Wykorzystanie sztucznej inteligencji w robotyce

„Sztuczna inteligencja (AI) to teoria i rozwój systemów komputerowych zdolnych do wykonywania zadań, które normalnie wymagają ludzkiej inteligencji, takich jak percepcja wzrokowa, rozpoznawanie mowy, podejmowanie decyzji i tłumaczenie między językami". -DICTIONARY.COM

Czyli sztuczna inteligencja ma zastąpić ludzi? Cóż, nie za bardzo. Nowoczesna sztuczna inteligencja ma na celu zwiększenie inteligencji maszyn w niektórych zadaniach, które zwykle wykonują ludzie. Nawet użycie słowa „inteligencja maszyn" jest nieco mylące, ponieważ trudno jest twierdzić, że maszyny w ogóle mają inteligencję, przynajmniej tak, jak myślimy o niej u ludzi. Zamiast filozoficznej debaty, skupmy się na tym, jak wykorzystać niektóre nowoczesne techniki sztucznej inteligencji na przykładzie prawdziwego robota. Jaką więc technikę sztucznej inteligencji możemy zastosować w naszym zrobotyzowanym samochodzie? Okazuje się, że w samochodzie jest kamera Pi, a widzenie komputerowe jest naprawdę trudne, więc zróbmy coś z tym. Sprawienie, by roboty widziały, jest łatwe, ale sprawienie, by zrozumiały, co widzą, jest wyjątkowo trudne. Jeśli chcesz uczyć się widzenia komputerowego za pomocą Pythona, sprawdź Projekty widzenia komputerowego z OpenCV i Pythonem 3 autorstwa Matthew Revera.

Projekt : Zejście na psy

Pokażemy, jak zbudować sieć neuronową uczącą się maszynowo i nauczyć ją rozpoznawać koty i psy. (Jest to umiejętność, którą powinny posiadać wszystkie roboty.) Będziemy trenować sieć, używając podzbioru 1000 obrazów z 25 000 elementów bazy danych zdjęć kotów i psów z bazy danych kotów i psów Kaggle, używając TensorFlow. TensorFlow to pakiet Pythona, który jest również przeznaczony do obsługi sieci neuronowych opartych na macierzach i grafach przepływowych. Jest podobny do NumPy, ale różni się pod jednym zasadniczym względem: TensorFlow jest przeznaczony do użytku w aplikacjach uczenia maszynowego i sztucznej inteligencji, dlatego zawiera biblioteki i funkcje zaprojektowane dla tych aplikacji.

Konfigurowanie projektu

W przypadku systemów Windows, Linux i Raspberry Pi sprawdź ten oficjalny link TensorFlow: https://www.tensorflow.org/install/pip. Pobierz TensorFlow i zainstaluj zgodnie z instrukcjami. Pobierz skróconą listę bazy danych kotów i psów tutaj

: https://github.com/switchdoclabs/CatsAndDogsTruncatedData.

Ma około 65 MB i jest dołączony do naszego oprogramowania na stronie dummies.com. Aby uzyskać więcej eksperymentów, pobierz pełny zestaw danych o psach i kotach z tego łącza: https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data.

Innym źródłem pełnych danych jest:

https://www.microsoft.com/en-us/download/details.aspx?id=54765. Rozpakuj główny folder, a następnie rozpakuj podfoldery test.zip i train.zip. Test zawiera obrazy testowe, które nie zostały sklasyfikowane, a train zawiera nasze dane treningowe, których użyjemy do trenowania naszej sieci neuronowej.

Uruchom następujące polecenie w katalogu programu, aby pobrać dane: git clone https://github.com/switchdoclabs/CatsAndDogsTruncatedData.git

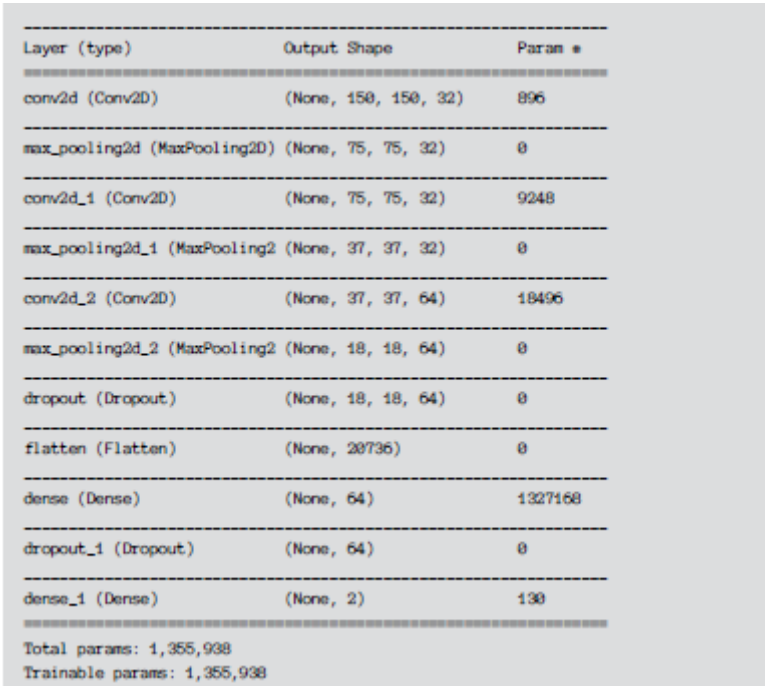
Teraz, gdy mamy gotowe dane, chodźmy trenować tę sieć.

Uczenie maszynowe za pomocą TensorFlow

Naszym celem w tej sekcji jest pełne wyszkolenie naszej sieci neuronowej uczenia maszynowego w zakresie różnic między kotami a psami, zweryfikowanie danych testowych, a następnie zapisanie wytrenowanej sieci neuronowej, abyśmy mogli faktycznie używać jej w naszym robocie. Wtedy zacznie się prawdziwa zabawa! Jeśli po uruchomieniu następującego programu zostanie wyświetlony komunikat ImportError: Brak nazwy modułu „seaborn", wpisz sudo pip3 install seaborn. Zaczynamy używać dość prostej dwuwarstwowej sieci neuronowej dla naszej sieci uczenia maszynowego kotów i psów. Dostępnych jest wiele bardziej złożonych sieci, które mogą dawać lepsze wyniki, ale mieliśmy nadzieję, że ta będzie wystarczająco dobra dla naszych potrzeb. Istnieje również możliwość użycia znacznie większego zestawu danych (nasz zestaw danych szkoleniowych obejmuje 1000 kotów i 1000 psów, ale w pełnym zbiorze danych dostępnych jest ponad 25 000 obrazów). Używanie prostej dwuwarstwowej sieci neuronowej w zbiorze danych kotów i psów nie działało zbyt dobrze, ponieważ osiągnęliśmy tylko około 51 procent wskaźnika wykrywalności (50 procent to tyle samo, co losowe zgadywanie), więc musieliśmy przejść do bardziej złożonego sieć neuronowa, która działa lepiej na złożonych obrazach. Możesz używać CNN (konwolucyjnych sieci neuronowych) zamiast prostych sieci neuronowych, augmentacji danych (zwiększanie liczby próbek szkoleniowych poprzez obracanie, przesuwanie i powiększanie obrazów) oraz wielu innych technik, które wykraczają poza zakres tej książki. Zamierzamy użyć dość standardowego sześciowarstwowego CNN zamiast naszej prostej sieci neuronowej. Zmieniliśmy warstwy modelu w naszym programie, aby korzystać z następującego sześciopoziomowego modelu warstw konwolucyjnych. Musisz po prostu pokochać to, jak łatwo Keras i TensorFlow radykalnie zmieniają sieć neuronową.

KONWOLUCYJNE SIECI NEURONOWE

Sieci CNN skanują obrazy i analizują je kawałek po kawałku, powiedzmy w oknie 5x5, które przesuwa się za każdym razem o długość kroku dwóch pikseli, aż obejmie całą wiadomość. To jak patrzenie na obraz za pomocą mikroskopu; widzisz tylko małą część obrazu w danym momencie, ale ostatecznie widzisz cały obraz. Za każdym razem, gdy przeglądamy dane, nazywamy epoką. Przejście do sieci CNN na Raspberry Pi wydłużyło czas pojedynczej epoki do 1000 sekund z 10-sekundowej epoki, którą mieliśmy w prostej sieci dwuwarstwowej. I ma wykorzystanie procesora na poziomie 352 procent, co oznacza, że używa 3,5 rdzenia w Raspberry Pi, maszynie, która ma łącznie tylko 4. Oznacza to dość wysokie wykorzystanie Raspberry Pi 3B. Mała płyta zużywa prawie 3,8 W w porównaniu z około 1,5 W normalnie. Możesz zobaczyć złożoność naszej nowej sieci, patrząc na model. podsumowanie() wyniki:



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 150, 150, 32)	896
max_pooling2d (MaxPooling2D)	(None, 75, 75, 32)	0
conv2d_1 (Conv2D)	(None, 75, 75, 32)	9248
max_pooling2d_1 (MaxPooling2	(None, 37, 37, 32)	0
conv2d_2 (Conv2D)	(None, 37, 37, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 18, 18, 64)	0
dropout (Dropout)	(None, 18, 18, 64)	0
flatten (Flatten)	(None, 20736)	0
dense (Dense)	(None, 64)	1327168
dropout_1 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 2)	130

Total params: 1,355,938
Trainable params: 1,355,938

Kod

```
#import libraries
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
import tensorflow as tf
from tensorflow.python.framework import ops
from tensorflow.examples.tutorials.mnist import input_data
from PIL import Image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import *
# load data
img_width = 150
img_height = 150
train_data_dir = 'data/train'
valid_data_dir = 'data/validation'
datagen = ImageDataGenerator(rescale = 1./255)
train_generator = datagen.flow_from_directory(
```

```
directory=train_data_dir,
target_size=(img_width,img_height),
classes=['dogs','cats'],
class_mode='binary',
batch_size=16)
validation_generator = datagen.flow_from_directory(directory=valid_data_dir,
target_size=(img_width,img_height),
classes=['dogs','cats'],
class_mode='binary',
batch_size=32)
# build model
model = tf.keras.Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(150, 150, 3), padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))
model.compile(optimizer=tf.train.AdamOptimizer(),
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
print (model.summary())
# train model
print('starting training....')
history = model.fit_generator(generator=train_generator,
```

```
steps_per_epoch=2048 // 16,epochs=20,

validation_data=validation_generator,validation_steps=832//16)

print('training finished!!')

# save coefficients

model.save("CatsVersusDogs.trained")

# Get training and test loss histories

training_loss = history.history['loss']

accuracy = history.history['acc']

# Create count of the number of epochs

epoch_count = range(1, len(training_loss) 1)

# Visualize loss history

plt.figure(0)

plt.plot(epoch_count, training_loss, 'r--')

plt.plot(epoch_count, accuracy, 'b--')

plt.legend(['Training Loss', 'Accuracy'])

plt.xlabel('Epoch')

plt.ylabel('History')

plt.grid(True)

plt.show(block=True);
```

Zapisz kod w pliku o nazwie CatsVersusDogs.py.

Badanie kodu

Kod ma 93 linie. To niesamowite, co możemy zrobić z tak małą liczbą linii kodu. Najpierw importujemy wszystkie biblioteki:

```
#import libraries

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import seaborn as sns

import tensorflow as tf

from tensorflow.python.framework import ops

from tensorflow.examples.tutorials.mnist import input_data
```

```
from PIL import Image

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.layers import *
```

Następnie przetwarzamy dane na tensory (macierze) w celu trenowania przez naszą sieć neuronową. Obrazy kotów i psów są przechowywane w oddzielnych katalogach pod danymi:

```
# load data

img_width = 150

img_height = 150

train_data_dir = 'data/train'

valid_data_dir = 'data/validation'

datagen = ImageDataGenerator(rescale = 1./255)

train_generator = datagen.flow_from_directory(

directory=train_data_dir,

target_size=(img_width,img_height),

classes=['dogs','cats'],

class_mode='binary',

batch_size=16)

validation_generator = datagen.flow_from_directory(directory=valid_data_dir,

target_size=(img_width,img_height),

classes=['dogs','cats'],

class_mode='binary',

batch_size=32)
```

Teraz budujemy sieć neuronową, która stanowi podstawę modelu uczenia maszynowego. Jest to sześciowarstwowa sieć neuronowa:

```
# build model

model = tf.keras.Sequential()
```

Pierwsza warstwa to konwolucyjna sieć neuronowa 2D, która zaczyna wydobywać cechy ze zdjęcia. Używamy RELU (prostowanej jednostki liniowej) jako funkcji aktywacji neuronów. Jest to dość powszechne:

```
model.add(Conv2D(32, (3, 3), input_shape=(150, 150, 3), padding='same',

activation='relu'))
```

Kolejna warstwa, MaxPooling2D, upraszcza funkcje dostępne w poprzedniej warstwie:

model.add(MaxPooling2D(pool_size=(2, 2)))

Następnie dodawane są kolejne dwie warstwy konwolucyjnych sieci neuronowych, po których następuje warstwa puli:

model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))

model.add(MaxPooling2D(pool_size=(2, 2)))

Jeden z problemów z sieciami neuronowymi nazywa się „nadmiernym” — gdy maszyna zbyt dokładnie dopasowuje dane, przez co sieć nie dopasowuje żadnych nowych, nieco odmiennych obrazów. Warstwy odrzucania pomagają tutaj, losowo ustawiając jednostki wejściowe na zero. Usuwamy 25 procent jednostek wejściowych z tej warstwy:

model.add(Dropout(0.25))

Teraz łączymy dane w jednowymiarową tablicę, a następnie używamy końcowej, gęsto połączonej warstwy 64 neuronów:

model.add(Flatten())

model.add(Dense(64, activation='relu'))

Następnie pomiń kolejne 50 procent jednostek wejściowych, aby pomóc w nadmiernym przeliczaniu:

model.add(Dropout(0.5))

I wreszcie nasza warstwa wyjściowa „Kot czy pies”:

model.add(Dense(2, activation='softmax'))

model.compile(optimizer=tf.train.AdamOptimizer(),

loss='sparse_categorical_crossentropy',

metrics=['accuracy'])

print (model.summary())

train model

To szkolenie na 1000 zdjęć zajmuje około pięciu godzin na Raspberry Pi 3B:

print('starting training....')

history = model.fit_generator(generator=train_generator,

steps_per_epoch=2048 // 16,epochs=20,

validation_data=validation_generator,validation_steps=832//16)

print('training finished!!')

Ten następny wiersz kodu zapisuje współczynniki do późniejszego wykorzystania. Oszczędza to 5 godzin na przebieg!

save coefficients

model.save("CatsVersusDogs.trained")

Na koniec generujemy wykres za pomocą MatPlotLib, który pokazuje, jak poprawia się dokładność z każdą epoką:

Get training and test loss histories

training_loss = history.history['loss']

accuracy = history.history['acc']

Create count of the number of epochs

epoch_count = range(1, len(training_loss) 1)

Visualize loss history

plt.figure(0)

plt.plot(epoch_count, training_loss, 'r--')

plt.plot(epoch_count, accuracy, 'b--')

plt.legend(['Training Loss', 'Accuracy'])

plt.xlabel('Epoch')

plt.ylabel('History')

plt.grid(True)

plt.show(block=True);

Wyniki

Zainstaluj bibliotekę h5py przed uruchomieniem tego programu. W przeciwnym razie instrukcja save nie zadziała.

sudo apt-get install python-h5py

Czas na przedstawienie! Uruchom następujące polecenie w oknie terminala:

python3 CatsVersusDogs.py

Wygenerowanie 20 epok treningu i zapisanie wyników w naszym pliku CatsVersusDogs.training zajmuje Raspberry Pi 3B około pięciu godzin. Migawka z ostatniej epoki jest następująca:

Epoch 20/20

128/128 [==============================] - 894s 7s/step - loss: 0.0996 - acc:

0.9609 - val_loss: 1.1069 - val_acc: 0.7356

training finished!!

Możesz bezpiecznie ignorować ostrzeżenia TensorFlow, takie jak:

/usr/lib/python3.5/importlib/_bootstrap.py:222: RuntimeWarning: compiletime

version 3.4 of module 'tensorflow.python.framework.fast_tensor_util' does not

match runtime version 3.5

return f(*args, **kwds)

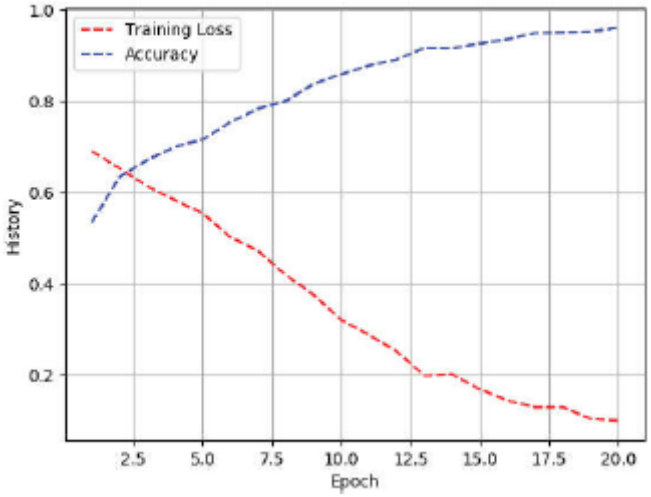
/usr/lib/python3.5/importlib/_bootstrap.py:222: RuntimeWarning: builtins.type

size changed, may indicate binary incompatibility. Expected 432, got 412

return f(*args, **kwds)

Zostanie to naprawione w nadchodzącej wersji.

Po 5-godzinnym szkoleniu osiągnęliśmy dokładność na poziomie 96 procent! Całkiem dobre. Rysunek pokazuje, jak poprawiła się dokładność podczas treningu.



Sądząc po kształcie krzywej, moglibyśmy być trochę lepsi, uruchamiając więcej epok, ale to jest wystarczające dla naszego przykładu.

Testowanie wytrenowanej sieci

Czas na wykonanie testu na zewnętrznym obrazie kota/psa. Wykorzystamy przeszkolone dane z powyższej sesji szkoleniowej dotyczącej sieci neuronowej, aby wykonać kilka prognoz dotyczących nowych obrazów w sieci neuronowej.

Wybraliśmy zdjęcie kota , ponieważ jest to zdjęcie o niskim kontraście, podczas gdy pies jest prawie standardowym zdjęciem psa .

Kod :

```
#import libraries
import numpy as np
import tensorflow as tf
from tensorflow.python.framework import ops
from PIL import Image
print("import complete")
# load model
img_width = 150
img_height = 150
class_names = ["Dog", "Cat"]
```

```
model = tf.keras.models.load_model(
"CatsVersusDogs.trained",compile=True)
print (model.summary())
# do cat single image
imageName = "Cat150x150.jpeg"
testImg = Image.open(imageName)
testImg.load()
data = np.asarray( testImg, dtype="float" )
data = np.expand_dims(data, axis=0)
singlePrediction = model.predict(data, steps=1)
NumberElement = singlePrediction.argmax()
Element = np.amax(singlePrediction)
print(NumberElement)
print(Element)
print(singlePrediction)
print ("Our Network has concluded that the file '"
imageName "' is a " class_names[NumberElement])
print (str(int(Element*100)) "% Confidence Level")
# do dog single image
imageName = "Dog150x150.JPG"
testImg = Image.open(imageName)
testImg.load()
data = np.asarray( testImg, dtype="float" )
data = np.expand_dims(data, axis=0)
singlePrediction = model.predict(data, steps=1)
NumberElement = singlePrediction.argmax()
Element = np.amax(singlePrediction)
print(NumberElement)
print(Element)
print(singlePrediction)
print ("Our Network has concluded that the file '"
```

imageName "' is a " class_names[NumberElement])

print (str(int(Element*100)) "% Confidence Level")

Wyjaśnienie kodu

Ten kod korzysta ze szkolenia wygenerowanego przez szkolenie z zestawem danych kotów i psów we wcześniejszej części tego rozdziału. Najpierw importujemy nasze biblioteki:

#import libraries

import numpy as np

import tensorflow as tf

from tensorflow.python.framework import ops

from PIL import Image

print("import complete")

load model

img_width = 150

img_height = 150

Skonfigurowaliśmy tablicę nazw klas, aby mieć nazwy klasyfikacyjne dla naszych obrazów:

class_names = ["Dog", "Cat"]

Oto miejsce, w którym ładujemy dane treningowe, które wygenerowaliśmy wcześniej dla modelu uczenia maszynowego sieci neuronowej:

model = tf.keras.models.load_model(

"CatsVersusDogs.trained",compile=True)

print (model.summary())

Teraz testujemy pojedynczy obraz kota:

do cat single image

imageName = "Cat150x150.jpeg"

testImg = Image.open(imageName)

testImg.load()

Konwertuj na tensor NumPy:

data = np.asarray(testImg, dtype="float")

Rozwiń wymiar, ponieważ ta funkcja szuka tablicy obrazów:

data = np.expand_dims(data, axis=0)

Teraz wykonujemy predykację na podstawie naszego obrazu:

singlePrediction = model.predict(data, steps=1)

```
NumberElement = singlePrediction.argmax()
Element = np.amax(singlePrediction)
print(NumberElement)
print(Element)
print(singlePrediction)
```

Zinterpretuj prognozę:

```
print ("Our Network has concluded that the file '"
imageName "' is a " class_names[NumberElement])
print (str(int(Element*100))  "+ % Confidence Level")
```

Następnie robimy to samo z pojedynczym obrazem psa:

```
# do dog single image
imageName = "Dog150x150.JPG"
testImg = Image.open(imageName)
testImg.load()
data = np.asarray( testImg, dtype="float" )
data = np.expand_dims(data, axis=0)
singlePrediction = model.predict(data, steps=1)
NumberElement = singlePrediction.argmax()
Element = np.amax(singlePrediction)
print(NumberElement)
print(Element)
print(singlePrediction)
print ("Our Network has concluded that the file '"
imageName "' is a " class_names[NumberElement])
print (str(int(Element*100)) + "% Confidence Level")
```

Wyniki

Zapisz kod w singleTestImage.py i uruchom za pomocą sudo python3 singleTest Image.py.

Oto wyniki:

```
import complete
WARNING:tensorflow:No training configuration found in save file: the model
   was *not* compiled. Compile it manually.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 150, 150, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 75, 75, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 37, 37, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 18, 18, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 18, 18, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 20736)             0
_________________________________________________________________
dense (Dense)                (None, 64)                1327168
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 130
=================================================================
Total params: 1,355,938
Trainable params: 1,355,938
Non-trainable params: 0
_________________________________________________________________
None
1
1.0
[[ 0.  1.]]
Our Network has concluded that the file 'Cat150x150.jpeg' is a Cat
100% Confidence Level
0
1.0
[[ 1.  0.]]
Our Network has concluded that the file 'Dog150x150.JPG' is a Dog
100% Confidence Level
```

Cóż, ta sieć działała bardzo dobrze. Zidentyfikował zarówno kota, jak i psa jako ich odpowiednie gatunki. Widzisz 100-procentowe przedziały ufności? W rzeczywistości jest to 99,99 procent lub coś w tym rodzaju i jest po prostu zaokrąglane do 100 procent przez formatowanie. Teraz, gdy mamy wyszkolony model i przetestowaliśmy go z kilkoma prawdziwymi obrazami, nadszedł czas, aby umieścić go w naszym robocie i użyć kamery Pi do zbadania psa i kota. Prawdziwym ograniczeniem sposobu, w jaki zbudowaliśmy tę sieć neuronową, jest to, że tak naprawdę patrzy ona tylko na obrazy kotów i psów i określa, czy obraz jest kotem czy psem. Jednak sieć klasyfikuje wszystko jako kota lub psa. Gdybyśmy mieli zbudować bardziej wszechstronną sieć, musielibyśmy ją wyszkolić, aby na przykład rozróżniała kota, psa lub sukienkę.

Przeprowadziliśmy jeszcze jeden test sieci, I zgodnie z oczekiwaniami sieć źle to zrozumiała:

Our Network has concluded that the file 'Dress150x150.JPG' is a Cat

Uczenie maszyny uczenia się w sposób ogólny to znacznie więcej. Jeśli pojawi się błąd, taki jak: undefined symbol: cblas_sgemm z importu numpy, spróbuj uruchomić program za pomocą sudo.

Zabieranie kotów i psów do naszego robota

Czas dodać nowe doświadczenie do robota z poprzedniego rozdziału. Zamierzamy zainstalować wyszkoloną sieć neuronową kotów i psów na robocie PiCar-B i użyć wbudowanych diod LED do wyświetlenia, czy wbudowana kamera Pi patrzy na kota czy na psa. Jak wspomniano wcześniej, nasza sieć neuronowa klasyfikuje prawie wszystko jako kota lub psa. Zatem to, co zamierzamy zrobić, to uruchomić klasyfikację sieci neuronowej, gdy zmieni się czujnik ultradźwiękowy, na przykład gdy pies lub kot może przejść przed robotem. Nasza konfiguracja testowa, ponieważ kot Johna nie chciał współpracować, jest pokazana na rysunku.

Ma robota wpatrującego się w ekran (uruchamiający czujnik ultradźwiękowy) pokazujący prezentację PowerPoint różnych kotów i psów.

Kod:

```
#!/usr/bin/python3
#using a neural network with the Robot
#import libraries
import numpy as np
import tensorflow as tf
from tensorflow.python.framework import ops
from PIL import Image
import RobotInterface
import time
import picamera
print("import complete")
RI = RobotInterface.RobotInterface()
# load neural network model
img_width = 150
img_height = 150
class_names = ["Dog", "Cat"]
model = tf.keras.models.load_model("CatsVersusDogs.trained",compile=True)
RI.centerAllServos()
RI.allLEDSOff()
# Ignore distances greater than one meter
DISTANCE_TO_IGNORE = 1000.0
# How many times before the robot gives up
DETECT_DISTANCE = 60
def bothFrontLEDSOn(color):
RI.allLEDSOff()
if (color == "RED"):
RI.set_Front_LED_On(RI.right_R)
RI.set_Front_LED_On(RI.left_R)
return
if (color == "GREEN"):
```

```
RI.set_Front_LED_On(RI.right_G)
RI.set_Front_LED_On(RI.left_G)
return
if (color == "BLUE"):
RI.set_Front_LED_On(RI.right_B)
RI.set_Front_LED_On(RI.left_B)
return
def checkImageForCat(testImg):
# check dog single image
data = np.asarray( testImg, dtype="float" )
data = np.expand_dims(data, axis=0)
singlePrediction = model.predict(data, steps=1)
print ("single Prediction =", singlePrediction)
NumberElement = singlePrediction.argmax()
Element = np.amax(singlePrediction)
print ("Our Network has concluded that the file '"
imageName "' is a " class_names[NumberElement])
return class_names[NumberElement]
try:
print("starting sensing")
Quit = False
trigger_count = 0
bothFrontLEDSOn("RED")
#RI.headTiltPercent(70)
camera = picamera.PiCamera()
camera.resolution = (1024, 1024)
camera.start_preview(fullscreen=False,
window=(150,150,100,100))
# Camera warm-up time
time.sleep(2)
while (Quit == False):
```

```
current_distance = RI.fetchUltraDistance()
print ("current_distance = ", current_distance)
if (current_distance < DETECT_DISTANCE):
trigger_count = trigger_count 1
print("classifying image")
camera.capture('FrontView.jpg')
imageName = "FrontView.jpg"
testImg = Image.open(imageName)
new_image = testImg.resize((150, 150))
new_image.save("FrontView150x150.jpg")
if (checkImageForCat(new_image) == "Cat"):
bothFrontLEDSOn("GREEN")
else:
bothFrontLEDSOn("BLUE")
time.sleep(2.0)
bothFrontLEDSOn("RED")
time.sleep(7.0)
except KeyboardInterrupt:
print("program interrupted")
print ("program finished")
```

Jak to działa

Większość poprzedniego kodu jest dość prosta i podobna do oprogramowania mózgu robota wcześniej . Jednak jedna część zasługuje na omówienie, a jest nią nasza funkcja klasyfikująca:

```
def checkImageForCat(testImg):
# check dog single image
data = np.asarray( testImg, dtype="float" )
data = np.expand_dims(data, axis=0)
singlePrediction = model.predict(data, steps=1)
print ("single Prediction =", singlePrediction)
NumberElement = singlePrediction.argmax()
Element = np.amax(singlePrediction)
```

print ("Our Network has concluded that the file '"

imageName "' is a " class_names[NumberElement])

return class_names[NumberElement]

Ta funkcja pobiera przychodzący obraz testowy (pobrany z kamery Pi, a następnie przeskalowany do obrazu o wymiarach 150 x 150 pikseli — format wymagany przez naszą sieć neuronową).

Wyniki

Zapisz program w pliku o nazwie robotVision.py i uruchom go — sudo python3 robotVision.py — aby maszyna zaczęła szukać kotów. Naprawdę powinieneś uruchomić ten program na GUI Raspberry Pi, abyś mógł zobaczyć podgląd małej kamery na ekranie. Oto niektóre wyniki z naszej konfiguracji testowej na rysunku powyżej:

current_distance = 20.05481719970703

classifying image

single Prediction = [[0. 1.]]

Our Network has concluded that the file 'FrontView.jpg' is a Cat

100.00% Confidence Level

current_distance = 20.038604736328125

classifying image

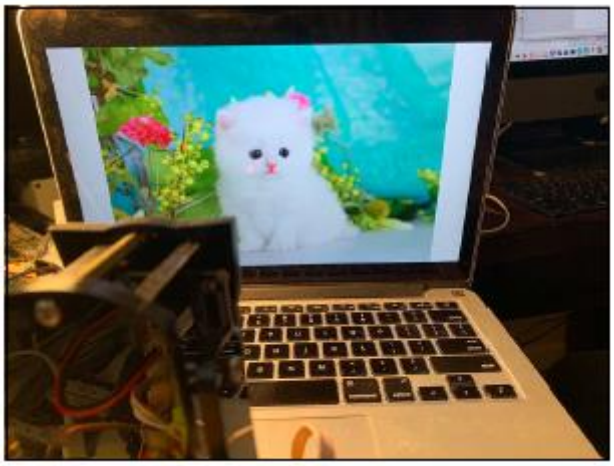
single Prediction = [[1. 0.]]

Our Network has concluded that the file 'FrontView.jpg' is a Dog

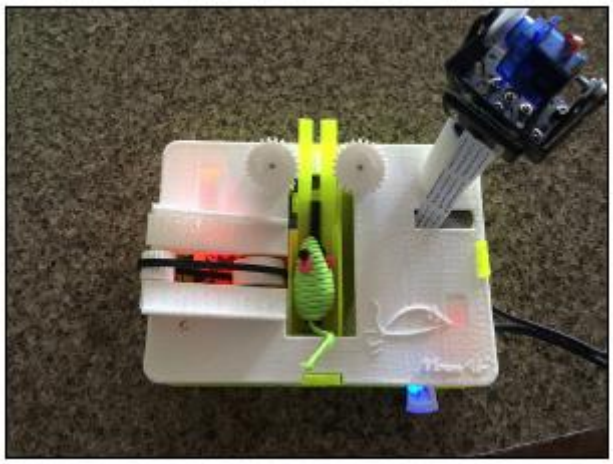
100.00% Confidence Level

current_distance = 19.977807998657227

Ogólnie wyniki były całkiem dobre. Znaleźliśmy różnice w rozpoznawaniu ze względu na oświetlenie, co nie było dużym zaskoczeniem. Jednak sieć konsekwentnie identyfikowała jedno zdjęcie jako psa, który w rzeczywistości był kotem.



Podejrzewamy, że to zwinięte uszy, ale jedną z naprawdę trudnych rzeczy w ustaleniu, co jest nie tak z modelem uczenia maszynowego sieci neuronowej, takim jak ten, jest to, że nie ma sposobu, aby naprawdę wiedzieć, w jaki sposób maszyna wykonała jego decyzja. Patrząc na sieć, możesz spojrzeć na 1,3 miliona parametrów i wag i nie możesz powiedzieć, gdzie jest źle. Po pewnym czasie zaczynasz wyczuwać, co działa, a co nie w uczeniu maszynowym, ale jeśli chodzi o ustalanie, co ta sieć robi dokładnie źle, po prostu nie masz szczęścia. Mamy tutaj nikczemny plan dotyczący wariantu tej sieci neuronowej w przyszłości. Zamierzamy użyć go w nowym projekcie, MouseAir opartym na Raspberry Pi, który uruchomi mysz-zabawkę, gdy kamera wykryje kota, ale nie wtedy, gdy wykryje psa. Możesz zobaczyć kamerę Pi w prawym górnym rogu rysunku . To zdecydowanie powinno być zabawne.



Inne rzeczy, które możesz zrobić dzięki technikom sztucznej inteligencji i robotowi. Jako część zaawansowanego mózgu Pythona w PiCar-B, istnieje wiele rzeczy, które możesz zrobić z sieciami neuronowymi i innymi technikami sztucznej inteligencji. Oto kilka przykładów:

Kot/nie kot

Przeszkolić naszą sieć Kot/Pies, aby koncentrowała się wyłącznie na kotach; wrzuca wszystko inne do kategorii „Not Cat”. Możesz to zrobić w obecnej sieci, ponieważ ten zestaw szkoleniowy ma tendencję do klasyfikowania wszystkiego, co nie wygląda jak kot, jako psa. Nie robi tego jednak cały czas. Pamiętasz zdjęcie sukienki na rysunku 4.4 (sklasyfikowane przez naszą sieć jako kot)?

Mikołaj/Nie Mikołaj

Weź strumieniowe wideo z naszego Raspberry Pi i od czasu do czasu chwyć ramkę i sprawdź za pomocą sieci neuronowej (wytrenowanej na obrazach Świętego Mikołaja), aby ustalić, czy Święty Mikołaj jest tutaj na Boże Narodzenie. Niemożliwe! Nie, czekaj. Zostało to zrobione w bardzo fajnej serii artykułów tutaj: https://www.pyimagesearch.com/2017/12/18/keras-deep-learning-raspberry-pi/.

Podążaj za piłką

Możesz zaprogramować Raspberry Pi za pomocą biblioteki o nazwie OpenCV (Open-Source Computer Vision), aby wykryć piłkę w strumieniu kamery i podać współrzędne obrazu. Następnie możesz jechać w jego kierunku. Możesz obrócić głowę i zmienić nachylenie, aby również jej szukać. Wersja tego projektu jest częścią oprogramowania Adeept dołączonego do PiCar-B. Możesz zobaczyć kod źródłowy wewnątrz server.py w katalogu serwera.

Używanie Alexy do sterowania robotem

Możesz użyć mocy programu Amazon Alexa, aby zlecić swojemu robotowi wykonanie różnych czynności. Jeśli chcesz zobaczyć, jak podłączyć Raspberry Pi do Alexy, aby wykonać właśnie taki projekt, zapoznaj się z ebookiem Voice Time: Connecting Your Raspberry Pi to the Amazon Alexa autorstwa Johna Shovica.

Sztuczna inteligencja i przyszłość robotyki

Naszym zdaniem sztuczna inteligencja to przyszłość robotyki. Ponieważ sprzęt staje się coraz tańszy, a techniki oprogramowania coraz łatwiej dostępne, robotów będzie coraz więcej zastosowań, nie tylko w produkcji, ale także w domu i na drodze. W nadchodzących latach spodziewamy się nowej sztucznej inteligencji w wielu nowych produktach konsumenckich. Jedyne pytanie, jakie mamy, brzmi: „Jak długo potrwa sztuczna inteligencja w moim tosterze?” Kupimy jeden, gdy wyjdą.