Interfejsy Człowiek-Robot – Projekt

Rok akademicki 2019/2020
WARiE, AIR, SW, sem. 1

Temat: System wizyjny do automatycznej klasyfikacji owoców

Wykonali:
Michał Kliczkowski (132073)
Tomasz Jankowski (132062)

Spis treści

1 Wstęp

1.1 Założenia projektowe

Projekt zakłada wyszukanie, bądź opracowanie bazy danych obrazów owoców - zarówno świeżych, jak i zepsutych, a także opracowanie modelu sieci neuronowej, której zadaniem jest klasyfikacja obrazów pod względem rodzaju oraz jakości owoców na nich występujących. Podstawową możliwością, jaką daje aplikacja jest możliwość wybrania konkretnego obrazu zawierającego owoc, a następnie uzyskanie informacji o jego klasyfikacji oraz jej pewności. Projekt bazuje na języku *Python* oraz bibliotece *Keras* (przy użyciu *Tensorflow*).

1.2 Baza danych

Znaleziona baza zdjęć składała się z fotografii różnych owoców w różnych stanach. Każdy z owoców został uwieczniony pod różnymi kątami dzięki zastosowaniu silnika wolnoobrotowego. Po zapoznaniu się z nią obrazy podzielone zostały na kategorie: jabłka świeże i nieświeże, gruszki świeże i nieświeże, winogrona świeże i nieświeże, awokado dojrzałe i niedojrzałe oraz wiśnie. Owoce podzielone na podgrupy „świeże" i „nieświeże" zostały podzielone ręcznie, na podstawie występowania defektów, bądź ich braku. Wzięto pod uwagę uszkodzenia, obicia, dziury oraz defekty skórki. Z tak przygotowanej bazy wybrane zostało 6087 obrazów uczących oraz 1010 obrazów testowych.

1.3 Sieć neuronowa

Jako że biblioteka *TensorFlow* umożliwia tworzenie różnych rodzajów sieci, początkowo przetestowana została sieć o samych warstwach „płaskich", mająca 128 neuronów w jednej warstwie ukrytej. Jako, że znaleziona baza danych była wręcz wzorcowo przygotowana, nawet tak prosta sieć dawała zadowalające rezultaty. Jednak w związku z faktem, iż dla danych macierzowych, jakimi są obrazy, biblioteka przewiduje zastosowanie warstw splotowych, podjęta została decyzja o zastosowaniu właśnie ich. Poprawa skuteczności była momentalnie zauważalna. Po testach przeróżnych konfiguracji struktury sieci ostateczna jej wersja prezentuje się w sposób następujący:

1. Warstwa splotowa o 32 filtrach wyjściowych i rdzeniu przekształcenia 3x3.
2. Warstwa odpytująca (wybierająca najwyższą wartość z fragmentów macierzy o rozmiarach 2x2).
3. Warstwa splotowa o 64 filtrach wyjściowych i rdzeniu przekształcenia 3x3,
4. Warstwa odpytująca (taka jak poprzednia).
5. Warstwa splotowa o 64 filtrach wyjściowych i rdzeniu przekształcenia 3x3,
6. Warstwa płaska o 64 neuronach.
7. Warstwa wyjściowa, przyporządkowująca dany obraz do jednej z dziewięciu możliwych klas.

W celu ograniczenia zjawiska przeuczenia, przed pierwszą warstwą splotową oraz pierwszą warstwą płaską zastosowano warstwę *Dropout*, która „zabija" losowe neurony. Dodatkowo zastosowanym mechanizmem jest *Early stopping*, kończący proces uczenia w przypadku, gdy przez 3 kolejne epoki nie nastąpi spadek wartości funkcji kosztu (wartości błędu). Dla 10 epok uczenia sieć na zbiorze testowym uzyskała skuteczność 99,75%, a wartość funkcji kosztu wyniosła zaledwie 0,0093 dla ponad 6000 obrazów.

1.4 Wykorzystane biblioteki

W celu zrealizowania projektu wykorzystano poniżej opisane biblioteki języka *Python*.

Keras - wysoce rozwinięta biblioteka umożliwiająca sprawne opracowywanie oraz dostosowywanie sieci neuronowych. W tym przypadku bazuje na bibliotece *Tensorflow* jako jej „back-end".

NumPy - biblioteka pozwalająca na przechowywanie obrazów w dedykowanym dla tej biblioteki kontenerze - tablicy NumPy (ang. *NumPy Array*).

OpenCV - biblioteka wykorzystująca mechanizmy z biblioteki *NumPy*, posiada funkcje umożliwiające przeprowadzanie operacji morfologicznych w szerokim zakresie.

tqdm - biblioteka umożliwiająca śledzenia działania pętli w postaci pasków postępu.

os - biblioteka umożliwiająca dostęp do funkcjonalności systemowych (np. odczyt i zapis plików).

tabulate - biblioteka umożliwiająca wypisywanie informacji w interfejsie linii komend w postaci tabeli.

1.5 Instrukcja uruchomienia

W celu uruchomienia przygotowanych programów, należy zainstalować interpreter języka *Python* w wersji 3.x wraz z wymaganymi bibliotekami. Proces instalacji poszczególnych bibliotek należy wykonać w systemowym wierszu poleceń przy użyciu polecenia *pip*.

Listing 1: Instalacja bibliotek

```
pip install opencv-python
pip install numpy
pip install tensorflow-gpu
pip install tqdm
pip install tabulate
```

W przypadku braku danej biblioteki (może zdarzyć się, że brakuje w systemie jednej z domyślnych), należy przyjrzeć się komunikatowi wysyłanemu przez interpreter i zainstalować wskazaną bibliotekę.

W celu przygotowania bazy obrazów tak, aby mogły być akceptowane przez opracowany model sieci neuronowej, należało napisać dodatkowe programy realizujące operacje m.in. sortowania obrazów na katalogi, czy zmiany nazw tychże obrazów. Nie stanowią one jednak kluczowej roli w samym opisie sieci neuronowej i sposobie jej działania, zatem nie zostały one ujęte w opracowaniu. Istotne programy uruchamia się w następujący sposób:

Listing 2: Uruchamianie programów

```
python training_app.py  # proces nauczania nowego modelu sieci
python testing_app.py  # proces testowania wskazanego modelu sieci
python classify.py  # klasyfikacja obrazu
```

2 Opracowane oprogramowanie

2.1 Proces uczenia sieci

Poniżej przedstawiono najważniejsze fragmenty kodu źródłowego, odpowiadającego za opis architektury sieci neuronowej, opatrzone odpowiednimi objaśnieniami.

Listing 3: Wczytanie obrazów

```
# Lista kategorii (nazwy folderow)
CATEGORIES = ["bad_apple", "bad_grape", "bad_pear", "cherry", "good_apple", "good_avocado",
      "good_grape", "good_pear", "ripe_avocado"]

# Wczytaj wszystkie obrazy i zapisz do zmiennej tablicowej
for category in CATEGORIES:
    path = os.path.join(DATADIR, category)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path, img))
        break
    break
```

Powyższy kod odpowiada za zapisanie wszystkich obrazów z folderów zawartych w zmiennej tablicowej *CATEGORIES* do zmiennej *img_array* w postaci macierzy *NumPy*.

Listing 4: Stworzenie zmiennej przechowującej dane uczące

```
# Zmienna przechowujaca dane uczace
training_data = []

# Funkcja konwertujaca wczesniejsze dane tablicowe do tablicy danych uczacych
def create_training_data():
    for category in CATEGORIES:
        path = os.path.join(DATADIR, category)
        class_num = CATEGORIES.index(category)
        for img in tqdm(os.listdir(path)):
            try:
                img_array = cv2.imread(os.path.join(path, img))
                training_data.append([img_array, class_num])
            except Exception as e:
                pass

# Wywolanie funkcji
create_training_data()
```

Tworzona jest pusta zmienna tablicowa, do której następnie wprowadzane są parami poszczególne obrazy i odpowiadające im klasy (np. świeże jabłko).

Listing 5: Architektura sieci neuronowej

```
# Early Stopping - zatrzymaj uczenie po 3 epokach bez poprawy
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

# Deklaracja sieci spolotowej
model = models.Sequential()

# Dropout - warstwa zabijajaca losowe neurony (100x100, random seed: 3)
model.add(Dropout(0.1, input_shape=(100, 100, 3)))

# Warstwy - 3 konwulsyjne oraz 2 dyskretyzacyjne
model.add(layers.Conv2D(32, (3, 3), activation='relu'))  # input_shape=(100, 100, 3)
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Splaszczenie wejscia (konwersja z macierzy do wektora)
model.add(layers.Flatten())

# Ponownie warstwa Dropout - 10% szansy na zabicie
model.add(Dropout(0.1))

# Dwie gesto polaczone warstwy sieci neuronowej
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(9))
```

Kolejno: implementacja techniki *Early stopping* pod kątem spadku funkcji kosztu, zadeklarowanie typu sieci neuronowej (splotowa), zadeklarowanie kolejnych warstw sieci (*Dropout*, konwulsyjne, *MaxPooling2D*, płaska oraz wyjściowe).

Listing 6: Kompilacja i uczenie modelu sieci

```
# Komplilacja opracowanego modelu sieci z ponizsza konfiguracja (algorytm optymalizacyjny
    Adam)
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Wytrenuj skonfigurowany model
history = model.fit(X, y, validation_split=0.01, epochs=10, callbacks=[callback])
```

Powyższy listing przedstawia kod odpowiadający za kompilację modelu sieci neuronowej z odpowiednią konfiguracją: algorytm optymalizacyjny *Adam*, precyzja jako metryka sieci. Model ten następnie jest trenowany na wcześniej dobranych danych testowych przez 10 epok. Warto zwrócić tutaj uwagę na parametr *validation_split* na poziomie 1%, który oznacza, że 1% danych testowych jest użytych do przewidywania dopasowania modelu do hipotetycznego zestawu testowego, ponieważ jawny zestaw testowy nie jest na tym etapie dostępny. Po procesie nauczania, model jest zapisywany na dysku.

```
Train on 6026 samples, validate on 61 samples
Epoch 1/10
6026/6026 [==============================] - 14s 2ms/sample - loss: 0.6039 - accuracy: 0.7816 - val_loss: 0.1886 - val_accuracy: 0.9836
Epoch 2/10
6026/6026 [==============================] - 11s 2ms/sample - loss: 0.0957 - accuracy: 0.9648 - val_loss: 0.1033 - val_accuracy: 0.9836
Epoch 3/10
6026/6026 [==============================] - 12s 2ms/sample - loss: 0.0446 - accuracy: 0.9861 - val_loss: 0.0809 - val_accuracy: 0.9672
Epoch 4/10
6026/6026 [==============================] - 11s 2ms/sample - loss: 0.0658 - accuracy: 0.9806 - val_loss: 0.0229 - val_accuracy: 1.0000
Epoch 5/10
6026/6026 [==============================] - 11s 2ms/sample - loss: 0.0891 - accuracy: 0.9736 - val_loss: 0.0417 - val_accuracy: 1.0000
Epoch 6/10
6026/6026 [==============================] - 11s 2ms/sample - loss: 0.0126 - accuracy: 0.9975 - val_loss: 0.0195 - val_accuracy: 0.9836
Epoch 7/10
6026/6026 [==============================] - 11s 2ms/sample - loss: 0.0125 - accuracy: 0.9957 - val_loss: 0.0143 - val_accuracy: 1.0000
Epoch 8/10
6026/6026 [==============================] - 11s 2ms/sample - loss: 0.0057 - accuracy: 0.9982 - val_loss: 0.0053 - val_accuracy: 1.0000
Epoch 9/10
6026/6026 [==============================] - 11s 2ms/sample - loss: 5.0334e-04 - accuracy: 1.0000 - val_loss: 0.0023 - val_accuracy: 1.0000
Epoch 10/10
6026/6026 [==============================] - 11s 2ms/sample - loss: 3.5508e-04 - accuracy: 1.0000 - val_loss: 6.6133e-04 - val_accuracy: 1.0000
```

Rysunek 1: Proces uczenia w programie

2.2 Weryfikacja działania na zbiorze testowym

W celu weryfikacji rzeczywistej skuteczności sieci neuronowej, należy poddać ją testom na zbiorze testowym. Dzięki temu, że są to dane, z którymi model ma styczność pierwszy raz, można łatwo określić czy nie doszło do przeuczenia. Jeżeli sieć uzyskiwałaby bardzo dobre wyniki na danych uczących, a jej skuteczność na zbiorze testowym drastycznie by spadała, oznaczałoby to, że model za bardzo dopasowuje się do danych podanych mu w procesie uczenia. Weryfikacja modelu opisanego w poprzednim rozdziale pokazała jednak, że sieć nauczyła się w sposób prawidłowy. Model, który uzyskał 99,75% skuteczności na zbiorze uczącym, uzyskał 98,32% na zbiorze testowym. Jest to wynik bardzo zadowalający, jednak należy pamiętać o tym, jak dobre dane udało się znaleźć. Tak czy inaczej, skuteczność na takim poziomie można uznać za satysfakcjonującą.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dropout (Dropout)            (None, 100, 100, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 98, 98, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 47, 47, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 21, 21, 64)        36928
_________________________________________________________________
flatten (Flatten)            (None, 28224)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 28224)             0
_________________________________________________________________
dense (Dense)                (None, 64)                1806400
_________________________________________________________________
dense_1 (Dense)              (None, 9)                 585
=================================================================
Total params: 1,863,305
Trainable params: 1,863,305
Non-trainable params: 0
_________________________________________________________________
1010/1010 - 1s - loss: 0.0461 - accuracy: 0.9832
Accuracy: 98.32%
Loss:  0.05
```

Rysunek 2: Proces weryfikacji poprawności w programie

W tabeli poniżej przedstawione są wyniki skuteczności modelu dla każdej z kategorii. Jak widać tylko dla dwóch z nich sieć nie uzyskała stuprocentowej skuteczności.

Tabela 1: Wyniki skuteczności dla pojedynczych klas

#	Jabłko		Gruszka		Winogrono		Awokado		Wiśnia
	świeże	nieświeże	świeże	nieświeże	świeże	nieświeże	świeże	nieświeże	świeże
Skuteczność [%]	93,94	100	100	98,78	100	100	100	100	100

Warto podkreślić, że 6,06% jabłek świeżych, które sieć sklasyfikowała źle, przyporządkowane zostało do kategorii jabłka nieświeże, tak samo 1,22% źle sklasyfikowanych nieświeżych gruszek zostało ocenionych jako gruszki świeże. Oznacza to, że sieć nieomylnie wykrywa z jakim rodzajem owocu ma do czynienia, czasami jednak pojawiają się problemy z określeniem kondycji konkretnego przypadku.

Poniżej przedstawiono fragment kodu źródłowego odpowiedzialnego za proces weryfikacji skuteczności sieci. Pozostałe ważne części kodu pokrywają się z listingami przedstawionymi w poprzednim podrozdziale – dotyczą one procesu wczytywania obrazów i zapisywania ich w postaci danych testowych.

Listing 7: Wczytanie modelu i podanie na wejście danych testowych

```
1 # Sciezka zapisu wytrenowanego modelu oraz jego odczyt
2 keras_model_path = os.path.join(main_dir, 'models', '10epochs_dropout01-01_valsplit001__1')
3 model = tf.keras.models.load_model(keras_model_path)
4
```

```
# Wyswietl podsumowanie modelu
model.summary()

# Wyswietl informacje o skutecznosci klasyfikacji danych testowych
loss, acc = model.evaluate(X, y, verbose=2)
print('Accuracy: {:5.2f}%'.format(100 * acc))
print('Loss: {:5.2f}'.format(loss))
```

Na początku wskazany wytrenowany model sieci zapisany na dysku jest wczytywany oraz wyświetlane są informacje na jego temat. Następnie, model wywoływany jest z użyciem danych testowych, a po zakończeniu tego procesu wyświetlane są informacje o skuteczności sieci neuronowej oraz wartość funkcji kosztu.

2.3 Klasyfikacja na wybranym obrazie

W celu pełnego spełnienia założeń projektowych i możliwości jednostkowej weryfikacji poprawności działania, zaimplementowano program *classify.py*, który pozwala nam na wybranie obrazu z przestrzeni dyskowej i jego klasyfikację przez sieć. Po uruchomieniu, program zapyta użytkownika o obraz, który sieć ma sklasyfikować. Należy tutaj podać **pełną** ścieżkę do wybranego obrazu w systemie, nie stosując przy zapisie cudzysłowów, czy apostrofów. Po klasyfikacji obrazu przez sieć zostaną wyświetlone informacje, których przykład przedstawiono poniżej.

```
(icr) D:\Projekty\python-fruits-recognition>python classify.py
2020-06-09 21:19:29.990388: I tensorflow/stream_executor/platfor
Podaj sciezke do obrazu: D:\Projekty\python-fruits-recognition\da

| Klasyfikacja          |   Prawdopodobienstwo |
|-----------------------+----------------------|
| Jablko (swieze)       |              -0.2614 |
| Jablko (nieswieze)    |              -0.0828 |
| Gruszka (swieza)      |               0.0982 |
| Gruszka (nieswieza)   |               0.5151 |
| Awokado (dojrzale)    |              -0.126  |
| Awokado (niedojrzale) |               0.1319 |
| Winogrono (swieze)    |              -0.0558 |
| Winogrono (nieswieze) |              -0.3607 |
| Wisnia                |               0.1651 |

 Najlepsze dopasowanie:  Gruszka (nieswieza)  - pewność:  0.5151
```

Rysunek 3: Przykładowe informacje o klasyfikacji

Listing 8: Wskazanie obrazu

```
# Przypisz argument wejsciowy do zmiennej
path = input('Podaj sciezke do obrazu: ')
```

Powyższy kod przypisuje argument wejściowy (w postaci pełnej ścieżki z rozszerzeniem) i zapisuje go do zmiennej „path”.

Listing 9: Klasyfikacja obrazu

```
# Zapisz obraz do macierzy NumPy
test_image = cv2.imread(path)

# Zwieksz wymiar macierzy (dostosowanie do wymagan sieci neuronowej)
test_image = np.expand_dims(test_image, axis=0)

# Uzyskanie predykcji dot. obrazu oraz konwersja do listy (tablicy)
result = model.predict(test_image)
result = result[0].tolist()
```

```
10
11 # Dostosowanie wynikow do zakresu [0-1]
12 result[:] = [i / 10000 for i in result]
13 result[:] = [round(i, 4) for i in result]
```

Wskazany obraz jest poddawany klasyfikacji przez sieć, a uzyskane informacje są przetwarzane do zakresu wartości [0-1] z czterema miejscami znaczącymi.

Listing 10: Informacja zwrotna

```
1  # Pierwsza kolumna tabeli
2  labels = ['Jablko (nieswieze)', 'Winogrono (nieswieze)', 'Gruszka (nieswieza)', 'Wisnia', '
       Jablko (swieze)', 'Awokado (dojrzale)', 'Winogrono (swieze)', 'Gruszka (swieza)', '
       Awokado (niedojrzale)']
3
4  # Wypisz wyniki
5  print()
6  print(tabulate([[labels[4], result[4]],
7                  [labels[0], result[0]],
8                  [labels[7], result[7]],
9                  [labels[2], result[2]],
10                 [labels[5], result[5]],
11                 [labels[8], result[8]],
12                 [labels[6], result[6]],
13                 [labels[1], result[1]],
14                 [labels[3], result[3]]],
15                 headers=['Klasyfikacja', 'Prawdopodobienstwo'], tablefmt='orgtbl'))
16 print("\n Najlepsze dopasowanie: ", labels[result.index(max(result))] , " - pewnosc: ", max
       (result))
```

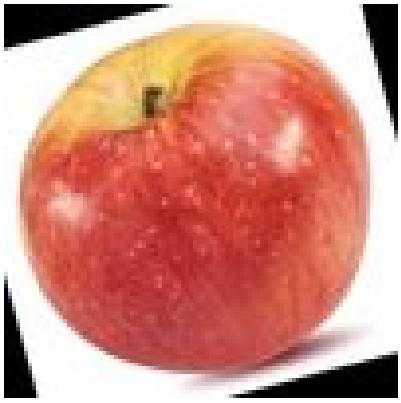
Powyższy kod odpowiada za znalezienie w danych wynikowych najlepszego dopasowania, a następnie wyświetlenia go użytkownikowi, wraz z informacją słowną o dokonanej klasyfikacji. Wyświetlana jest również tabela prezentująca pełną klasyfikację (dla wszystkich kategorii).

3 Podsumowanie

3.1 Wnioski i uwagi

Na podstawie przeprowadzonej weryfikacji poprawności działania opracowanego modelu sieci neuronowej, można stwierdzić, że sieć spełnia swoje zadanie i w zadowalającym stopniu rozpoznaje owoce oraz ich kondycję na obrazach. Należy jednak wziąć pod uwagę, że zarówno obrazy uczące, jak i testowe zostały perfekcyjnie przygotowane – pojedyncze owoce znajdują się na przezroczystym tle, a jakość jest bardzo dobra. Problem zaczyna się pojawiać, gdy spróbujemy sklasyfikować owoc z innej bazy danych, którą dostosowaliśmy do wymagań nałożonych przez model sieci. Niestety, dysponuje ona jedynie zdjęciami jabłek (z podziałem na ich jakość) oraz zawiera czarno-białe tło, owoce są również obrócone. Po weryfikacji skuteczności opracowanego modelu na innej bazie danych okazało się, że wyniki są dramatycznie niskie, ponieważ poprawność wykrycia świeżych i nieświeżych jabłek wyniosła kolejno 3,11% oraz 5,64%. Problem prawdopodobnie pojawiłby się również w przypadku wielu owoców na jednym obrazie. Jest to spowodowane zbyt małą liczbą obrazów znajdujących się w wybranej bazie obrazów, przy czym warto zaznaczyć, że jest to największa dostępna publicznie taka baza. W celu usprawnienia skuteczności opracowanej struktury sieci neuronowej należałoby zatem wytrenować model na znacznie obszerniejszej bazie. Warto również nadmienić, że nie zdecydowano się na poszerzenie wybranej bazy obrazów o drugą z nich właśnie w celu analizy skuteczności modelu przy innej charakterystyce obrazów testowych.

Rysunek 4: Obraz z wybranej bazy



Rysunek 5: Obraz z drugiej bazy

3.2 Potencjalny rozwój

Kierunki potencjalnego dalszego rozwoju opracowanego rozwiązania przedstawiono poniżej.

1. Implementacja znacznie większej liczby obrazów w celach nauki sieci neuronowej.
2. Klasyfikacja większej liczby owoców.
3. Analiza opracowanej struktury sieci neuronowej przy poszerzeniu danych uczących w celu jej potencjalnego ulepszenia przy uzyskiwaniu niesatysfakcjonującej skuteczności.
4. System automatycznego usuwania tła i tym samym wyodrębniania obiektów (owoców) ze zdjęć, jeśli program stwierdziłby taką potrzebę.
5. Optymalizacja opracowanego oprogramowania w celu uzyskania większej wydajności obliczeniowej.

3.3 Skalowalność

Dzięki skalowalnej strukturze aplikacji można w łatwy sposób rozwinąć jej możliwości. W przypadku chęci dodania kolejnych obrazów do aktualnej bazy danych, wystarczy je przenieść do odpowiedniego folderu, upewniając się, że spełniają one wymogi, tj. rozmiar 100x100 i rozszerzenie *„.jpg"*. W przypadku chęci dodania kolejnych kategorii (owoców), wystarczy jedynie dopisać odpowiednie dane do zmiennych „CATEGORIES" (pliki *testing_app.py* oraz *training_app.py*) oraz do zmiennej „labels" (plik *classify.py*). Po wykonaniu wymienionych czynności, sieć nie będzie miała żadnego problemu w procesach nauczania oraz weryfikacji skuteczności z użyciem zaktualizowanej bazy obrazów.

3.4 Czasy wykonywania

Poniżej przedstawiono przykładowe czasy wykonywania kolejno operacji trenowania sieci, weryfikacji skuteczności na zbiorze testowym oraz klasyfikacji wybranego obrazu, zarówno przy użyciu GPU i CPU. Oczywistym jest, iż wyniki te znacznie różnią się w zależności od użytej konfiguracji sprzętowej – w tym przypadku jest to CPU *Ryzen 5 2600* oraz GPU *GeForce GTX650Ti 2GB*.

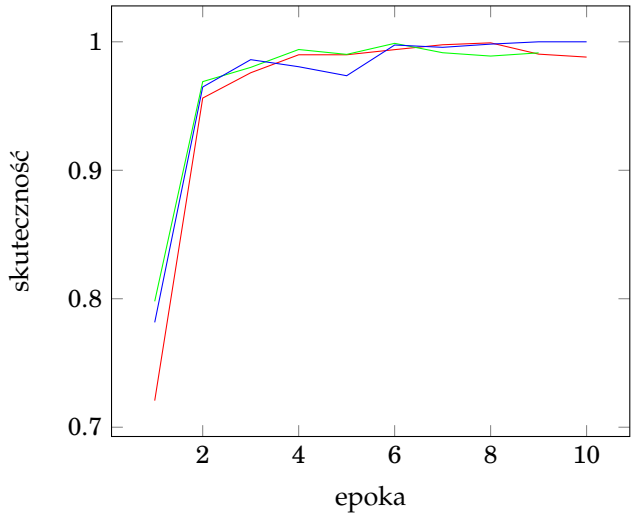
Tabela 2: Przykładowe uzyskane czasy wykonywania

	Proces uczenia		Weryfikacja na danych testowych		Klasyfikacja wybranego obrazu	
	GPU	CPU	GPU	CPU	GPU	CPU
Czas wykonywania [s]	278,95	320,3	8,74	3,53	3,18	1,71

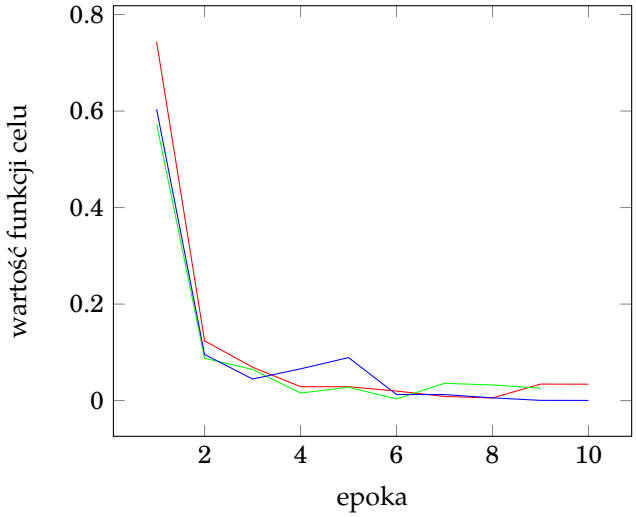
Z tabeli wynika, że czasy wykonywania drastycznie spadają w przypadku użycia GPU do procesu uczenia, co jest spowodowane złożonością obliczeniową wykonywanych operacji. Co ciekawe, w procesach weryfikacji na danych testowych oraz klasyfikacji wybranego obrazu, lepsze wyniki osiąga CPU, gdyż te operacje nie charakteryzują się dużą złożonością obliczeniową, ale również baza obrazów jest w tym przypadku mniejsza.

Dodatek A - charakterystyki uczenia sieci

Poniżej przedstawiono charakterystyki prezentujące zmianę skuteczności oraz wartości funkcji kosztu wraz z postępem procesu nauczania sieci neuronowej. Prezentowane wartości są wynikiem z trzech różnych procesów uczenia się sieci neuronowej o ostatecznie opracowanej strukturze dla dziesięciu epok.



Rysunek 6: Zmiana skuteczności na przestrzeni epok



Rysunek 7: Zmiana wartości funkcji celu na przestrzeni epok

Analizując powyższe charakterystyki możemy stwierdzić, że najlepiej wytrenowanym modelem sieci jest ten reprezentowany przez kolor niebieski. Osiąga on najwyższą skuteczność klasyfikacji na danych uczących (100%) oraz posiada najniższą wartość funkcji celu ($3{,}5508 * 10^{-4}$). Widzimy również, że charakterystyki modelu reprezentowanego przez kolor zielony nie osiągają 10 epoki. Jest to spowodowane nienastępującym spadkiem wartości funkcji kosztu przez trzy kolejne epoki – technika *Early stopping*.

Dodatek B - optymalna wartość parametru *validation_split*

Poszukiwania optymalnego parametru *validation_split* dokonywano wykonując trzy procesy nauczania modelu sieci neuronowej, a następnie poszukiwaniu najwyższej wartości oraz najwyższej średniej wartości dla danych wyników, które przedstawiono w tabeli poniżej.

Tabela 3: Wyniki poszukiwań

validation_split		wynik	średnia
30%	Pierwsza próba	97,62	94,55
	Druga próba	91,98	
	Trzecia próba	94,06	
20%	Pierwsza próba	97,92	93,57
	Druga próba	95,45	
	Trzecia próba	87,33	
10%	Pierwsza próba	93,76	95,71
	Druga próba	98,02	
	Trzecia próba	95,35	
5%	Pierwsza próba	95,15	96,31
	Druga próba	96,34	
	Trzecia próba	97,43	
1%	Pierwsza próba	98,32	96,83
	Druga próba	97,23	
	Trzecia próba	94,95	

Z tabeli jasno wynika, że zarówno najwyższy uzyskany wynik skuteczności, jak i najwyższa wartość średnia skuteczności występuje dla parametru *validation_split* wynoszącego 1%, zatem zaimplementowano taką właśnie wartość w opracowanym modelu sieci neuronowej.

Bibliografia

[1] *Fusing Color and Texture Cues to Categorize the Fruit Diseases from Images*
Autorzy: Shiv Ram Dubey, Anand Singh Jalal
Data: grudzień 2014

[2] *Fruit recognition from images using deep learning*
Autorzy: Horea Mureșan,Mihai Oltean
Data: czerwiec 2018

[3] *Fruits Classification Using Image Processing Techniques*
Autorzy: Chithra Pl, M Henila
Data: marzec 2019

[4] https://keras.io

[5] https://www.tensorflow.org/guide

[6] https://en.wikipedia.org

[7] https://stackoverflow.com

[8] https://radiopaedia.org