- **Veri zenginleştirme**
  - **dogs-cats veri tabanından seçilen verilerle ağ eğitimi**
  - **Veri zenginleştirme ile aşırı uydurmanın (overfitting) azaltılması**
- **Önceden eğitilmiş ağ (pretrained network)**

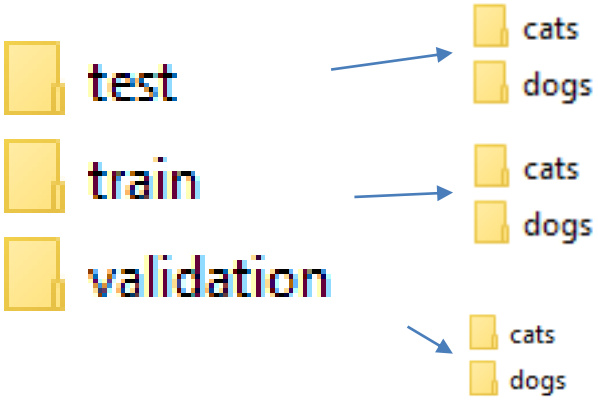
# Veri zenginleştirme (data augmentation)

- Veri arttırma ile modellin genelleştirme yapabilme kabiliyeti artırılır ve aşırı uydurma (overfitting) azaltılır.
- Veri arttırma, mevcut eğitim örneklerinden daha fazla eğitim verisi oluşturma yaklaşımını benimser.
- Örnekleri, inandırıcı görünen görüntüler veren bir dizi rasgele dönüşümle zenginleştirir.
- Veri zenginleştirmeyi açıklamadan önce ImageDataGenerator kullanılarak Cat-Dog tanıma örneği gerçekleştirilecektir

# Veri zenginleştirme (data augmentation)

- Veri zenginleştirmeyi açıklamadan önce ImageDataGenerator kullanılarak Cat-Dog tanıma örneği gerçekleştirilecektir.
- Cat-Dog örneği üzerinde veri zenginleştirme uygulanarak validation accuracy sonuçlarının iyileştiği gösterilecektir.
- Veri zenginleştirme ile birlikte önceden eğitilmiş ağ kullanılarak başarım ölçülecektir.
- Önceden eğitilmiş ağ üzerinde fine tuning yapılacak başarıma etkisi incelenecektir.

# Örnek: Küçük veri seti üzerinde veri zenginleştirme




- Bu uygulamada örnek veri setinden seçilen görüntüler ile eğitim gerçekleştirildikten sonra görüntüler üzerinde zenginleştirme yapılarak tekrar eğitim gerçekleştirilerek sonuçlar karşılaştırılacaktır.

Veri setinin içinden seçilen 4000 görüntü kullanılacaktır. Bunlar:

- Eğitim amaçlı toplam 2000 görüntü
- Geçerleme amaçlı toplam 1000 görüntü
- Test amaçlı 1000 görüntü
- https://www.kaggle.com/c/dogs-vs-cats/data


# Örnek: Ağ yapısı

- Önceki uygulamalara göre daha büyük boyutlu ve daha karmaşık bir problemi ele alacak ağ yapısında, 4 adet konvolüsyon katmanı ve 4 adet Max pooling kullanılmıştır.
- İlk katmana uygulanan 150x150 boyutlu görüntü son max pooling işleminden sonra 7x7 boyutlu 128 adet görüntüye dönüşmüştür.
- Düzleştirme (Flatten) işlemi ile eleman sayısı 7x7x128=6272 olan bir vektöre dönüştürülerek tam bağlantılı yoğun (Dense) ağa giriş olarak uygulanmıştır.
- Gizli katmanında 512 nöron bulunan yoğun ağın çıkışında ise sınıflandırmayı yapacak bir sigmoid nöron kullanılmıştır.

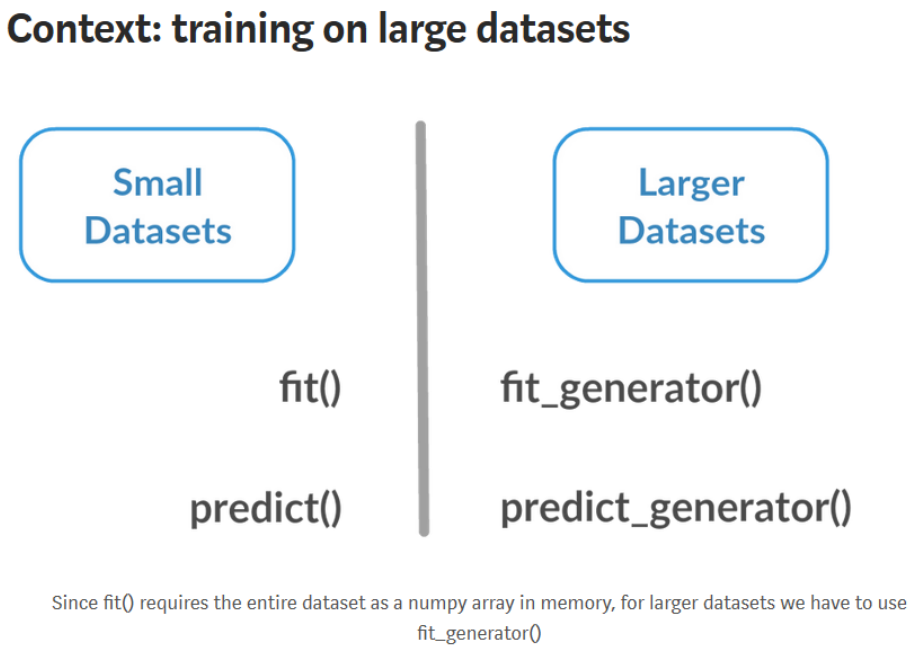
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()

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

```
_________________________________________________________________
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
_________________________________________________________________
```

Kaynak: https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/5.2-using-convnets-with-small-datasets.ipynb

# Büyük veri setleri üzerinde eğitim

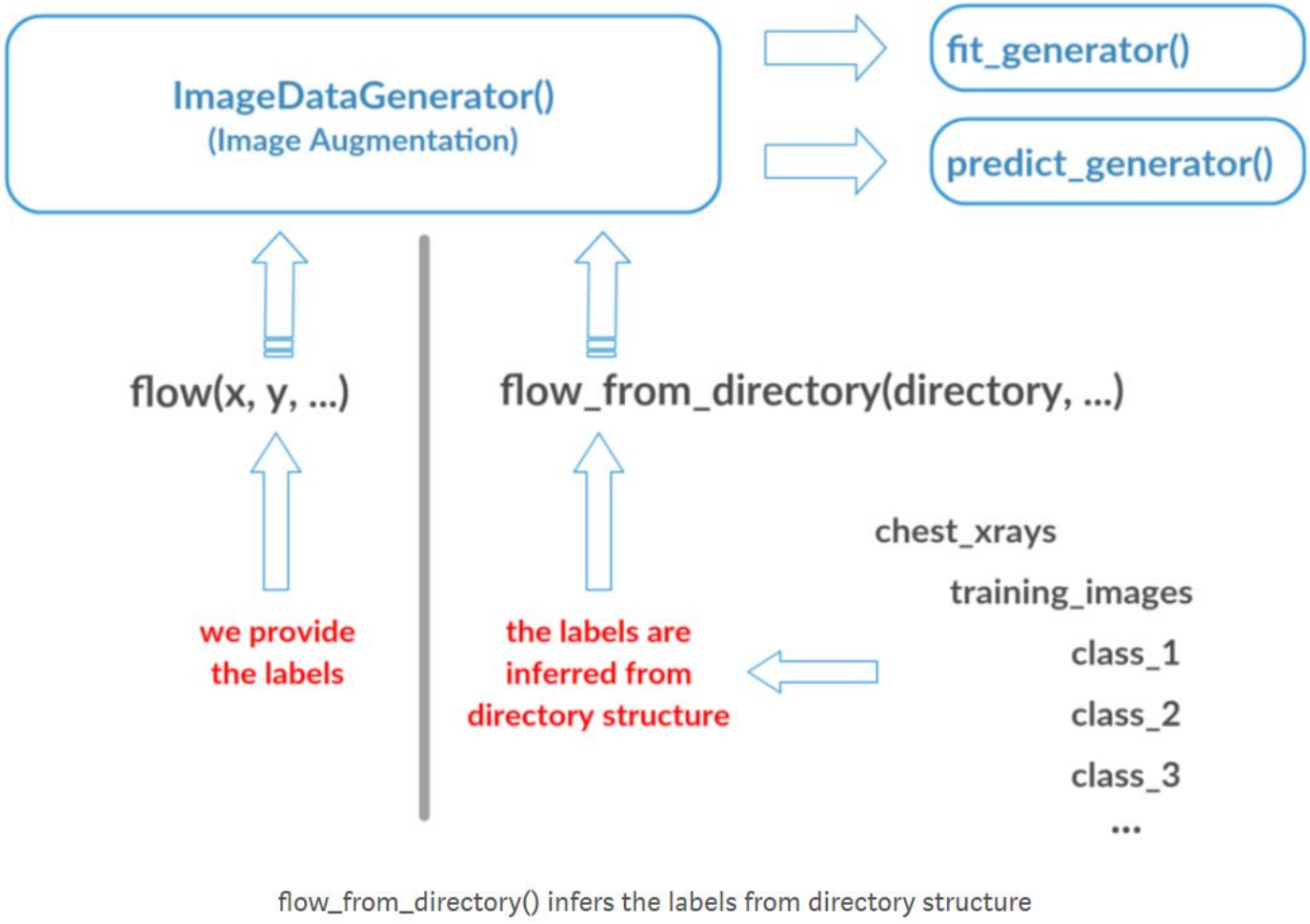
- Büyük veri setleri üzerinde eğitim gerçekleştirilirken genelde tüm veriyi belleğe getirmek yerine parçalar halinde getirilir. Bu durumda fit() yerine fit_generator(), predict() yerine predict_generator() kullanılır.




In Keras, using *fit()* and *predict()* is fine for smaller datasets which can be loaded into memory. But in practice, for most practical-use cases, almost all datasets are large and cannot be loaded into memory at once.

Kaynak: https://medium.com/difference-engine-ai/keras-a-thing-you-should-know-about-keras-if-you-plan-to-train-a-deep-learning-model-on-a-large-fdd63ce66bd2

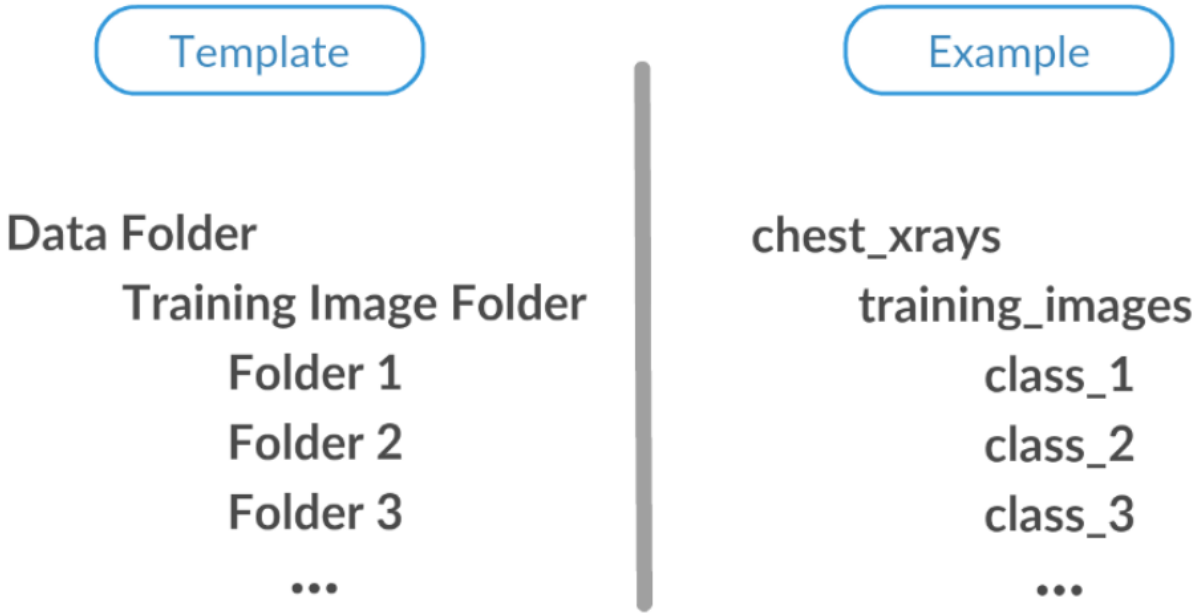
# Büyük veri setleri üzerinde eğitim




Kaynak: https://medium.com/difference-engine-ai/keras-a-thing-you-should-know-about-keras-if-you-plan-to-train-a-deep-learning-model-on-a-large-fdd63ce66bd2

# Büyük veri setleri üzerinde eğitim

- To *fit()*, or *fit_generator()* using *flow()* via *ImageDataGenerator()*, we supply the labels ourselves.
- *flow_from_directory()* automatically infers the labels from the directory structure of the folders containing images. Every subfolder inside the training-folder(or validation-folder) will be considered a target class.




flow_from_directory() automatically infers the labels from the directory structure of the folders

Kaynak: https://medium.com/difference-engine-ai/keras-a-thing-you-should-know-about-keras-if-you-plan-to-train-a-deep-learning-model-on-a-large-fdd63ce66bd2

# Örnek: ImageDataGenerator

```
from keras.preprocessing.image import ImageDataGenerator
# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 150x150
        target_size=(150, 150),
        #No. of images to be yielded from the generator per batch.
        batch_size=20,
        # Since we use binary_crossentropy loss,
        #we need binary labels
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

Veriler 0-1 arasına ölçeklendi

Tüm görüntüler 150x150 boyutlu

Yığın (batch) başına görüntü sayısı 20

İkili sınıflandırma

Aynı işlemler geçerleme (validation) için de gerçekleştiriliyor

# Örnek: Veri zenginleştirme

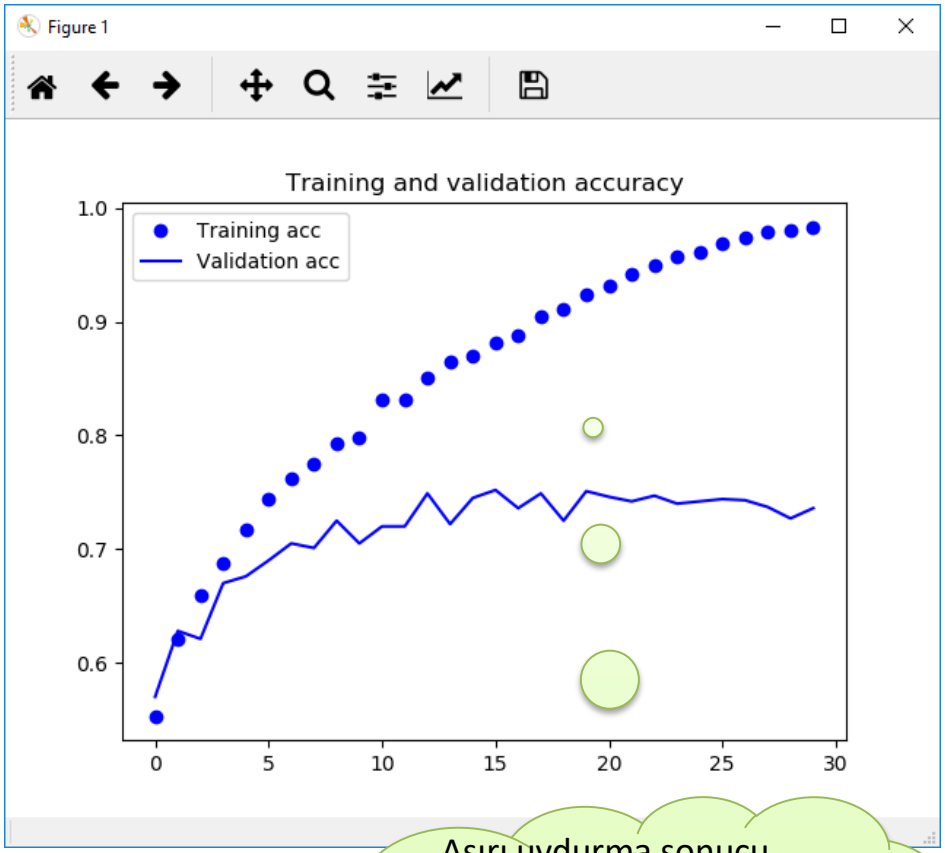
ImageDataGenerator kullandığımız için fit_generatör ile eğitiyoruz

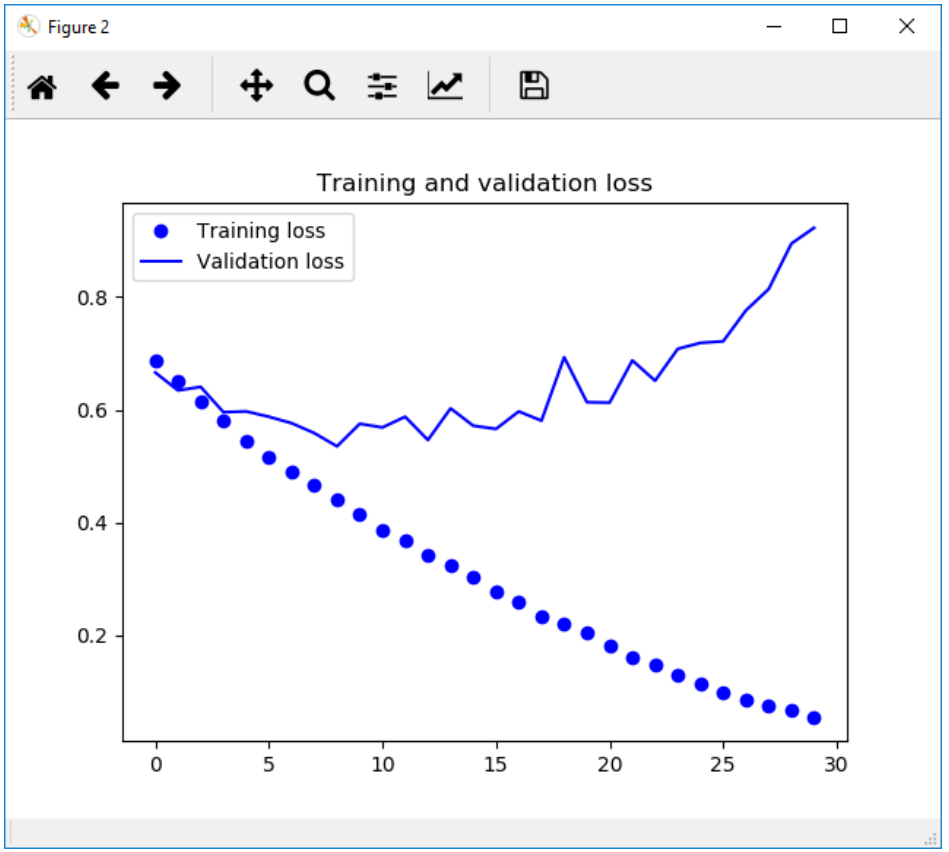
steps_per_epoch : ağırlıklar bir epoch içinde kaç kez güncelleneceğini belirtir. steps_per_epoch = TotalTrainingSamples / TrainingBatchSize

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)


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

validation_steps = TotalvalidationSamples / ValidationBatchSize

# Örnek: Eğitim sonucu




Aşırı uydurma sonucu eğitim (training) ve geçerleme (validation) arasındaki fark veri zenginleştirme ile azaltılabilir.

# Veri zenginleştirme (data augmentation)

```
fnames = [os.path.join(train_cats_dir, fname) for fname in os.listdir(train_cats_dir)]
# We pick one image to "augment"
img_path = fnames[10]

# Read the image and resize it
img = image.load_img(img_path, target_size=(150, 150))

# Convert it to a Numpy array with shape (150, 150, 3)
x = image.img_to_array(img)

# Reshape it to (1, 150, 150, 3)
x = x.reshape((1,) + x.shape)
# The .flow() command below generates batches of randomly transformed images.
# It will loop indefinitely, so we need to `break` the loop at some point!
i = 0
for batch in datagen.flow(x, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 4 == 0:
        break

plt.show()
```

Örnek bir görüntü üzerinde zenginleştirme

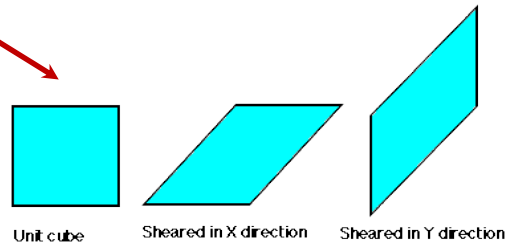
**fill_mode**: One of {"constant", "nearest", "reflect" or "wrap"}. Default is 'nearest'. Points outside the boundaries of the input are filled according to the given mode:

- 'constant': kkkkkkkk|abcd|kkkkkkkk (cval=k)
- 'nearest': aaaaaaaa|abcd|dddddddd
- 'reflect': abcddcba|abcd|dcbaabcd
- 'wrap': abcdabcd|abcd|abcdabcd

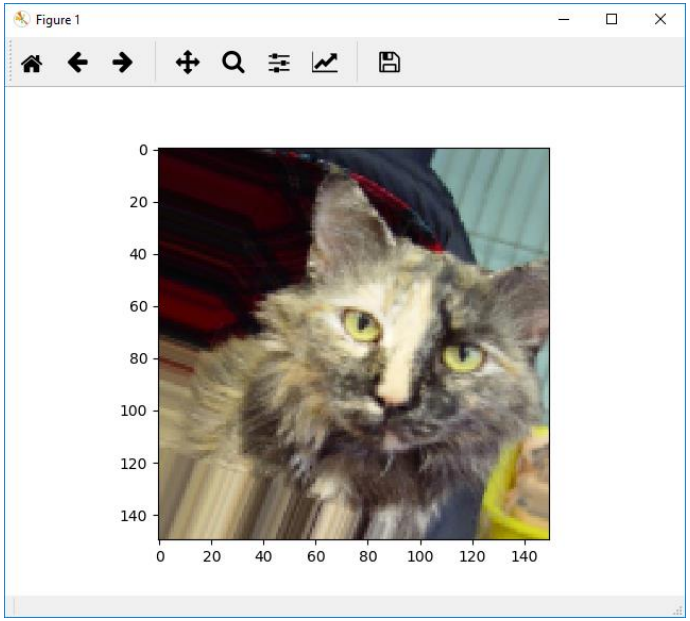
# ImageDataGenerator class

- **ImageDataGenerator class:** Generate batches of tensor image data with real-time data augmentation. The data will be looped over (in batches).
- **rotation_range:** is a value in degrees (0-180), a range within which to randomly rotate pictures.
- **width_shift and height_shift:** are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally.
- **shear_range:** is for randomly applying shearing transformations.
- **zoom_range:** is for randomly zooming inside pictures.
- **horizontal_flip:** is for randomly flipping half of the images horizontally -- relevant when there are no assumptions of horizontal asymmetry (e.g. real-world pictures).
- **fill_mode:** is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift.

```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

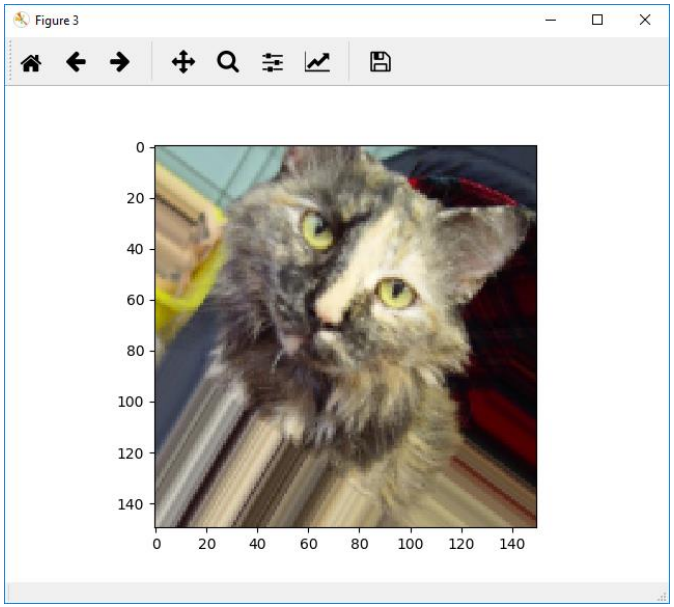


# Veri zenginleştirme (data augmentation)







# Veri zenginleştirme (data augmentation)

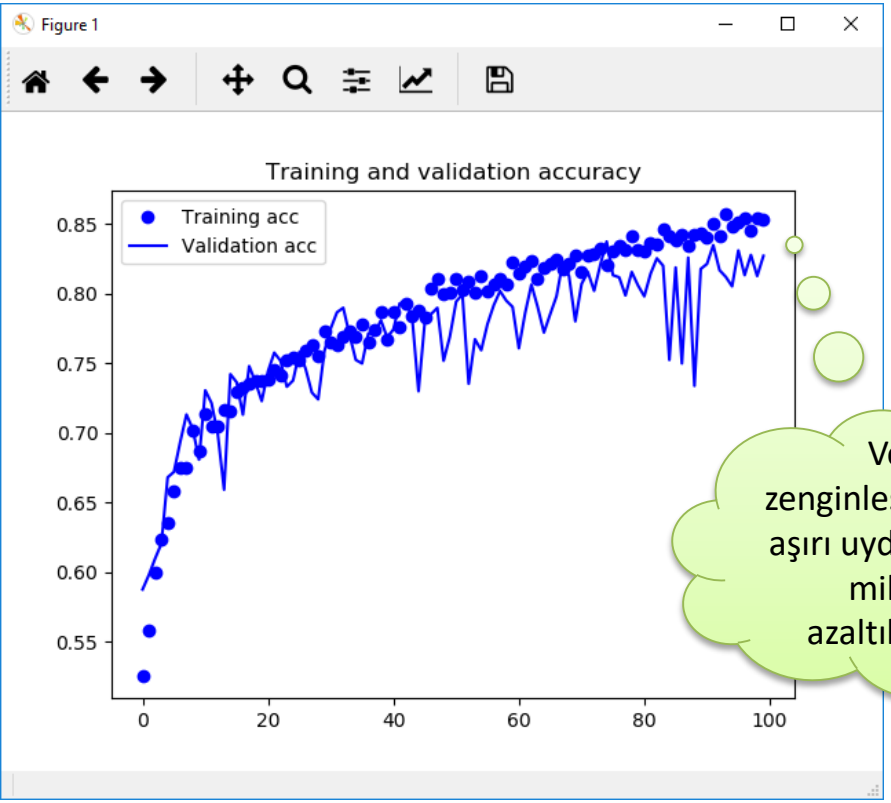
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

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

Aşırı uydurmayı (overfitting) düşürmek için dropout eklendi.

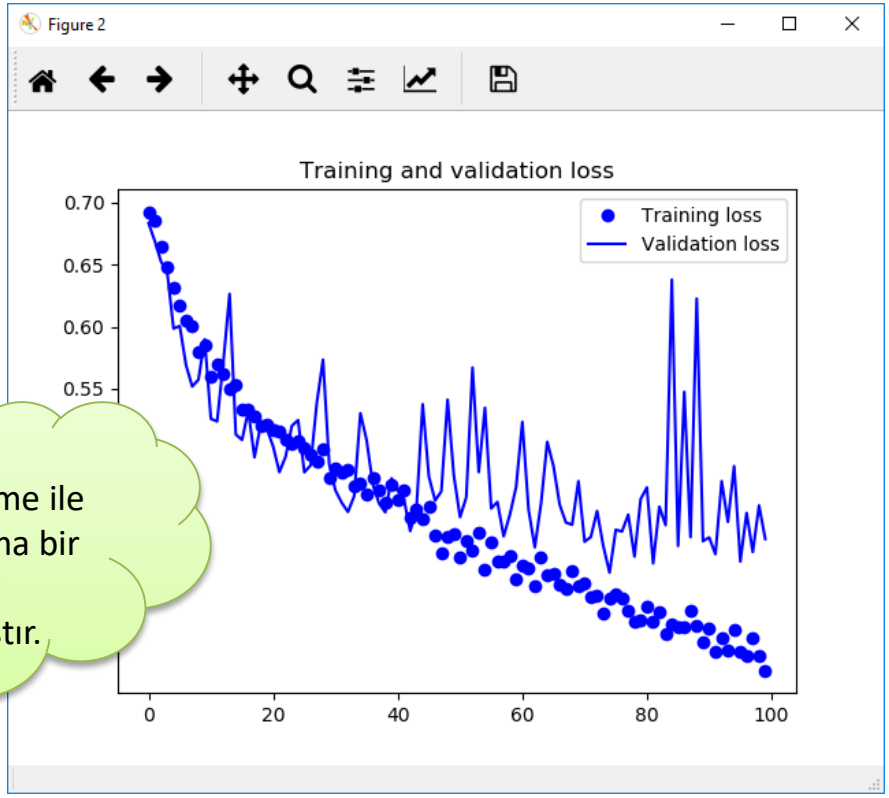
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)
```

Aşırı uydurmayı (overfitting) düşürmek için zenginleştirme eklendi.

# Örnek: Zenginleştirme sonrası eğitim sonucu

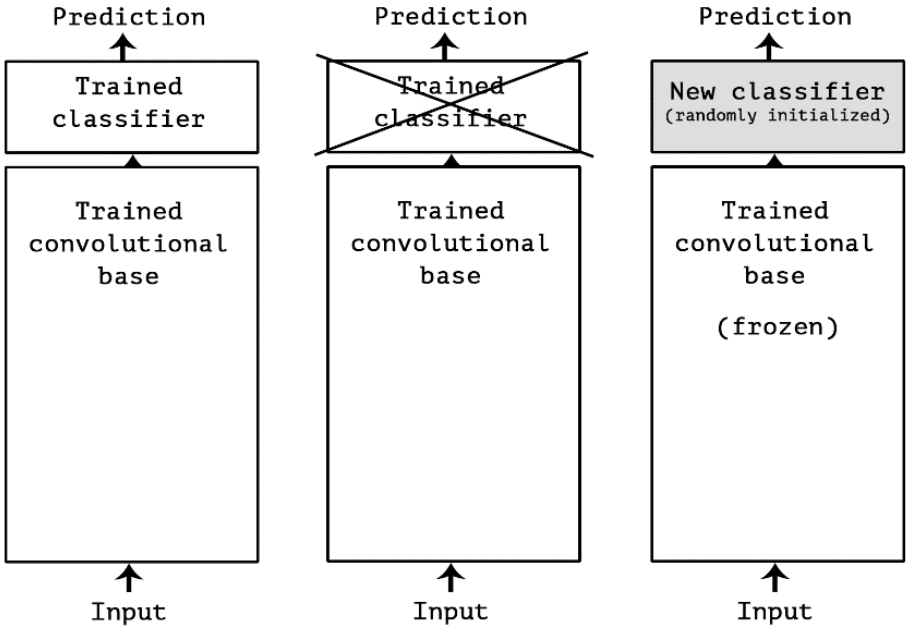



Veri zenginleştirme ile aşırı uydurma bir miktar azaltılmıştır.



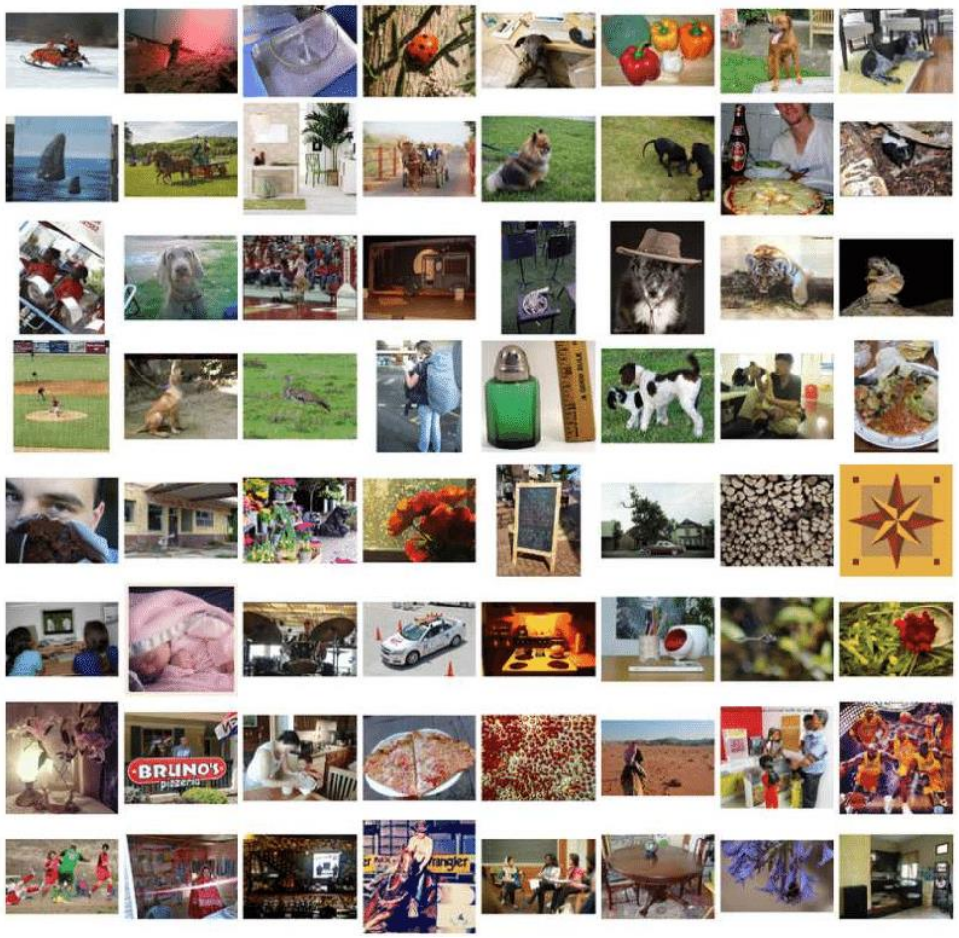
# • Önceden eğitilmiş ağ (pretrained network)

- Küçük görüntü veri setleri hakkında derin öğrenmeye yönelik yaygın ve etkili bir yaklaşım, önceden eğitilmiş (pretrained) bir ağdan yararlanmaktır.
- Önceden eğitilmiş bir ağ, daha önce büyük bir veri setinde, genellikle büyük ölçekli bir görüntü sınıflandırma görevinde eğitilmiş, ve sonra kullanılmak üzere kaydedilmiş bir ağdır.
- Bu orijinal veri seti yeterince büyük ve yeterince genelse, önceden eğitilmiş ağ tarafından öğrenilen spatial (konuma bağlı) özellik hiyerarşisi, görsel dünyamızın genel bir modeli olarak etkili bir şekilde hareket edebilir.
- Bu nedenle, özellikleri yeni problemler orijinal problemden tamamen farklı sınıflar içerebilse de, birçok farklı bilgisayarlı görme problemi için faydalı olabilir.



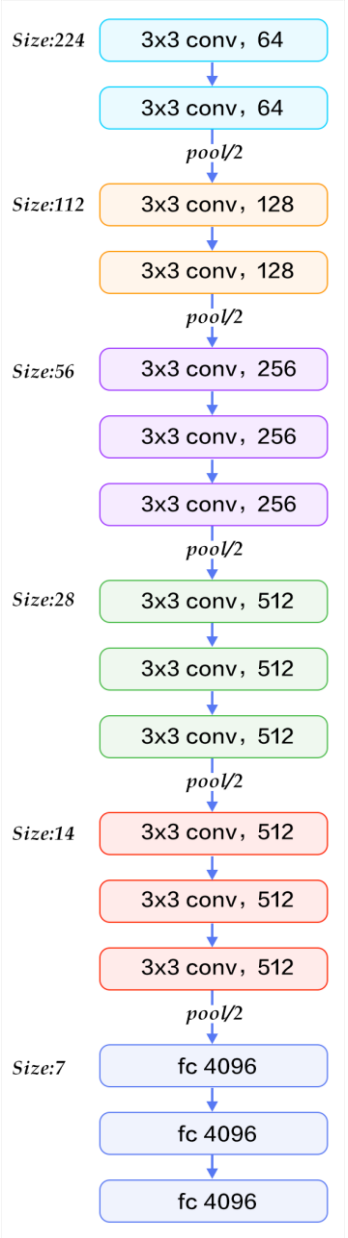

Kaynak: https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/5.3-using-a-pretrained-convnet.ipynb

# ImageNet veri tabanı

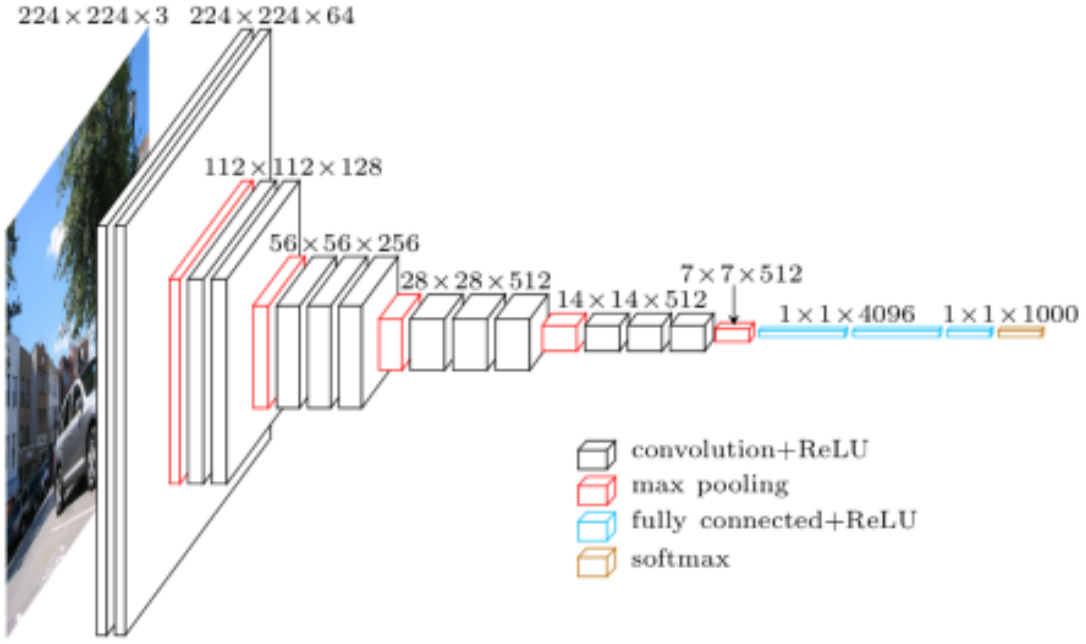



- 1.4 milyon etiketli görsel
- 1000 farklı sınıf.

# VGG16




- VGG16 :ImageNet için yaygın olarak kullanılan convnet mimarisi.



# Örnek: VGG16 ile cat-dog veri setinin eğitimi

```
import numpy as np
from keras.preprocessing.image import ImageDataGenerator

base_dir = 'E:\kagglecatsanddogs_3367a\PetImages\cats_and_dogs_small'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

datagen = ImageDataGenerator(rescale=1./255)
batch_size = 20
```

```
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))

conv_base.summary()
```

Önceden eğitilmiş ağı yükle

```
def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(
        directory,
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i += 1
        if i * batch_size >= sample_count:
            # Note that since generators yield data indefinitely in a loop,
            # we must `break` after every image has been seen once.
            break
    return features, labels
```

Seçilen görüntüleri ağa uygulayıp ağ çıkışını *features* dizisine kaydet. Görüntülerle ilgili etiketleri de *labels* dizisine kaydet

# Örnek: VGG16 ile cat-dog veri setinin eğitimi

```
train_features, train_labels = extract_features(train_dir, 2000)
validation_features, validation_labels = extract_features(validation_dir, 1000)
test_features, test_labels = extract_features(test_dir, 1000)

train_features = np.reshape(train_features, (2000, 4 * 4 * 512))
validation_features = np.reshape(validation_features, (1000, 4 * 4 * 512))
test_features = np.reshape(test_features, (1000, 4 * 4 * 512))
```

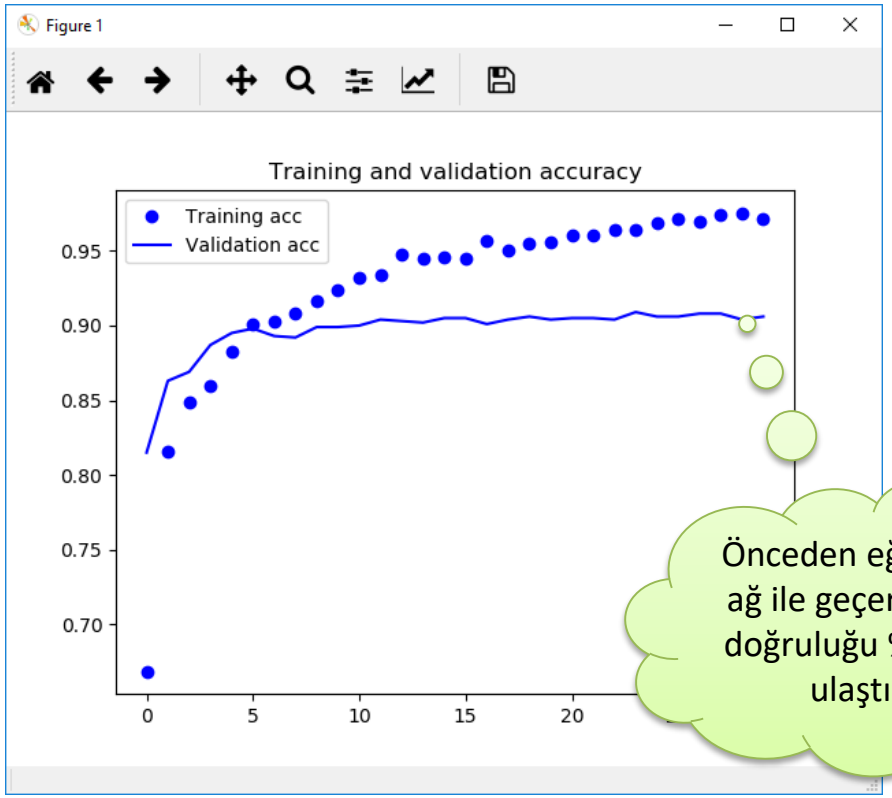
Train, validation ve test görüntüleri, tanımladığımız extract_features() ile önceden eğitimli ağa uygulanır.

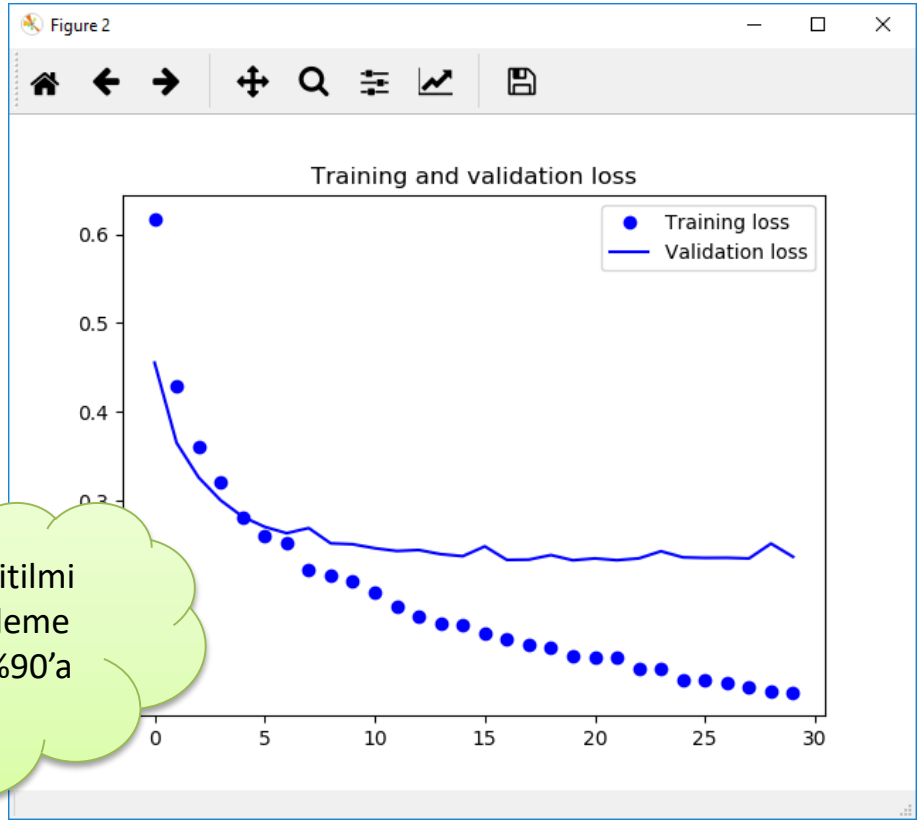
Eğiteceğimiz ağ sadece Dense katmanlardan oluşuyor. Conv2D katmanlarını hazır kullandığımız için eğiteceğimiz modelin içerisinde tekrar tanımlanmaz.

```
from keras import models
from keras import layers
from keras import optimizers

model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=20,
                    validation_data=(validation_features, validation_labels))
```

Önceden eğitimli konvolüsyon ağının çıkışındaki veriler Dense Layer katmanına uygulanarak ağı eğitmek için kullanılır.

# Örnek: VGG16 ile cat-dog veri setinin eğitimi




Önceden eğitilmi ağ ile geçerleme doğruluğu %90'a ulaştı

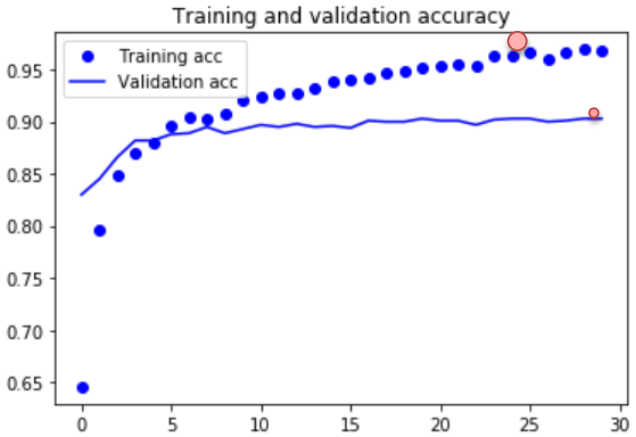
# VGG16 ile cat-dog veri setinin eğitimi

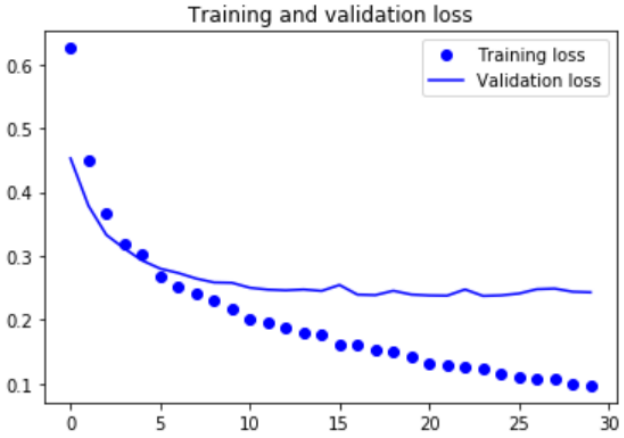
```
In [11]: model = models.Sequential()
    ...: model.add(conv_base)
    ...: model.add(layers.Flatten())
    ...: model.add(layers.Dense(256, activation='relu'))
    ...: model.add(layers.Dense(1, activation='sigmoid'))

In [11]:

In [12]: model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0
_________________________________________________________________

In [13]: conv_base.trainable=False

In [14]: model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 2,097,665
Non-trainable params: 14,714,688
_________________________________________________________________
```

Modeli tanımlarken konvolüsyon katmanı olarak VGG16 ağını kullanıyoruz.

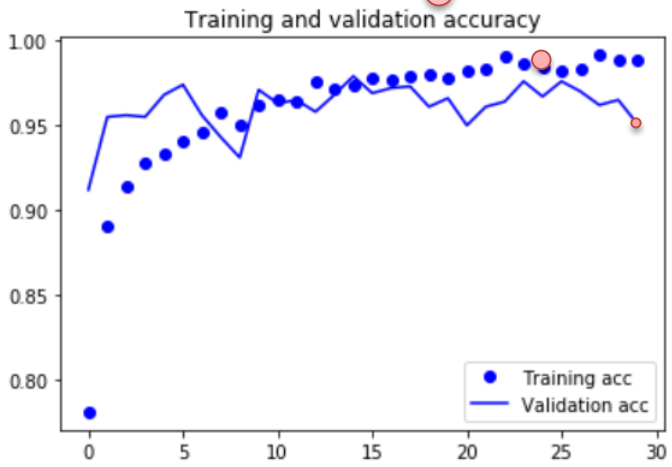
Öncenden eğitilmiş ağın parametrelerini kullanmak için donduruyoruz. Dondurmazsak VGG16 mimarisine ait parametreleri de eğitebiliriz. Ancak VGG16 daha önce 1.4 milyon görüntü ile eğitilmiş olduğu için, daha az veri ile eğittiğimiz ağın başarımını artırabilir. Daha önceki örnekte de bu ağı kullanmıştık ve eğitimden önce görüntüleri önce VGG16'dan geçirmiştik. Bu örnekte VGG16'yı da model yapısına eklediğimiz için buna gerek yok. Bu örnekte ayrıca veri zenginleştirme de yapılarak başarım artışı elde edilecektir.

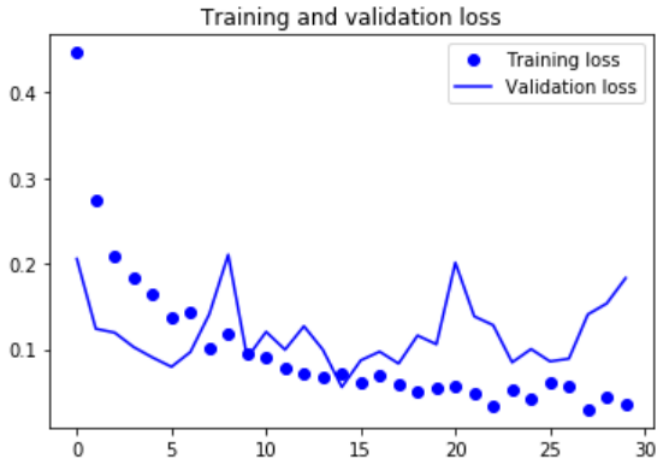
Ön eğitimli ile başarım %90 civarında elde edilmiştir.



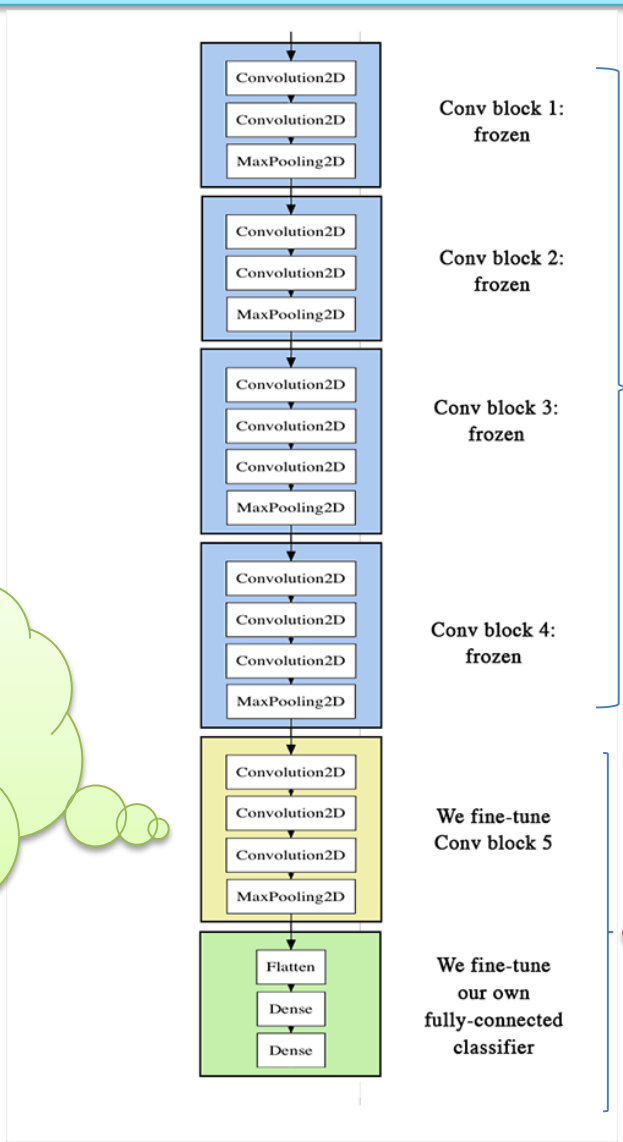




Ön eğitimli ağ ve veri zenginleştirme ile başarım %95 civarına çıkmıştır.



# • Önceden eğitilmiş ağ üzerinde ince ayar (fine tuning)



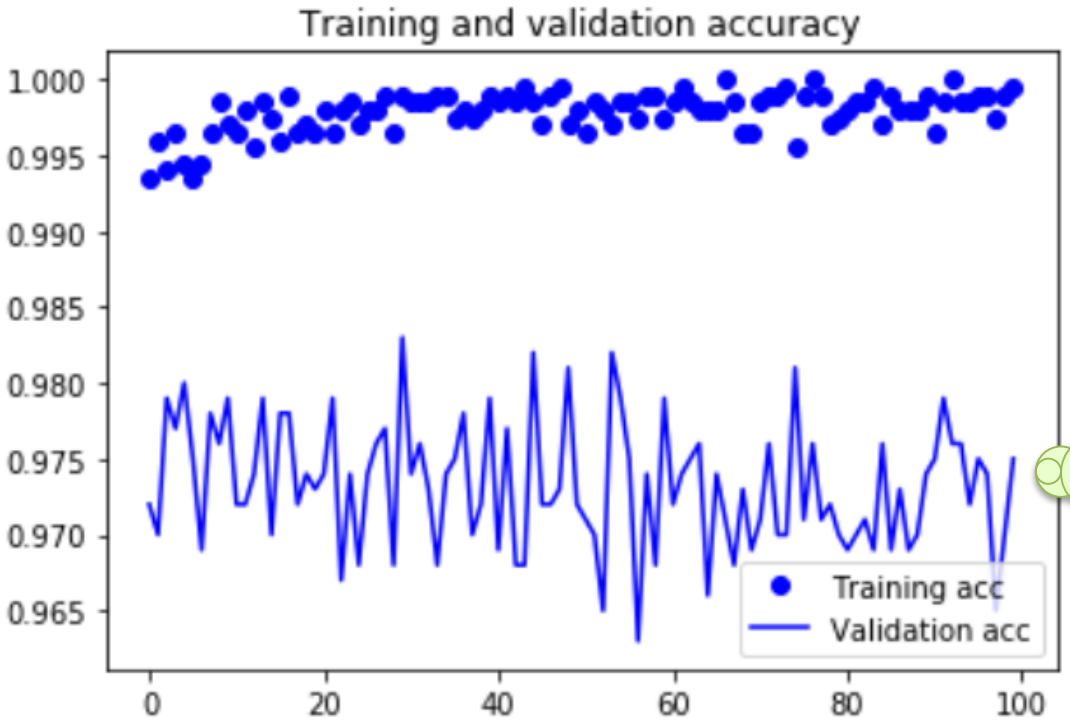

Önceden eğitilmiş katmanlar. Eğitim sırasında ağırlıklar önceden belirlendiği şekilde sabit kalır.

Önceden eğitilmiş konvolüsyon ağının sonundaki bloğu da eğitime dahil ediyoruz.

Eğitime dahil edilecek katmanlar

Kaynak: https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/5.3-using-a-pretrained-convnet.ipynb

# Örnek: İnce ayar sonucunda eğitim ve geçerleme doğrulukları




Önceden eğitilmiş ağın içerisindeki conv block 5 eğitime dahil edilmesiyle daha önce %90 olarak elde edilen geçerleme doğruluğu %97 civarına ulaştı