

Kütüphanelerin Projeye Dahil Edilmesi

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout
from keras.layers import Conv2D, MaxPooling2D, Activation
from keras.datasets import cifar100
from keras.utils import to_categorical
from keras import models
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
import numpy as np

import matplotlib.pyplot as plt
import matplotlib.image as img
```

Keras kütüphanesinden gerekli olan model , katmanlar , kullanılacak veri seti , Fotoğraf Ön İşleme , numpy ve sonuçların görselleştirilmesi için matplotlib' i projeye dahil ederiz.

```
(train_images, train_labels), (test_images, test_labels) = cifar100.load_data()
```

Cifar100 veriseti eğitim verileri, etiketleri , test verileri , etiketleri şeklinde belleğe aktarılır.

Klasör Hiyerarşisinin Oluşturulması

```
import os, shutil
```

 işletim sistemine ait fonksiyonların yerine getirebilmek adına projeye dahil edilir.

```
base_dir=os.getcwd()
base_dir=base_dir+"\\Datasets\\"

os.mkdir(base_dir)

train_dir=os.path.join(base_dir, "training")
os.mkdir(train_dir)
os.mkdir(train_dir+"\\baby")
os.mkdir(train_dir+"\\bed")
os.mkdir(train_dir+"\\mouse")
os.mkdir(train_dir+"\\pear")
os.mkdir(train_dir+"\\snail")
os.mkdir(train_dir+"\\wardrobe")

test_dir=os.path.join(base_dir, "test")
os.mkdir(test_dir)
os.mkdir(test_dir+"\\baby")
os.mkdir(test_dir+"\\bed")
os.mkdir(test_dir+"\\mouse")
os.mkdir(test_dir+"\\pear")
os.mkdir(test_dir+"\\snail")
os.mkdir(test_dir+"\\wardrobe")
```

Cifar100 Veri Seti Keras üzerinden bilgisayara dahil edildikten sonra 6 ayrı sınıfa ayırmak gerekmektedir. Bunun için ilk olarak kaynak kodunun çalışma dizinini “os.getcwd()” komutu ile elde ederek “base_dir” olarak saklanır. Bu dizine verileri tutulacak “Datasets” adında bir klasör oluşturulur. Bu işlem “os.mkdir(base_dir)” komutu ile sağlanır.

Eğitim ve test verilerinin saklanacağı dosyalar oluşturulur ,ardından ilgili sınıflara ait tüm dosyalar “os.mkdir” komutu ile oluşturulur.

Verisetlerinin Ayırıştırılması

Her bir sınıfa ait 500 adet eğitim verisi, 100 adet test verisi bulunmaktadır.

Cifar100 veri setinde ise 100 adet sınıfa ait 500*100=50000 adet veri vardır. Bu verileri For Döngüsünde 50000 kez gezerek her bir label ‘a denk gelen sınıfı belirleyerek If then Else ‘ler ile ilintili sınıflara ait fotoğraflar programatik olarak kaydedilir.

```
for i in range(0,50000):
    if train_labels[i][0] == 2 :
        img.imsave(train_dir+"\\baby\\"+str(i)+".png", train_images[i])
        #baby dosyasına fotoyu kaydet train
    elif train_labels[i][0] == 5 :
        img.imsave(train_dir+"\\bed\\"+str(i)+".png", train_images[i])
        #bed dosyasına fotoyu kaydet
    elif train_labels[i][0] == 50 :
        img.imsave(train_dir+"\\mouse\\"+str(i)+".png", train_images[i])
        #mouse dosyasına fotoyu kaydet
    elif train_labels[i][0] == 57 :
        img.imsave(train_dir+"\\pear\\"+str(i)+".png", train_images[i])
        #pear dosyasına fotoyu kaydet
    elif train_labels[i][0] == 77 :
        img.imsave(train_dir+"\\snail\\"+str(i)+".png", train_images[i])
        #snail dosyasına fotoyu kaydet
    elif train_labels[i][0] == 94 :
        img.imsave(train_dir+"\\wardrobe\\"+str(i)+".png", train_images[i])
        #wardrobe dosyasına fotoyu kaydet
```


Her bir If koşulu o an ki indise ait eğitim verisinin etiketinin işaret ettiği indis değeri kontrol edilerek sınıfa ait bloğa girer, ardından img.save metodu ile kayıt edilecek yol belirtilir ve ilgili fotoğraf , fotoğrafları içeren diziden çekilerek kayıt edilir.


Test verilerini de / test yolu altında aynı algoritma ile gerekli sınıfları ayırarak kayıt işlemi tamamlanır.


Ad	Değiştirme tarihi
test	11.04.2020 16:06
training	11.04.2020 16:06


Ad	Değiştirme tarihi
baby	11.04.2020 16:07
bed	11.04.2020 16:07
mouse	11.04.2020 16:07
pear	11.04.2020 16:07
snail	11.04.2020 16:07
wardrobe	11.04.2020 16:07


EnesYapmaz > Datasets > training > baby



202.png



260.png



308.png


382.png


1664.png


1696.png


1705.png


1778.png

Her bir eğitim veri setinde 500 adet veri bulunmaktadır



Herbir sınıfa ait örnek resimler

Normalizasyon

Her bir fotoğrafın her bir pikseli ilk olarak 0-255 aralığında modellenmiş olarak sisteme aktarılır, ancak ağımızı eğitmeden önce bu görüntülere Normalizasyon yaparak her bir piksel değerini 0-1 aralığına indiririz.

Bu işlem için ImageDataGenerator sınıfı kullanılabilir ve uygulaması şu şekildedir;

```
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
```

Not: Normalizasyon yapmak ağıımızın daha kolay karar vermesini sağlar , validation ve accuracy i yükseltir.

Kayıtlı Görüntülerin Belleğe Aktarılması

Görüntüleri “dosyadan okumak için” train_datagen değişkenleri içerisinde flow_from_directory metodu kullanılır ve parametre olarak “Dosya Yolu”, “Herbir fotoğraf için Genişlik , Yükseklik değeri ”, “Batch Size” , “Sonuçlandırılacak Sınıf Modu” değerleri verilir.

```
train_generator=train_datagen.flow_from_directory(#dosyadan okur
    train_dir,
    target_size=(32,32),
    batch_size=20,
    class_mode='categorical'
)
```

Batch Size = Yığın (batch) başına görüntü sayısı 20 olarak belirlenir. 20 tane görüntüyü girişe uygular ve toplam hataya bakarak back prop ile ağımlar eğitilir.

- Herbir iterasyonda kullanılan eğitim örnekleri sayısı.

Class Mode = "Categorical" olması çıkışta 2 den fazla sınıfa ait karar verilecek anlamına gelir ve output 'ta her bir sınıfa' 0 -1 aralığında bir oran atanır ve en yüksek olan değer'e karar kılınır.

Bu parametrenin Binary olması 2 sınıfa ait bir karar verilecek anlamına gelir ve sonuçta karar kılınan sınıfa 1 alternatifine ise 0 değeri atanır.

```
validation_generator = test_datagen.flow_from_directory(#dosyadan okur
    test_dir,
    target_size=(32, 32),
    batch_size=20,
    class_mode='categorical')
```

Ağın eğitimi sırasında daha önce hiç görmediği verilere karşı tepkisini ölçmek adına validation set 'ler kullanılır ve ağın o anki durumunu kontrol eder. Bunu validation accuracy ve validation loss olacak şekilde değerlendirmelerimizde kullanırız.

Burada ise test verileri elde edilir ve validation_generator değişkenine atanır.

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

Dosya yolundan tüm Veriler elde edildikten sonra train_generator içerisinde data_batch ve labels_batch içerisine alınır ve shape metodu ile veri ve etiketlerin adetleri ekrana yazdırılır.


```
Found 3000 images belonging to 6 classes.  
Found 600 images belonging to 6 classes.  
data batch shape: (20, 32, 32, 3)  
labels batch shape: (20, 6)
```

train_datagen.flow_from_directory ve test_datagen.flow_from_directory

metodları çalıştırıldıktan sonra 6 sınıfa ait toplam train ve test 'verilerinin sayılarını çıkarır.

Data batch shape 'de batch size 20 , her bir görüntü için 32x32 boyut ve görüntüleri RGB olduğundan için red , green , blue değerlerine ait 3 adet matris olduğunu ifade eder.

Labels batch shape 'de ise 20 adet görüntüye ait 6 farklı sınıf'ı temsil eder.

1-BaseModel

Ağ yapısının oluşturulması

```
#model  
model = Sequential()  
model.add(Conv2D(32, (3, 3), activation='relu',padding='same', input_shape=(32, 32, 3)))  
model.add(Conv2D(32, (3, 3),padding='same', activation='relu'))  
model.add(MaxPooling2D((2, 2)))  
model.add(Conv2D(64, (3, 3),padding='same', activation='relu'))  
model.add(MaxPooling2D((2, 2)))  
model.add(Conv2D(64, (3, 3),padding='same', activation='relu'))  
model.add(MaxPooling2D((2, 2)))  
  
model.summary()  
  
model.add(Flatten())  
model.add(Dense(256, activation='relu'))  
model.add(Dense(6, activation='softmax'))
```

Sequential, oluşturulan modele katmanların sırayla verilebilmesine olanak tanıyan bir model çeşididir.

Konvolüsyon katmanı bir girdi görüntüsünü alıp, görüntüdeki çeşitli görünüşleri/nesneleri birbirinden ayırabilmek için kullandığımız bir Derin Öğrenme algoritmasıdır.

İçerdiği parametreler de ise ;

-İlk satırda 32 adet convolution katmanının öğreneceği filtre sayısıdır.

-(3,3) ise her bir filtrenin 3x3 olduğunu belirtir.

- Aktivasyon fonksiyonunun Relu olması sinir ağında her bir hücreye uygulanan fonksiyonu belirtir.

-Relu giriş değerini 0 ve 1 arasında korur. Relu yerine sigmoid, tanh gibi aktivasyon fonksiyonları da kullanılabilir ancak Relu'nun diğerlerine göre hesap yükü hafiftir ve başarıımı da daha karmaşık hesap yükü gerektiren fonksiyonlara oranla yakındır. Bu sebeple Relu tercih edildi.

-Padding'in "same" olması demek filtre uygulandıktan sonra fotoğrafa kırpma yapmadan önce boş kalan alanları Default bir değer olan 0 ile doldurur ve kırpma işlemi gerçekleşir, sonuçta ise görüntünün boyutunda eksilme olmaz.

Görüntünün boyutu oldukça küçük olduğu için ağı görüntüdeki öz nitelikleri daha iyi seçebilmesi adına görüntünün küçülmesini istemeyiz.

- input_shape = (32, 32, 3) ise görüntümüz 32x32 'lik 3 ise rgb renkli bir fotoğraf olduğunu temsil eder. Her bir renge ait değerler ayrı bir dizide tutulur.

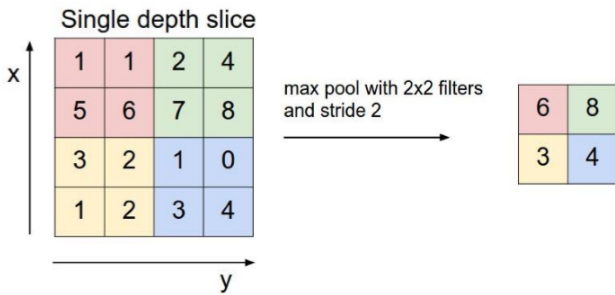
- MaxPooling2D (2,2) ise 2x2' lik bir filtre uygular ve ilgili gruptan En yüksek değer alınarak kırpma işlemi yapılır.

Kullandığım Convolution Katmanı sayısı;

Giriş katmanı ile beraber toplam 4 tanedir.

Toplam 3 adet ise MaxPooling yaptım.

Son olarak 2 adet Dense Layer ekledim ki sonuncusu çıkış katmanıdır.



-Flatten ağıların sonucunda çıkmış olan katmanların Dense Layer'a eklenmesi için tek boyutlu bir vektör haline getirip Dense layer' ın inputlarına verme yarar.

-Dense Layer, düzenli olarak bağlı sinir ağı katmanıdır.

-256 her bir katmandaki hücre sayısını gösterir ve input (giriş) verilerini temsil eder. Ne kadar az hücre sayısı ile doğruluk oranını arttırsak o kadar iyidir.

Çünkü tahmin sırasında her birinin hesaplanması gerekir.

-Aktivasyon fonksiyonu tekrardan Relu olarak belirlenir

-Son olarak 6 farklı çıkışı olduğu için çıkış parametremizi 6 verdik. Activation fonksiyonunun "softmax" olması sonuçları olasılık dağılımı şeklinde gösterir.

Yumuşak bir çıkışı vardır keskin olarak 1 veya 0 demez , her bir çıkışa 1 ve 0 arasında bir oran verir.

-Her bir sınıfın oranlarını topladığımızda toplam 1.0 olur.

Model: "sequential_7"

Layer (type)	Output Shape	Param #
conv2d_17 (Conv2D)	(None, 32, 32, 32)	896
conv2d_18 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_13 (MaxPooling)	(None, 16, 16, 32)	0
conv2d_19 (Conv2D)	(None, 16, 16, 64)	18496
max_pooling2d_14 (MaxPooling)	(None, 8, 8, 64)	0
conv2d_20 (Conv2D)	(None, 8, 8, 64)	36928
max_pooling2d_15 (MaxPooling)	(None, 4, 4, 64)	0
Total params: 65,568		
Trainable params: 65,568		
Non-trainable params: 0		

İlk Conv katmanında Shape (32, 32 ,32) olarak belirtilmiş; bunun nedeni her bir fotoğraf (İlk 2' kısım görüntü boyutu) 32x32 'dir ve (3.sıradaki) 32 adet kernel vardır.

-(3*3 (Maske Boyutu) * 3 (RGB olduğu için 3 ayrı matris) * 32 (Toplam Kernel)) + 32 (Bias) = 896 => Katmandaki eğitilecek parametre sayısı

-(32 (Giriş katmanı) * 32 (2.katman) * 3 * 3 (Maske Boyutları) + 32 (Bias)) = 9248

-Max Pooling ile görüntüler yarı yarıya küçüldü ve padding' de same olduğundan dolayı kırılma olmadı.

-64 (3.katmanda ki filtre sayısı) * 32 (2.katmanın çıkışları) * 3 * 3 (Maske Boyutları) + 64 (Bias) =18496

- Max Pooling ile görüntüler yarı yarıya küçüldü ve padding' de same olduğundan dolayı kırılma olmadı.

$$- 64 * 64 * 3 * 3 + 64 = 36928$$

Total Params = 36928 + 18496 + 9248 + 896 = 65.568

Trainable params = 65.568 çünkü Tüm parametreleri eğitmek istiyoruz

max_pooling2d_15 (MaxPooling (None, 4, 4, 64))		0
flatten_5 (Flatten)	(None, 1024)	0
dense_9 (Dense)	(None, 256)	262400
dense_10 (Dense)	(None, 6)	1542
=====		
Total params: 329,510		
Trainable params: 329,510		
Non-trainable params: 0		

Flatten ile Dense layer a geçiş yapabilmek adına tek boyutlu bir vektör oluşturuldu.

-4*4 (Width x Height) * 64 (Son Conv katmanı kernel sayısı) =1024

-Dense_9 : Dense layer'da 256 adet input var =>

$$(1024 (Flatten) * 256(input) + 256 (bias)) = 262400$$

Flatten : Tek boyutlu dense layer 'a input olacak vektör

-Çıkış Katmanı : (256 (1.Dense layer) * 6 (Çıkış katmanı) +6(bias)) = 1542

Modelin Derlemesi

```
#modeli derleme
#olasılıklı bir çıkış beklendiği zaman ctg_crossentropy kullanıyoruz
from keras import optimizers
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(lr=1e-3), metrics=['acc'])
```

Modelin derlenmesi için oluşturduğumuz model'den compile fonksiyonu çağırılır. Modeli eğitim için ayarlar.

Loss fonksiyonu tasarlanan modelin hata oranını aynı zamanda başarımını ölçen bir fonksiyondur.

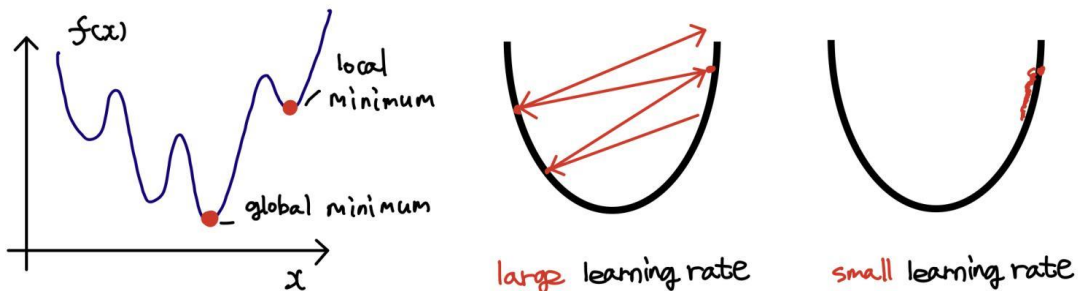
Loss fonksiyonu olarak “categorical_crossentropy” verilir. Nedeni ise, olasılıklı bir çıkış beklediğimiz içindir. İkili bir çıkış olsaydı **binary_crossentropy** ifadesine yer verirdik.

Metric olarak accuracy değerini belirttim.

RMSprop optimizer momentumlu Gradient Descent algoritmasına benzer.

Optimizer için RMSprop verdim ve learning rate 'i 1e-4.

- Learning Rate (öğrenme oranı hızı) bir Hyperparameter'dir ve bunu daima kendimiz ayarlarız.
- Learning rate değerimiz çok büyükse o zaman local'de bir minimum noktasını bulup global minimumu atlayıp kaçırabiliriz.
- Learning rate eğer çok küçük olursa çok yavaş ilerleriz ve çok zaman kaybederiz. Local ve Global minimum noktaları sistemin eğiminin ($m = \text{cost}/w$) sıfır olduğu noktalardır. Cost ise ağıdaki maliyettir.



Modelin Eğitilmesi

Ağın eğitiminin başladığı kısım

```
history=model.fit_generator(train_generator,  
                             steps_per_epoch=150,  
                             epochs=100,  
                             validation_data=validation_generator,  
                             validation_steps=30  
                             )  
model.save('cifar100-ImageAug')
```

Modelin eğitilmesi için verdiğim parametreler;

Bir Epoch, tüm veri kümesinin sinir ağı üzerinden sadece bir kez ileri ve geriye aktarılmasıdır ve ağırlıkların güncellenmesi için birimdir.

-Train_generator = flow_from_directory' den dönen değişken'dir içerisinde train batch'leri bulunmaktadır.

-Steps_per_epoch Ağırlıkların bir epoch içinde kaç kez güncelleneceğini belirtir.

Steps_per_epoch = 150 = Eğitim verisi Sayısı / batch_size

500 (Herbir sınıf için örnek sayısı) * 6 (Sınıf adedi) / 20 =150

-Validation_data ise eğitim sırasında test için kullanılacak test verilerini belirtir.

Validation_steps = TotalvalidationSamples / ValidationBatchSize

Kaç adımda bir doğrulama yapılacağını belirtir.

```

Epoch 1/100
150/150 [=====] - 15s 102ms/step - loss: 1.7455 - acc: 0.2443 - val_loss: 1.6753 - val_acc: 0.3500
Epoch 2/100
150/150 [=====] - 14s 95ms/step - loss: 1.5978 - acc: 0.3693 - val_loss: 1.6281 - val_acc: 0.3367
Epoch 3/100
150/150 [=====] - 14s 96ms/step - loss: 1.4897 - acc: 0.4167 - val_loss: 1.2869 - val_acc: 0.3800
Epoch 4/100
150/150 [=====] - 14s 95ms/step - loss: 1.3998 - acc: 0.4623 - val_loss: 1.2717 - val_acc: 0.4850
Epoch 5/100
150/150 [=====] - 14s 94ms/step - loss: 1.3144 - acc: 0.5080 - val_loss: 1.4975 - val_acc: 0.4967
Epoch 6/100
150/150 [=====] - 14s 95ms/step - loss: 1.2486 - acc: 0.5377 - val_loss: 1.5493 - val_acc: 0.4817
Epoch 7/100
150/150 [=====] - 14s 97ms/step - loss: 1.1924 - acc: 0.5490 - val_loss: 1.5955 - val_acc: 0.4900
Epoch 8/100
150/150 [=====] - 14s 96ms/step - loss: 1.1532 - acc: 0.5707 - val_loss: 1.3582 - val_acc: 0.5100 ETA: 6s - loss: 1.1626 - acc: 0.5608
Epoch 9/100
150/150 [=====] - 14s 95ms/step - loss: 1.1098 - acc: 0.5993 - val_loss: 1.0705 - val_acc: 0.5700
Epoch 10/100
150/150 [=====] - 14s 96ms/step - loss: 1.0723 - acc: 0.6007 - val_loss: 1.2164 - val_acc: 0.5600
Epoch 11/100
150/150 [=====] - 15s 100ms/step - loss: 1.0317 - acc: 0.6140 - val_loss: 1.6103 - val_acc: 0.5367
Epoch 12/100
150/150 [=====] - 16s 105ms/step - loss: 0.9998 - acc: 0.6297 - val_loss: 1.3413 - val_acc: 0.5950
Epoch 13/100
150/150 [=====] - 15s 100ms/step - loss: 0.9773 - acc: 0.6413 - val_loss: 1.0584 - val_acc: 0.6000
Epoch 14/100
150/150 [=====] - 15s 99ms/step - loss: 0.9397 - acc: 0.6530 - val_loss: 0.7819 - val_acc: 0.6333
Epoch 15/100
150/150 [=====] - 14s 96ms/step - loss: 0.9138 - acc: 0.6650 - val_loss: 0.4700 - val_acc: 0.6400
Epoch 16/100
150/150 [=====] - 15s 99ms/step - loss: 0.8775 - acc: 0.6783 - val_loss: 0.8895 - val_acc: 0.6433
Epoch 17/100
150/150 [=====] - 15s 103ms/step - loss: 0.8618 - acc: 0.6840 - val_loss: 0.9503 - val_acc: 0.6250
Epoch 18/100
150/150 [=====] - 14s 96ms/step - loss: 0.8297 - acc: 0.6963 - val_loss: 0.9235 - val_acc: 0.6567
Epoch 19/100
150/150 [=====] - 14s 96ms/step - loss: 0.8143 - acc: 0.7013 - val_loss: 1.0103 - val_acc: 0.6717
Epoch 20/100
150/150 [=====] - 14s 94ms/step - loss: 0.7848 - acc: 0.7117 - val_loss: 0.7676 - val_acc: 0.6717
Epoch 48/100
150/150 [=====] - 14s 93ms/step - loss: 0.2836 - acc: 0.9060 - val_loss: 0.9146 - val_acc: 0.7233
Epoch 49/100
150/150 [=====] - 14s 94ms/step - loss: 0.2707 - acc: 0.9137 - val_loss: 1.9408 - val_acc: 0.7100 ETA: 4s - loss: 0.2598 - acc: 0.9184114/150
[=====>.....] - ETA: 3s - loss: 0.2712 - acc: 0.9145
Epoch 50/100
150/150 [=====] - 14s 93ms/step - loss: 0.2604 - acc: 0.9180 - val_loss: 1.5282 - val_acc: 0.7467
Epoch 51/100
150/150 [=====] - 14s 93ms/step - loss: 0.2437 - acc: 0.9260 - val_loss: 0.4466 - val_acc: 0.7283
Epoch 52/100
150/150 [=====] - 14s 94ms/step - loss: 0.2397 - acc: 0.9247 - val_loss: 1.1602 - val_acc: 0.7267
Epoch 53/100
150/150 [=====] - 14s 95ms/step - loss: 0.2207 - acc: 0.9320 - val_loss: 0.9585 - val_acc: 0.7300
Epoch 54/100
150/150 [=====] - 14s 93ms/step - loss: 0.2090 - acc: 0.9393 - val_loss: 1.6846 - val_acc: 0.7150
Epoch 55/100
150/150 [=====] - 14s 93ms/step - loss: 0.1992 - acc: 0.9437 - val_loss: 0.5370 - val_acc: 0.7233
Epoch 56/100
150/150 [=====] - 14s 94ms/step - loss: 0.1854 - acc: 0.9477 - val_loss: 0.7310 - val_acc: 0.7100
Epoch 57/100
150/150 [=====] - 15s 98ms/step - loss: 0.1767 - acc: 0.9503 - val_loss: 1.8207 - val_acc: 0.6950
Epoch 58/100
150/150 [=====] - 14s 96ms/step - loss: 0.1714 - acc: 0.9460 - val_loss: 1.5401 - val_acc: 0.7300
Epoch 59/100
150/150 [=====] - 14s 96ms/step - loss: 0.1588 - acc: 0.9580 - val_loss: 0.8373 - val_acc: 0.7367
Epoch 60/100
150/150 [=====] - 14s 95ms/step - loss: 0.1470 - acc: 0.9633 - val_loss: 0.6621 - val_acc: 0.7183
Epoch 61/100
150/150 [=====] - 14s 95ms/step - loss: 0.1371 - acc: 0.9603 - val_loss: 1.6746 - val_acc: 0.7083
Epoch 62/100
150/150 [=====] - 14s 94ms/step - loss: 0.1270 - acc: 0.9650 - val_loss: 1.6408 - val_acc: 0.6717
Epoch 63/100
150/150 [=====] - 14s 94ms/step - loss: 0.1181 - acc: 0.9660 - val_loss: 0.9583 - val_acc: 0.6983
Epoch 64/100
150/150 [=====] - 14s 94ms/step - loss: 0.1149 - acc: 0.9683 - val_loss: 1.0131 - val_acc: 0.7117
Epoch 65/100
150/150 [=====] - 14s 94ms/step - loss: 0.0992 - acc: 0.9767 - val_loss: 1.7072 - val_acc: 0.7167
Epoch 66/100
150/150 [=====] - 14s 94ms/step - loss: 0.1032 - acc: 0.9703 - val_loss: 2.1982 - val_acc: 0.7300

```

```

Epoch 82/100
150/150 [=====] - 14s 94ms/step - loss: 0.0355 - acc: 0.9920 - val_loss: 1.9805 - val_acc: 0.7050
Epoch 83/100
150/150 [=====] - 14s 95ms/step - loss: 0.0316 - acc: 0.9930 - val_loss: 1.7076 - val_acc: 0.7233
Epoch 84/100
150/150 [=====] - 14s 93ms/step - loss: 0.0264 - acc: 0.9940 - val_loss: 1.5426 - val_acc: 0.7333
Epoch 85/100
150/150 [=====] - 14s 93ms/step - loss: 0.0241 - acc: 0.9957 - val_loss: 1.3415 - val_acc: 0.7233
Epoch 86/100
150/150 [=====] - 14s 93ms/step - loss: 0.0301 - acc: 0.9927 - val_loss: 2.1649 - val_acc: 0.7200
Epoch 87/100
150/150 [=====] - 14s 93ms/step - loss: 0.0182 - acc: 0.9957 - val_loss: 1.1617 - val_acc: 0.7100
Epoch 88/100
150/150 [=====] - 14s 93ms/step - loss: 0.0209 - acc: 0.9947 - val_loss: 2.4045 - val_acc: 0.7250
Epoch 89/100
150/150 [=====] - 14s 93ms/step - loss: 0.0224 - acc: 0.9940 - val_loss: 1.6627 - val_acc: 0.7267
Epoch 90/100
150/150 [=====] - 14s 93ms/step - loss: 0.0226 - acc: 0.9943 - val_loss: 1.7119 - val_acc: 0.7100
Epoch 91/100
150/150 [=====] - 14s 93ms/step - loss: 0.0227 - acc: 0.9933 - val_loss: 1.7073 - val_acc: 0.6800
Epoch 92/100
150/150 [=====] - 14s 94ms/step - loss: 0.0286 - acc: 0.9953 - val_loss: 1.2850 - val_acc: 0.6917
Epoch 93/100
150/150 [=====] - 14s 93ms/step - loss: 0.0141 - acc: 0.9967 - val_loss: 0.7455 - val_acc: 0.7283
Epoch 94/100
150/150 [=====] - 14s 93ms/step - loss: 0.0221 - acc: 0.9940 - val_loss: 1.6698 - val_acc: 0.7267
Epoch 95/100
150/150 [=====] - 14s 95ms/step - loss: 0.0184 - acc: 0.9947 - val_loss: 5.3771 - val_acc: 0.7100
Epoch 96/100
150/150 [=====] - 14s 93ms/step - loss: 0.0137 - acc: 0.9963 - val_loss: 1.5119 - val_acc: 0.7233
Epoch 97/100
150/150 [=====] - 14s 93ms/step - loss: 0.0238 - acc: 0.9940 - val_loss: 1.5850 - val_acc: 0.7183
Epoch 98/100
150/150 [=====] - 14s 93ms/step - loss: 0.0084 - acc: 0.9987 - val_loss: 1.4397 - val_acc: 0.7183
Epoch 99/100
150/150 [=====] - 14s 93ms/step - loss: 0.0167 - acc: 0.9960 - val_loss: 2.2173 - val_acc: 0.7233
Epoch 100/100
150/150 [=====] - 14s 93ms/step - loss: 0.0100 - acc: 0.9973 - val_loss: 1.8078 - val_acc: 0.7400

```

Eğitim aşamasını adım adım seyrettiğimizde acc değerinin en son 0.9973 e vardığını görüyoruz, bu şekilde bir sonucun olmasını isteriz ancak val_acc 0.7400 ile ezberleme olduğunu açık bir şekilde gösteriyor.

```
model.save('cifar100-BaseLarn')
```

Ağımızın eğitim sırasındaki adımlarını grafiksel olarak yorumlamak adına history değişkenine atanır ve model daha sonra kullanılmak için kayıt edilir.

```

acc=history.history['acc']
val_acc=history.history['val_acc']
loss=history.history['loss']
val_loss=history.history['val_loss']

```

- Eğitimin accuracy ve validation değerleri alınır.
- Kayıp fonksiyonunun loss ve validation değerleri alınır.

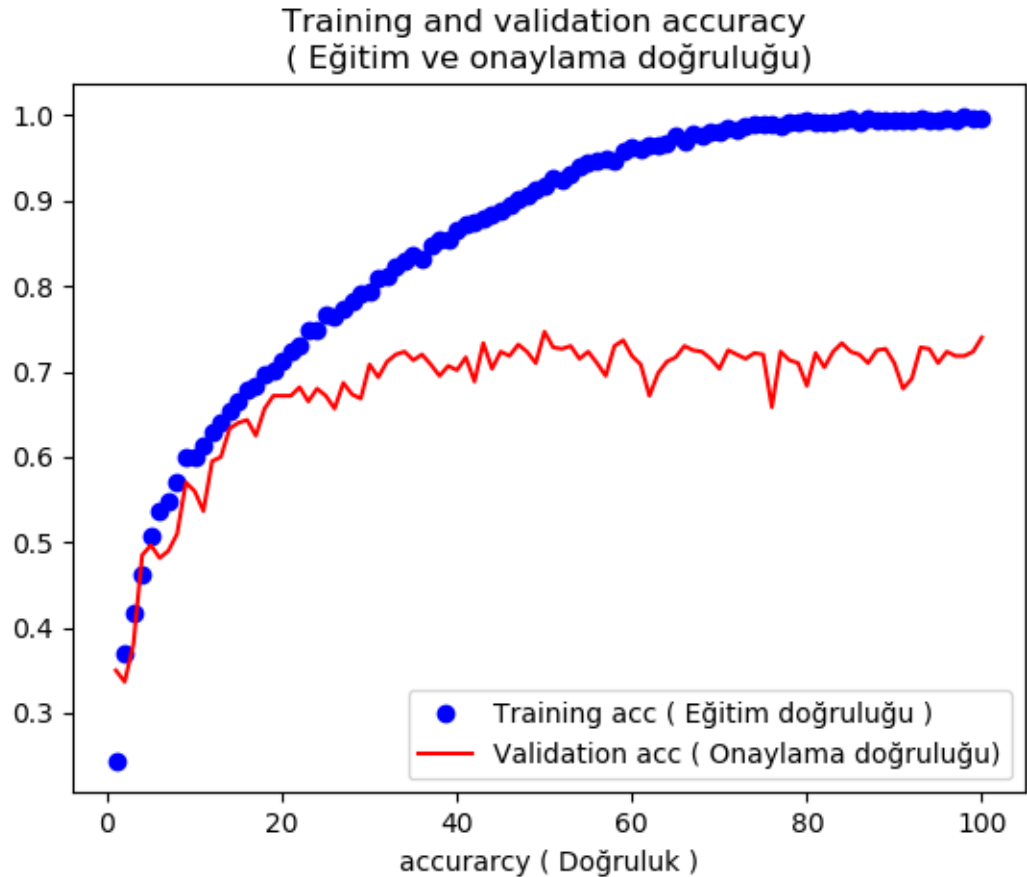

```
plt.plot(epochs,acc,'bo',label='Training acc ( Eđitim dođruluđu )')
plt.plot(epochs,val_acc,'r',label='Validation acc ( Onaylama dođruluđu )')
plt.title('Training and validation accuracy \n( Eđitim ve onaylama dođruluđu )')
plt.xlabel('epochs ( Epoklar )')
plt.xlabel('accuracy ( Dođruluk )')
plt.legend()

plt.figure()

plt.plot(epochs,loss,'bo',label='Training loss ( Eđitim Kayıpları )')
plt.plot(epochs,val_loss,'r',label='Validation loss ( Dođrulama Kayıpları )')
plt.title('Training and validation loss \n( Eđitim ve dođrulama kayıpları )')
plt.xlabel('epochs ( Epoklar )')
plt.xlabel('loss ( Kayıplar )')
plt.legend()

plt.figure()
```

Mathplotlib kütüphanesi kullanılarak validation ve acc değeri görselleştirilir.



Sonuçlar ;

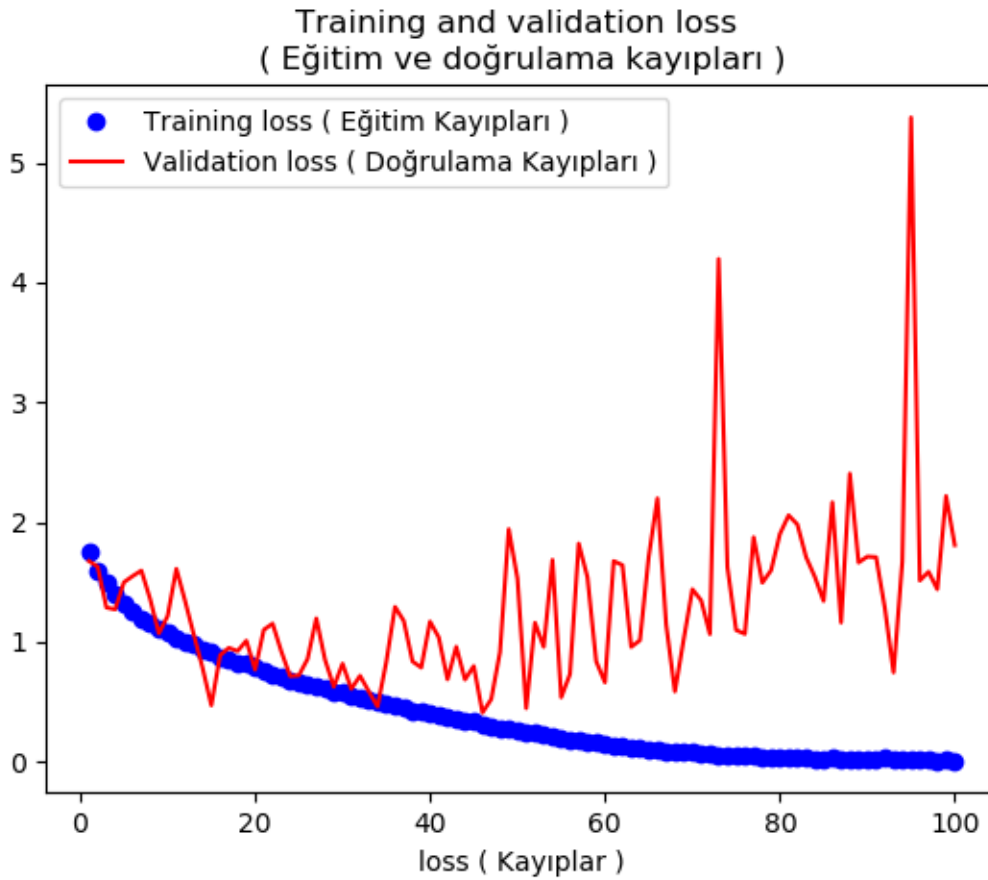
Burada Accurarcy sonuçlarını incelersek %70 'den sonra ađımızın ezberleme yaptığını görebiliriz.

Nedeni ise Acc %100 ulaşırken , Test verileriyle validate etmek istediđimizde dođruluk oranının %70 'lerde kaldığını görebiliriz.

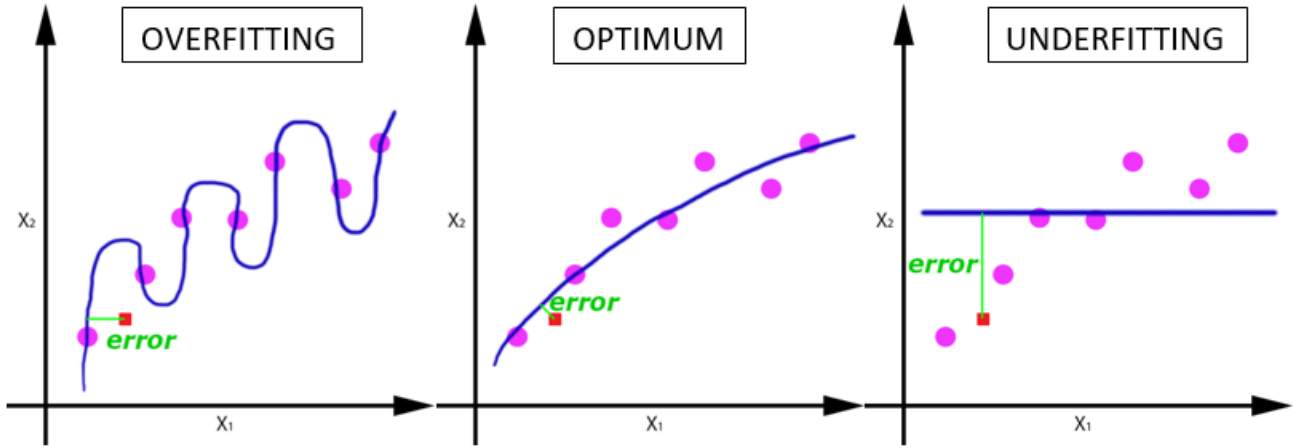
Training Accuracy Eğitim için kullandığı verilere dayanarak test edildiğinde doğru cevabı verebilmiştir ancak, harici bir kaynaktan yani Test verilerinden gelen örneklerin ağı tahminine tabi tuttuğumuzda sonucun pek 'de doğru olmadığı kanısına varabiliyoruz.

Validation Accuracy bize daha keskin ve doğru bir bakış açısı sağlar.

Bu ezberlenin azaltılması adına birkaç yöntem vardır bizim uyguladığımız yöntemler ezberlemenin en fazla olduğu Convolution katmanlarının aralarına Dropout eklemek ve bir başka yöntem olan ImageAugmentation yaparak Train datalarımızı çeşitli rotasyonlara tabi tutarak görüntülerimizi arttırmaktır.



Eğitim sırasında beklediğimiz kayıpların 0 a yaklaşmasıdır çünkü Loss 'un küçük olması ağı girdi ve çıktısı modellemesindeki başarımın ne kadar iyi olduğunu gösterir, Loss fonksiyonlarının temel hedefi ise beklenen ve üretilen çıktının eşitliğinin kontrol edilmesidir. Görüldüğü üzere Training Loss 0' a yaklaşıyor, fakat Validation Loss aynı ölçüde değil. Nedeni ezberlemedir, sonuç olarak Dropout ve ImageAugmentation ile bu makası daraltabiliriz.



Beklenen Ağımızın Optimum düzeyde olmasıdır. Overfitting olduğunda ezberleme gerçekleşmiş olur ki yalnızca verilen dataset’lerde tahmin yapabilir durumdadır.

“DROPOUT EKLEYELİM”

Underfitting oluyorsa dropout değerlerini fazla seçmişizdir.

Ezberlemeyi azaltmanın yollarından bir tanesi de parametre sayısını azaltmaktır.

2-Dropout

Ağdaki genelleştirmeyi arttırmak ve ezberlemeyi azaltmak adına Dropout fonksiyonunu ekleriz.

```
#model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu',padding='same', input_shape=(32, 32, 3)))
model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25)) | ←
model.add(Conv2D(64, (3, 3),padding='same', activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25)) ←
model.add(Conv2D(64, (3, 3),padding='same', activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25)) ←
```

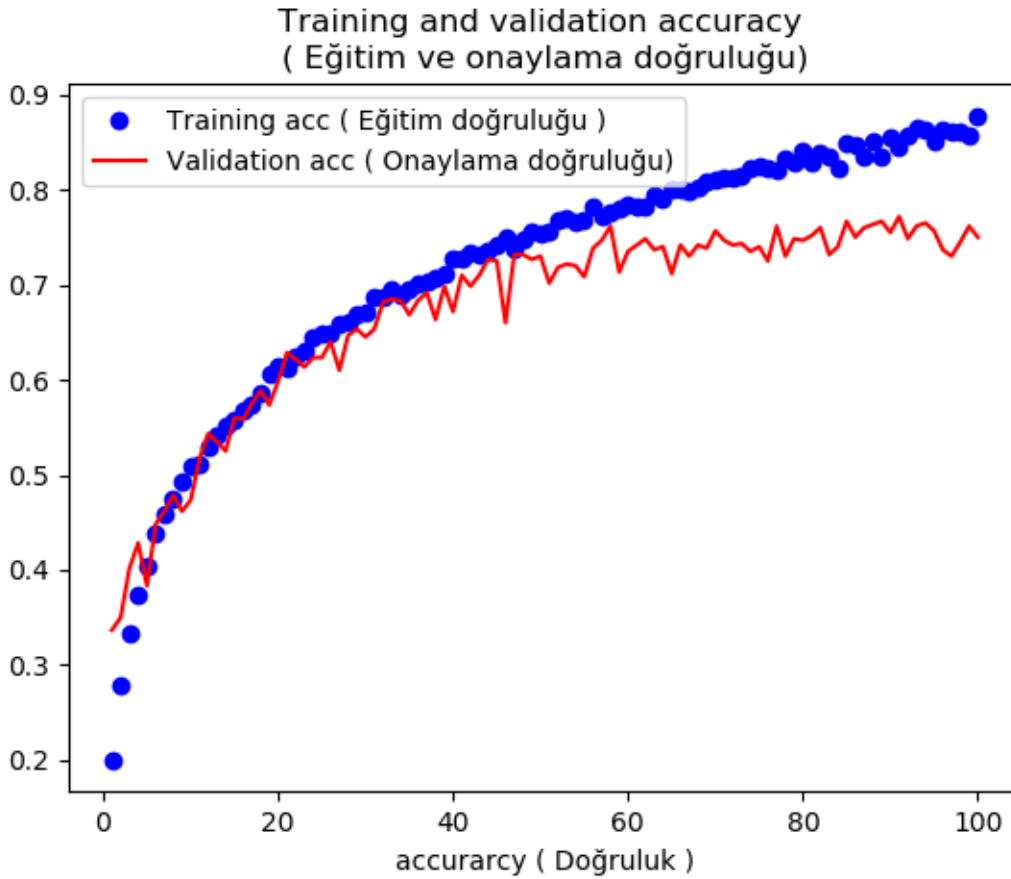
Ok ile belirtilen katmanların arasına Dropout eklenmiştir. %25 oranında iki katman arasındaki Dğümleri koparacaktır. Bu ise ezberlemenin yaşandığı noktalara denk gelecek validation acc ‘ değeri artacak, sonuçta ağımızın başarıımı artacaktır.

```

Epoch 88/100
150/150 [=====] - 15s 103ms/step - loss: 0.4183 - acc: 0.8507 - val_loss: 0.5646 - val_acc: 0.7633
Epoch 89/100
150/150 [=====] - 15s 102ms/step - loss: 0.4312 - acc: 0.8340 - val_loss: 0.4616 - val_acc: 0.7667
Epoch 90/100
150/150 [=====] - 16s 103ms/step - loss: 0.4131 - acc: 0.8557 - val_loss: 0.5352 - val_acc: 0.7550
Epoch 91/100
150/150 [=====] - 15s 102ms/step - loss: 0.4032 - acc: 0.8450 - val_loss: 0.5779 - val_acc: 0.7717
Epoch 92/100
150/150 [=====] - 15s 100ms/step - loss: 0.3845 - acc: 0.8560 - val_loss: 0.3891 - val_acc: 0.7483
Epoch 93/100
150/150 [=====] - 15s 101ms/step - loss: 0.3927 - acc: 0.8650 - val_loss: 0.4329 - val_acc: 0.7617
Epoch 94/100
150/150 [=====] - 15s 101ms/step - loss: 0.3816 - acc: 0.8633 - val_loss: 0.4906 - val_acc: 0.7650
Epoch 95/100
150/150 [=====] - 15s 101ms/step - loss: 0.3897 - acc: 0.8500 - val_loss: 0.7112 - val_acc: 0.7567
Epoch 96/100
150/150 [=====] - 15s 100ms/step - loss: 0.3874 - acc: 0.8630 - val_loss: 0.4273 - val_acc: 0.7367
Epoch 97/100
150/150 [=====] - 15s 101ms/step - loss: 0.3804 - acc: 0.8603 - val_loss: 2.1299 - val_acc: 0.7300
Epoch 98/100
150/150 [=====] - 15s 100ms/step - loss: 0.3760 - acc: 0.8600 - val_loss: 0.5613 - val_acc: 0.7450
Epoch 99/100
150/150 [=====] - 15s 101ms/step - loss: 0.3714 - acc: 0.8570 - val_loss: 1.1034 - val_acc: 0.7617
Epoch 100/100
150/150 [=====] - 15s 103ms/step - loss: 0.3572 - acc: 0.8770 - val_loss: 0.5170 - val_acc: 0.7500

```

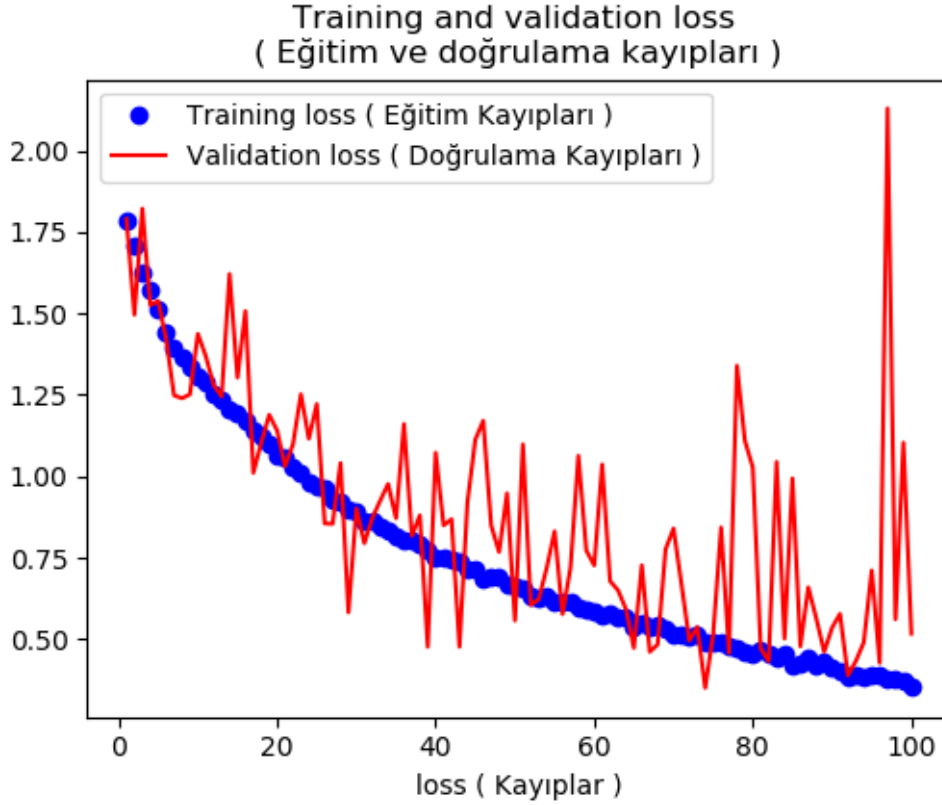
Ağımız eğitime başlandıktan sonra, sonraki 88 ve 100. Epoch'ların sonuçlarını burada gözlemlemekteyiz. Görüldüğü üzere Accuracy ve val_accuracy arasındaki fark oldukça azalmıştır. Bununla birlikte loss ve val_loss değerleri arasındaki makas gitgide kısalmıştır ve 0 a yakınsamaya devam etmiştir. Bu bize ezberlemenin önüne geçtiğimizi ifade eder.



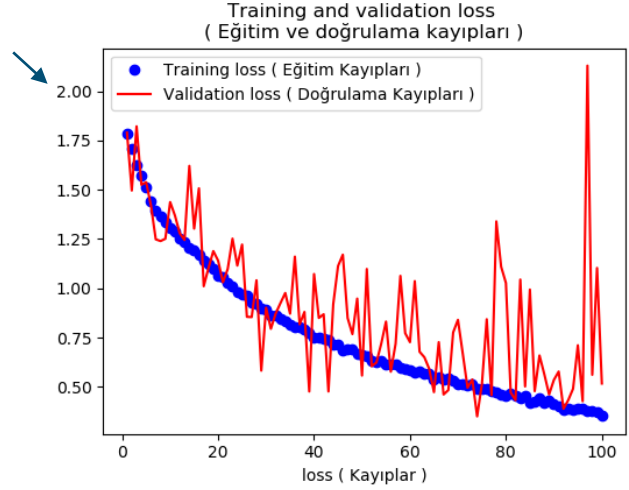
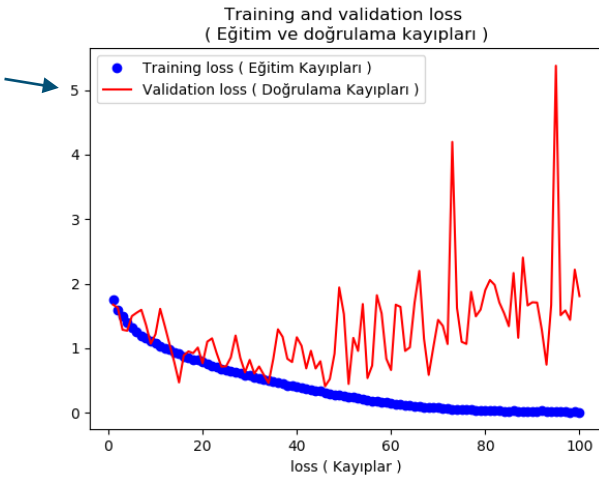
Ezberlemenin yaşanabileceği katmanların aralarına Dropout eklendikten sonra görüldüğü üzere acc ve validation acc arasındaki makas gitgide daralmaya başlıyor 100

epoch sonunda ortalama olarak ağızımız %80 civarında bir doğruluk değere sahip diyebiliriz bu görsele göre.

Aynı zamanda Dropout yaparak bağlantılarını kopardığı hücrelerin, ağıdaki ezberlemenin olduğu alanlara denk geldi tezini öne sürebiliriz.



Dropout ekledikten sonra farkediyoruz ki Training ve validation kayıpları da önemli ölçüde azalmış. Karşılaştırmak gerekirse Dropout eklemeyen önce maksimum kayıp 90.epoch civarlarında 5 değerine kadar gitmişken Dropout sonrası maximum 2 değerine ulaşmış, aynı zamanda benzer genlik değerlerinde seyretilmiştir.



3-Veri Zenginleştirme

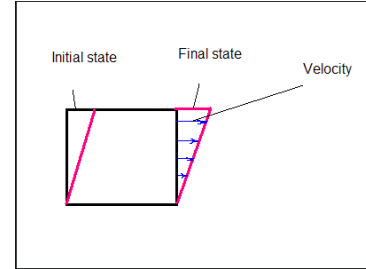
```
#data augmentation*****
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(

    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,
    vertical_flip=True,
    fill_mode='nearest'
)
```

Veri zenginleştirme yapmak verilerime ölçeklendirme,rotasyon değişikliği,yakınlaştırma,Döndürme vb gibi özellikler ekleyerek.Verilerimizi arttırmaya yarar. Bu işlem daha çok eğitilecek veri kümesinin yeterince fazla olmadığı durumlarda kullanılır.

ImageDataGenerator sınıfını import ettikten sonra her bir veriye uygulanacak özellikler ;

- 0-1 aralığına düşürme yani Normalizasyon
- Rotatın_range =(0-180) derece aralığında rotasyon yaptırır
- width_shift and height_shift = Random olarak görüntüyü dikey ve yatay hale getirir



- shear_range = Görüntüye makaslama yapar örn :
- zoom_range = Yakınlaştırır.
- horizontal_flip = Yatay döndürme
- vertical_flip = Dikey döndürme
- fill_mode = Yapılan işlemlerden sonra kaybolan piksellerde nasıl müdahale yapılacağını belirtir ki nearest en yakın pikselle doldur anlamındadır.

```
history=model.fit_generator(train_generator,
                             steps_per_epoch=600,
                             epochs=60,
                             validation_data=validation_generator,
                             validation_steps=30
                             )
model.save('cifar100-ImageAug')
```

Steps_per_epoch burada “600” olması gerekiyor ki nedeni train dataların sayısı image augmentation yapıldığı için 4 katına çıkıyor.

Epoch ‘lar ise benzer bir oranla aşağı çekildi.

```
#model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu',padding='same', input_shape=(32, 32, 3)))
model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D((2, 2)))
#model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3),padding='same', activation='relu'))
model.add(MaxPooling2D((2, 2)))
#model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3),padding='same', activation='relu'))
model.add(MaxPooling2D((2, 2)))
#model.add(Dropout(0.25))

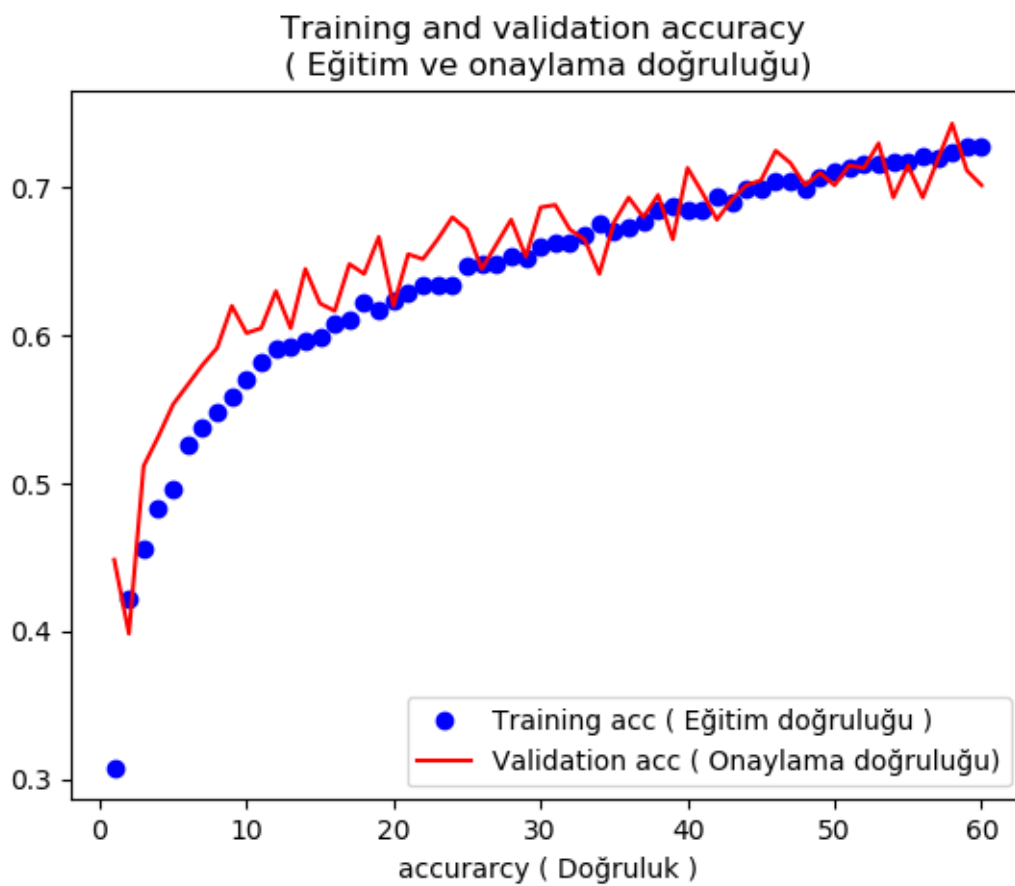
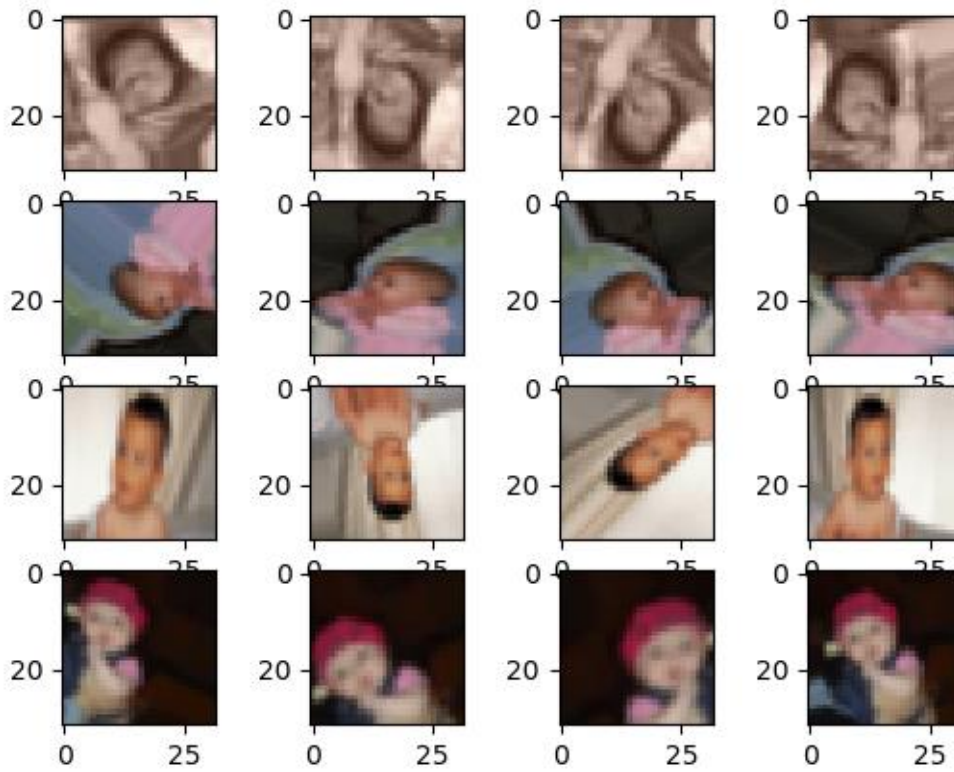
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(6, activation='softmax'))
```

```
Epoch 42/60
600/600 [=====] - 57s 96ms/step - loss: 0.8416 - acc: 0.6942 - val_loss: 0.4714 - val_acc: 0.6783
Epoch 43/60
600/600 [=====] - 57s 95ms/step - loss: 0.8335 - acc: 0.6902 - val_loss: 0.8561 - val_acc: 0.6917
Epoch 44/60
600/600 [=====] - 57s 95ms/step - loss: 0.8277 - acc: 0.6992 - val_loss: 1.1262 - val_acc: 0.7017
Epoch 45/60
600/600 [=====] - 57s 95ms/step - loss: 0.8236 - acc: 0.6998 - val_loss: 0.8549 - val_acc: 0.7050
Epoch 46/60
600/600 [=====] - 167s 279ms/step - loss: 0.8124 - acc: 0.7047 - val_loss: 1.0346 - val_acc: 0.7256
Epoch 47/60
600/600 [=====] - 63s 105ms/step - loss: 0.8112 - acc: 0.7048 - val_loss: 0.6735 - val_acc: 0.7167
Epoch 48/60
600/600 [=====] - 59s 99ms/step - loss: 0.8027 - acc: 0.6996 - val_loss: 0.7976 - val_acc: 0.7017
Epoch 49/60
600/600 [=====] - 58s 97ms/step - loss: 0.8042 - acc: 0.7073 - val_loss: 0.6899 - val_acc: 0.7100
Epoch 50/60
600/600 [=====] - 58s 97ms/step - loss: 0.7969 - acc: 0.7110 - val_loss: 0.4342 - val_acc: 0.7017
Epoch 51/60
600/600 [=====] - 58s 97ms/step - loss: 0.7842 - acc: 0.7141 - val_loss: 0.6989 - val_acc: 0.7150
Epoch 52/60
600/600 [=====] - 58s 96ms/step - loss: 0.7796 - acc: 0.7161 - val_loss: 0.8540 - val_acc: 0.7133
Epoch 53/60
600/600 [=====] - 57s 96ms/step - loss: 0.7729 - acc: 0.7157 - val_loss: 1.0226 - val_acc: 0.7300
Epoch 54/60
600/600 [=====] - 57s 95ms/step - loss: 0.7659 - acc: 0.7170 - val_loss: 0.6153 - val_acc: 0.6933
Epoch 55/60
600/600 [=====] - 57s 95ms/step - loss: 0.7693 - acc: 0.7178 - val_loss: 0.7143 - val_acc: 0.7150
Epoch 56/60
600/600 [=====] - 57s 95ms/step - loss: 0.7611 - acc: 0.7218 - val_loss: 1.0858 - val_acc: 0.6933
Epoch 57/60
600/600 [=====] - 58s 96ms/step - loss: 0.7552 - acc: 0.7204 - val_loss: 0.7276 - val_acc: 0.7183
Epoch 58/60
600/600 [=====] - 73s 122ms/step - loss: 0.7546 - acc: 0.7233 - val_loss: 0.5288 - val_acc: 0.7433
Epoch 59/60
600/600 [=====] - 77s 128ms/step - loss: 0.7471 - acc: 0.7284 - val_loss: 0.6865 - val_acc: 0.7117
Epoch 60/60
600/600 [=====] - 69s 114ms/step - loss: 0.7468 - acc: 0.7283 - val_loss: 1.1236 - val_acc: 0.7017
```

Bir önceki model için eklenmiş olan Dropout 'lar kaldırıldı ve ağı tekrar eğitildi. Bu aşamada ağın accuracy değeri %72 civarlarında kaldı ancak Epoch 'sayısının artmasıyla ağın başarımı dahada yükselecektir. Bununla birlikte validation_accuracy 'arasındaki farkta olabildiğince minimum düzeye inmiş bulunmakta.

Bu ağı eğitimi CPU gücü ile 1 saat sürdü.

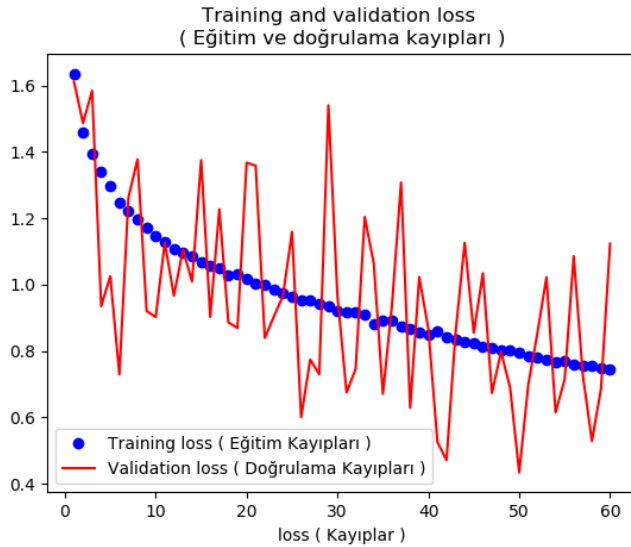
Görüntü eğitilme esnasında örnek olarak veri setlerimi şu şekilde varyasyonlara uğrattıyor;



Veri zenginleştirme yapıldıktan sonraki eğitimin doğruluğu %72 civarına yükseldi, validation acc 'de kendisini takip etmekte. Ezberleme olmadığını görmekteyiz.

Bu demek oluyor ki ağımızın herbir eğitimde kullanmış olduğu verileri tahmin etme becerisiyle kaydırılma,yakınlaştırılma , shift etme gibi.. yukarıda açıklamış olduğumuz Image Augmentation teknikleriyle beraber, validation için kullandığımız temel test verilerine vermiş olduğu yanıt,tepki,sınıflandırma becerisi benzer sonuçlar üretmektedir ve ağımızın ezberlemeyi bıraktığı yönündeki gözlemlerimizi yapabilmekteyiz.

Dropout ile beraber yapılırsa sonuçlar daha tatmin edici bir rakama yükselebileceğini öngörmekteyim.

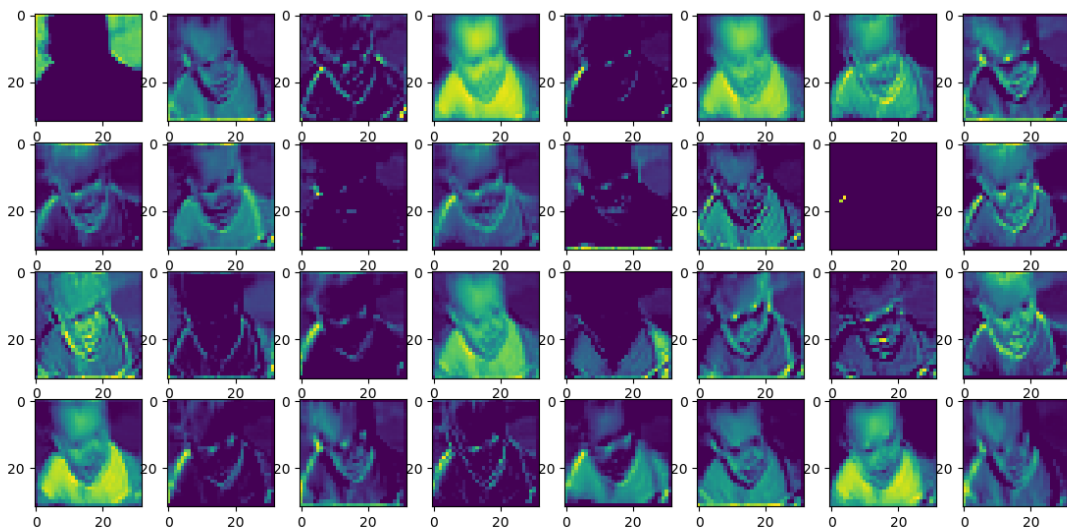


Bu görüntüde ise kayıp fonksiyonunu incelemekteyiz. Görüldüğü üzere 60 epoch ta kayıp fonksiyonumuzun başarımı validation ile denk gitmese de peak yaptığı değer 1.6 ile sınırlandırılmış. Burada ağımızın herbir epoch'da ki herbir veriye verdiği cevap ile olması gereken sınıfın farkına göre bir kayıp sonucu ortaya çıkarıyoruz ki ,validation loss training loss ile beraber seyretmemiş olsada sonuçları minimuma yaklaşma eğilimindedir.

Convulation Layer'lar da görüntü tahmin edilirken neler oluyor ?



Orijinal Test Fotoğrafi



İlk Conv katmanınının 0'dan 32 .kernel a gelene kadar görüntüde yaptığı işlemler yukarıdaki gibidir. Görüldüğü üzere her bir kernel 'da resmin farklı bir kısmını kontrol ediyor ve sonucunda bir sonraki katmana geçerek benzer şekilde kendi katmanına özgü daha spesifik özel kısımlarını ayırt etmeye çalışıyor.

Görüntü tahmin edildiğinde aldığımız yanıt ;

```
In [28]: y=model.predict(img_tensor)
...: y.max()
...: y.argmax()
Out[28]: 0

In [29]: y.max()
Out[29]: 0.9854409

In [30]:
```

%98.5 oranla görüntünün

Bebek olduğunu ifade ediyor