Transfer Learning

Neural Networks for Machine Learning Applications 2023

Sakari Lukkarinen

Metropolia University of Applied Sciences

Contents

- What is transfer learning
- Transfer learning workflow
- Example: Cats and Dogs (from TensorFlow)
  - Data preprocessing
  - Showing example images
  - Configuring for performance
  - Data augmentation
  - Transfer learning configuration and modeling
- Exercises
  - Practice during the class and application to Case 2

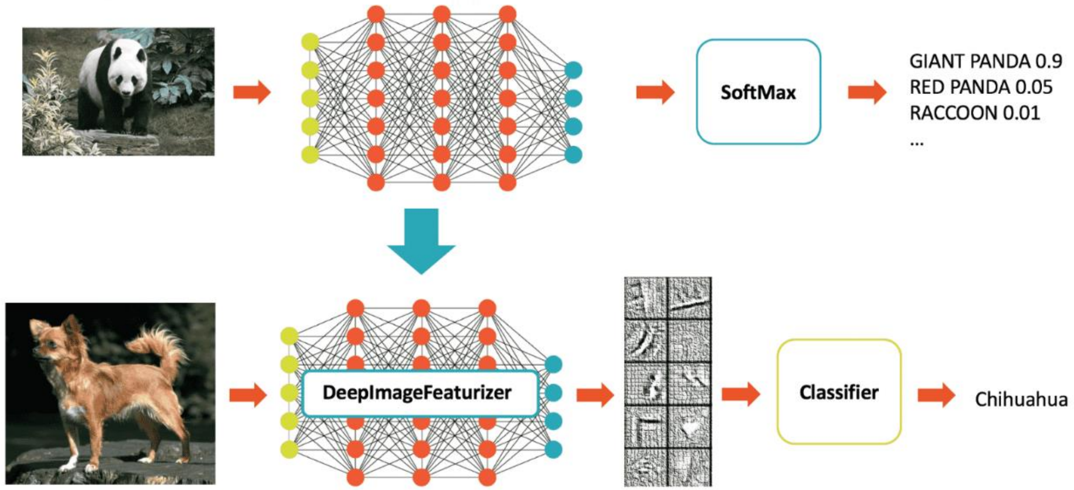
What is transfer learning?

Transfer learning consists of

1. Taking *features* learned on one problem, and
2. leveraging them on a new, similar problem.

For instance, features from a model that has learned to identify panda images may be useful to kick-start a model meant to dog images.

Transfer learning is usually done for tasks where your dataset has too little data to train a full-scale model from scratch.

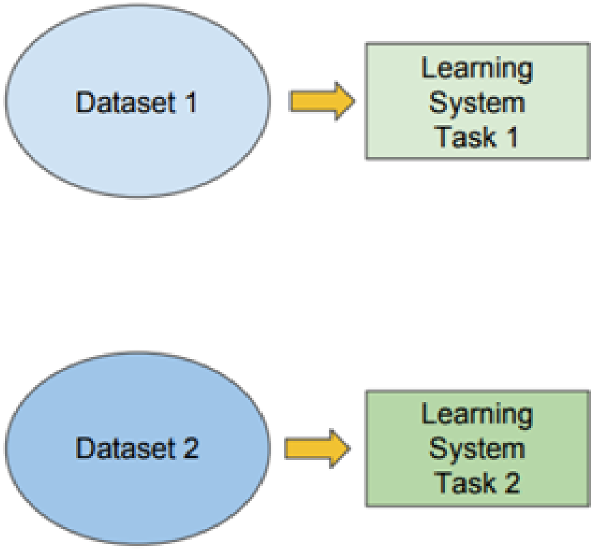


[Speed up image labeling using transfer learning]

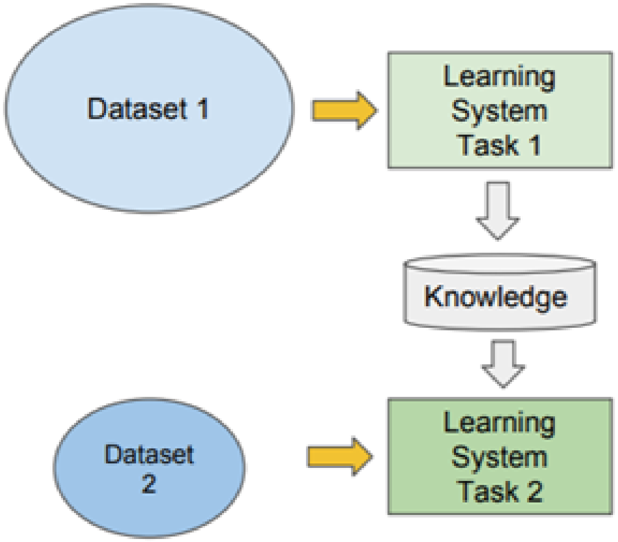
[Transfer learning and fine-tuning | TensorFlow Core]

Traditional ML vs Transfer Learning

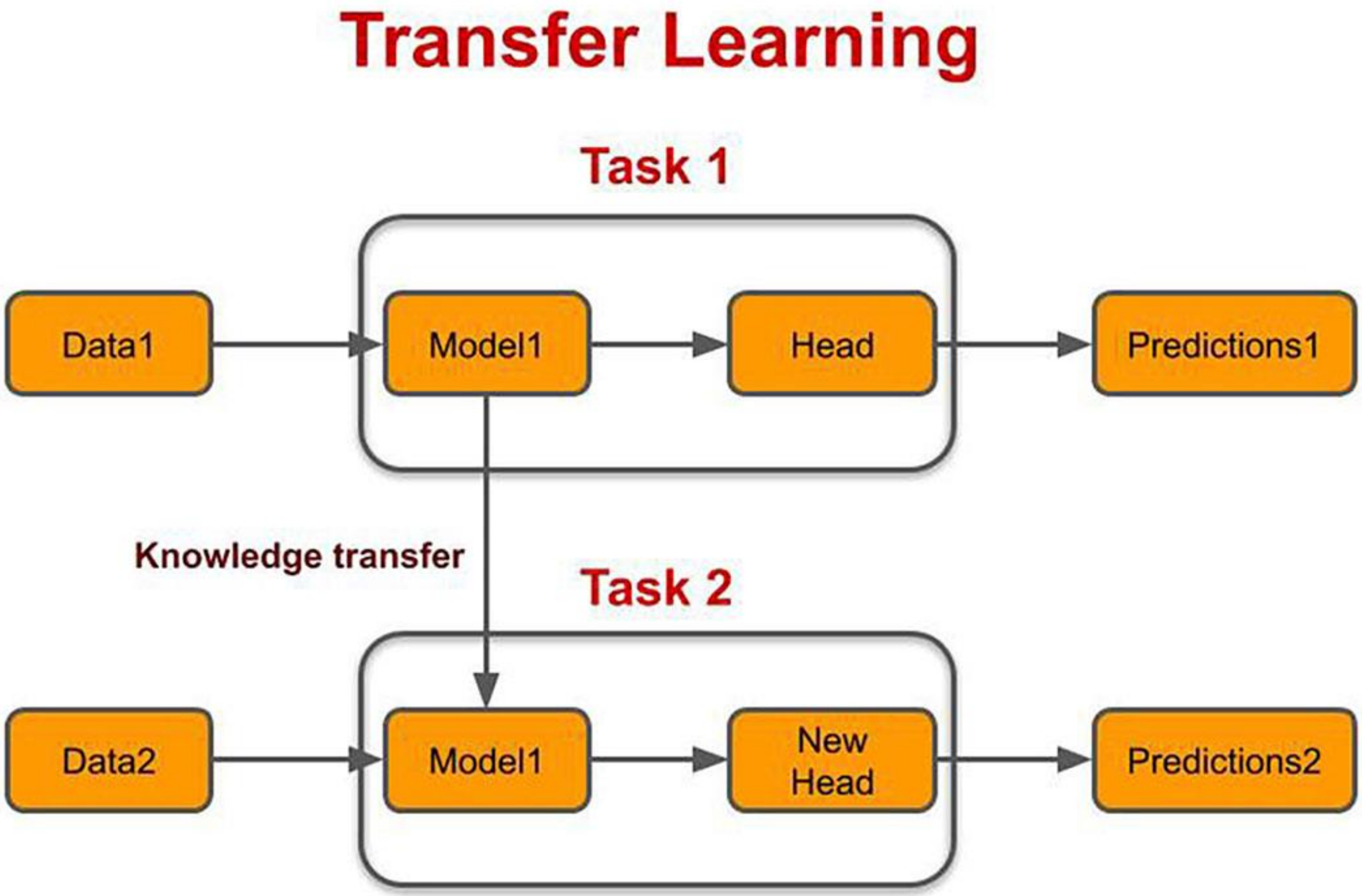
- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data



[A Comprehensive Hands-on Guide to Transfer Learning]



Transfer learning in NLP (NLP = Natural Language Processing)

Transfer Learning Workflow

The most common transfer learning workflow in the context of deep learning is the following:

1. Take layers from a previously trained model.
2. Freeze them, so as to avoid destroying any of the information they contain during future training rounds.
3. Add some new, trainable layers on top of the frozen layers. They will learn to turn the old features into predictions on a new dataset.
4. Train the new layers on your dataset.
5. (Fine-tuning)

A last, optional step, is **fine-tuning**, which consists of unfreezing the entire model you obtained above (or part of it), and re-training it on the new data with a very low learning rate.

This can potentially achieve meaningful improvements, by incrementally adapting the pretrained features to the new data.

Transfer learning and fine-tuning | TensorFlow Core

Example: Cats and Dogs

In this tutorial, you will learn how to classify images of cats and dogs by using transfer learning from a pre-trained network.

Workflow:

1. Examine and understand the data
2. Build an input pipeline
3. Compose the model
   - Load the pretrained basemodel
   - Stack the classification layers
4. Train the model
5. Evaluate the model

[Transfer learning and fine-tuning | TensorFlow Core]

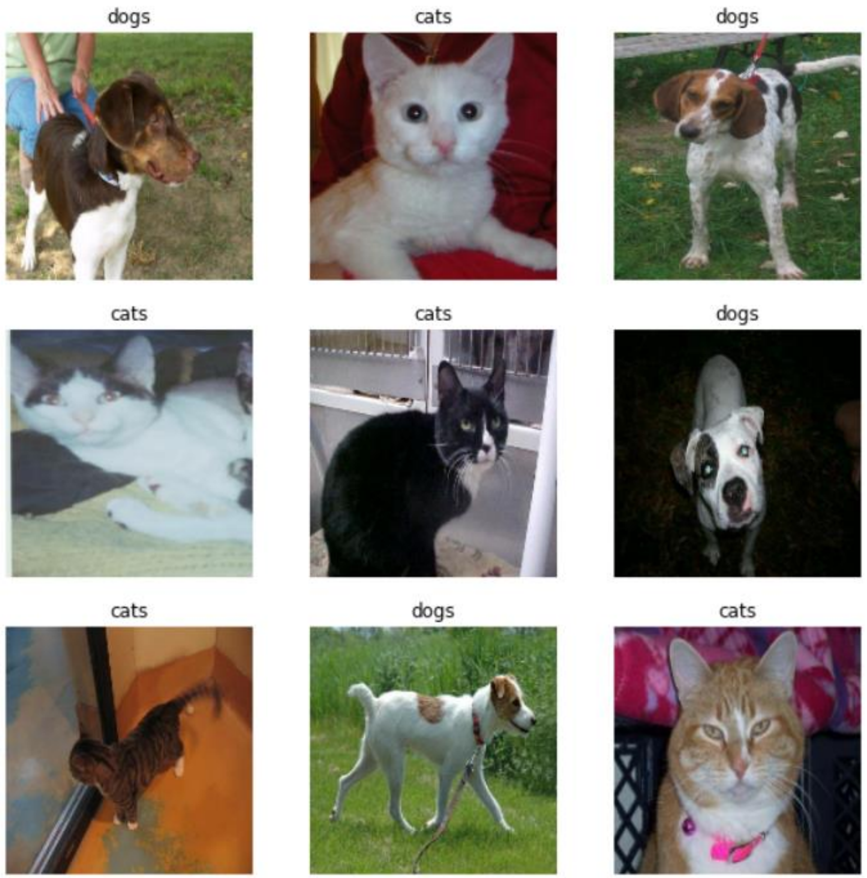
Image data preprocessing

```
_URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'
path_to_zip = tf.keras.utils.get_file('cats_and_dogs.zip', origin=_URL, extract=True)
PATH = os.path.join(os.path.dirname(path_to_zip), 'cats_and_dogs_filtered')

train_dir = os.path.join(PATH, 'train')
validation_dir = os.path.join(PATH, 'validation')

BATCH_SIZE = 32
IMG_SIZE = (160, 160)

train_dataset = image_dataset_from_directory(train_dir,
                                             shuffle=True,
                                             batch_size=BATCH_SIZE,
                                             image_size=IMG_SIZE)
```

Show some images

```
1  class_names = train_dataset.class_names
2
3  plt.figure(figsize=(10, 10))
4  for images, labels in train_dataset.take(1):
5    for i in range(9):
6      ax = plt.subplot(3, 3, i + 1)
7      plt.imshow(images[i].numpy().astype("uint8"))
8      plt.title(class_names[labels[i]])
9      plt.axis("off")
```

1. Get the class names
2. (empty line)
3. Create a figure
4. Take one batch out from the training dataset
5. Loop over the subplots
6. (until 9) Show the images



Configure the dataset for performance

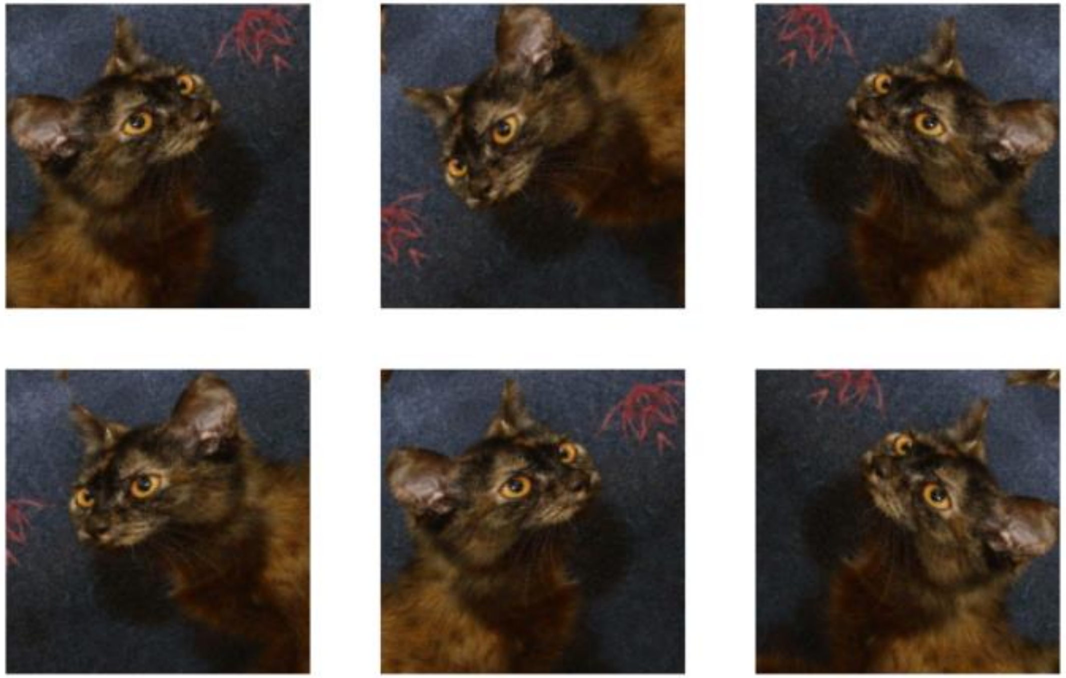
Use buffered prefetching to load images from disk without having I/O become blocking. To learn more about this method see the [data performance](#) guide.

```
AUTOTUNE = tf.data.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

WARNING: **Might not work on your laptop! If you get errors, comment out.**

Use data augmentation

```
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.2),
])
```



Flip & Rotate

Create the base model from the pre-trained convnets

You will create the base model from the **MobileNet V2** model developed at Google. This is pre-trained on the ImageNet dataset, a large dataset consisting of 1.4M images and 1000 classes.

```
# Create the base model from the pre-trained model MobileNet V2
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

Remember: You need to specify your image shape (IMG_SHAPE) and use: include_top = False

Freeze the convolutional base

It is important to *freeze* the convolutional base before you compile and train the model. Freezing (by setting layer.trainable = False) prevents the *weights* in a given layer from being *updated during training*.

```
base_model.trainable = False
```

Important note about BatchNormalization layers: for details, see the Transfer learning guide.

Base model architecture (long list …)

```
# Let's take a look at the base model architecture
base_model.summary()
```

```
Model: "mobilenetv2_1.00_160"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 160, 160, 3) 0
__________________________________________________________________________________________________
Conv1 (Conv2D)                  (None, 80, 80, 32)   864         input_1[0][0]
__________________________________________________________________________________________________
bn_Conv1 (BatchNormalization)   (None, 80, 80, 32)   128         Conv1[0][0]
__________________________________________________________________________________________________
```

…

```
__________________________________________________________________________________________________
Conv_1 (Conv2D)                 (None, 5, 5, 1280)   409600      block_16_project_BN[0][0]
__________________________________________________________________________________________________
Conv_1_bn (BatchNormalization)  (None, 5, 5, 1280)   5120        Conv_1[0][0]
__________________________________________________________________________________________________
out_relu (ReLU)                 (None, 5, 5, 1280)   0           Conv_1_bn[0][0]
==================================================================================================
Total params: 2,257,984
Trainable params: 0
Non-trainable params: 2,257,984
__________________________________________________________________________________________________
```

Add a classification head

What does global average pooling 2D mean?

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
prediction_layer = tf.keras.layers.Dense(1)
```

Build the model

Build a model *by chaining* together the data augmentation, rescaling, base_model and feature extractor layers using the [*Keras Functional API*]().

As previously mentioned, use training=False as our model contains a BatchNormalization layer.

```
inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

1. Input layer
2. Data augmentation layer
3. Preprocessing layer
4. Base model
5. Global average layer
6. Dropout layer
7. Prediction layer
8. Model

Compile the model

Compile the model before training it.

Since there are two classes, use a binary cross-entropy loss with from_logits = True since the model provides a linear output.

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(lr=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Where comes the linear output? Where it has been defined (default value of something)?

Model summary

```
model.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 160, 160, 3)]     0
_________________________________________________________________
sequential (Sequential)      (None, 160, 160, 3)       0
_________________________________________________________________
tf_op_layer_RealDiv (TensorF [(None, 160, 160, 3)]     0
_________________________________________________________________
tf_op_layer_Sub (TensorFlowO [(None, 160, 160, 3)]     0
_________________________________________________________________
mobilenetv2_1.00_160 (Functi (None, 5, 5, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dropout (Dropout)            (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 1281
=================================================================
Total params: 2,259,265
Trainable params: 1,281
Non-trainable params: 2,257,984
_________________________________________________________________
```

Can you identify the following layers?

1. Input
2. Data augmentation
3. Preprocessing
4. Base model
5. Global average
6. Dropout
7. Prediction

How many parameters are your training?

Practices

- Study the example code
  - Case 2 transfer learning
- Try other pretrained models
  - See Keras applications
- Advanced options
  - Transfer learning with TensorFlow Hub
  - TensorFlow Hub
  - TensorFlow Hub: Image Classification examples