9. TRANSFER LEARNING, EMBEDDINGS AND META-LEARNING

Stephan Robert-Nicoud
HEIG-VD/HES-SO

Credit: Andres Perez-Uribe

Objectives

- Understand how can we profit from pre-trained deep neural network models to develop new applications
- Apply the transfer learning methodology using your own data
- Understand the concept of vector embeddings
- Understand the concept of meta-learning and how to learn from few data

Microsoft’s Seeing AI app

Turns the visual world into an audible experience

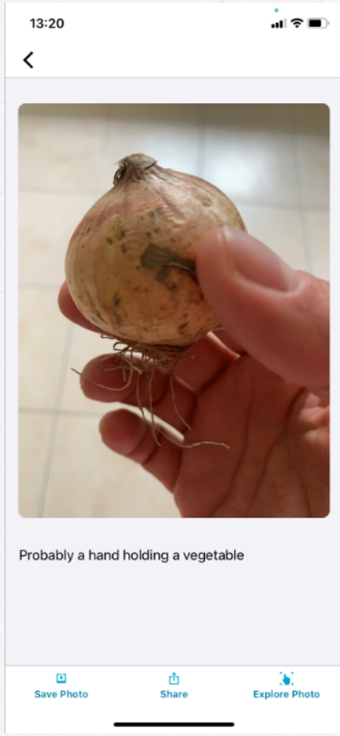
currency bills

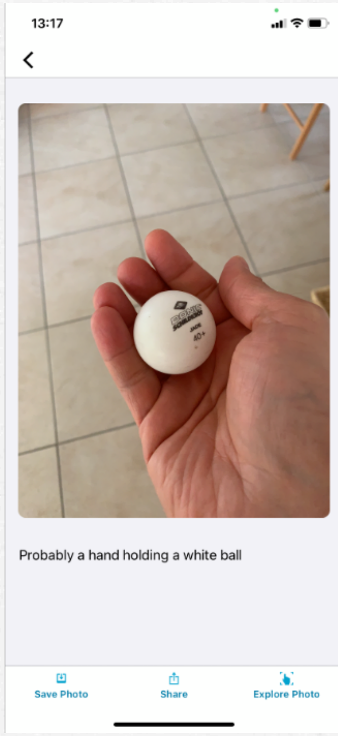
scenes & photo description

read texts

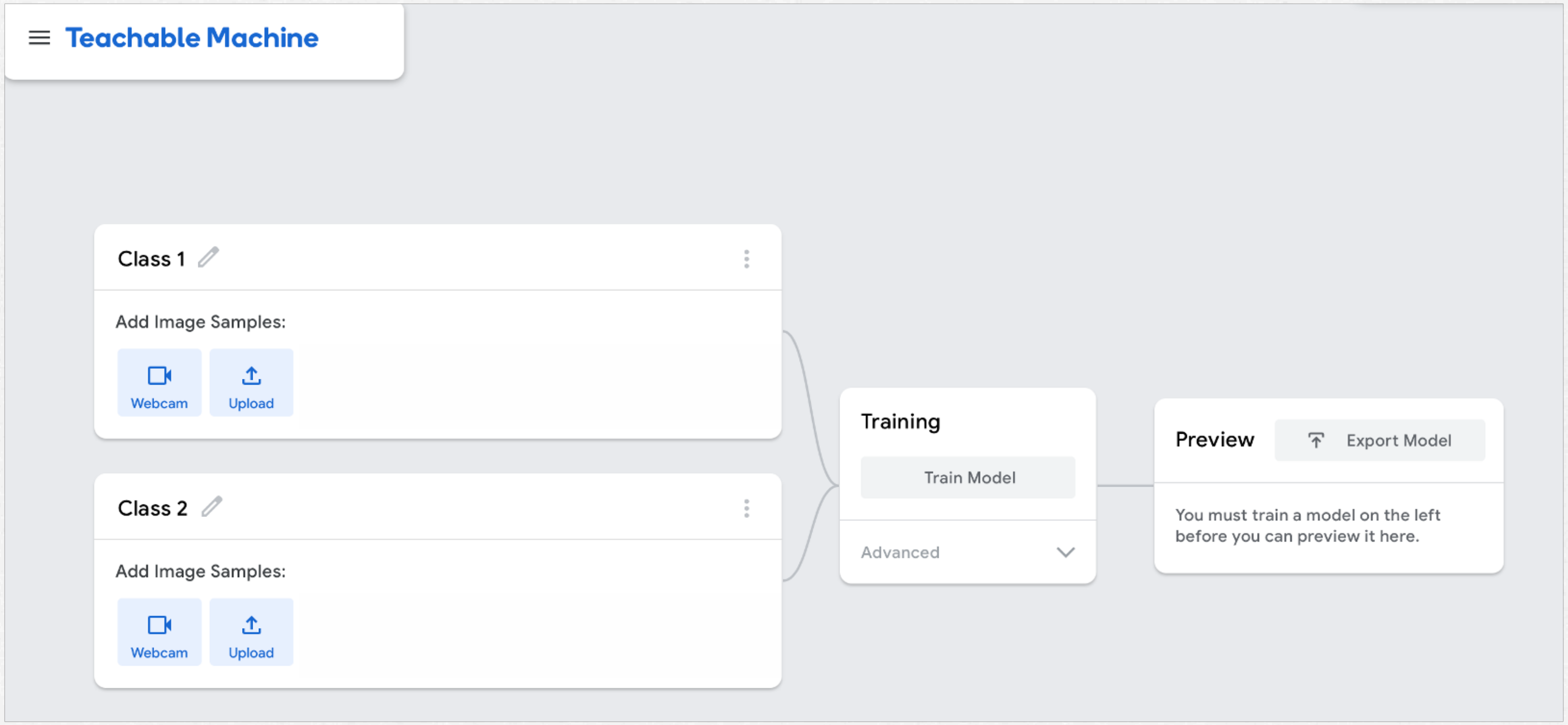
object recognition

Seeing AI object recognition





Teachable machine

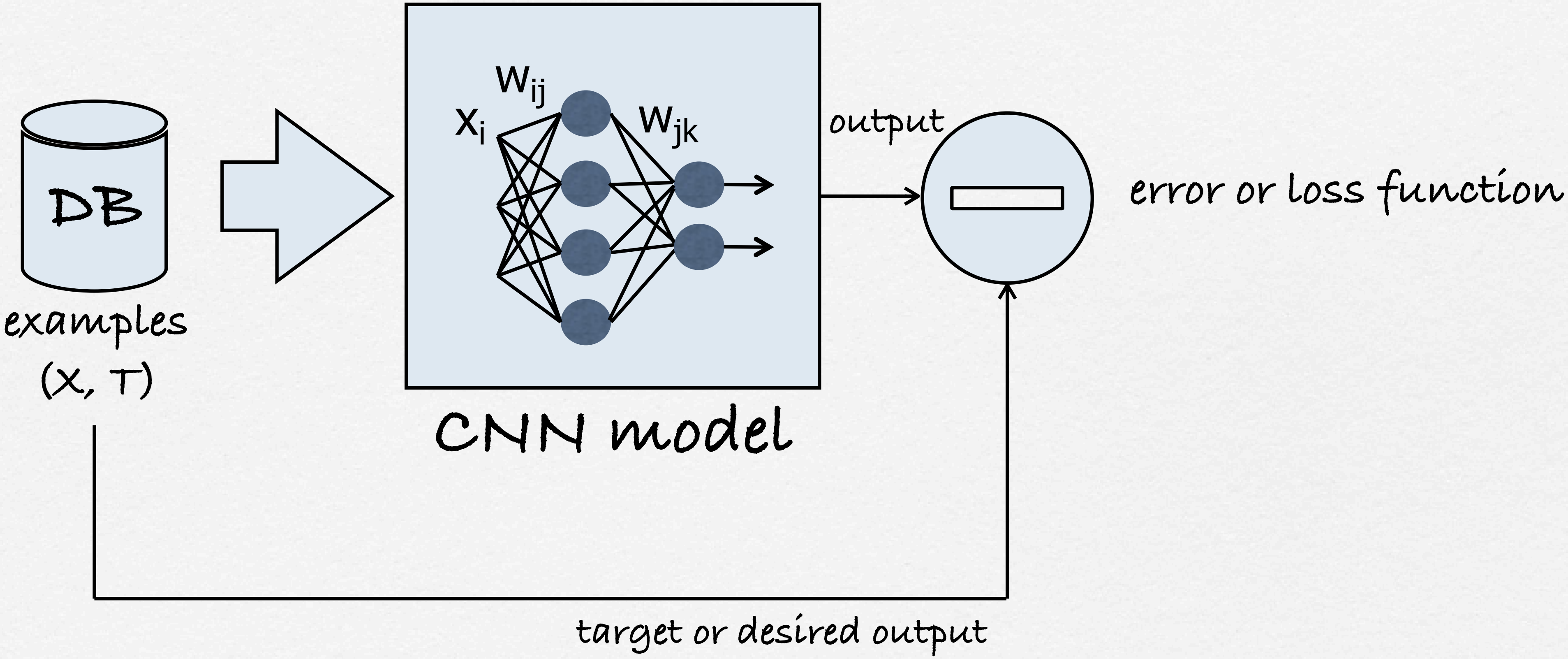


https://teachablemachine.withgoogle.com/

Three amazing observations

- The teachable machine can be trained with small data!
  - there is no need for Big Data
- The teachable machine can be trained on a standard machine
  - No GPUs are needed and the training does not take too long
- The teachable machine can work on a browser and may work on an embedded device!

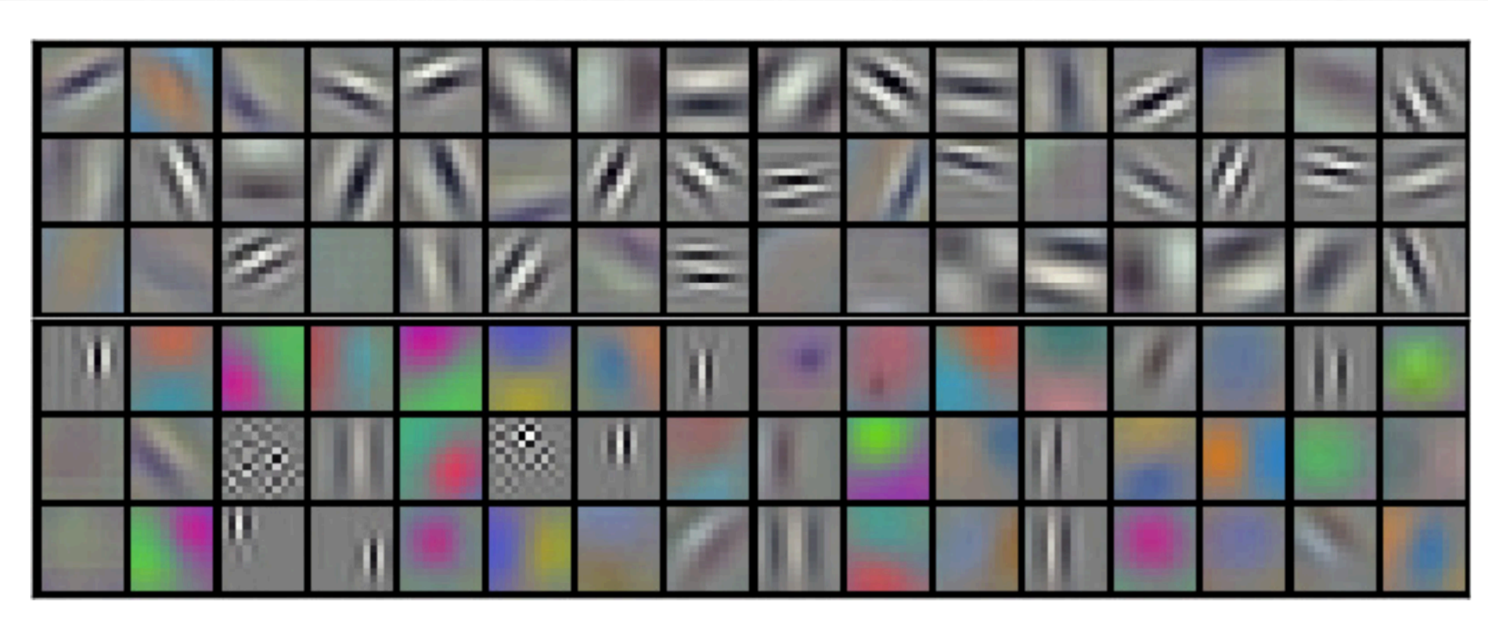
Neural Networks' data requirement



The more weights to learn the more data is necessary to avoid overfitting

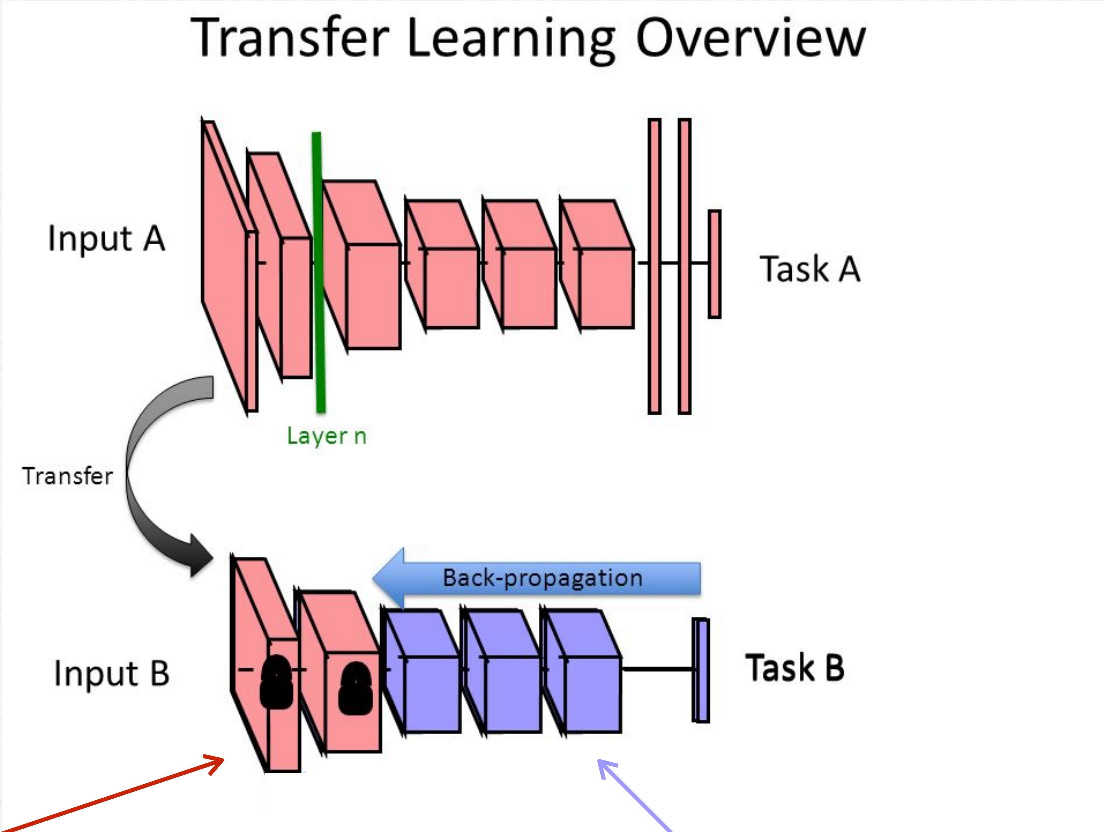
CNN learned filters

- Many CNNs learn Gabor-like filters or color blob detection in the first layers and many feature detectors obtained by training a CNN with a large database appear to be useful for other image processing tasks.



Transfer learning

- The idea is to use the first layers of a CNN that was previously trained (i.e., with lots of data) and expect to be able to fine-tune only the subsequent ones in order to use it for a new task.

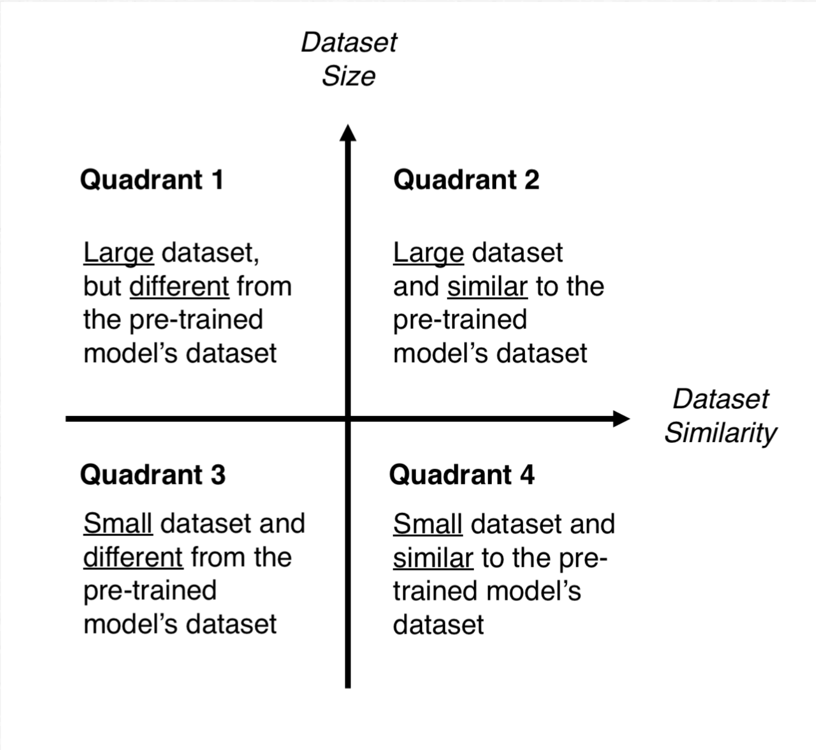


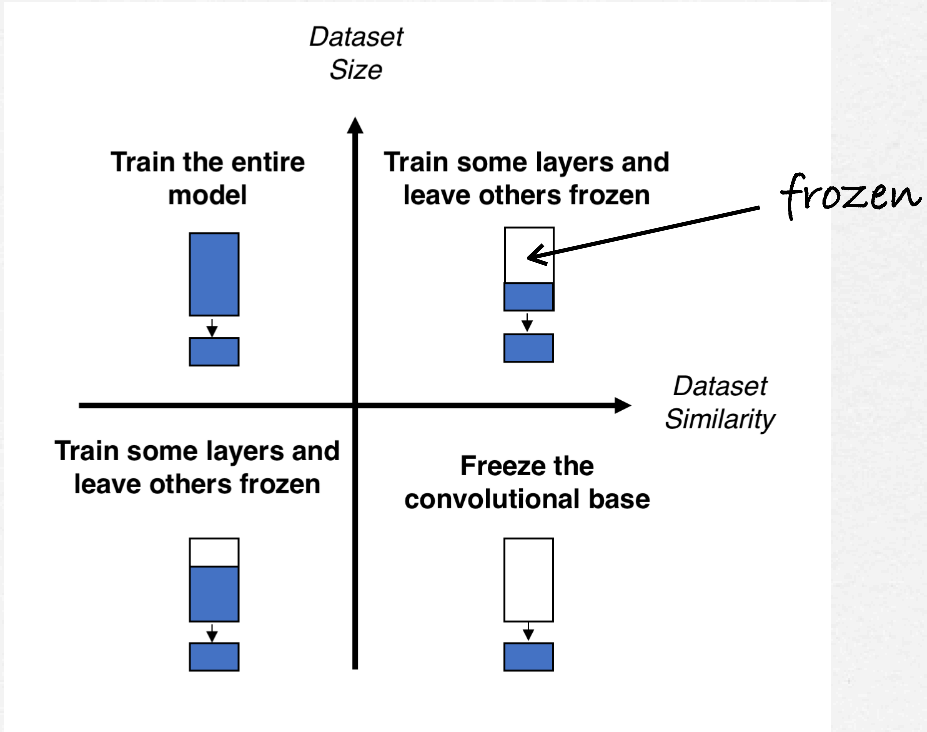
frozen weights (copied from the pre-trained model)

adapted weights

https://arxiv.org/abs/1411.1792

Quadrants of transfer learning

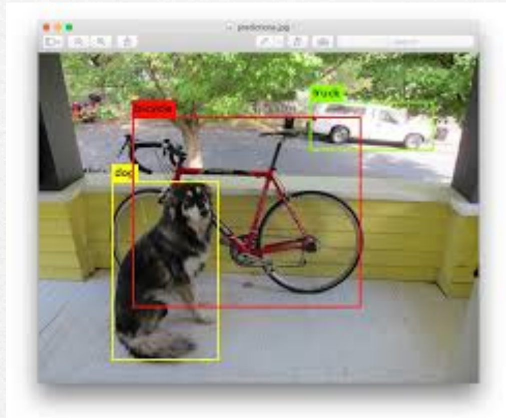




from Fei Fei Li and https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

Pre-trained CNNs

- Object recognition
    - VGGs, ResNets, Inceptions, DenseNets
    - MobileNet (light model for embedded systems)
- Face recognition
    - VGG-Face
- Object localization
    - Mask R-CNN, YOLO, SSD
- Semantic segmentation
    - U-NET
- Pose estimation
    - PoseNet, OpenPose
- X-Ray diagnosis
    - CheXNet



Some available models

Available models

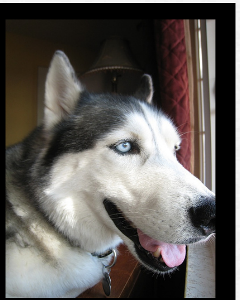
Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140.0	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1
EfficientNetB5	118	83.6%	96.7%	30.6M	312	579.2	25.3
EfficientNetB6	166	84.0%	96.8%	43.3M	360	958.1	40.4
EfficientNetB7	256	84.3%	97.0%	66.7M	438	1578.9	61.6
EfficientNetV2B0	29	78.7%	94.3%	7.2M	-	-	-
EfficientNetV2B1	34	79.8%	95.0%	8.2M	-	-	-
EfficientNetV2B2	42	80.5%	95.1%	10.2M	-	-	-
EfficientNetV2B3	59	82.0%	95.8%	14.5M	-	-	-
EfficientNetV2S	88	83.9%	96.7%	21.6M	-	-	-
EfficientNetV2M	220	85.3%	97.4%	54.4M	-	-	-
EfficientNetV2L	479	85.7%	97.5%	119.0M	-	-	-

from https://keras.io/applications/

Using a pre-trained CNN for object recognition with Keras

Let's simply read the weights of a pre-trained model (e.g., Resnet-50 trained with the ImageNet database) and use it to recognize the object in a given image:

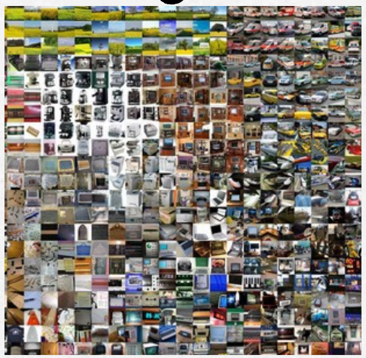
```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np

model = ResNet50(weights='imagenet')

img_path = '/ILSVRC2012_val_00005019.JPEG'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
#Predicted: [('n02109961', 'Eskimo_dog', 0.48957556), ('n02110185', 'Siberian_husky', 0.35920256), ('n02110063',
'malamute', 0.15049036)]
```



from https://keras.io/applications/

Transfer learning using MobileNet

ImageNet



1.2M images

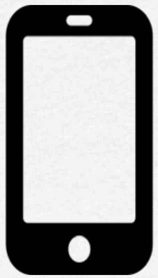
MobileNet
(pre-trained
model by
Google)

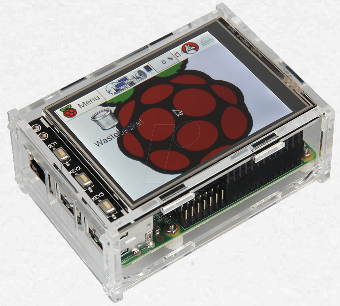
+ MyImages

fine-tuning

Customized
system

Increasing
number of
available models





Example: Tensorflow for Poets

https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/

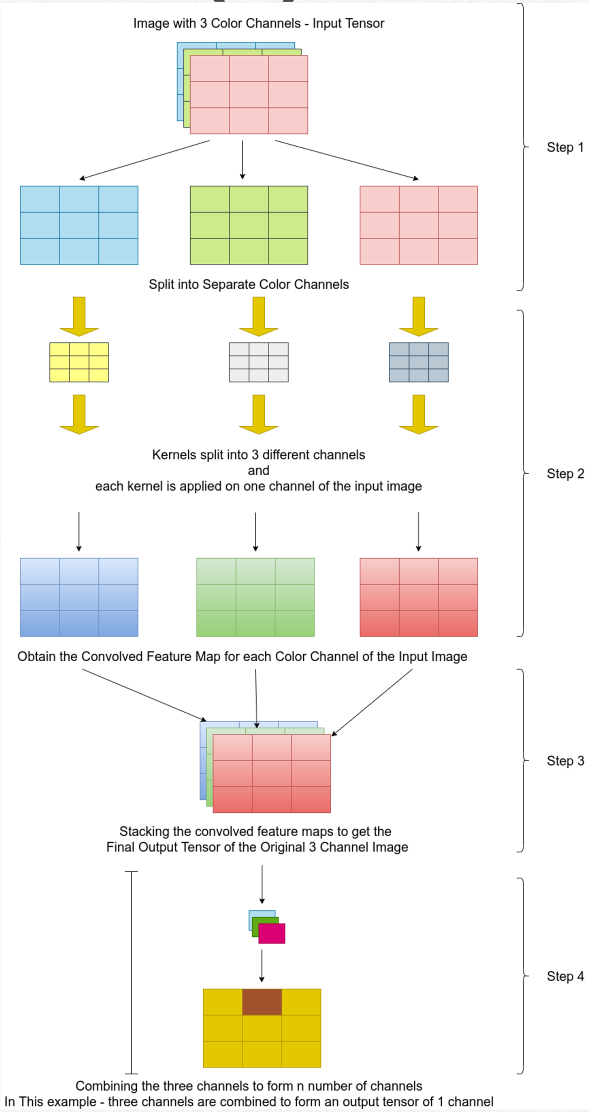
MobileNets

- MobileNet is a class of efficient models for mobile and embedded vision applications.
- Introduced by a team from Google in 2017.
- They reach comparable performances to larger architectures but using fewer parameters.

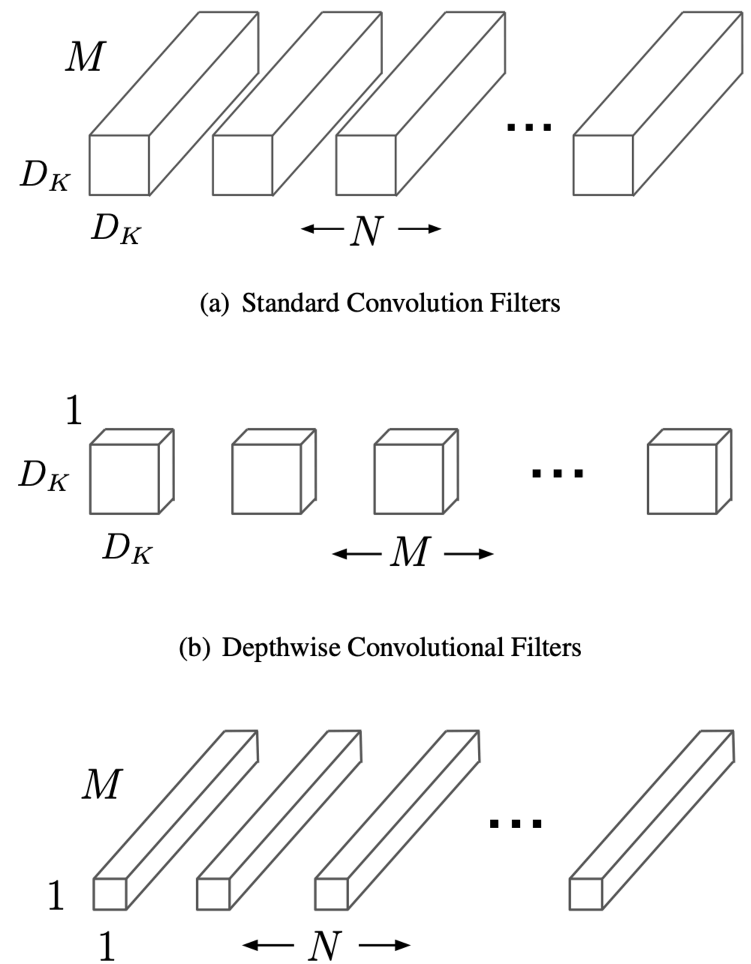
Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Convolutions in MobileNets (1)

- MobileNets use depth-wise separable convolutions to build light weight deep neural networks (e.g., less parameters).
- In general, when we processes a color image, convolutions are applied on all channels and the result is a single "image" (feature map) mixing the channels.
- In depth-wise convolutions the channels are first kept separate (steps 1-3). A 1x1 layer of convolutions is finally used to combine the multiple channels (step 4).



Convolutions in MobileNets (2)



(a) Standard Convolution Filters

(b) Depthwise Convolutional Filters

(c) 1 × 1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

- Example: suppose a convolution layer based on N 3x3 filters (Dk=3) and processing an RGB image (M=3) of size HxW
- The normal convolutions require HxWx(DkxDk)xMxN mult-adds or HxWx27xN
- Depthwise convolutions require HxWx(DkxDk)xM + HxWxMxN mult-adds or HxWx27+HxWx3xN

from https://arxiv.org/pdf/1704.04861v1.pdf

Typical transfer learning process

- Identify the pre-trained model you would like to use
- Load the model and its weights
- Modify the last layers (drop original output layer and replace it by dense layers and an output that matches the number of classes of the new task)
- Freeze the first layers and set the last one to “trainable”.
- Re-compile the new model, train and evaluate.

HEIG VD

Transfer learning using Keras (1)

The following example defines a new model based on the MobileNet architecture taking all but the last layer. It computes the average of the features computed with the convolutional layers (e.g., using a layer GlobalAveragePooling2D), it adds a Dense layer (1024 neurons) and defines a new input for a 3 classes problem using a softmax activation function.

```
from tensorflow.keras.applications.mobilenet import MobileNet
from tensorflow.keras.applications.mobilenet import preprocess_input, decode_predictions
from keras.layers import Dense,GlobalAveragePooling2D

base_model=MobileNet(weights='imagenet',include_top=False) #imports the mobilenet model and discards
the last 1000 neuron layer.

x=base_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x) #we add dense layers so that the model can learn more complex
functions and classify for better results.
predictions=Dense(3,activation='softmax')(x) #final layer with softmax activation

new_model=Model(inputs=base_model.input,outputs=predictions)
```

from https://towardsdatascience.com/transfer-learning-using-mobilenet-and-keras-c75daf7ff299

Transfer learning using Keras (2)

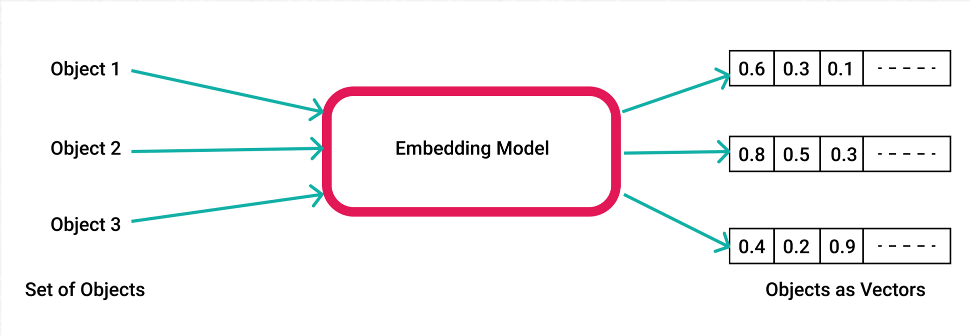
The following code prints the layers composing the new model defined in the previous slide. The second part of the code sets the first 87 layers to "non-trainable" (we also say that we freeze that part of the model) and sets the final two Dense layers to trainable. Finale we compile the new model and train it with the new data.

```
for i,layer in enumerate(new_model.layers):
  print(i,layer.name)

# Freeze the first 87 layers
for layer in new_model.layers[:87]:
    layer.trainable=False
for layer in new_model.layers[87:]:
    layer.trainable=True

# Compile the new model
new_model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])

# Fine-tune the new model
new_model.fit(new_train_data, epochs=epochs, validation_data=validation_new_data)
```

from https://towardsdatascience.com/transfer-learning-using-mobilenet-and-keras-c75daf7ff299

Vector Embeddings (1)

- One of the most fascinating concepts in ML: any object (image, text document, sound, etc) can be reduced to a vector of numerical values, which we can consider to be features of those objects.



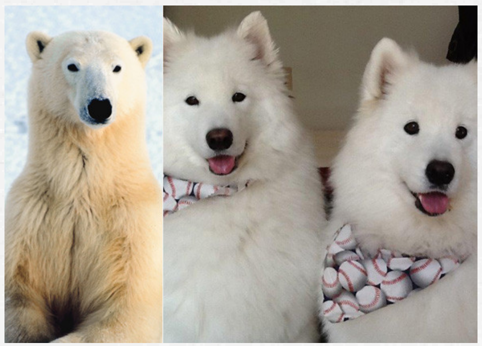
- For example, the output of the convolutional part of a CNN computes a vector that "characterizes" the input image. That output can be used as a vector embedding.

 https://www.pinecone.io/learn/vector-embeddings/

Vector Embeddings (2)

- Something special about vectors that makes them so useful is that such a representation makes it possible to translate semantic similarity as perceived by humans to proximity in a vector space.
- We expect that similar images produce similar embeddings or feature vectors and that different objects do produce different vectors.

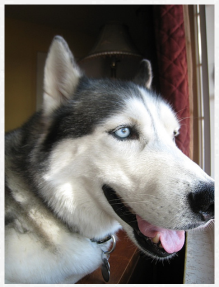


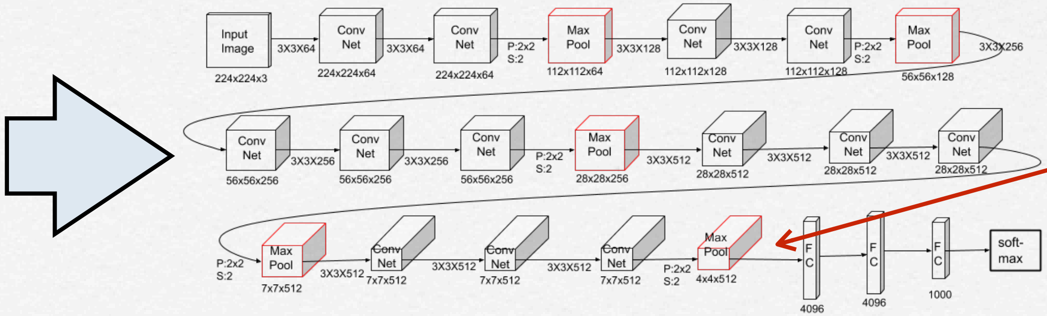


Using a pre-trained CNN for object characterization with Keras

This example uses a pre-trained VGG-16 model (using the ImageNet database) to compute a vector of features from an image.

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input
import numpy as np

model = VGG16(weights='imagenet', include_top=False)

img_path = '/ILSVRC2012_val_00005019.JPEG'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

features = model.predict(x)
embedding = GlobalAveragePooling2D()(features)
```





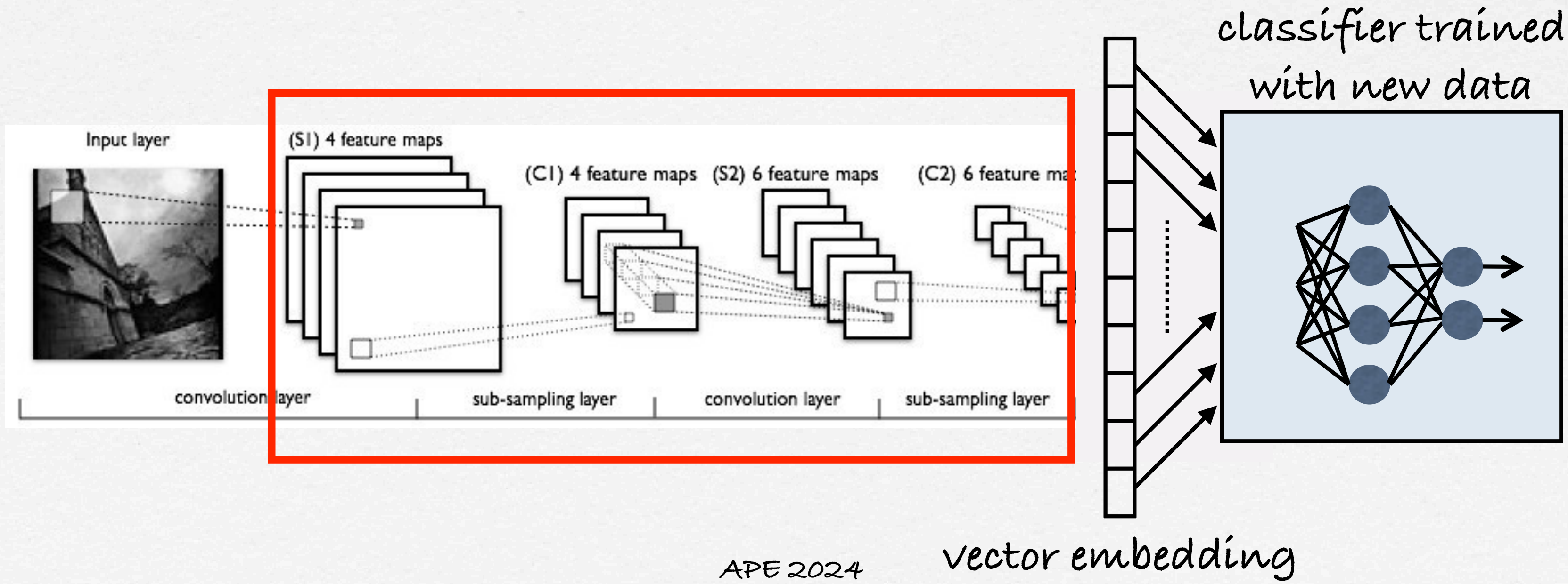
features is a vector of 7x7x512 values

embedding is a vector of 512 values

from https://keras.io/applications/

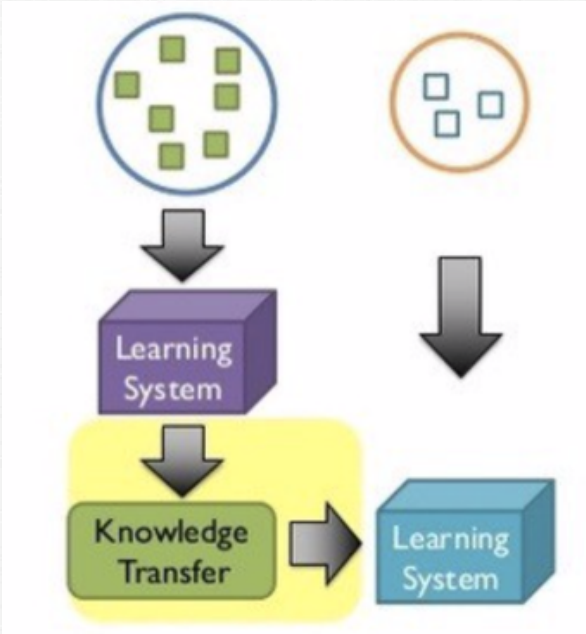
Embeddings for transfer learning

- An alternative way of performing transfer learning consists on using a pre-trained model to compute vector embeddings from the input data and using those vectors as inputs to a new model (which can be any sort of ML model, e.g., K-NN) that is trained using the new data.



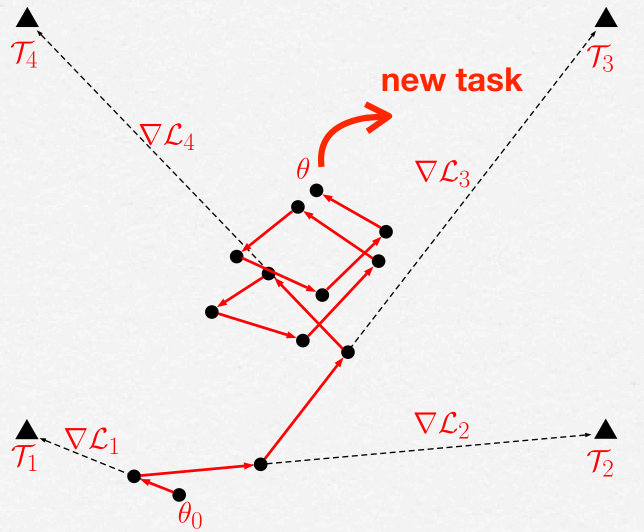
Few-shot learning

- We refer to few-shot learning when our training set contains very few examples.
- One way to deal with such a problem is by using meta-learning, which aims to improve learning across different tasks or datasets instead of specializing on a single one.
- The idea goes like this: train a model to solve different tasks and expect it to be almost ready to solve a new one.



Meta learning

- The idea is to train a model for a certain number of epochs on a variety of learning tasks (τ_1, τ_2,...), such that at the end, it can solve a new learning task using only a small number of training samples.

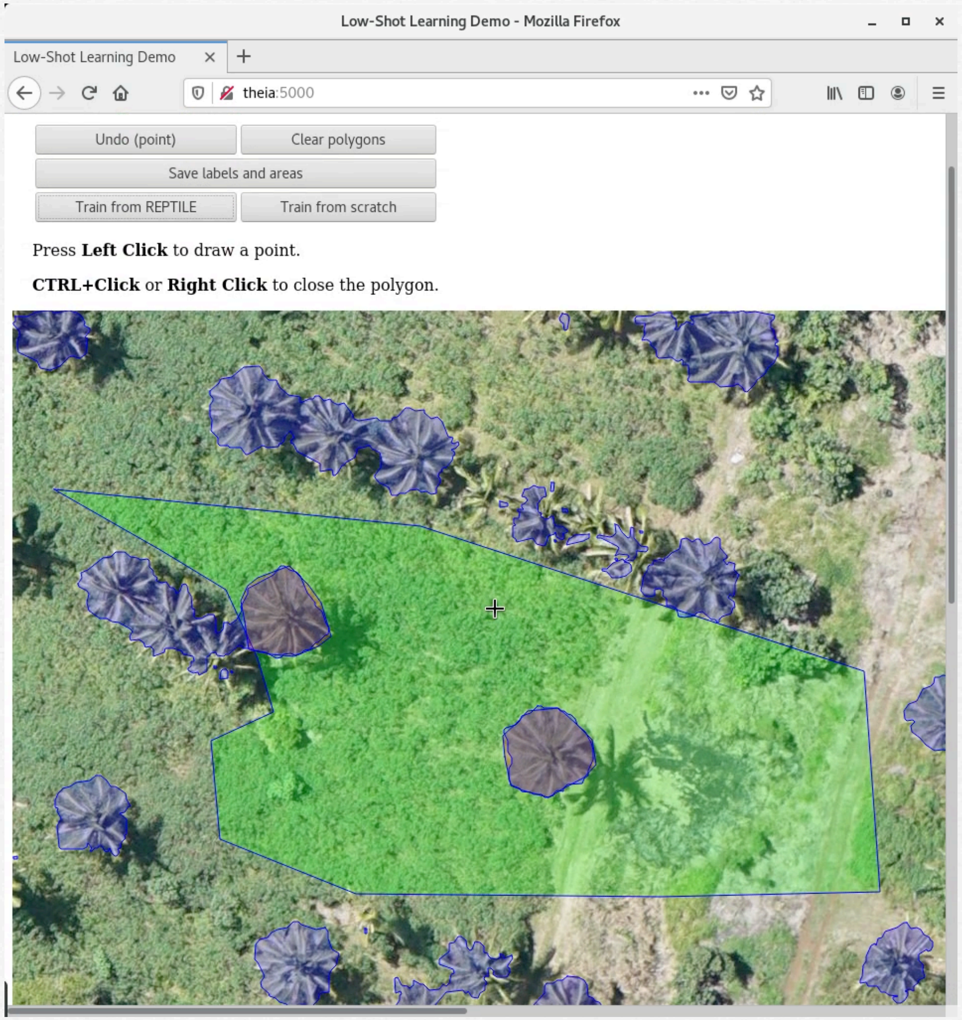


- A simple algorithm that implements this approach is called REPTILE and was developed by OpenAI.

figure from Satizabal and Perez-Uribe (2021)

http://doi.org/10.1007/978-3-030-85030-2_42

Few-shot learning DEMO using REPTILE



Developed at HEIG-VD in collaboration with Picterra S.A.; funded by the Swiss Space Center