

Detekcja psów i kotów

Łukasz Niedźwiadek, 180102 Jakub Sachajko, 179976

Piotr Tomczyk, 175816

May, 2021

Spis treści

1	Wstęp	2
1.1	Opis zadania do zrobienia	2
1.2	Co zostało wykonane	2
1.3	Jakie biblioteki zostały wykorzystane	2
2	Dane	3
2.1	Dane posiadane	3
2.2	Edytowanie danych	3
3	Próby rozwiązania zadania	5
3.1	Próba 1 - Sieć o czterech kategoriach	5
3.2	Próba 2 - Rozpoznawanie pyszczka przy użyciu Haar Cascade	5
3.3	Próba 3 - Rozpoznawanie Pies,Kot,Inne oraz czy zwierze jest nasze	8
3.3.1	Model pies, kot, inne	8
3.3.2	Model kot czy Hiro	11
3.3.3	Model pies czy Cezar	12
3.3.4	Podsumowanie	14
4	GUI	15
4.1	Interfejs	15
4.2	Przykładowe wyniki	15

1 Wstęp

1.1 Opis zadania do zrobienia

Naszym zadaniem było stworzenie aplikacji, która jest w stanie określić czy na podanym zdjeciu znajduje się konkretny pies(Cezar) czy konkretne kot(Hiro).

1.2 Co zostało wykonane

Po uruchomieniu naszego projekt pojawia sie interfejs z przyciskiem wybierz, po przyciśnięciu przycisku będzie można wybrać zdjecie, które podda się analizie. Zdjecie to pojawi się pod przyciskiem i z prawej strony będzie mieć opis czy na danym zdjeciu znajduje się kot, pies czy inny obiekt oraz jeżeli kot to czy nasz i analogicznie w przypadku psa. Możliwa jest ponowna selekcja zdjecia.

1.3 Jakie biblioteki zostały wykorzystane

Do wykonania tego projektu zostały wykorzystane biblioteki takie jak:

- pickle - zapis danych
- tkinter - interfejs
- tensorflow - sieć neuronowa
- keras - sieć neuronowa
- cv2 - obróbka zdjęć

Kluczowymi bibliotekami w tym projekcie były tensorflow, keras oraz cv2. Dwie pozostałe biblioteki pomagały przy edytowaniu i przechowywaniu zdjęć.

2 Dane

2.1 Dane posiadane

Do wykonania naszego projektu potrzebowaliśmy duża ilość zdjęć psów kotów oraz zdjęć otoczenia, na których dane zwierzęta nie występują. Znalezliśmy i pobraliśmy dataset który posiadał 12500 zdjęć psów oraz 12500 zdjęć kotów. Zdjęcia te zostały pobrane ze strony kaggle.com. Z tej samej strony pobraliśmy zdjęcia negatywne, które znaleźliśmy także na kaggle.com w ilości 3000 noszące nazwę Negative images for Haar Cascade.

Najbardziej czasochłonna część okazała się jednak robienie zdjęć swoim zwierzakom. Próba zrobienia wielu zdjęć w różnym oświetleniu w różnych pozycjach z różnych urządzeń wymagała planowania i systematyczności. Udało nam się zrobić 200 zdjęć Cezara oraz Hiro.

2.2 Edytowanie danych

Do dodania naszych danych do folderu i podzieleniu ich na psy, koty oraz inne pobieramy zdjęcia z kategorii(folderów) w których się znajdują a następnie zamieniamy je na obrazy 200 x 200. Następnie Appendujemy do trainingData tablice pikseli w odcieniach szarości, zamiast RGB żeby model nie sugerował się kolorem zwierzęcia, oraz tablice dzięki której bedzie wiadomo jakie zdjęcie zostało dodane. Następnie mieszamy całą liste i zapisujemy ją przy pomocy biblioteki pickle do plików PhotosGray.pickle oraz ClassGray.pickle.

```
1 training_data = []
2 Data = "C:/Users/Kuba/Desktop/SemestrIV/Sztuczna Inteligencja/Project/PetImagesGeneral"
3 Categories = ["Cat", "Dog", "Other"]
4 imgSize = 200
5
6 def create_data():
7     for category in Categories:
8         path = os.path.join(Data, category) #path
9         classNumber = Categories.index(category)
10        for img in os.listdir(path):
11            try:
12                listOne=[0,0,0]
13                listOne[classNumber]=1
14                imgArray = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
15                newArray = cv2.resize(imgArray,(imgSize, imgSize))
16                training_data.append([newArray, listOne])
17            except Exception as e:
18                pass
19
20    create_data()
21
22    random.shuffle(training_data)
23
24    Photos = []
25    Class = []
26
27
```

```
28 for features, label in training_data:  
29     Photos.append(features)  
30     Class.append(label)  
31  
32 PhotosGray = np.array(Photos).reshape(-1, imgSize, imgSize, 1)  
33 ClassGray = np.array(Class)  
34  
35 #zapisywanie do pliku  
36 pickle_out = open("PhotosGray.pickle", "wb")  
37 pickle.dump(PhotosGray, pickle_out)  
38 pickle_out.close()  
39  
40 pickle_out = open("ClassGray.pickle", "wb")  
41 pickle.dump(ClassGray, pickle_out)
```

3 Próby rozwiązaania zadania

3.1 Próba 1 - Sieć o czterech kategoriach

Naszym pierwszym pomysłem na rozwiązaanie zadania było stworzenie jednego modelu categorical convolutional neural network, której kategorie brzmiały następująco: Pies, Kot, Hiro, Cezar. Po załadowaniu danych oraz zaimplementowaniu sieci oraz testowaniu jej wyniki były zdecydowanie poniżej naszych oczekiwani. Ze względu na małą ilość zdjęć Hiro i Cezara, uznaliśmy że dysproporcja zdjęć 12500 dla na przykład psa i 130 dla kota przy użyciu funkcji shuffle i pobraniu 10 procent z końca próbek validujących, model ten osiągał najlepsze wyniki jeżeli rozróżniał ogólnie psa i ogólnie kota pomijając Hiro i Cezara. Co wiecej był on prawie przekonany i widząc zdjęcie cezara przewidywał, że jest to kot w 99 procentach. W skrócie pomimo wysokiego accuracy bo dochodziło ono do 82 procent rozróżnienie przez ten model Hiro i cezara stał się praktycznie niemożliwy. Napotkaliśmy dwa problemy, które wzięliśmy pod uwagę w przyszłych działaniach :

- Ilość próbek dla każdej kategorii musi być taka sama
- Cezar rasy Yorkshire Terrier jest bardzo podobny do kotów

3.2 Próba 2 - Rozpoznawanie pyszczka przy użyciu Haar Cascade

Następna próba osiągnięcia celu było użycie Haar Cascade, w celu rozpoznania pyszczka. Pierwszym krokiem było przekonwertowanie 125 zdjęć Hira na bitmapy. Następnie używając ObjectMarkera, trzeba było ręcznie zaznaczyć pyszczki kota na zdjęciach. Dane nie były tworzone z myślą rozpoznawania pyszczka więc na niektórych zdjęciach niemożliwym było zaznaczenie głowy ze względu na pozycje. Zaznaczyliśmy obszar który zawierał brodę, uszy, oraz część wasów. Oto przykład danych w pliku info.txt, w którym zapisywaliśmy współrzędne.

```
hiro (1) resized.bmp
  1. rect x=219      y=53      width=193      height=254
hiro (100) resized.bmp
  1. rect x=18 y=223  width=329      height=437
hiro (102) resized.bmp
  1. rect x=5  y=167  width=334      height=499
hiro (104) resized.bmp
  1. rect x=358     y=208     width=407      height=504
hiro (107) resized.bmp
```

Teraz musielimy stworzyć inny sposób na przechowywania danych o naszych współrzędnych, wiec wybraliśmy wektor. Oto komenda bat:

```
createsamples.exe -info positive/info.txt -vec vector/hirovector.vec -num 204 -w 24  
-h 24
```

Num jest odpowiedzialny za ilość zdjęć, w oraz h za wielkość próbek

Po stworzeniu wektora zaczeliśmy trenować nasz algorytm. Musielimy posiadać również co najmniej 100 próbek negatywnych.

Rysunek 1: Przykład próbki negatywnej.



```
haartraining.exe -data cascades -vec vector/facevector.vec -bg negative/bg.txt -npos 200  
-nneg 200 -nstages 15 -mem 1024 -mode ALL -w 24 -h 24
```

npos to liczba próbek pozytywnych

nneg to liczba próbek negatywnych

nstages to liczba epok

Po zakończeniu treningu, wyniki pojawiły się w postaci folderów z plikami tekstowymi dla każdego epocha czyli przejścia modelu przez wszystkie zdjęcia.

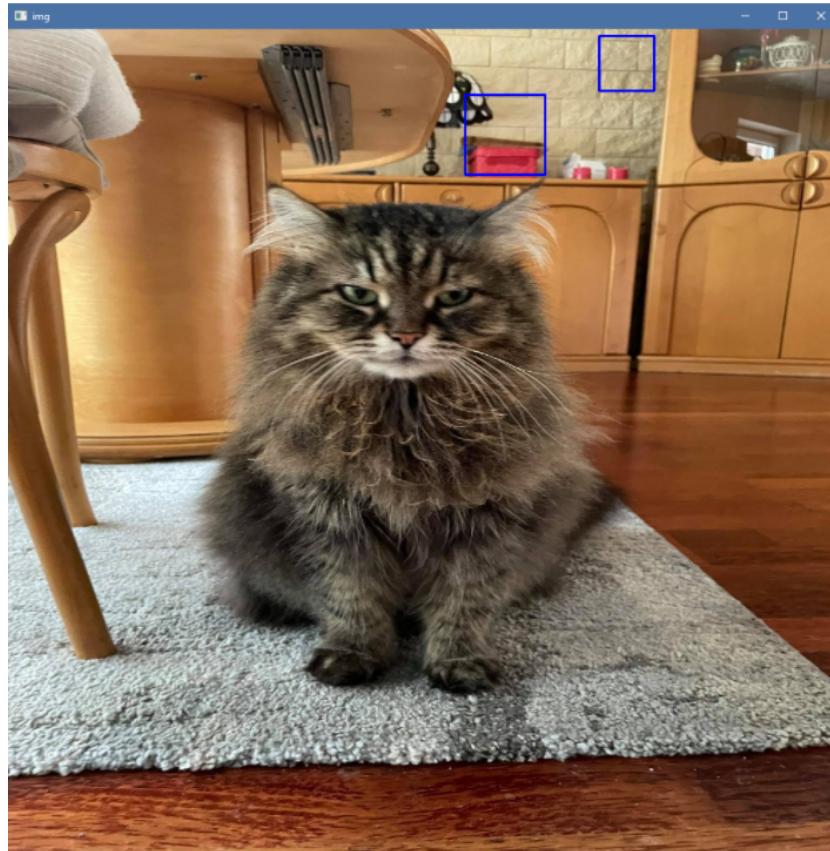
Ostatnim krokiem było przekonwertowanie otrzymanych wyników przy użyciu haarcvconv.exe na plik o rozszerzeniu xml.

Ostatnim krokiem było załadowanie tego do naszego programu w pythonie oraz sprawdzenie działanie.

przy użyciu linijek przypisujących otrzymana kaskade do zmiennej przy użyciu cv2 oraz sprawdzeniu czy na danym zdjeciu istnieje taki obiekt(pyszczek Hiro):

```
HiroCascade = cv2.CascadeClassifier('Hiro.xml')
faces = HiroCascade.detectMultiScale(gray, 1.01, 7, minSize=(20, 20), flags=cv2.CASCADE_SCALE_IMAGE)
```

Rysunek 2: Przykład działania Haar Cascade.



Jak można zauważyć na zdjeciu zaznaczone kwadratami miejsca nie sa twarza danego zwierzecia. Wyniki były absolutnie niesatysfakcjonujace, pomimo zmiany paramatrów wykrywania twarzy, zaczynajac od wielkości a kończąc na dokładności wykrywania, wyniki nie poprawiły się znaczaco. Dlatego właśnie porzuciliśmy ten pomysł.

3.3 Próba 3 - Rozpoznawanie Pies,Kot,Inne oraz czy zwierze jest nasze

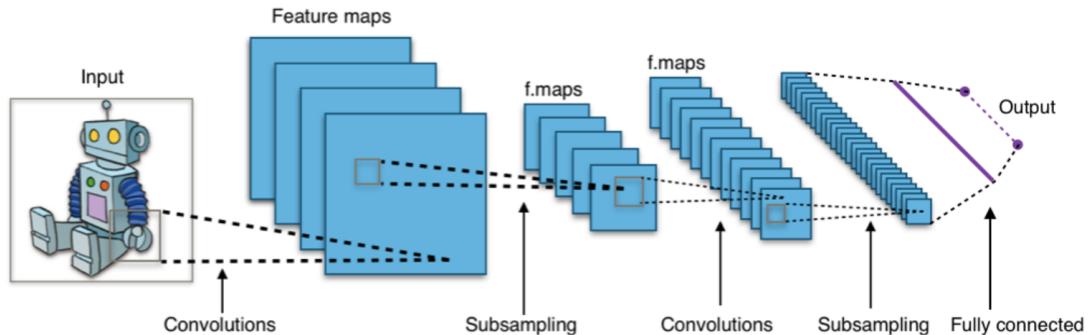
Jest to nasza ostatnia próba i to właśnie ja zaprezentujemy. Założyliśmy że zrobimy trzy modele:

- Model rozpoznajacy pies, kot, inne
- Model rozpoznajacy czy kot to Hiro
- Model rozpoznajacy czy pies to Cezar

3.3.1 Model pies, kot, inne

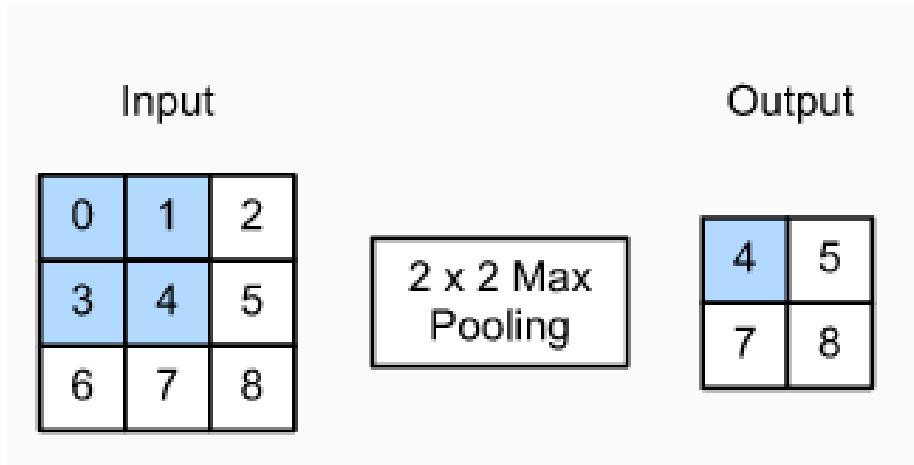
Do stworzenia danego modelu użyliśmy trzech zestawów danych co sprawiło że nasza klasyfikacja bedzie kategoryczna a nie binarna. Do Stworzenia modelu użyliśmy teorii z convolutional neural network. Zaczynajac od samego pojęcia jest to cześć uczenia głebokiego stosowana do analizy obrazu. Sieć ta jest siecią wielowartusową i jest stosowana przy filtrowaniu i uwypuklaniu cech danej części.

Rysunek 3: Przykład działania Convolutional neural network.



Sieć ta posiada warstwy wśród których możemy wyróżnić rozmiar, ilość oraz typ jak na przykład warstwa pooling layers. Pooling layers działa pobierając wycinek danego zdjęcia w naszym przypadku 3×3 i wybierając największe wartości z pomniejszych wycinków 2×2 tworzą nowy wycinek 2×2 .

Rysunek 4: Przykład działania dla wycinków 3x3 MaxPooling(2x2).



Parametry mogą być różne i dają zróżnicowane wyniki validacji. Zaczynając od ilości warstw, wielkości warstw, oraz wielkości wycinków pobieranych do max poolingu oraz sam rozmiar wyjściowy maxpoolingu. Dlatego aby nasz model był jak najdokładniejszy użyliśmy tensorboard czyli część biblioteki tensorflow do analizy naszych wyników przy różnych parametrach. Sprawdzane były parametry layerDensity od 1 do 4, layerSize[32,64] oraz ilość warstw z MaxPoolingiem(2,2) od 1 do 6.

W naszym przypadku najlepszy okazał się model który posiadał density = 1, layerSize = 32 oraz ilość warstw maxPooling = 6. Pomimo, że szara linia na wykresie skacze wyżej na dolnym wykresie możemy zauważać, że szara linia od dawna rośnie, oznacza to, że dany model staje się przetrenowany i zaczyna zapamiętywać próbki validacyjne.

Po ponownym uruchomieniu programu w celu zmiany rozmiaru obrazów na 200 x 200 pikseli, program poprzez użycie shuffle do przemieszania próbek wzął inne obrazy validujące dlatego trafność naszego modelu spadła do 73 procent.

Na końcu modelu musimy podać Dense(3) ponieważ posiadamy trzy kategorie, a do jednej z nich musi zostać zapisany. Zamiast używać w ostatnim activation sigmoid, który zwracałby 1 lub 0 my użyliśmy softmax, który zwraca prawdopodobieństwo z przedziału od 0 do 1 dla każdej opcji.

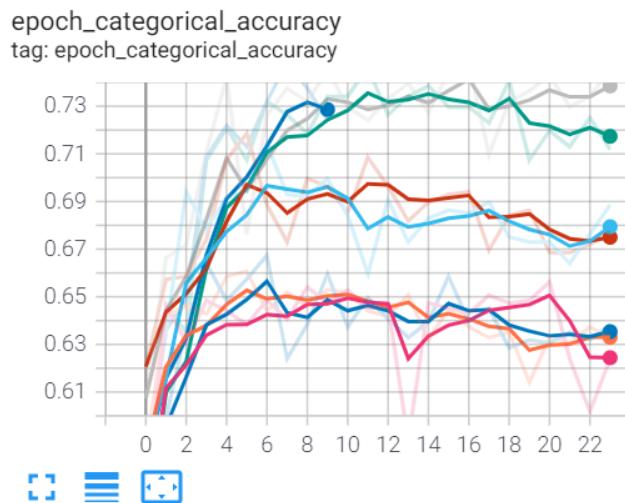
ustawiona została także zmienna validationSplit=0.1, która bierze 10 procent wszystkich zdjęć i je odkłada, żeby sprawdzić jak dobry jest model. Następnie model jest zapisywany

```

1 model.add(Dense(3))
2 model.add(Activation('softmax'))
3 model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=['categorical_accuracy'])
4 model.fit(X, Y, epochs=11, batch_size = 32, validation_split=0.1, callbacks=[tensorboard])
5 model.save("CDO.model")

```

Rysunek 5: Przykład validacji wykresu w zależności od layerDensity oraz ilości warst MaxPooling.

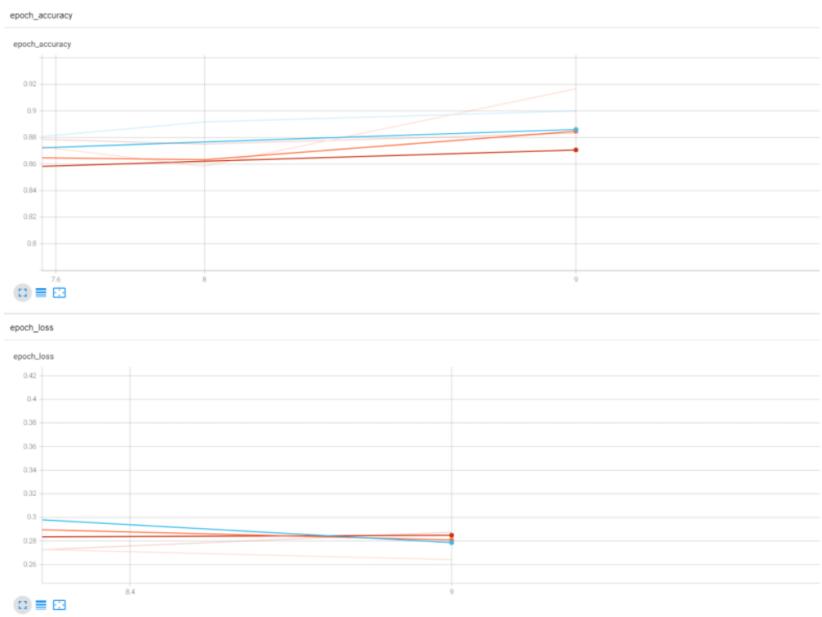


epoch_loss



3.3.2 Model kot czy Hiro

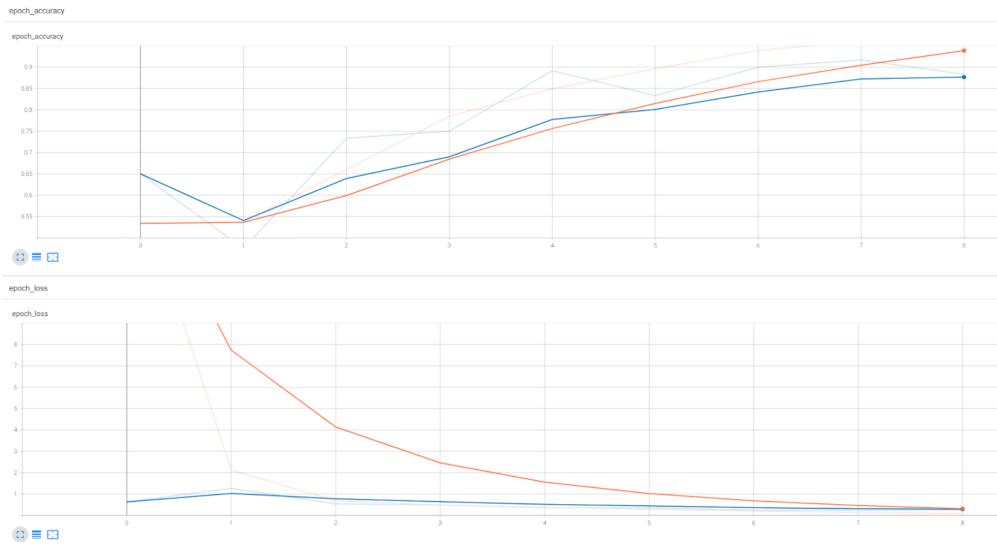
W podobny sposób jak w podpunkcie 3.1.1 zrobiliśmy rozpoznanie Kot-Hiro. Do stworzenia modelu w tym przypadku użyliśmy binary Crossentropy ze względu na tylko dwa rodzaje zdjęć. Przy tworzeniu modelu użyliśmy activationLayer = sigmoid ponieważ są tylko dwa wybory i wyniki zwarcane przez sigmoid do 0 lub 1. Podczas analizowania wyników w tensorboard wybraliśmy trzy najlepsze zestawy parametrów, które przedstawiamy na wykresach poniżej:



Najlepszym zestawem oznaczony na wykresach kolorem niebieskim okazał się dla parametrów:

- denseLayer = 1
- layerSize = 128
- convLayer = 1

Jest on w stanie osiągać nawet do 90 procent poprawności na zestawie walidacyjnym. Oto on przedstawiony na wykresie:



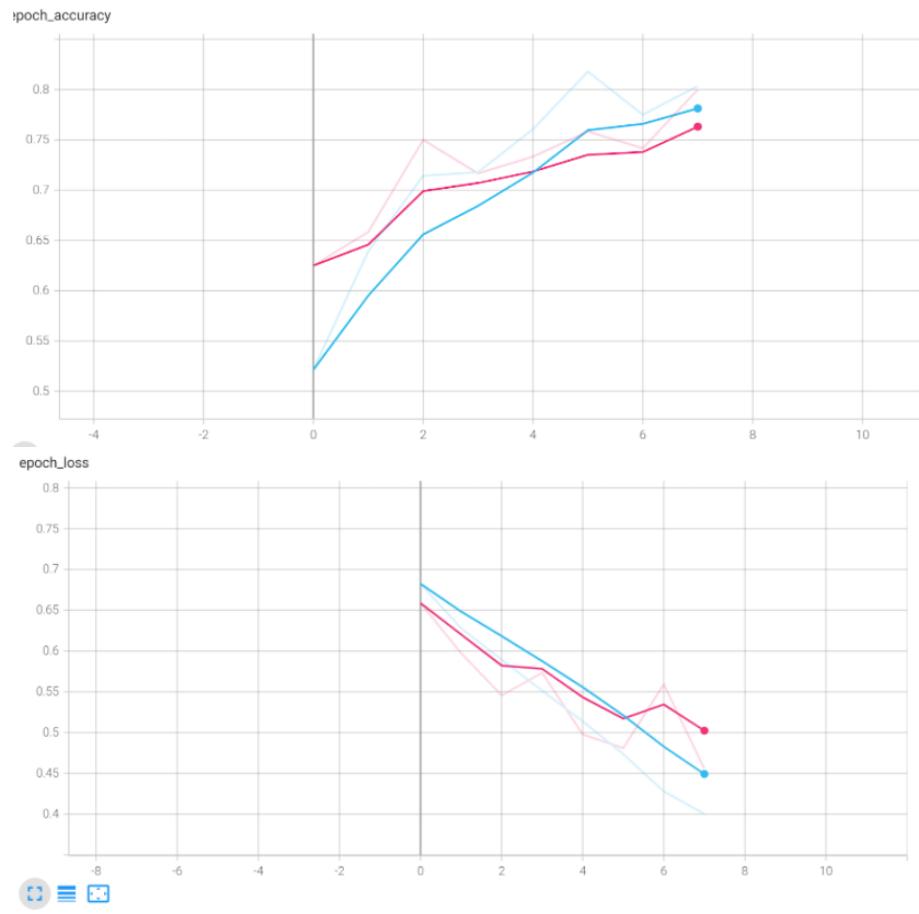
Początkowo testy robiliśmy na 15 epokach, ale po stwierdzeniu przeuczenia na epoce 10, zmniejszyliśmy ilość epok do 9.

3.3.3 Model pies czy Cezar

Rozpoznawanie Pies-Cezar zrobiliśmy analogicznie do rozpoznawania Kot-Hiro, jednak wyniki walidacji nie były aż tak dobre, ponieważ osiągneliśmy tylko około 78 procent.

Najbardziej optymalne ustawienia dla danego zestawu danych to:

- denseLayer = 0
- layerSize = 32
- convLayer = 2



3.3.4 Podsumowanie

Podsumowując, przeprowadziliśmy obszerne testy dla każdej z sieci, dobraliśmy parametry dla najlepszej efektywności oraz zapisaliśmy modele do plików aby złączyć je w GUI. Rezultaty sprostały naszym oczekiwaniom.

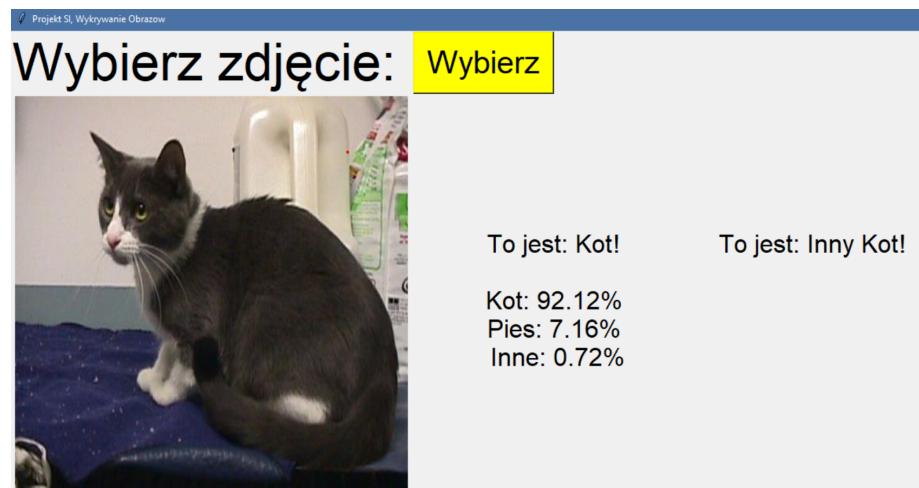
4 GUI

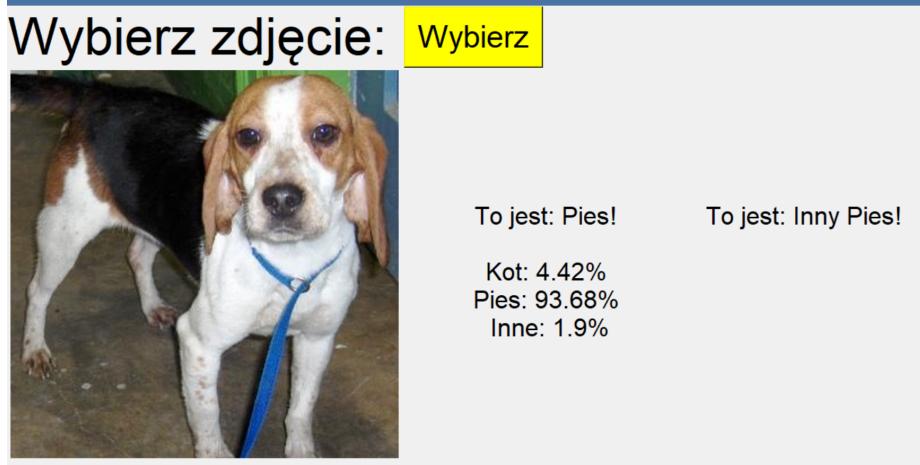
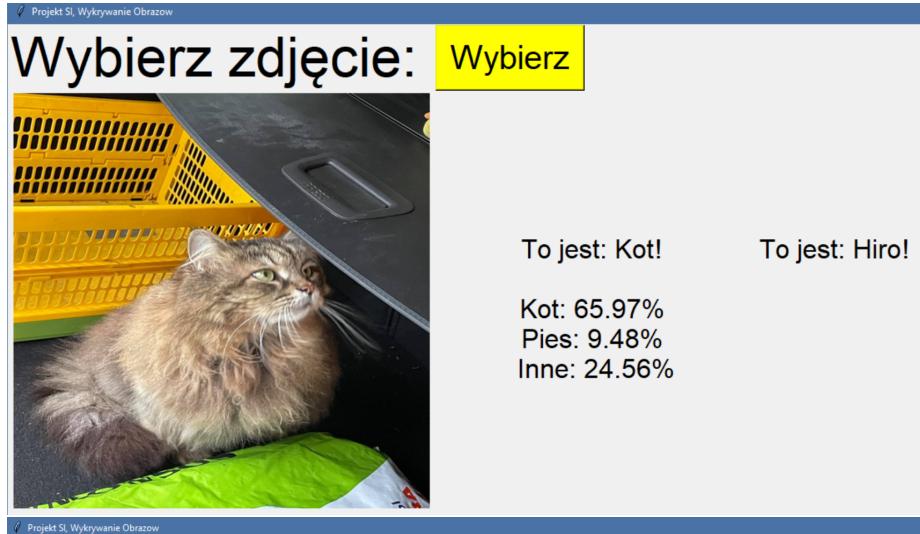
4.1 Interfejs

Do skonstruowania graficznego interfejsu użytkownika użyliśmy biblioteki tkinter, importując z niej FileDialog oraz MessageBox. Do wyświetlenia wybranego zdjęcia użyliśmy biblioteki PIL, z której zainportowaliśmy ImageTk oraz Image. Działanie wygląda następująco: po wybraniu zdjęcia przez użytkownika w funkcji clicked wywoywanej po naciśnięciu przycisku ładujemy wszystkie 3 modele, a następnie sprawdzamy, która opcja ma najwięcej procent. W zależności od wyniku modyfikowaliśmy wyjściowy string wyświetlany po prawej stronie zdjęcia.

4.2 Przykładowe wyniki

W tej części zaprezentujemy kilka przykładów działania programu zaprezentowanych w zdjęciach poniżej:







Projekt SI, Wykrywanie Obrazów

Wybierz zdjęcie:

Wybierz



To jest: Inna opcja!

Kot: 8.53%

Pies: 10.19%

Inne: 81.28%