Sprawozdanie z projektu

Wiktor Płużek

Spis treści

[2 Cel projektu 1](#_Toc106735035)

[2.1 Wybrany problem 1](#_Toc106735036)

[3 Przygotowanie danych 2](#_Toc106735037)

[3.1 Wybór zbiorów danych 2](#_Toc106735038)

[3.2 Różnice w strukturze zapisu zbiorów 3](#_Toc106735039)

[4 Implementacja sieci neuronowych 3](#_Toc106735040)

[4.1 Przypisywanie typów pokemona po nazwie pliku 3](#_Toc106735041)

[4.2 Tworzenie zbiorów treningowych i testowych 4](#_Toc106735042)

[4.3 Tworzenie warstw sieci 4](#_Toc106735043)

[5 Podsumowanie 7](#_Toc106735044)

# Cel projektu

Celem projektu jest bliższe poznanie działania uczenia maszynowego, a dokładniej sieci neuronowych, w celu rozwiązywania problemów związanych z klasyfikacją, regresją czy analizą.

## Wybrany problem

Wybranym przeze mnie problemem jest przetwarzanie obrazów, a dokładniej klasyfikacja typu pokemona na podstawie jego zdjęcia.

Dla przykładowego pokemona Beedrill o typach Robak, Trujący, ze zdjęciem:

Obraz zawierający obiekt na zewnątrz

Opis wygenerowany automatycznie

Program powinien poprawnie sklasyfikować typ pokemona jako jeden z dwóch: Robak lub Trujący.

# Przygotowanie danych

## Wybór zbiorów danych

Najważniejszą rzeczą potrzebną do dobrej klasyfikacji jest dobry zbiór danych do nauki modelu. W tym celu pobrałem ze strony ze zbiorami danych dwa zbiory zdjęć pokemonów:

* Ponad 2500 zdjęć, około 5-6 zdjęć dla każdego pokemona:

<https://www.kaggle.com/datasets/hlrhegemony/pokemon-image-dataset>

* Kolejne 809 zdjęć, po 1 dla każdego pokemona, jednak ten zbiór zawierał bardzo użyteczny zbiór jaki pokemon posiada które typy, dzięki czemu zaoszczędziłem dużo czasu na przepisywaniu tych informacji z wiki.

<https://www.kaggle.com/datasets/vishalsubbiah/pokemon-images-and-types?resource=download>

## Różnice w strukturze zapisu zbiorów

Niestety forma zapisu zdjęć nie była ze sobą kompatybilna, ponieważ pierwszą wersję programu zrobiłem na mniejszym zbiorze, w którym zdjęcia miały nazwę nazwa\_pokemona.png, tak też zaimplementowałem w programie odnajdywanie poprawnych typów – po nazwie pokemona zawartej w nazwie zdjęcia.

Drugi zbiór – większy, był nieco inaczej zorganizowany, gdyż miał on strukturę Folder o nazwie pokemona / 0.jpg, 1.jpg … ,  
Musiałem więc napisać niewielki skrypt, który zamienił tę strukturę na taką, która odpowiadałaby mojej implementacji

import os  
  
for dir in os.listdir('./images'):  
 for img in os.listdir('./images/' + dir):  
 olddir = './images/' + dir + '/' + img  
 newdir = './images\_new/' + dir + '.' + img.split('.')[0] + '.' + img.split('.')[1]  
 os.rename(dir, newdir)

Skrypt ten zwyczajnie brał każde zdjęcie z folderu o nazwie pokemona i przenosił je ze zmienioną nazwą z 0.jpg -> nazwa\_pokemona.0.jpg do folderu ze wszystkimi zdjęciami pokemonów.

Dodatkowo, większy zbiór zawierał kilkadziesiąt pokemonów, które nie były zawarte w arkuszu danych z typami pokemonów, musiałem więc niestety ręcznie je tam dodawać.

# Implementacja sieci neuronowych

## Przypisywanie typów pokemona po nazwie pliku

Do nauki ważne jest, aby podczas treningu model wiedział, czy jego predykcja jest poprawna – w tym celu razem ze zdjęciem w zbiorze treningowym przesyłane są faktyczne typy pokemona, które są wyciągane z arkusza, w którym każdemu pokemonowi przypisane są jego typy

def label\_img(img):  
 word\_label = img.split('.')[0].lower()  
 label\_arr = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
 for i in pokemon\_dict[word\_label]:  
 for index in range(0, len(pokemon\_types)):  
 if pokemon\_types[index] == i:  
 label\_arr[index] = 1  
 return label\_arr

## Tworzenie zbiorów treningowych i testowych

Ze zbioru wszystkich 3350 zdjęć, 80% z nich użyłem do zbioru treningowego

def create\_train\_data():  
 training\_data = []  
 for img in tqdm(os.listdir(IMAGES)[:int(len(os.listdir((IMAGES))) \* 0.8)]):  
 label = label\_img(img)  
 path = os.path.join(IMAGES, img)  
 img = cv2.imread(path)  
 img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  
 img = cv2.resize(img, (IMG\_SIZE, IMG\_SIZE))  
 training\_data.append([np.array(img), np.array(label)])  
 shuffle(training\_data)  
 np.save('train\_data.npy', training\_data)  
 print("create\_train\_data done")  
 return training\_data

A 20% z nich do zbioru testowego

def process\_test\_data():  
 testing\_data = []  
 for img in tqdm(os.listdir(IMAGES)[int(len(os.listdir((IMAGES))) \* 0.8):]):  
 path = os.path.join(IMAGES, img)  
 label = label\_img(img)  
 img = cv2.imread(path)  
 img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  
 img = cv2.resize(img, (IMG\_SIZE, IMG\_SIZE))  
 testing\_data.append([np.array(img), np.array(label)])  
  
 shuffle(testing\_data)  
 np.save('test\_data.npy', testing\_data)  
 print("process\_test\_data done")  
 return testing\_data

## Tworzenie warstw sieci

Do implementacji sieci neuronowych postanowiłem wykorzystać bibliotekę tensorflow, mając nadzieję na późniejsze uruchomienie uczenia na karcie graficznej dla lepszych efektów.

Początkowo zastosowałem bardzo proste sieci – niewiele warstw oraz niewiele iteracji.

10 iteracji, ok. 3 min., skuteczność ~10%

convnet = input\_data(shape=[None, IMG\_SIZE, IMG\_SIZE, 3], name='input')  
  
convnet = conv\_2d(convnet, 32, 4, activation='relu')  
convnet = max\_pool\_2d(convnet, 4)  
  
convnet = conv\_2d(convnet, 64, 3, activation='relu')  
convnet = max\_pool\_2d(convnet, 3)  
  
convnet = fully\_connected(convnet, 8192, activation='relu')  
  
convnet = fully\_connected(convnet, 18, activation='softmax')  
convnet = regression(convnet, optimizer='adam', learning\_rate=LR, loss='categorical\_crossentropy', name='targets')

Następnie postanowiłem stworzyć bardziej złożone warstwy – więcej oraz złożone z większej ilości połączeń.

Ilość iteracji – 20, Ok. 10 min., skuteczność ~18%

convnet = input\_data(shape=[None, IMG\_SIZE, IMG\_SIZE, 3], name='input')  
  
convnet = conv\_2d(convnet, 32, 8, activation='relu')  
convnet = max\_pool\_2d(convnet, 8)  
  
convnet = conv\_2d(convnet, 64, 10, activation='relu')  
convnet = max\_pool\_2d(convnet, 10)  
  
convnet = conv\_2d(convnet, 128, 8, activation='relu')  
convnet = max\_pool\_2d(convnet, 8)  
  
convnet = conv\_2d(convnet, 256, 8, activation='relu')  
convnet = max\_pool\_2d(convnet, 8)  
  
convnet = conv\_2d(convnet, 128, 8, activation='relu')  
convnet = max\_pool\_2d(convnet, 8)  
  
convnet = conv\_2d(convnet, 64, 6, activation='relu')  
convnet = max\_pool\_2d(convnet, 6)  
  
convnet = conv\_2d(convnet, 32, 5, activation='relu')  
convnet = max\_pool\_2d(convnet, 5)  
  
convnet = fully\_connected(convnet, 1024, activation='relu')  
  
convnet = fully\_connected(convnet, 18, activation='softmax')  
convnet = regression(convnet, optimizer='adam', learning\_rate=LR, loss='categorical\_crossentropy', name='targets')

Metodą prób i błędów, zauważyłem jednak, że sieć osiągała najlepsze wyniki dla średnio skomplikowanej sieci z większą ilością iteracji.

skuteczność ~21%

convnet = input\_data(shape=[None, IMG\_SIZE, IMG\_SIZE, 3], name='input')  
  
convnet = conv\_2d(convnet, 32, 6, activation='relu')  
convnet = max\_pool\_2d(convnet, 6)  
  
convnet = conv\_2d(convnet, 64, 12, activation='relu')  
convnet = max\_pool\_2d(convnet, 12)  
  
convnet = conv\_2d(convnet, 64, 6, activation='relu')  
convnet = max\_pool\_2d(convnet, 6)  
  
convnet = conv\_2d(convnet, 32, 4, activation='relu')  
convnet = max\_pool\_2d(convnet, 4)  
  
convnet = fully\_connected(convnet, 4096, activation='relu')  
  
convnet = fully\_connected(convnet, 18, activation='softmax')  
convnet = regression(convnet, optimizer='adam', learning\_rate=LR, loss='categorical\_crossentropy', name='targets')

Niestety pomimo wielu prób, obejrzanych poradników i wpisów na stackoverflow, nie udało mi się uruchomić uczenia sieci na karcie graficznej, więc ograniczyłem się dla tej sieci jedynie do 40 iteracji – co trwało ok. 15 min.

Jednak okazało się, że po tej ilości iteracji, model został przetrenowany dla danych treningowych, więc próbowałem zmniejszać tę ilość aż osiągnąłem lepsze wartości – przy 25 iteracjach, co zajęło jedynie ok. 8 min., program osiągnął dotychczas najwyższe – 30% skuteczności.

Co ciekawe, czasami, bez żadnych zmian w sieci, model przypisywał każdemu pokemonowi ten sam typ, przez co nawet po 30 iteracjach zdobywał zaledwie ~7% skuteczności…

Uruchomienie modelu:

train = train\_data  
test = test\_data  
  
X = np.array([i[0] for i in train]).reshape(-1, IMG\_SIZE, IMG\_SIZE, 3)  
Y = [i[1] for i in train]  
  
test\_x = np.array([i[0] for i in test]).reshape(-1, IMG\_SIZE, IMG\_SIZE, 3)  
test\_y = [i[1] for i in test]  
  
model.fit({'input': X}, {'targets': Y}, n\_epoch=15, validation\_set=({'input': test\_x}, {'targets': test\_y}),  
 snapshot\_step=500, show\_metric=True, run\_id="pokemon")

Wyświetlenie wyników:

fig = plt.figure()  
  
sum = [0,0]  
for num, data in enumerate(test\_data[:81]):  
 actual\_type = data[1]  
 img\_data = data[0]  
  
 y = fig.add\_subplot(9, 9, num + 1)  
 orig = img\_data  
 data = img\_data.reshape(IMG\_SIZE, IMG\_SIZE, 3)  
 model\_out = model.predict([data])[0]  
 # 'Normal', 'Fire', 'Water', 'Grass', 'Flying', 'Fighting', 'Poison', 'Electric', 'Ground', 'Rock',  
 # 'Psychic', 'Ice', 'Bug', 'Ghost', 'Steel', 'Dragon', 'Dark', 'Fairy']  
  
 if np.max(model\_out) == model\_out[0]:  
 if actual\_type[0] == 1:  
 sum[0]+=1  
 str\_label = 'Normal'  
 elif np.max(model\_out) == model\_out[1]:  
 if actual\_type[1] == 1:  
 sum[0]+=1  
 str\_label = 'Fire'

…

elif np.max(model\_out) == model\_out[17]:  
 if actual\_type[17] == 1:  
 sum[0]+=1  
 str\_label = 'Fairy'  
 sum[1]+=1  
  
 y.imshow(orig)  
 plt.title(str\_label)  
 y.axes.get\_xaxis().set\_visible(False)  
 y.axes.get\_yaxis().set\_visible(False)  
  
print("Guessed % = " + str(sum[0]/sum[1]) + " Random guessing % = " + str(1/18))  
plt.show()

# Podsumowanie

Po wielu testach dla różnych kombinacji warstw konwolucyjnych i poolingowych, doszedłem do wniosku, że moja implementacja rozpoznawania typów pokemonów nie radziła sobie najlepiej – najlepszym osiągniętym przez nią wynikiem była celność zgadnięć na poziomie 30%, co faktycznie jest sporym przeskokiem ponad losowym strzelaniem, które dla obecnych 18 typów pokemonów wyniosłoby ok. 5,6%. Jednak osobiście uważam, że taka skuteczność nie jest niczym wybitnym, może gdybym posiadał większe zasoby obliczeniowe, lub lepszy zbiór danych do nauki, byłbym w stanie osiągnąć lepszy wynik.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Niemniej jednak, uważam że zastosowania uczenia maszynowego i sieci neuronowych mają ogromny potencjał w dzisiejszym świecie, gdzie potrzebna jest klasyfikacja i wykrywanie najróżniejszych obiektów przez maszyny, które mogą być odpowiedzialne m.in. za nasze bezpieczeństwo.