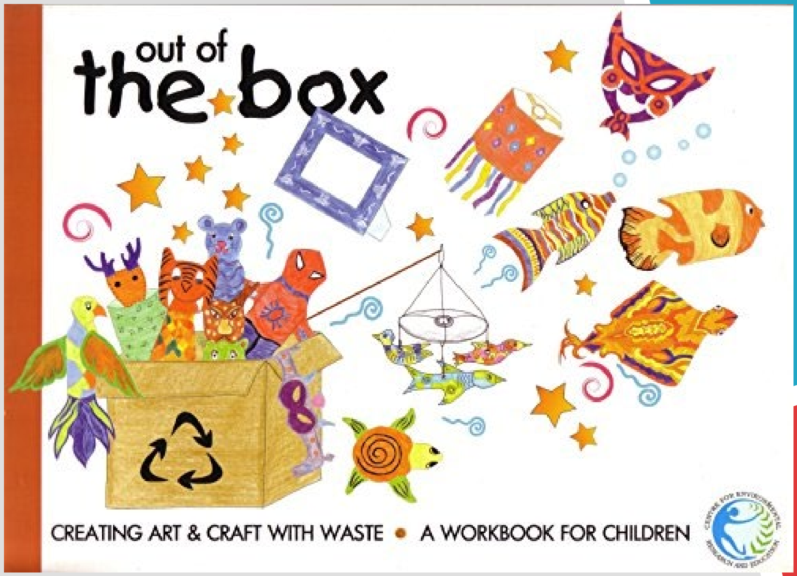
5장 *Deep learning for computer vision*

"Out of the Box"



5.2 Training a convnet from scratch on a small dataset

- classifying images as dogs or cats - 4,000 pictures of cats and dogs (2,000 cats, 2,000 dogs)
- 2,000 pictures for training—1,000 for partially training, 500 for validation, and 500 for testing.
- 2,000 training samples - classification accuracy of 71%
- data augmentation - mitigating overfitting, 82%
- feature extraction with a **pretrained network** - accuracy of 90% to 96%
- fine-tuning a **pretrained network** - final accuracy of 97%

5.2 Training a convnet from scratch on a small dataset

5.2.1 The relevance of deep learning for small-data problems

- convnets learn local, translation-invariant without the need for any custom feature engineering
- deep-learning models are by nature highly repurposable - an image-classification or speech-to-text model trained on a large-scale dataset and reuse it on a significantly different problem with only minor changes.
- many pretrained models (usually trained on the Image-Net dataset) are now publicly available for download and can be used to bootstrap powerful vision models out of very little data.

5.2 Training a convnet from scratch on a small dataset

5.2.2 Downloading the data

- Dogs vs. Cats dataset - Kaggle as part of a computer-vision competition in late 2013, won by entrants who used convnets (95% accuracy)
- download the original dataset from
  www.kaggle.com/c/dogs-vs-cats/data
- The pictures are medium-resolution color JPEGs. Figure 5.8 shows some examples.

Figure 5.8 Samples from the Dogs vs. Cats dataset. Sizes weren't modified: the samples are heterogeneous in size, appearance, and so on.

5.2 Training a convnet from scratch on a small dataset

5.2.2 Downloading the data

- This dataset contains 25,000 images of dogs and cats (12,500 from each class) and is 543 MB (compressed).
- training set - 1,000 samples * 2 class
- validation set with 500 samples * 2 class
- test set with 500 samples * 2 class

5.2 Training a convnet from scratch on a small dataset

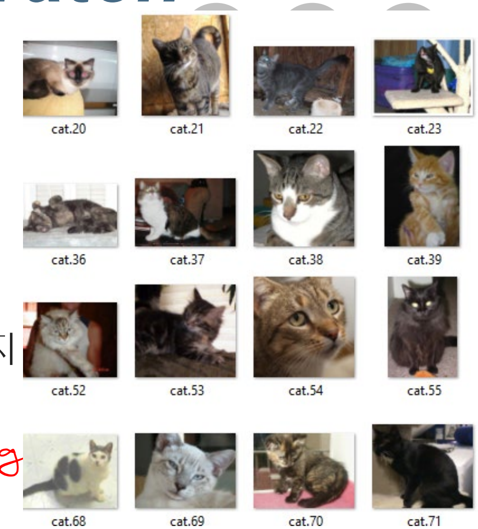
5.2.2 Downloading the data

Listing 5.4 Training the convnet on MNIST images

```
import os, shutil
original_dataset_dir = './datasets/cats_and_dogs/train' # 원본 데이터셋
base_dir = './datasets/cats_and_dogs_small' # 소규모 데이터셋
os.mkdir(base_dir)
train_dir = os.path.join(base_dir, 'train') # 훈련, 검증, 테스트 분할
os.mkdir(train_dir)
validation_dir = os.path.join(base_dir, 'validation')
os.mkdir(validation_dir)
test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)
train_cats_dir = os.path.join(train_dir, 'cats') # 훈련용 고양이
os.mkdir(train_cats_dir)
train_dogs_dir = os.path.join(train_dir, 'dogs') # 훈련용 강아지
os.mkdir(train_dogs_dir)
validation_cats_dir = os.path.join(validation_dir, 'cats') # 검증용 고양이
os.mkdir(validation_cats_dir)
validation_dogs_dir = os.path.join(validation_dir, 'dogs') # 검증용 강아지
os.mkdir(validation_dogs_dir)
test_cats_dir = os.path.join(test_dir, 'cats') # 테스트용 고양이
os.mkdir(test_cats_dir)
test_dogs_dir = os.path.join(test_dir, 'dogs') # 테스트용 강아지
os.mkdir(test_dogs_dir)
```

. / train / cats
/ dogs
. / validation / cats
/ dogs
. / test / cats
/ dogs

5.2 Training a convnet from scratch on a small dataset



5.2.2 Downloading the data

Listing 5.4 Training the convnet on MNIST images

/cat.253.jpg

```
fnames=['cat.{}.jpg'.format(i) for i in range(1000)]
                                              # 처음 1,000개의 고양이 이미지
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)   /cat.253.jpg
    dst = os.path.join(train_cats_dir, fname)
    shutil.copyfile(src, dst) # train_cats_dir에 복사
fnames=['cat.{}.jpg'.format(i) for i in range(1000, 1500)]#다음 500개 고양이 이미지
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_cats_dir, fname)
    shutil.copyfile(src, dst) # validation_cats_dir에 복사
fnames=['cat.{}.jpg'.format(i) for i in range(1500, 2000)] #다음 500개 고양이 이미지
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_cats_dir, fname)
    shutil.copyfile(src, dst) # test_cats_dir에 복사
```

5.2 Training a convnet from scratch on a small dataset

5.2.2 Downloading the data

Listing 5.4 Training the convnet on MNIST images

```
fnames=['dog.{}.jpg'.format(i) for i in range(1000)] # 처음 1,000개의 강아지 이미지
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_dogs_dir, fname)
    shutil.copyfile(src, dst) # train_dogs_dir에 복사
fnames=['dog.{}.jpg'.format(i) for i in range(1000, 1500)]#다음 500개 강아지 이미지
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_dogs_dir, fname)
    shutil.copyfile(src, dst) # validation_dogs_dir에 복사
fnames=['dog.{}.jpg'.format(i) for i in range(1500, 2000)]#다음 500개 강아지 이미지
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_dogs_dir, fname)
    shutil.copyfile(src, dst) # test_dogs_dir에 복사
```

5.2 Training a convnet from scratch on a small dataset

5.2.2 Downloading the data

- As a sanity check, let's count how many pictures are in each training split (train/validation/test):

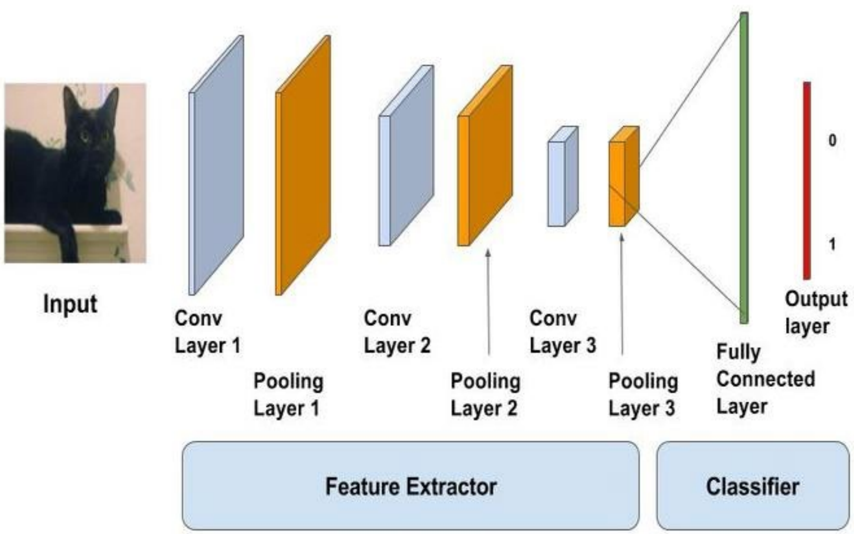
```
>>> print('total training cat images:', len(os.listdir(train_cats_dir)))
total training cat images: 1000
>>> print('total training dog images:', len(os.listdir(train_dogs_dir)))
total training dog images: 1000
>>> print('total validation cat images:', len(os.listdir(validation_cats_dir)))
total validation cat images: 500
>>> print('total validation dog images:', len(os.listdir(validation_dogs_dir)))
total validation dog images: 500
>>> print('total test cat images:', len(os.listdir(test_cats_dir)))
total test cat images: 500
>>> print('total test dog images:', len(os.listdir(test_dogs_dir)))
total test dog images: 500
```

- 2,000 training images
- 1,000 validation images
- 1,000 test images

5.2 Training a convnet from scratch on a small dataset

5.2.3 Building your network

- one more `Conv2D` + `MaxPooling2D` stage
- inputs of size 150×150 → feature maps of size 7×7 just before the `Flatten` layer.
- binary-classification problem - `Dense` layer of size 1 with a `sigmoid` activation.



5.2 Training a convnet from scratch on a small dataset

5.2.3 Building your network

Listing 5.5 Instantiating a small convnet for dogs vs. cats classification

```
from keras import layers from keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3),
      activation='relu', input_shape=(150, 150, 3))) # 148×148
model.add(layers.MaxPooling2D((2, 2)))                # 74×74
model.add(layers.Conv2D(64, (3, 3), activation='relu'))# 72×72
model.add(layers.MaxPooling2D((2, 2)))                # 36×36
model.add(layers.Conv2D(128, (3, 3), activation='relu'))# 34×34
model.add(layers.MaxPooling2D((2, 2)))                # 17×17
model.add(layers.Conv2D(128, (3, 3), activation='relu'))# 15×15
model.add(layers.MaxPooling2D((2, 2)))                # 7×7
model.add(layers.Flatten())                           # 6272
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

5.2 Training a convnet from scratch on a small dataset

5.2.3 Building your network

```
>>> model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

5.2 Training a convnet from scratch on a small dataset

5.2.3 Building your network

- compilation step - `RMSprop` optimizer
- ended with one sigmoid unit - binary crossentropy as the loss

Listing 5.6 Configuring the model for training

```
from keras import optimizers
model.compile(loss='binary_crossentropy',
        optimizer=optimizers.RMSprop(lr=1e-4),
        metrics=['acc'])
```

5.2 Training a convnet from scratch on a small dataset

5.2.4 Data preprocessing

- steps for getting it into the network are roughly as follows:
  1. Read the picture files.
  2. Decode the JPEG content to RGB grids of pixels.
  3. Convert these into floating-point tensors.
  4. Rescale the pixel values (between 0 and 255) to the [0, 1].

- Keras has a module with image-processing helper tools, located at `keras.preprocessing.image`.
- class **`ImageDataGenerator`** - automatically turn image files on disk into batches of preprocessed tensors.

5.2 Training a convnet from scratch on a small dataset

5.2.4 Data preprocessing

Listing 5.7 Using `ImageDataGenerator` to read images from directories

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255) # Rescale
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
         #이미지를 불러올 때 폴더명에 맞춰 자동으로 labelling
        train_dir,   # 타깃 디렉터리
        target_size=(150, 150), # JPEG content to 150×150 RGB
        batch_size=20,
        class_mode='binary') # 이진 레이블, 2개 폴더-cats, dogs
validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

/cats 0
/dogs 1

5.2 Training a convnet from scratch on a small dataset

5.2.4 Data preprocessing

- generator yields these batches indefinitely: break the iteration loop at some point:

```
for data_batch, labels_batch in train_generator:
    print('data batch shape:', data_batch.shape)
    print('labels batch shape:', labels_batch.shape)
    break
data batch shape: (20, 150, 150, 3)
labels batch shape: (20,)
```

- fit_generator = fit – yield batches of inputs and targets indefinitely
- **steps_per_epoch**: 20 batches from the generator, 100 steps until you see target of 2,000 samples.
- **validation_steps**: 20 batches from the generator, 50 steps until you see validation of 1,000 samples.

Listing 5.8 Fitting the model using a batch generator

```
history = model.fit_generator(train_generator, # 20
    steps_per_epoch=100, # 20 batches*100 steps=2000
    epochs=30,
    validation_data=validation_generator,
    validation_steps=50) # 20 batches * 50 steps=1000
```

5.2 Training a convnet from scratch on a small dataset

5.2.4 Data preprocessing

Listing 5.9 Saving the model

```
model.save('cats_and_dogs_small_1.h5')
```

Listing 5.10 Displaying curves of loss and accuracy during training

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))  30
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

5.2 Training a convnet from scratch on a small dataset

5.2.4 Data preprocessing

- overfitting - training accuracy reaches nearly 100%, whereas the validation accuracy stalls at 70–72%.
- The validation loss reaches its minimum after only five epochs and then stalls, whereas the training loss keeps decreasing linearly until it reaches nearly 0.
- relatively few training samples (2,000) - dropout and weight decay (L2 regularization), specific to computer vision: *data augmentation*

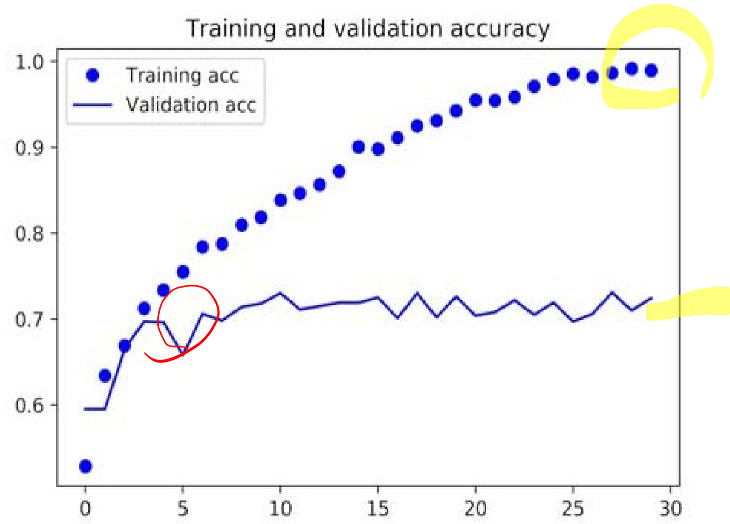


Figure 5.9 Training and validation accuracy

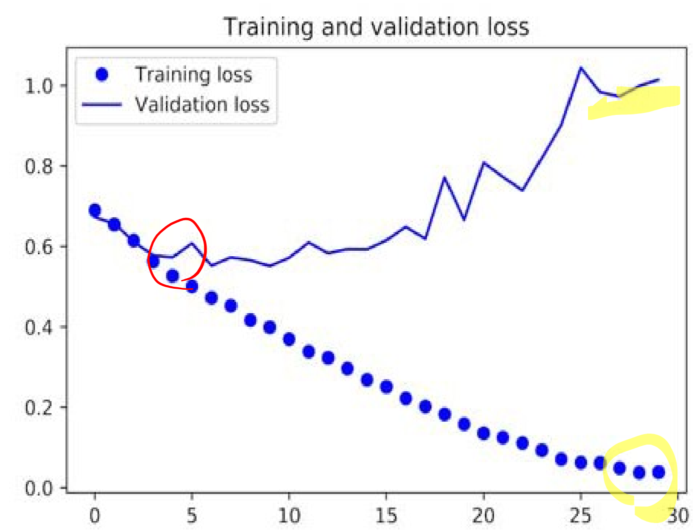


Figure 5.10 Training and validation loss

5.2 Training a convnet from scratch on a small dataset

- p.182-p.192의 Dogs vs. Cats 프로그램을 실행하고 아래와 같이 그림을 그리고 설명하시오.