Step 1 — Install libraries

In [1]:
```
from PIL import Image
import os
import cv2

import numpy as np

import keras
from keras.utils import np_utils
```

```
Using TensorFlow backend.
```

Step 2 — Prepare Dataset

In [2]:
```
data=[]
labels=[]
#
canes=os.listdir("Animals/raw-img/canes")
for cane in canes:
    imag=cv2.imread("Animals/raw-img/canes/"+cane)
    img_from_ar = Image.fromarray(imag, 'RGB')
    resized_image = img_from_ar.resize((50, 50))
    data.append(np.array(resized_image))
    labels.append(0)

cavallos=os.listdir("Animals/raw-img/cavallos")
for cavallo in cavallos:
    imag=cv2.imread("Animals/raw-img/cavallos/"+cavallo)
    img_from_ar = Image.fromarray(imag, 'RGB')
    resized_image = img_from_ar.resize((50, 50))
    data.append(np.array(resized_image))
    labels.append(1)

elefantes=os.listdir("Animals/raw-img/elefantes")
for elefante in elefantes:
    imag=cv2.imread("Animals/raw-img/elefantes/"+elefante)
    img_from_ar = Image.fromarray(imag, 'RGB')
    resized_image = img_from_ar.resize((50, 50))
    data.append(np.array(resized_image))
    labels.append(2)

farfallas=os.listdir("Animals/raw-img/farfallas")
for farfalla in farfallas:
    imag=cv2.imread("Animals/raw-img/farfallas/"+farfalla)
    img_from_ar = Image.fromarray(imag, 'RGB')
    resized_image = img_from_ar.resize((50, 50))
    data.append(np.array(resized_image))
    labels.append(3)

gallinas=os.listdir("Animals/raw-img/gallinas")
for gallina in gallinas:
    imag=cv2.imread("Animals/raw-img/gallinas/"+gallina)
    img_from_ar = Image.fromarray(imag, 'RGB')
    resized_image = img_from_ar.resize((50, 50))
    data.append(np.array(resized_image))
    labels.append(4)

gattos=os.listdir("Animals/raw-img/gattos")
for gatto in gattos:
    imag=cv2.imread("Animals/raw-img/gattos/"+gatto)
    img_from_ar = Image.fromarray(imag, 'RGB')
    resized_image = img_from_ar.resize((50, 50))
    data.append(np.array(resized_image))
    labels.append(5)

muccas=os.listdir("Animals/raw-img/muccas")
for mucca in muccas:
    imag=cv2.imread("Animals/raw-img/muccas/"+mucca)
    img_from_ar = Image.fromarray(imag, 'RGB')
    resized_image = img_from_ar.resize((50, 50))
```

```
    data.append(np.array(resized_image))
    labels.append(6)

pecoras=os.listdir("Animals/raw-img/pecoras")
for pecora in pecoras:
    imag=cv2.imread("Animals/raw-img/pecoras/"+pecora)
    img_from_ar = Image.fromarray(imag, 'RGB')
    resized_image = img_from_ar.resize((50, 50))
    data.append(np.array(resized_image))
    labels.append(7)


ragnos=os.listdir("Animals/raw-img/ragnos")
for ragno in ragnos:
    imag=cv2.imread("Animals/raw-img/ragnos/"+ragno)
    img_from_ar = Image.fromarray(imag, 'RGB')
    resized_image = img_from_ar.resize((50, 50))
    data.append(np.array(resized_image))
    labels.append(8)

scoiattolos=os.listdir("Animals/raw-img/scoiattolos")
for scoiattolo in scoiattolos:
    imag=cv2.imread("Animals/raw-img/scoiattolos/"+scoiattolo)
    img_from_ar = Image.fromarray(imag, 'RGB')
    resized_image = img_from_ar.resize((50, 50))
    data.append(np.array(resized_image))
    labels.append(9)
```

Since the “data” and “labels” are normal array , convert them to numpy arrays-

In [3]:
```
animals=np.array(data)
labels=np.array(labels)
```

Now save these numpy arrays so that you dont need to do this image manipulation again.

In [4]:
```
np.save("animals",animals)
np.save("labels",labels)
```

Now shuffle the “animals” and “labels” set so that you get good mixture when you separate the dataset into train and test

In [5]:
```
s=np.arange(animals.shape[0])
np.random.shuffle(s)
animals=animals[s]
labels=labels[s]
```

Make a variable num_classes which is the total number of animal categories and a variable data_length which is size of dataset

In [6]:
```
num_classes=len(np.unique(labels))
data_length=len(animals)
```

Divide data into test and train

In [7]:
```
(x_train,x_test)=animals[(int)(0.1*data_length):],animals[:(int)(0.1*
data_length)]
x_train = x_train.astype('float32')/255
x_test = x_test.astype('float32')/255
train_length=len(x_train)
test_length=len(x_test)
```

Divide labels into test and train

In [8]:
```
(y_train,y_test)=labels[(int)(0.1*data_length):],labels[:(int)(0.1*da
ta_length)]
```

Make labels into One Hot Encoding

In [9]:
```
#One hot encoding
y_train=keras.utils.to_categorical(y_train,num_classes)
y_test=keras.utils.to_categorical(y_test,num_classes)
```

Step 3 — Making Keras model

In [10]:
```
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout
#make model
model=Sequential()
model.add(Conv2D(filters=16,kernel_size=2,padding="same",activation=
"relu",input_shape=(50,50,3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32,kernel_size=2,padding="same",activation=
"relu"))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64,kernel_size=2,padding="same",activation=
"relu"))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(500,activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(10,activation="softmax"))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 50, 50, 16)        208       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 25, 25, 16)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 25, 25, 32)        2080      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 64)        8256      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 64)          0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 6, 64)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 500)               1152500   
_________________________________________________________________
dropout_2 (Dropout)          (None, 500)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5010      
=================================================================
Total params: 1,168,054
Trainable params: 1,168,054
Non-trainable params: 0
_________________________________________________________________
```

Compile the model

In [11]:
```
# compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
```

Step 4 — Train the model

In [12]:
```
model.fit(x_train,y_train,batch_size=50
          ,epochs=100,verbose=1)
```

```
Epoch 1/100
23562/23562 [==============================] - 42s 2ms/step - loss:
1.7240 - accuracy: 0.3988
Epoch 2/100
23562/23562 [==============================] - 35s 2ms/step - loss:
1.3166 - accuracy: 0.5527
Epoch 3/100
23562/23562 [==============================] - 31s 1ms/step - loss:
1.1477 - accuracy: 0.6080
Epoch 4/100
23562/23562 [==============================] - 29s 1ms/step - loss:
1.0454 - accuracy: 0.6410
Epoch 5/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.9523 - accuracy: 0.6777
Epoch 6/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.8774 - accuracy: 0.7012
Epoch 7/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.7916 - accuracy: 0.7305
Epoch 8/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.7195 - accuracy: 0.7542
Epoch 9/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.6479 - accuracy: 0.7787
Epoch 10/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.5723 - accuracy: 0.8046
Epoch 11/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.5139 - accuracy: 0.8253
Epoch 12/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.4497 - accuracy: 0.8437
Epoch 13/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.3959 - accuracy: 0.8644
Epoch 14/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.3566 - accuracy: 0.8752
Epoch 15/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.3103 - accuracy: 0.8925
Epoch 16/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.2833 - accuracy: 0.9021
Epoch 17/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.2551 - accuracy: 0.9110
Epoch 18/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.2299 - accuracy: 0.9204
Epoch 19/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.2053 - accuracy: 0.9284
```

```
Epoch 20/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1988 - accuracy: 0.9316
Epoch 21/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1806 - accuracy: 0.9389
Epoch 22/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1720 - accuracy: 0.9396
Epoch 23/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1658 - accuracy: 0.9429
Epoch 24/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1520 - accuracy: 0.9475
Epoch 25/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1478 - accuracy: 0.9498
Epoch 26/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1432 - accuracy: 0.9514
Epoch 27/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1429 - accuracy: 0.9510
Epoch 28/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1361 - accuracy: 0.9523
Epoch 29/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1272 - accuracy: 0.9573
Epoch 30/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1278 - accuracy: 0.9567
Epoch 31/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1307 - accuracy: 0.9555
Epoch 32/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1206 - accuracy: 0.9585
Epoch 33/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1239 - accuracy: 0.9587
Epoch 34/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1137 - accuracy: 0.9626
Epoch 35/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1122 - accuracy: 0.9621
Epoch 36/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1109 - accuracy: 0.9627
Epoch 37/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1018 - accuracy: 0.9649
Epoch 38/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0983 - accuracy: 0.9674
```

```
Epoch 39/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1048 - accuracy: 0.9651
Epoch 40/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1039 - accuracy: 0.9650
Epoch 41/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.1069 - accuracy: 0.9652
Epoch 42/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0953 - accuracy: 0.9681
Epoch 43/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0852 - accuracy: 0.9706
Epoch 44/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0956 - accuracy: 0.9669
Epoch 45/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0942 - accuracy: 0.9677
Epoch 46/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0905 - accuracy: 0.9700
Epoch 47/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0924 - accuracy: 0.9694
Epoch 48/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0930 - accuracy: 0.9691
Epoch 49/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0857 - accuracy: 0.9712
Epoch 50/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0832 - accuracy: 0.9722
Epoch 51/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0897 - accuracy: 0.9694
Epoch 52/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0779 - accuracy: 0.9751
Epoch 53/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0824 - accuracy: 0.9717
Epoch 54/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0859 - accuracy: 0.9720
Epoch 55/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0777 - accuracy: 0.9760
Epoch 56/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0814 - accuracy: 0.9732
Epoch 57/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0815 - accuracy: 0.9731
```

```
Epoch 58/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0729 - accuracy: 0.9744
Epoch 59/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0778 - accuracy: 0.9742
Epoch 60/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0792 - accuracy: 0.9736
Epoch 61/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0717 - accuracy: 0.9761
Epoch 62/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0717 - accuracy: 0.9764
Epoch 63/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0779 - accuracy: 0.9735
Epoch 64/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0660 - accuracy: 0.9779
Epoch 65/100
23562/23562 [==============================] - 4577s 194ms/step - los
s: 0.0768 - accuracy: 0.9760
Epoch 66/100
23562/23562 [==============================] - 33s 1ms/step - loss:
0.0684 - accuracy: 0.9775
Epoch 67/100
23562/23562 [==============================] - 29s 1ms/step - loss:
0.0701 - accuracy: 0.9756
Epoch 68/100
23562/23562 [==============================] - 31s 1ms/step - loss:
0.0669 - accuracy: 0.9784
Epoch 69/100
23562/23562 [==============================] - 29s 1ms/step - loss:
0.0755 - accuracy: 0.9750
Epoch 70/100
23562/23562 [==============================] - 29s 1ms/step - loss:
0.0740 - accuracy: 0.9764
Epoch 71/100
23562/23562 [==============================] - 29s 1ms/step - loss:
0.0696 - accuracy: 0.9751
Epoch 72/100
23562/23562 [==============================] - 29s 1ms/step - loss:
0.0665 - accuracy: 0.9780
Epoch 73/100
23562/23562 [==============================] - 29s 1ms/step - loss:
0.0716 - accuracy: 0.9767
Epoch 74/100
23562/23562 [==============================] - 29s 1ms/step - loss:
0.0755 - accuracy: 0.9753
Epoch 75/100
23562/23562 [==============================] - 29s 1ms/step - loss:
0.0618 - accuracy: 0.9798
Epoch 76/100
23562/23562 [==============================] - 29s 1ms/step - loss:
0.0716 - accuracy: 0.9766
```

```
Epoch 77/100
23562/23562 [==============================] - 26s 1ms/step - loss:
0.0658 - accuracy: 0.9786
Epoch 78/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0655 - accuracy: 0.9775
Epoch 79/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0655 - accuracy: 0.9781
Epoch 80/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0652 - accuracy: 0.9783
Epoch 81/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0603 - accuracy: 0.9798
Epoch 82/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0635 - accuracy: 0.9787
Epoch 83/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0653 - accuracy: 0.9778
Epoch 84/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0697 - accuracy: 0.9776
Epoch 85/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0588 - accuracy: 0.9805
Epoch 86/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0626 - accuracy: 0.9797
Epoch 87/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0680 - accuracy: 0.9786
Epoch 88/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0648 - accuracy: 0.9802
Epoch 89/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0671 - accuracy: 0.9792
Epoch 90/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0581 - accuracy: 0.9817
Epoch 91/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0575 - accuracy: 0.9809
Epoch 92/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0544 - accuracy: 0.9818
Epoch 93/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0620 - accuracy: 0.9790
Epoch 94/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0620 - accuracy: 0.9804
Epoch 95/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0623 - accuracy: 0.9803
```

```
Epoch 96/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0538 - accuracy: 0.9834
Epoch 97/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0549 - accuracy: 0.9818
Epoch 98/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0623 - accuracy: 0.9806
Epoch 99/100
23562/23562 [==============================] - 27s 1ms/step - loss:
0.0593 - accuracy: 0.9805
Epoch 100/100
23562/23562 [==============================] - 28s 1ms/step - loss:
0.0621 - accuracy: 0.9806
```

Out[12]:

```
<keras.callbacks.callbacks.History at 0x7f61204a8da0>
```

Step 5 — Test the model

In [13]:

```
score = model.evaluate(x_test, y_test, verbose=1)
print('\n', 'Test accuracy:', score[1])
```

```
2617/2617 [==============================] - 1s 388us/step

 Test accuracy: 0.6725257635116577
```

Step 6 — Predicting on single images

In [15]:
```
def convert_to_array(img):
    im = cv2.imread(img)
    img = Image.fromarray(im, 'RGB')
    image = img.resize((50, 50))
    return np.array(image)
def get_animal_name(label):
    if label==0:
        return "cane"
    if label==1:
        return "cavallo"
    if label==2:
        return "elefante"
    if label==3:
        return "farfalla"
    if label==4:
        return "gallina"
    if label==5:
        return "gatto"
    if label==6:
        return "mucca"
    if label==7:
        return "pecora"
    if label==8:
        return "ragno"
    if label==9:
        return "scoiattolo"
def predict_animal(file):
    print("Predicting .................................")
    ar=convert_to_array(file)
    ar=ar/255
    label=1
    a=[]
    a.append(ar)
    a=np.array(a)
    score=model.predict(a,verbose=1)
    print(score)
    label_index=np.argmax(score)
    print(label_index)
    acc=np.max(score)
    animal=get_animal_name(label_index)
    print(animal)
    print("The predicted Animal is a "+animal+" with accuracy =    "+
str(acc))
```

In [16]:
```
predict_animal("skurl2.jpg")
```

```
Predicting .................................
1/1 [==============================] - 0s 32ms/step
[[3.6593981e-05 1.2800480e-06 1.5365894e-06 2.0043242e-07 2.5707851e-
03
  3.1246153e-05 7.8855453e-05 6.5164762e-03 1.9206333e-05 9.9074382e-
01]]
9
scoiattolo
The predicted Animal is a scoiattolo with accuracy =    0.9907438
```

Alternative: Use the Python API to export directly to TF.js Layers format

In [17]:
```
#import tensorflowjs as tfjs
import  tensorflowjs as tfjs

import tensorflow as tf
tf.__version__
```

Out[17]: '2.1.0'

In [ ]:
```
# load json and create model
#from keras.models import model_from_json

#json_file = open('model.json', 'r')
#loaded_model_json = json_file.read()
#json_file.close()
#loaded_model = model_from_json(loaded_model_json)
# load weights into new model
#loaded_model.load_weights("model.h5")
#loaded_model.compile(loss='categorical_crossentropy', optimizer='ada
m',
 #                 metrics=['accuracy'])
print("Loaded model from disk")
```

In [25]:
```
os.getcwd()
tfjs.converters.save_keras_model(model, '/home/liam/Desktop/info370')
```

In [ ]:
```
```