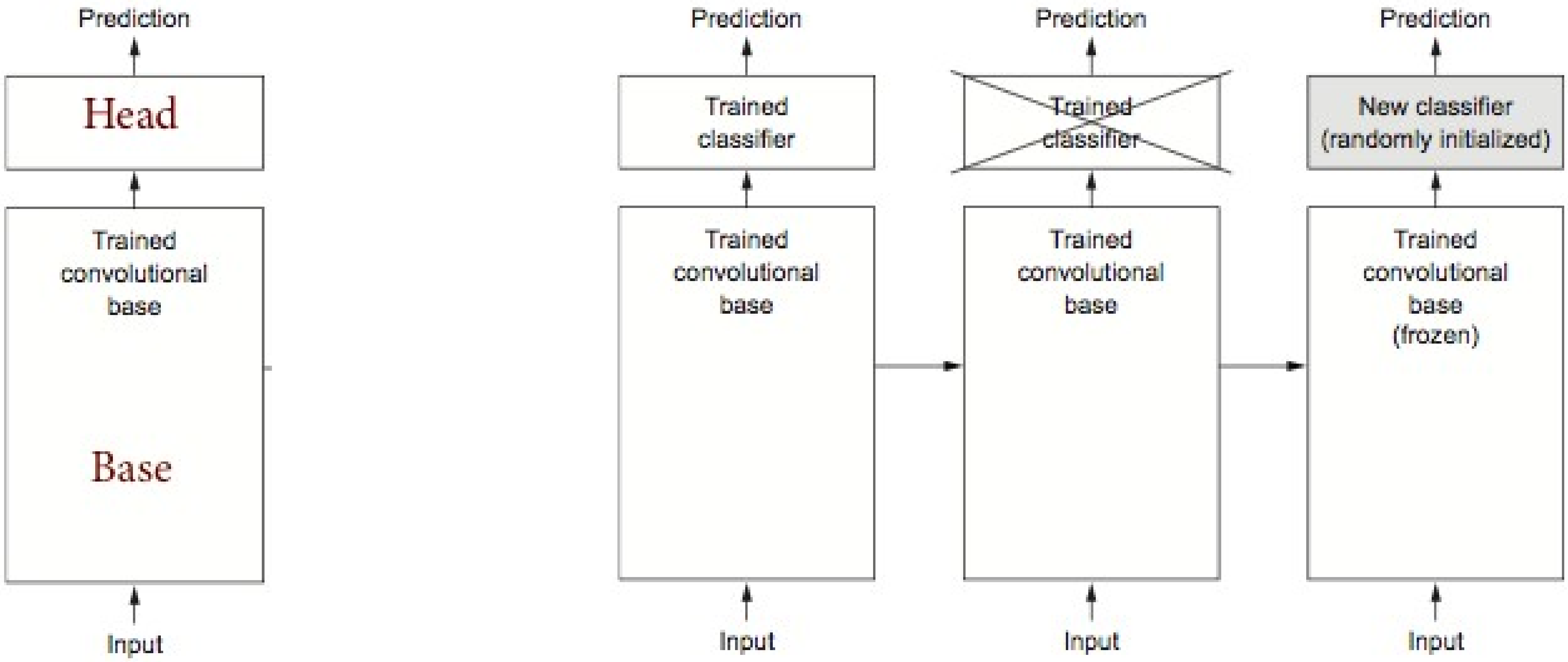
Transfer Learning for Deep Learning

Instead of training a network from scratch:
- Take a network trained on a different domain for a different **source task**
- Adapt it for your domain and your **target task**

Variations
- Same domain, different task.
- Different domain, same task.



Transfer the Feature Extraction



Representation Extraction

Use representations learned by big net to extract features from new samples, which are then fed to a new classifier:

- Keep (frozen) convolutional **base** from big model

- Throw away **head** FC layers since these have no notion of space, and convolutional base is more generic

- Since there are both dogs and flowers in ImageNet you could use the head FC layers as well but by throwing it away you can learn more from new dog/cat images

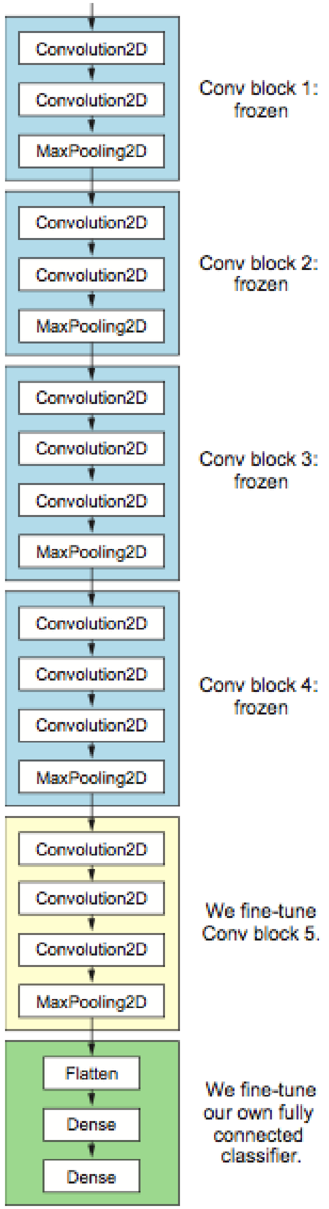
Fine-tuning

Up to now we have frozen the entire convolutional base.

Earlier layers learn highly generic feature maps (edges, colors, textures) while later layers learn abstract concepts (dog's ear).

To particularize the model to our task, we can tune the later layers

We must be very careful not to have big gradient updates.



Procedure for Fine-tuning

1. Freeze the convolutional base

2. Train the new fully connected head, keeping the convolutional base fixed. This will get their parameters away from random and in a regime of smaller gradients

3. Unfreeze some or all "later" layers in the base net

4. Now train the base net and FC net together.

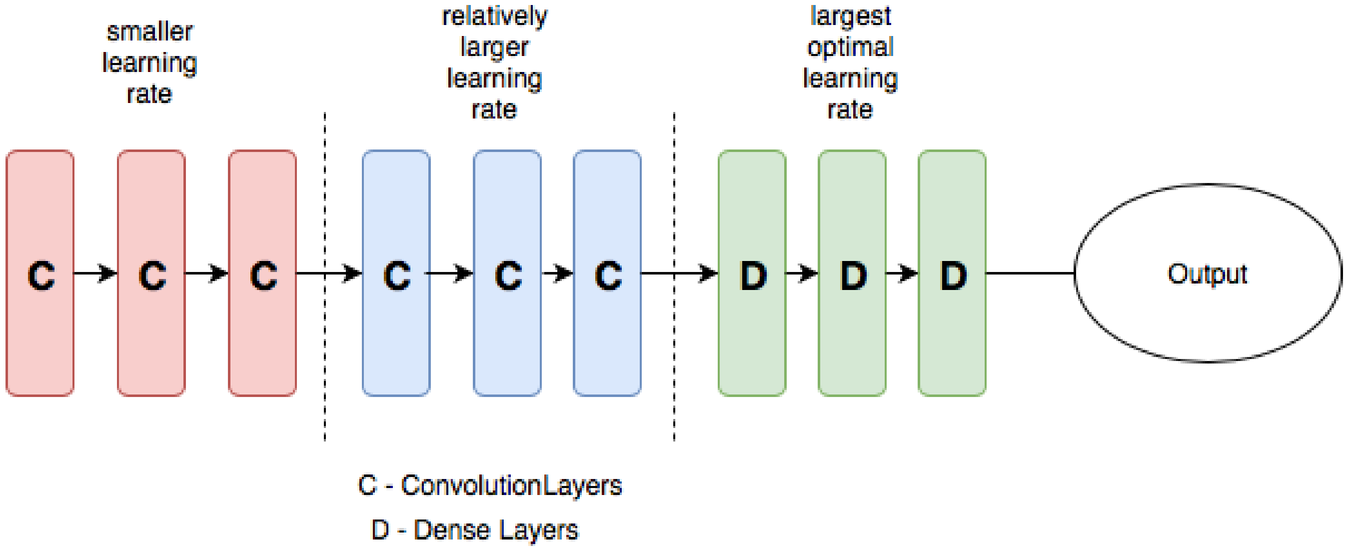
Since you are now in a better part of the loss surface already, gradients won't be terribly high, but we still need to be careful. Thus use a **very low learning rate**.

Transfer Learning for Deep Learning: Differential Learning Rates

Train different layers at different rates

Each "earlier" layer or layer group can be trained at 3x-10x smaller learning rate than the next "later" one.

One could even train the entire network again this way until we overfit and then step back some epochs.



State of the Art Deep Models:

Some of the good pre-trained models for transfer-learning

AlexNet

VGGs (16-19)

Inception (AKA Google-Net)

ResNet

MobileNet

DenseNet

State of the Art Deep Models:

Some of the good pre-trained models for transfer-learning

AlexNet

VGGs (16-19)

Inception (AKA Google-Net)

ResNet

MobileNet

DenseNet

Mobile Net: A light weight model

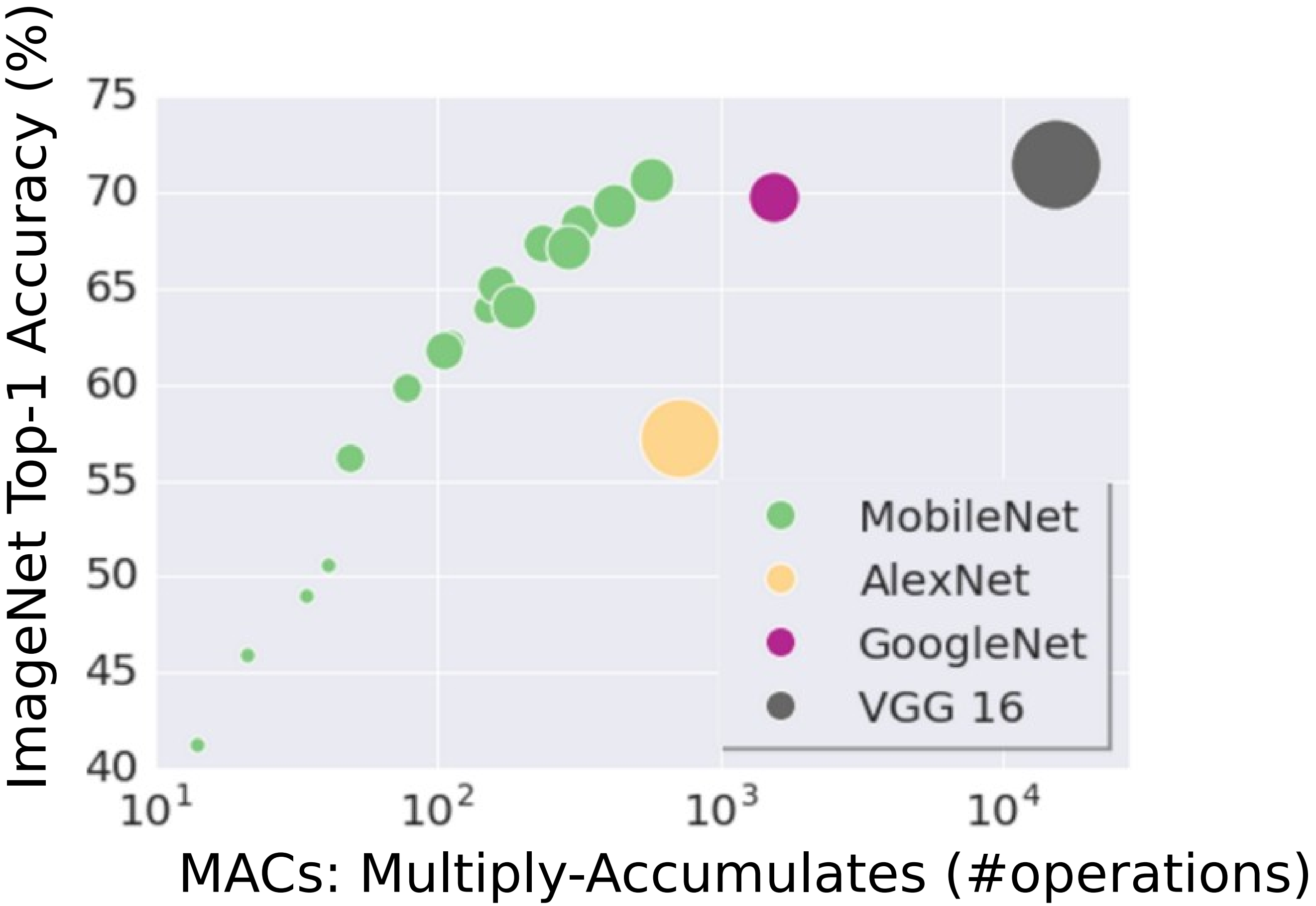
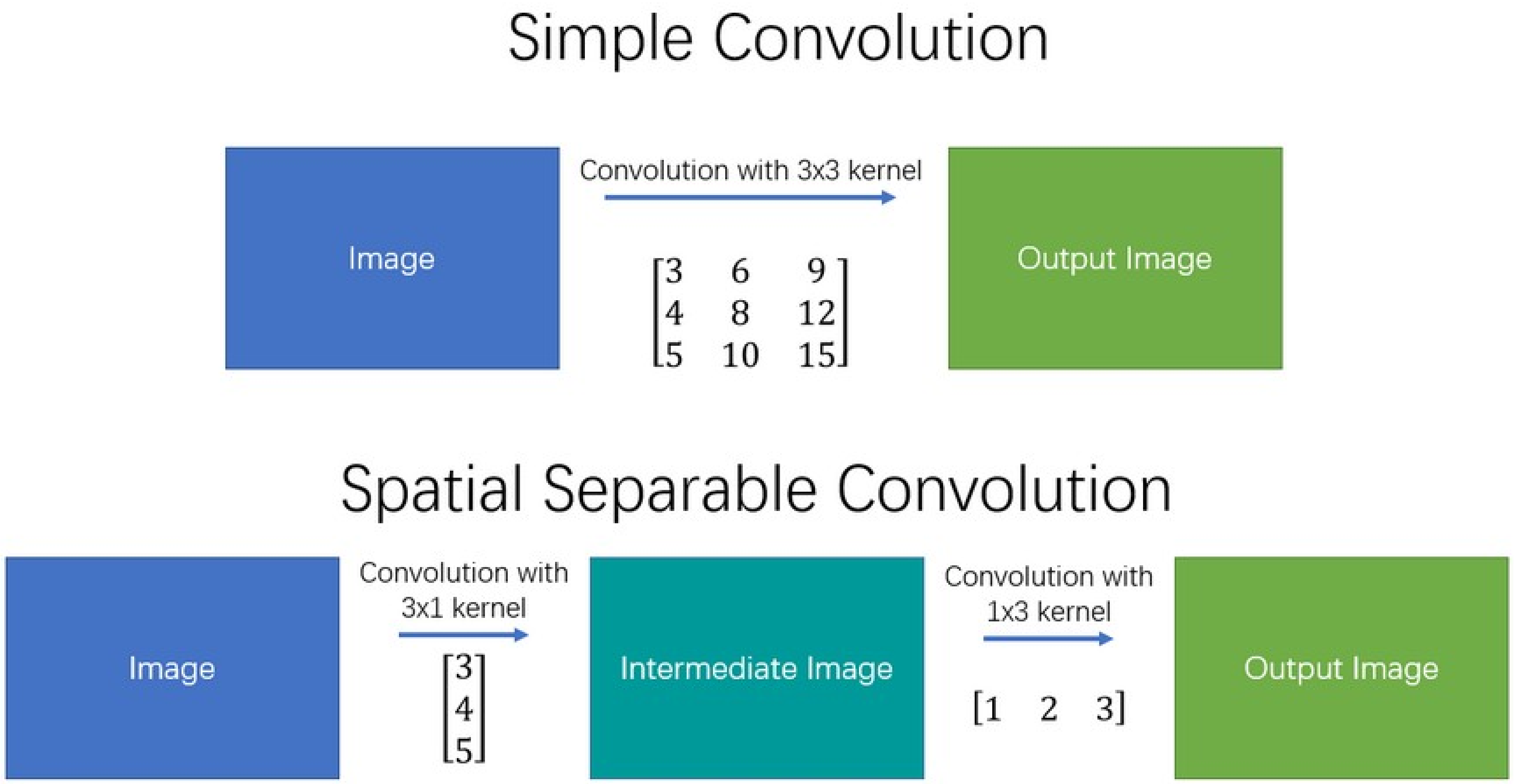


Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

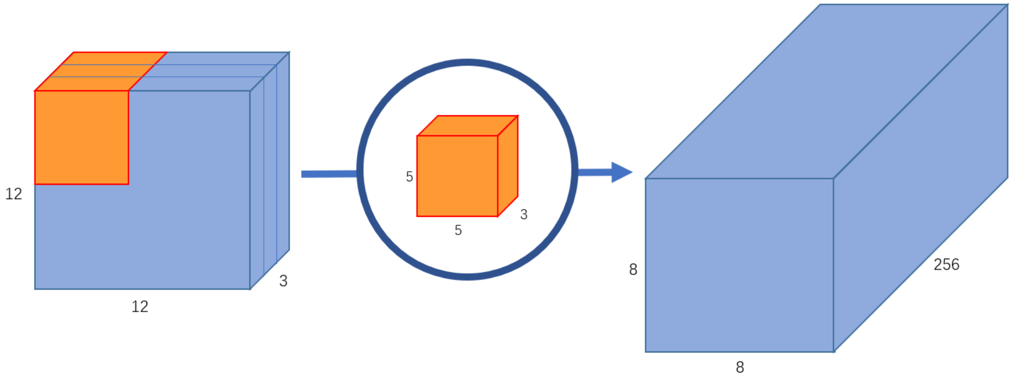
MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications (arXiv.1704.04861)

Key Idea. Separable Convolution



Standard Convolution

A standard convolution filters and combines inputs into a new set of outputs in one step.



Input: 12x12x3
Filter: 5x5x3x256

Output: 8x8x256
(no padding)

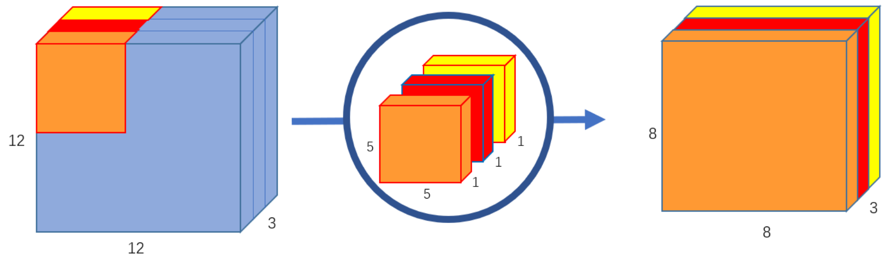
Input: 12x12x3
Filter: 5x5x3x256

Output: 8x8x256
(no padding)

MACs: (5x5)x3x256x(12x12) ~ 2.8M
Parameters: (5x5x3)x256 + 256 ~ 20K

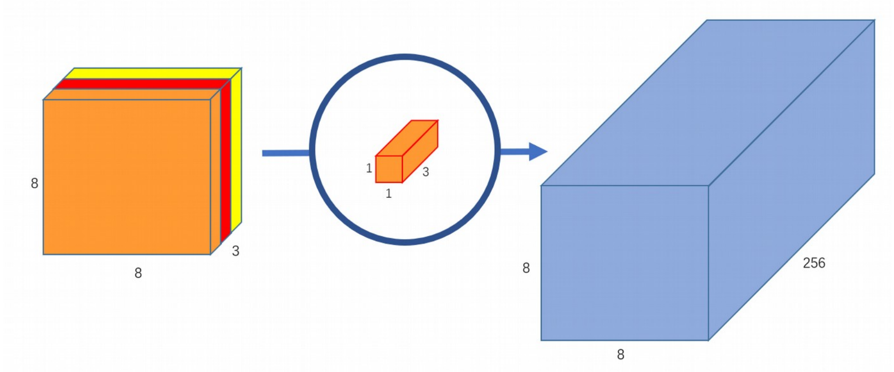
Depthwise separabe Convolution

The depthwise separable convolution makes 2 steps: A depthwise convolution and a pointwise convolution.



Input: 12x12x3
Filter: 5x5x3

Output: 8x8x3
(no padding)

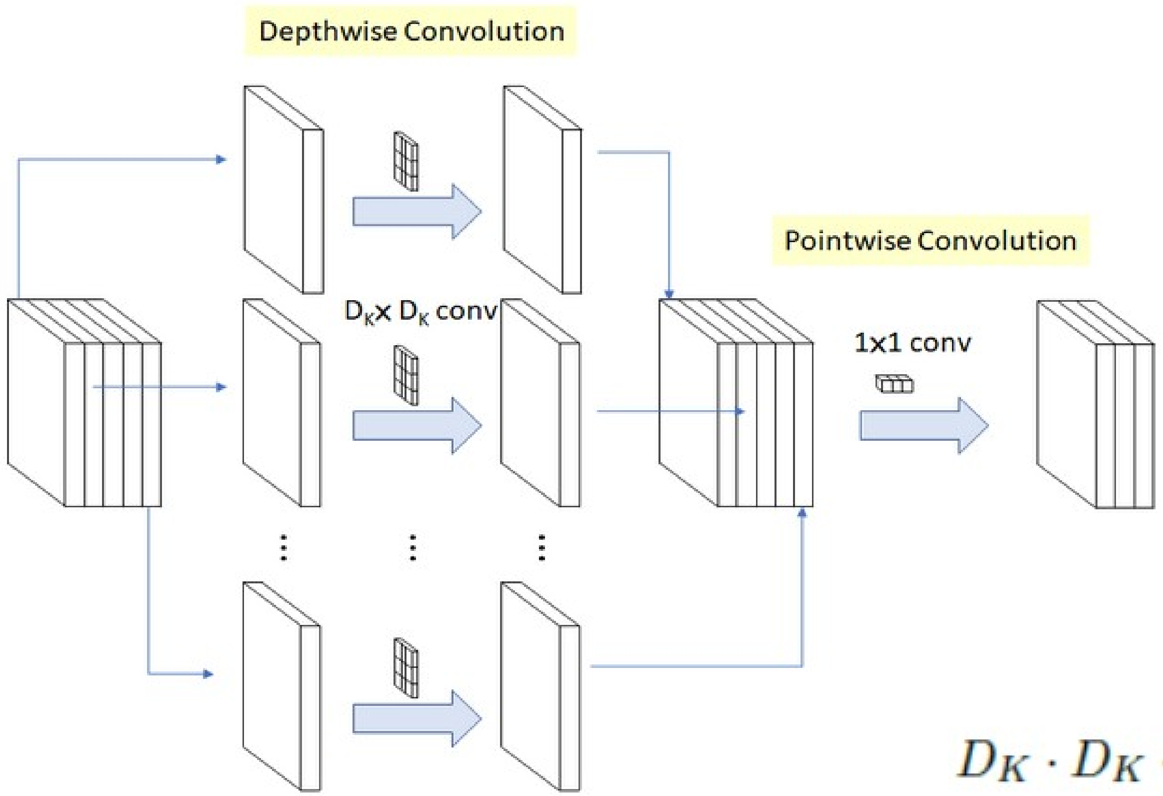


Input: 8x8x3
Filter: 1x1x3x256

Output: 8x8x256
(no padding)

MACs: (5x5)x3x(12x12) + 3x256x(8x8) ~ 60K
Parameters: (5x5x3 + 3) + (1x1x3x256+256) ~ 1K

Computation Reduction



M input channels

N output channels.

D_K the filter (kernel) size

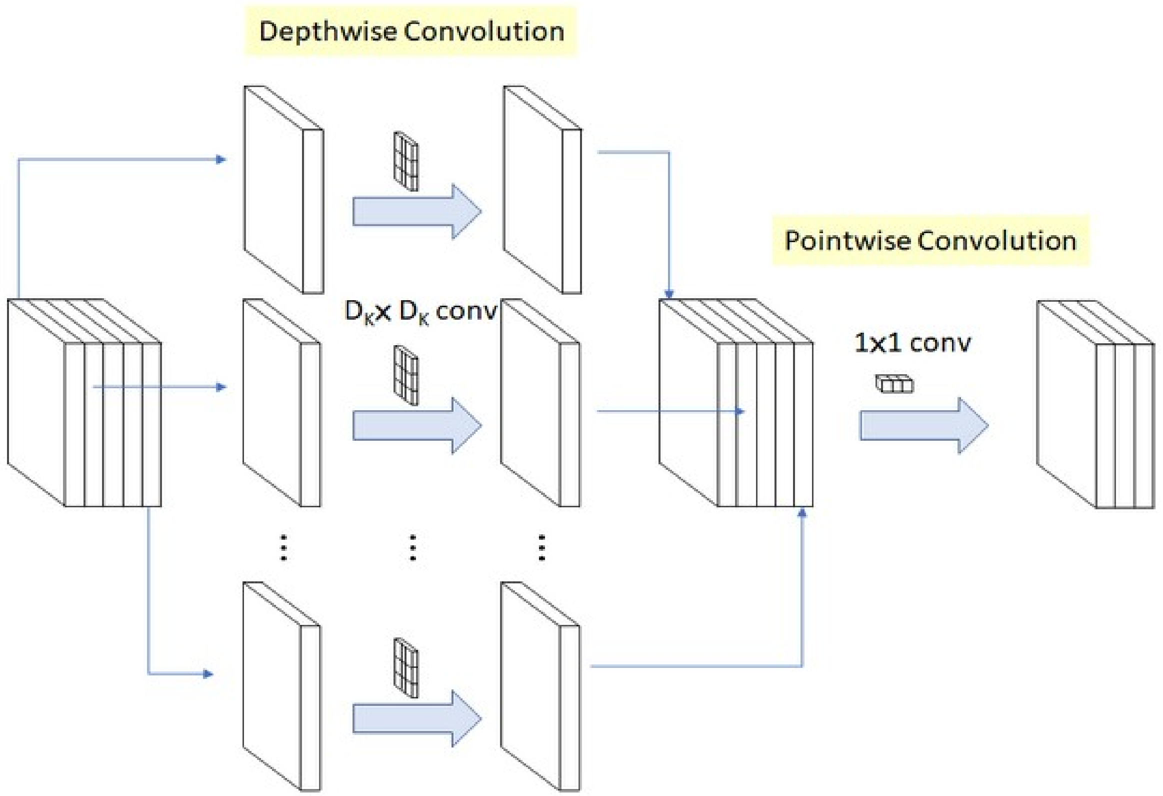
D_F the feature map size

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$$

Depthwise Separable Convolution Cost: Depthwise Convolution Cost (Left), Pointwise Convolution Cost (Right)

Computation Reduction



M input channels

N output channels.

D_K the filter (kernel) size

D_F the feature map size

The computation Reduction comparing to standard convolution is

$$\frac{1}{N} + \frac{1}{D_K^2}$$

Let’s have some Action (coding)

Training the MobileNet

Consider a small data set of 5 groups and totally less than 1K labeled images.
Can we use this small data set to train a deep and very expressive network such as the MobileNet?

Load the un-trained MobileNet

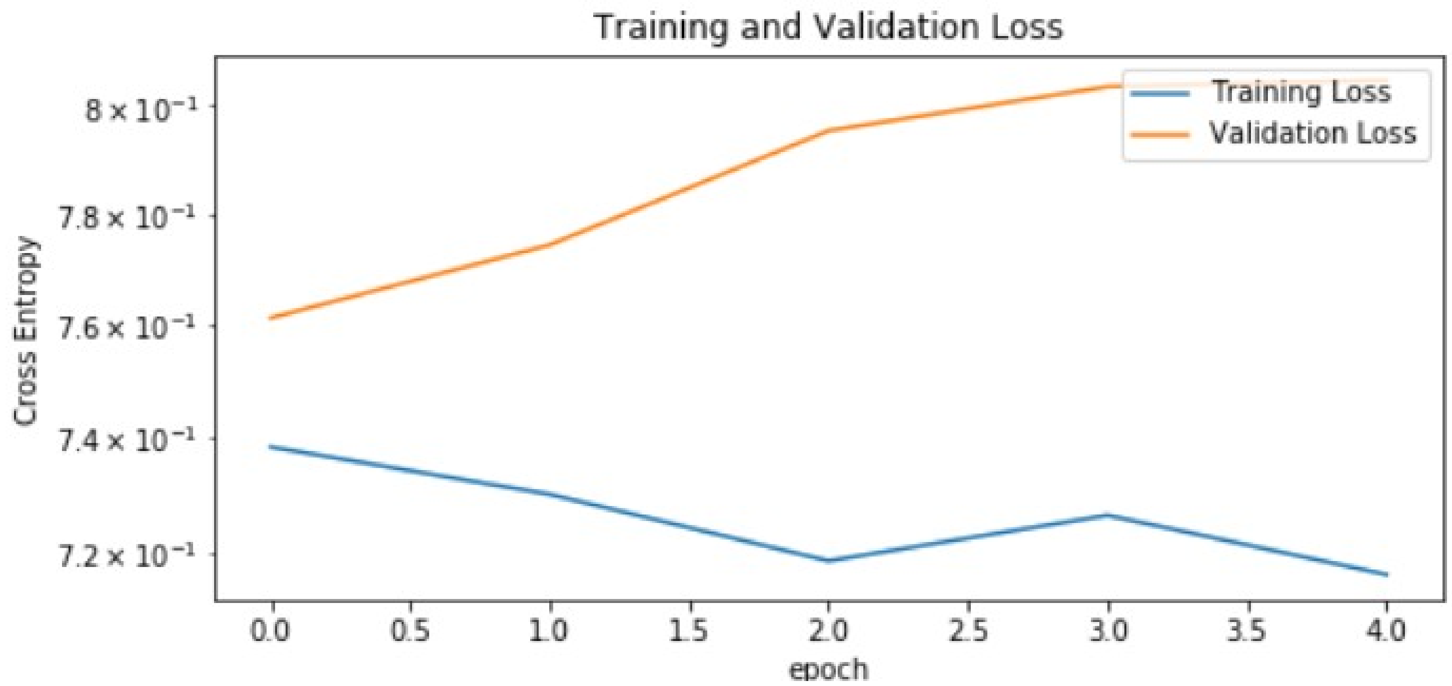
```
# Loading the un-trained weights model
mobile_from_scratch = tf.keras.applications.mobilenet.MobileNet(input_shape=IMG_SHAPE, weights=None,
                                                                 classes=len(listGroupsTrain))
```

Data Generator

```
:  train_datagen=ImageDataGenerator(preprocessing_function=preprocess_input,
                                                   horizontal_flip=True,
                                                   rotation_range=45,
#                                                   zoom_range=[0.8,1.0]
                                 )

test_datagen=ImageDataGenerator(preprocessing_function=preprocess_input)

# TRAINING set
pathTrain = pathFolder + 'trainData/'
listGroupsTrain = os.listdir(pathTrain) # the directory path

# TESTING set
pathTest = pathFolder + 'testData/'
listGroupsTest = os.listdir(pathTest) # the directory path

# Load the data into the ImageDataGenerator
train_generator=train_datagen.flow_from_directory(pathTrain,
                                                 target_size=(IMG_SIZE,IMG_SIZE),
                                                 color_mode='rgb',
                                                 batch_size=64,
                                                 class_mode='categorical',
                                                 shuffle=True,
                                                 classes=listGroupsTrain)


test_generator=test_datagen.flow_from_directory(pathTest,
                                                 target_size=(IMG_SIZE,IMG_SIZE),
                                                 color_mode='rgb',
                                                 batch_size=64,
                                                 class_mode='categorical',
                                                 shuffle=False,
                                                 classes=listGroupsTest)
```

```
Found 843 images belonging to 5 classes.
Found 104 images belonging to 5 classes.
```

Compile and train the MobileNet



Transfer Learning with MobileNet

Loading the pre-trained MobileNet network

```
# Choose the weights pretrained in the imagenet dataset
mobile = tf.keras.applications.mobilenet.MobileNet(weights='imagenet')
```

```
mobile.summary()
```

```
Model: "mobilenet_1.00_224"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
conv1_pad (ZeroPadding2D)    (None, 225, 225, 3)       0
_________________________________________________________________
conv1 (Conv2D)               (None, 112, 112, 32)      864
_________________________________________________________________
conv1_bn (BatchNormalization (None, 112, 112, 32)      128
```

Helper functions

This function prepares the images for the MobileNet: input shape: (1, 224, 224, 3)

```
def prepare_image(img_path, img_size = IMG_SIZE):
    img = image.load_img(img_path, target_size=(img_size, img_size))
    img_array = image.img_to_array(img)
    img_array_expanded_dims = np.expand_dims(img_array, axis=0)
    return keras.applications.mobilenet.preprocess_input(img_array_expanded_dims)
```

Another helper function for doing the classification

```
def mobileClassifier(imagePath, pathFolder=pathFolder, mobile=mobile):
    imagePathFull = pathFolder + imagePath
    preprocessed_image = prepare_image(imagePathFull)
    # Use mobileNet to classify the image
    predictions = mobile.predict(preprocessed_image)
    results = imagenet_utils.decode_predictions(predictions)
    return results
```

Classify on a new dataset

```
mobileClassifier('pcaData/labrador/5.labrador_retriever.jpg')
```

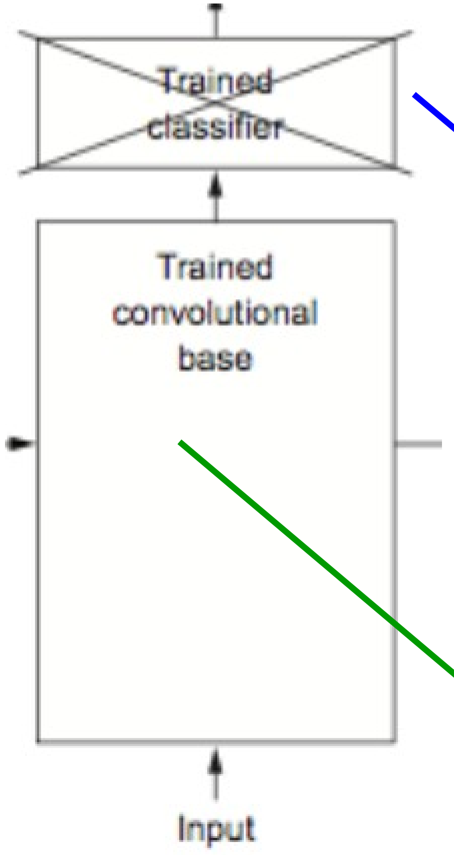
```
[[('n02099712', 'Labrador_retriever', 0.9703214),
  ('n02099601', 'golden_retriever', 0.014126321),
  ('n02104029', 'kuvasz', 0.0036305177),
  ('n02099849', 'Chesapeake_Bay_retriever', 0.0017487509),
  ('n02108422', 'bull_mastiff', 0.0011949409)]]
```

```
mobileClassifier('trainData/tulipsTrain/100930342_92e8746431_n.jpg')
```

```
[[('n03930313', 'picket_fence', 0.29128855),
  ('n02280649', 'cabbage_butterfly', 0.123460084),
  ('n12057211', "yellow_lady's_slipper", 0.09701885),
  ('n11939491', 'daisy', 0.088766344),
  ('n02281406', 'sulphur_butterfly', 0.07786752)]]
```

Is there any problem? Where is it? Detecting the problem we know what to transfer and what to train

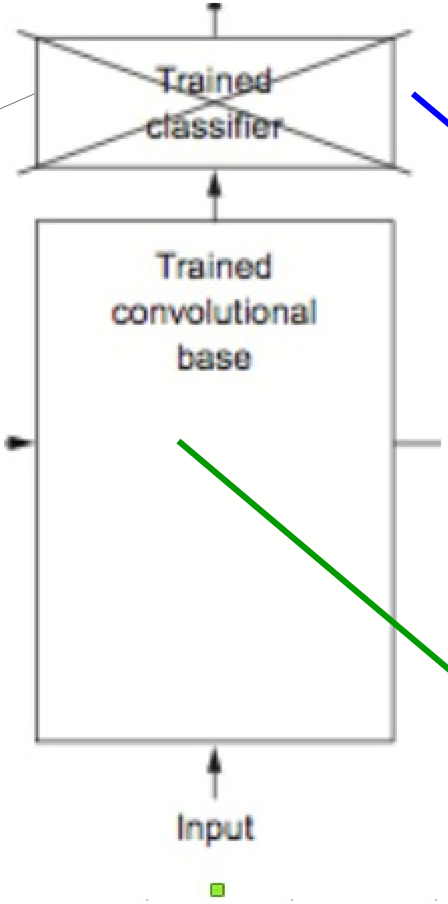
Classify on a new dataset



Define the base model

```
#your code here
transferModel_base = tf.keras.applications.MobileNet(input_shape=IMG_SHAPE, weights='imagenet', include_top=False)
```

Classify on a new dataset



Add a AveragePooling
and then
add one or more new FC layers

Define the base model

```
#your code here
transferModel_base = tf.keras.applications.MobileNet(input_shape=IMG_SHAPE, weights='imagenet', include_top=False)
```

That’s it

For the homework you should use GPUs to accelerate the training. You can use the **JupyterHub** at Canvas

Thank you!

References

- https://towardsdatascience.com/transfer-learning-using-mobilenet-and-keras-c75daf7ff299
- https://towardsdatascience.com/review-mobilenetv1-depthwise-separable-convolution-light-weight-model-a382df364b69
- https://www.alibabacloud.com/blog/part-3-image-classification-using-features-extracted-by-transfer-learning-in-keras_595291
- https://www.tensorflow.org/tutorials/images/transfer_learning
- https://arxiv.org/abs/1704.04861

Supplementary Material

Strategies

There are different transfer learning strategies and techniques, which can be applied **based on the domain, task** at hand, **and the availability of data:**

- **Inductive Transfer learning**: The source and target have **same domains**, yet the they have **different tasks** (e.g. documents written in the same language, but unbalanced labels).

- **Unsupervised Transfer Learning:** The source and target have **same domains**, with a focus on **unsupervised tasks in the target** domain. The source and target domains are similar, but the tasks are different. In this scenario, labeled data is unavailable in either of the domains.

- **Transductive Transfer Learning:** There are similarities between the source and target tasks, but the corresponding domains are different. The source domain has a lot of labeled data, while the target domain has none.

Strategies

A few categories of the approaches for Transfer Learning

- **Instance transfer:** Reusing knowledge from the source domain to the target task (ideal scenario). In most cases, the source domain data cannot be reused directly.

- **Feature-representation transfer:** Minimize domain divergence and reduce error rates by identifying good feature representations

- **Parameter transfer:** This approach works on the assumption that the models for related tasks share some parameters or prior distribution of hyperparameters.