

Chapter 14

Transfer Learning

오 세 종

Contents



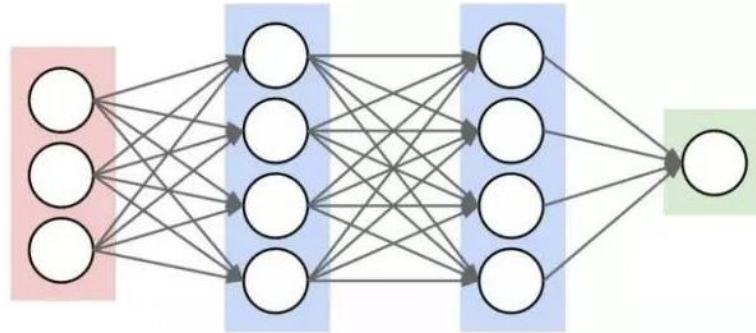
1. Summary
2. Use pre tuned model (Vgg16)
3. Transfer learning using MobileNetV3
3. EfficientNet
4. Keras Regression
5. Deep learning application for computer vision

1. Summary

- Transfer learning (전이학습)
 - 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.
 - For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks.
 - imageNet 문제를 해결하는데 사용한 DNN 모델을 다른 image 분류 문제에 활용
 - Neural network architecture
 - weights

1. Summary

- 다른 모델의 활용 방법
 - Architecture만 활용
 - Weight 값 학습은 초기화 상태에서 새로 실시. 많은 시간 필요
 - Weight 값 까지 활용
 - 모델에 포함된 weight 값을 기초로 새로운 문제에 맞게 재학습



잘 구축된 모델 = Good Architecture +
Well tuned weights

1. Summary

- Keras Applications are deep learning models that are made available alongside pre-trained weights.
- These models can be used for prediction, feature extraction, and fine-tuning

Keras Applications

- Xception
- EfficientNet B0 to B7
- VGG16 and VGG19
- ResNet and ResNetV2
- MobileNet and MobileNetV2
- DenseNet
- NasNetLarge and NasNetMobile
- InceptionV3
- InceptionResNetV2

Ref: <https://keras.io/api/applications/>

1. Summary

Available models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201

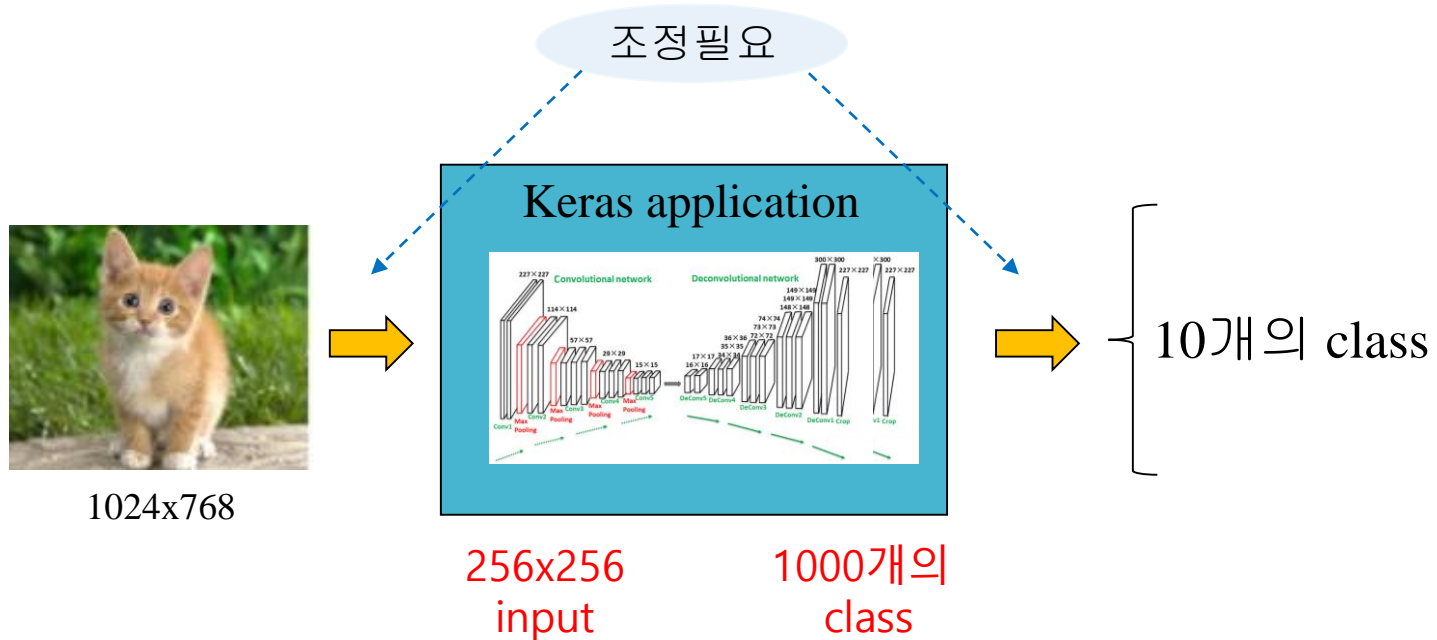
↓
지지율이 높은 상위 5개 클래스 가운데 정답이 포함될 확률

1. Summary

NASNetMobile	23 MB	0.744	0.919	5,326,716	-
<u>NASNetLarge</u>	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	-	-	5,330,571	-
EfficientNetB1	31 MB	-	-	7,856,239	-
EfficientNetB2	36 MB	-	-	9,177,569	-
EfficientNetB3	48 MB	-	-	12,320,535	-
EfficientNetB4	75 MB	-	-	19,466,823	-
EfficientNetB5	118 MB	-	-	30,562,527	-
EfficientNetB6	166 MB	-	-	43,265,143	-
EfficientNetB7	256 MB	-	-	66,658,687	-

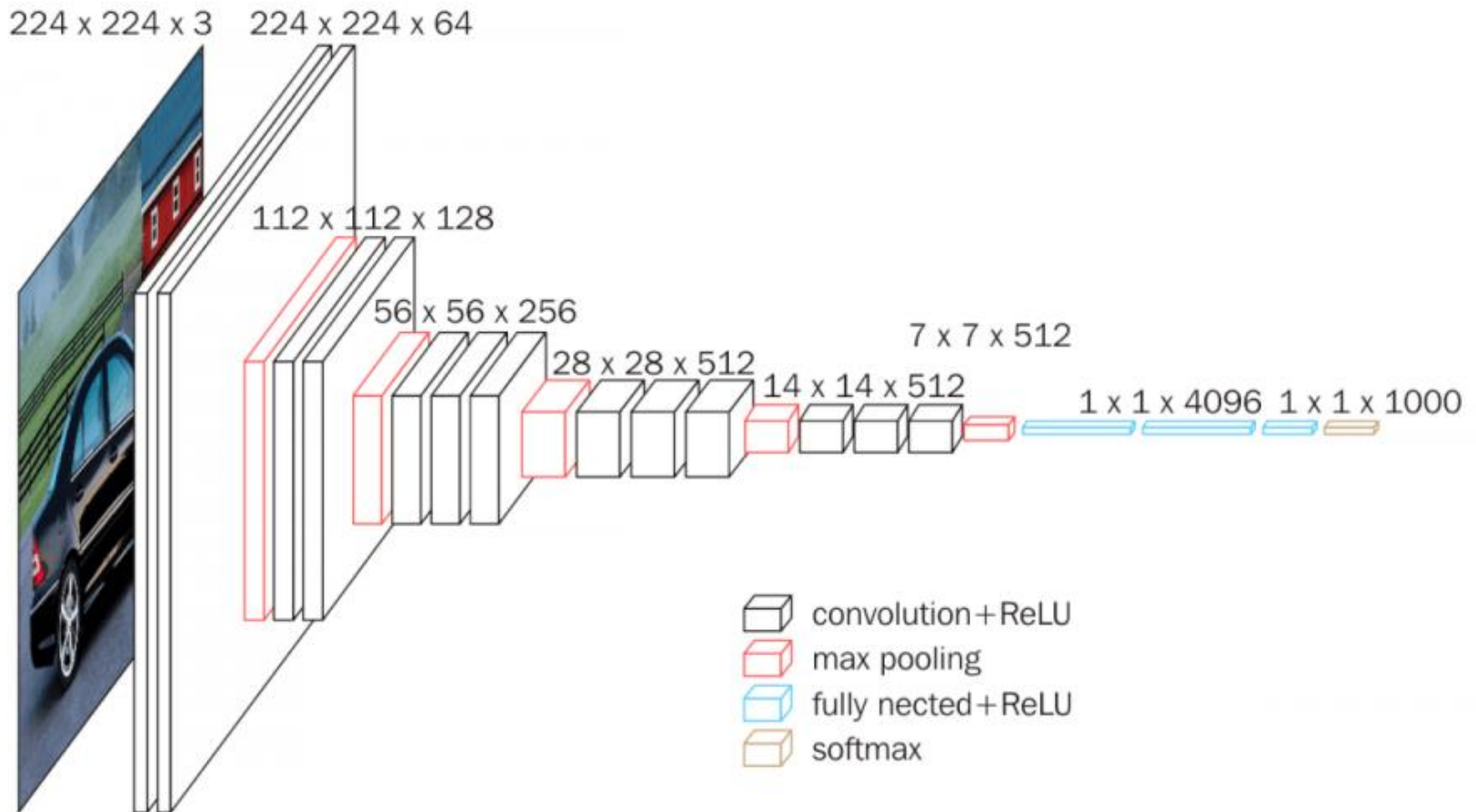
1. Summary

- 이 모델들은 imagenet 의 데이터에 맞추어 구조가 만들어짐
- 다른 작업에 사용하려면 입력과 출력의 dimension 이 다를 수 있다.
- Keras 에서는 이런 부분을 조정할 수 있도록 지원한다



2. Use pre tuned model (Vgg16)

- VGG16 architecture



2. Use pre tuned model (Vgg16)

- Predict a random image using VGG16
 - Load VGG16 model (pre tuned)
 - Prepare a image
 - Predict the image
 - (no modification of VGG16)



2. Use pre tuned model (Vgg16)

14.VGG_case1.py

```
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
from keras.applications.vgg16 import VGG16

# load the model
model = VGG16() # take a long time

# load an image from file
image = load_img('D:/data/sample_img_1.jpg', target_size=(224, 224))

# convert the image pixels to a numpy array
image = img_to_array(image)

# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
```

2. Use pre tuned model (Vgg16)

```
# prepare the image for the VGG model
image = preprocess_input(image)
```

```
In [35]: image.shape
Out[35]: (1, 224, 224, 3)
```

```
# predict the probability across all output classes
pred = model.predict(image)
```

```
In [37]: pred
```

```
Out[37]:
```

```
array([[2.02823500e-08, 2.36752442e-07, 4.87318896e-09, 1.45811576e-08,
        2.95860527e-08, 6.06516650e-08, 8.11169087e-09, 1.99178729e-07,
        1.34378226e-07, 2.45495016e-07, 1.73777636e-07, 4.73473506e-07,
        4.38422518e-07, 9.14644076e-08, 3.74206678e-07, 1.15517921e-07,
        2.07582545e-07, 1.64247808e-07, 1.92927331e-07, 3.33448384e-07,
        8.47046966e-09, 4.27838813e-08, 4.27860840e-08, 1.17808099e-07,
        1.19830617e-07, 2.76811019e-08, 4.42640697e-08, 2.22004218e-07,
        6.31745465e-08, 3.23924428e-06, 4.34751257e-08, 2.62859260e-07,
        1.53438549e-07, 5.41724745e-08, 2.16927365e-08, 1.60927467e-08,
        1.78756821e-07, 3.80799996e-08, 1.06899286e-07, 1.05639970e-07,
        7.63831919e-08, 6.99591993e-08, 5.16803986e-08, 6.35227693e-08])
```

1000 개의 class에 대한
확률값

```
In [38]: pred.shape
```

```
Out[38]: (1, 1000)
```

2. Use pre tuned model (Vgg16)

```
# convert the probabilities to class labels
label = decode_predictions(pred)
```

```
In [40]: label
Out[40]:
[[('n03063599', 'coffee_mug', 0.7272259),
 ('n03063689', 'coffeepot', 0.10312535),
 ('n07930864', 'cup', 0.06428892),
 ('n04398044', 'teapot', 0.032623097),
 ('n03950228', 'pitcher', 0.025435064)]]
```



```
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
```

```
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
In [42]: print('%s (%.2f%%)' % (label[1], label[2]*100))
coffee_mug (72.72%)
```

```
>>> model.summary()
Model: "vgg16"
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	1,792
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	4,480
block2_conv2 (Conv2D)	(None, 112, 112, 128)	4,480
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	16,640
block3_conv2 (Conv2D)	(None, 56, 56, 256)	16,640
block3_conv3 (Conv2D)	(None, 56, 56, 256)	16,640
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	43,008
block4_conv2 (Conv2D)	(None, 28, 28, 512)	43,008
block4_conv3 (Conv2D)	(None, 28, 28, 512)	43,008
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	43,008
block5_conv2 (Conv2D)	(None, 14, 14, 512)	43,008
block5_conv3 (Conv2D)	(None, 14, 14, 512)	43,008
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102,764,544
fc2 (Dense)	(None, 4096)	16,781,312
predictions (Dense)	(None, 1000)	4,097,000

Total params: 138,357,544 (527.79 MB)




[실습1]

- 다음과 같이 이미지 파일을 업로드 하면 분류 결과를 알려주는 Python GUI 프로그램을 작성하시오

AI Image Classifier

Load image



Classification result

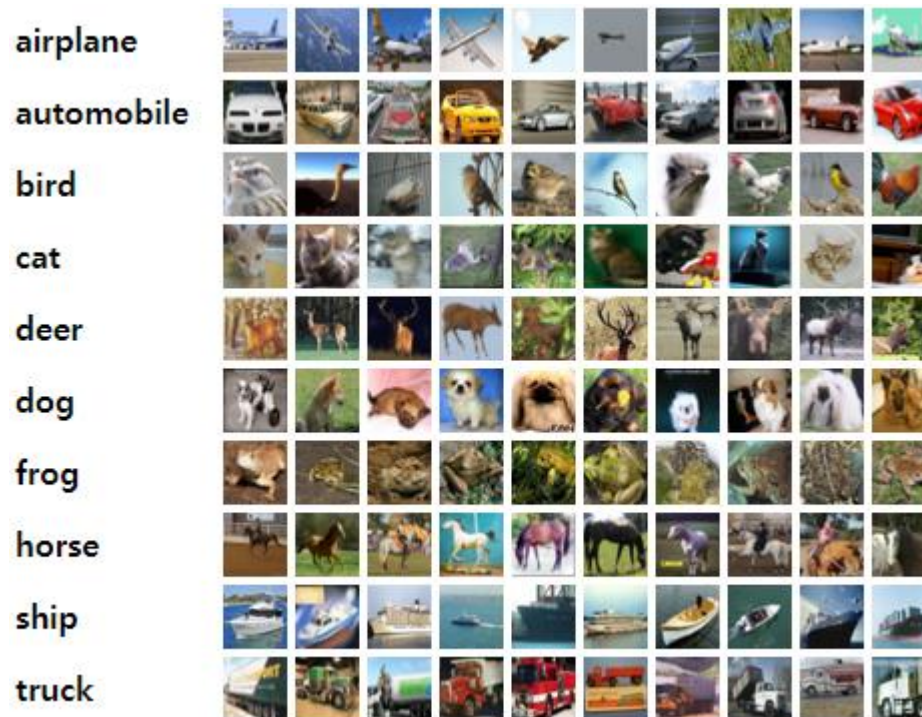
1.	coffee_mug	(0.73)
2.	coffeepot	(0.10)
3.	cup	(0.64)

[실습1]

- GUI 는 어떤 패키지를 이용해도 상관 없음 (PyQT, tkinter)
 - 윈도우에 표시되는 이미지는 원본의 크기와 상관 없이 높이를 일정하게 한다 (가로,세로 비율 유지)
 - 사용자가 [Load Image] 버튼을 클릭하면 이미지 파일을 선택할 수 있는 팝업창(탐색기) 이 표시되도록 한다.
 - 화면 디자인(배치)은 약간 달라도 상관 없음
-
- 참고 코드 : <https://www.codespeedy.com/displaying-an-image-using-pyqt5-in-python/>

3. Transfer learning using MobileNetV3

- CIFAR-10 classification
 - The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class.
 - There are 50000 training images and 10000 test images.



3. Transfer learning using MobileNetV3

14.VMobileNetV3_cifar.py

```
from keras import optimizers
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dropout, Flatten, Dense
from tensorflow.python.keras.utils import np_utils
from keras.applications import MobileNetV3Small
import keras.backend as K

K.clear_session()

# set up base model
img_width, img_height = 32, 32
base_model = MobileNetV3Small(weights='imagenet',
                                include_top=False,          # output 부분 사용x
                                input_shape= (img_width, img_height, 3))

base_model.summary()

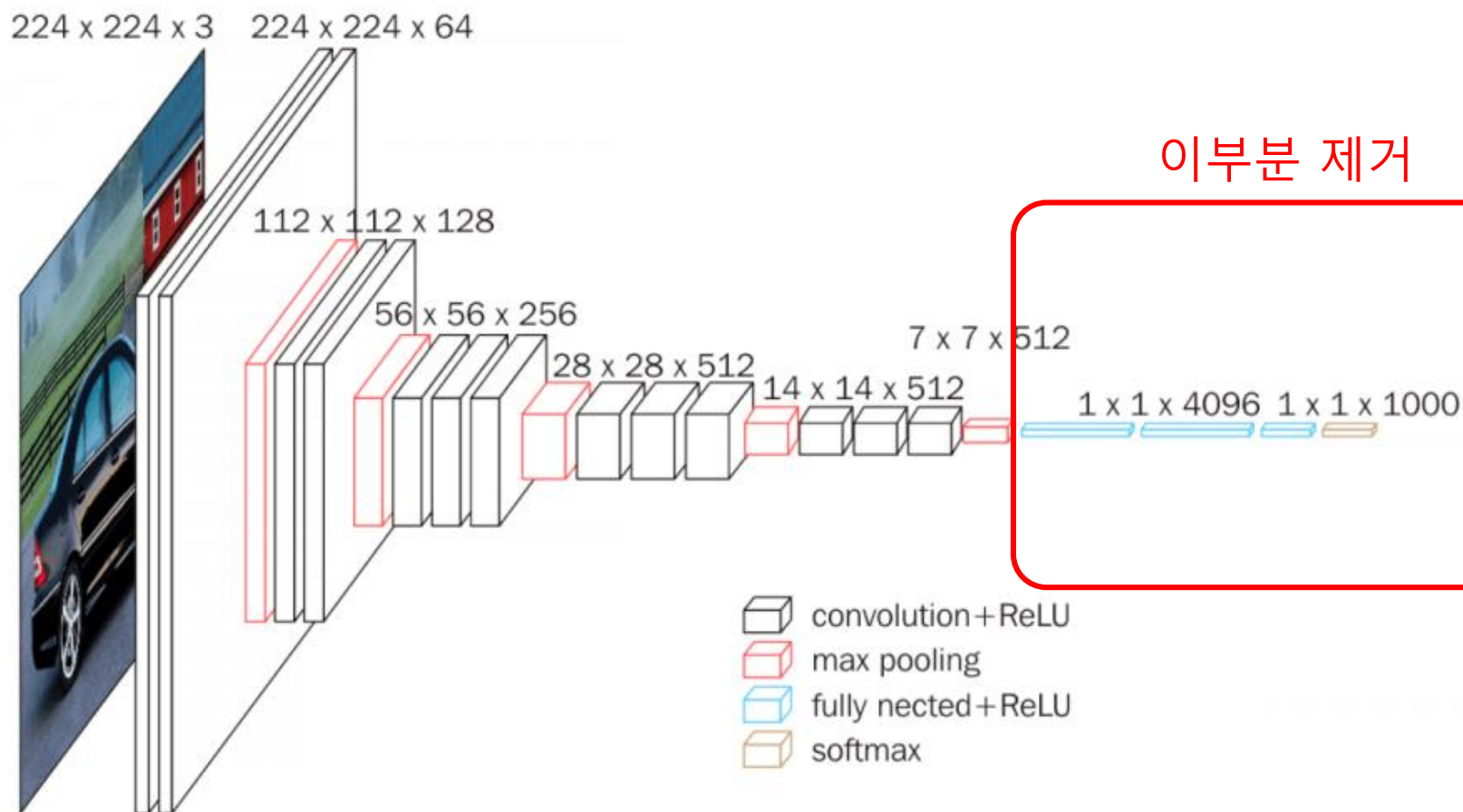
nb_epoch = 50      # 시간이 오래 걸리니 10 정도로 해서 테스트한다
nb_classes = 10
```

```
>>> base_model.summary()
Model: "MobileNetV3Small"
```

expanded_conv_10_project_bn (BatchNormalization)	(None, 1, 1, 96)	384	expanded_conv_10_project[...
expanded_conv_10_add (Add)	(None, 1, 1, 96)	0	expanded_conv_9_add[0][0], expanded_conv_10_project_...
conv_1 (Conv2D)	(None, 1, 1, 576)	55,296	expanded_conv_10_add[0][0]
(BatchNormalization)			
activation_17 (Activation)	(None, 1, 1, 576)	0	conv_1_bn[0][0]

Total params: 939,120 (3.58 MB)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 939,120 (3.58 MB)

Fully connect layers 가 제거됨



3. Transfer learning using MobileNetV3

```
# load dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
y_train = np_utils.to_categorical(y_train, nb_classes)
y_test = np_utils.to_categorical(y_test, nb_classes)

# define model
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(1024,activation=('relu')))
model.add(Dense(512,activation=('relu')))
model.add(Dense(256,activation=('relu')))
model.add(Dense(128,activation=('relu')))
model.add(Dense(10,activation=('softmax')))

model.summary()
```

3. Transfer learning using MobileNetV3

```
>>> model.summary()  
Model: "sequential"
```

Layer (type)	Output Shape	Param #
MobileNetV3Small (Functional)	(None, 1, 1, 576)	939,120
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 1024)	590,848
dense_1 (Dense)	(None, 512)	524,800
dense_2 (Dense)	(None, 256)	131,328
dense_3 (Dense)	(None, 128)	32,896
dense_4 (Dense)	(None, 10)	1,290

Total params: 2,220,282 (8.47 MB)

추가된 부분

3. Transfer learning using MobileNetV3

```
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.SGD(learning_rate=1e-2),
              metrics=['accuracy'])

model.fit(X_train, y_train,
        validation_data=(X_test, y_test),
        epochs=nb_epoch,
        batch_size=200,
        verbose=1)

# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("loss: %.2f" % scores[0])
print("acc: %.2f" % scores[1])
```


3. Transfer learning using MobileNetV3

```
Epoch 47/50
250/250 [=====] - 41s 166ms/step - loss: 0.1880 - accuracy: 0.5830
- val_loss: 0.2077 - val_accuracy: 0.5320
Epoch 48/50
250/250 [=====] - 41s 163ms/step - loss: 0.1868 - accuracy: 0.5869
- val_loss: 0.2066 - val_accuracy: 0.5354
Epoch 49/50
250/250 [=====] - 41s 163ms/step - loss: 0.1861 - accuracy: 0.5891
- val_loss: 0.2061 - val_accuracy: 0.5374
Epoch 50/50
250/250 [=====] - 41s 165ms/step - loss: 0.1850 - accuracy: 0.5899
- val_loss: 0.2056 - val_accuracy: 0.5398
<keras.src.callbacks.History object at 0x00002952D90AA10>
...

>>> print("loss: %.2f" % scores[0])
loss: 0.21
>>> print("acc: %.2f" % scores[1])
acc: 0.54
```

3. Transfer learning using MobileNetV3

- Note

- 전이학습에서 base model 의 parameter 값들을 갱신할지 말지를 정할 수 있음
- `base_model.trainable = False`
 - base model 의 parameter 값들은 고정하고, 사용자가 추가한 layer 들의 parameter에 대해서만 갱신이 일어나도록 함
 - 학습시 갱신할 parameter의 수가 줄어들어 학습이 빠르게 진행될 수 있으나, base model이 고정됨으로 해서 전이학습 모델의 성능이 저하될 가능성도 있음

3. Transfer learning using MobileNetV3

- Note

- base_model.trainable = True (기본 값이라 실행할 필요 없음)

```
=====
Total params: 2220282 (8.47 MB)
Trainable params: 2208170 (8.42 MB)
Non-trainable params: 12112 (47.31 KB) → 오류 역전파시 갱신되지 않는 파라미터
```

- base_model.trainable = False

```
# set up base model
img_width, img_height = 32, 32
base_model = MobileNetV3Small(weights='imagenet',
                                include_top=False, # output 부분 사용x
                                input_shape= (img_width, img_height, 3))

base_model.trainable = False
```

```
=====
Total params: 2220282 (8.47 MB)
Trainable params: 1281162 (4.89 MB)
Non-trainable params: 939120 (3.58 MB)
```



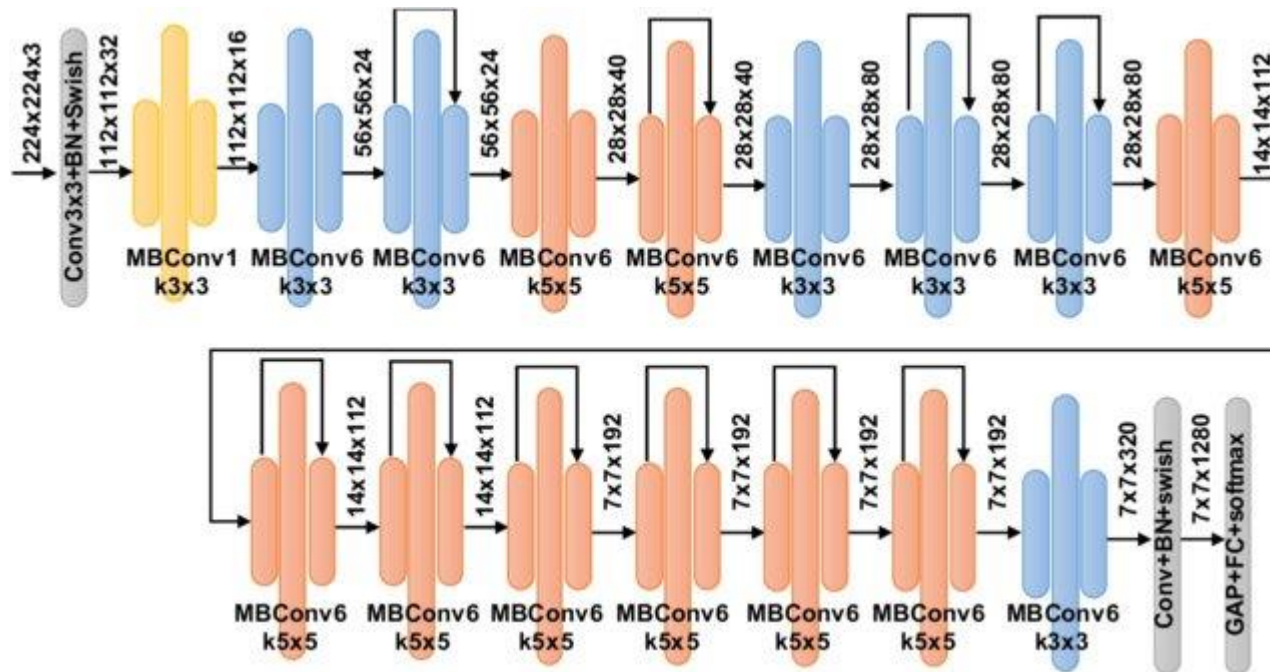
4. EfficientNet

- EfficientNet, first introduced in [Tan and Le, 2019](#) is among the most efficient models (i.e. requiring least FLOPS for inference) that reaches State-of-the-Art accuracy on both imagenet and common image classification transfer learning tasks.
- EfficientNet provides a family of models (B0 to B7) that represents a good combination of efficiency and accuracy on a variety of scales.
- efficiency-oriented base model (B0) to surpass models at every scale

https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/

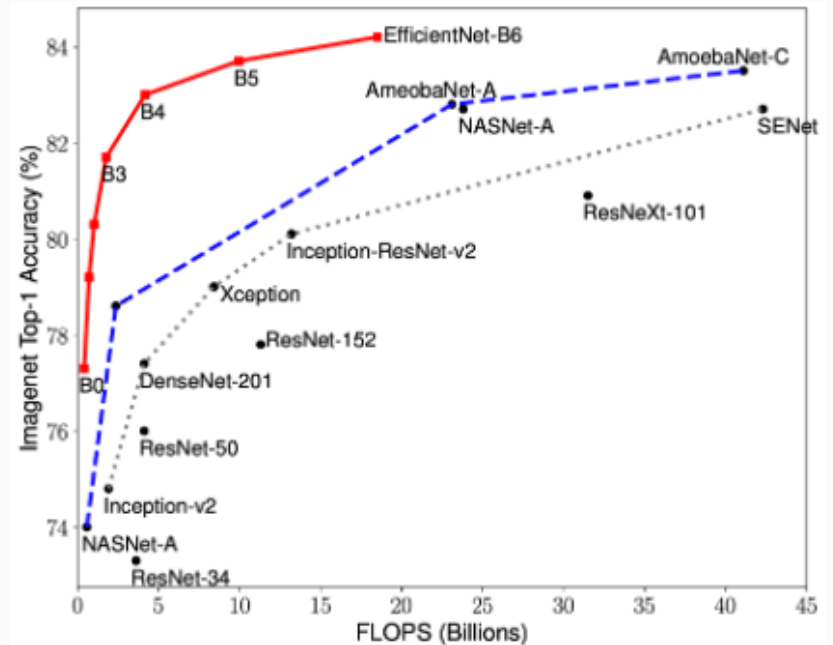
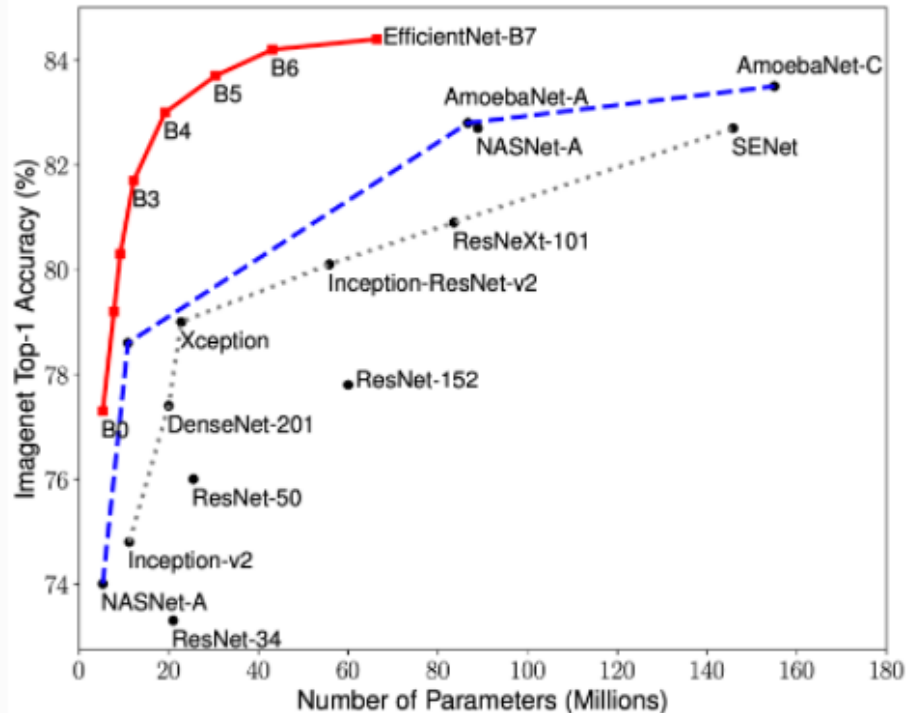
4. EfficientNet

- EfficientNet B0 architecture



https://www.researchgate.net/publication/348470984_Classification_of_Remote_Sensing_Images_Using_EfficientNet-B3_CNN_Model_with_Attention/figures?lo=1

4. EfficientNet



4. EfficientNet

- B0 to B7 variants of EfficientNet
 - Resolution: Resolutions not divisible by 8, 16, etc. cause zero-padding near boundaries of some layers which wastes computational resources.
 - Depth and width: The building blocks of EfficientNet demands channel size to be multiples of 8.

Base model	resolution
EfficientNetB0	224
EfficientNetB1	240
EfficientNetB2	260
EfficientNetB3	300
EfficientNetB4	380
EfficientNetB5	456
EfficientNetB6	528
EfficientNetB7	600

[실습 2]

- efficientnetB0 모델을 이용하여 cifar10 dataset 을 분류하기 위한 전이학습 모델을 구현하고 성능을 평가하시오.
- 3절의 전이학습 코드를 참조
- 모델의 구조는 다음과 같이 만든다.

Layer (type)	Output Shape	Param #
efficientnetb0 (Functional)	(None, 1, 1, 1280)	4049571
flatten (Flatten)	(None, 1280)	0
dense (Dense)	(None, 256)	327936
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 10)	2570

Dropout rate = 0.5

[실습 2]

- Epoch 는 2, batch size 는 200 으로 한다
- 모델 컴파일은 다음과 같이 한다

```
model.compile(loss='categorical_crossentropy',  
              optimizer=optimizers.SGD(lr=1e-3, momentum=0.9),  
              metrics=['accuracy'])
```

- 제출사항
 - 전이학습 모델 architecture (model.summary() 결과)
 - Model fitting 실행 결과

```
Epoch 1/2  
250/250 [=====] - 220s 829ms/step - loss: 0.3620 - accuracy: 0.2016 - val_loss: 0.2700 - val  
Epoch 2/2  
250/250 [=====] - 207s 829ms/step - loss: 0.2632 - accuracy: 0.4080 - val_loss: 0.2136 - val  
<keras.src.callbacks.History object at 0x000002951EB4DA10>
```



5. Keras Regression

- Regression 은 출력 값이 1개, one-hot encoding 불필요
- Boston housing dataset

BostonHousing 데이터 설명

[01] CRIM	자치시(town) 별 1인당 범죄율
[02] ZN	25,000 평방피트를 초과하는 거주지역의 비율
[03] INDUS	비소매상업지역이 점유하고 있는 토지의 비율
[04] CHAS	찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)
[05] NOX	10ppm 당 농축 일산화질소
[06] RM	주택 1가구당 평균 방의 개수
[07] AGE	1940년 이전에 건축된 소유주택의 비율
[08] DIS	5개의 보스턴 직업센터까지의 접근성 지수
[09] RAD	방사형 도로까지의 접근성 지수
[10] TAX	10,000 달러 당 재산세를
[11] PTRATIO	자치시(town)별 학생/교사 비율
[12] B	$1000(Bk-0.63)^2$, 여기서 Bk는 자치시별 흑인의 비율을 말함.
[13] LSTAT	모집단의 하위계층의 비율(%)
[14] MEDV	본인 소유의 주택가격(중앙값) (단위: \$1,000)

http://dator.co.kr/?vid=ctg258&mid=textyle&document_srl=1721307

5. Keras Regression

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from keras.datasets import boston_housing

# load dataset
(X_train, y_train), (X_test, y_test) = boston_housing.load_data()

model = Sequential()
model.add(Dense(16, input_dim=13, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1))                                     # output

model.compile(loss='mean_squared_error', optimizer='adam')

model.fit(X_train, y_train, epochs=200, batch_size=10)
```

```
Epoch 1/200
404/404 [=====] - 1s 2ms/step - loss: 271.7087
Epoch 2/200
404/404 [=====] - 0s 116us/step - loss: 132.0523
Epoch 3/200
404/404 [=====] - 0s 109us/step - loss: 91.2605
Epoch 4/200
404/404 [=====] - 0s 111us/step - loss: 80.1819
Epoch 5/200
404/404 [=====] - 0s 106us/step - loss: 76.2772
Epoch 6/200
404/404 [=====] - 0s 116us/step - loss: 75.0198
Epoch 7/200
404/404 [=====] - 0s 116us/step - loss: 79.4039
Epoch 8/200
404/404 [=====] - 0s 109us/step - loss: 74.7349
Epoch 9/200
404/404 [=====] - 0s 116us/step - loss: 70.2591
Epoch 10/200
Epoch 194/200
404/404 [=====] - 0s 111us/step - loss: 28.7584
Epoch 195/200
404/404 [=====] - 0s 113us/step - loss: 28.9097
Epoch 196/200
404/404 [=====] - 0s 116us/step - loss: 29.4230
Epoch 197/200
404/404 [=====] - 0s 109us/step - loss: 28.9272
Epoch 198/200
404/404 [=====] - 0s 111us/step - loss: 27.5968
Epoch 199/200
404/404 [=====] - 0s 106us/step - loss: 27.4105
Epoch 200/200
404/404 [=====] - 0s 165us/step - loss: 26.2822
Out[56]: <keras.callbacks.callbacks.History at 0x25c289dbd08>
```

5. Keras Regression

```
Y_prediction = model.predict(X_test).flatten()

for i in range(10):
    real_price = y_test[i]
    predicted_price = Y_prediction[i]
    print('Real Price: {:.3f}, Predicted Price: {:.3f}'.format(real_price,
                                                                predicted_price))
```

```
Real Price: 7.200, Predicted Price: 9.471
Real Price: 18.800, Predicted Price: 21.751
Real Price: 19.000, Predicted Price: 23.354
Real Price: 27.000, Predicted Price: 31.055
Real Price: 22.200, Predicted Price: 25.802
Real Price: 24.500, Predicted Price: 21.640
Real Price: 31.200, Predicted Price: 29.827
Real Price: 22.900, Predicted Price: 26.122
Real Price: 20.500, Predicted Price: 19.287
Real Price: 23.200, Predicted Price: 21.618
```



6. Deep learning application for computer vision

- <https://machinelearningmastery.com/applications-of-deep-learning-for-computer-vision/>
- (1) Image Classification



airplane

automobile

bird

cat

deer

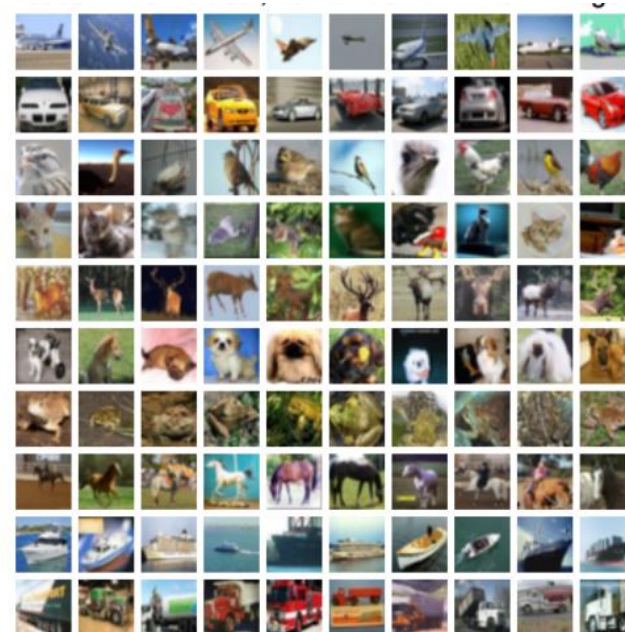
dog

frog

horse

ship

truck

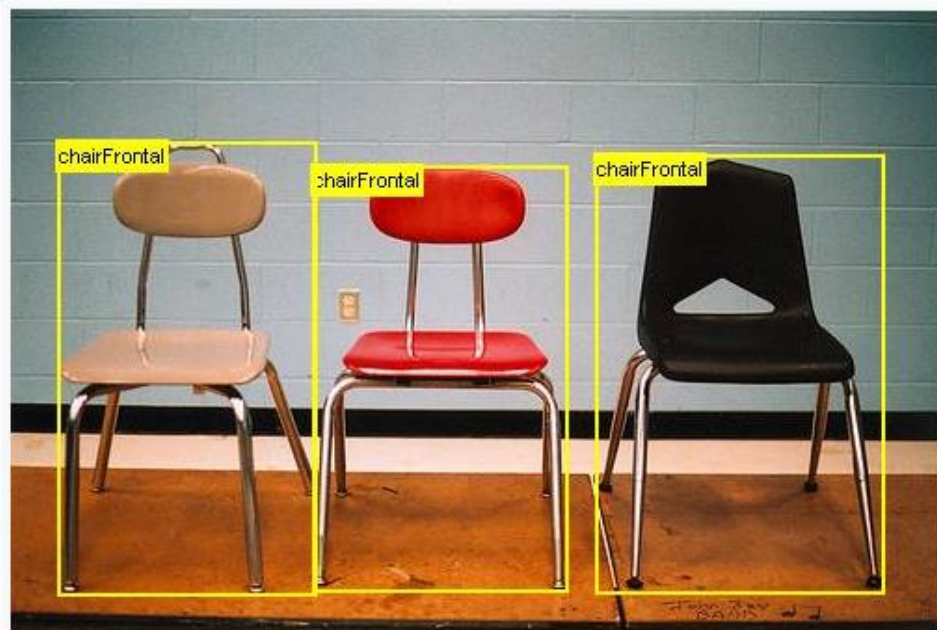


6. Deep learning application for computer vision

- (2) Image Classification With Localization



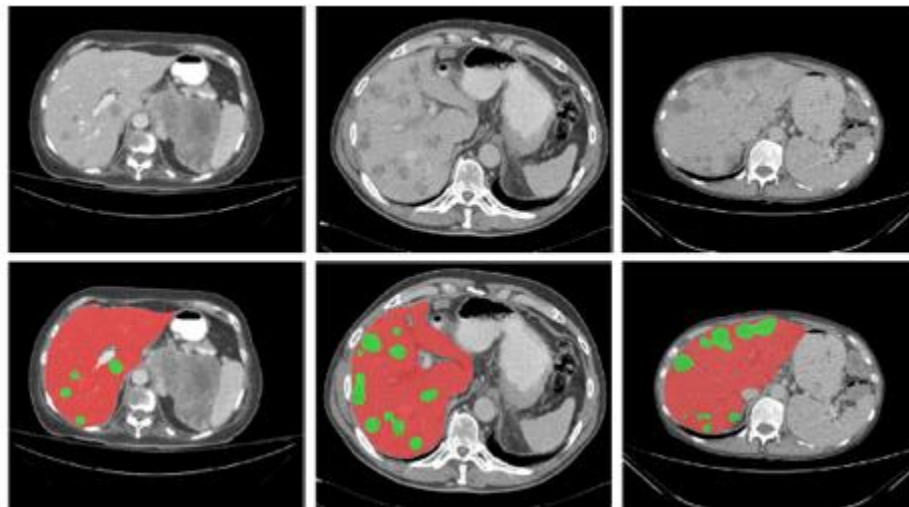
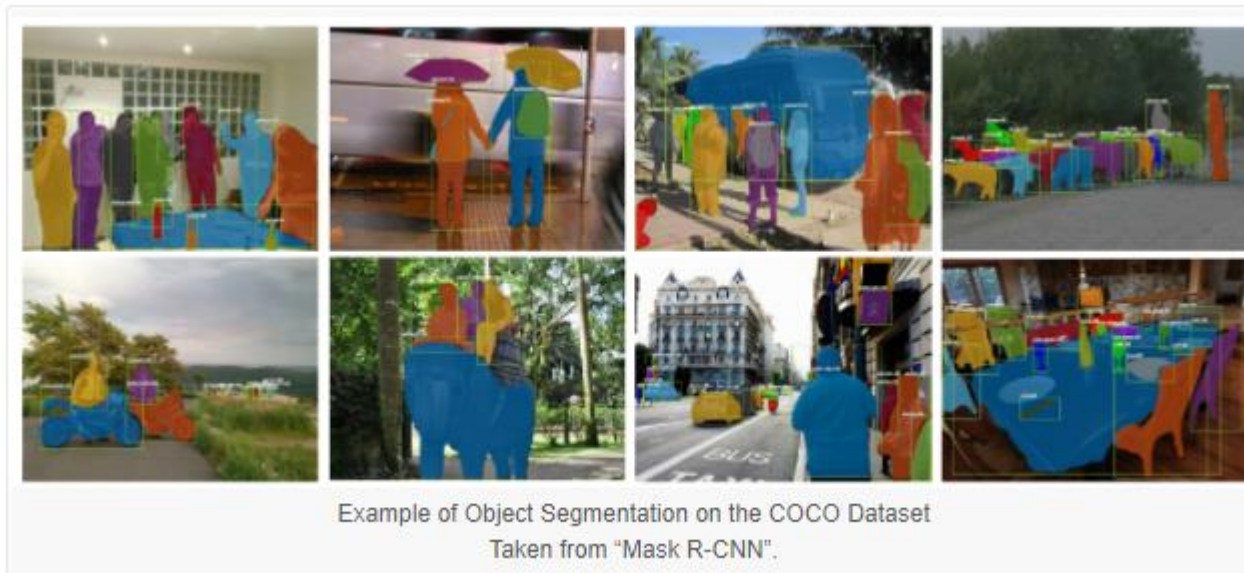
Example of Image Classification With Localization of a Dog from VOC 2012



Example of Image Classification With Localization of Multiple Chairs From VOC 2012

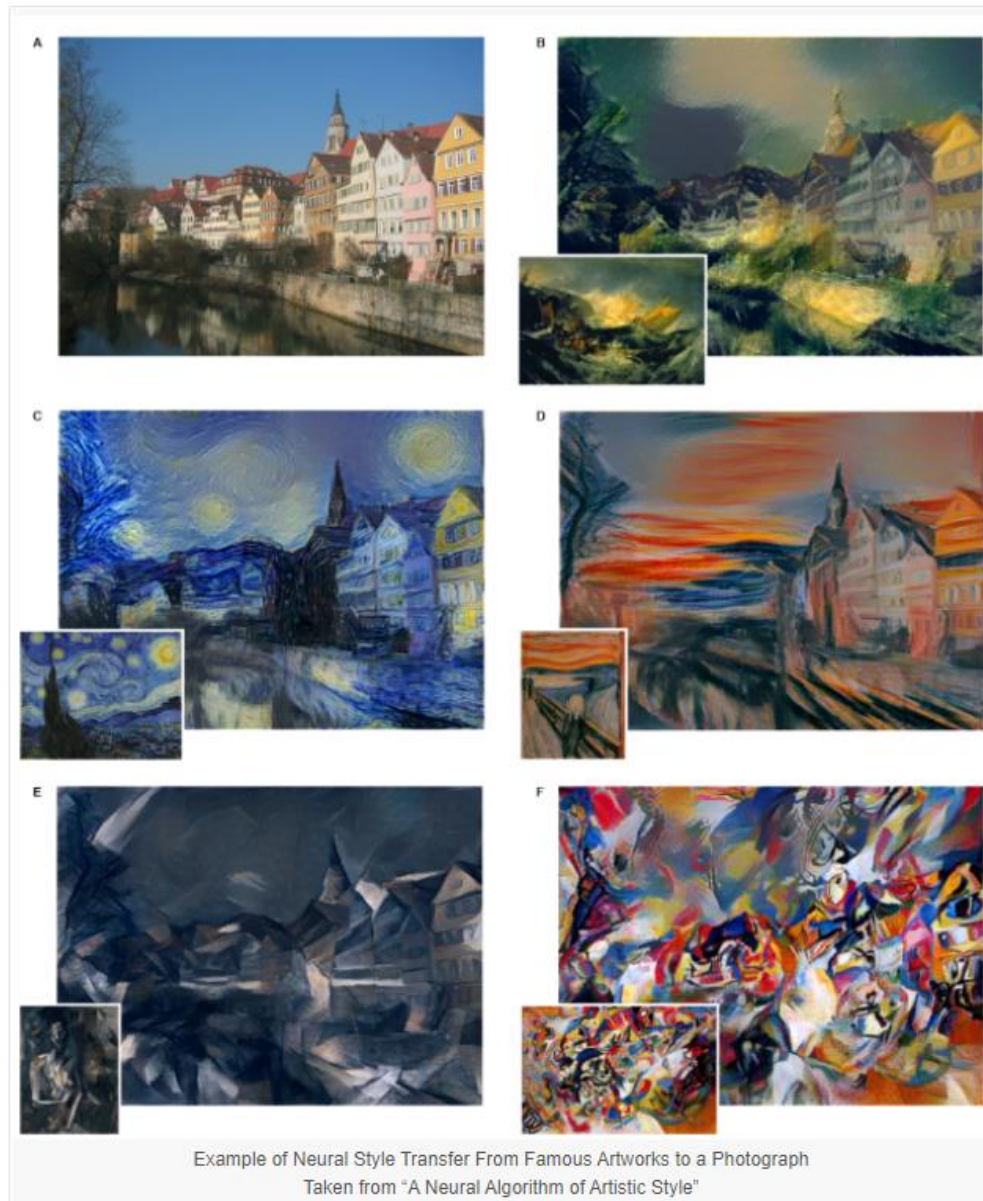
6. Deep learning application for computer vision

- (4) Object Segmentation



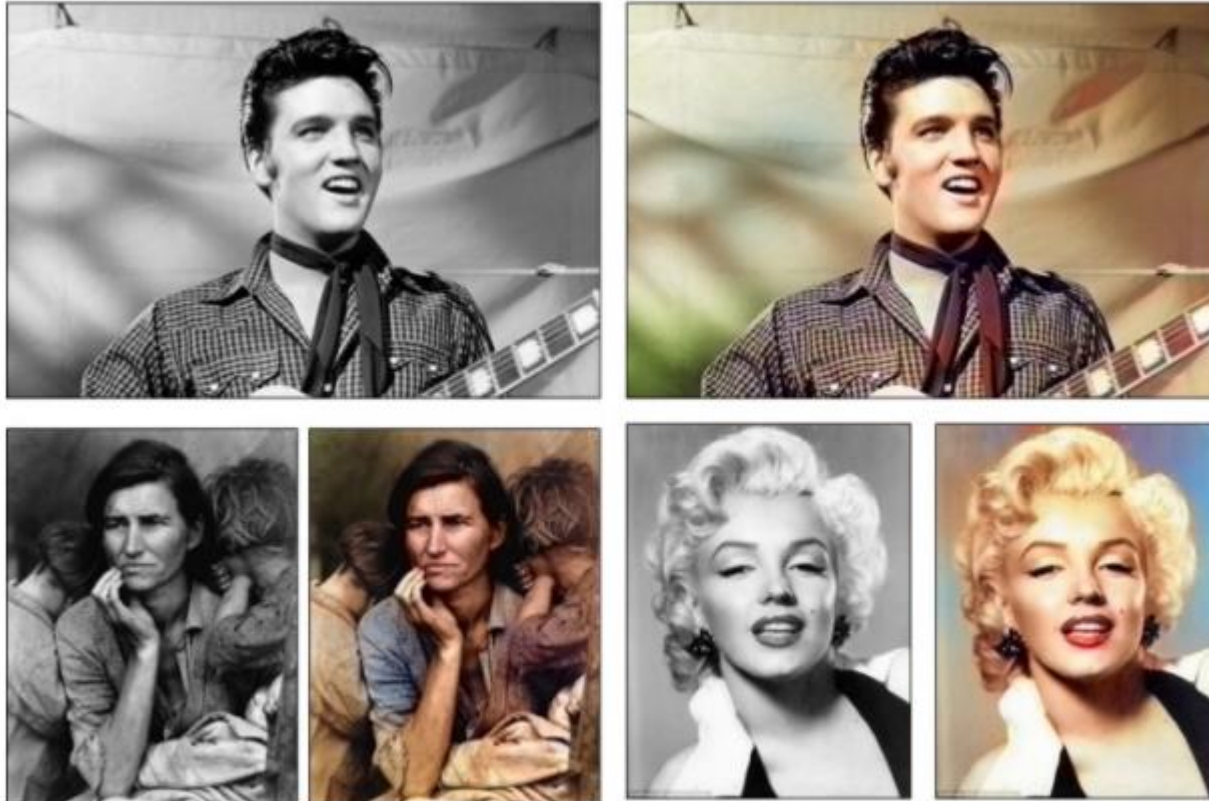
6. Deep learning application for computer vision

- (5) Style Transfer



6. Deep learning application for computer vision

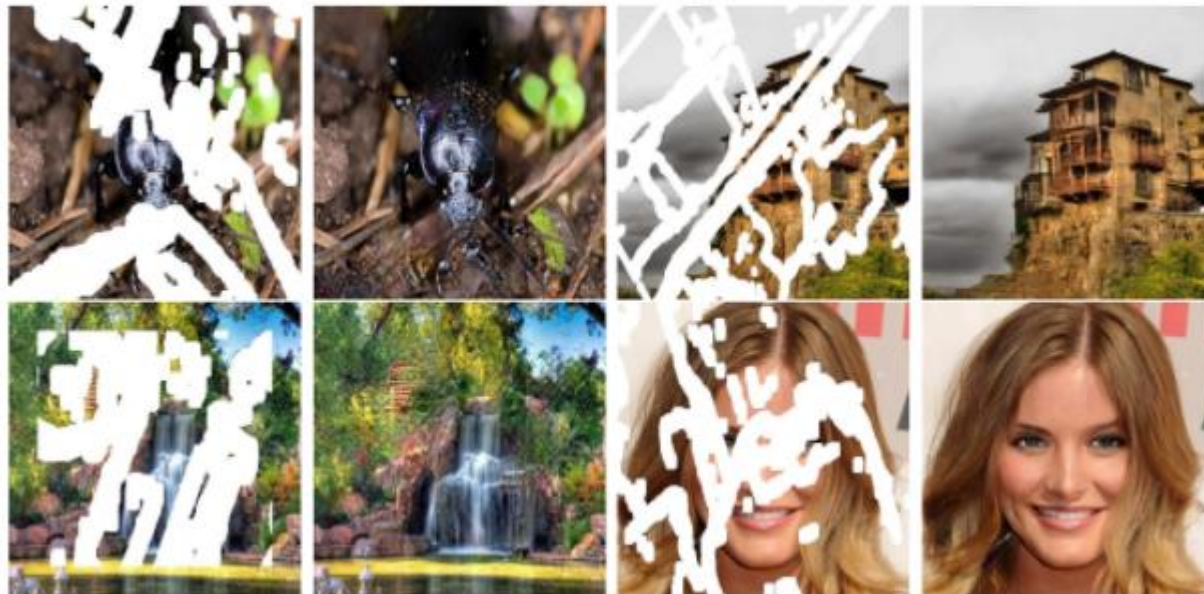
- (6) Image Colorization



Examples of Photo Colorization
Taken from "Colorful Image Colorization"

6. Deep learning application for computer vision

- (7) Image Reconstruction

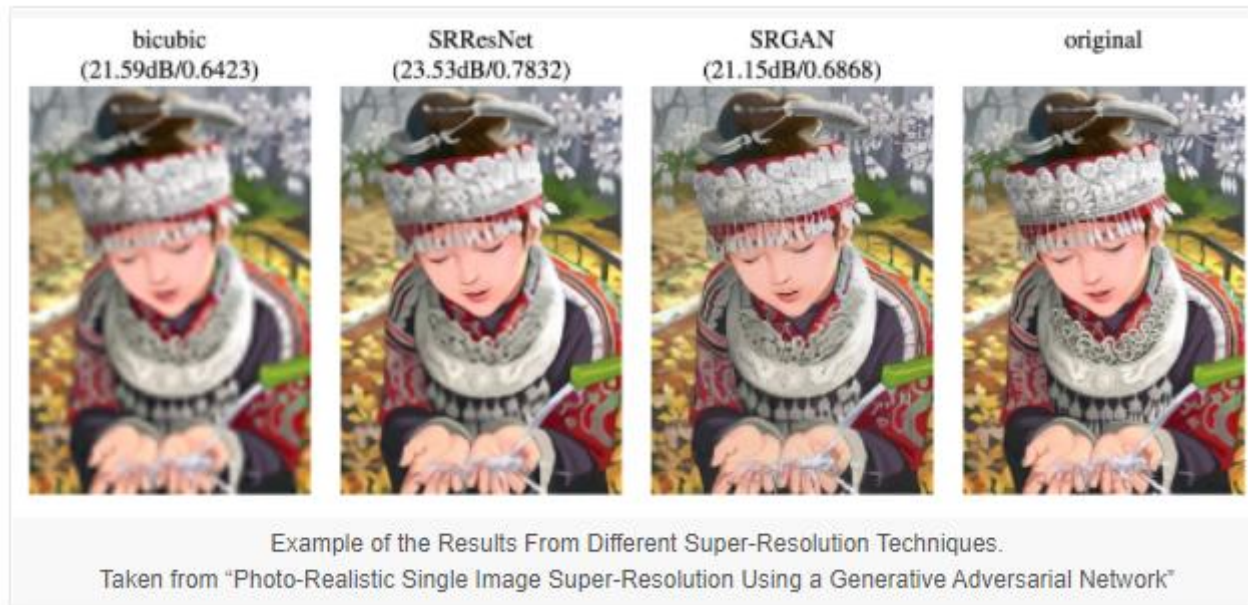


Example of Photo Inpainting.

Taken from "Image Inpainting for Irregular Holes Using Partial Convolutions"

6. Deep learning application for computer vision

- (8) Image Super-Resolution



6. Deep learning application for computer vision

- (9) Image Synthesis

