2023

CNN IMAGE CLASSIFIER

TECHNIKI KLASYFIKACJI DANYCH

JAKUB BINIEK, MICHAŁ STRUS

Temat projektu

Tematyką naszego projektu jest budowa Klasyfikatora Obrazów za pomocą konwolucyjnej sieci neuronowej. Zamysłem projektu jest analizowanie obrazów w celu odpowiedniej klasyfikacji zdjęć na przykładzie zdjęć zwierząt z zadanego zbioru danych jako kot lub pies.

Wstępny setup i ładowanie danych

W pierwszym etapie naszego projektu importujemy potrzebne nam biblioteki do budowy klasyfikatora obrazów. W naszym projekcie korzystamy z biblioteki tensorflow, seaborn, keras, numpy, matplotlib, openCV oraz imghdr. Import wygląda następująco:

```
[ ]  import numpy as np
```

```
[ ]  import tensorflow as tf
     import os
```

```
[ ]  import cv2
     import imghdr
     from matplotlib import pyplot as plt
     import seaborn as sns
     sns.set()
```

Ponadto w tym etapie projektu, z zadanego zbioru danych usunęliśmy dane w innym rozszerzeniu niż .jpg .png .jpeg oraz .bmp. Kod wygląda następująco:

```
[ ]  for image_class in os.listdir(data_dir):
       for image in os.listdir(os.path.join(data_dir, image_class)):
         image_path = os.path.join(data_dir, image_class, image)
         try:
           img = cv2.imread(image_path)
           tip = imghdr.what(image_path)
           if tip not in image_extens:
             print("Wrong format of file {}".format(image_path))
             os.remove(image_path)
         except Exception as e:
           print("Issue with image {}".format(image_path))
```

W następnym kroku dokonaliśmy załadowania danych z zbioru danych pochodzącego z kaggle.com zawierającego zdjęcia zwierząt. Wczytujemy dane przez API z biblioteki keras. W argumentach funkcji podajemy ścieżkę do folderu, w którym znajdują się dwa pod-foldery zawierające klasy „cats” oraz „dogs”. Klasą 1 są psy, a klasą 0 są koty. W tym etapie stworzyliśmy iterator w pythonie w celu przechodzenia po obiektach.

```
[ ]  data = tf.keras.utils.image_dataset_from_directory('/content/drive/MyDrive/data-science-bootcamp/data')

     Found 10027 files belonging to 2 classes.
```

```
[ ]  data_iterator = data.as_numpy_iterator()
```

```
[ ]  data_iterator

     <tensorflow.python.data.ops.dataset_ops._NumpyIterator at 0x7f1aeeb8ba30>
```

```
     batch = data_iterator.next()
```

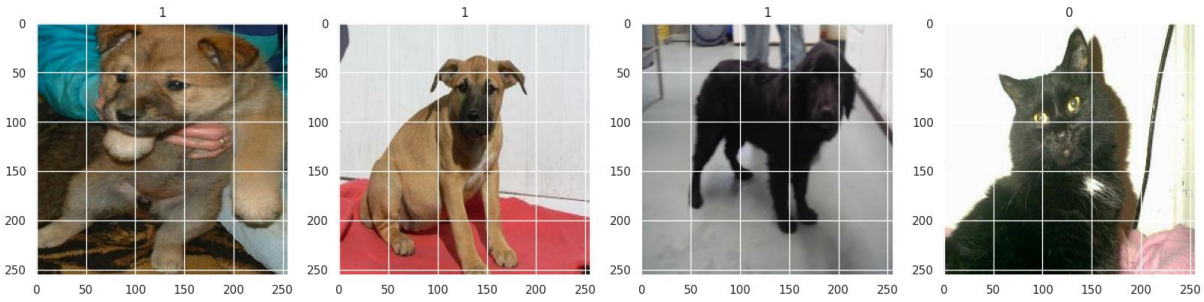
Images size as numpy arrays

```
[ ]  batch[0].shape

     (32, 256, 256, 3)
```

Class 1 = dog, Class 0 = cat

Wyświetlenie czterech przykładowych obiektów wygląda następująco:



Preprocessing danych

Po załadowaniu danych, w naszym projekcie przeszliśmy do etapu przygotowania danych do modelu. Najpierw skalujemy zmienne do przedziału [0,1] w celu dodania ich do konwolucyjnej sieci neuronowej.

```
[ ]  data = data.map(lambda x,y: (x/255, y))
```

Następnie podzieliliśmy dane na zbiór uczący, walidacyjny oraz testowy w proporcjach odpowiednio 0.7 , 0.2 oraz 0.1

```
[ ]  len(data)
```

```
[ ]  train_size = int(len(data)*0.7)
     val_size = int(len(data)*0.2)+1
     test_size = int(len(data)*0.1)+1
```

Z naszego zbioru danych wyciągamy rozmiar, który został określony powyżej. W kodzie wygląda to następująco:

```
[ ]  train = data.take(train_size)
     val = data.skip(train_size).take(val_size)
     test = data.skip(train_size+val_size).take(test_size)
```

Budowanie modelu

W kolejnym etapie projektu tworzymy model sekwencyjny z wbudowanych funkcji z biblioteki keras. Do modelu dodajemy warstwę konwolucyjną z 16 kernelami rozmiaru 3 x 3. Jako funkcję aktywacji wybieramy funkcję „relu". Jako input_shape w pierwszej konwoluncji podajemy rozmiar obrazów.

Następnie używamy Max Poolingu, aby wyodrębnić poszukiwane cechy ze zdjęć. Tą procedurę wykonujemy trzy-krotnie by zmniejszyć rozmiar zbioru danych załadowanych do modelu. Ponadto trzy-krotnie używamy BatchNormalization w celu normalizacji danych.

```
model = Sequential()
model.add(Conv2D(16, (3,3), 1, activation = 'relu', input_shape = (256,256,3)))
model.add(BatchNormalization())
model.add(MaxPooling2D())


model.add(Conv2D(32, (3,3), 1, activation = 'relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D())


model.add(Conv2D(16, (3,3), 1, activation = 'relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D())
```

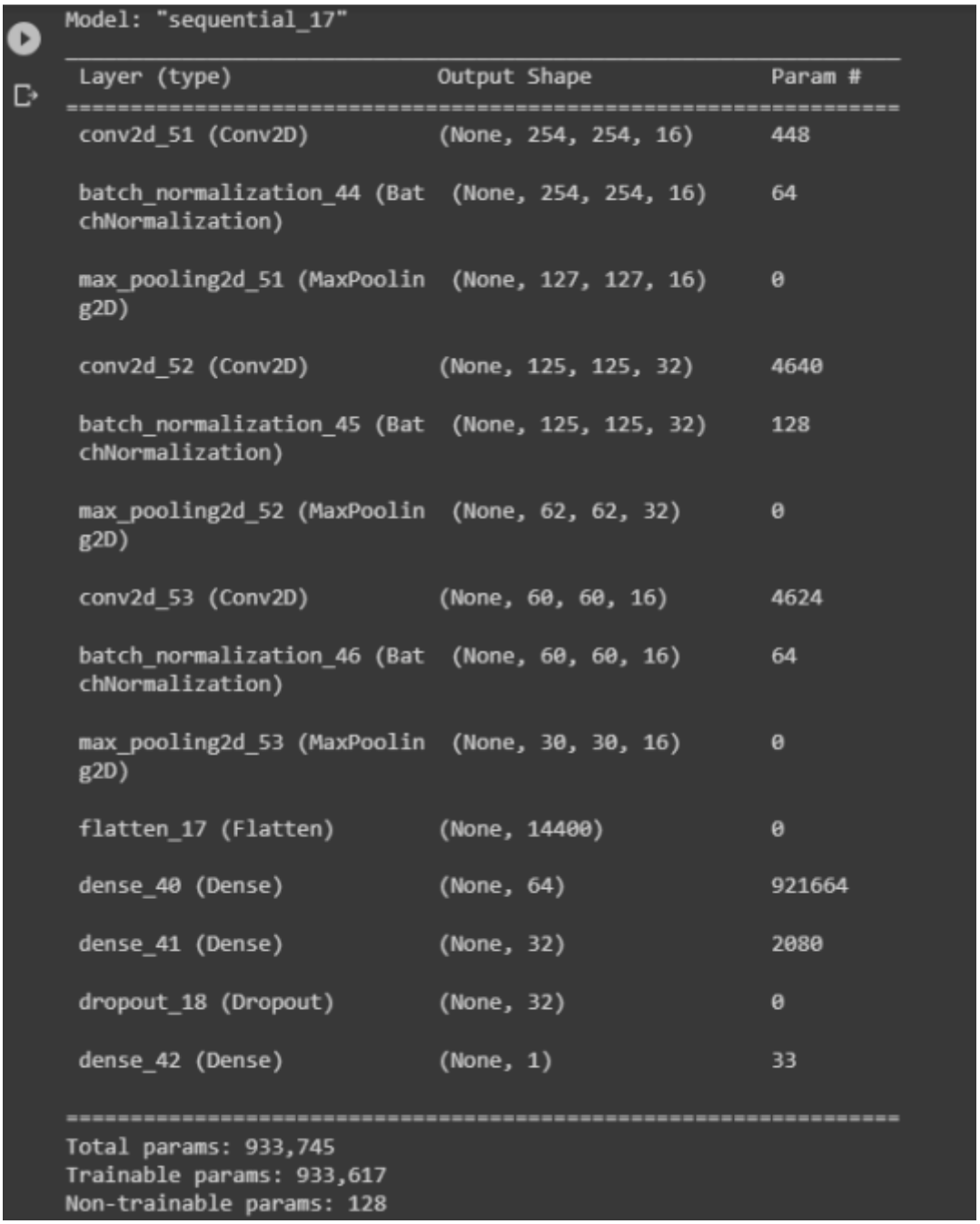
Następnie korzystamy z funkcji Flatten w celu wypłaszczenia danych, co tworzy macierz jednowymiarową rozmiaru 1 x 14400. Następnie dodajemy warstwę neuronową z 64 neuronami z funkcją aktywacji „relu". Następnie dodajemy kolejną warstwę z 32 neuronami. W ostatniej warstwie sieci dodajemy warstwę neuronową z 1 neuronem ponieważ mamy wyłącznie dwie klasy. Funkcją aktywacji ostatniej warstwy jest funkcja „sigmoid". Wybraliśmy ją aby pokazać prawdopodobieństwo przynależności danego obrazu do danej klasy. Pomiędzy warstwami dense dodajemy warstwę dropout aby zerowała nam 0.2 obserwacji, w celu zmniejszenia overfittingu.

```
model.add(Flatten())
model.add(Dense(64, activation = 'relu'))

model.add(Dense(32, activation = 'relu'))

model.add(Dropout(0.2))
model.add(Dense(1, activation = 'sigmoid'))
model.summary()
```

Wywołanie funkcji summary na bazie modelu zwraca podsumowanie modelu postaci:



```
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_51 (Conv2D)          (None, 254, 254, 16)      448

 batch_normalization_44 (Bat  (None, 254, 254, 16)     64
 chNormalization)

 max_pooling2d_51 (MaxPoolin  (None, 127, 127, 16)     0
 g2D)

 conv2d_52 (Conv2D)          (None, 125, 125, 32)      4640

 batch_normalization_45 (Bat  (None, 125, 125, 32)     128
 chNormalization)

 max_pooling2d_52 (MaxPoolin  (None, 62, 62, 32)       0
 g2D)

 conv2d_53 (Conv2D)          (None, 60, 60, 16)        4624

 batch_normalization_46 (Bat  (None, 60, 60, 16)       64
 chNormalization)

 max_pooling2d_53 (MaxPoolin  (None, 30, 30, 16)       0
 g2D)

 flatten_17 (Flatten)        (None, 14400)             0

 dense_40 (Dense)            (None, 64)                921664

 dense_41 (Dense)            (None, 32)                2080

 dropout_18 (Dropout)        (None, 32)                0

 dense_42 (Dense)            (None, 1)                 33

=================================================================
Total params: 933,745
Trainable params: 933,617
Non-trainable params: 128
```

Następnie kompilujemy nasz model wstawiając jako optimizer funkcję optymalizacji „Adam”. Jako funkcję straty wstawiamy funkcję „binary_crossentropy”. Wybraliśmy tą funkcję ze względu na dobre dopasowanie do funkcji „sigmoid”. Jako metrykę ustawiliśmy „accuracy”, w celu zbadania jakości modelu.

Dopasowujemy naszą sieć do danych uczących w czasie 10 epok. Dodajemy early stopping czyli właściwość kontrolująca jakość modelu. Jako parametry funkcji ustawiamy 5 jako wartość parametru patience. Ustawiamy monitor jako „val_loss” aby na tej podstawie wybrać najlepsze wagi. Parametr min_delta ustawiamy na wartość 0.001. Dodajemy checkpoint, który przywraca nam najlepsze wagi oraz zapisuje model do ścieżki podanej w argumencie funkcji.

```
[ ]  early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta = 0.001,
                          patience = 5, restore_best_weights = False)

     checkpoint = tf.keras.callbacks.ModelCheckpoint('/content/drive/MyDrive/data-science-bootcamp',
                          monitor="val_loss", mode="min",
                          save_best_only=True, verbose=1)
```

```
[ ]  opt = tf.keras.optimizers.Adam(learning_rate=0.001)
```

```
[ ]  model.compile(optimizer = opt, loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
[ ]  history = model.fit(train, epochs = 20, validation_data = val,callbacks = [early_stop, checkpoint])
```

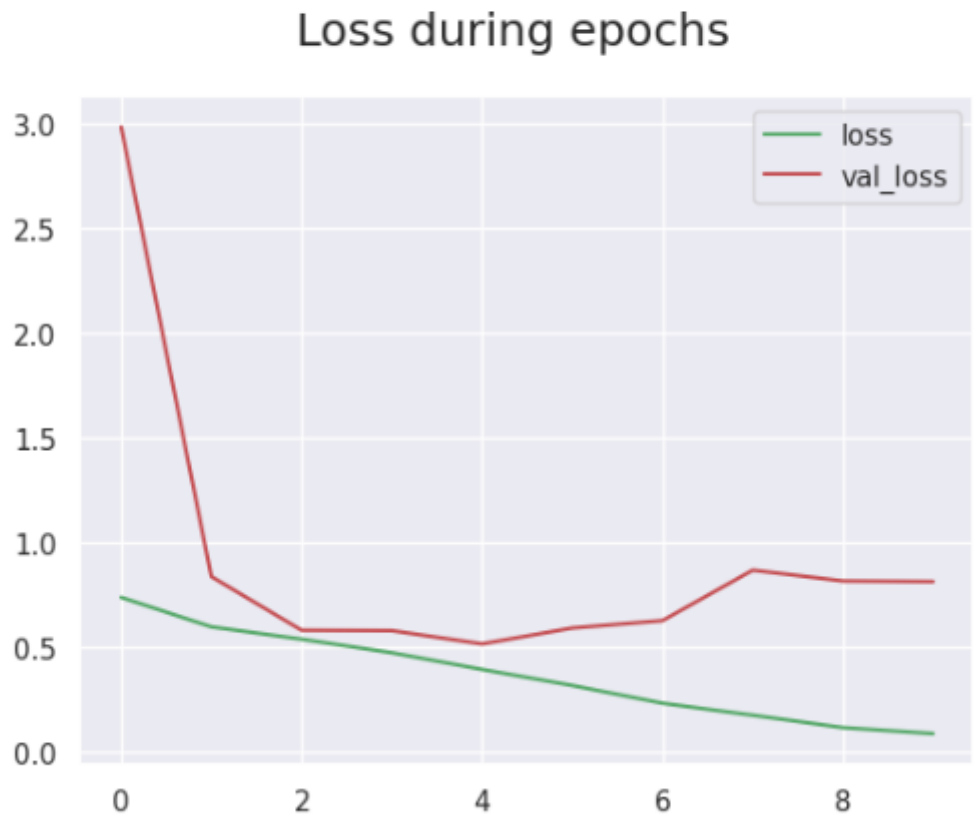
Historia dopasowywania modelu wygląda następująco:

```
Epoch 1/20
219/219 [==============================] - ETA: 0s - loss: 0.7405 - accuracy: 0.5952
Epoch 1: val_loss improved from inf to 2.98544, saving model to /content/drive/MyDrive/data-science-bootcamp
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 3 of 3). These functions will not be directly callable after loading.
219/219 [==============================] - 68s 295ms/step - loss: 0.7405 - accuracy: 0.5952 - val_loss: 2.9854 - val_accuracy: 0.4886
Epoch 2/20
219/219 [==============================] - ETA: 0s - loss: 0.5996 - accuracy: 0.6702
Epoch 2: val_loss improved from 2.98544 to 0.83959, saving model to /content/drive/MyDrive/data-science-bootcamp
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 3 of 3). These functions will not be directly callable after loading.
219/219 [==============================] - 45s 203ms/step - loss: 0.5996 - accuracy: 0.6702 - val_loss: 0.8396 - val_accuracy: 0.5600
Epoch 3/20
219/219 [==============================] - ETA: 0s - loss: 0.5407 - accuracy: 0.7205
Epoch 3: val_loss improved from 0.83959 to 0.58354, saving model to /content/drive/MyDrive/data-science-bootcamp
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 3 of 3). These functions will not be directly callable after loading.
219/219 [==============================] - 67s 304ms/step - loss: 0.5407 - accuracy: 0.7205 - val_loss: 0.5835 - val_accuracy: 0.7014
Epoch 4/20
219/219 [==============================] - ETA: 0s - loss: 0.4760 - accuracy: 0.7725
Epoch 4: val_loss improved from 0.58354 to 0.58158, saving model to /content/drive/MyDrive/data-science-bootcamp
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 3 of 3). These functions will not be directly callable after loading.
219/219 [==============================] - 65s 296ms/step - loss: 0.4760 - accuracy: 0.7725 - val_loss: 0.5816 - val_accuracy: 0.7108
Epoch 5/20
219/219 [==============================] - ETA: 0s - loss: 0.3964 - accuracy: 0.8148
Epoch 5: val_loss improved from 0.58158 to 0.51820, saving model to /content/drive/MyDrive/data-science-bootcamp
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 3 of 3). These functions will not be directly callable after loading.
219/219 [==============================] - 66s 298ms/step - loss: 0.3964 - accuracy: 0.8148 - val_loss: 0.5182 - val_accuracy: 0.7654
Epoch 6/20
219/219 [==============================] - ETA: 0s - loss: 0.3199 - accuracy: 0.8557
Epoch 6: val_loss did not improve from 0.51820
219/219 [==============================] - 64s 290ms/step - loss: 0.3199 - accuracy: 0.8557 - val_loss: 0.5941 - val_accuracy: 0.7540
Epoch 7/20
219/219 [==============================] - ETA: 0s - loss: 0.2351 - accuracy: 0.9014
Epoch 7: val_loss did not improve from 0.51820
219/219 [==============================] - 64s 289ms/step - loss: 0.2351 - accuracy: 0.9014 - val_loss: 0.6293 - val_accuracy: 0.7505
Epoch 8/20
219/219 [==============================] - ETA: 0s - loss: 0.1778 - accuracy: 0.9308
Epoch 8: val_loss did not improve from 0.51820
219/219 [==============================] - 44s 197ms/step - loss: 0.1778 - accuracy: 0.9308 - val_loss: 0.8702 - val_accuracy: 0.7346
Epoch 9/20
219/219 [==============================] - ETA: 0s - loss: 0.1188 - accuracy: 0.9545
Epoch 9: val_loss did not improve from 0.51820
219/219 [==============================] - 43s 195ms/step - loss: 0.1188 - accuracy: 0.9545 - val_loss: 0.8187 - val_accuracy: 0.7490
Epoch 10/20
219/219 [==============================] - ETA: 0s - loss: 0.0899 - accuracy: 0.9665
Epoch 10: val_loss did not improve from 0.51820
219/219 [==============================] - 62s 281ms/step - loss: 0.0899 - accuracy: 0.9665 - val_loss: 0.8159 - val_accuracy: 0.7892
```

Ocena modelu

Na bazie zmiennej history stworzyliśmy wykresy obrazujące funkcje straty w czasie 20 epok na zbiorze uczącym i zbiorze walidacyjnym.

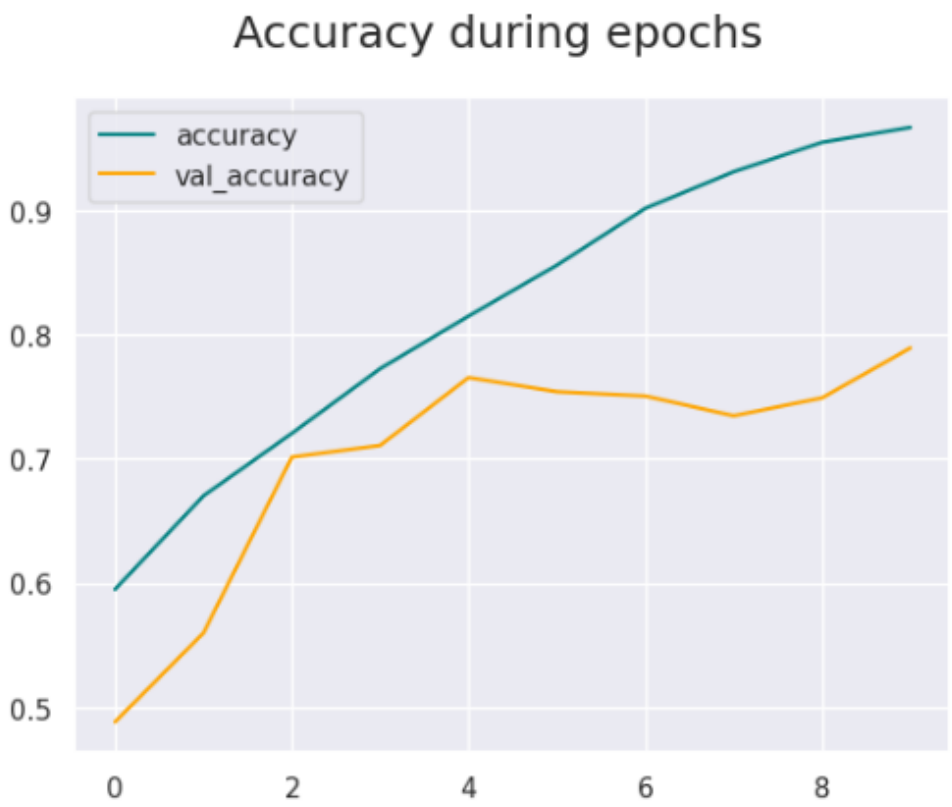
```
[ ]  fig = plt.figure()
     plt.plot(history.history['loss'], color = 'g', label = 'loss')
     plt.plot(history.history['val_loss'], color = 'r', label = 'val_loss')
     fig.suptitle('Loss during epochs', fontsize = 18)
     plt.legend(loc = 'upper right')
     plt.show()
```



Jak widać na powyższym wykresie funkcja straty malała do 4 epoki osiągając wartość ok. 0.52. Jest to zjawisko nietypowe, ponieważ wraz z epokami val_accuracy rośnie i val_loss również. Sprawdzaliśmy czy było to spowodowane learning rate, jednak nie był to powód tego zjawiska.

Ponadto na bazie zmiennej history wykonaliśmy wykres przedstawiający jakość modelu na zbiorze uczącym i zbiorze walidacyjnym.

```
[ ]  fig = plt.figure()
     plt.plot(history.history['accuracy'], color = 'teal', label = 'accuracy')
     plt.plot(history.history['val_accuracy'], color = 'orange', label = 'val_accuracy')
     fig.suptitle('Accuracy during epochs', fontsize = 18)
     plt.legend(loc = 'upper left')
     plt.show()
```



Wnioskując po powyższym wykresie możemy stwierdzić że metryka dokładności zwiększała się wraz z epokami. Podobnie jak w przypadku val_loss, mogłoby to świadczyć o overfittingu modelu, aczkolwiek użyliśmy paru technik regularyzacji oraz normalizacji w celu poprawy jakości modelu co częściowo udało nam się osiągnąć.

Na zbiorze testowym testowaliśmy naszą sieć konwolucyjną trzema różnymi metrykami Precision, Recall oraz BinaryAccuracy. Wszystkie metryki dają wynik powyżej 0.7 co świadczy o dobrym dopasowaniu modelu do danych.

```
[ ] from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy
```

```
[ ] pre = Precision()
    re = Recall()
    acc = BinaryAccuracy()
```

Testing our neural network

```
[ ] for batch in test.as_numpy_iterator():
      X,y = batch
      yhat = model.predict(X)
      pre.update_state(y, yhat)
      re.update_state(y, yhat)
      acc.update_state(y, yhat)
```

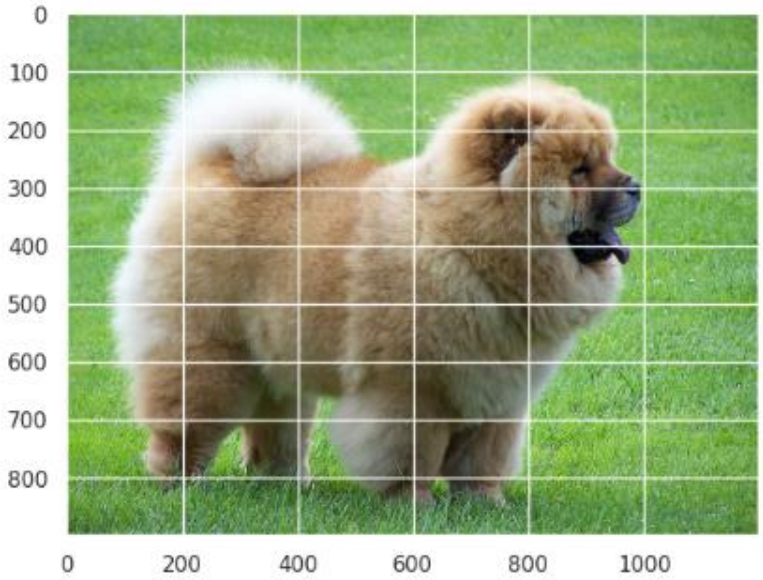
```
1/1 [==============================] - 0s 228ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 120ms/step
```

```
[ ] print(f'Precision: {pre.result()}, Recall: {re.result()}, Accuracy: {acc.result()}')
```

```
Precision: 0.7316103577613831, Recall: 0.7682672142982483, Accuracy: 0.7547357678413391
```

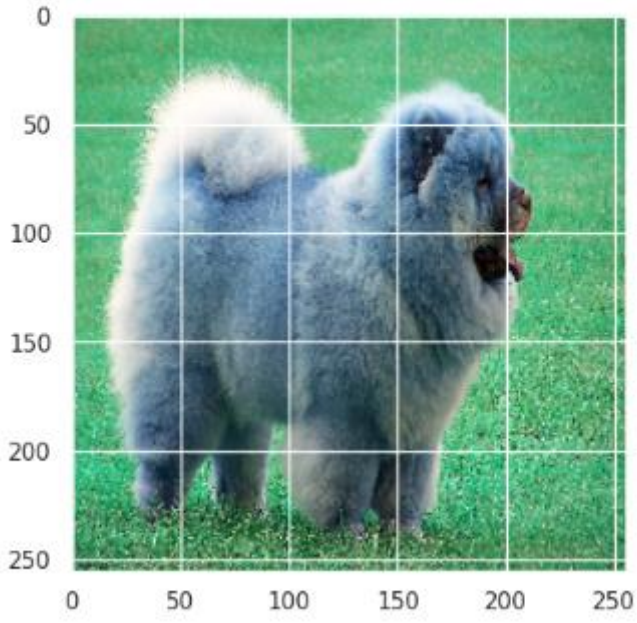
W ostatnim etapie projektu dokonaliśmy testowania naszego modelu poprzez analizę wyniku z obrazu z poza zbioru danych. Ładowanie testu oraz sam obraz prezentuję się następująco:

```
[ ]  img = cv2.imread('/content/drive/MyDrive/data-science-bootcamp/1200px-ChowChow2Szczecin.jpg')
     plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
     plt.show()
```



Ustawiamy rozmiar zdjęcia do rozmiaru zadanego w naszej sieci konwolucyjnej.

```
[ ]  resize = tf.image.resize(img, (256,256))
     plt.imshow(resize.numpy().astype(int))
     plt.show()
```



Przed analizowaniem zdjęcia ponadto trzeba zmienić mu wymiar w celu dopasowania do innych tensorów wykorzystanych w sieci.

```
[ ]  yhat = model.predict(np.expand_dims(resize/255, 0))
```

```
1/1 [==============================] - 0s 28ms/step
```

```
[ ]  yhat
```

```
array([[0.93173856]], dtype=float32)
```

```
[ ]  if yhat > 0.5:
       print('Dog')
     else:
       print('Cat')
```

```
Dog
```

Jeżeli prawdopodobieństwo jest większe niż 0.5 to zwierzę klasyfikowane jest jako pies a poniżej poziomu 0.5 jako kot.