Metody rozpoznawania obrazów
Zadanie 3 – raport

Mateusz Kocot

20 listopada 2021

Spis treści

1 Zbiór danych

1.1 Przygotowanie danych

Zbiór danych, który utworzyłem, składa się z trzech rodzajów wyposażenia żołnierza Starożytnego Rzymu: gladius (miecz), scutum (tarcza) oraz lorica (zbroja). Pobrałem po 333 obrazy z każdej kategorii. Za pierwszym razem usunąłem wszystkie niewłaściwe przykłady, natomiast ostatnia sieć radziła sobie zbyt dobrze i nie mogłem znaleźć nawet jednego obrazu klasyfikowanego błędnie. Z tego powodu, ostatecznie usunąłem tylko kilkanaście najbardziej rażących przypadków.

1.2 Wczytanie obrazów

Do wczytania użyłem funkcji `tf.keras.utils.image_dataset_from_directory`. Oprócz załadowania danych, pozwala ona na przeskalowanie obrazów oraz podzielenie ich na zbiór treningowy i testowy. Przykładowo, pobranie części treningowej:

```
train_ds = tf.keras.utils.image_dataset_from_directory(
    root_dir,
    label_mode='categorical',
    image_size=(32, 32),
    batch_size=32,
    shuffle=True,
    seed=111,
    validation_split=0.2,
    subset='training')
```

`root_dir` zawiera ścieżkę do katalogu z obrazami. By otrzymać część testową, użyłem `subset='validation'`.

Przykładowe obrazy ze zbioru (w rozdzielczości 256×256) wraz z ich etykietami przedstawiłem na rys. 1.

Rys. 1: Próbka 10 obrazów ze zbioru

2 Sieci konwolucyjne

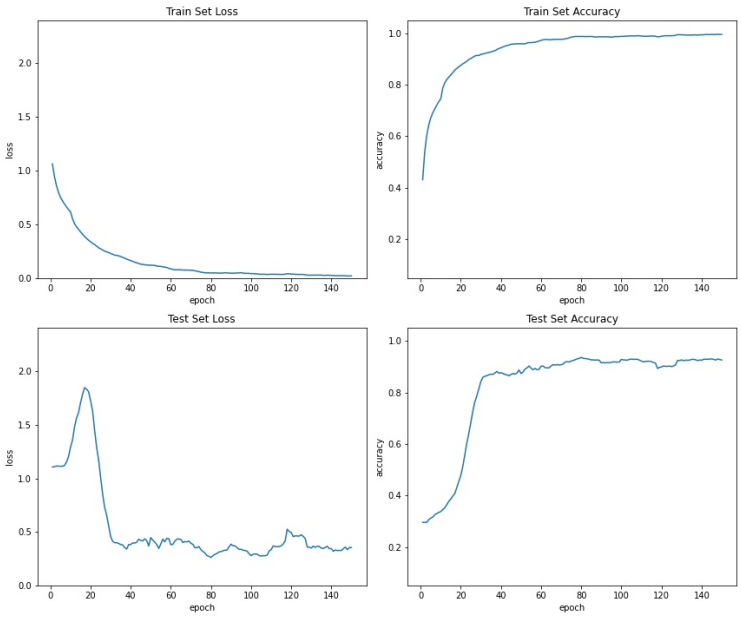
2.1 Sieć z zadania 3. trenowana od zera

Wytrenowałem sieć z zadania 3. Jako metodę optymalizacji wybrałem SGD z `learning_rate=0.0005` oraz `momentum=0.9`, a jako funkcję straty – `CategoricalCrossentropy`. W ten sposób trenowane będą też kolejne sieci.

Udało się uzyskać całkiem dobre wyniki. Ostatecznie *accuracy* dla zbioru testowego wyniosło ok. 93%. Wykresy funkcji straty oraz *accuracy* w zależności od epoki pokazałem na rys. 2.

2.2 Sieć z zadania 3. – *transfer learning*

Po wczytaniu sieci zadania 3. (`model_2_base`) zamroziłem jej wagi:



Rys. 2: Wyniki sieci z zadania 3. trenowanej od zera

```
model_2_base.trainable = False
```

Dzięki temu, parametr `trainable` został ustawiony na `False` we wszystkich warstwach sieci. Dla warstw *batch normalization* oznacza to pracę w trybie inferencji.

Następnie utworzyłem `model_2` poprzez użycie wszystkich warstw z sieci `model_2_base` oraz dodanie nowej, niewytrenowanej warstwy na końcu:

```
from tensorflow.keras import layers as tf_layers
model_2 = tf.keras.Sequential(layers=[
    *model_2_base.layers[:-1],
    tf_layers.Dense(3, activation='softmax')
], name='model_2')
```

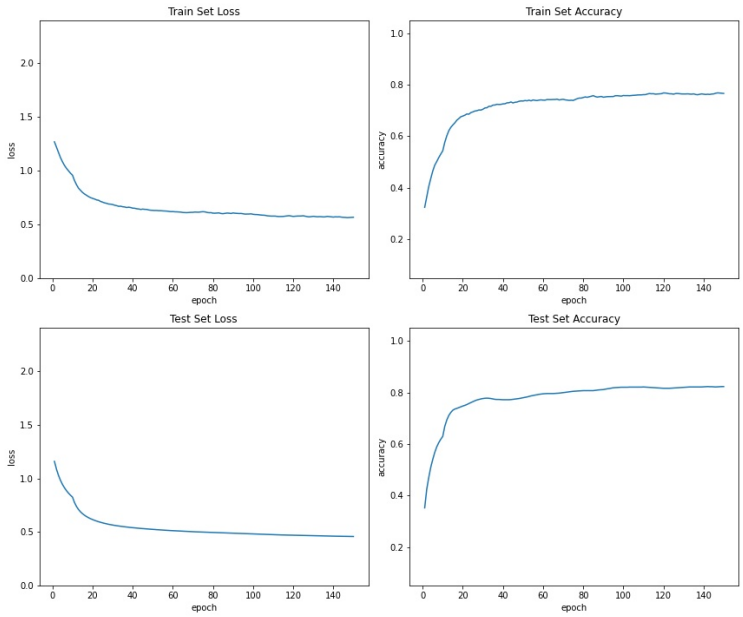
Domyślnie warstwy w Tensorflow-ie mają ustawione `trainable=True`, co oznacza, że wagi dodanej na końcu warstwy nie są zamrożone.

Wyniki tej sieci wypadają sporo gorzej od poprzedniczki (rys. 3). Prawdopodobnie jest to spowodowane specyfiką klas w zbiorze Cifar10 – są to pojazdy oraz zwierzęta, czyli klasy sporo różniące się od klas przygotowanych przeze mnie w tym zadaniu.

2.3 Xception

Pobrałem model od razu bez wszystkich (dwóch) warstw wzwyż począwszy od GAP (parametr `include_top`) oraz zamroziłem jego wagi. Następnie dodałem nową warstwę GAP oraz warstwę *dense*:

```
model_3_base = tf.keras.applications.Xception(
```



Rys. 3: Wyniki sieci z zadania 3. trenowanej z użyciem *transfer learning*

```
    include_top=False,
    input_shape=(256, 256, 3))
model_3_base.trainable = False

output = tf_layers.GlobalAveragePooling2D()(model_3_base.output)
output = tf_layers.Dense(3, activation='softmax')(output)
model_3 = tf.keras.Model(inputs=model_3_base.input, outputs=output)
```

Wciąż nie udało się pobić modelu pierwszego (rys. 4).

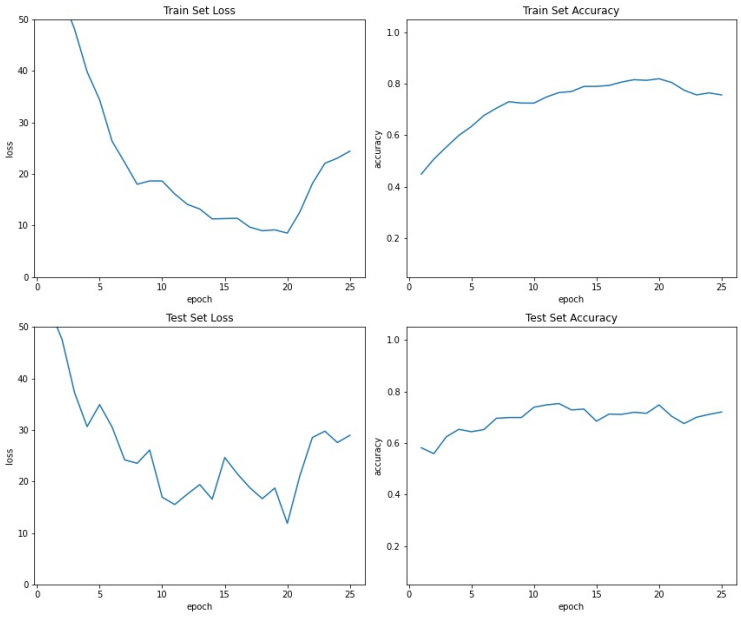
2.4 Xception - *fine-tuning*

Wczytałem i rozmroziłem poprzednią sieć. Ustawiłem także warstwy typu *batch normalization* na tryb inferencji:

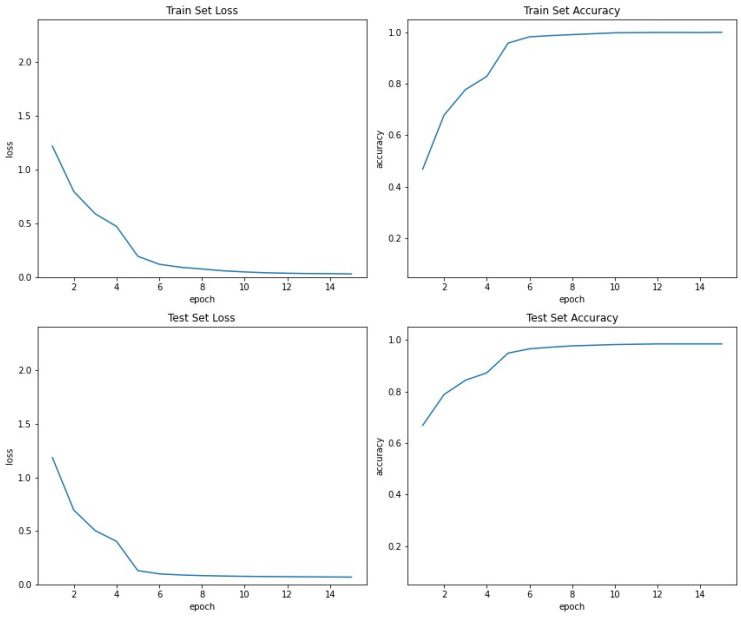
```
model_4 = tf.keras.models.clone_model(model_3)
model_4.load_weights('./saved_weights/model_3/model_3_weights')
model_4.trainable = True

for layer in model_4.layers:
    if layer.name.startswith('batch_normalization'):
        layer.trainable = False
```

Wytrenowałem nową sieć, tym razem używając 10 razy mniejszego `learning_rate` (`0.00005`). W końcu udało się uzyskać bardzo dobre wyniki (rys. 5)! *Accuracy* dla zbioru treningowego oraz, co ważniejsze, dla zbioru testowego mocno zbliżyło się do 100% (98.47% dla zbioru testowego).



Rys. 4: Wyniki sieci Xception przed *fine-tuningiem*

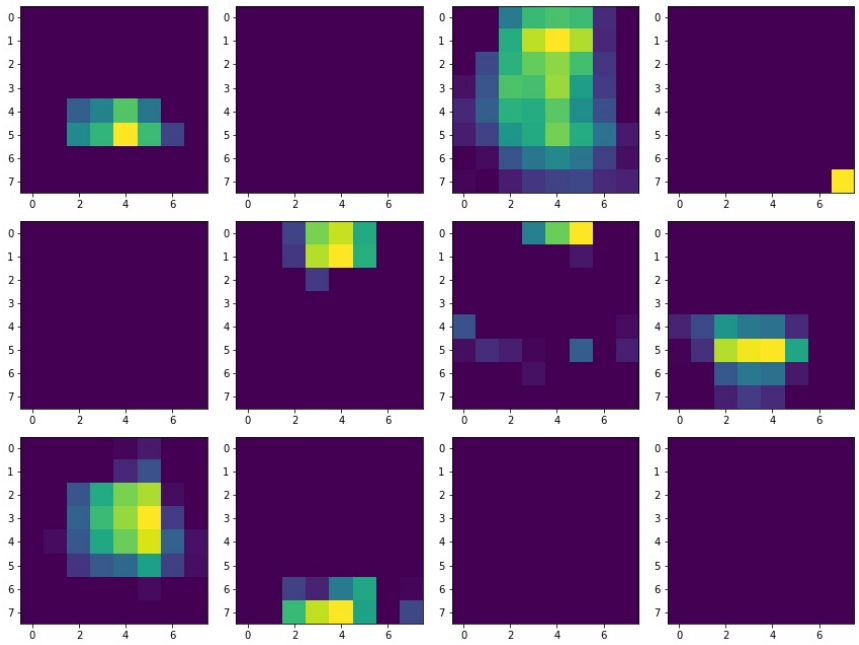


Rys. 5: Wyniki sieci Xception po *fine-tuningu*

3 Próba interpretacji

3.1 Obrazy klasyfikowane poprawnie

Podałem na wejście sieci jeden z obrazów klasyfikowanych poprawnie. Zawartości kilkunastu kanałów wyjściowych sieci Xception (przed warstwą GAP) pokazałem na rys. 6. Dokładnie widać, gdzie mniej więcej znajdują się wykryte cechy.



Rys. 6: Przykładowe kanały na wyjściu sieci Xception (przed warstwą GAP)

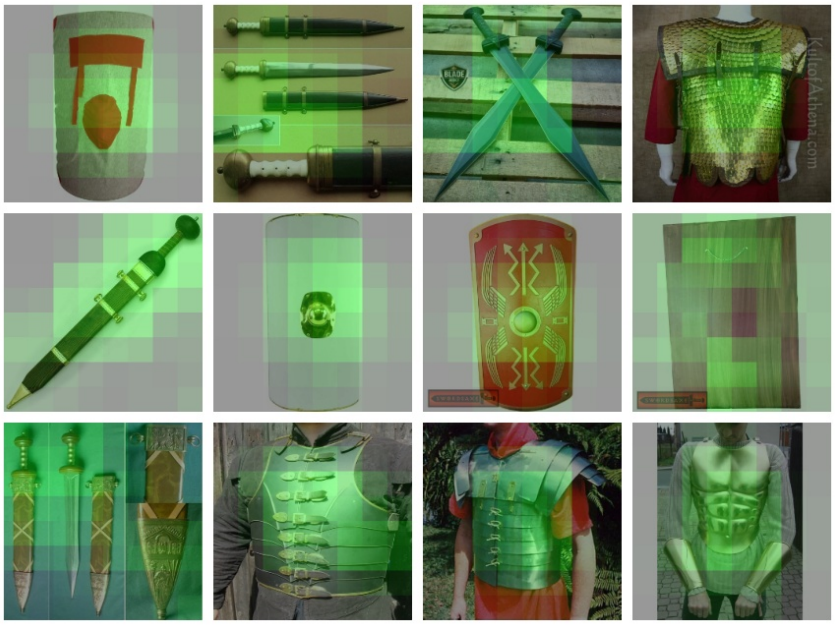
Następnie zaimplementowałem sumowanie wszystkich kanałów z odpowiednimi wagami w taki sposób, by otrzymać *heatmapę*:

```
heatmap = np.sum(x_channels *
    model_4.layers[-1].weights[0][:, np.argmax(y)], axis=2)
```

`x_channels` to wspomniane wcześniej kanały, a `np.argmax(y)` to etykieta rozpatrywanego obrazu.

Wykonałem takie *heatmapy* dla kilkunastu obrazów klasyfikowanych poprawnie, odpowiednio przeskalowałem do rozmiaru 256×256 oraz nałożyłem na kanały zielone wejściowych obrazów. Efekt przedstawiłem na rys. 7.

Oprócz przybliżonych lokalizacji przedmiotów na obrazach, można także zaobserwować, jakie elementy mają mniej więcej wpływ na klasyfikację do danej klasy. Przykładowo, na wykrycie scutum największy wpływ ma jej środkowa część (umbo).



Rys. 7: *Heatmapy* nałożone na przykładowe, poprawnie klasyfikowane obrazy

3.2 Obrazy klasyfikowane błędnie

Podobną procedurę chciałem wykonać dla kilkunastu obrazów klasyfikowanych błędnie, natomiast okazało się, że sieć nie radzi sobie z zaledwie trzema obrazami ze zbioru testowego. Jak widać na rys. 8, nie są to obrazy typowe dla odpowiadającym im kategoriom.

- Pierwszy obraz przedstawia porozrzucane tarcze, które na domiar złego nie są typu rzymskiego scutum. O wykryciu gladiusa przesądziła prawdopodobnie dość wyraźna, prosta krawędź między tarczami.
- Drugi przypadek jest bardzo ciekawy. Gladius jest bardzo wyraźnie widoczny, natomiast sieć wybrała zbroję. Najpewniej stało się tak, gdyż jasne elementy miecza znajdują się na tle czerwonego materiału, co jest charakterystyczne dla wielu obrazów ze zbroją (np. przedostatni na rys. 7).
- Na trzecim obrazku tarcza widoczna jest w tle. Będąca na pierwszym planie miarka w istocie najbardziej przypomina wykryty gladius.

Rys. 8: *Heatmapy* nałożone na błędnie klasyfikowane obrazy