

- ...an bir evrişimli sinir ağının...
...nümkündür. Küçük veri setlerinde çansı... değerli bir yoldur.
- Öznitelik çıkarımı tamamlamak için hassas ayar ile daha önce öğrendiğiniz gösterimleri yeni probleme uyarlamak mümkün olur. Böylece performansı biraz artırabilirsiniz.

Bu noktada, resim sınıflandırma görevi ile özellikle de az veriniz varken baş etmek için gereken araçları öğrendiniz.

5.4 Evrişimli Sinir Ağının Öğrendiklerini Görselleştirmek

Derin öğrenme için "kara kutu" denildiğini duyarsınız: Öğrenilen gösterimlerin insan tarafından görülmesi, anlaşılması ve çözümlenmesi çok zordur. Bazı derin öğrenme modelleri için kısmen doğru olsa da evrişimli sinir ağları için doğru değildir. Evrişimli sinir ağları tarafından öğrenilen gösterimler görsel kavramların gösterimi olduklarından görselleştirme için çok uygundur. 2013 yılından beri görselleştirme ve anlamlandırma için birçok teknik geliştirildi. Hepsine deginmeyeceğiz ama en ulaşılabilir olanlarının üzerinden geçeceğiz:

- *Evrişimli sinir ağlarının ara çıktılarını görselleştirmek* –Birbirini takip eden katmanların girdilerini nasıl dönüştürdüklerini anlamak ve her bir evrişim filtresi hakkında bilgi sahibi olabilmek için kullanışlıdır.
- *Evrişim filtrelerini görselleştirmek* –Hangi görsel örüntülerin ya da kavramların öğrenildiğini anlamak için kullanışlıdır.
- *Bir resimde sınıf aktivasyon ısı haritalarını görselleştirmek* –Bir resmin hangi bölümlerinin verilen sınıf'a ait olduğunu anlamak için kullanılır ve nesnenin yeri hakkında bilgi verir.

Birinci metot için -ara çıktıları görselleştirme- Bölüm-5.2'de kedi-köpek sınıfıdan sıfırdan eğittiğimiz küçük evrişimli sinir ağını kullanacağız. Diğer iki teknik için Bölüm-5.3'te tanadığınız VGG16 modelini kullanacağız.

5.4.1 Ara Çıktıları Görselleştirmek

Ara çıktıları görselleştirmek farklı evrişim veya biriktirme katmanlarının belli bir girdi için öznitelik harmasını (katman çıktısı bazen *aktivasyon* olarak adlandırılır ve aktivasyon fonksiyonunun çıktısı anlamındadır) görüntülemektedir. Bu, bize ağın öğrendiği farklı filtrelerde girdinin nasıl parçalara ayrıldığı hakkında bilgi verir. Nitelik haritalarını üç boyutta (genişlik, yükseklik, kanal) görselleştirmeyi istersiniz. Her kanal göreceli bağımsız nitelikler kodladığından görselleştirmede tüm kanalları 2B resim olarak çizdirmek en iyisidir. Önce Bölüm-5.2'de kaydettığınız modeli yükleyelim:

```
>>> from keras.models import load_model
>>> model = load_model('cats_and_dogs_small_2.h5')
>>> model.summary()

Layer (type)          Output Shape         Param #
conv2d_5 (Conv2D)     (None, 148, 148, 32)    896
maxpooling2d_5 (MaxPooling2D) (None, 74, 74, 32)    0
conv2d_6 (Conv2D)     (None, 72, 72, 64)      18496
maxpooling2d_6 (MaxPooling2D) (None, 36, 36, 64)    0
conv2d_7 (Conv2D)     (None, 34, 34, 128)     73856
maxpooling2d_7 (MaxPooling2D) (None, 17, 17, 128)    0
conv2d_8 (Conv2D)     (None, 15, 15, 128)     147584
maxpooling2d_8 (MaxPooling2D) (None, 7, 7, 128)    0
flatten_2 (Flatten)   (None, 6272)           0
dropout_1 (Dropout)   (None, 6272)           0
dense_3 (Dense)      (None, 512)            3211776
```

```

dense_4 (Dense)          (None, 1)
=====
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0

```

Şimdi de ağı eğitmekte kullanmadığınız bir kedi resmi bulun.

Kod 5.25 : Bir resmi önişlemek

```



```

Bu resmi görelim (Şekil-5.24):

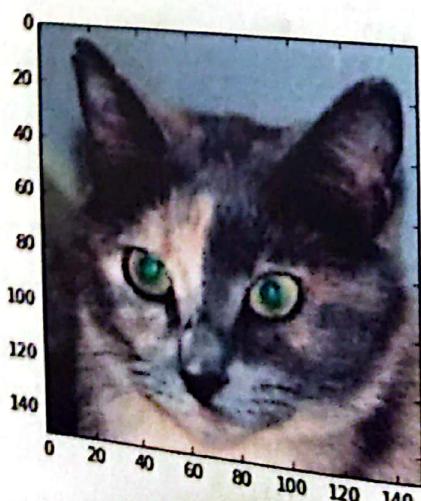
Kod 5.26 : Test resmini görmek

```

import matplotlib.pyplot as plt

plt.imshow(img_tensor[0])
plt.show()

```



Sekil 5.24. *test resmi*

Bakmak istediğiniz nitelik haritalarını çıkartmak için girdi olarak resim yiğinları verdiğinizde, çıktı olarak tüm evrişim ve biriktirme katmanlarının aktivasyonlarını veren bir Keras modeli oluşturacaksınız. Bunun için Keras'in model sınıfını kullanacaksınız. Sonuçta elde edeceğiniz model aşina olduğunuz Sequential modeller gibi girdilere karşılık gelen çıktıları eşleyecek. Model sınıfını kullanıldığınızda Sequential sınıfından farklı olarak çoklu çıktı alabileceksiniz. Model sınıfının detayları için Bölüm-7.1'e bakabilirsiniz.

Kod 5.27 : Giriş ve çıkış tensörü listesinden model oluşturmak

```
from keras import models

# İlk sekiz katmanın çıktılarını alır.
layer_outputs = [layer.output for layer in model.layers[:8]]

# Verilen model girdisine göre çıkış döndüren bir model örneği oluşturur.
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
```

Girdi olarak bir resim verildiğinde model özgün modelin katman aktivasyonlarını çıktı olarak verecek. Bu kitapta ilk defa çoklu çıktı ile karşılaşışınız, şimdide dek gördüğünüz modellerin sadece bir girdisi ve bir çıktıtı vardı. Genel olarak bir model herhangi bir sayıda girdi ve çıktıya sahip olabilir. Bu model, bir girdiye sekiz çıktı verecek: Her katman aktivasyonu bir çıktı olacak şekilde.

Kod 5.28 : Modeli tahmin modunda çalışırmak

```
activations = activation_model.predict(img_tensor) # Her katman aktivasyonu
# için bir Numpy dizisi döner.
```

Test resmine ilk evrişimli katmanın aktivasyonları:

```
>>> first_layer_activation = activations[0]
>>> print(first_layer_activation.shape)
(1, 148, 148, 32)
```

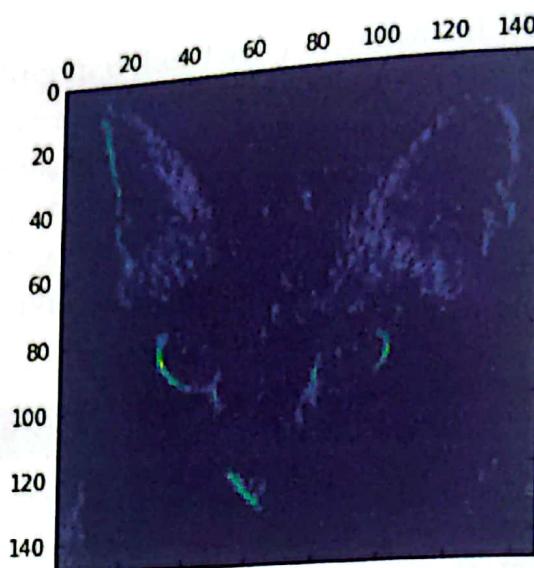
148×148 'lik 32 kanallı bir öznitelik haritası var. Örnek olarak ilk katmanın aktivasyonlarının dördüncü kanalını çizdirmeyi deneyelim (Şekil-5.25).

Kod 5.29 : Dördüncü kanalı görselleştirmek

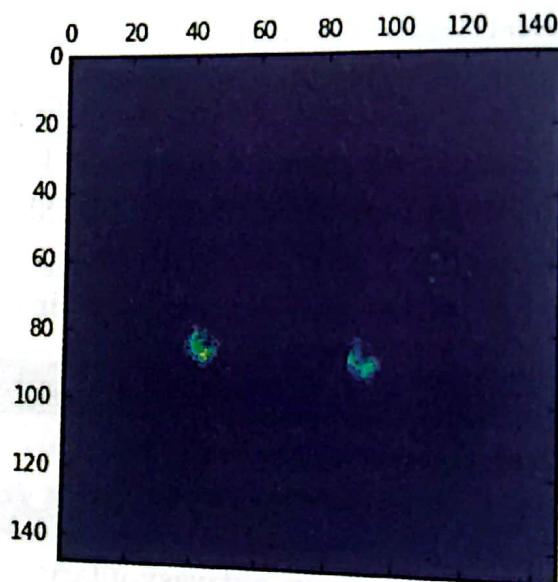
```
import matplotlib.pyplot as plt

plt.matshow(first_layer_activation[0, :, :, 4], cmap='viridis')
```

Bu kanal diagonal köşe dedektörüne benzıyor. Şimdiye yedinci kanalı çizdirelim (Şekil-5.26). Sizin kanallarınızın evrişim katmanlarının öğrendikleri deterministik olamadığı için farklı olabileceğini unutmayın.



Şekil 5.25: İlk katmanın dördüncü kanalının test resmi için aktivasyonu



Şekil 5.26: İlk katmanın yedinci kanalının test resmi için aktivasyonu

Kod 5.30 : Yedinci kanalı görselleştirmek

```
plt.matshow(first_layer_activation[0, :, :, 7], cmap='viridis')
```

Şekil-5.26'da görülen aktivasyon açık yeşil nokta dedektörüne benziyor: Kedinin gözlerini yakalamak için bu noktada ağıın tüm aktivasyonlarını görselleştirelim (Şekil-5.27). Bunun için sekiz aktivasyon haritasının her kanalını alıp kanallar yan yana olacak şekilde büyük bir resim tensörü oluşturacaksınız.

Kod 5.31 : Tüm kanalların tüm aktivasyonlarını görselleştirmek

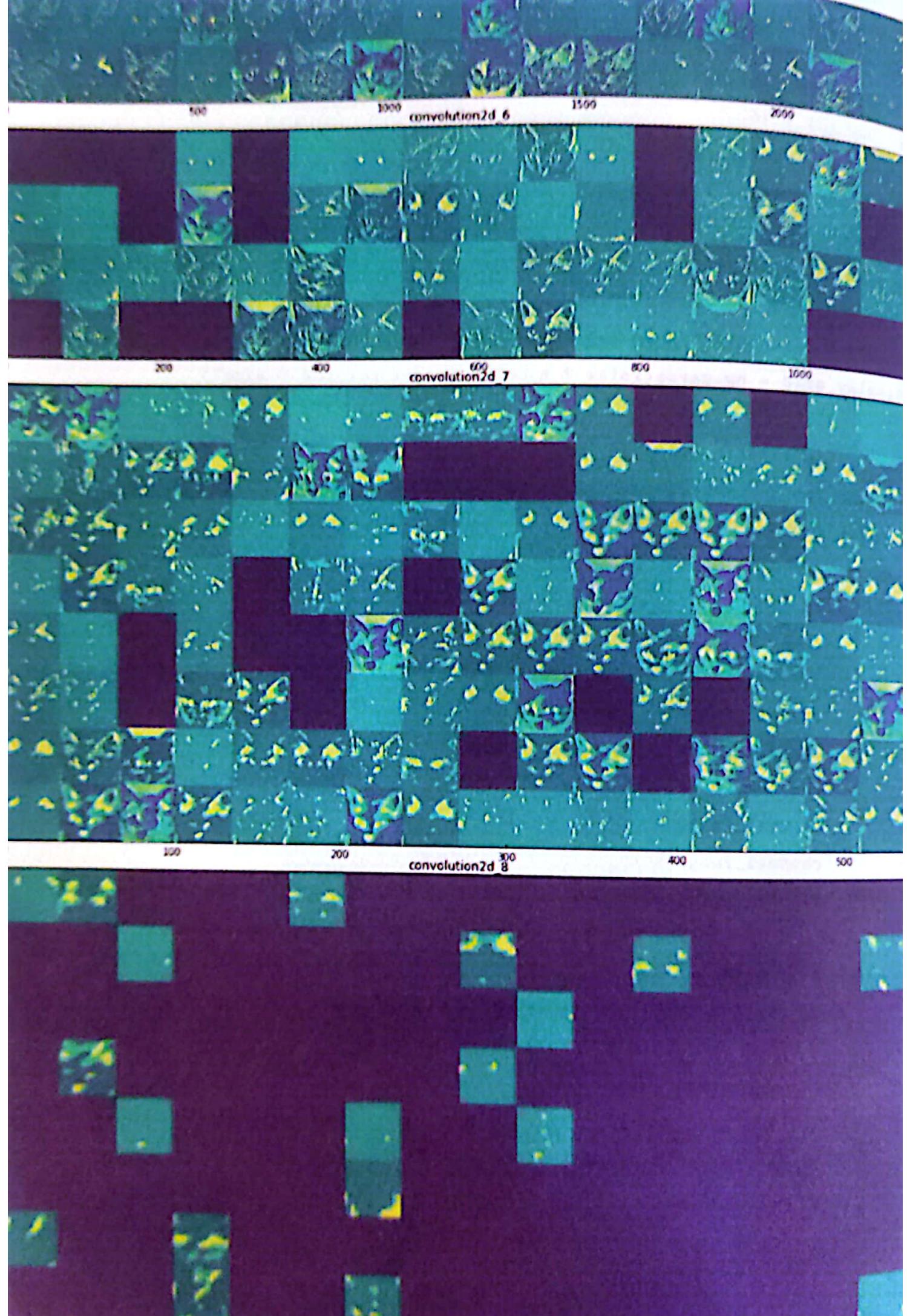
```
layer_names = []
for layer in model.layers[:8]:
    layer_names.append(layer.name) # Katman isimlerini şekilde yazdırma için
images_per_row = 16
# Nitelik haritalarını gösterir
for layer_name, layer_activation in zip(layer_names, activations):
    n_features = layer_activation.shape[-1] # Öznitelik haritasındaki öznitelik sayısı
    size = layer_activation.shape[1] # Katmanlar (1, size, size, n_features) şeklinde

    n_cols = n_features // images_per_row # Aktivasyon kanallarını bu matris üzerine döşer
    display_grid = np.zeros((size * n_cols, images_per_row * size))

    for col in range(n_cols): # Tüm filtreleri yatayda bir grid olarak döşer
        for row in range(images_per_row):
            channel_image = layer_activation[0,
                                              :, :,
                                              col * images_per_row + row]

            # Niteliklerin görsel olarak daha iyi görünmesi için son işlemi yapar
            channel_image -= channel_image.mean()
            channel_image /= channel_image.std()
            channel_image *= 64
            channel_image += 128
            channel_image = np.clip(channel_image, 0, 255).astype('uint8')
            display_grid[col * size : (col + 1) * size, # Gridi gösterir
                         row * size : (row + 1) * size] = channel_image

    scale = 1. / size
    plt.figure(figsize=(scale * display_grid.shape[1],
                        scale * display_grid.shape[0]))
    plt.title(layer_name)
    plt.grid(False)
    plt.imshow(display_grid, aspect='auto', cmap='viridis')
```



Burada not edilmesi gereken birkaç husus var:

- İlk katman çeşitli kenar dedektörleri koleksiyonu gibidir. O aşamada aktivasyonlar neredeyse ilk girdi ile aynı bilgiyi saklarlar.
- Yukarıda doğru gittikçe aktivasyonlar giderek daha soyut hâle gelir ve görsel olarak daha az anlamlanır. "Kedinin kulağı" veya "kedinin gözü" gibi yüksek seviye konseptleri kodlarlar. Yüksek gösterimler resmin içeriği ile ilgili giderek daha az veri taşıırken, resmin ait olduğu sınıf ile ilgili daha çok veri taşırlar.
- Aktivasyonların seyrekliği katmanın derinliği arttıkça artar: İlk katmanda tüm filtreler girdi ile aktif olurken ilerleyen katmanlarda bazı filtreler siyahdır. Bu da filtre tarafından kodlanan örüntünün o resimde olmadığını gösterir.

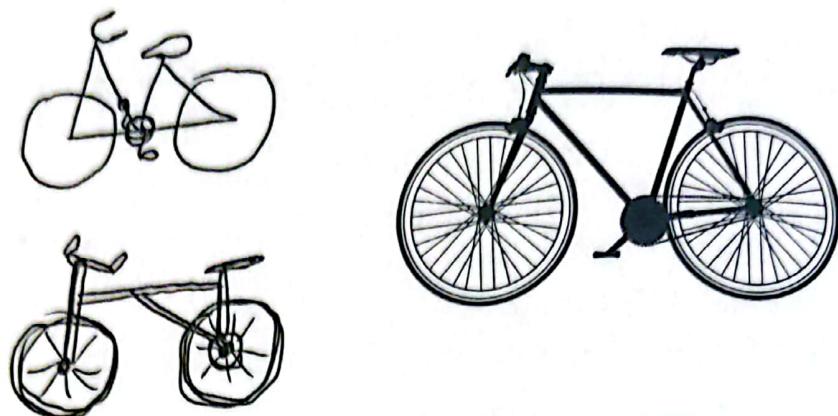
Derin sinir ağlarının öğrendiği gösterimlerin en genel özelliklerini kanıtladık: Katmanların derinleştirikçe öğrendikleri gösterimler soyutlaşır. Yüksek katmanların aktivasyonları verilen girdi ile ilgili giderek daha az veri bulundururken, hedefe ait verileri giderek daha çok bulundururlar (bizim örneğimizde kedi ve köpek sınıfları). Derin sinir ağları *bilgi damıtma hattı* gibidir: Ham veri (örnekte RGB resim) ilerledikçe tekraren dönüşüme uğrar ve gereksiz bilgi atılırken kullanışlı veriye yakınlaşılır ve arıtılır (örneğin resmin ait olduğu sınıf).

Bu benzetme insan ve hayvanların dünyayı algılama yoludur: İnsanlar bir sahneyi gördükten birkaç saniye sonra soyut nesnelerin (bisiklet, ağaç) varlığını hatırlayabilir ama nesnelerin görüşüslərini hatırlayamaz. Hafızanızdan bisiklet çizmeye kalksanız tüm hayatınız boyunca yüzlerce bisiklet görmüş olmanızı rağmen doğru bir şey çizme şansınız azdır (Şekil-5.28). Şu an bunu deneyin: Bu kesinlikle doğru bir etkidir. Beyniniz girdileri tamamen soyut bir şekilde öğrenir -gereksiz ayrıntıları filtreleyerek yüksek seviyeli kavramları öğrenir- çevre unsurlarının nasıl olduğunu hatırlamak zordur.

5.4.2 Evrişim Filtrelerini Görselleştirmek

Evrişimli sinir ağlarının öğrendiği filtreleri incelemenin bir diğer yolu da her filtrenin hangi örüntüye cevap verdiği göstermektir. Bu, *girdi uzayında gradyan çıkışını* kullanarak yapılabilir: Siyah bir resimden başlayarak girdi değerlerine gradyan inişi uygulamak, evrişimli sinir ağında girdinin birfiltreye olan cevabını enbüyütür. Sonuçta elde edilen girdi resmi seçilen filtrenin cevabını en büyük yapacaktır.

Süreç oldukça basittir: İstenen bir evrişim katmanının istenen bir filtresinin değerlerini enbüyütcecek bir kayıp fonksiyonu yazacaksınız. Sonra stokastik gradyan inişi ile girdi resmi öyle ayarlayacaksınız ki bu aktivasyon değeri en



Şekil 5.28: Solda: Kafadan bisiklet resmi çizme denemeleri. Sağda: Bisikletin neye benzediğine dair sematik bir gösterim

büyük olsun. Örneğin ImageNet'te öneğitilmiş VGG16 ağının `block3_conv1` katmanının sıfırıncıfiltresi için bir kayıp fonksiyonu aşağıdadır.

Kod 5.32 : Filtre görselleştirmesi için kayıp fonksiyonu tanımlamak

```
from keras.applications import VGG16
from keras import backend as K

model = VGG16(weights='imagenet',
                include_top=False)

layer_name = 'block3_conv1'
filter_index = 0

layer_output = model.get_layer(layer_name).output
loss = K.mean(layer_output[:, :, :, filter_index])
```

Gradyan inişini gerçeklemek için bu kaybin verilen girdiye göre gradyanına ihtiyacınız olacak. Bunu Keras'ta backend modülündeki `gradient` fonksiyonu ile yapabilirsiniz.

Kod 5.33 : Girdi için kaybin gradyanını almak

```
# gradient metodu geriye tensör listesi döndürür(Bu durumda büyüklüğü 1).
# Dolayısıyla ilk elemanı olan tensör alınır.
grads = K.gradients(loss, model.input)[0]
```

Gradyan inişi sürecinin düzgün ilerlemesi için gözle görünmeyen bir hileye ihtiyacı var: Gradyan tensörünü L_2 normu ile normalize etmek (tensördeki değerlerin karelerinin ortalamasının karekökü). Böylece girdiye yapılacak güncellemelerin şiddetinin aynı aralıkta kalmasını sağlar.

Kod 5.34 : Gradyan normalizasyon hilesi

```
# Sıfıra bölme hatası olmaması için 1e-5 ekle.
grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)
```

Bir girdi için gradyan ve kayıp tensörünü hesaplamalısınız. Bir Keras fonksiyonu tanımlayarak bunu yapabilirsiniz: `iterate` fonksiyonu Numpy tensörünü alır ve iki Numpy tensörlü bir liste döner: Kayıp ve gradyan değeri.

Kod 5.35 : Numpy girdileri için Numpy çıktısı verme

```
iterate = K.function([model.input], [loss, grads])

import numpy as np
loss_value, grads_value = iterate([np.zeros((1, 150, 150, 3))])
```

Bu noktada Python döngüsü ile stokastik gradyan inişi yapabilirsiniz.

Kod 5.36 : Stokastik gradyan inişi ile kaybin enbüütülmesi

```
# Bir miktar gürültü eklenmiş gri bir resimden başla.
input_img_data = np.random.random((1, 150, 150, 3)) * 20 + 128.

step = 1. # Her gradyan güncellemesinin şiddetini
for i in range(40): # Gradyan çıkışını 40 kez çalışır.
    loss_value, grads_value = iterate([input_img_data])

    input_img_data += grads_value * step # Kaykı enbüütünen yöne doğru girdiyi günceller.
```

Sonuçta elde edilen resim tensörü $(1, 150, 150, 3)$ şeklinde ve $[0, 255]$ aralığında tam sayı olmayan değerlerdendir. Yani elde ettiğiniz resmi görüntüleyebilmek için son işleminden geçirmelisiniz. Bunu aşağıdaki basit yardımcı fonksiyon ile yapabilirsiniz.

Kod 5.37 : Tensörü resim hâline getirmek için yardımcı fonksiyon

```
def deprocess_image(x):
    x -= x.mean()           # Tensörü normalize eder.
    x /= (x.std() + 1e-5)   # Merkez 0 ve standart
    x *= 0.1                # sapması 1 olacak şekilde.
    x += 0.5
    x = np.clip(x, 0, 1)     # RGB resme
    x *= 255
    x = np.clip(x, 0, 255).astype('uint8') # dönüştürür.

    return x
```

Artık bütün parçalar hazır. Şimdi hepsini, katman ismini ve filtre indeksini parametre olarak alan ve geriye seçili filtre aktivasyonunu enbüyükten örüntüyyi temsil eden resim tensörünü döndürecek bir Python fonksiyonunda birlestirelim.

Kod 5.38 : Filtre görselleştirme oluşturmak için fonksiyon

```
def generate_pattern(layer_name, filter_index, size=150):
    layer_output = model.get_layer(layer_name).output      # Katmanın n.inci filtresinin
    loss = K.mean(layer_output[:, :, :, filter_index])     # aktivasyonunu enbüyükten kaydırır
                                                        # fonksiyonunu oluşturur.

    grads = K.gradients(loss, model.input)[0] # Girdinin bu kayba göre gradyanını hesaplar.
    grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5) # Normalizasyon hilesi

    iterate = K.function([model.input], [loss, grads]) # Kayıp ve gradyanları döndürür.

    # Bir miktar gürültü eklenmiş gri bir resimden başla.
    input_img_data = np.random.random((1, size, size, 3)) * 20 + 128.

    step = 1.

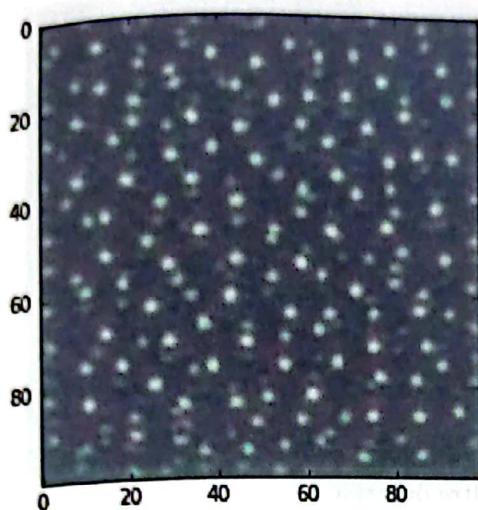
    for i in range(40): # Gradyan çıkışı 40 kez çalışır.
        loss_value, grads_value = iterate([input_img_data])
        input_img_data += grads_value * step

    img = input_img_data[0]
    return deprocess_image(img)
```

Şimdi bu fonksiyonu `block3_conv1` katmanının sıfırıncı kanalında deneyelim (Şekil-5.29'a bakınız).

```
>>> plt.imshow(generate_pattern('block3_conv1', 0))
```

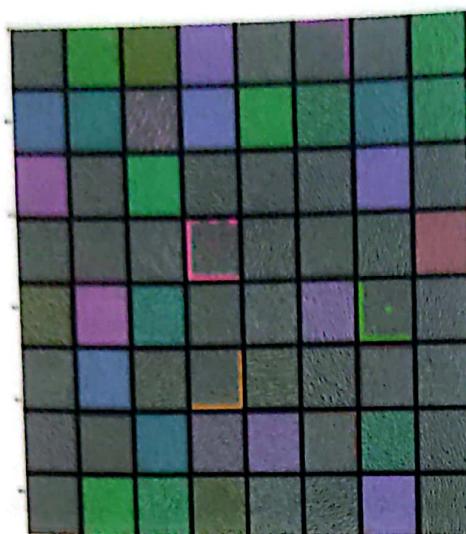
Görüldüğü kadarıyla `block3_conv1` katmanının sıfırıncı kanalı polka-dot örüntüsüne cevap veriyor. Şimdi eğlence zamanı, tüm katmanlardaki tüm filtreleri görselleştirelim. Sadelik için her evrişim bloğunun ilk evrişim katmanının ilk 64 filtresine bakacağız (`block1_conv1`, `block2_conv1`, `block3_conv1`, `block4_conv1`, `block5_conv1`). Çıktıları, 64×64 'lük filtre örüntülerinin 8×8 'i grid üzerinde herfiltre arasında bir miktar siyah boşluk olacak şekilde düzenleyin (Şekil-5.30 - 5.33'e bakınız).



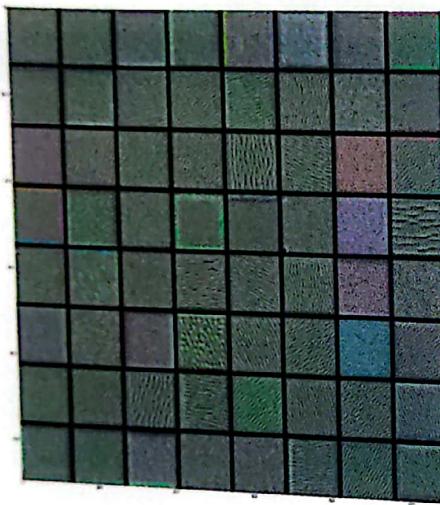
Şekil 5.29: block3_conv1 katmanın sıfırıncı kanalının en büyük cevabı verdiği filtre

Kod 5.39 : Bir katmandaki tüm filtre çıktılarını oluşturma

```
layer_name = 'block1_conv1'  
size = 64  
margin = 5  
  
# Sonuçları kaydetmek için boş bir resim.  
results = np.zeros((8 * size + 7 * margin, 8 * size + 7 * margin, 3))  
  
for i in range(8):          # Sonuç gridinin satırlarında döngü oluşturur.  
    for j in range(8):      # Sonuç gridinin sütunlarında döngü oluşturur.  
  
        # Seçili katmanda i + (j * 8).nci filtre için örüntü oluşturur.  
        filter_img = generate_pattern(layer_name, i + (j * 8), size=size)  
  
        horizontal_start = i * size + i * margin  
        horizontal_end = horizontal_start + size  
        vertical_start = j * size + j * margin  
        vertical_end = vertical_start + size  
        results[horizontal_start: horizontal_end,  
                vertical_start: vertical_end, :] = filter_img  
  
plt.figure(figsize=(20, 20))  
plt.imshow(results)
```



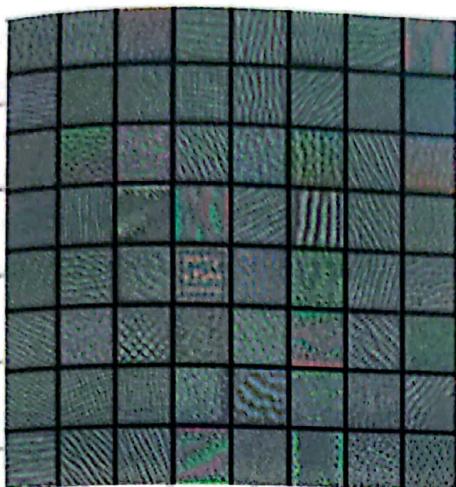
Şekil 5.30: `block1_conv1` katmanın
filtre örüntüleri



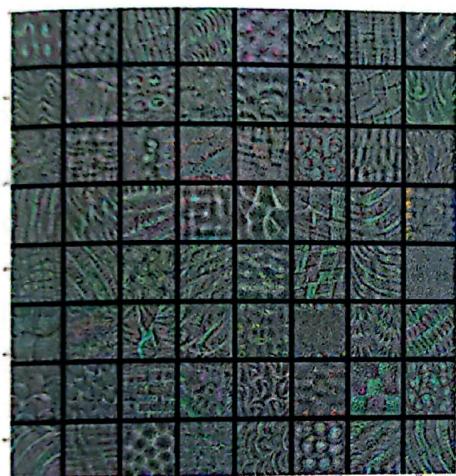
Şekil 5.31: `block2_conv1` katmanın
filtre örüntüleri

Bu filtre görselleri evrişimli sinir ağlarının dünyayı nasıl gördüğünü size söylüyor: Evrişimli sinir ağının her katmanı, girdileri bir filtre kombinasyonu olarak ifade edilebilecek filtre koleksiyonu öğreniyor. Bu, Fourier dönüşümünün sinyali cosinüs fonksiyonlarına bölmesine benzer. Evrişimli sinir ağındaki filtreler giderek karmaşıklasır ve ilerledikçe arıtılır.

- Modelin ilk katmanının (`block1_conv1`) filtreleri basit yön belirten köşeleri ve renkleri (bazı yerlerde renkli köşeler) kodlar.
- (`block1_conv1`) filtreleri köşelerin ve renklerin kombinasyonu olan dokuları kodluyor.
- İllerleyen katmanlardaki filtreler doğal resimlerde bulunan dokulara benzemeye başlıyor: Tüyü, göz, yaprak vb.



Şekil 5.32: block3_conv1 katmanın filtre örüntüleri



Şekil 5.33: block4_conv1 katmanın filtre örüntüleri

5.4.3 Sınıf Aktivasyon Isı Haritalarını Görselleştirmek

Son bir görselleştirme teknigi daha gösterecegiz: Evrisim sinir agının sınıflandırma için kararını verirken resmin hangi bölümünün etken olduğunu anlamak için faydalıdır. Bu, özellikle evrisimli sinir ağlarını öğrenme sürecinde hata ayıklama yapmak -örneğin sınıflandırma hatası- için faydalıdır. Bu genel kategorideki tekniklere *sınıf aktivasyon haritası* (CAM-Class Activation Map) görselleştirme denir ve aktivasyon haritalarının girdi resimleri üzerine uygulanmasından ibarettir. Sınıf aktivasyon haritası 2B grid şeklinde belirli bir çıktı sınıfına ait skorlardır. Girdi resimdeki, her noktada hesaplanır ve o noktanın ilgili sınıfı sınıflandırmaya olan etkisini gösterir. Örneğin, bir girdi resim kedi-köpek evrisimli sinir ağına beslendiğinde CAM size "kedi" sınıfı için girdinin hangi bölümlerinin kediye benzediğini veya "köpek" sınıfı için girdinin hangi bölümlerinin köpeğe benzediğini görme imkani sağlar.

"Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization"²⁰ isimli makalede tanımlanan metodu kullanacaksınız. Bir girdi

²⁰Ramprasaath R. Selvaraju vd., arXiv (2017), <https://arxiv.org/abs/1610.02391>.

için evrişimli katmanın çıktı nitelik haritasını alır. Daha sonra bu nitelik haritasındaki her kanalın girdinin sınıfına göre gradyan değeriyle ağırlıklandırılır. Bu süreci anlamak için şöyle düşünebilirsiniz: "Sınıf tanımlanmasında herhangi bir kanalın ne kadar önemli olduğunu" bularak "girdinin farklı kanalları ne kadar aktif hâle getirdiğini" ağırlıklandırılmış uzamsal haritasından "girdinin çıktı sınıfını ne kadar yoğunlukta aktif hâle getirdiğini" gösteren uzamsal bir harita oluşturuyorsunuz.

Bu teknigi VGG16 ağını kullanarak göstereceğiz.

Kod 5.40 : Öncedilimis VGG16 ağını yüklemek

```
from keras.applications.vgg16 import VGG16
```

```
model = VGG16(weights='imagenet') # Daha önce sınıflandırıcıyı kullanmadığınızda,
# Bu defa kullanacaksınız.
```

Şekil-5.34'teki (Creative Commons lisanslı) iki tane Afrika filini -muhtemelen anne ve yavrusu çayırda geziniyor- düşünün. VGG16 modelinin okuyabileceği bir formata dönüştürelim: Model, keras.applications.vgg16.preprocess_input yardımcı fonksiyonu içinde bulunan bir kaç kurala göre ön işleminden geçirilmiş 224×224 'lük resimler ile eğitilmiştir. Resmi okuyup 224×224 'e yeniden boyutlandırıp float32 Numpy tensörü hâline getirmelisiniz.



Şekil 5.34: Test resmi

Kod 5.41 : VGG16 için resim girdisine ön işlem yapmak

```
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input, decode_predictions
import numpy as np
```

```



```

Şimdi öneğitimli ağı çalıştırıp tahmin vektörünü okunabilir bir formatta yazdırın:

```

>>> preds = model.predict(x)
>>> print('Predicted:', decode_predictions(preds, top=3)[0])
Predicted: [(u'n02504458', u'African_elephant', 0.92546833),
(u'n01871265', u'tusker', 0.070257246),
(u'n02504013', u'Indian_elephant', 0.0042589349)]

```

Bu resim için en yüksek skorlu üç sınıf:

- Afrika fili (%92.54 olasılık değer ile)
- Yaban domuzu (%7.02 olasılık değer ile)
- Hint fili (%0.04 olasılık değer ile)

Ağ kaç tane olduğunu bilmeksızın bir fil olduğunu tanıdı. Tahmin vektöründeki sınıf indeksi 386 olan "Afrika fili" en büyük aktivasyonu alan sınıfıtır.

```

>>> np.argmax(preds[0])
386

```

Resmin hangi parçalarının "Afrika fili"ne benzetildiğini görmek için Grad-CAM süreci oluşturalım.

Kod 5.42 : Grad-CAM algoritmasını oluşturmak

```

# Tahmin vektöründeki "Afrika fili" girdisi
african_elephant_output = model.output[:, 386]

# VGG16'nın son evrişim katmanı block5_conv3'un çıktı nitelik haritası
last_conv_layer = model.get_layer('block5_conv3')

# "Afrika fili" sınıfının block5_conv3'un çıktı nitelik haritasına göre gradyanı
grads = K.gradients(african_elephant_output, last_conv_layer.output)[0]

```

```

# (512,) şeklinde bir vektör. Her elemanı gradyanın nitelik haritasının
# bir kanalına göre ortalama yoğunluğu
pooled_grads = K.mean(grads, axis=(0, 1, 2))

# Tanımladığınız niceliklerin değerlerine erişmenizi sağlar:
# Örnek bir resim verilmişken pooled_grads ve block5_conv3'ün çıktı öznitelik haritası
iterate = K.function([model.input],
                     [pooled_grads, last_conv_layer.output[0]])

# İki filin örnek resimleri verilmişken bu iki niceliğin Numpy dizileri olarak
# pooled_grads_value, conv_layer_output_value = iterate([x])

for i in range(512): # Öznitelik çıktı haritasındaki her bir kanalı "bu kanal ne kadar
    # nemlidir" ile
    conv_layer_output_value[:, :, i] *= pooled_grads_value[i] # "fil" sınıfına göre çapar

# Sonuçtaki öznitelik çıktı haritasının kanal türünden ortalaması sınıf aktivasyonunun 131
# haritasıdır.
heatmap = np.mean(conv_layer_output_value, axis=-1)

```

Görselleştirmek için ısı harmasını da 0 ile 1 arasına normalize edeceksize Sonuçlar için Şekil-5.35'e bakabilirsiniz.

Kod 5.43 : İşi haritası son işlem

```

heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)

```

Son olarak OpenCV kullanarak elde ettiğiniz ısı harmasını resmin üzerine uygulayın (Şekil-5.36).

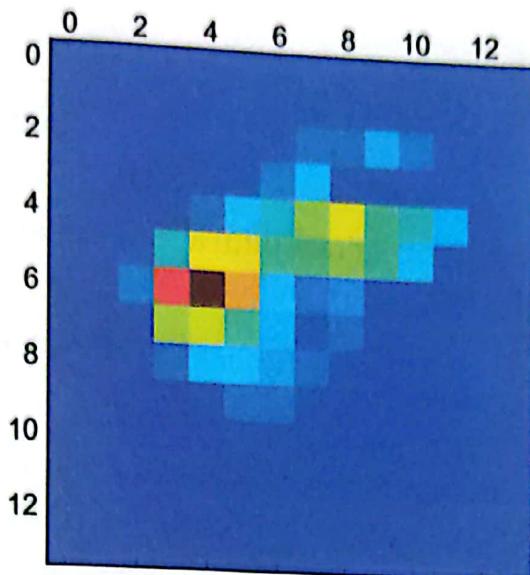
Kod 5.44 : Özgün resme ısı harmasını eklemek

```

import cv2

img = cv2.imread(img_path)
heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
heatmap = np.uint8(255 * heatmap)
heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
superimposed_img = heatmap * 0.4 + img
cv2.imwrite('/Users/fchollet/Downloads/elephant_cam.jpg', superimposed_img)

```



Sekil 5.35: Test resmi için Afrika fili sınıfı ısı haritası



Sekil 5.36: Sınıf aktivasyon ısı haritasının özgün resim üzerine uygulanması

Bu görselleştirme iki önemli soruya cevap veriyor:

- Ağ neden bu resimde bir Afrika fili olduğunu düşünüyor?
- Afrika fili resimde nerede?

Dahası yavru filin kulakları oldukça aktif: Muhtemelen ağ, Hint fili ile Afrika fili arasındaki farkı bu şekilde anlıyor.

Bölüm Özeti:

- Görsel sınıflandırma problemlerinde en iyi araç evrişimli sinir ağlarıdır.
- Evrişimli sinir ağları görsel dünyayı tanımlamak için hiyerarsık örüntüler ve kavramlar öğrenirler.
- Evrişimli sinir ağları kara kutu değildir. Öğrendiği şeyler görselleş tirilebilir.
- Resim sınıflandırma problemini çözmek için kendi evrişimli sinir ağınızı eğitebilecek bilgiye sahipsiniz.
- Aşırı uydurma ile mücadele de veri seti çeşitlendirmenin nasıl kullanıldığını biliyorsunuz.
- Öneğitimli bir evrişimli sinir ağını kullanarak öznitelik çıkarımı ve hassas ayarı nasıl yapacağınızı öğrendiniz.
- Evrişimli sinir ağlarının filtrelerinin öğrendiklerini görselleştirebilrsiniz ve elbette sınıf aktivasyon haritalarını kullanabilirsiniz.