Psy i koty

... czyli klasyfikacja obrazów i manipulacja dużą ilością danych

Karolina Kotlowska

13.1. Pobieramy dane

Załadujemy zbiór danych z kolekcji TensorFlow Datasets (https://www.tensorflow.org/datasets). Zbiór jest widoczny jako obiekt klasy `tf.data.Dataset` zapewniającej funkcjonalność podobną do strumieni.

Przy pierwszym uruchomieniu dane zostaną pobrane. Później będzie można je czytać wielokrotnie.

In [5]:

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds

# setattr(tfds.image_classification.cats_vs_dogs, '_URL',"https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-8368-6DEBA77B919F/kagglecatsanddogs_5340.zip")
ds = tfds.load('cats_vs_dogs', split='train', batch_size=None,as_supervised=True)
```

Liczba obrazów to:

In [6]:

```
len(ds)
```

Out[6]:

```
23262
```

13.1.1 Wyświetlmy przykładowe obrazy

Przed wyświetleniem zostaną przeskalowane do tych samych rozmiarów

In [7]:

```
plt.rcParams["figure.figsize"] = (10,10)
it = ds.as_numpy_iterator()
labels=['Cat','Dog']
input_size=100
for i in range(9):
  ax = plt.subplot(330 + 1 + i)
  image, label = next(it)
  image = tf.image.resize(image, size=[256,256], method=tf.image.ResizeMethod.NE
AREST_NEIGHBOR)
  ax.set_title(labels[label])
  plt.imshow(image)
plt.show()
```

13.1.2 Załadujemy obrazy i zapiszemy w tablicach numpy

Obraz jest trójwymiarową tablicą (szerokość, wysokość, kanały koloru). Aby umieścić wszystkie obrazy w jednej tablicy 4D musza mieć identyczne rozmiary.

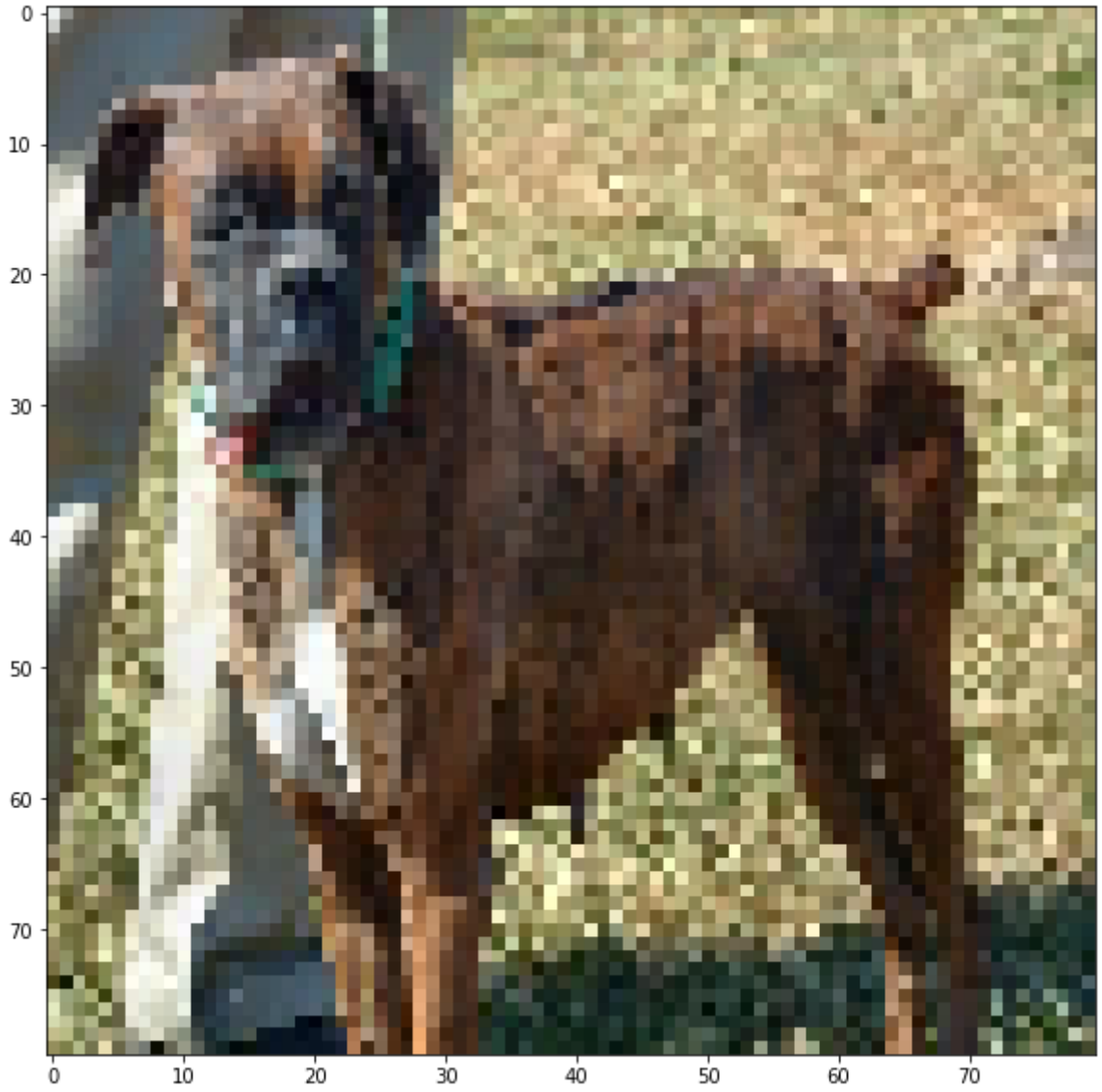
In [8]:

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds
# from keras.preprocessing.image import img_to_array
from tensorflow.keras.utils import img_to_array
from tqdm import tqdm

def load_images_to_numpy(name,resize_to=[80,80]):
  ds = tfds.load(name, split='train', batch_size=None,as_supervised=True)
  images = []
  labels=[]
  for image, label in tqdm(ds):
    image = tf.image.resize(image, size=resize_to, method=tf.image.ResizeMethod.
NEAREST_NEIGHBOR)
    image = img_to_array(image)/255
    images.append(image)
    labels.append(label.numpy())

  # return images,labels
  X=np.array(images)
  y=np.array(labels)
  return X,y

X,y = load_images_to_numpy('cats_vs_dogs')

print(X.shape)
print(y.shape)
```

```
100%|██████████| 23262/23262 [00:58<00:00, 400.21it/s]

(23262, 80, 80, 3)
(23262,)
```

In [9]:

```
plt.imshow(X[0]);
```



13.2. Klasyfikacja zbioru załadowanego do pamięci

Wypróbujemy kilka klasyfiaktorów z biblioteki sklearn. Przedtem musimy spłaszczyć X do postaci dwuwymiarowej tablicy.

Nie oczekujemy tu dobrych wyników, ponieważ standardowe algorytmy traktują obraz, jak długi wektor pikeseli, a nie biorą pod uwagę cech lokalnych.

Gdyby przeskalować obraz do rozmiarów 4x4 otrzymalibyśmy informacje o kolorach w różnych regionach obrazu. Może pozwoliłoby to odróżnić krajobraz morski i lądowy, ale nie psy i koty.

In [10]:

```
X = np.reshape(X,(X.shape[0],-1))
print(X.shape)
```

```
(23262, 19200)
```

TODO 13.2.1

- Wypróbuj wpierw GaussianNB, a potem co najmniej 3 wybrane kolejne klasyfikatory
- Jaka jest efektywność klasyfikacji?
- Co się dzieje z pamięcią
- Zestaw informacje w postaci tabeli

In [11]:

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
import sklearn

clf = GaussianNB()
# clf = RandomForestClassifier(n_estimators=100,verbose=1)
# clf = MultinomialNB()
# clf = LogisticRegression()
# clf = KNeighborsClassifier(n_neighbors=5) #umieszcza wszystkie dane w pamięci
- zużywa bardzo pamięć
# clf = SGDClassifier(loss='log')
# clf = SVC(kernel='rbf',gamma='auto')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, rando
m_state = 123)
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print(f'Accuracy:{sklearn.metrics.accuracy_score(y_test,y_pred)}')
print(f'F1:{sklearn.metrics.f1_score(y_test,y_pred,average="macro")}')
```

```
Accuracy:0.5612376450365277
F1:0.5605205089886541
```

In [12]:

```
clf = RandomForestClassifier(n_estimators=100,verbose=1)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, rando
m_state = 123)
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print(f'Accuracy:{sklearn.metrics.accuracy_score(y_test,y_pred)}')
print(f'F1:{sklearn.metrics.f1_score(y_test,y_pred,average="macro")}')
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurr
ent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:  4.5min finishe
d
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurr
ent workers.

Accuracy:0.6403094112591319
F1:0.6396961880884169

[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.3s finishe
d
```

In [13]:

```
clf = LogisticRegression()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, rando
m_state = 123)
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print(f'Accuracy:{sklearn.metrics.accuracy_score(y_test,y_pred)}')
print(f'F1:{sklearn.metrics.f1_score(y_test,y_pred,average="macro")}')
```

```
c:\Users\krzyc\anaconda3-1\lib\site-packages\sklearn\linear_model\_l
ogistic.py:814: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as sh
own in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver option
s:
    https://scikit-learn.org/stable/modules/linear_model.html#logist
ic-regression
  n_iter_i = _check_optimize_result(

Accuracy:0.5762784701332188
F1:0.5762608630130106
```

In [14]:

```
clf = KNeighborsClassifier(n_neighbors=5)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, rando
m_state = 123)
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print(f'Accuracy:{sklearn.metrics.accuracy_score(y_test,y_pred)}')
print(f'F1:{sklearn.metrics.f1_score(y_test,y_pred,average="macro")}')
```

```
Accuracy:0.5487752470992694
F1:0.5253455952630719
```

Najlepiej sprawdził się RandomForestClassifier.

Trenowanie pochłonęło sporo pamięci. KNeighborsClassifier umieszcza wszystkie dane w pamięci - zużywa bardzo pamięć.

Model	Accuracy	F1
GaussianNB	0.5612376	0.5605205
RandomForestClassifier	0.6438633	0.6330582
LogisticRegression	0.5758487	0.5758108
KNeighborsClassifier	0.5487752	0.5253456

13.3. Partial fit

Zbiór wszystkich obrazów załadowany do pamięci zużywa około 30% pamięci środowiska Colab. Wydzielenie `X_train` i `X_test` to kolejne 30%. W przypadku niektórych klasyfikatorów brak jest miejsca na model...

W przypadku dużych zbiorów danych rozwiązanie może polegać na wykonaniu pętli, w której

- odczytuje się fragment danych (batch) ze źródła
- dane te są uzyte do uczenia (uaktualnienia modelu)

W biblotece sklearn wybrane klasyfikatory i algorytmy regresji mają służącą do tego metodę `partial_fit()`.

13.3.1 Organizacja odczytu danych

- Obrazy będą czytane z dysku
- Podczas odczytu dokonywana będzie konwersja za pomocą funkcji img_convert

In [15]:

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds
# from keras.preprocessing.image import img_to_array
from tensorflow.keras.utils import img_to_array

from tqdm import tqdm

dataset = tfds.load('cats_vs_dogs', split='train', batch_size=None,as_supervised
=True)


def img_convert(image,label):
  image = tf.image.resize(image, size=[80,80], method=tf.image.ResizeMethod.NEAR
EST_NEIGHBOR)
  image = image / 255
  return (image,label)


dataset = dataset.map(img_convert)
dataset = dataset.shuffle(buffer_size=100)
```

13.3.2 Podział na strumień danych uczących i testowych

Jest to podział sztywny. Funkcje take() i skip() przesuwają granice iteracji, ale nie zapewniają losowego wyboru.

Strumień (dataset) może losowo zmieniać kolejność danych, ale stosując bufor wewnętrzny o pewnym zadanym rozmiarze.

In [16]:

```
train_size = int(0.7*len(dataset))

ds_train = dataset.take(train_size)
ds_test = dataset.skip(train_size)
print(len(ds_train),len(ds_test))

ds_test = ds_test.batch(256, drop_remainder=False)
ds_train = ds_train.batch(128, drop_remainder=False)
```

```
16283 6979
```

13.3.3 Iteracja po danych

In [17]:

```
for batch,labels in  ds_train:
  X=batch.numpy()
```

13.3.4 Opcjonalnie: zawartość dataset może zostać zwielokrotniona

Można wówczas wielokrotnie iterować przez te same elementy. Wielokrotna iteracja ma sens, dla modeli wyznaczanych w procesie optymalizacji (np. za pomocą metod gradientowych.) Wówczas przy równoczensym zastosowaniu losowania
możliwe jest opuszczenie minimum lokalnego funkcji kosztu.

In [18]:

```
epochs=2
ds_train=ds_train.repeat(epochs)
```

13.3.5 Klasyfikacja

TODO 13.3.1

- Proszę wyznaczyć X na podstawie danych w zmiennej batch oraz y na podstawie labels. X nalezy spłaszczyć do tablicy 2-wymiarowej.
- Dla których (poniższych klasyfikatorów) wielokrotne przetwarzanie ('ogladanie') tych samych danych danych może mieć sens?
- Wypróbuj i porównaj **dwa klasyfikatory** implementujące metodę `partial_fit()` . Jaka jest ich efektywność?

In [19]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import PassiveAggressiveClassifier


from tqdm import tqdm

import sklearn

# clf = MultinomialNB()
# clf = GaussianNB()
# clf = PassiveAggressiveClassifier()
clf = SGDClassifier(loss='perceptron')


# for i,(batch,labels) in tqdm( enumerate(ds_train) ):
for i,(batch,labels) in enumerate(ds_train) :
  X=batch.numpy()
  X = np.reshape(X,(X.shape[0],-1))
  y=labels.numpy()
  clf.partial_fit(X,y,classes=[0,1])


# predict

y_true_list = []
y_pred_list = []
for i, (batch, labels) in enumerate(ds_test):
    X = batch.numpy()
    X = np.reshape(X, (X.shape[0], -1))
    y = labels.numpy()
    y_pred = clf.predict(X)
    y_true_list.extend(y.tolist())
    y_pred_list.extend(y_pred.tolist())

y_true = np.hstack(y_true_list)
# y_true.shape

y_pred=np.hstack(y_pred_list)
# y_pred.shape

print(f'Accuracy:{sklearn.metrics.accuracy_score(y_true,y_pred)}')
print(f'F1:{sklearn.metrics.f1_score(y_true,y_pred,average="macro")}')
print(f'auroc:{sklearn.metrics.roc_auc_score(y_true,y_pred,average="macro")}')
```

```
Accuracy:0.5929216220088838
F1:0.5900312108139593
auroc:0.5932807726421603
```

In [20]:

```
clf = MultinomialNB()

# for i,(batch,labels) in tqdm( enumerate(ds_train) ):
for i,(batch,labels) in enumerate(ds_train) :
  X=batch.numpy()
  X = np.reshape(X,(X.shape[0],-1))
  y=labels.numpy()
  clf.partial_fit(X,y,classes=[0,1])

# predict

y_true_list = []
y_pred_list = []
for i, (batch, labels) in enumerate(ds_test):
    X = batch.numpy()
    X = np.reshape(X, (X.shape[0], -1))
    y = labels.numpy()
    y_pred = clf.predict(X)
    y_true_list.extend(y.tolist())
    y_pred_list.extend(y_pred.tolist())

y_true = np.hstack(y_true_list)
# y_true.shape

y_pred=np.hstack(y_pred_list)
# y_pred.shape

print(f'Accuracy:{sklearn.metrics.accuracy_score(y_true,y_pred)}')
print(f'F1:{sklearn.metrics.f1_score(y_true,y_pred,average="macro")}')
print(f'auroc:{sklearn.metrics.roc_auc_score(y_true,y_pred,average="macro")}')
```

```
Accuracy:0.5601088981229403
F1:0.5600167487263904
auroc:0.5600504061002419
```

In [21]:

```
clf = PassiveAggressiveClassifier()

# for i,(batch,labels) in tqdm( enumerate(ds_train) ):
for i,(batch,labels) in enumerate(ds_train) :
  X=batch.numpy()
  X = np.reshape(X,(X.shape[0],-1))
  y=labels.numpy()
  clf.partial_fit(X,y,classes=[0,1])

# predict

y_true_list = []
y_pred_list = []
for i, (batch, labels) in enumerate(ds_test):
    X = batch.numpy()
    X = np.reshape(X, (X.shape[0], -1))
    y = labels.numpy()
    y_pred = clf.predict(X)
    y_true_list.extend(y.tolist())
    y_pred_list.extend(y_pred.tolist())

y_true = np.hstack(y_true_list)
# y_true.shape

y_pred=np.hstack(y_pred_list)
# y_pred.shape

print(f'Accuracy:{sklearn.metrics.accuracy_score(y_true,y_pred)}')
print(f'F1:{sklearn.metrics.f1_score(y_true,y_pred,average="macro")}')
print(f'auroc:{sklearn.metrics.roc_auc_score(y_true,y_pred,average="macro")}')
```

```
Accuracy:0.5406218655967904
F1:0.46612554829406794
auroc:0.5420751471613843
```

Porównanie klasyfikatorów:

Model	Accuracy	F1 Score	AUROC
SGDClassifier	0.5002	0.3390	0.5020
MultinomialNB	0.5597	0.5596	0.5596
PassiveAggressiveClassifier	0.5406	0.4661	5420

13.3.6 Sprawdźmy metryki dla zbioru uczącego

In [22]:

```
y_true_list=[]
y_pred_list=[]
for i,(batch,labels) in enumerate(ds_train):
  X=batch.numpy()
  X = np.reshape(X,(X.shape[0],-1))
  y=labels.numpy()
  y_pred = clf.predict(X)
  y_true_list.append(y)
  y_pred_list.append(y_pred)

y_true = np.hstack(y_true_list)
# y_true.shape

y_pred=np.hstack(y_pred_list)
y_pred.shape

print(f'Accuracy:{sklearn.metrics.accuracy_score(y_true,y_pred)}')
print(f'F1:{sklearn.metrics.f1_score(y_true,y_pred,average="macro")}')
print(f'auroc:{sklearn.metrics.roc_auc_score(y_true,y_pred,average="macro")}')
```

```
Accuracy:0.5642080697660137
F1:0.49360421839379365
auroc:0.5645985227193009
```

In [23]:

```
from sklearn.metrics import confusion_matrix
conf_mat = confusion_matrix(y_true, y_pred)
print(conf_mat)
```

```
[[ 3107 13193]
 [  999 15267]]
```

13.4. Sieć neuronowa

Zbudujemy prostą sieć neuronową korzystającą z komponentów platformy TensorFlow/keras. Sieć składa się z:

- warstwy wejściowej (takiej jak rozmiary obrazu, czyli [100,100,3])
- warstwy spłaszczającej do postaci wektora [300000]
- warstwy ukrytej liczącej 100 neuronów z funkjcą aktywacji relu (https://en.wikipedia.org/wiki/Rectifier_(neural_networks))
- warstwy wyjściowej z dwoma neuronami i funkcją aktywacji softmax (https://en.wikipedia.org/wiki/Softmax_function)

> Liczba neuronów wyjściowych odpowiada liczbie klas. Odczytujemy z nich prawdopodobieństwo i wybieramy większe z nich.
>
> Alternatywna konfiguracja w przypadku klasyfikacji binarnej to jeden neuron wyjściowy z funkcją aktywacji sigmoid. Przypomina to regresję logistyczną. W takim przypadku wybieramy 0 dla prawdopodobieństwa p < 0.5 oraz 1 dla p>=0.5.
>
> Funkcją kosztu jest *crossentropy* - odpowiednik *logloss* znanej z regresji logistycznej.
>
> **Uwaga** Podobnie jak zwykłe klasyfikatory - to nie jest architektura odpowiednia do klasyfikacji obrazów.

In [24]:

```
from keras.layers import Input, Dense, Flatten
from keras import Model

def model_builder(input_shape,labels_count):
  model_input = Input(input_shape)
  x= Flatten()(model_input)
  x = Dense(input_shape[1],activation = 'relu')(x)
  x = Dense(labels_count,activation='softmax')(x)
  model = Model(model_input, x, name='simple')
  model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', me
trics=['accuracy'] )
  return model

model = model_builder((80,80,3),2)
```

Wyświetlmy informacje o modelu. Ma dużo parametrów, ponad 3 000 000.

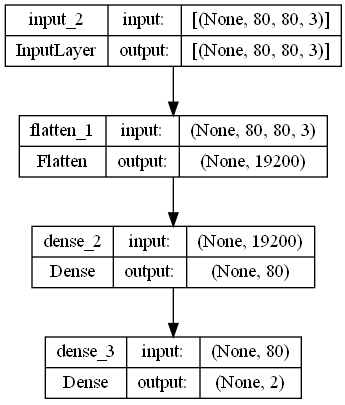
Wejście ma rozmiar [None, 100, 100, 3] Podczas uczenia to None zostanie zastąpione wymiarami wsadu batch (tu 128).

In [25]:

```
from keras.utils import plot_model

model.summary()
print()
plot_model(model, show_shapes=True, show_layer_names=True, to_file='model.png')
from IPython.display import Image
Image(retina=True, filename='model.png',width = 1000, height = 1000)
```

```
Model: "simple"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 80, 80, 3)]       0

 flatten_1 (Flatten)         (None, 19200)             0

 dense_2 (Dense)             (None, 80)                1536080

 dense_3 (Dense)             (None, 2)                 162

=================================================================
Total params: 1,536,242
Trainable params: 1,536,242
Non-trainable params: 0
_________________________________________________________________
```

Out[25]:



Uczymy sieć.

- Każda epoka to jednokrotna iteracja po zbiorze danych.
- Podział na wsady (batch) jest automatycznie dokonywany przez strumień
- Podczas czytania batcha kolejne obrazy zostaną załadowane z dysku, przeskalowane i umieszczone w pamięci
- Mniej więcej w podobnym czasie poprzedni batch powinien być przetworzony na GPU
- Po każdej epoce następuje walidacja na zbiorze testowym. To przedłuża czas przetwarzania epoki (ale umożliwia przygotowanie wykresów)
- Niestety kazda epoka trwa około 37-38 sekund

In [26]:

```
history = model.fit(ds_train, epochs=20, batch_size=128,verbose=1,validation_dat
a=ds_test)
```

```
Epoch 1/20
256/256 [==============================] - 20s 76ms/step - loss: 1.0
472 - accuracy: 0.5412 - val_loss: 0.7275 - val_accuracy: 0.5641
Epoch 2/20
256/256 [==============================] - 20s 79ms/step - loss: 0.7
990 - accuracy: 0.5818 - val_loss: 0.8316 - val_accuracy: 0.5350
Epoch 3/20
256/256 [==============================] - 21s 82ms/step - loss: 0.7
135 - accuracy: 0.6109 - val_loss: 0.7920 - val_accuracy: 0.5528
Epoch 4/20
256/256 [==============================] - 22s 87ms/step - loss: 0.6
978 - accuracy: 0.6165 - val_loss: 0.6954 - val_accuracy: 0.6025
Epoch 5/20
256/256 [==============================] - 21s 81ms/step - loss: 0.6
431 - accuracy: 0.6433 - val_loss: 0.6824 - val_accuracy: 0.6075
Epoch 6/20
256/256 [==============================] - 21s 82ms/step - loss: 0.6
481 - accuracy: 0.6435 - val_loss: 0.7245 - val_accuracy: 0.5938
Epoch 7/20
256/256 [==============================] - 21s 81ms/step - loss: 0.6
282 - accuracy: 0.6550 - val_loss: 0.7575 - val_accuracy: 0.5777
Epoch 8/20
256/256 [==============================] - 20s 76ms/step - loss: 0.6
428 - accuracy: 0.6499 - val_loss: 0.7445 - val_accuracy: 0.5797
Epoch 9/20
256/256 [==============================] - 19s 75ms/step - loss: 0.6
239 - accuracy: 0.6563 - val_loss: 0.6400 - val_accuracy: 0.6340
Epoch 10/20
256/256 [==============================] - 20s 76ms/step - loss: 0.6
047 - accuracy: 0.6656 - val_loss: 0.6826 - val_accuracy: 0.6114
Epoch 11/20
256/256 [==============================] - 20s 76ms/step - loss: 0.6
005 - accuracy: 0.6739 - val_loss: 0.6663 - val_accuracy: 0.6161
Epoch 12/20
256/256 [==============================] - 19s 75ms/step - loss: 0.5
963 - accuracy: 0.6784 - val_loss: 0.6886 - val_accuracy: 0.6164
Epoch 13/20
256/256 [==============================] - 20s 77ms/step - loss: 0.5
793 - accuracy: 0.6915 - val_loss: 0.6927 - val_accuracy: 0.6146
Epoch 14/20
256/256 [==============================] - 21s 83ms/step - loss: 0.5
911 - accuracy: 0.6823 - val_loss: 0.7020 - val_accuracy: 0.6080
Epoch 15/20
256/256 [==============================] - 21s 80ms/step - loss: 0.5
691 - accuracy: 0.7007 - val_loss: 0.6725 - val_accuracy: 0.6207
Epoch 16/20
256/256 [==============================] - 20s 78ms/step - loss: 0.5
646 - accuracy: 0.7046 - val_loss: 0.6459 - val_accuracy: 0.6414
Epoch 17/20
256/256 [==============================] - 20s 77ms/step - loss: 0.5
543 - accuracy: 0.7131 - val_loss: 0.6516 - val_accuracy: 0.6371
Epoch 18/20
256/256 [==============================] - 20s 77ms/step - loss: 0.5
451 - accuracy: 0.7211 - val_loss: 0.7432 - val_accuracy: 0.6025
Epoch 19/20
256/256 [==============================] - 20s 79ms/step - loss: 0.5
444 - accuracy: 0.7202 - val_loss: 0.6549 - val_accuracy: 0.6383
Epoch 20/20
256/256 [==============================] - 20s 76ms/step - loss: 0.5
384 - accuracy: 0.7245 - val_loss: 0.7738 - val_accuracy: 0.5991
```

TODO 13.4.1

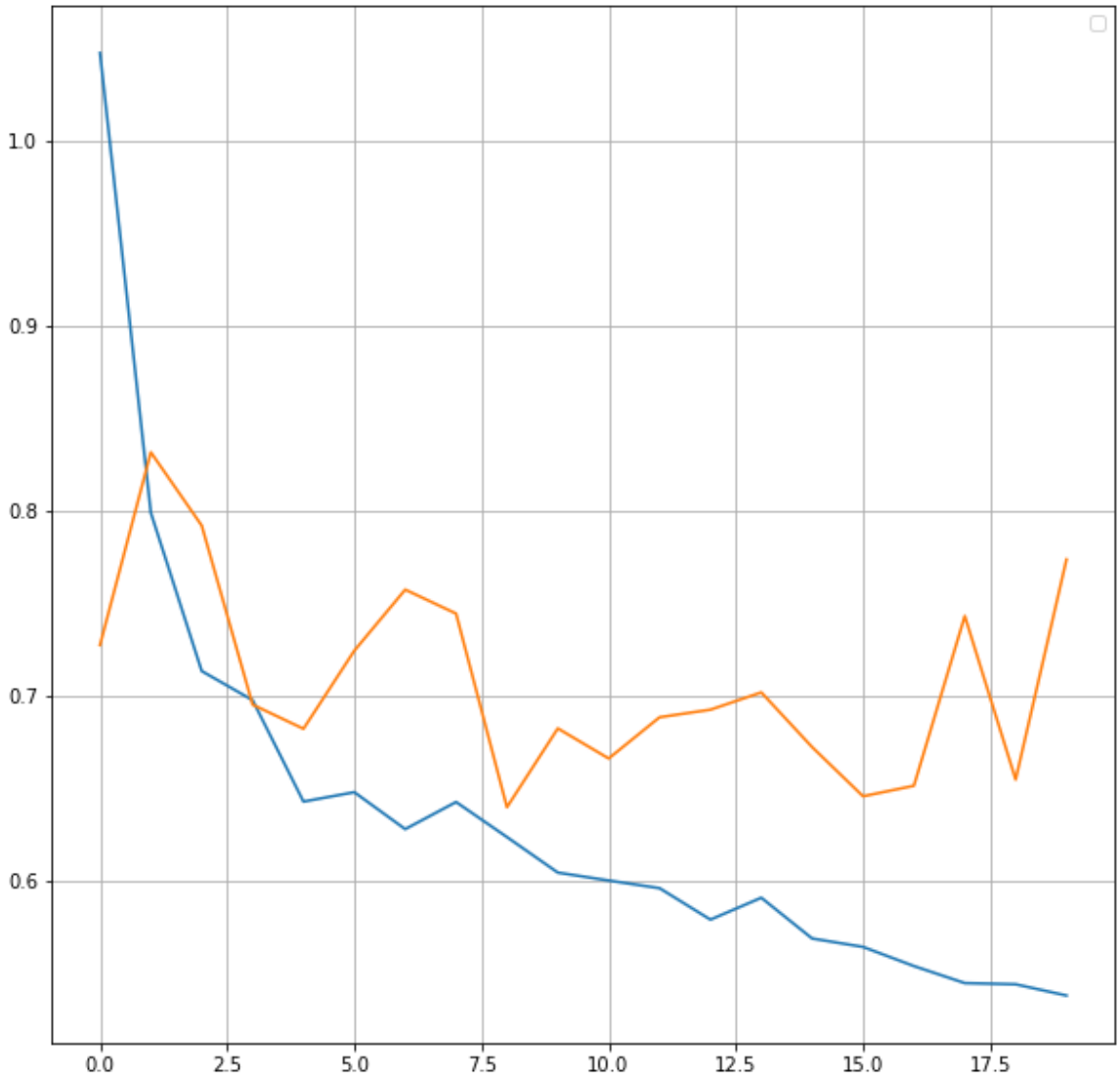
- Wyświetl wykresy z danymi zawartymi w history

In [27]:

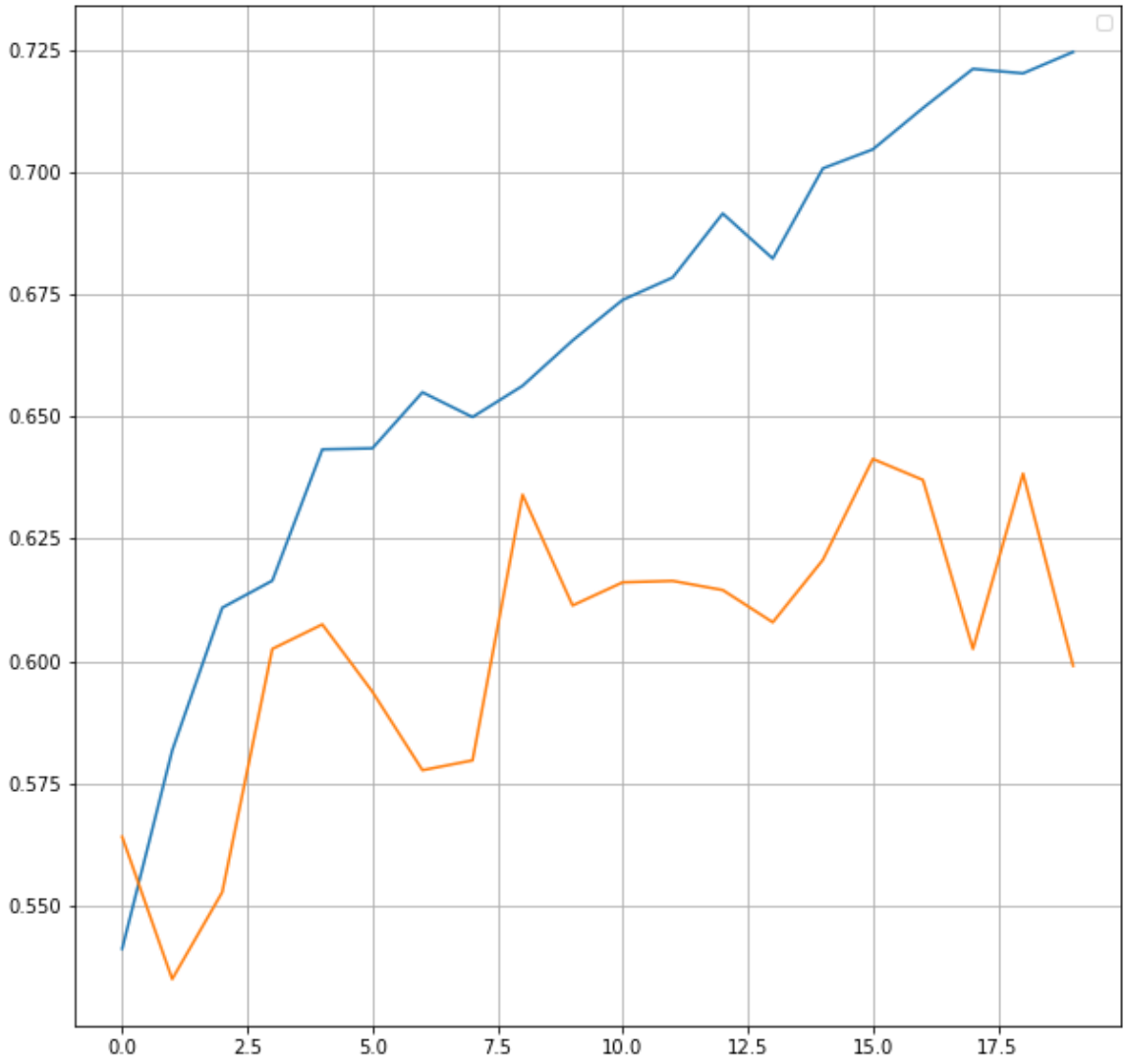
```
print(history.history.keys())
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend()
plt.grid()
plt.show()

plt.figure()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend()
plt.grid()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in leg
end.  Note that artists whose label start with an underscore are ign
ored when legend() is called with no argument.
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```




```
WARNING:matplotlib.legend:No artists with labels found to put in leg
end.  Note that artists whose label start with an underscore are ign
ored when legend() is called with no argument.
```



13.5. Zbiór danych w pamieci

Spróbujemy przetwarzać zbiór danych w pamięci. To powinno przyspieszyć. Colab ma słabe CPU i kod wykonywany po stronie hosta działa wolno.

Zresetuj środowisko wykonawcze

13.5.1 Funkcja ładująca zbiór danych do macierzy numpy

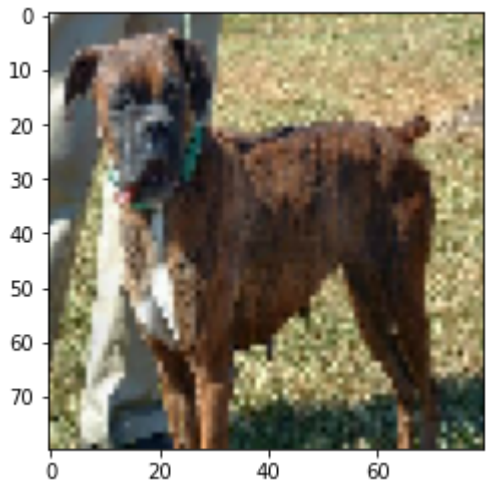
In [2]:

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow.keras.utils import img_to_array
from tqdm import tqdm

def load_images_to_numpy(name,resize_to=[80,80]):
  ds = tfds.load('cats_vs_dogs', split='train', batch_size=None,as_supervised=Tr
ue)
  images = []
  labels=[]
  for image, label in tqdm(ds):
    image = tf.image.resize(image, size=resize_to, method=tf.image.ResizeMethod.
NEAREST_NEIGHBOR)
    image = img_to_array(image)/255
    images.append(image)
    labels.append(label.numpy())

  # return images,labels
  X=np.array(images)
  y=np.array(labels)
  return X,y
```

In [3]:

```
X,y = load_images_to_numpy('cats_vs_dogs')
plt.imshow(X[0]);
```

```
100%|██████████| 23262/23262 [00:50<00:00, 460.19it/s]
```



13.5.2 Sprawdzamy ile mamy pamięci

In [4]:

```
import psutil
psutil.virtual_memory()
```

Out[4]:

```
svmem(total=16977661952, available=2503389184, percent=85.3, used=14
474272768, free=2503389184)
```

13.5.3 Duża monolityczna funkcja do ucznia modeli i ich oceny

Napiszemy funkcję, która:

1. Wydzieli zbiór uczący i testowy
2. Wywoła zewnętrzną funkcję do budowy modelu
3. Przeprowadzi uczenie z walidacją
4. Przetestuje model na zbiorze testowym
5. Wypisze metryki

TODO 13.5.1

- Ad 1. Podziel dane w proporcjach określonych przez test_size za pomocą train_test_split
- Ad 3. Wywołaj model.fit
- Ad 3. Wyświetl historię
- Ad 4. Wyznacz przewidywane etykiety y_pred
- Ad 5. Oblicz metryki

In [5]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
ore
import matplotlib.pyplot as plt
import psutil

def split_train_test(X, y, model_builder, epochs=20, test_size=0.3, batch_size=1
28, use_validation_data=True, display_history=True):
    print(psutil.virtual_memory())
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_siz
e)

    ds_train = tf.data.Dataset.from_tensor_slices((X_train, y_train)).shuffle(10
00)
    ds_train = ds_train.batch(batch_size, drop_remainder=True)

    ds_test = tf.data.Dataset.from_tensor_slices((X_test, y_test))
    ds_test = ds_test.batch(batch_size)

    model = model_builder()
    validation_data = None
    if use_validation_data:
        validation_data = ds_test
    history = model.fit(ds_train, epochs=epochs, validation_data=validation_dat
a)

    if display_history:
        print(history.history.keys())
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.legend()
        plt.grid()
        plt.show()

        plt.figure()
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.legend()
        plt.grid()
        plt.show()

    ds_test = tf.data.Dataset.from_tensor_slices((X_test, y_test))
    ds_test = ds_test.batch(128, drop_remainder=False)

    predict_proba = model.predict(ds_test)
    print(predict_proba.shape)
    y_pred = np.argmax(predict_proba, axis=1)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')
    f1 = f1_score(y_test, y_pred, average='macro')

    print(f'acc={acc}')
    print(f'prec={prec}')
    print(f'recall={recall}')
    print(f'f1={f1}')

    print(psutil.virtual_memory())
    return model
```

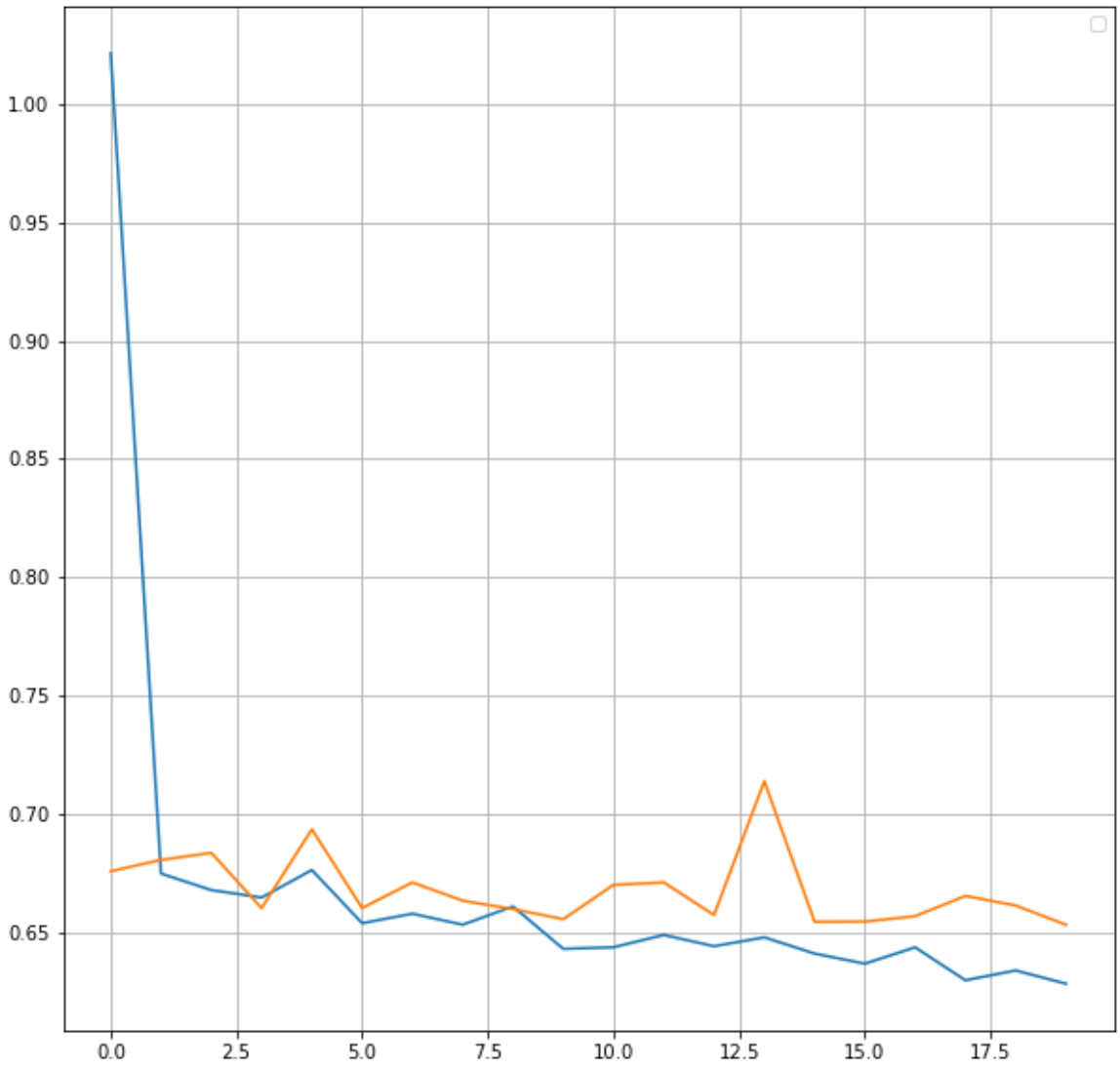
13.5.4 Wywołanie

In [32]:

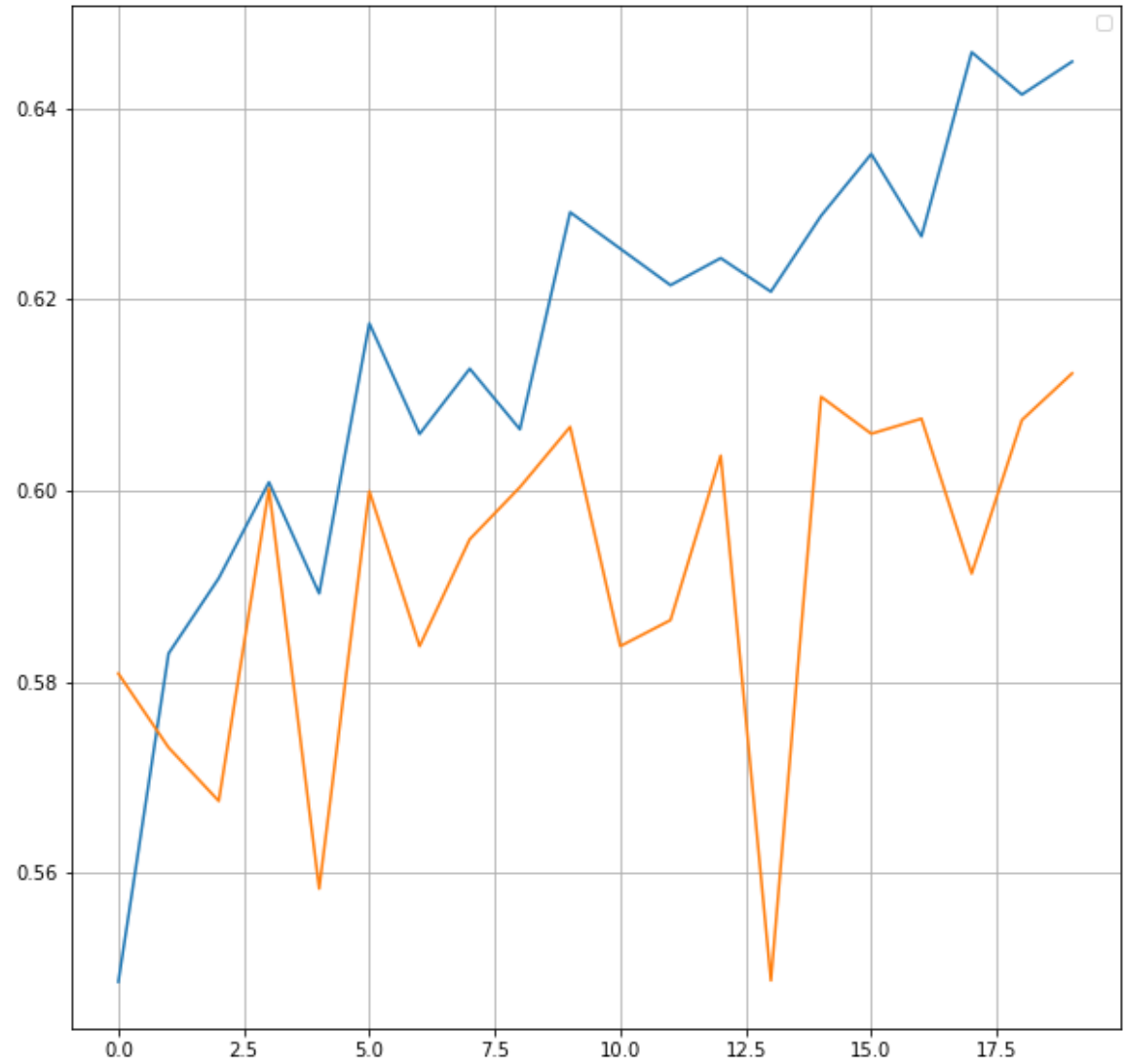
```
m =  split_train_test(X,y,lambda: model_builder((80,80,3),2))
```

```
svmem(total=16977661952, available=4780638208, percent=71.8, used=12
197023744, free=4780638208)
Epoch 1/20
127/127 [==============================] - 5s 37ms/step - loss: 1.02
14 - accuracy: 0.5487 - val_loss: 0.6759 - val_accuracy: 0.5809
Epoch 2/20
127/127 [==============================] - 4s 32ms/step - loss: 0.67
49 - accuracy: 0.5830 - val_loss: 0.6807 - val_accuracy: 0.5731
Epoch 3/20
127/127 [==============================] - 4s 31ms/step - loss: 0.66
79 - accuracy: 0.5909 - val_loss: 0.6837 - val_accuracy: 0.5676
Epoch 4/20
127/127 [==============================] - 4s 32ms/step - loss: 0.66
48 - accuracy: 0.6009 - val_loss: 0.6603 - val_accuracy: 0.6002
Epoch 5/20
127/127 [==============================] - 4s 29ms/step - loss: 0.67
64 - accuracy: 0.5893 - val_loss: 0.6936 - val_accuracy: 0.5584
Epoch 6/20
127/127 [==============================] - 4s 30ms/step - loss: 0.65
40 - accuracy: 0.6175 - val_loss: 0.6603 - val_accuracy: 0.5999
Epoch 7/20
127/127 [==============================] - 4s 34ms/step - loss: 0.65
80 - accuracy: 0.6059 - val_loss: 0.6712 - val_accuracy: 0.5838
Epoch 8/20
127/127 [==============================] - 3s 28ms/step - loss: 0.65
33 - accuracy: 0.6128 - val_loss: 0.6635 - val_accuracy: 0.5949
Epoch 9/20
127/127 [==============================] - 4s 32ms/step - loss: 0.66
10 - accuracy: 0.6064 - val_loss: 0.6599 - val_accuracy: 0.6004
Epoch 10/20
127/127 [==============================] - 4s 32ms/step - loss: 0.64
31 - accuracy: 0.6291 - val_loss: 0.6556 - val_accuracy: 0.6067
Epoch 11/20
127/127 [==============================] - 4s 35ms/step - loss: 0.64
38 - accuracy: 0.6253 - val_loss: 0.6701 - val_accuracy: 0.5838
Epoch 12/20
127/127 [==============================] - 4s 34ms/step - loss: 0.64
90 - accuracy: 0.6215 - val_loss: 0.6712 - val_accuracy: 0.5865
Epoch 13/20
127/127 [==============================] - 4s 34ms/step - loss: 0.64
42 - accuracy: 0.6243 - val_loss: 0.6574 - val_accuracy: 0.6037
Epoch 14/20
127/127 [==============================] - 4s 34ms/step - loss: 0.64
80 - accuracy: 0.6208 - val_loss: 0.7139 - val_accuracy: 0.5488
Epoch 15/20
127/127 [==============================] - 4s 32ms/step - loss: 0.64
11 - accuracy: 0.6288 - val_loss: 0.6545 - val_accuracy: 0.6098
Epoch 16/20
127/127 [==============================] - 4s 32ms/step - loss: 0.63
69 - accuracy: 0.6352 - val_loss: 0.6546 - val_accuracy: 0.6060
Epoch 17/20
127/127 [==============================] - 4s 29ms/step - loss: 0.64
38 - accuracy: 0.6266 - val_loss: 0.6569 - val_accuracy: 0.6075
Epoch 18/20
127/127 [==============================] - 3s 27ms/step - loss: 0.62
98 - accuracy: 0.6459 - val_loss: 0.6655 - val_accuracy: 0.5913
Epoch 19/20
127/127 [==============================] - 4s 29ms/step - loss: 0.63
40 - accuracy: 0.6414 - val_loss: 0.6614 - val_accuracy: 0.6074
Epoch 20/20
```

```
127/127 [==============================] - 4s 33ms/step - loss: 0.62
84 - accuracy: 0.6449 - val_loss: 0.6533 - val_accuracy: 0.6123

WARNING:matplotlib.legend:No artists with labels found to put in leg
end.  Note that artists whose label start with an underscore are ign
ored when legend() is called with no argument.

dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```




```
WARNING:matplotlib.legend:No artists with labels found to put in leg
end.  Note that artists whose label start with an underscore are ign
ored when legend() is called with no argument.
```




```
55/55 [==============================] - 1s 15ms/step
(6979, 2)
acc=0.6122653675311649
prec=0.6125414634516392
recall=0.6120915094409354
f1=0.6118024533612233
svmem(total=16977661952, available=3190104064, percent=81.2, used=13
787557888, free=3190104064)
```

TODO 13.5.2

- Oceń zużycie pamięci
- Czy pamięć została zwolniona po wyjściu z funkcji?

Zużycie pamięci przed: svmem(total=16977661952, available=4780638208, percent=71.8, used=12197023744, free=4780638208)

Zużycie pamięci po:

svmem(total=16977661952, available=3190104064, percent=81.2, used=13787557888, free=3190104064)

Wyraźnie widać że procentowe zużycie pamięci znacznie wzrosło - o około 10 punktów procentowe.

Nie, pamięć nie została zwolniona po wyjściu z funkcji

13.6. Konwolucyjna sieć neuronowa CNN

- Sieci konwolucyjne automatycznie ekstrahują cechy z obrazu stosując operacje konwolucji.
- Nie posługują się ustalonymi manualnie wagami filtrów konwolucji, ale wyznaczają je w trakcie uczenia

Zaprezentowane tu modele pochodzą ze strony [How to Classify Photos of Dogs and Cats (with 97% accuracy) (https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/)](https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/) ale zostały nieco zmodyfikowane

- Rozmiary obrazów sa mniejsze (80 x 80) vs (224 x 224). Zapewne wyniki będa gorsze
- Zastosowane sa modele z dwoma neuronami wyjściowym

13.6.1 Prosty model

In [33]:

```
from keras.layers import Conv2D, Dense, Flatten, MaxPooling2D
from keras import Sequential
from keras.optimizers import SGD
from keras.applications.vgg16 import VGG16
from keras.models import Model


def define_simple_model():
        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_u
niform', padding='same', input_shape=(80, 80, 3)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Flatten())
        model.add(Dense(128, activation='relu', kernel_initializer='he_unifor
m'))
        model.add(Dense(2, activation='softmax'))
        # compile model
        model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
        return model

model = define_simple_model()
```

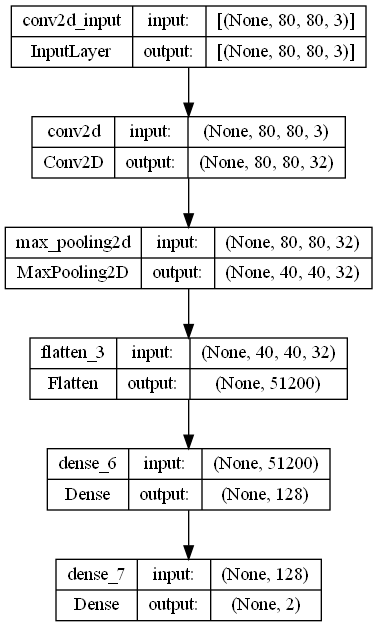
Inspekcja modelu

In [34]:

```
from keras.utils import plot_model

model.summary()
print()
plot_model(model, show_shapes=True, show_layer_names=True, to_file='model.png')
from IPython.display import Image
Image(retina=True, filename='model.png',width = 1000, height = 1000)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 80, 80, 32)        896

 max_pooling2d (MaxPooling2D  (None, 40, 40, 32)       0
 )

 flatten_3 (Flatten)         (None, 51200)             0

 dense_6 (Dense)             (None, 128)               6553728

 dense_7 (Dense)             (None, 2)                 258

=================================================================
Total params: 6,554,882
Trainable params: 6,554,882
Non-trainable params: 0
_________________________________________________________________
```

Out[34]:



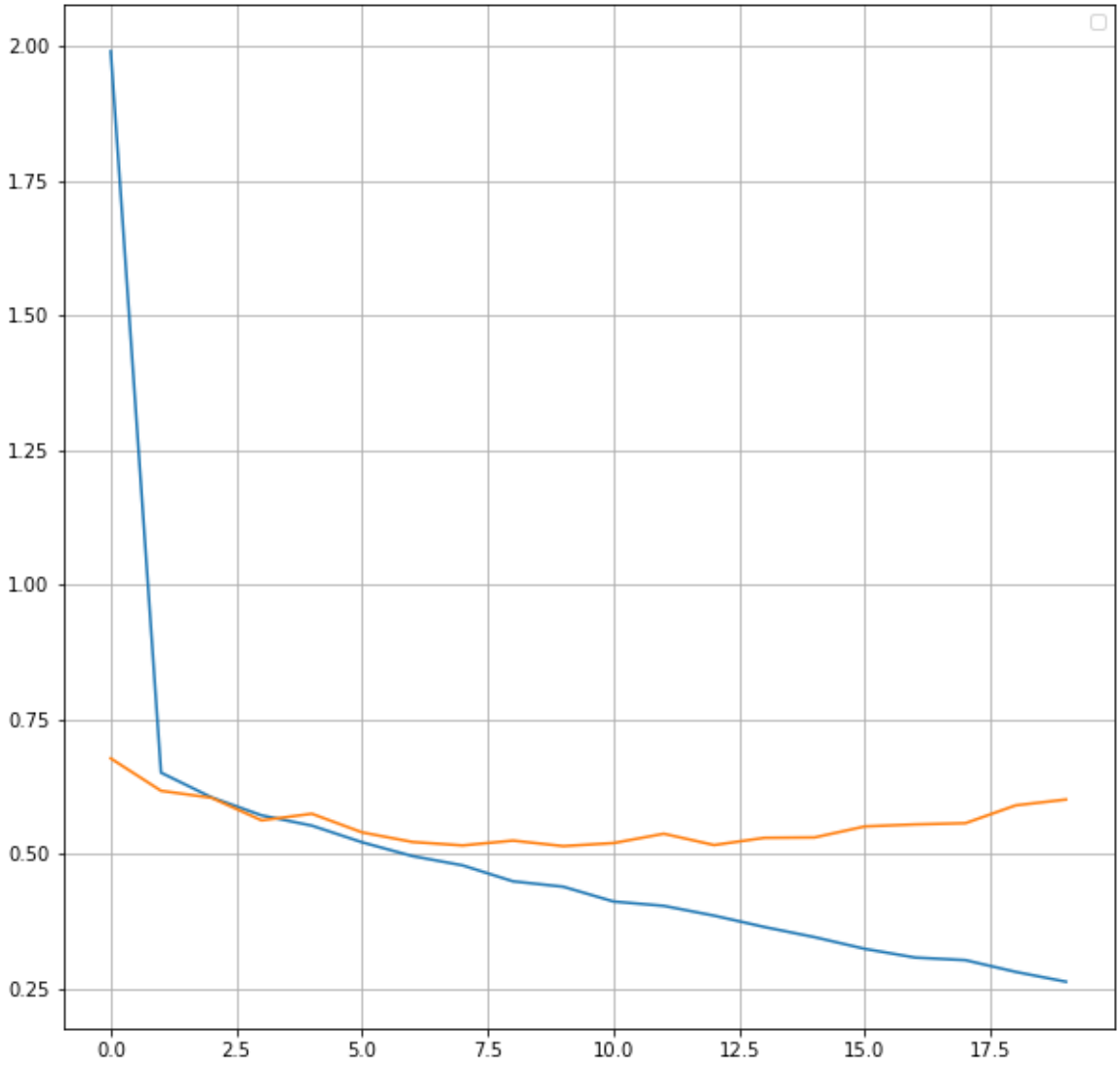
Uczenie i ocena

In [35]:

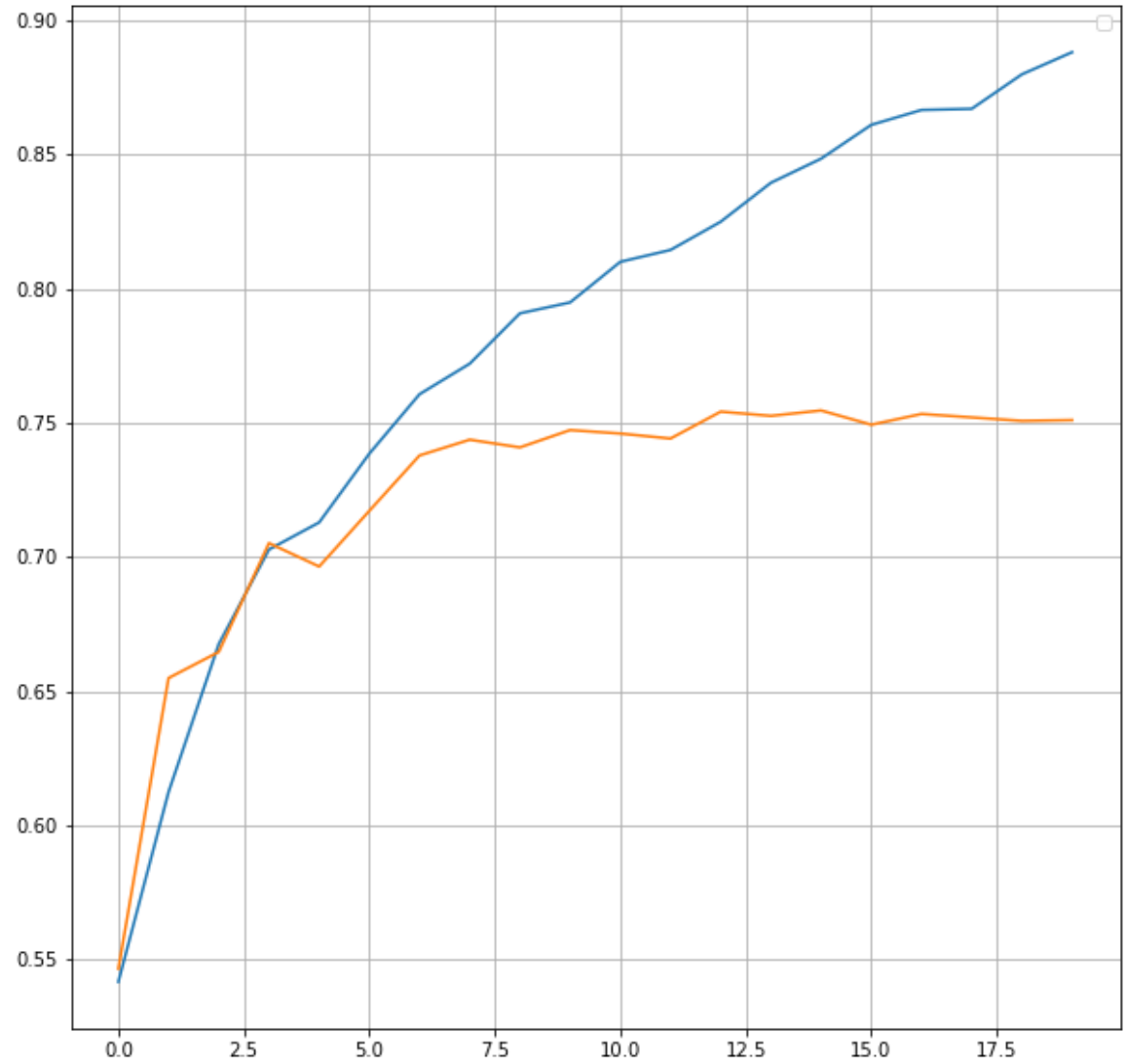
```
m =  split_train_test(X,y,define_simple_model,epochs=20)
```

```
svmem(total=16977661952, available=4925558784, percent=71.0, used=12
052103168, free=4925558784)
Epoch 1/20
127/127 [==============================] - 56s 433ms/step - loss: 1.
9912 - accuracy: 0.5418 - val_loss: 0.6775 - val_accuracy: 0.5466
Epoch 2/20
127/127 [==============================] - 53s 417ms/step - loss: 0.
6510 - accuracy: 0.6125 - val_loss: 0.6173 - val_accuracy: 0.6550
Epoch 3/20
127/127 [==============================] - 52s 411ms/step - loss: 0.
6054 - accuracy: 0.6676 - val_loss: 0.6045 - val_accuracy: 0.6647
Epoch 4/20
127/127 [==============================] - 51s 403ms/step - loss: 0.
5717 - accuracy: 0.7029 - val_loss: 0.5625 - val_accuracy: 0.7053
Epoch 5/20
127/127 [==============================] - 52s 410ms/step - loss: 0.
5527 - accuracy: 0.7130 - val_loss: 0.5747 - val_accuracy: 0.6965
Epoch 6/20
127/127 [==============================] - 51s 402ms/step - loss: 0.
5217 - accuracy: 0.7386 - val_loss: 0.5401 - val_accuracy: 0.7173
Epoch 7/20
127/127 [==============================] - 53s 414ms/step - loss: 0.
4961 - accuracy: 0.7608 - val_loss: 0.5223 - val_accuracy: 0.7379
Epoch 8/20
127/127 [==============================] - 51s 405ms/step - loss: 0.
4788 - accuracy: 0.7721 - val_loss: 0.5159 - val_accuracy: 0.7438
Epoch 9/20
127/127 [==============================] - 53s 416ms/step - loss: 0.
4492 - accuracy: 0.7909 - val_loss: 0.5248 - val_accuracy: 0.7409
Epoch 10/20
127/127 [==============================] - 53s 421ms/step - loss: 0.
4391 - accuracy: 0.7950 - val_loss: 0.5146 - val_accuracy: 0.7474
Epoch 11/20
127/127 [==============================] - 51s 405ms/step - loss: 0.
4117 - accuracy: 0.8101 - val_loss: 0.5202 - val_accuracy: 0.7461
Epoch 12/20
127/127 [==============================] - 51s 404ms/step - loss: 0.
4036 - accuracy: 0.8145 - val_loss: 0.5376 - val_accuracy: 0.7442
Epoch 13/20
127/127 [==============================] - 52s 407ms/step - loss: 0.
3855 - accuracy: 0.8250 - val_loss: 0.5166 - val_accuracy: 0.7543
Epoch 14/20
127/127 [==============================] - 51s 404ms/step - loss: 0.
3644 - accuracy: 0.8396 - val_loss: 0.5299 - val_accuracy: 0.7527
Epoch 15/20
127/127 [==============================] - 51s 406ms/step - loss: 0.
3454 - accuracy: 0.8485 - val_loss: 0.5307 - val_accuracy: 0.7547
Epoch 16/20
127/127 [==============================] - 52s 407ms/step - loss: 0.
3237 - accuracy: 0.8612 - val_loss: 0.5512 - val_accuracy: 0.7494
Epoch 17/20
127/127 [==============================] - 51s 403ms/step - loss: 0.
3076 - accuracy: 0.8667 - val_loss: 0.5549 - val_accuracy: 0.7534
Epoch 18/20
127/127 [==============================] - 51s 402ms/step - loss: 0.
3028 - accuracy: 0.8671 - val_loss: 0.5572 - val_accuracy: 0.7521
Epoch 19/20
127/127 [==============================] - 51s 403ms/step - loss: 0.
2809 - accuracy: 0.8800 - val_loss: 0.5903 - val_accuracy: 0.7508
Epoch 20/20
```

```
127/127 [==============================] - 51s 404ms/step - loss: 0.
2627 - accuracy: 0.8882 - val_loss: 0.6012 - val_accuracy: 0.7511

WARNING:matplotlib.legend:No artists with labels found to put in leg
end.  Note that artists whose label start with an underscore are ign
ored when legend() is called with no argument.

dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```




```
WARNING:matplotlib.legend:No artists with labels found to put in leg
end.  Note that artists whose label start with an underscore are ign
ored when legend() is called with no argument.
```




```
55/55 [==============================] - 6s 98ms/step
(6979, 2)
acc=0.7511104742799828
prec=0.7538574243172546
recall=0.7518064901397714
f1=0.7507429121935926
svmem(total=16977661952, available=4906016768, percent=71.1, used=12
071645184, free=4906016768)
```

13.6.2 Zaawansowany model korzystający z tarnsfer learning

In [7]:

```
from keras.layers import Conv2D, Dense, Flatten, MaxPooling2D
from keras import Sequential
from keras.optimizers import SGD
from keras.applications.vgg16 import VGG16
from keras.models import Model

# Na podstawie
# https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-netwo
rk-to-classify-photos-of-dogs-and-cats/

def define_vgg16_model():
        # load model
        model = VGG16(include_top=False, input_shape=(80, 80, 3))
        # mark loaded layers as not trainable
        for layer in model.layers:
                layer.trainable = False
        # add new classifier layers
        flat1 = Flatten()(model.layers[-1].output)
        class1 = Dense(128, activation='relu', kernel_initializer='he_uniform')
(flat1)
        output = Dense(2, activation='softmax')(class1)
        # define new model
        model = Model(inputs=model.inputs, outputs=output)
        # compile model
        model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
        return model

model = define_vgg16_model()
```

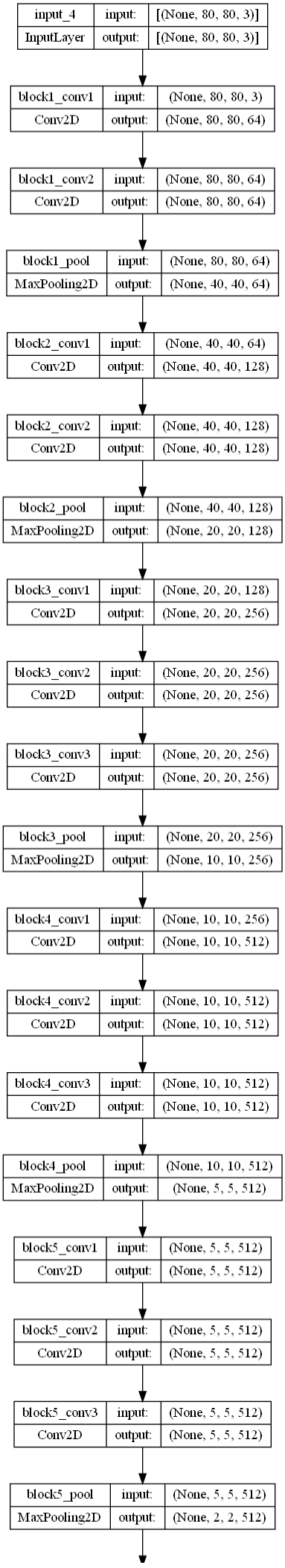
Inspekcja modelu

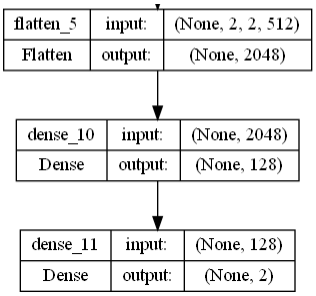
In [37]:

```
from keras.utils import plot_model

model.summary()
print()
plot_model(model, show_shapes=True, show_layer_names=True, to_file='model.png')
from IPython.display import Image
Image(retina=True, filename='model.png',width = 1000, height = 1000)
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 80, 80, 3)]       0

 block1_conv1 (Conv2D)       (None, 80, 80, 64)        1792

 block1_conv2 (Conv2D)       (None, 80, 80, 64)        36928

 block1_pool (MaxPooling2D)  (None, 40, 40, 64)        0

 block2_conv1 (Conv2D)       (None, 40, 40, 128)       73856

 block2_conv2 (Conv2D)       (None, 40, 40, 128)       147584

 block2_pool (MaxPooling2D)  (None, 20, 20, 128)       0

 block3_conv1 (Conv2D)       (None, 20, 20, 256)       295168

 block3_conv2 (Conv2D)       (None, 20, 20, 256)       590080

 block3_conv3 (Conv2D)       (None, 20, 20, 256)       590080

 block3_pool (MaxPooling2D)  (None, 10, 10, 256)       0

 block4_conv1 (Conv2D)       (None, 10, 10, 512)       1180160

 block4_conv2 (Conv2D)       (None, 10, 10, 512)       2359808

 block4_conv3 (Conv2D)       (None, 10, 10, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 5, 5, 512)         0

 block5_conv1 (Conv2D)       (None, 5, 5, 512)         2359808

 block5_conv2 (Conv2D)       (None, 5, 5, 512)         2359808

 block5_conv3 (Conv2D)       (None, 5, 5, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 2, 2, 512)         0

 flatten_5 (Flatten)         (None, 2048)              0

 dense_10 (Dense)            (None, 128)               262272

 dense_11 (Dense)            (None, 2)                 258

=================================================================
Total params: 14,977,218
Trainable params: 262,530
Non-trainable params: 14,714,688
_________________________________________________________________
```

Out[37]:


input_4 | input: | [(None, 80, 80, 3)]
InputLayer | output: | [(None, 80, 80, 3)]
block1_conv1 | input: | (None, 80, 80, 3)
Conv2D | output: | (None, 80, 80, 64)
block1_conv2 | input: | (None, 80, 80, 64)
Conv2D | output: | (None, 80, 80, 64)
block1_pool | input: | (None, 80, 80, 64)
MaxPooling2D | output: | (None, 40, 40, 64)
block2_conv1 | input: | (None, 40, 40, 64)
Conv2D | output: | (None, 40, 40, 128)
block2_conv2 | input: | (None, 40, 40, 128)
Conv2D | output: | (None, 40, 40, 128)
block2_pool | input: | (None, 40, 40, 128)
MaxPooling2D | output: | (None, 20, 20, 128)
block3_conv1 | input: | (None, 20, 20, 128)
Conv2D | output: | (None, 20, 20, 256)
block3_conv2 | input: | (None, 20, 20, 256)
Conv2D | output: | (None, 20, 20, 256)
block3_conv3 | input: | (None, 20, 20, 256)
Conv2D | output: | (None, 20, 20, 256)
block3_pool | input: | (None, 20, 20, 256)
MaxPooling2D | output: | (None, 10, 10, 256)
block4_conv1 | input: | (None, 10, 10, 256)
Conv2D | output: | (None, 10, 10, 512)
block4_conv2 | input: | (None, 10, 10, 512)
Conv2D | output: | (None, 10, 10, 512)
block4_conv3 | input: | (None, 10, 10, 512)
Conv2D | output: | (None, 10, 10, 512)
block4_pool | input: | (None, 10, 10, 512)
MaxPooling2D | output: | (None, 5, 5, 512)
block5_conv1 | input: | (None, 5, 5, 512)
Conv2D | output: | (None, 5, 5, 512)
block5_conv2 | input: | (None, 5, 5, 512)
Conv2D | output: | (None, 5, 5, 512)
block5_conv3 | input: | (None, 5, 5, 512)
Conv2D | output: | (None, 5, 5, 512)
block5_pool | input: | (None, 5, 5, 512)
MaxPooling2D | output: | (None, 2, 2, 512)

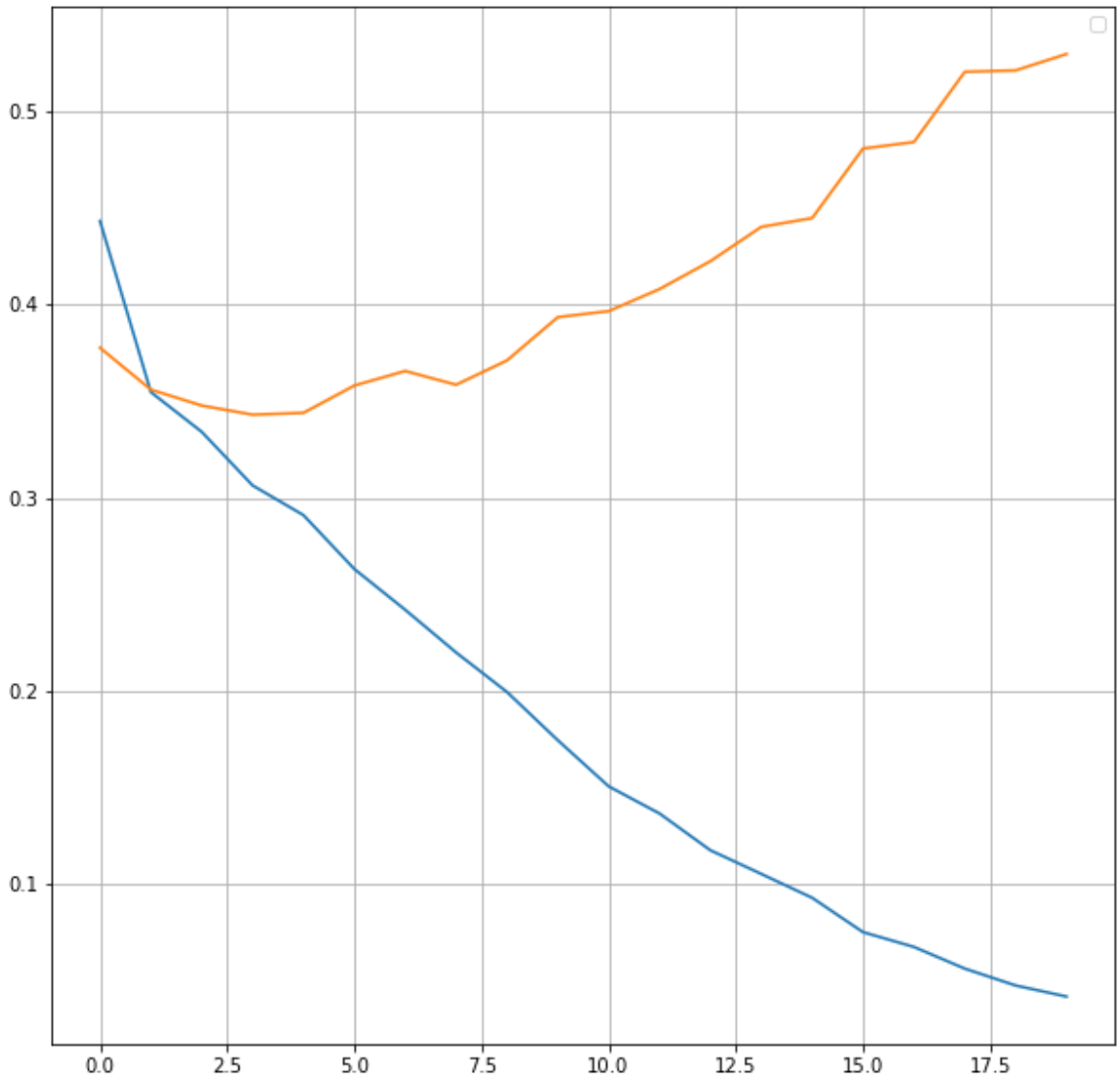

flatten_5 input: (None, 2, 2, 512)
Flatten output: (None, 2048)
dense_10 input: (None, 2048)
Dense output: (None, 128)
dense_11 input: (None, 128)
Dense output: (None, 2)

In [38]:

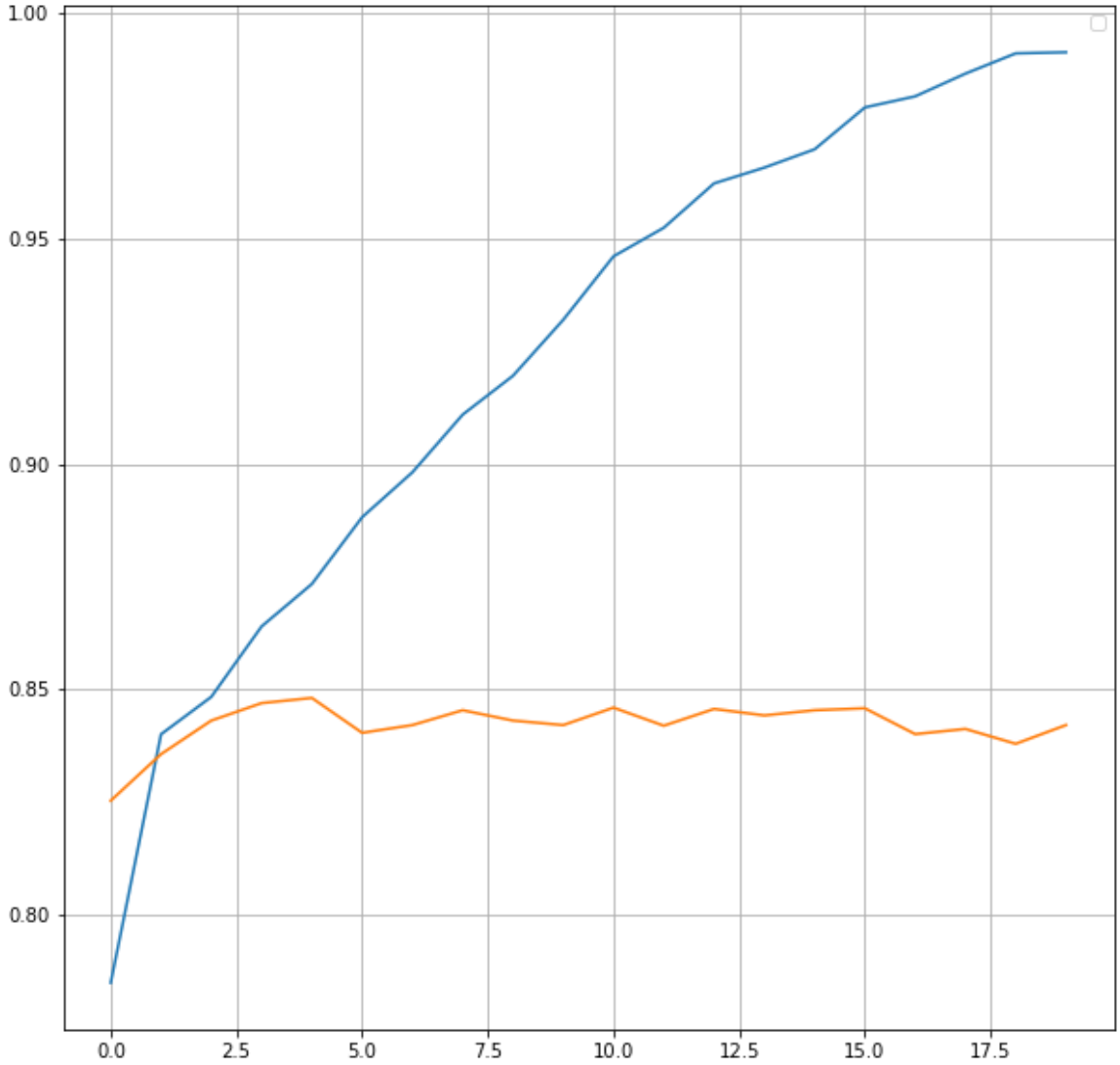
```
m =  split_train_test(X,y,define_vgg16_model,epochs=20)
```

```
svmem(total=16977661952, available=6163402752, percent=63.7, used=10
814259200, free=6163402752)
Epoch 1/20
127/127 [==============================] - 348s 3s/step - loss: 0.44
32 - accuracy: 0.7849 - val_loss: 0.3776 - val_accuracy: 0.8253
Epoch 2/20
127/127 [==============================] - 341s 3s/step - loss: 0.35
46 - accuracy: 0.8401 - val_loss: 0.3559 - val_accuracy: 0.8356
Epoch 3/20
127/127 [==============================] - 344s 3s/step - loss: 0.33
41 - accuracy: 0.8484 - val_loss: 0.3477 - val_accuracy: 0.8431
Epoch 4/20
127/127 [==============================] - 345s 3s/step - loss: 0.30
63 - accuracy: 0.8640 - val_loss: 0.3430 - val_accuracy: 0.8470
Epoch 5/20
127/127 [==============================] - 358s 3s/step - loss: 0.29
09 - accuracy: 0.8734 - val_loss: 0.3439 - val_accuracy: 0.8481
Epoch 6/20
127/127 [==============================] - 347s 3s/step - loss: 0.26
31 - accuracy: 0.8882 - val_loss: 0.3580 - val_accuracy: 0.8404
Epoch 7/20
127/127 [==============================] - 366s 3s/step - loss: 0.24
20 - accuracy: 0.8982 - val_loss: 0.3656 - val_accuracy: 0.8421
Epoch 8/20
127/127 [==============================] - 378s 3s/step - loss: 0.21
99 - accuracy: 0.9110 - val_loss: 0.3584 - val_accuracy: 0.8454
Epoch 9/20
127/127 [==============================] - 338s 3s/step - loss: 0.19
94 - accuracy: 0.9196 - val_loss: 0.3710 - val_accuracy: 0.8431
Epoch 10/20
127/127 [==============================] - 331s 3s/step - loss: 0.17
44 - accuracy: 0.9320 - val_loss: 0.3934 - val_accuracy: 0.8421
Epoch 11/20
127/127 [==============================] - 338s 3s/step - loss: 0.15
06 - accuracy: 0.9461 - val_loss: 0.3965 - val_accuracy: 0.8460
Epoch 12/20
127/127 [==============================] - 339s 3s/step - loss: 0.13
65 - accuracy: 0.9524 - val_loss: 0.4080 - val_accuracy: 0.8420
Epoch 13/20
127/127 [==============================] - 343s 3s/step - loss: 0.11
75 - accuracy: 0.9623 - val_loss: 0.4224 - val_accuracy: 0.8457
Epoch 14/20
127/127 [==============================] - 339s 3s/step - loss: 0.10
52 - accuracy: 0.9658 - val_loss: 0.4401 - val_accuracy: 0.8442
Epoch 15/20
127/127 [==============================] - 340s 3s/step - loss: 0.09
29 - accuracy: 0.9699 - val_loss: 0.4447 - val_accuracy: 0.8454
Epoch 16/20
127/127 [==============================] - 340s 3s/step - loss: 0.07
51 - accuracy: 0.9791 - val_loss: 0.4807 - val_accuracy: 0.8458
Epoch 17/20
127/127 [==============================] - 339s 3s/step - loss: 0.06
75 - accuracy: 0.9816 - val_loss: 0.4840 - val_accuracy: 0.8401
Epoch 18/20
127/127 [==============================] - 341s 3s/step - loss: 0.05
63 - accuracy: 0.9867 - val_loss: 0.5204 - val_accuracy: 0.8412
Epoch 19/20
127/127 [==============================] - 338s 3s/step - loss: 0.04
75 - accuracy: 0.9911 - val_loss: 0.5211 - val_accuracy: 0.8379
Epoch 20/20
```

```
127/127 [==============================] - 329s 3s/step - loss: 0.04
18 - accuracy: 0.9914 - val_loss: 0.5296 - val_accuracy: 0.8421

WARNING:matplotlib.legend:No artists with labels found to put in leg
end.  Note that artists whose label start with an underscore are ign
ored when legend() is called with no argument.

dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```




```
WARNING:matplotlib.legend:No artists with labels found to put in leg
end.  Note that artists whose label start with an underscore are ign
ored when legend() is called with no argument.
```




```
55/55 [==============================] - 118s 2s/step
(6979, 2)
acc=0.8420977217366384
prec=0.8428989329964571
recall=0.8421772289162761
f1=0.8420257150979455
svmem(total=16977661952, available=2992447488, percent=82.4, used=13
985214464, free=2992447488)
```

13.6.3 Zobaczmy koty i psy...

In [39]:

```
ds = tfds.load('cats_vs_dogs', split='train', batch_size=None,as_supervised=Tru
e).shuffle(1000)
it = ds.as_numpy_iterator()
plt.rcParams["figure.figsize"] = (10,10)
labels=['Cat','Dog']
input_size=80
for i in range(9):
  ax = plt.subplot(330 + 1 + i)
  image, label = next(it)
  image = tf.image.resize(image, size=[80,80], method=tf.image.ResizeMethod.NEAR
EST_NEIGHBOR)
  batched_image = tf.expand_dims(image, axis=0)
  pp = m.predict(batched_image)
  c = np.argmax(pp)
  ax.set_title(labels[label]+" - predicted:"+labels[c])
  ax.imshow(image)
plt.show()
```

```
1/1 [==============================] - 1s 651ms/step
1/1 [==============================] - 0s 112ms/step
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 65ms/step
```

TODO 13.6.1

- Przeanalizuj wykresy. Ile epok wystarczyło wybrać?
- Jak zmienia się wartość funkcji kosztu i trafność dla zbioru uczącego?
- Jak scharakteryzujesz sytuację począwszy od 5-6 epoki?

Wystarczyło wybrać 5 epok.

Funkcja kosztu maleje przez cały okres trenowania sieci. Trafność dla zbioru uczącego rośnie logarytmicznie.

Od 5-6 epoki val_loss dla zbioru testowego zaczyna rosnąć. Parametr val_loss odnosi się do wartości funkcji straty obliczanej na zestawie walidacyjnym podczas procesu uczenia modelu maszynowego. Celem uczenia maszynowego jest minimalizacja funkcji straty, a więc uzyskanie malejącego parametru val_loss.

13.6.4 Twój model

TODO 13.6.2

- Wybierz jeden z modeli ze strony How to Classify Photos of Dogs and Cats (with 97% accuracy) (https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/)
- Dostosuj analogicznie do innej reprezentacji wyjścia (2 neurony)
- Wyświetl strukture
- Przeprowadź uczenie i ocenę

In [8]:

```
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_unifo
rm', padding='same', input_shape=(80, 80, 3)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(1, activation='sigmoid'))
    opt = SGD(lr=0.001, momentum=0.9)
    model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accurac
y'])
    return model

model = define_model()
```

```
c:\Users\krzyc\anaconda3-1\lib\site-packages\keras\optimizers\optimi
zer_v2\gradient_descent.py:111: UserWarning: The `lr` argument is de
precated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

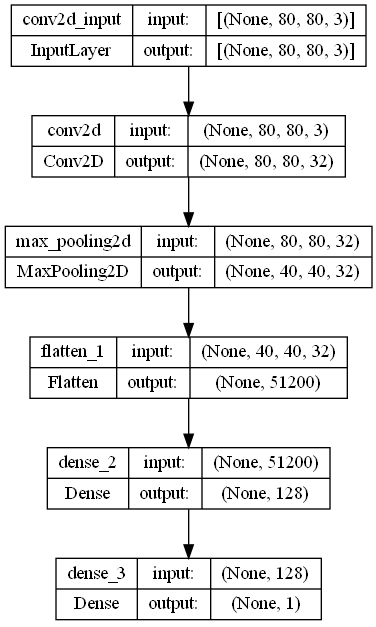
Inspekcja modelu

In [10]:

```
from keras.utils import plot_model

model.summary()
print()
plot_model(model, show_shapes=True, show_layer_names=True, to_file='model.png')
from IPython.display import Image
Image(retina=True, filename='model.png',width = 1000, height = 1000)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 80, 80, 32)        896

 max_pooling2d (MaxPooling2D  (None, 40, 40, 32)       0
 )

 flatten_1 (Flatten)         (None, 51200)             0

 dense_2 (Dense)             (None, 128)               6553728

 dense_3 (Dense)             (None, 1)                 129

=================================================================
Total params: 6,554,753
Trainable params: 6,554,753
Non-trainable params: 0
_________________________________________________________________
```

Out[10]:



Uczenie i ocena

TODO 13.6.3

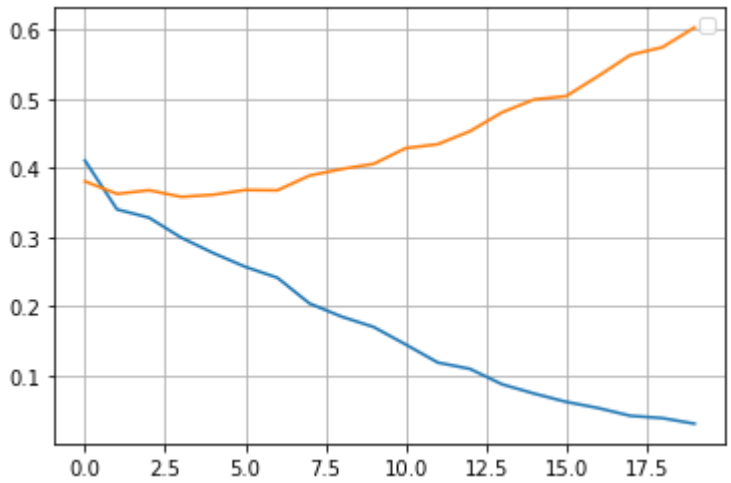
- Porównaj złożoność wybranego modelu z innymi i skomentuj wyniki

In [12]:

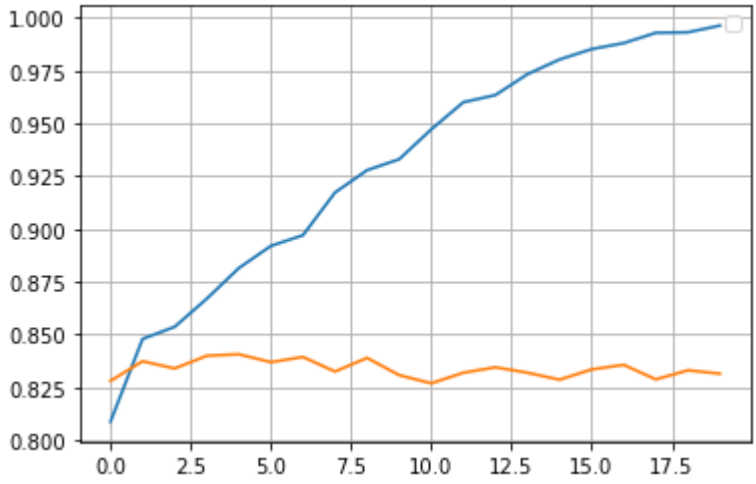
```
m =  split_train_test(X,y,define_vgg16_model,epochs=20)
```

```
svmem(total=16977661952, available=874242048, percent=94.9, used=161
03419904, free=874242048)
Epoch 1/20
127/127 [==============================] - 343s 3s/step - loss: 0.41
10 - accuracy: 0.8087 - val_loss: 0.3809 - val_accuracy: 0.8281
Epoch 2/20
127/127 [==============================] - 347s 3s/step - loss: 0.34
03 - accuracy: 0.8479 - val_loss: 0.3628 - val_accuracy: 0.8374
Epoch 3/20
127/127 [==============================] - 340s 3s/step - loss: 0.32
83 - accuracy: 0.8537 - val_loss: 0.3680 - val_accuracy: 0.8339
Epoch 4/20
127/127 [==============================] - 333s 3s/step - loss: 0.29
94 - accuracy: 0.8669 - val_loss: 0.3584 - val_accuracy: 0.8399
Epoch 5/20
127/127 [==============================] - 334s 3s/step - loss: 0.27
71 - accuracy: 0.8815 - val_loss: 0.3615 - val_accuracy: 0.8407
Epoch 6/20
127/127 [==============================] - 333s 3s/step - loss: 0.25
71 - accuracy: 0.8920 - val_loss: 0.3684 - val_accuracy: 0.8369
Epoch 7/20
127/127 [==============================] - 333s 3s/step - loss: 0.24
13 - accuracy: 0.8970 - val_loss: 0.3679 - val_accuracy: 0.8394
Epoch 8/20
127/127 [==============================] - 334s 3s/step - loss: 0.20
41 - accuracy: 0.9173 - val_loss: 0.3893 - val_accuracy: 0.8325
Epoch 9/20
127/127 [==============================] - 334s 3s/step - loss: 0.18
50 - accuracy: 0.9278 - val_loss: 0.3987 - val_accuracy: 0.8389
Epoch 10/20
127/127 [==============================] - 337s 3s/step - loss: 0.17
00 - accuracy: 0.9330 - val_loss: 0.4062 - val_accuracy: 0.8308
Epoch 11/20
127/127 [==============================] - 338s 3s/step - loss: 0.14
46 - accuracy: 0.9472 - val_loss: 0.4289 - val_accuracy: 0.8269
Epoch 12/20
127/127 [==============================] - 335s 3s/step - loss: 0.11
85 - accuracy: 0.9600 - val_loss: 0.4347 - val_accuracy: 0.8319
Epoch 13/20
127/127 [==============================] - 331s 3s/step - loss: 0.10
94 - accuracy: 0.9634 - val_loss: 0.4534 - val_accuracy: 0.8345
Epoch 14/20
127/127 [==============================] - 333s 3s/step - loss: 0.08
71 - accuracy: 0.9733 - val_loss: 0.4807 - val_accuracy: 0.8319
Epoch 15/20
127/127 [==============================] - 333s 3s/step - loss: 0.07
36 - accuracy: 0.9803 - val_loss: 0.4994 - val_accuracy: 0.8286
Epoch 16/20
127/127 [==============================] - 341s 3s/step - loss: 0.06
16 - accuracy: 0.9852 - val_loss: 0.5043 - val_accuracy: 0.8335
Epoch 17/20
127/127 [==============================] - 341s 3s/step - loss: 0.05
27 - accuracy: 0.9880 - val_loss: 0.5334 - val_accuracy: 0.8356
Epoch 18/20
127/127 [==============================] - 339s 3s/step - loss: 0.04
15 - accuracy: 0.9929 - val_loss: 0.5639 - val_accuracy: 0.8288
Epoch 19/20
127/127 [==============================] - 338s 3s/step - loss: 0.03
81 - accuracy: 0.9931 - val_loss: 0.5751 - val_accuracy: 0.8331
Epoch 20/20
```

```
127/127 [==============================] - 331s 3s/step - loss: 0.03
00 - accuracy: 0.9963 - val_loss: 0.6036 - val_accuracy: 0.8315

WARNING:matplotlib.legend:No artists with labels found to put in leg
end.  Note that artists whose label start with an underscore are ign
ored when legend() is called with no argument.

dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```




```
WARNING:matplotlib.legend:No artists with labels found to put in leg
end.  Note that artists whose label start with an underscore are ign
ored when legend() is called with no argument.
```




```
55/55 [==============================] - 101s 2s/step
(6979, 2)
acc=0.831494483450351
prec=0.8318805726689342
recall=0.8316826730692277
f1=0.8314840174660438
svmem(total=16977661952, available=2973536256, percent=82.5, used=14
004125696, free=2973536256)
```

In [ ]:

```
ds = tfds.load('cats_vs_dogs', split='train', batch_size=None,as_supervised=Tru
e).shuffle(1000)
it = ds.as_numpy_iterator()
plt.rcParams["figure.figsize"] = (10,10)
labels=['Cat','Dog']
input_size=100
for i in range(9):
  ax = plt.subplot(330 + 1 + i)
  image, label = next(it)
  image = tf.image.resize(image, size=[100,100], method=tf.image.ResizeMethod.NE
AREST_NEIGHBOR)
  batched_image = tf.expand_dims(image, axis=0)
  pp = m.predict(batched_image)
  c = np.argmax(pp)
  ax.set_title(labels[label]+" - predicted:"+labels[c])
  ax.imshow(image)
plt.show()
```

Porównanie trzech ostatnich modeli w tym notatniku

Model 1:

Liczba warstw konwolucyjnych: 1 Liczba warstw poolingowych: 1 Liczba warstw gęstych: 2 Całkowita liczba warstw: 6

Model 2:

Liczba warstw konwolucyjnych: 1 Liczba warstw poolingowych: 1 Liczba warstw gęstych: 1 Liczba warstw wyjściowych: 1 Całkowita liczba warstw: 5

Model 3:

Liczba warstw konwolucyjnych: 1 Liczba warstw poolingowych: 1 Liczba warstw gęstych: 2 Liczba warstw wyjściowych: 1 Całkowita liczba warstw: 6