



INSTITUTO
METRÓPOLE
DIGITAL



Transfer Learning

01

Fundamentals

02

Feature Extractors

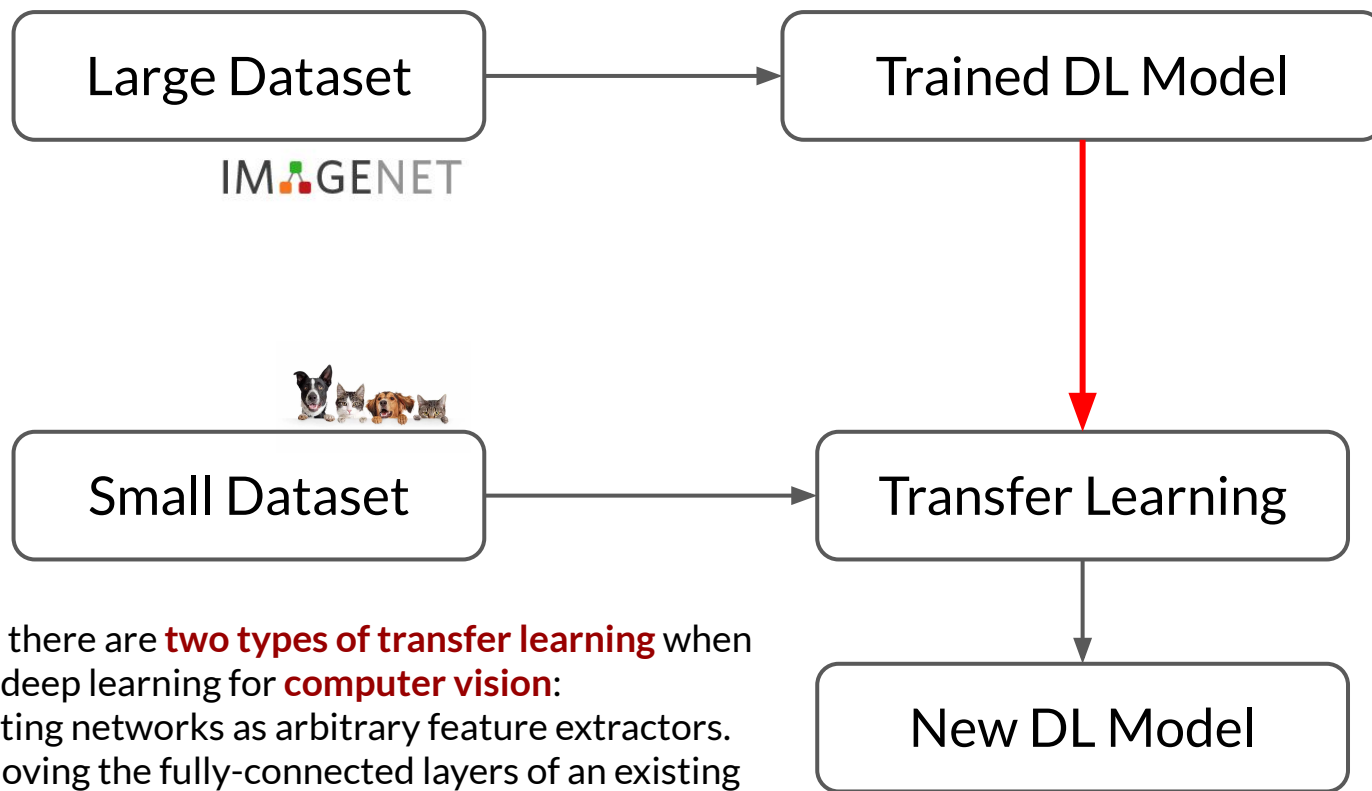
Transfer Learning

03

Fine-Tuning

04

Challenges &
Case Studies

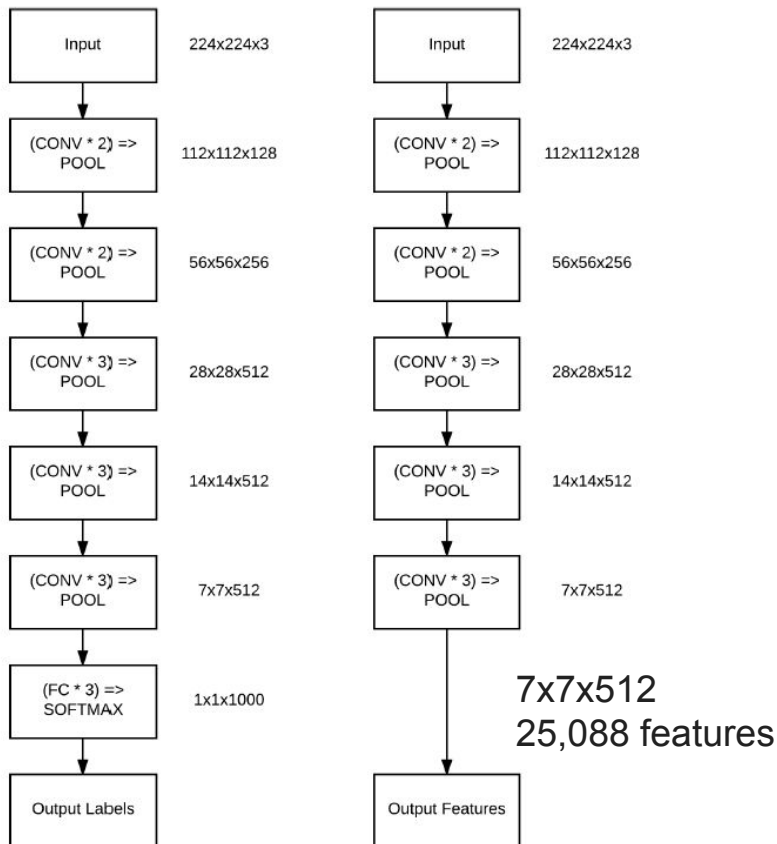


In general, there are **two types of transfer learning** when applied to deep learning for **computer vision**:

1. Treating networks as arbitrary feature extractors.
2. Removing the fully-connected layers of an existing network, placing new FC layer set on top of the CNN, and fine-tuning these weights (and optionally previous layers) to recognize object classes.

Transfer Learning: Extracting features with a pre-trained CNN

VGG 16



```
# import the necessary packages
from tensorflow.keras.applications import VGG16

model = VGG16(weights="imagenet", include_top=False)
features = model.predict(batchImages, batch_size=bs)
```

Shallow ML Classifier
(Logistic Regression, RF, Xgboost, etc)



Image	Features 25088 columns	Class
#01		Cat
#02		Cat
#N		Dog





caltech101_features.hdf5

label_names

```
0: faces
1: leopards
...
100: yin_yang
```

labels

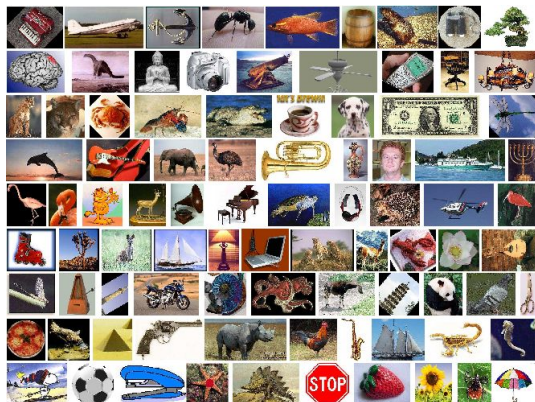
```
0: 75
1: 13
...
8676: 3
```

features

```
0: 0.91, 0.88, 0.96, ..., 0.12
1: 0.68, 0.54, 0.43, ..., 0.83
...
8676: 0.98, 0.76, 0.33, ..., 0.59
```

Caltech 101

8677 instances



Animals: Cat, Dog & Panda

3000 instances



Flowers 17

1360 instances



```

# INPUTS
# path to input dataset
dataset = "animals"

# path to output HDF5 file
output = "animals/hdf5/features.hdf5"

# size of feature extraction buffer
buffer_size = 1000

# store the batch size in a convenience variable
bs = 32

# feature extraction
feature_extraction(dataset,output,buffer_size,bs)

# train and evaluate
train_and_evaluate(output)

```

```

Database keys ['features', 'label_names', 'labels']
[INFO] tuning hyperparameters...
[INFO] best hyperparameters: {'C': 0.1}
[INFO] evaluating...

```

	precision	recall	f1-score	support
cats	0.97	1.00	0.98	264
dogs	0.99	0.96	0.98	250
panda	1.00	1.00	1.00	236
accuracy			0.99	750
macro avg	0.99	0.99	0.99	750
weighted avg	0.99	0.99	0.99	750

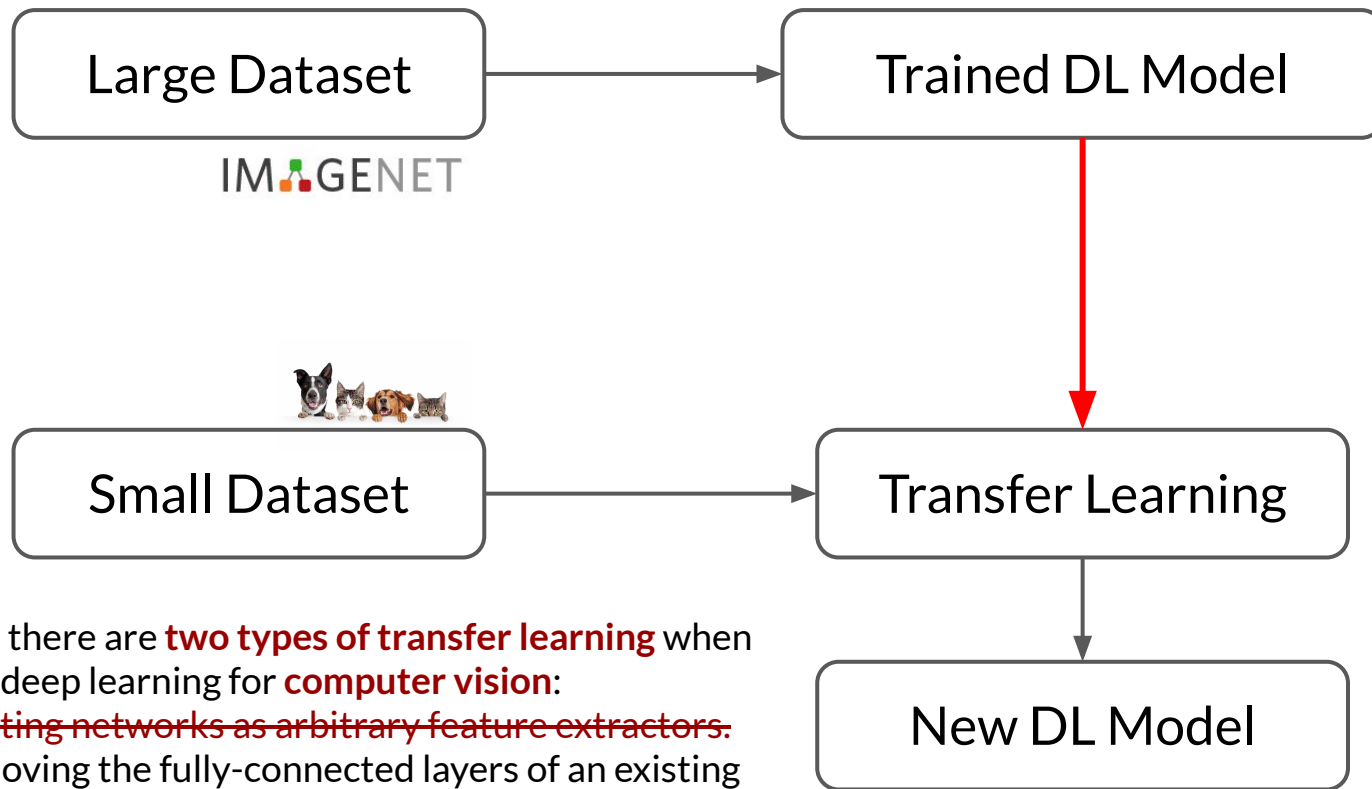


	precision	recall	f1-score	support
Faces	0.98	0.99	0.99	119
Faces_easy	0.99	0.99	0.99	109
Leopards	0.98	1.00	0.99	55
Motorbikes	1.00	1.00	1.00	195
accordion	1.00	1.00	1.00	12
airplanes	1.00	1.00	1.00	214
•	•	•	•	
•	•	•	•	
•	•	•	•	
watch	1.00	0.98	0.99	63
water_lilly	1.00	0.33	0.50	12
wheelchair	1.00	1.00	1.00	14
wild_cat	0.90	0.90	0.90	10
windsor_chair	1.00	1.00	1.00	15
wrench	1.00	0.89	0.94	9
yin_yang	0.88	0.88	0.88	16
accuracy			0.95	2170
macro avg	0.94	0.93	0.93	2170
weighted avg	0.96	0.95	0.95	2170

Caltech 101

	precision	recall	f1-score	support
bluebell	0.95	1.00	0.97	19
buttercup	1.00	0.87	0.93	23
coltsfoot	1.00	0.96	0.98	23
cowslip	0.67	0.80	0.73	25
crocus	1.00	0.90	0.95	20
daffodil	0.78	0.95	0.86	19
daisy	0.94	1.00	0.97	15
dandelion	1.00	0.89	0.94	19
fritillary	0.91	1.00	0.95	20
iris	1.00	0.86	0.92	21
lilyvalley	0.86	0.95	0.90	20
pansy	0.88	1.00	0.93	14
snowdrop	0.90	0.95	0.93	20
sunflower	1.00	1.00	1.00	18
tigerlily	1.00	0.94	0.97	18
tulip	0.89	0.65	0.76	26
windflower	0.95	1.00	0.98	20
accuracy			0.91	340
macro avg	0.93	0.92	0.92	340
weighted avg	0.92	0.91	0.91	340

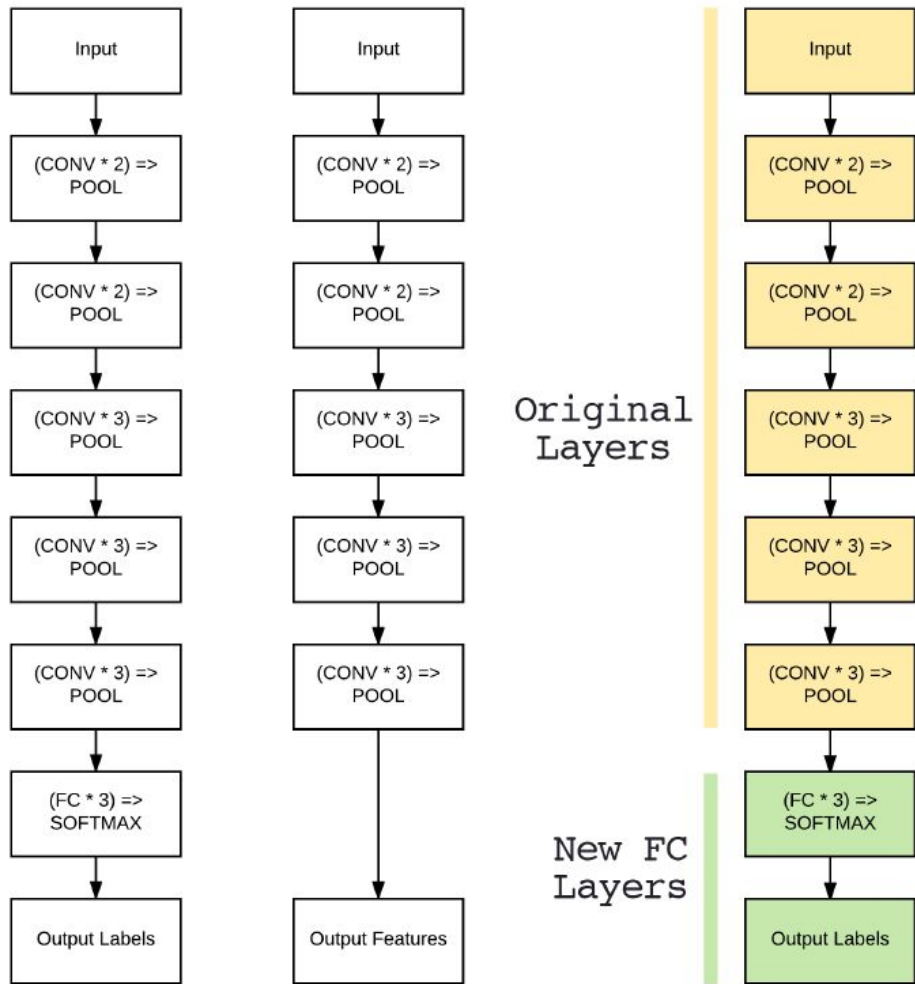
Flowers 17



In general, there are **two types of transfer learning** when applied to deep learning for **computer vision**:

- ~~1. Treating networks as arbitrary feature extractors.~~
2. Removing the fully-connected layers of an existing network, placing new FC layer set on top of the CNN, and **fine-tuning** these weights (and optionally previous layers) to recognize object classes.

Fine-Tuning



```
# a fully connect network
class FCHeadNet:
    @staticmethod
    def build(baseModel, classes, D):
        # initialize the head model that will be placed on top of
        # the base, then add a FC layer
        headModel = baseModel.output
        headModel = Flatten(name="flatten")(headModel)
        headModel = Dense(D, activation="relu")(headModel)
        headModel = Dropout(0.5)(headModel)

        # add a softmax layer
        headModel = Dense(classes, activation="softmax")(headModel)

        # return the model
        return headModel
```

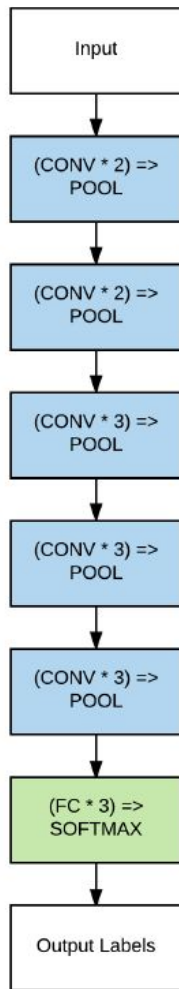
$(FC * 3) \Rightarrow$
SOFTMAX

Old FC
Layers

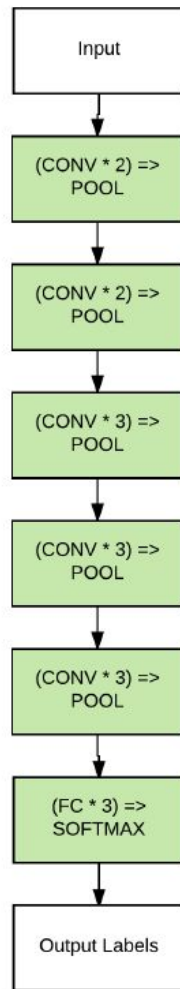


Freeze Early
Layers in
Network

Only Train
FC Layers



Unfreeze Early
Layers & Train
All



RMSprop is frequently used in situations where we need to quickly obtain reasonable performance (first stage - left image).

SGD using a very small learning rate (second stage - right image)

Tip!

```
# loop over all layers in the base
# model and freeze them so they
# will *not* be updated during the
# training process
for layer in baseModel.layers:
    layer.trainable = False
```



Let's do a fine tuning using VGG 16 over Flowers 17

Flowers 17
1360 instances



Previous result using feature extraction

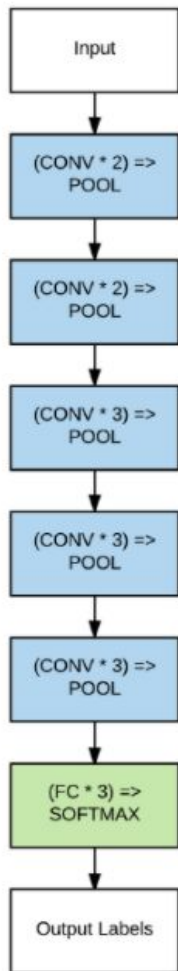
	precision	recall	f1-score	support
bluebell	0.95	1.00	0.97	19
buttercup	1.00	0.87	0.93	23
coltsfoot	1.00	0.96	0.98	23
cowslip	0.67	0.80	0.73	25
crocus	1.00	0.90	0.95	20
daffodil	0.78	0.95	0.86	19
daisy	0.94	1.00	0.97	15
dandelion	1.00	0.89	0.94	19
fritillary	0.91	1.00	0.95	20
iris	1.00	0.86	0.92	21
lilyvalley	0.86	0.95	0.90	20
pansy	0.88	1.00	0.93	14
snowdrop	0.90	0.95	0.93	20
sunflower	1.00	1.00	1.00	18
tigerlily	1.00	0.94	0.97	18
tulip	0.89	0.65	0.76	26
windflower	0.95	1.00	0.98	20
accuracy			0.91	340
macro avg	0.93	0.92	0.92	340
weighted avg	0.92	0.91	0.91	340

Stage #01

Epochs: 25, RMSProp (0.001), FC (256)

Freeze Early
Layers in
Network

Only Train
FC Layers



[INFO] evaluating after initialization...

	precision	recall	f1-score	support
bluebell	0.94	0.79	0.86	19
buttercup	0.90	0.95	0.92	19
coltsfoot	0.79	0.94	0.86	16
cowslip	0.77	0.85	0.81	20
crocus	0.73	0.89	0.80	18
daffodil	0.78	0.78	0.78	23
daisy	1.00	0.95	0.97	20
dandelion	0.94	0.80	0.86	20
fritillary	1.00	0.86	0.92	21
iris	1.00	1.00	1.00	16
lilyvalley	0.95	0.95	0.95	22
pansy	1.00	0.91	0.95	23
snowdrop	0.95	0.83	0.88	23
sunflower	0.95	0.95	0.95	20
tigerlily	0.82	0.93	0.87	15
tulip	0.69	0.74	0.71	27
windflower	0.95	1.00	0.97	18
accuracy			0.88	340
macro avg	0.89	0.89	0.89	340
weighted avg	0.89	0.88	0.88	340

Stage #02

```
# import the necessary packages
from tensorflow.keras.applications import VGG16

# whether or not to include top of CNN
include_top = 0

# load the VGG16 network
print("[INFO] loading network...")
model = VGG16(weights="imagenet", include_top= include_top > 0)
print("[INFO] showing layers...")

# loop over the layers in the network and display them to the
# console
for (i, layer) in enumerate(model.layers):
    print("[INFO] {} \t {}".format(i, layer.__class__.__name__))
```

VGG-16 without FC

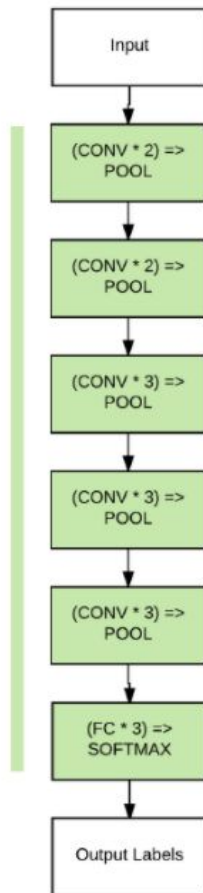
```
[INFO] loading network...
[INFO] showing layers...
[INFO] 0      InputLayer
[INFO] 1      Conv2D
[INFO] 2      Conv2D
[INFO] 3      MaxPooling2D
[INFO] 4      Conv2D
[INFO] 5      Conv2D
[INFO] 6      MaxPooling2D
[INFO] 7      Conv2D
[INFO] 8      Conv2D
[INFO] 9      Conv2D
[INFO] 10     MaxPooling2D
[INFO] 11     Conv2D
[INFO] 12     Conv2D
[INFO] 13     Conv2D
[INFO] 14     MaxPooling2D
[INFO] 15     Conv2D
[INFO] 16     Conv2D
[INFO] 17     Conv2D
[INFO] 18     MaxPooling2D
```



Stage #02

Unfreeze Early
Layers & Train
All

```
# now that the head FC layers have been
# trained/initialized, lets
# unfreeze the final set of CONV layers and
# make them trainable
for layer in baseModel.layers[15:]:
    layer.trainable = True
```



VGG-16 without FC

```
[INFO] loading network...
[INFO] showing layers...
[INFO] 0      InputLayer
[INFO] 1      Conv2D
[INFO] 2      Conv2D
[INFO] 3      MaxPooling2D
[INFO] 4      Conv2D
[INFO] 5      Conv2D
[INFO] 6      MaxPooling2D
[INFO] 7      Conv2D
[INFO] 8      Conv2D
[INFO] 9      Conv2D
[INFO] 10     MaxPooling2D
[INFO] 11     Conv2D
[INFO] 12     Conv2D
[INFO] 13     Conv2D
[INFO] 14     MaxPooling2D
[INFO] 15     Conv2D
[INFO] 16     Conv2D
[INFO] 17     Conv2D
[INFO] 18     MaxPooling2D
```



Feature extraction

Stage #02 fine tuning
SGD (0.001), epochs = 100

	precision	recall	f1-score	support
bluebell	0.95	1.00	0.97	19
buttercup	1.00	0.87	0.93	23
coltsfoot	1.00	0.96	0.98	23
cowslip	0.67	0.80	0.73	25
crocus	1.00	0.90	0.95	20
daffodil	0.78	0.95	0.86	19
daisy	0.94	1.00	0.97	15
dandelion	1.00	0.89	0.94	19
fritillary	0.91	1.00	0.95	20
iris	1.00	0.86	0.92	21
lilyvalley	0.86	0.95	0.90	20
pansy	0.88	1.00	0.93	14
snowdrop	0.90	0.95	0.93	20
sunflower	1.00	1.00	1.00	18
tigerlily	1.00	0.94	0.97	18
tulip	0.89	0.65	0.76	26
windflower	0.95	1.00	0.98	20
accuracy			0.91	340
macro avg	0.93	0.92	0.92	340
weighted avg	0.92	0.91	0.91	340

	precision	recall	f1-score	support
bluebell	0.90	0.95	0.92	19
buttercup	1.00	0.95	0.97	19
coltsfoot	0.83	0.94	0.88	16
cowslip	0.90	0.90	0.90	20
crocus	0.85	0.94	0.89	18
daffodil	0.91	0.87	0.89	23
daisy	1.00	0.95	0.97	20
dandelion	0.89	0.85	0.87	20
fritillary	1.00	0.90	0.95	21
iris	1.00	1.00	1.00	16
lilyvalley	1.00	0.95	0.98	22
pansy	1.00	0.91	0.95	23
snowdrop	0.92	1.00	0.96	23
sunflower	1.00	0.90	0.95	20
tigerlily	1.00	1.00	1.00	15
tulip	0.81	0.93	0.86	27
windflower	1.00	1.00	1.00	18
accuracy			0.94	340
macro avg	0.94	0.94	0.94	340
weighted avg	0.94	0.94	0.94	340



Dogs vs. Cats

Create an algorithm to distinguish dogs from cats



Kaggle · 213 teams · 7 years ago

Overview

Data

Notebooks

Discussion

Leaderboard

Rules

Team

Overview

Description

Prizes

Evaluation

Winners

In this competition, you'll write an algorithm to classify whether images contain either a dog or a cat. This is easy for humans, dogs, and cats. Your computer will find it a bit more difficult.



HDF5
AlexNet
ResNet50
Transfer Learning

http://bit.do/alexnet_epoch45

