Eğitimde kullandığım sınıfların görüntüleri aşağıdaki gibidir;

Şimdi sıralı olarak kodlarımızı açıklayarak anlatalım.

```
import keras
import os, shutil
from keras.preprocessing.image import ImageDataGenerator
from keras import layers
from keras import models
from keras import optimizers
from keras.layers.core import Dropout
from keras.utils import to_categorical
import matplotlib.pyplot as plt

keras.__version__
```

Ödev için yukarıdaki kutuphaneleri inport ediyoruz keras açık kaynaklı bir sinir ağı kütüphanesidir ve arka planda tensorflow'u kullanır projemız bir görüntü sınıflandırma olduğu için bunu kullanıcagız daha sonra Shell util kütüphanesini ekledik bunun ile dosya işlemleri yapacağız imagedata genareator ile görsellerimiz üzerinde oynamalar yapacağız optimizasyon fonksiyonu agı eğitmek içinmodelde kullanıcaz dropout u ise ezberı engellemek

```
original_dataset_train_cattle_dir = 'TRAIN/cattle'

original_dataset_test_cattle_dir = 'TEST/cattle'

original_dataset_train_elephant_dir = 'TRAIN/elephant'

original_dataset_test_elephant_dir = 'TEST/elephant'

original_dataset_train_fox_dir = 'TRAIN/fox'

original_dataset_test_fox_dir = 'TEST/fox'

original_dataset_train_leopard_dir = 'TRAIN/leopard'

original_dataset_test_leopard_dir = 'TEST/leopard'

original_dataset_train_shark_dir = 'TRAIN/shark'

original_dataset_test_shark_dir = 'TEST/shark'

original_dataset_train_table_dir = 'TRAIN/table'

original_dataset_test_table_dir = 'TEST/table'
```

için kullanacağız görsel kategorizasyonu olacak bu yüzden categorıcal clasentropi kullanılcak ve plt ile grafiklerimizi çizeceğiz.

Yukarıdaki kod bloğunda bizim işlemleri yapacağımız görsellerin yollarını değişkenlerde tutuyoruz programımız cifar100 klasörünün içinde çalıştığı için path ler kısa bu şekilde her kategori için test ve train olarka ayrı ayrı tuttuk ileride kopyalama işleminde kullanacağız.

```
base_dir = '/Users/ismailkaya/Downloads/Compressed/Benimsinifim'
os.mkdir(base_dir)

# validation and test splits,bolumlerimiz

train_dir = os.path.join(base_dir, 'train')
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)
```

Yukarıdaki kod parçasında kendi istediğimiz bir yerde oluşacak dosyanın adresını base de tuttuk ve train,validasion,test klasörlerini oluşturduk.

```
# Directory with our training cattle pictures
train_cattle_dir = os.path.join(train_dir, 'cattle')
os.mkdir(train_cattle_dir)
# Directory with our training elephant pictures
train_elephant_dir = os.path.join(train_dir, 'elephant')
os.mkdir(train_elephant_dir)
# Directory with our training fox pictures
train_fox_dir = os.path.join(train_dir,
'fox') os.mkdir(train_fox_dir)
# Directory with our training leopard pictures
train_leopard_dir = os.path.join(train_dir, 'leopard')
os.mkdir(train_leopard_dir)
# Directory with our training shark pictures
train_shark_dir = os.path.join(train_dir, 'shark')
os.mkdir(train_shark_dir)
# Directory with our training table pictures
train_table_dir = os.path.join(train_dir,
'table') os.mkdir(train_table_dir)
#-----------------------------------------------------------
# Directory with our validation cattle pictures
validation_cattle_dir = os.path.join(validation_dir, 'cattle')
os.mkdir(validation_cattle_dir)

# Directory with our validation elephant pictures
validation_elephant_dir = os.path.join(validation_dir, 'elephant')
os.mkdir(validation_elephant_dir)
# Directory with our validation fox pictures
validation_fox_dir = os.path.join(validation_dir,
'fox') os.mkdir(validation_fox_dir)
```

```
# Directory with our validation leopard pictures
validation_leopard_dir = os.path.join(validation_dir, 'leopard')
os.mkdir(validation_leopard_dir)
# Directory with our validation shark pictures
validation_shark_dir = os.path.join(validation_dir, 'shark')
os.mkdir(validation_shark_dir)

# Directory with our validation table pictures
validation_table_dir = os.path.join(validation_dir, 'table')
os.mkdir(validation_table_dir)
#-------------------------------------------------------------
# Directory with our test cattle pictures
test_cattle_dir = os.path.join(test_dir, 'cattle')
os.mkdir(test_cattle_dir)

# Directory with our test elephant pictures
test_elephant_dir = os.path.join(test_dir, 'elephant')
os.mkdir(test_elephant_dir)
# Directory with our test fox pictures
test_fox_dir = os.path.join(test_dir,
'fox') os.mkdir(test_fox_dir)

# Directory with our test leopard pictures
test_leopard_dir = os.path.join(test_dir, 'leopard')
os.mkdir(test_leopard_dir)
# Directory with our test shark pictures
test_shark_dir = os.path.join(test_dir, 'shark')
os.mkdir(test_shark_dir)
```

```
# Directory with our test table pictures
test_table_dir = os.path.join(test_dir, 'table')
os.mkdir(test_table_dir)
```

Yukarıdaki kodda oluşturduğumuz validation, train, test klasörlerimizin içerisine 6 şart tane kendi kategorilerimizin klasörünü oluşturduk.

```
# Directory with our training cattle pictures
train_cattle_dir = os.path.join(train_dir,
'cattle') os.mkdir(train_cattle_dir)
# Directory with our training elephant pictures
train_elephant_dir = os.path.join(train_dir, 'elephant')
os.mkdir(train_elephant_dir)
# Directory with our training fox pictures
train_fox_dir = os.path.join(train_dir, 'fox')
os.mkdir(train_fox_dir)
# Directory with our training leopard pictures
train_leopard_dir = os.path.join(train_dir, 'leopard')
os.mkdir(train_leopard_dir)
# Directory with our training shark pictures
train_shark_dir = os.path.join(train_dir,
'shark') os.mkdir(train_shark_dir)
# Directory with our training table pictures
train_table_dir = os.path.join(train_dir, 'table')
os.mkdir(train_table_dir)
#-----------------------------------------------------------
# Directory with our validation cattle pictures
validation_cattle_dir = os.path.join(validation_dir, 'cattle')
os.mkdir(validation_cattle_dir)

# Directory with our validation elephant pictures
validation_elephant_dir = os.path.join(validation_dir, 'elephant')
os.mkdir(validation_elephant_dir)
```

```
# Directory with our validation fox pictures
validation_fox_dir = os.path.join(validation_dir,
'fox') os.mkdir(validation_fox_dir)

# Directory with our validation leopard pictures
validation_leopard_dir = os.path.join(validation_dir, 'leopard')
os.mkdir(validation_leopard_dir)
# Directory with our validation shark pictures
validation_shark_dir = os.path.join(validation_dir, 'shark')
os.mkdir(validation_shark_dir)

# Directory with our validation tabble pictures
validation_table_dir = os.path.join(validation_dir,
'table') os.mkdir(validation_table_dir)
#-------------------------------------------------------------
# Directory with our test cattle pictures
test_cattle_dir = os.path.join(test_dir, 'cattle')
os.mkdir(test_cattle_dir)

# Directory with our test fox pictures
test_elephant_dir = os.path.join(test_dir, 'elephant')
os.mkdir(test_elephant_dir)
# Directory with our test fox pictures
test_fox_dir = os.path.join(test_dir,
'fox') os.mkdir(test_fox_dir)

# Directory with our test leopard pictures
test_leopard_dir = os.path.join(test_dir, 'leopard')
os.mkdir(test_leopard_dir)
# Directory with our test shark pictures
test_shark_dir = os.path.join(test_dir, 'shark')
os.mkdir(test_shark_dir)
```

```
# Directory with our test table pictures

test_table_dir = os.path.join(test_dir, 'table')

os.mkdir(test_table_dir)
```

Yukarıdaki kodda ise oluşturduğumuz klasörlere kaynak klasörümüzden aldığımız görsellerimizi yerleştiriyoruz sırasıyla önce train e 6 tane kategorimizi kopyalıyoruz sonra validation a sonra teste validasyon ve teste aynı görselleri kopyaladık 500 tane train e 100 tane aynı görselı valıdasyon ve teste kopyalayarak kendi görsellerimizden oluşan klasörümüzü oluşturduk.

1.Normal Yapıda Eğittiğimiz Ağın Modeli 1 :

```
model = models.Sequential()
model.add(layers.Conv2D(16,(3, 3), activation='relu',
                        padding='same',
                        input_shape=(32, 32, 3)))
 model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        padding='same'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(64, (3, 3), activation='relu',
                        padding='same'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128, (3, 3), activation='relu',
                        padding='same'))
model.add(layers.MaxPooling2D())
model.add(layers.Flatten())  # dizi haline getirmek icin
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(6, activation='softmax'))
model.summary()
model.compile(loss='categorical_crossentropy',
              optimizer=keras.optimizers.Adam(),
              metrics=['acc'])
```

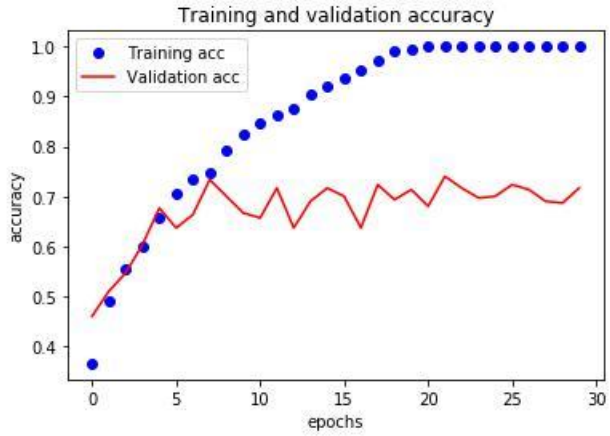
Yukarıdaki kodda ağ modelimizi oluşturuyoruz 4 konvolüsyon katmanı 2 dense leyerdan oluşuyor konvalüsyon katmanlarımı filtreleri kademeli olarak artacak şekilde yaptım ve bu şekilde ağın başarısını artırmak istedim 3*3 lük filtre kullandım çünkü nekadar kucuk olursa okadar başarılı olacağını biliyorum ve 5*5 lik denemelerde yaptım ve 3*3 daha başarılı olduğunu gördüm aktivasyon fonksiyolu olarak relu

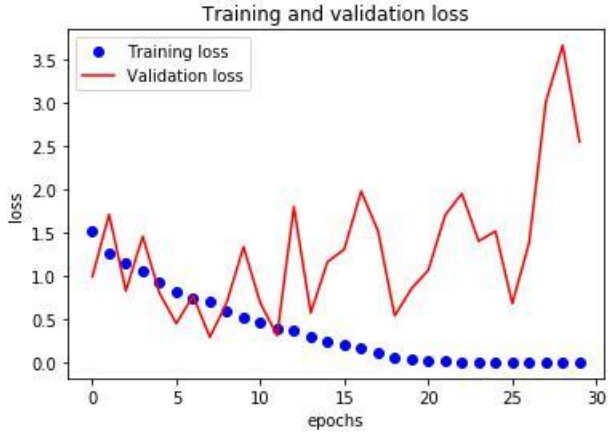
kullandık padding oluşturduk kenarlara geldiğinde çekirdeğimizin taşmalarını öndedim. Göresellerimizin boyutu 32 pixel olduğu için input shape de 32 *32 belirttik ve rekli olduğu için 3 yazıyoruz.

Maxpoolingleri sınırlı kullandım cunku görsel sadece 32 pixel ve çok küçültmememiz gerekiyor. Daha sonra dizi haline getirip birde dense layer kullanıyoruz ve ondada relu ve softmax kullandık aktivasyon fonksiyanu kullandık. 6 olarak kategorilerimizi belirttik loss fonksiyonnu olarak categorical crossentropy kullandık cunku 6 kategorimiz var ve çok kategorılıde bınary kullanılamaz optimazer olarak adam kullandım RMS kullandığımda nan değeri alıyodum loss da buyuzden adam tercih ettim.

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=25,
     epochs=30,
     validation_data=validation_generator,
      validation_steps=30)
#grafiklerimizi cizdirelim
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```

Yukarıdaki kodda ise history kullanarak grafiklerimizi çizdiricez. model fiti kullanarak epoch sayımızı epoch içerisinde ki adımımızı ve validation epoch umuzu belirleyerek çalıştırıyoruz.bu değerleri birçok deneme sonnucunda belirledim step fazla olursa ezbere yaklaşıyor az oluşa başarı düşüyordu buyuzden en iyisi bu adım ise yeterli idi bu ağda %60 üzerini almak için validsayondana en iyisi yine 50 idi arttığı zaman başarı düşüyordu. Daha sonra grafiklerimizi çizdirdik epoch accuricy ve loss değerlerimizi göreceğiz ve grafiğimiz aşağıdaki gibi geldi.





Ağ ezbere gitti ancak %60 başarının üzerine çıktık bunu drop ve zenginleştirme ile düzenleyeceğiz diğer adımlarda şuanlık çıktımı bu. Ağ ezberlediği için acc ve validation acc birbirinden uzak.

2. Dropout kullanarak ağı ezberber yapmasını engellediğimiz kod aşağıda;

```
model = models.Sequential()
model.add(layers.Conv2D(16,(3, 3), activation='relu',
                padding='same',
                input_shape=(32, 32, 3)))
model.add(Dropout(0.1))
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                padding='same'))
model.add(layers.MaxPooling2D(2,2))
model.add(Dropout(0.1))
model.add(layers.Conv2D(64, (3, 3), activation='relu',
                padding='same'))
model.add(layers.MaxPooling2D(2,2))
model.add(Dropout(0.2))
model.add(layers.Conv2D(128, (3, 3), activation='relu',
                padding='same'))
model.add(layers.MaxPooling2D())
model.add(Dropout(0.6))
model.add(layers.Flatten()) # dizi haline getirmek icin
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(6, activation='softmax'))
model.summary()


model.compile(loss='categorical_crossentropy',
        optimizer=keras.optimizers.Adam(),
        metrics=['acc'])
```

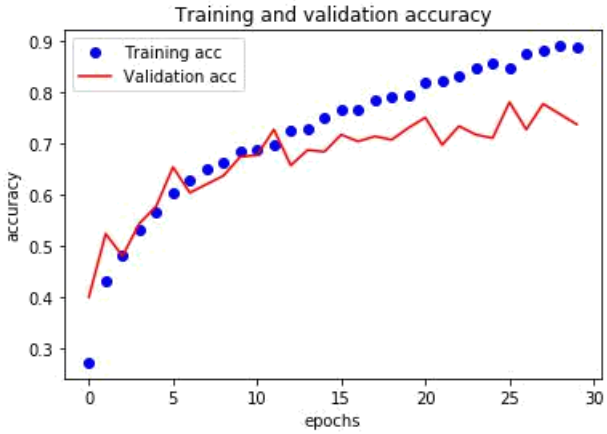
Yukarıdaki kodu yorumlayacak olursam sadece ağı eğittiğimiz kısım olan yeri değiştirerek dropout eklediğimiz kısmı çalıştırabiliyoruz burada dropout uydurmayı ezberlemeyi yani overfittingi engeller bunu bağlantıları kopartarak yapar bende denyerek 0.1 0.2 ve 0.6 olarak kullandım bunu belirlememdeki sebep ise parametrenin cok olduğu yere buyuk bag koparmaları koyarak hem ezberi acc yı kontrol altına aldım hemde başarımı arttırdım bunun grafıgını ve parametre tablomu aşağıya ekliyorum.

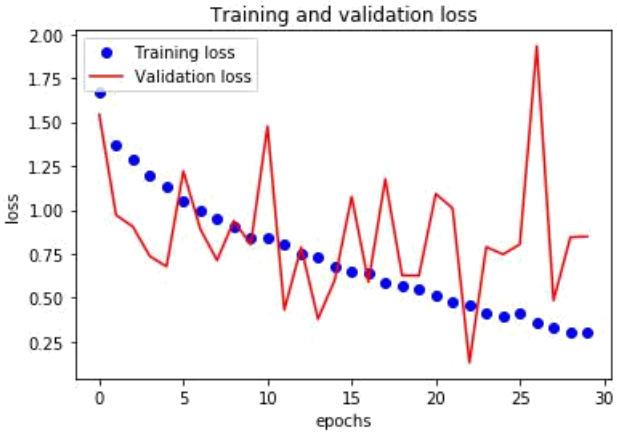
Layer (type)	Output Shape	Param #
conv2d_296 (Conv2D)	(None, 32, 32, 16)	448
dropout_50 (Dropout)	(None, 32, 32, 16)	0
conv2d_297 (Conv2D)	(None, 32, 32, 32)	4640
max_pooling2d_219 (MaxPoolin	(None, 16, 16, 32)	0
dropout_51 (Dropout)	(None, 16, 16, 32)	0
conv2d_298 (Conv2D)	(None, 16, 16, 64)	18496
max_pooling2d_220 (MaxPoolin	(None, 8, 8, 64)	0
dropout_52 (Dropout)	(None, 8, 8, 64)	0
conv2d_299 (Conv2D)	(None, 8, 8, 128)	73856
max_pooling2d_221 (MaxPoolin	(None, 4, 4, 128)	0
dropout_53 (Dropout)	(None, 4, 4, 128)	0
flatten_74 (Flatten)	(None, 2048)	0
dense_147 (Dense)	(None, 256)	524544
dense_148 (Dense)	(None, 6)	1542

Total params: 623,526

Trainable params: 623,526

Non-trainable params: 0



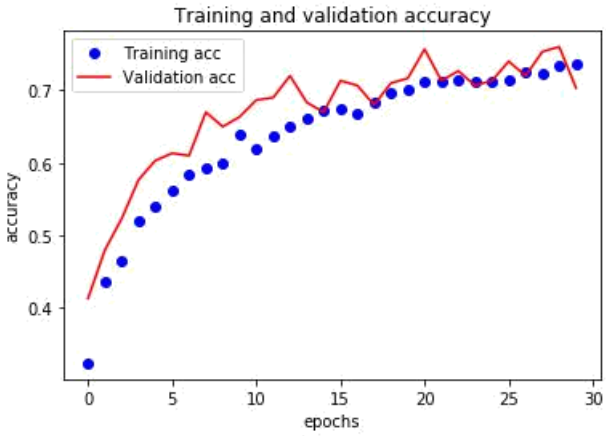


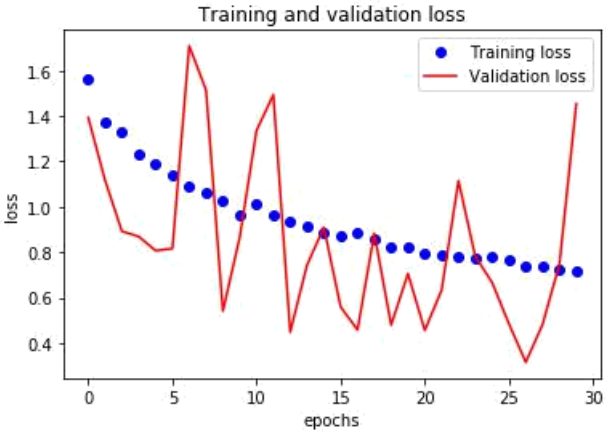
Parameterelerimiz görülüyor danse layer da çok fazla ve ondan oraya 0.6 verdim ve bu değerlerin grafiğe başarıya yansıdığınıda görüyorsunuz.

3.Şimdi burada artık aynı ağı dropout suz şekilde zenginleştirme kullanarak ezberin önüne geçecez ve başarıyı arttıracağız .

```
# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255,
                                   horizontal_flip=True,
                                   rotation_range=40,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   fill_mode='nearest')
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
      train_dir,
      target_size=(32, 32),
      batch_size=200,
      class_mode='categorical')
validation_generator =
      test_datagen.flow_from_directory( validation_dir,
      target_size=(32, 32),
      batch_size=10,
      class_mode='categorical')
```

Kodumuza bu kodları eklediğimizde hiçbir şeyi değiştirmeden çalıştırıyoruz çünkü yukarıda fit_generator ile çalıştırıyorduk model1 olark belirttiğim ağ modeli kısmı değiştirilecek sadece. Bu bilgiyi de verdiğime göre hemen kodlarımızı yorumlayalım öncelikle bu kod her seferinde datayı değiştiriyor buyuzden ezberleme gibi bir şey söz konusu bile olamıyor oldukça da başarılı sonuç aldım. Train içerisindeki görsellerde oynama yaptım ve bunları yeniden boyutlandırdım ters çevirdim yanal olarak daha sonra yukardan ve yanal olarak görseli bir miktar kaydırıyoruz ve bir miktar yaklaştırıyoruz altındaki kodda ise hedef uzunluğumuzu ve bach size ımızı belirliyoruz ilkinde 200 de bir diğerinde 10 da bir olacak şekilde çalıştırdık ve oldukça başarılı oldu bubizim güncellememizi belirler ve deneme yanılma ile buldum ve sonucu grafik olarak aşağıya bırakıyorum.





Dropout kullanmadan veri zenginleştirerek training acc ve validation acc yi birbirinine yakınlaştırarak ezberin önüne geçebildik. BAŞARILI bir sonuç elde ettik tüm grafikler %70 seviyelerinde.