Advanced Machine Learning

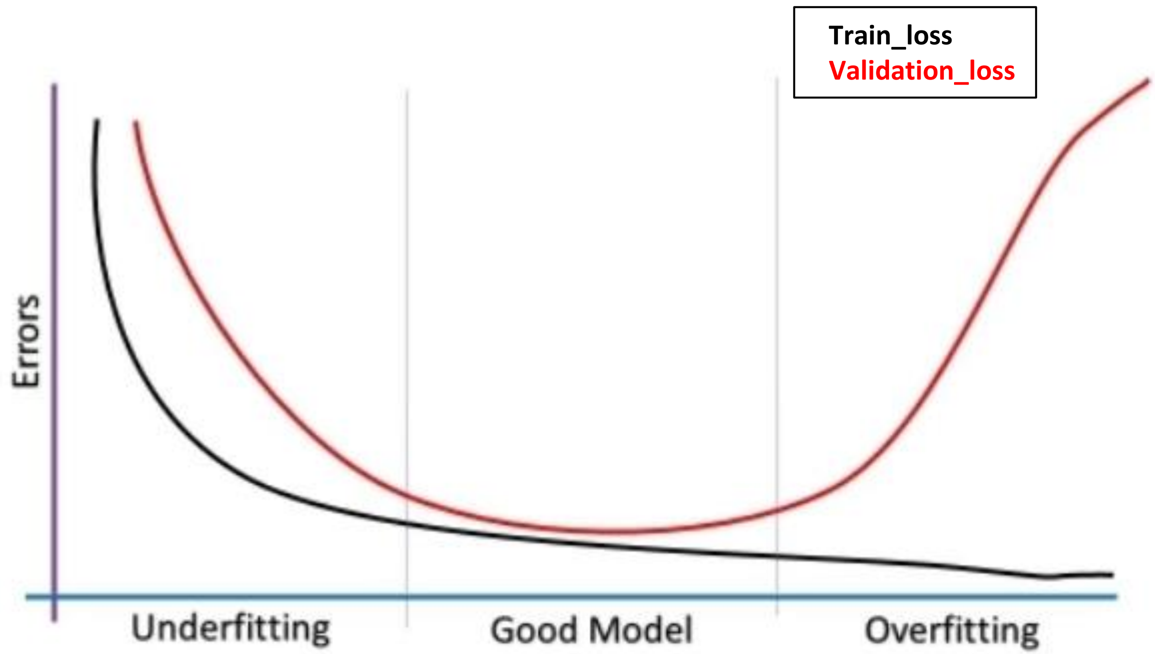
Bilel GUETARNI, PhD

bilel.guetarni@junia.com

JUNIA ISEN

PowerPoint slides from **Halim Benhabiles, PhD** and **Farès BOUGOURZI, PhD**

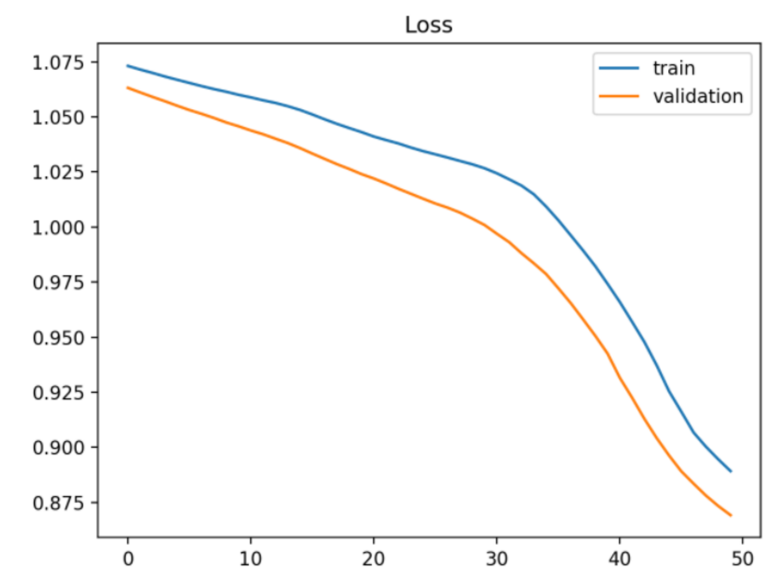
Analyzing the behavior of your trained model using Keras – TensorFlow core



https://www.baeldung.com/cs/learning-curve-ml

Analyzing the behavior of your trained model using Keras – TensorFlow core

Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set.

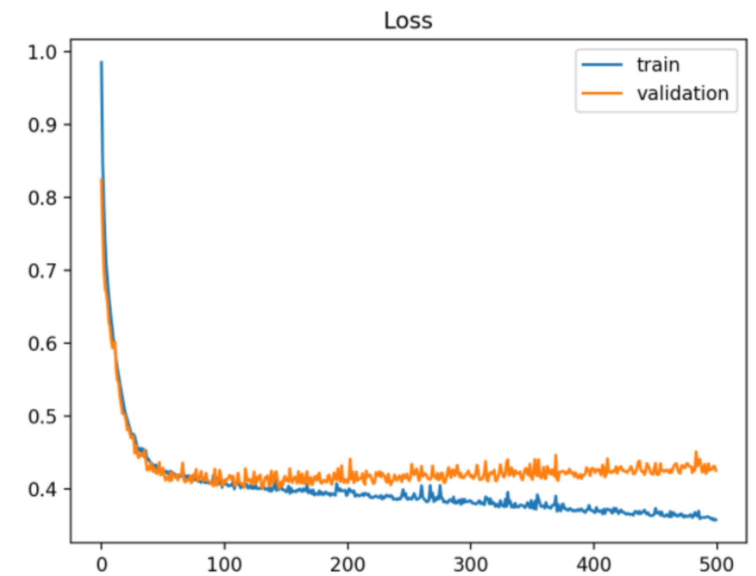


Example of Training Learning Curve Showing an Underfit Model That Requires Further Training

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Analyzing the behavior of your trained model using Keras – TensorFlow core

> *… fitting a more flexible model requires estimating a greater number of parameters. These more complex models can lead to a phenomenon known as overfitting the data, which essentially means they follow the errors, or noise, too closely.*



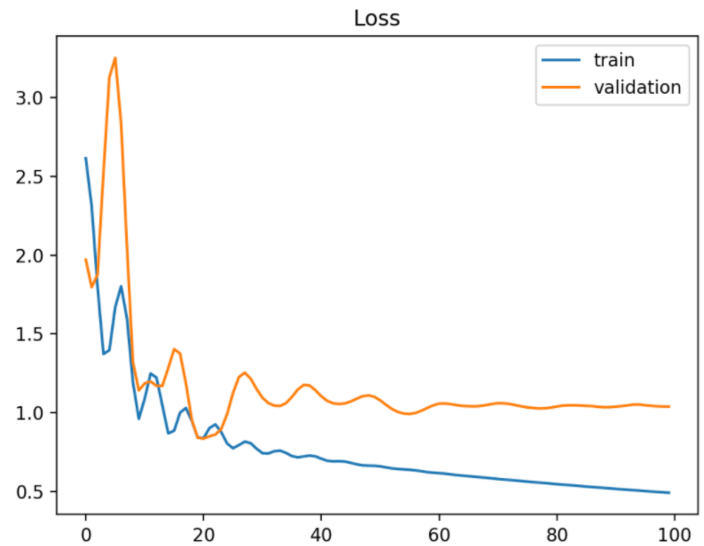
Example of Train and Validation Learning Curves Showing an Overfit Model

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Analyzing the behavior of your trained model using Keras – TensorFlow core

Unrepresentative Train Dataset

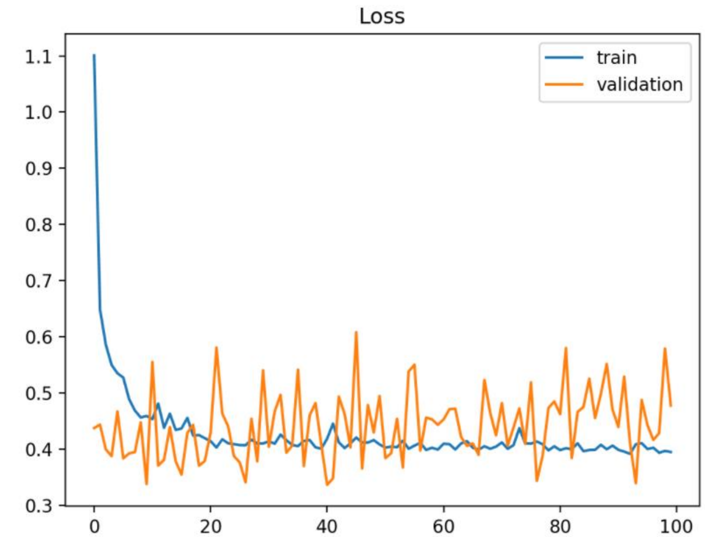
loss improvement, but a large gap remains between both curves



Example of Train and Validation Learning Curves Showing a Training Dataset That May Be too Small Relative to the Validation Dataset

Unrepresentative Validation Dataset

validation loss that shows noisy movements around the training loss



Example of Train and Validation Learning Curves Showing a Validation Dataset That May Be too Small Relative to the Training Dataset

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

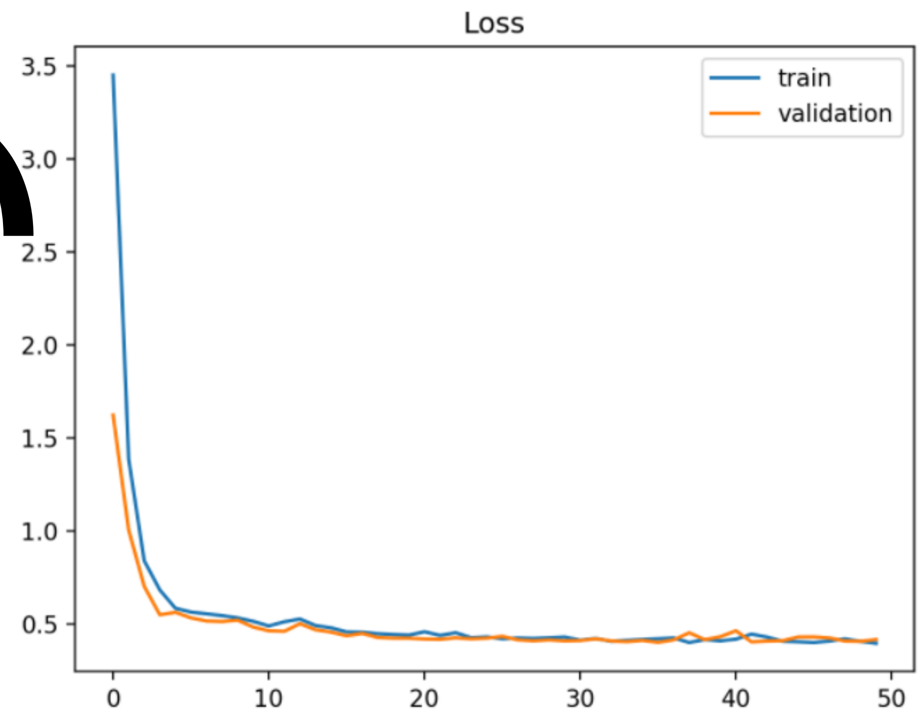
Analyzing the behavior of your trained model using Keras – TensorFlow core

A plot of learning curves shows a good fit if:

- The plot of training loss decreases to a point of stability.
- The plot of validation loss decreases to a point of stability and has a small gap with the training loss.

Continued training of a good fit will likely lead to an overfit.

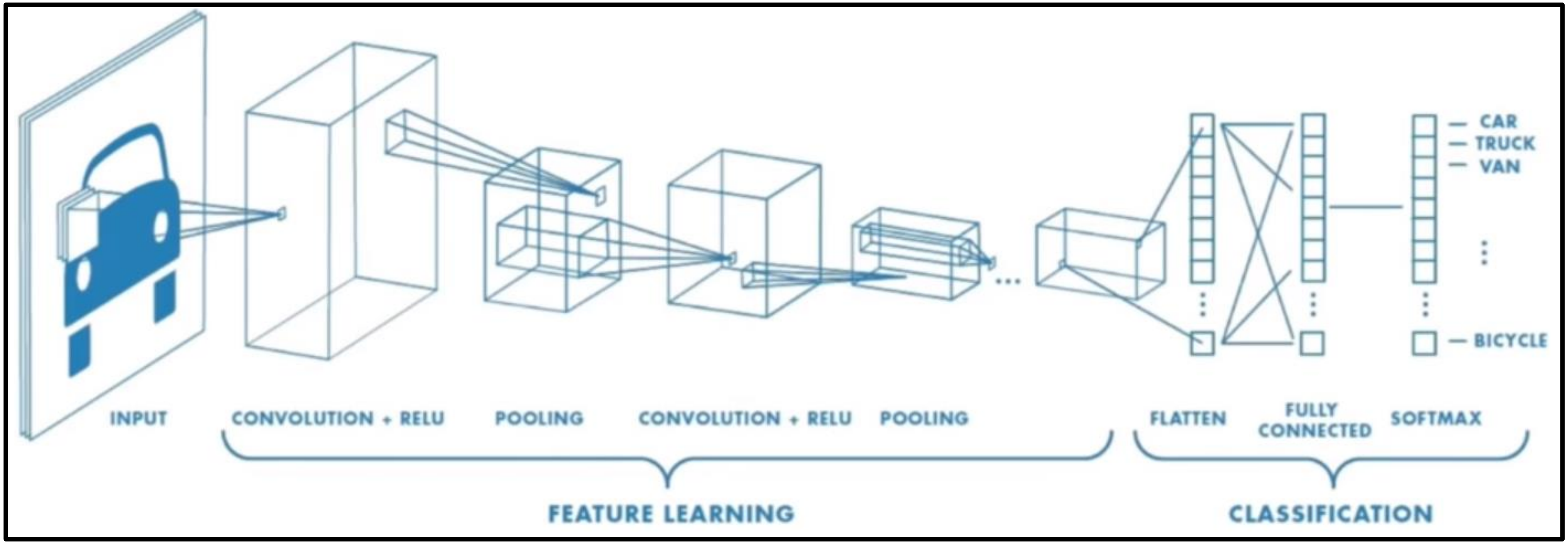
Model capacity improved by
- Transfer learning
- Data augmentation



Example of Train and Validation Learning Curves Showing a Good Fit

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Transfer learning

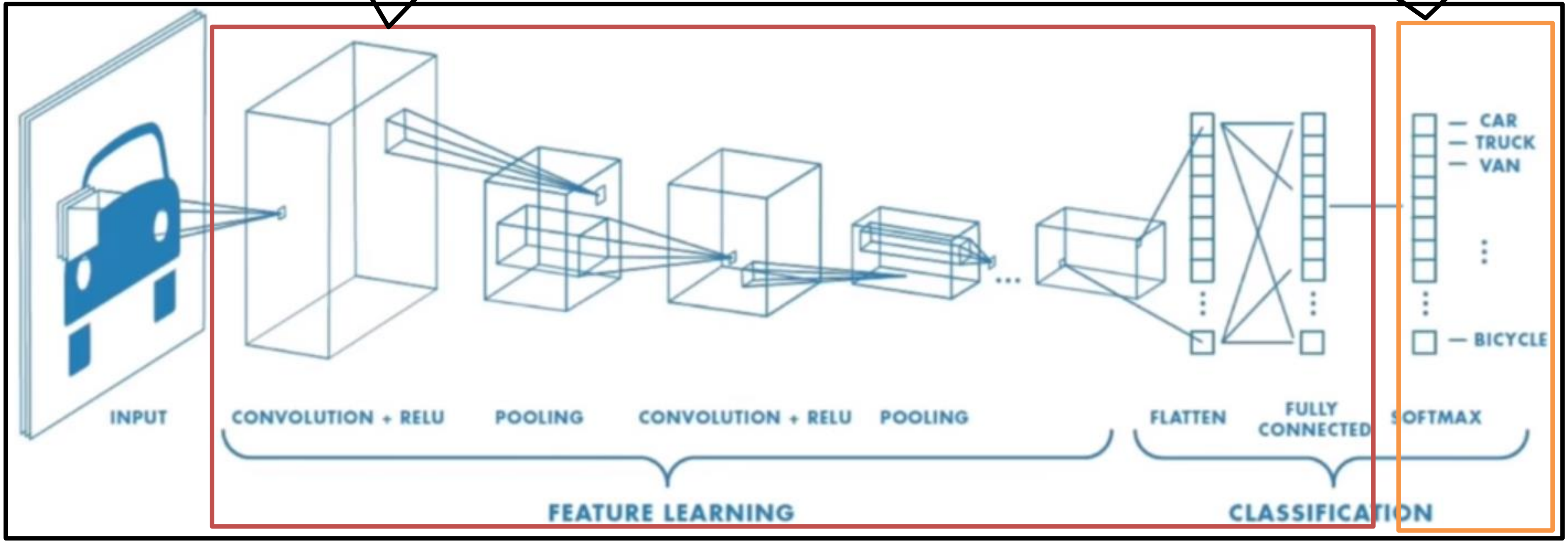


Model trained from scratch
Image extracted from
https://www.youtube.com/watch?v=H-HVZJ7kGI0

Transfer learning

Freeze some layers of the model with best parameters. Already pre-trained on a large dataset

Set the new size of the last layer to the new problem



Train the adapted model (pre-trained) on the new dataset

Transfer learning strategies

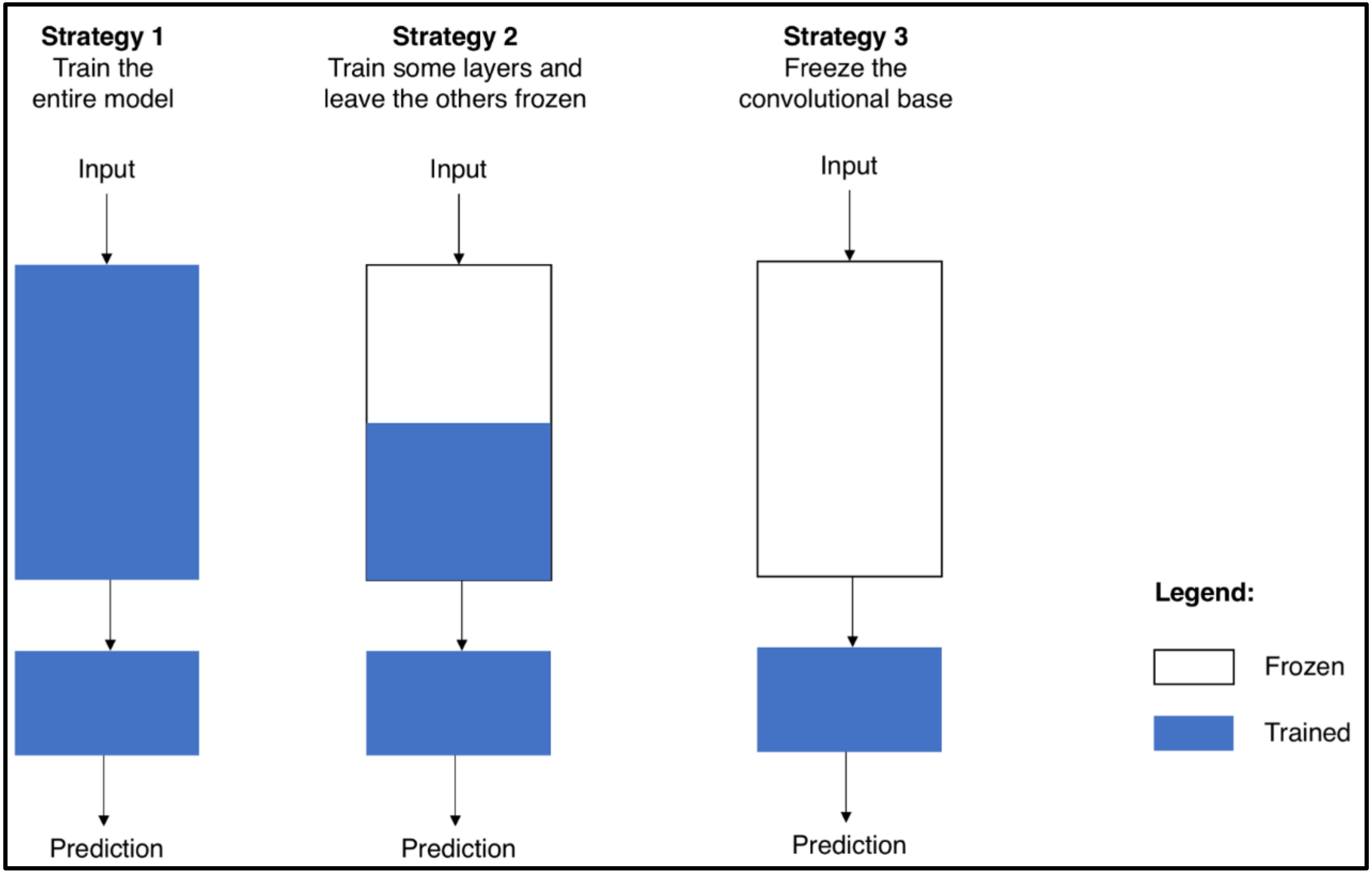


Figure extracted from
https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

Transfer learning – similarity matrix and strategy choice

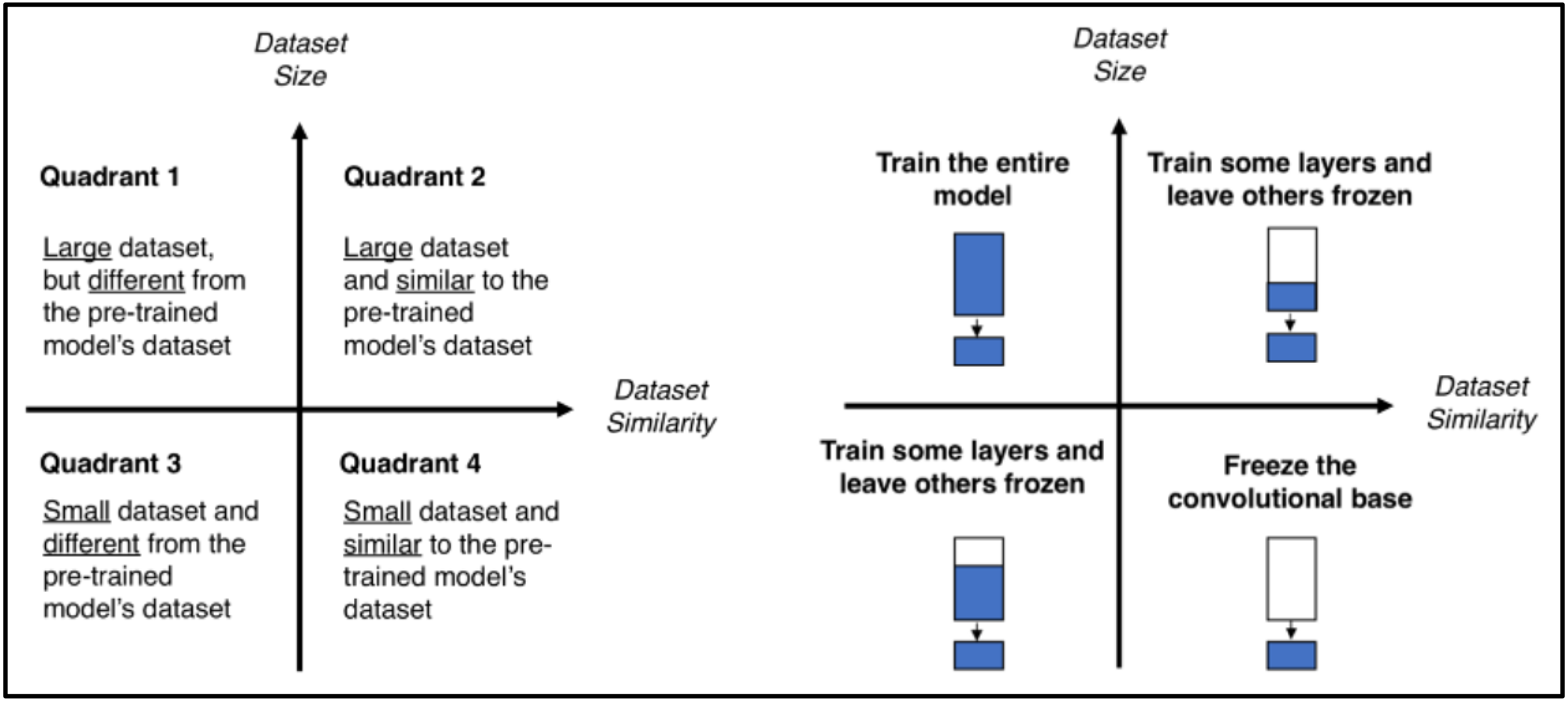


Figure extracted from
https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

Transfer learning

Parameters	Training from scratch	Transfer Learning
Method	Build CNN from scratch	Only last few layers need to be trained
Tuning	Need to tune large number of hyperparameters	Only a few hyperparameters need to be tuned
Computation	Large computation power is required (multiple GPUs)	Less computation power needed (can even work with CPUs)
Dataset	Huge dataset needed to avoid overfitting	A small dataset is enough
Training time	May take weeks or even months	May take hours to train

Figure extracted from
https://www.slideshare.net/RuchaGole/understanding-cnn

Transfer learning with models pre-trained on ImageNet

- Classification

Over 15 million labelled high-resolution images with 22000 categories

ImageNet Challenge

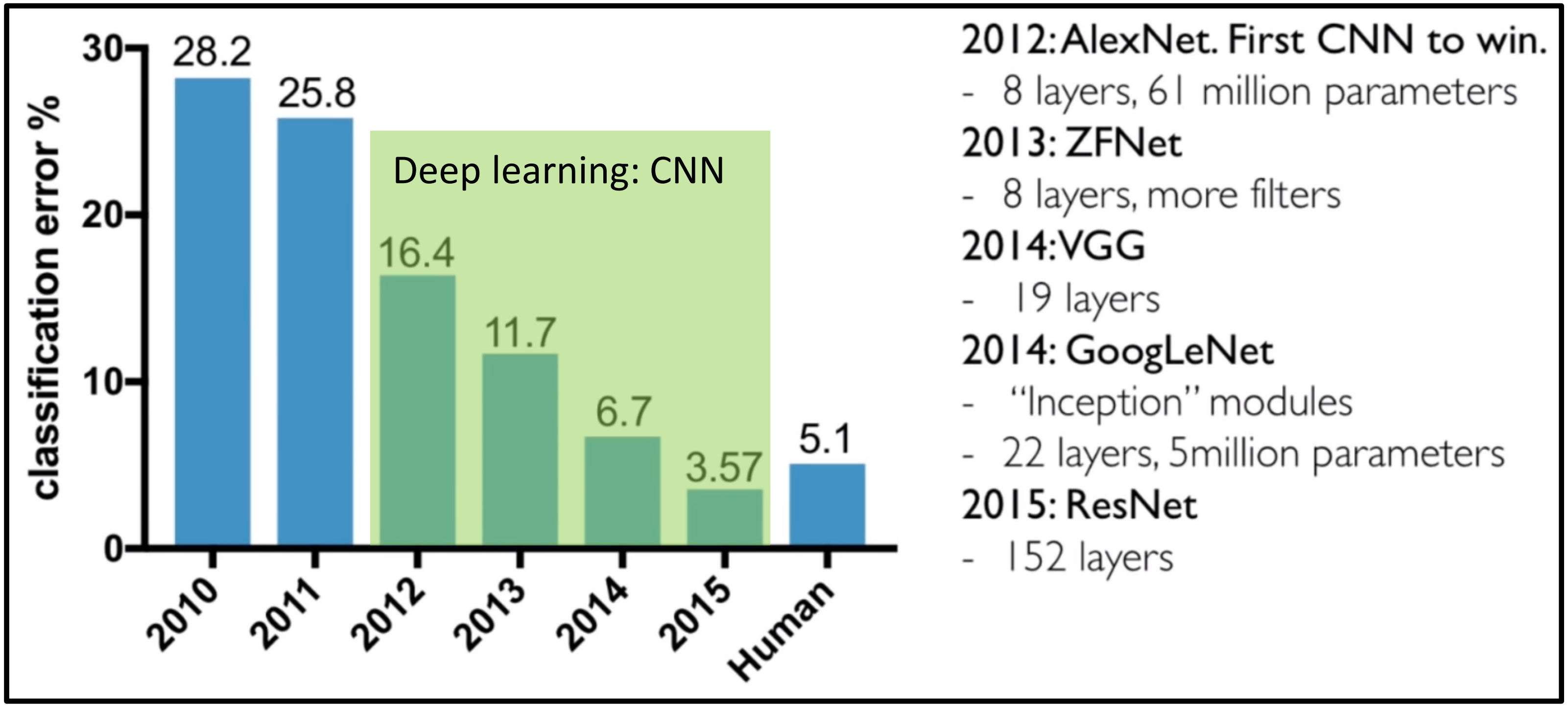
http://www.image-net.org/challenges/LSVRC/

A subset of ImageNet with 1000 categories and 1.2 million training images

Image extracted from
https://www.youtube.com/watch?v=H-HVZJ7kGI0

CNNs applications

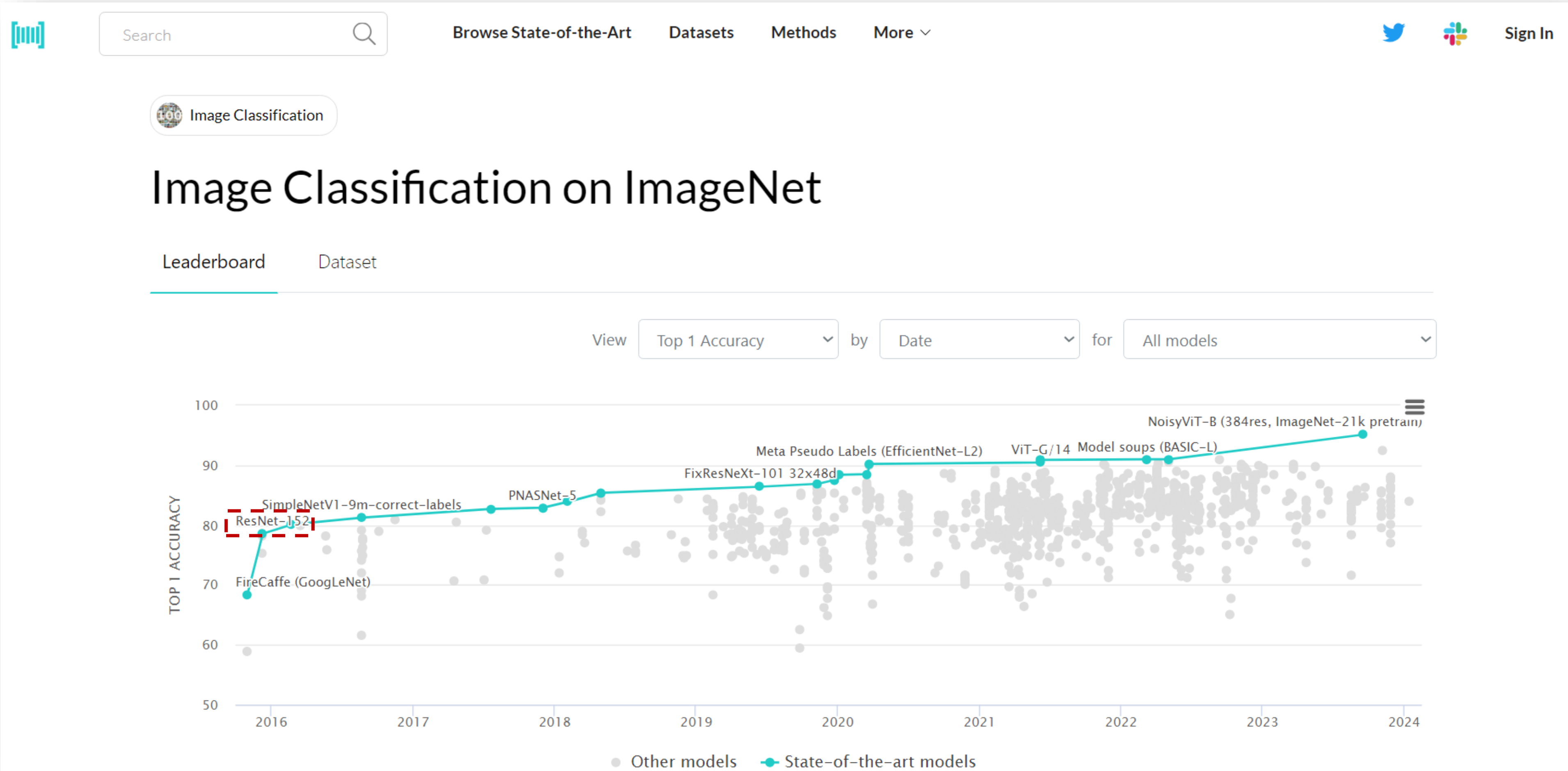
- Classification



5 top error rate (category miss predicted in the 5 top ones).
Image extracted from
https://www.youtube.com/watch?v=H-HVZJ7kGI0

CNNs applications

- Classification



Technical example of transfer learning - https://keras.io/api/applications/

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

Technical example of transfer learning - https://keras.io/api/applications/

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
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	-	-	-
ConvNeXtTiny	109.42	81.3%	-	28.6M	-	-	-
ConvNeXtSmall	192.29	82.3%	-	50.2M	-	-	-
ConvNeXtBase	338.58	85.3%	-	88.5M	-	-	-
ConvNeXtLarge	755.07	86.3%	-	197.7M	-	-	-
ConvNeXtXLarge	1310	86.7%	-	350.1M	-	-	-

For a detailed coverage of modern CNN architectures:

8. Modern Convolutional Neural Networks — Dive into Deep Learning 1.0.3 documentation (d2l.ai)

Neural Network Architectures. Deep neural networks and Deep Learning… | by Eugenio Culurciello | Towards Data Science

Technical example of transfer learning - https://keras.io/api/applications/

Usage examples for image classification models

Classify ImageNet classes with ResNet50

```
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 = 'elephant.jpg'
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: [(u'n02504013', u'Indian_elephant', 0.82658225), (u'n01871265', u'tusker', 0.1122
```

Technical example of transfer learning - https://keras.io/api/applications/

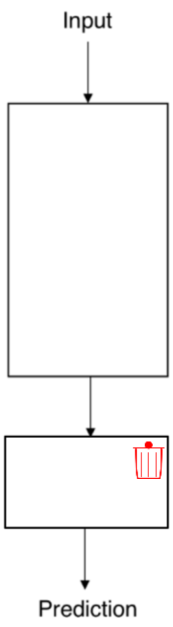
Fine-tune InceptionV3 on a new set of classes

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False)

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)
# and a logistic layer -- let's say we have 200 classes
predictions = Dense(200, activation='softmax')(x)

# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)
```



Technical example of transfer learning - https://keras.io/api/applications/

Fine-tune InceptionV3 on a new set of classes

```python
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False)

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)
# and a logistic layer -- let's say we have 200 classes
predictions = Dense(200, activation='softmax')(x)

# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)
```

Technical example of transfer learning - https://keras.io/api/applications/

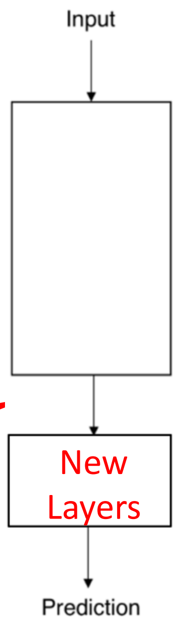
Fine-tune InceptionV3 on a new set of classes

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False)

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)
# and a logistic layer -- let's say we have 200 classes
predictions = Dense(200, activation='softmax')(x)

# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)
```



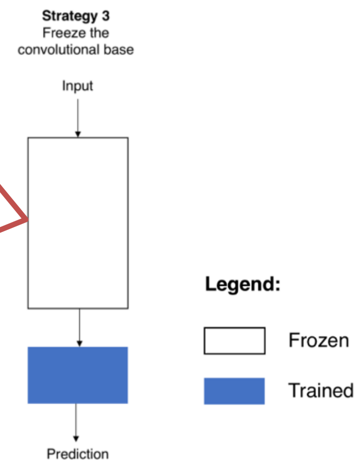
Technical example of transfer learning - https://keras.io/api/applications/

Fine-tune InceptionV3 on a new set of classes

```
# first: train only the top layers (which were randomly initialized)
# i.e. freeze all convolutional InceptionV3 layers
for layer in base_model.layers:
    layer.trainable = False

# compile the model (should be done *after* setting layers to non-trainable)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')

# train the model on the new data for a few epochs
model.fit(...)
```



Technical example of transfer learning - https://keras.io/api/applications/

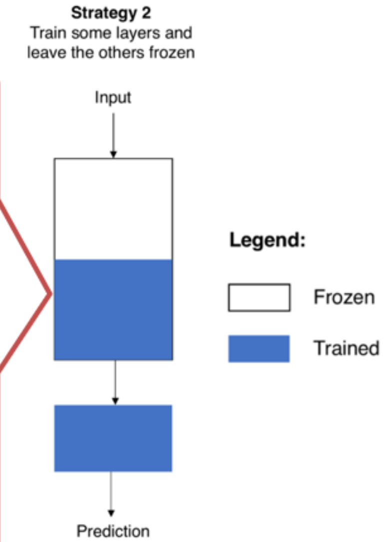
Fine-tune InceptionV3 on a new set of classes

```
# and train the remaining top layers.

# let's visualize layer names and layer indices to see how many layers
# we should freeze:
for i, layer in enumerate(base_model.layers):
   print(i, layer.name)

# we chose to train the top 2 inception blocks, i.e. we will freeze
# the first 249 layers and unfreeze the rest:
for layer in model.layers[:249]:
   layer.trainable = False
for layer in model.layers[249:]:
   layer.trainable = True
model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy')

# we train our model again (this time fine-tuning the top 2 inception blocks
# alongside the top Dense layers
model.fit(...)
```



Technical example of transfer learning - https://keras.io/api/applications/

Extract features with VGG16

```
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)
model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None, None, 3)]   0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, None, None, 64)    1792
                         ⋮
_________________________________________________________________
block5_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, None, None, 512)   0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

Technical example of transfer learning - https://keras.io/api/applications/

Extract features from an arbitrary intermediate layer with VGG19

```python
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg19 import preprocess_input
from tensorflow.keras.models import Model
import numpy as np

base_model = VGG19(weights='imagenet')
model = Model(inputs=base_model.input, outputs=base_model.get_layer('block4_pool').output)

img_path = 'elephant.jpg'
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)

block4_pool_features = model.predict(x)
```

Lab session Keras – TensorFlow core

- Using VGG19 apply two strategies of transfer learning for training CIFAR10 and CIFAR100
  - For each run on datasets show training/validation accuracy curves
  - Confirm which strategy improved your accuracy from previous session
- For technical support follow
  - https://keras.io/api/applications/