Robienie uczenia maszynowego w Pythonie

Co to znaczy nauczyć się czegoś? Jedna z definicji to „przyswojenie i opanowanie tego, co już o czymś wiadomo, oraz poszerzone wyjaśnienie znaczenia tej wiedzy". Inna definicja mówi, że „uczenie się to względnie trwała zmiana w wiedzy lub zachowaniu danej osoby, spowodowana doświadczeniem". Przy obecnym (i najprawdopodobniej jeszcze przez jakiś czas) stanie uczenia maszynowego jest to drugie określenie, które najlepiej pasuje do obecnego stanu sztucznej inteligencji. Nasza kultura rozwinęła algorytmy i programy, które mogą uczyć się o danych i danych sensorycznych oraz stosować tę wiedzę w nowych sytuacjach. Jednak nasze maszyny nie „rozumieją" niczego z tego, czego się nauczyły. Właśnie zgromadzili dane o swoich danych wejściowych i przekształcili te dane wejściowe w pewien rodzaj danych wyjściowych. Jednak nawet jeśli maszyna nie „rozumie" tego, czego się nauczyła, nie oznacza to, że nie można zrobić imponujących rzeczy za pomocą technik uczenia maszynowego, które zostaną omówione w tym rozdziale. Być może te techniki, które teraz opracowujemy, mogą doprowadzić do czegoś znacznie bardziej imponującego w przyszłości. Co to znaczy, że maszyna się czegoś uczy? Zastosujemy przybliżony pomysł, że jeśli maszyna może pobierać dane wejściowe i za pomocą jakiegoś procesu przekształcać te dane wejściowe w użyteczne dane wyjściowe, to możemy powiedzieć, że maszyna czegoś się nauczyła. Ta definicja ma szerokie znaczenie. Pisząc prosty program dodający dwie liczby, nauczyłeś tę maszynę czegoś. Nauczyła się dodawać dwie liczby. W tym rozdziale skupimy się na uczeniu maszynowym w sensie wykorzystania algorytmów i modeli statystycznych, które stopniowo poprawiają ich wydajność w konkretnym zadaniu. Jeśli brzmi to bardzo podobnie do naszych eksperymentów z sieciami neuronowymi w części 2, masz rację. Uczenie maszynowe dotyczy w dużej mierze sieci neuronowych, ale obejmuje także inne wyrafinowane techniki.

Uczenie się poprzez szukanie rozwiązań we wszystkich niewłaściwych miejscach

Jednym z prawdziwych problemów związanych z uczeniem maszynowym i ogólnie sztuczną inteligencją jest ustalenie, w jaki sposób algorytm może znaleźć najlepsze rozwiązanie. Słowo operacyjne jest tam najlepsze. Skąd wiemy, że dane rozwiązanie jest najlepsze? To naprawdę kwestia wyznaczenia celów i ich osiągnięcia (rozwiązanie może nie być najlepsze, ale może wystarczająco dobre). Niektórzy porównują problem znalezienia „najlepszego" rozwiązania do problemu osoby, która w mglisty dzień wędruje po okolicy, próbując znaleźć najwyższą górę w okolicy. Wspinasz się na górę i docierasz na szczyt, a potem ogłaszasz: „Jestem na najwyższej górze". Cóż, jesteś na najwyższej górze, którą widzisz, ale nie możesz widzieć przez mgłę. Jeśli jednak zdefiniujesz swój cel jako bycie na szczycie góry o wysokości ponad 1000 stóp i jesteś na wysokości 1250 stóp, to osiągnąłeś swój cel. Nazywa się to lokalnymi maksimami i może, ale nie musi, być najlepszymi dostępnymi maksimami. W tym rozdziale większość ustawiania celów (treningu maszyny) będzie wykonywana przy użyciu znanych rozwiązań problemu: najpierw wytrenujemy naszą maszynę, a następnie zastosujemy trening do nowych, podobnych przykładów problemu. Istnieją trzy główne typy algorytmów uczenia maszynowego:

* Uczenie nadzorowane: ten typ algorytmu buduje model danych, który zawiera zarówno dane wejściowe, jak i wyjściowe. Dane są znane jako dane treningowe. Jest to rodzaj uczenia maszynowego, który pokazujemy w tym rozdziale. *Uczenie bez nadzoru: w przypadku tego typu algorytmu dane zawierają tylko dane wejściowe, a algorytmy szukają w danych struktur i wzorców.

* Uczenie się przez wzmacnianie: ten obszar dotyczy oprogramowania podejmującego działania w oparciu o pewnego rodzaju kumulatywną nagrodę. Algorytmy te nie zakładają znajomości dokładnego modelu matematycznego i są stosowane, gdy dokładne modele są niedostępne. To najbardziej złożony obszar uczenia maszynowego, który może przynieść największe owoce w przyszłości.

Mając to na uwadze, przejdźmy do uczenia maszynowego za pomocą Pythona.

Klasyfikowanie ubrań za pomocą uczenia maszynowego

Skorzystamy z ogólnodostępnej szkoleniowej bazy danych Fashion-MNIST (Zmodyfikowany Narodowy Instytut Standardów i Technologii), która zawiera 60 000 produktów modowych z dziesięciu kategorii. Zawiera dane w formacie 28x28 pikseli z 6000 pozycji w każdej kategorii.

Daje nam to naprawdę interesujący zestaw danych do zbudowania aplikacji uczenia maszynowego Tens lub Flow/Keras = o wiele bardziej interesujący niż standardowa baza danych uczenia maszynowego MNIST, która zawiera tylko odręczne znaki.

Szkolenia i nauka z TensorFlow

Ponownie użyjemy TensorFlow/Keras do zbudowania kilku przykładów uczenia maszynowego i przyjrzenia się ich wynikom. Więcej informacji na temat TensorFlow i Keras można znaleźć w części 2. W tym przypadku stosujemy to samo pięcioetapowe podejście, które zastosowaliśmy do budowy sieci warstwowych za pomocą Keras w rozdziale 2.

1. Załaduj i sformatuj swoje dane.

2. Zdefiniuj model i warstwy swojej sieci neuronowej.

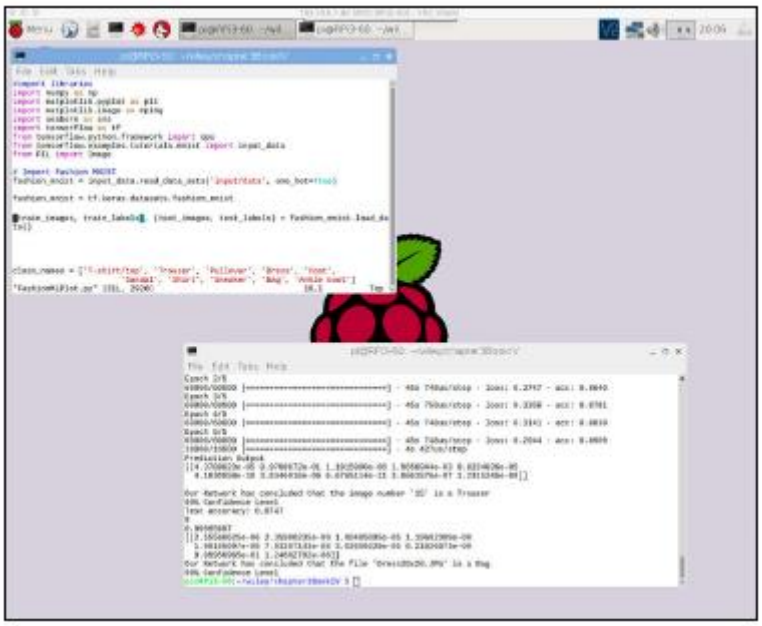
3. Skompiluj model.

4. Dopasuj i wytrenuj swojego modela.

5. Oceń model.

Konfigurowanie środowiska oprogramowania dla tej Części

Większość czynności opisanych tutaj odbywa się, jak zwykle, w wierszu poleceń, ponieważ nadal trzeba wpisać kod i uruchomić oprogramowanie. Jednak zamierzamy wyświetlić trochę grafiki na ekranie i użyć MatPlotLib do oceny działania twojego programu uczenia maszynowego, więc uruchom GUI (graficzny interfejs użytkownika), jeśli jeszcze tego nie zrobiłeś. Jeśli korzystasz z bezgłowego Raspberry Pi, dodaj klawiaturę, mysz i monitor lub zatrzymaj się teraz i uruchom VNC (wirtualny komputer sieciowy). pomyśl o użyciu monitora komputera jako wyświetlacza na drugim komputerze - w tym przypadku Raspberry Pi. Wiele linków w Internecie opisuje, jak to zrobić i jak wywołać GUI na głównym komputerze. W tym rozdziale używamy VNC na bezgłowym Raspberry Pi. Jeśli chcesz, możesz podłączyć mysz, klawiaturę i monitor bezpośrednio do Raspberry Pi. Rysunek pokazuje GUI działające na Raspberry Pi (w rzeczywistości działa na VNC na naszym Macu, ale nie widać tego na tym obrazku).



Świetne źródło samouczków dotyczących konfigurowania oprogramowania i podłączania Raspberry Pi znajduje się na stronie www.raspberrypi.org. Jeśli brakuje niektórych bibliotek, których używamy w tym przykładzie, wyszukaj w Internecie, jak zainstalować je na konkretnym komputerze. Każda konfiguracja jest trochę inna. Na przykład, jeśli brakuje ci seaborn, wyszukaj „instalowanie biblioteki python seaborn na [nazwa twojej maszyny]". Jeśli wyszukujesz Raspberry Pi pod hasłem seaborn, znajdziesz „sudo pip3 install seaborn".

Tworzenie sieci uczenia maszynowego do wykrywania typów ubrań

Nasz główny przykład uczenia maszynowego w Pythonie wykorzystuje format MNIST (MNIST oznacza, że jest to zbiór obrazów w skali szarości o rozdzielczości 28x28 pikseli).

0 T-shirt/top

1 spodnie

2 pulowery

3 Sukienka

4 Płaszcz

5 Sandał

6 Koszula

7 Trampki

8 Torba

9 But do kostki

Pobieranie danych -zestaw danych Fashion-MNIST

Okazuje się, że jest to całkiem proste, chociaż pierwsze załadowanie go na komputer zajmie trochę czasu. Po uruchomieniu programu po raz pierwszy użyje on danych Fashion-MNIST skopiowanych na Twój komputer.

Trenowanie sieci

Będziemy trenować naszą sieć neuronową uczącą się maszynowo, używając wszystkich 60 000 obrazów ubrań: 6000 obrazów w każdej z dziesięciu kategorii. Testowanie naszej sieci Nasza przeszkolona sieć zostanie przetestowana na trzy różne sposoby: 1) zestaw 10 000 zdjęć szkoleniowych ze zbioru danych Fashion_MNIST; 2) wybrany obraz ze zbioru danych Fashion_MNIST; oraz 3) zdjęcie kobiecej sukni. Ta pierwsza wersja programu przeprowadzi test na zestawie 10 000 plików z bazy danych Fashion_MNIST. Poniżej znajduje się nasz kod Pythona wykorzystujący TensorFlow, NumPy i Keras dla sieci Fashion_MNIST. Używając nano (lub swojego ulubionego edytora tekstu), otwórz plik o nazwie MTensorFlow.py i wprowadź następujący 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

# Import Fashion MNIST

fashion_mnist = input_data.read_data_sets('input/data',

one_hot=True)

fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) \

= fashion_mnist.load_data()

class_names = ['T-shirt/top', 'Trouser',

'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

train_images = train_images / 255.0

test_images = test_images / 255.0

model = tf.keras.Sequential()

model.add(tf.keras.layers.Flatten(input_shape=(28,28)))

model.add(tf.keras.layers.Dense(128, activation='relu' ))

model.add(tf.keras.layers.Dense(10, activation='softmax' ))

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

loss='sparse_categorical_crossentropy',

metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=5)

test with 10,000 images

test_loss, test_acc = model.evaluate(test_images, test_labels)

print('10,000 image Test accuracy:', test_acc)

Łamanie kodu

Po przeczytaniu opisu programu Tensform/Keras w części 2 ten kod powinien wyglądać o wiele bardziej znajomo. W tej sekcji podzielimy to na nasz pięcioetapowy proces Keras. Najpierw importujemy wszystkie biblioteki potrzebne do uruchomienia naszego przykładowego modelu dwuwarstwowego. Zauważ, że TensorFlow domyślnie zawiera Keras. I po raz kolejny widzimy naszego przyjaciela NumPy jako preferowany sposób obsługi macierzy.

#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

1. Załaduj i sformatuj swoje dane. Tym razem korzystamy z wbudowanej możliwości odczytu zestawu danych. Wie, czym są te dane, dzięki instrukcji import z tensorflow.examples.tutorials.mnist w poprzednim kodzie.

Import Fashion MNIST

fashion_mnist = input_data.read_data_sets('input/data',

one_hot=True)

fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) \

= fashion_mnist.load_data()

Tutaj nadajemy opisowe nazwy dziesięciu klasom w danych Fashion_MNIST.

class_names = ['T-shirt/top', 'Trouser',

'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

Tutaj zmieniamy wszystkie obrazy, aby były skalowane od 0,0 do 1,0, a nie od 0 do 255.

train_images = train_images / 255.0

test_images = test_images / 255.0

2. Zdefiniuj model i warstwy sieci neuronowej. Ponownie, tutaj świeci prawdziwa moc Keras. Bardzo łatwo jest dodać więcej warstw neuronowych oraz zmienić ich rozmiary i funkcje aktywacji. Stosujemy również obciążenie do naszej funkcji aktywacji (relu), w tym przypadku za pomocą softmax, dla końcowej warstwy wyjściowej.

model = tf.keras.Sequential()

model.add(tf.keras.layers.Flatten(input_shape=(28,28)))

model.add(tf.keras.layers.Dense(128, activation='relu'))

model.add(tf.keras.layers.Dense(10, activation='softmax'))

3. Skompiluj swój model. Używamy funkcji straty sparse_categorical_crossentropy. Ta funkcja jest dla nas nowością w tej książce. Używa się go, gdy każdej kategorii ubrań przypisano inną liczbę całkowitą, tak jak w tym przykładzie. ADAM (metoda dla optymalizacja stochastyczna) jest dobrym domyślnym optymalizatorem. Zapewnia metodę dobrze dopasowaną do problemów, które są duże pod względem danych i/lub parametrów.

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

loss='sparse_categorical_

crossentropy',

metrics=['accuracy'])

Rzadka kategoryczna krzyżentropia to funkcja straty używana do pomiaru błędu między kategoriami w zbiorze danych. Kategoryczny odnosi się do faktu, że dane mają więcej niż dwie kategorie (binarne) w zbiorze danych. Sparse odnosi się do używania pojedynczej liczby całkowitej do odwoływania się do klas (w naszym przykładzie 0–9). Entropia (miara nieporządku) odnosi się do mieszanki danych między kategoriami.

4. Dopasuj i wytrenuj swojego modela.

Wybrałem liczbę epok jako tylko 5 ze względu na czas potrzebny na uruchomienie modelu dla naszych przykładów. Śmiało zwiększaj! Tutaj ładujemy tablice NumPy dla danych wejściowych do naszej sieci (baza danych train_images).

model.fit(train_images, train_labels, epochs=5)

5. Oceń model.

Funkcja model.evaluate służy do porównywania danych wyjściowych wytrenowanej sieci w każdej epoce i generuje test_acc i test_loss dla twoich informacji w każdej epoce przechowywanych w zmiennej historii.

test with 10,000 images

test_loss, test_acc = model.evaluate(test_images,

test_labels)

print('10,000 image Test accuracy:', test_acc)

Wyniki szkolenia i ewaluacja

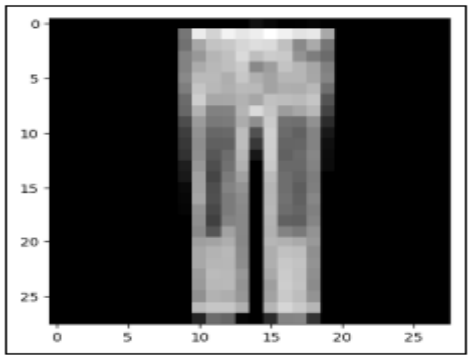
Uruchomiłem swój program na Raspberry Pi 3B. W tym momencie możesz bezpiecznie zignorować ostrzeżenia o niezgodności kodu i przyszłe ogłoszenia o wycofaniu. Oto wyniki programu:

```
Epoch 1/5
60000/60000 [==============================] - 44s 726us/step - loss: 0.5000 -
   acc: 0.8244
Epoch 2/5
60000/60000 [==============================] - 42s 703us/step - loss: 0.3751 -
   acc: 0.8652
Epoch 3/5
60000/60000 [==============================] - 42s 703us/step - loss: 0.3350 -
   acc: 0.8767
Epoch 4/5
60000/60000 [==============================] - 42s 701us/step - loss: 0.3124 -
   acc: 0.8839
Epoch 5/5
60000/60000 [==============================] - 42s 703us/step - loss: 0.2960 -
   acc: 0.8915
10000/10000 [==============================] - 4s 404us/step
10,000 image Test accuracy: 0.878
```

Zasadniczo wyniki testów mówią, że dzięki naszej dwuwarstwowej neuronowej sieci uczenia maszynowego prawidłowo klasyfikujemy 87 procent testowej bazy danych zawierającej 10 000 obrazów. Zwiększyliśmy liczbę epok do 50 i zwiększyliśmy ją do zaledwie 88,7 procent dokładności. Dużo dodatkowych obliczeń przy niewielkim wzroście dokładności.

Testowanie pojedynczego obrazu testowego

Następnie należy przetestować pojedynczy obraz z bazy danych Fashion_MNIST.



Dodaj ten kod na końcu swojego programu i ponownie uruchom oprogramowanie:

Prediction Output

[[1.2835168e-05 9.9964070e-01 6.2637120e-08 3.4126092e-04 4.4297972e-06

7.8450663e-10 6.2759432e-07 9.8717527e-12 1.2729484e-08 1.1002166e-09]]

Our Network has concluded that the image number '15' is a Trouser

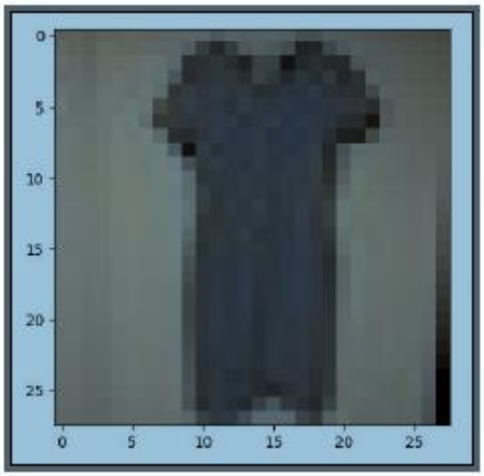
99% Confidence Level

Woohoo! Zadziałało. Prawidłowo zidentyfikował zdjęcie jako spodnie. Pamiętaj jednak, że ogólny poziom dokładności danych testowych wynosił tylko 87 procent. Następnie przejdź do naszego własnego zdjęcia.

Testowanie na zdjęciach zewnętrznych

Aby wykonać ten test, John wyjął sukienkę z szafy swojej żony, powiesił ją na ścianie

i zrobił jej zdjęcie swoim iPhonem. Następnie, korzystając z Podglądu na naszym Macu, przekonwertowaliśmy go do rozdzielczości 28 x 28 pikseli w dół z 3024 x 3024 pikseli prosto z telefonu.



Dobra, kilka bardzo ważnych uwag. Po pierwsze 28x28 pikseli nie daje zbyt wyraźnego obrazu. Jednak porównując Rysunek 6 z Rysunkiem. 4 z bazy Fashion-MNIST, nasze zdjęcie i tak wygląda lepiej. Większość poniższego kodu dotyczy uporządkowania danych z naszego obrazu JPG do formatu wymaganego przez TensorFlow. Powinieneś być w stanie użyć tego kodu, aby łatwo dodawać własne zdjęcia do większej liczby eksperymentów:

uruchom Nasz obraz testowy

przeczytaj zdjęcie sukienki testowej

```
imageName = "Sukienka 28x28.JPG"

testImg = Image.open(imageName)

testImg.load()

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

dane = tf.image.rgb_to_grayscale(dane)

dane = dane/255.0

dane = tf.transpose(dane, pozwolenie=[2,0,1])

singlePrediction = model.predict(dane,kroki=1)

print("Wyjście prognozy")

print(singlePrediction)

wydrukować()

NumberElement = singlePrediction.argmax()

Element = np.amax(singlePrediction)

print("Nasza sieć stwierdziła, że plik '"

+ imageName „” to „ class_names[NumberElement])

print (str(int(Element*100)) "% poziom ufności")
```

Wyniki, runda 1

Powinniśmy zacząć od stwierdzenia, że te wyniki nie bardzo nas uszczęśliwiły, jak wkrótce się przekonacie. Umieściliśmy plik Dress28x28.JPG w tym samym katalogu, w którym znajduje się nasz program, i przeprowadziliśmy szkolenie z całej epoki. Oto wyniki:

Prediction Output

[[1.2717753e-06 1.3373902e-08 1.0487850e-06 3.3525557e-11 8.8031484e-09

7.1847245e-10 1.1177938e-04 8.8322977e-12 9.9988592e-01 3.2957085e-12]]

Our Network has concluded that the file 'Dress28x28.JPG' is a Bag

99% Confidence Level

Tak więc nasz program uczenia maszynowego sieci neuronowej, po sklasyfikowaniu 60 000 zdjęć i 6000 zdjęć sukienek, zakończył się z 99-procentowym poziomem pewności. . . czekaj na to . . . że suknia żony Johna to torba. Tak więc pierwszą rzeczą, jaką zrobiliśmy później, było zwiększenie epok treningowych do 50 i ponowne uruchomienie programu. Oto wyniki z tego biegu:

Prediction Output

[[3.4407502e-33 0.0000000e 00 2.5598763e-33 0.0000000e 00 0.0000000e 00

0.0000000e 00 2.9322060e-17 0.0000000e 00 1.0000000e 00 1.5202169e-39]]

Our Network has concluded that the file 'Dress28x28.JPG' is a Bag

100% Confidence Level

Sukienka to wciąż torba, ale teraz nasz program jest w 100% pewny, że sukienka to torba. Hmmm. Ilustruje to jeden z problemów związanych z uczeniem maszynowym. Będąc w 100 procentach pewnym, że zdjęcie przedstawia torbę, gdy jest to sukienka, nadal jest w 100 procentach błędne. W czym tkwi prawdziwy problem? Prawdopodobnie konfiguracja sieci neuronowej nie jest wystarczająco dobra, aby odróżnić sukienkę od torby. Widzieliśmy, że dodatkowe okresy treningowe wcale nie pomogły, więc następną rzeczą do spróbowania jest zwiększenie liczby neuronów na naszym ukrytym poziomie. Jakie są inne rzeczy, które można spróbować poprawić? Okazuje się, że jest ich wiele. Możesz użyć CNN (konwolucyjnych sieci neuronowych), rozszerzania danych (zwiększanie liczby próbek szkoleniowych poprzez obracanie, przesuwanie i powiększanie tych obrazów) oraz wielu innych technik, które wykraczają poza zakres tego wprowadzenia do uczenia maszynowego. Zrobiliśmy jeszcze jeden eksperyment. Zmieniliśmy warstwy modelu w naszym programie, aby używać następującego czteropoziomowego modelu warstw konwolucyjnych. Po prostu uwielbiamy to, jak łatwo Keras i TensorFlow radykalnie zmieniają sieć neuronową. Sieci CNN skanują obrazy i analizują je kawałek po kawałku, powiedzmy w oknie 5x5, które za każdym razem przesuwa się o krok o dwa piksele, aż obejmie całą wiadomość. To jak patrzenie na obraz za pomocą mikroskopu; w danym momencie widzisz tylko małą część obrazu, ale w końcu widzisz cały obraz. Przejście do sieci CNN na moim Raspberry Pi zwiększyło czas pojedynczej epoki do 1,5 godziny z poprzedniej epoki 10 sekund.

Kod modelu CNN

Ten kod ma taką samą strukturę jak ostatni program. Jedyną istotną zmianą jest dodanie nowych warstw dla sieci CNN:

```
#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

# Import Fashion MNIST

fashion_mnist = input_data.read_data_sets('input/data',

one_hot=True)

fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) \

= fashion_mnist.load_data()

class_names = ['T-shirt/top', 'Trouser',
```

```
'Pullover', 'Dress', 'Coat',
'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
train_images = train_images / 255.0
test_images = test_images / 255.0
# Prepare the training images
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1)
# Prepare the test images
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1)
model = tf.keras.Sequential()
input_shape = (28, 28, 1)
model.add(tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=input_shape))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.Dropout(0.25))
model.add(tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.25))
model.add(tf.keras.layers.Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.Dropout(0.25))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.5))
```

```
model.add(tf.keras.layers.Dense(10, activation='softmax'))

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

loss='sparse_categorical_crossentropy',

metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=5)

# test with 10,000 images

test_loss, test_acc = model.evaluate(test_images, test_labels)

print('10,000 image Test accuracy:', test_acc)

#run test image from Fashion_MNIST data

img = test_images[15]

img = (np.expand_dims(img,0))

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

print ("Prediction Output")

print(singlePrediction)

print()

NumberElement = singlePrediction.argmax()

Element = np.amax(singlePrediction)

print ("Our Network has concluded that the image number '15' is a "

+ class_names[NumberElement])

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

Wyniki, runda 2

To uruchomienie na Raspberry Pi 3B zajęło około siedmiu godzin. Wyniki były następujące:

```
10,000 image Test accuracy: 0.8601

Prediction Output

[[5.9128129e-06 9.9997270e-01 1.5681641e-06 8.1393973e-06 1.5611777e-06

7.0504888e-07 5.5174642e-06 2.2484977e-07 3.0045830e-06 5.6888598e-07]]

Our Network has concluded that the image number '15' is a Trouser
```

Kluczową liczbą jest tutaj dokładność testu 10 000 obrazów. Wynosząc 86 procent, był w rzeczywistości niższy niż nasza poprzednia, prostsza sieć neuronowa wykorzystująca uczenie maszynowe (87 procent). Dlaczego się to stało? Prawdopodobnie jest to przypadek związany z „nadmiarem” danych treningowych. Model CNN taki jak ten może wykorzystywać do uczenia złożone modele wewnętrzne (wiele milionów możliwości) i może prowadzić do overfttingu, co oznacza, że wyuczona sieć lepiej rozpoznaje zbiór uczący, ale traci zdolność rozpoznawania nowych danych testowych. Wybór sieci

neuronowej uczenia maszynowego do pracy z danymi jest jedną z głównych decyzji, które podejmiesz w swoim projekcie. Jednak zrozumienie funkcji aktywacji, zarządzania przerwami i funkcji utraty również głęboko wpłynie na wydajność Twojego programu uczenia maszynowego. Optymalizacja wszystkich tych parametrów naraz to trudne zadanie, które wymaga badań i doświadczenia. Niektóre z nich to naprawdę nauka o rakietach!

Wizualizacja za pomocą MatPlotLib

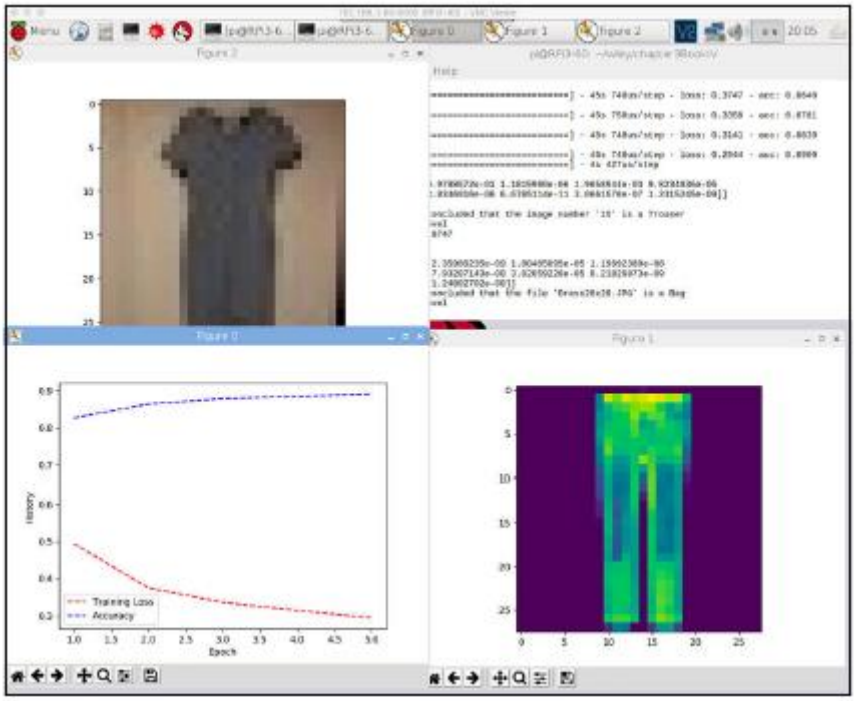
Teraz, gdy przenieśliśmy się do środowiska programistycznego opartego na graficznym interfejsie użytkownika, ponownie uruchomimy nasz kod podstawowy i przeprowadzimy analizę przebiegu przy użyciu MatPlotLib. Do tych eksperymentów używamy Raspberry Pi, ale możesz użyć komputera Mac, PC lub innego systemu Linux i zasadniczo zrobić to samo. Jeśli możesz zainstalować TensorFlow, MatPlotLib i Python w swoim systemie komputerowym, możesz przeprowadzić te eksperymenty. Aby zainstalować MatPlotLib na swoim Raspberry Pi, wpisz pip3 install matplotlib. Dodajemy zmienną historii do danych wyjściowych modelu.fit w celu zebrania danych. Następnie dodajemy polecenia MatPlotLib do tworzenia wykresów strat i dokładności z naszych epok, a następnie dodajemy wykresy dla naszych dwóch indywidualnych testów obrazu. Rysunek przedstawia wyniki działania tego programu. Używając nano (lub swojego ulubionego edytora tekstu), otwórz plik o nazwie FMTensorFlowPlot. py i wprowadź następujący 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

# Import Fashion MNIST

fashion_mnist = input_data.read_data_sets('input/data', one_hot=True)

fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.

load_data()

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

train_images = train_images / 255.0

test_images = test_images / 255.0

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

```
model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
model.add(tf.keras.layers.Dense(128, activation='relu' ))
model.add(tf.keras.layers.Dense(10, activation='softmax' ))
model.compile(optimizer=tf.train.AdamOptimizer(),
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=2)
# 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.show(block=False);
plt.pause(0.001)
test_loss, test_acc = model.evaluate(test_images, test_labels)
#run test image from Fashion_MNIST data
img = test_images[15]
plt.figure(1)
plt.imshow(img)
plt.show(block=False)
plt.pause(0.001)
img = (np.expand_dims(img,0))
singlePrediction = model.predict(img,steps=1)
print ("Prediction Output")
```

```
print(singlePrediction)
print()
NumberElement = singlePrediction.argmax()
Element = np.amax(singlePrediction)
print ("Our Network has concluded that the image number '15' is a "
class_names[NumberElement])
print (str(int(Element*100)) + "% Confidence Level")
print('Test accuracy:', test_acc)
# read test dress image
imageName = "Dress28x28.JPG"
testImg = Image.open(imageName)
plt.figure(2)
plt.imshow(testImg)
plt.show(block=False)
plt.pause(0.001)
testImg.load()
data = np.asarray( testImg, dtype="float" )
data = tf.image.rgb_to_grayscale(data)
data = data/255.0
data = tf.transpose(data, perm=[2,0,1])
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")
plt.show()
```

Wyniki działania tego programu przedstawiono na rysunku



Okno oznaczone jako Rysunek 0 pokazuje dane dotyczące dokładności dla każdej z pięciu epok szkolenia uczenia maszynowego i widać, że dokładność powoli rośnie z każdą epoką. Okno oznaczone na rysunku 1 pokazuje zdjęcie testowe użyte do pierwszego testu rozpoznawania (znalazło parę spodni, co jest prawidłowe), a na koniec okno oznaczone na rysunku 2 pokazuje zdjęcie sukienki, które nadal jest błędnie zidentyfikowane jako torba. Harumf.