Projekt z Analizy Obrazów

Animal Detector

Patryk Radoń, Gabriel Chęć, Patryk Kubica

Akademia Górniczo-Hutnicza im. Stanisława Staszica
Wydział Fizyki i Informatyki Stosowanej
Fizyka Techniczna /Informatyka Stosowana
28 stycznia 2020

Spis treści

1. Uruchamianie programu

Program działa na następujących systemach:

- Ubuntu 16.04 lub nowszy (64-bit)

- macOS 10.12.6 (Sierra) lub nowszy (64-bit)

- Windows 7 lub nowszy (64-bit)

Aby uruchomić nasz program wymagane jest zainstalowanie/posiadanie **Pythona 3** (najlepiej do wersji 3.7 , na wersji 3.8 program jeszcze nie działa poprawnie) wraz z **pip 19.0 lub nowszą**.

Wymagane są także następujące biblioteki (najnowsze wersje):

- **matplotlib** (3.1.2)
- **numpy** (1.18.1)
- **PyQt5** (5.14.1)
- **PILLOW** (7.0.0)
- **pandas** (0.25.3)
- **seaborn** (0.9.0)
- **pathlib** (1.0.1)
- **ipython** (7.11.1)
- **tensorflow** (2.1.0)

Wszystkie biblioteki dostępne z poziomu: pip install "nazwa biblioteki".

Mając już pobrane wszystkie biblioteki należy przejść do folderu:
Image-Analysis-Project\gui

Uruchamiamy plik main.py

Stworzony przez nas program w obecnej wersji obsługuje tylko zdjęcia w formacie **.jpg / .jpeg**

2. Czym jest nasz projekt?

Nasz projekt polega na stworzeniu systemu klasyfikacji gatunku zwierząt znajdujących się na zdjęciach.

Wykorzystujemy w tym celu stworzoną przez nas konwolucyjną sieć neuronową (CNN). Przy jej użyciu tworzymy odpowiedni model, który następnie użyty jest do klasyfikacji zadanego przez użytkownika obrazu zwierzęcia. Zwierzę przypisane może być do jednego z 10 gatunków:

pies, kot, słoń, owca, pająk, wiewiórka, krowa, kura, koń, motyl.

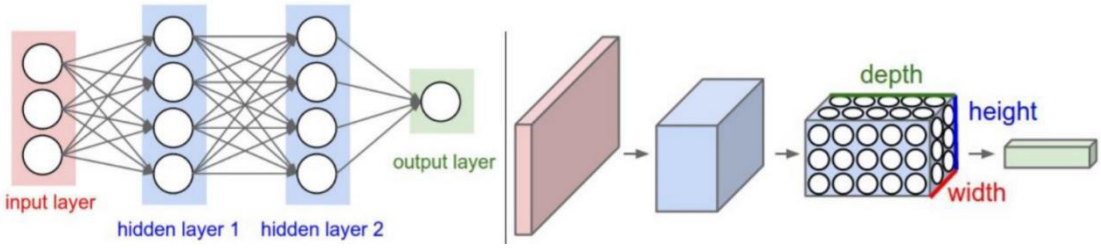
Program przedstawia także wykres, z jaką "pewnością" stworzony przez nas algorytm klasyfikuje znajdujące się na obrazie zwierzę do określonego gatunku.

3. Algorytmy, idea i co realizuje

Convolutional Neural Network

W projekcie wykorzystujemy głęboką sieć konwolucyjną (CNN) - potrafi ona stopniowo filtrować różne części danych uczących i wyostrzać ważne cechy w procesie dyskryminacji wykorzystanym do rozpoznawania lub klasyfikacji wzorców.

“Konwolucyjne sieci neuronowe (Convolutional Neural Network - CNN) składają się z jednej lub wielu warstw konwolucyjnych (typowych dla kroku próbkowania, określającego subwzorce), a następnie przez jedną lub w pełni połączone warstwy tak jak w klasycznej wielowarstwowej sieci, np. MLP, SVM, SoftMax itp. Głęboka sieć konwolucyjna zawiera wiele warstw. Sieci kowolucyjne są łatwe do uczenia, gdyż zawierają mniej parametrów (wykorzystując te same wagi) niż typowe sieci neuronowe z dokładnością do ilości warstw konwolucyjnych i ich rozmiaru. Ten rodzaj sieci neuronowych jest predestynowany do obliczeń na strukturach 2D (tj. obrazy).
Sieci konwolucyjne porządkują jednostki obliczeniowe („neurony”) w strukturze 3D: szerokość, wysokość i głębokość. Neurony w każdej warstwie są połączone do małego regionu warstwy poprzedniej zamiast do wszystkich, jak to jest w typowych sieciach neuronowych. Ponadto CNN redukuje pełne obrazy do pojedynczych wektorów wyjściowych reprezentujących wyniki dla poszczególnych klas, jak przedstawiono na poniższym rysunku.



[Rysunek 1] Porównanie typowych i głębokich konwolucyjnych architektur sieci [http://home.agh.edu.pl/~horzyk/lectures/ai/SztucznaInteligencja-UczenieG%C5%82%C4%99bokichSieciNeuronowych.pdf]

Sieć konwolucyjna jest zwykle sekwencją warstw, które transformują jeden obraz danych do drugiego poprzez funkcję różniczkowalną (w celu umożliwienia wykorzystania algorytmu propagacji wstecznej błędów do dostrojenia parametrów sieci neuronowej).” [6]

VGG-16 i VGG-19

W tej sieci stosowane są mniejsze filtry, ale sieć została zbudowana tak, aby była głębsza niż inne sieci CNN.

“Liczba 16 w nazwie VGG odnosi się do faktu, że ma ona 16 warstw, które mają pewne ciężary. Jest to dość duża sieć i ma w sumie około 138 milionów parametrów. Jednak prostota jej architektury sprawia, że jest całkiem atrakcyjna ze względu na swoją jednolitość. Istnieje kilka warstw konwekcyjnych, po których następuje warstwa “pooling”, która zmniejsza wysokość i szerokość. Jeśli spojrzymy na liczbę używanych przez nas filtrów możemy zobaczyć, że mamy 64 filtry, a następnie podwajamy je do 128, następnie do 256, aż w ostatnich warstwach używamy 512 filtrów. Liczba używanych filtrów podwaja sę na każdym kroku lub podwaja każdy stos warstw konwekcyjnych. Sieć neuronowa VGG-19 jest jeszcze większa niż VGG-16. Główną wadą jest fakt iż jest to dość duża sieć pod względem liczby parametrów do przeszkolenia.” [7]

Próbowaliśmy zastosować VGG-19 jednak po zaestymowaniu czasu trenowania na kilka dni uznaliśmy że nie dysponujemy odpowiednimi zasobami do stworzenia takiej sieci.

W projekcie używamy modelu CNN na wzór sieci typu VGG o lekko uproszczonej strukturze pozwalającej na jego trenowanie na dostępnych zasobach technologicznych oraz czasowych.

Projekt modelu został zrealizowany w oparciu o biblioteke TensorFlow 2.0, a także pomocniczo tensorflow.layers oraz tensorflow.keras.

Model_generator.ipynb (tworzenie i trenowanie sieci neuronowej - generowanie modelu)

Rozpoczynamy od stworzenia architektury sieci (używamy w tym celu gotowej architektury):

```
test_datagen = ImageDataGenerator(rescale=1./255,
                                  rotation_range=45,
                                  width_shift_range=.15,
                                  height_shift_range=.15,
                                  horizontal_flip=True,
                                  zoom_range=0.5
                                  )
train_datagen = ImageDataGenerator(rescale=1./255)
IMG_WIDTH,IMG_HEIGHT = (128,128)batch_size = 32
```

Kolejnym krokie było uczenie naszej sieci. Wykorzystaliśmy do tego już zebrane dane dostępne na stronie

|https://www.kaggle.com/alessiocorrado99/animals10?fbclid=IwAR3MnwUWmFrbZ9MoMF0iYBNbS719tJlq-9cTzZowArNVkxG4aRYxuOBn33k| [1]

“Wszystkie obrazy zostały zebrane z “obrazów Google” i zostały sprawdzone przez człowieka. Istnieje kilka błędnych danych symulujących rzeczywiste warunki.” [1]
Ze względu na dostępną moc obliczeniową nie używaliśmy wszystkich zdjęć dostępnych w podanej bazie. W naszym przypadku liczba obrazów dla każdej kategorii wahała się między 1 a 2,5 tysiąca.

```
train_generator = train_datagen.flow_from_directory(directory='./data/animals10/train', classes=['dog','cat','elephant','sheep','spider','sqirle','cow','chicken','horse','butterfly'], batch_size=batch_size, target_size=(IMG_WIDTH,IMG_HEIGHT))
```

Do trenowania wczytujemy 14181 obrazów należących do 10 klas. (wszystkie obrazy są przeskalowane do tego samego rozmiaru)

```
test_generator = train_datagen.flow_from_directory(directory='./data/animals10/valid', classes=['dog','cat','elephant','sheep','spider','sqirle','cow','chicken','horse','butterfly'], batch_size=batch_size, target_size=(IMG_WIDTH,IMG_HEIGHT))
```

Do testowania wczytujemy 4564 obrazów należących do 10 klas. (wszystkie obrazy są przeskalowane do tego samego rozmiaru)

Następnym krokiem jest tworzenie przez nas naszego modelu

```
from tensorflow.keras import regularizers
```

```
model = Sequential(
    [Conv2D(50, 3, padding='same', activation='relu', input_shape=(IMG_WIDTH,IMG_HEIGHT,3)),
     MaxPool2D(),
     Dropout(0.2),
     Conv2D(32, 3, padding='same', activation='relu'),
     MaxPool2D(),
     Dropout(0.2),
     Conv2D(32, 3, padding='same', activation='relu'),
     MaxPool2D(),
     Dropout(0.2),
     Conv2D(16, 3, padding='same', activation='relu'),
     MaxPool2D(),
     Dropout(0.2),
     Flatten(),
     Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
     Dropout(0.2),
     Dense(10, activation='softmax')
    ])#new_model_1
```

Powyższy fragment kodu zwraca nam listę skalarów.

Jak wspomnieliśmy wcześniej, liczba zdjęć zwierząt dla każdej kategorii nie była taka sama. Zdecydowanie najwięcej zdjęć znajduje się w kategoriach psy. Np dla kategorii koty i słonie użytych zdjęć jest mniej. Z powodu mniejszej ilości zdjęć w niektórych kategoriach, zastosowaliśmy 20% Dropout w celach regularyzacji (co warstwe `Dropout(0.2),`)

Następnie konfigurujemy model trenowania.

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Optimizer - nazwa optymalizatora (wybieramy jedną z wbudowanych klas optymalizatora)[3]. W naszym przypadku używamy optymalizatora implementującego algorytm Adama. “Optymalizacja Adama jest metodą stochastycznego spadku gradientu, która opiera się na adaptacyjnej ocenie momentów pierwszego i drugiego rzędu. Według artykułu Adam: A Method for Stochastic Optimization. Kingma et al., 2014 metoda ta jest wydajna obliczeniowo, ma małe wymagania pamięciowe, niezmienne w stosunku do przekątnej zmiany skali gradientów i dobrze nadaje się do problemów, które są duże pod względem danych/parametrów” [4]

Loss - w naszym przypadku oblicza utrate entropii krzyżowej między etykietami i prognozami. Zwraca instancje strat[3]

Metrics - lista wskaźników do oceny przez model podczas szkolenia i testów. Używamy najczęsciej używanego wskaźnika ‘accuracy’ [3]

Tworzymy także scieżkę zapisu naszego modelu:

```
keras_model_path = "./new_model_1"
```

```
model= tf.keras.models.load_model(keras_model_path)
```

Następnie trenujemy nasz model dla określonej liczby “epok” (epochs - liczba epok trenowania modelu. Epoka jest iteracją wszystkich podanych danych - train_generator; validation_data - dane na podstawie których można ocenić stratę i wszelkie metryki modelu na końcu każdej epoki, w naszym przypadku - test_generator) i zapisujemy model.:

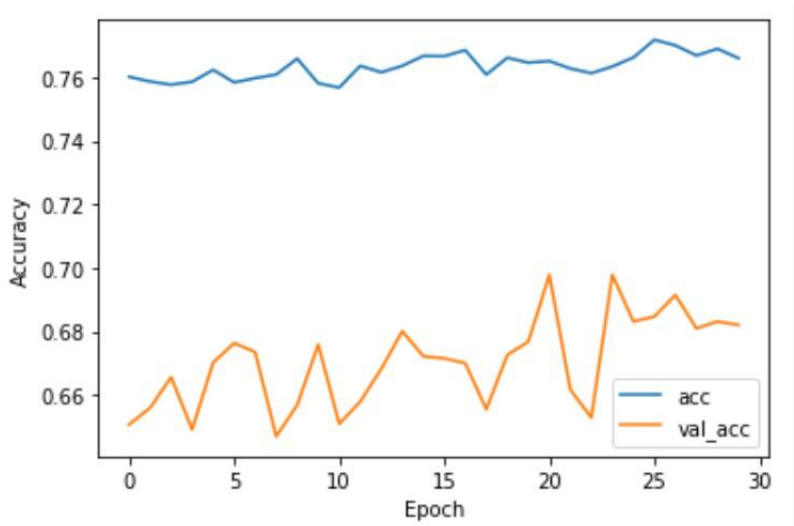
Używamy tutaj wcześniej wczytanych obrazów do trenowania (train_generator) oraz testowania modelu

```
hist = model.fit_generator(train_generator, epochs=30,
                           validation_data=test_generator)model.save(keras_model_path)
model.save(keras_model_path)
```

Powyższy kod zwraca obiekt typu Hist. Jego atrybut Hist.history jest zapisem wartości strat treningowych i wartości metryk w kolejnych epokach, a także wartości utraty walidacji i wartości metryk walidacji.

Stworzyliśmy także wykres prezentujący jak zmienia się trafność wraz z kolejnymi epokami: (prezentuje to jak sieć uczy się wraz z kolejnymi iteracjami)

```
plt.plot(hist.history['acc'], label='acc')
plt.plot(hist.history['val_acc'], label = 'val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
```



Zmniejszenie learning rate lub zwiększenie batch-size mogłoby pomóc w pozbyciu się oscylacji i pozwolić na dalszy trening.

Stworzoną przez nas sieć nazywamy - “new_model_1”.
W kolejnej części projektu jest ona przez nas używana jako model do klasyfikacji zwierząt

ImgPredictor.py (Użycie stworzonego modelu do klasyfikacji zwierząt)

Program odpowiada za przypisanie zwierzęcia znajdującego się na zdjęciu do określonego gatunku.

Po zaimportowaniu odpowiednich bibliotek wczytujemy w pliku ImgPredictor.py nasz model (sieć neuronową). Używamy do tego biblioteki tensorflow.keras.models

```
(from tensorflow.keras.models import load_model) :

loaded_model = load_model('./../new_model_1')
```

Następnie wczytujemy zadany przez użytkownika obraz i zmieniamy jego wielkość na określony przez nas rozmiar (128x128).

```
def load_image(file_name):
    img = Image.open(file_name)
    img.load()
    img = img.resize((IMG_WIDTH, IMG_HEIGHT))
    data = np.asarray(img, dtype="int32")
    data = data.reshape(1, IMG_WIDTH, IMG_HEIGHT, 3)
    data = tf.cast(data, tf.float32)
    data /= 255

    return data
```

Kolejnym krokiem jest użycie stworzonej przez nas sieci do klasyfikacji zawartości zdjęcia do danej klasy (gatunku):

```
img = load_image(PATH)
predictions = loaded_model.predict_on_batch(img)
```

Wynik tej operacji program zwraca jako listę z prawdopodobieństwem dopasowania obrazu do danej klasy.

Następnie tworzymy histogram zawierający informacje o tym do jakiego gatunku i w jakim stopniu zostało przypisane zwierzę znajdujące się na obrazie.

```
plt.tight_layout()
figsize = (17, 7)
fig, axes = plt.subplots(1, 2, figsize=figsize)

axes[0].imshow(Image.open(PATH))
classes = ['dog', 'cat', 'elephant', 'sheep', 'spider', 'sqirle', 'cow', 'chicken',
'horse', 'butterfly']

frame = pd.DataFrame(data=[list(predictions[0])], columns=classes)
sns.barplot(x=classes, y=predictions[0], ax=axes[1])
buf = io.BytesIO()
fig.savefig(buf)
buf.seek(0)
return ImageQt(Image.open(buf))
```

Efektem użycia tego programu (tej części projektu) jest uzyskanie wspomnianego wyżej histogramu zawierającego informacje o tym do jakiego gatunku i w jakim stopniu zostało przypisane zwierzę znajdujące się na obrazie.

mainView.py (GUI)

Część projektu odpowiadająca za interfesj graficzny. Wszystkie używane tutaj elementy (widget-y) zostały stworzone przy użyciu biblioteki PyQt5.

Program zaraz po uruchomieniu daje tylko możliwośc wybrania obrazu (w formacie .jpg) przez użytkownika, który zostanie poddany klasyfikacji.
W przypadku wybrania pliku w innym formacie pojawi się komunikat o błędzie.

```
self.error_dialog = QtWidgets.QMessageBox()
self.error_dialog.setIcon(QtWidgets.QMessageBox.Critical)
self.error_dialog.setText("Error")
self.error_dialog.setInformativeText('File must be one of the following types: ' + str(self.types))
self.error_dialog.setWindowTitle("Error")
```

Po wybraniu zdjęcia we właściwym formacie zostanie ono wyświetlone (już w zmienionym przez nas w poprzednim pliku rozmiarze) po lewej stronie głównego okna.

Na środku wyświetlany jest wynik użycia stworzonego przez nas programu dla klasyfikacji gatunków (ImgPredictor.py).

```
from ImgPredictor import load_file
```

Wyświetlamy histogram prezentującego z jaką "pewnością" program klasyfikuje zwierzę z zadanego przez użytkownika obrazu do danego gatunku.

4. Błędy/ ograniczenia naszego programu

Nie udało nam się zaimplementować pełnej wersji VGG-16 ze względu na brak możliwości obliczeniowej. Zmuszeni byliśmy skrócić model do postaci reprezentacyjnej (skróconej). Przy [batch_size=32] sieć sieć okazała się wystarczająco skuteczna. (efektem próby trenowania VGG-19 jest stworzenie modelu new_model2, jest to jednak tylko początkowa próba trenowania, model ten jest nieskuteczny i nieużywany przez nas).

Sieć w testach gotowej aplikacji okazała się nieskuteczna w rozpoznawaniu czarnobiałych zdjęć. W przyszłości jednym z możliwych rozwiązań byłoby dodanie do zestawu danych treningowych kopii losowej części obecnych zdjęć w skali szarości.

Oczywistym ograniczeniem programu jest także ilość gatunków (klas) zwierząt do jakich program może dopasować zwierzę znajdujące się na obrazie. W przypadku obrazu zawierającego zdjęcie zwierzęcia z gatunku innego niż 10 przez nas zadanych program będzie działał poprawnie, ale sama sieć neuronowa przypisze zwierzę do kilku gatunków z podobnym procentem "pewności".

Program na zdjęciach przedstawiających wiele zwierząt tego samego gatunku z reguły zachowuje się poprawnie. W przypadku gdy zwierzęta znajdujące się na zdjęciu należą do różnych określonych przez nas gatunków, program przypisuje je do tego gatunku, którego przedstawicieli na obrazie jest więcej (przykł. 1 pies 4 owce, program rozpoznaje na obrazie tylko owce).
Decydującym czynnikiem w przypadku wielu gatunków jest także ile miejsca na obrazie zajmuje poszczególne zwierzę. Ważnym czynnikiem przy klasyfikacji jest także które cechy przypisane do danego gatunku są dla sieci lepiej widoczne. Problem pojawia się także w przypadku gdy zwierzęta z różnych gatunków "zlewają się" na obrazie w jeden obiekt. Przykładowo na zdjęciu psa z motylem klasyfikator gubi się i wskazuje że na obrazie jest kot. W celu uniknięcia tych problemów należy dostarczyć sieci więcej zdjęć do testowania, a także jeżeli jest taka możliwość podzielić zdjęcie na mniejsze fragmenty (oddzielić gatunki) i klasyfikować każdy osobno.

Problem pojawią się także przy klasyfikacji zdjęć przedstawiających wiewiórki. Program częściej przypisuje je jako psy lub koty. Wskazane jest więc kilkukrotne zwiększenie liczby danych zawierających zdjęcia wiewiórek. Umożliwi to lepszą naukę sieci w kwestii rozpoznawania tej grupy. Podobny błąd występuje przy motylach, które czasami rozpoznawane są jako pająki.

Dodatkowo w celu rozwinięcia programu można dodać opcję wczytywania innych formatów plików.

5. Podział obowiązków

Stworzenie sieci neuronowej, jej nauka oraz testowanie, znalezienie i przygotowanie obrazów do jej nauki - **Patryk Radoń**

GUI, testowanie działania sieci/programu - **Gabriel Chęć**

Dokumentacja, testowanie działania programu - **Patryk Kubica**

6. Podsumowanie

Stworzony przez nas projekt spełnia nasze założenia. Klasyfikator w większości przypadków działa prawidłowo. Czasem przypisuje właściwy gatunek, jednak z niską "pewnością". Przyczyną zaniżonej "pewności" w niektórych przypadkach mogą być "niestandardowe" warunki w jakich użytkownik wykonał zdjęcie. Sieć nie jest bowiem nauczona odczytywać i klasyfikować właściwie dane z takich zdjęć. Spowodowane jest to ograniczeniami czasowymi jak i technologicznymi, gdyż wskazane byłoby dostarczyć sieci więcej danych do nauki oraz testowani, co w konsekwencji wydłuży czas nauki sieci znacząco. Rozwiąże to jednak większość błędów/ograniczeń. Możemy jednak stwierdzić że wszystkie stworzone przez nas algorytmy działają poprawnie. Program działa w sposób zadowalający, klasyfikator wymaga jednak dalszej nauki.

7. Bibliografia

[1] https://www.kaggle.com/alessiocorrado99/animals10?fbclid=IwAR3MnwUWmFrbZ9MoMF0iYBNbS719tJlq-9cTzZowArNVkxG4aRYxuOBn33k
[2] https://www.tensorflow.org/
[3] https://www.tensorflow.org/api_docs/python/tf/keras/Model
[4] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam
[5] http://home.agh.edu.pl/~horzyk/lectures/ai/SztucznaInteligencja-UczenieG%C5%82%C4%99bokichSieciNeuronowych.pdf
[6] http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/
[7] http://datahacker.rs/deep-learning-vgg-16-vs-vgg-19/

W załączniku znajduje się też kilka zdjęć na których testowaliśmy poprawność działania programu.