Klasyfikacja obrazów przy użyciu sztucznych sieci neuronowych

Autorzy: Kamil Kućma

Wydział Zarządzania
Kierunek studiów: Informatyka i Ekonometria
Rok: II
Tryb: Stacjonarnie
Przedmiot: Elementy sztucznej inteligencji

Opis problemu

Celem projektu jest stworzenie sztucznej sieci neuronowej, która umożliwia klasyfikację zdjęć do danej kategorii. Zbiór zdjęć, pobrany ze strony kaggle.com (Link do danych), przedstawia zdjęcia ułożenia dłoni do popularnej gry w kamień papier nożyce. Cały zestaw, zawierający ponad 2 tysiące zdjęć, podzielony jest właśnie na te trzy kategorie (kamień, papier, nożyce), gdzie w każdej jest ponad 700 zdjęć, każde w rozdzielczości 300x200px w modelu RGB zawierającym 3 kanały. Celem projektu jest dobranie takich współczynników, aby dokładność uczenia zbudowanej sieci była jak największa. Dodatkowo, po wczytaniu dodatkowego zdjęcia do programu, ten zakwalifikuje je do jednej z powyższych kategorii
.

Odwołanie do innych prac:

RPS Classify EfficientNetB7

W pracy autor wykonuje model do klasyfikacji obrazów z tego samego zestawu.
Model ma klasyfikować obrazy do 3 kategorii paper, rock, scissors. Wczytywanie i przetwarzanie zdjęć oraz przygotowanie danych do modelu wygląda podobnie z uwzględnieniem tego, że autor pozostawia zdjęcia w modelu RGB oraz dzieli on zestaw na 3 podzbiory (dodatkowo do walidacji). Autor tworząc model, nie używa funkcji Sequential (każda warstwa ma jedno wejście i jedno wyjście), lecz tworzy nieco inny model z użyciem funkcji EfficientNetB7. Dodaje dwie warstwy,z niewielkimi parametrami units, epoch i bach_size, z inną funkcją aktywacji ostatniej warstwy oraz inną metodą uczenia. Również pokazuje otrzymane wyniki w postaci wykresów. Otrzymana dokładność uczenia modelu oscyluje w okolicach 95% i jest bardzo podobna do otrzymanej przez nas.

Z racji, iż wykorzystywany dataset nie jest wykorzystywany powszechnie jako przykład do deep learningu, to odwołujemy się również do prac, w których wykorzystano inne zbiory obrazów, lecz w których cel jest taki sam.

https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/

W tym przypadku, autor dysponuje datasetem zawierającym dwie kategorie zdjęć - psów i kotów, lecz ich liczba jest znacznie większa - w suma zdjęć wynosi 25 tysięcy. Wczytanie zdjęć i ich przeróbka wygląda bardzo podobnie - również wczytuje zdjęcia z dysku, zmienia ich rozmiar, pozbywa się 3 kanałowego koloru, normalizuje i kategoryzuje odpowiednie zmienne. Różni się to tym, że autor ma już zbiór zdjęć podzielony na dysku dodatkowo na test i trening przez co nie musi go rozdzielać w programie. Następnie tworzy model wyglądający podobnie, z podobnymi parametrami, również ma 5 warstw z tymi samymi funkcjami aktywującymi poza ostatnią. Następnie prezentuje wyniki na wykresach. W porównaniu do naszych wyników, otrzymane wyniki są słabe, dokładność wynosi 60% (a trafienie wyniku to 50/50) a zbiór uczący jest bardzo duży - 20tys. zdjęć. Autor wyciąga wniosek, że tego rodzaju model z takimi rodzajami warstw nie służy do uczenia na dwuwymiarowych danych (np. obrazach) i należy to robić używając konwolucyjnej sieci. Jednak zarówno nasza, jak i poprzednia praca pokazują, że również używając takiej sieci można uzyskać zadowalające wyniki.

ANALIZA CZYNNIKÓW

Units (wielkość wyjścia warstwy):

	liczba unitsów	funkcja aktywacji
Warstwa 1	2400	relu
Warstwa 2	800	relu
Warstwa 3	800	relu
Warstwa 4	800	relu
Warstwa 5	3	sigmoid

batch_size	epochs	optimizer
32	70	adam

Test loss	test accuracy
0.43318501114845276	0.8482632637023926

	liczba unitsów	funkcja aktywacji
Warstwa 1	2400	relu
Warstwa 2	600	relu
Warstwa 3	150	relu
Warstwa 4	30	relu
Warstwa 5	3	sigmoid

batch_size	epochs	optimizer
32	70	adam

Test loss	test accuracy
0.2915635108947754	0.9269406199455261

	liczba unitsów	funkcja aktywacji
Warstwa 1	2400	relu
Warstwa 2	30	relu
Warstwa 3	150	relu
Warstwa 4	600	relu
Warstwa 5	3	sigmoid

batch_size	epochs	optimizer
32	70	adam

Test loss	test accuracy
1.0984296798706055	0.3418647050857544

	liczba unitsów	funkcja aktywacji
Warstwa 1	2400	relu
Warstwa 2	600	relu
Warstwa 3	900	relu
Warstwa 4	600	relu
Warstwa 5	3	sigmoid

batch_size	epochs	optimizer
32	70	adam

Test loss	test accuracy
0.4525897800922394	0.8555758595466614

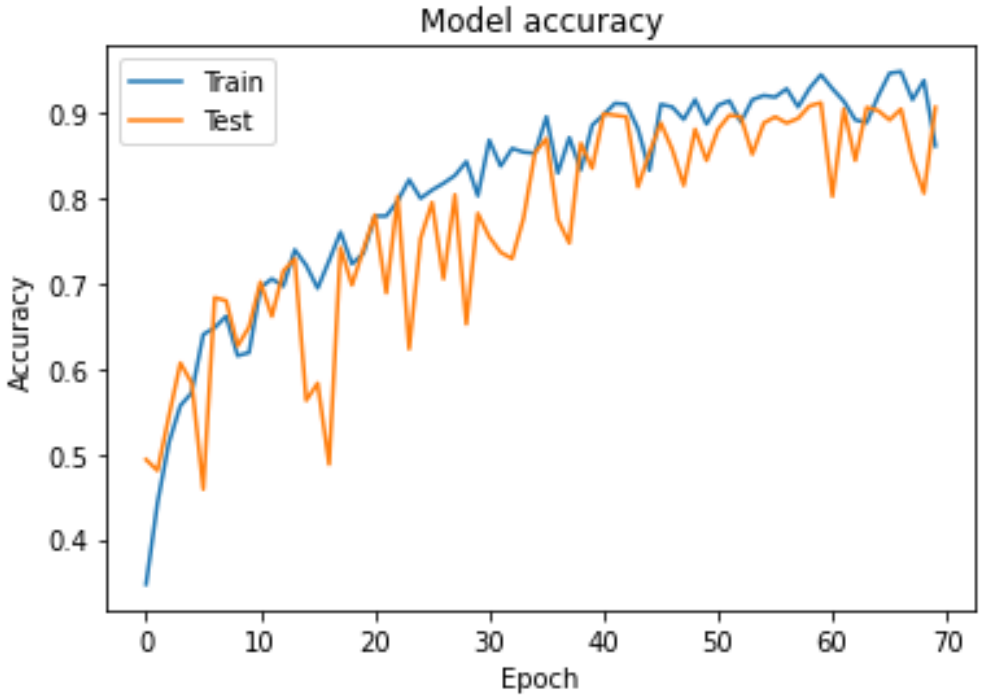
	liczba unitsów	funkcja aktywacji
Warstwa 1	2400	relu
Warstwa 2	500	relu
Warstwa 3	200	relu
Warstwa 4	20	relu
Warstwa 5	3	sigmoid

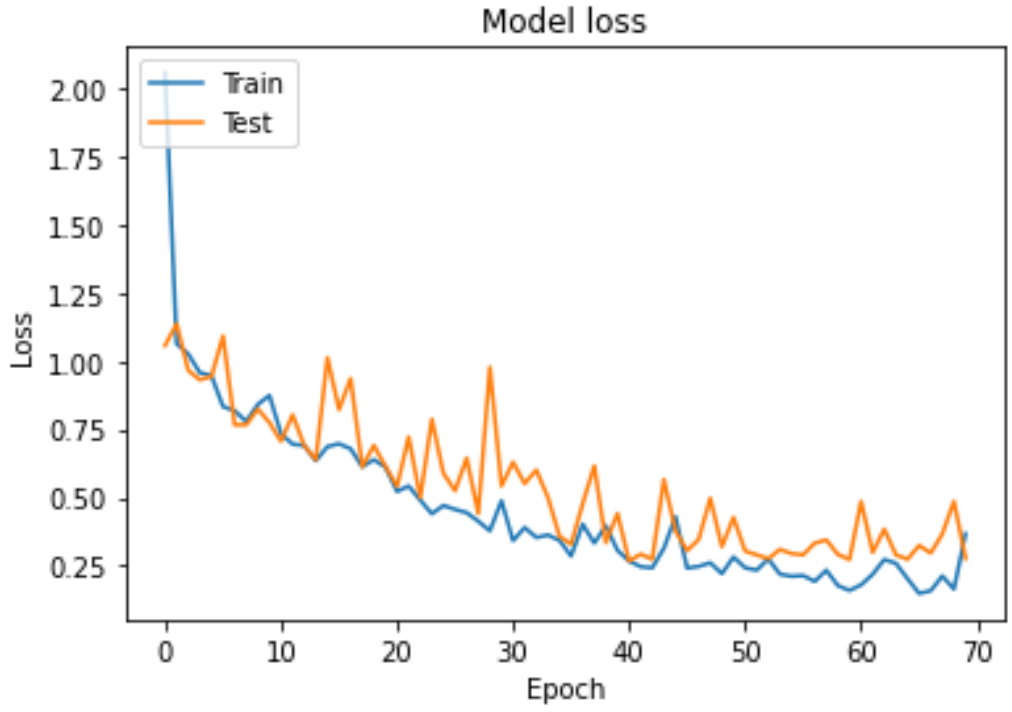
batch_size	epochs	optimizer
32	70	adam

Test loss	test accuracy
0.23567311465740204	0.9021602187444375

Przyjmując, na wejściu 2400 a na wyjściu 3, wyniki są najlepsze, gdy liczba units maleje liniowo/logarytmicznie, gdy jest stała lub zwiększa/zmniejsza się na zmianę, wyniki się pogarszają. Nieznaczne zmiany wielkości, przy zachowaniu odpowiednich tendencji nie mają większego wpływu na wynik. Do kolejnych kroków przyjmuje wartości 2400 600 150 30 3

Wykresy dla najlepszej kombinacji:





WARSTWY:

2 warstwy:

	liczba unitsów	funkcja aktywacji
Warstwa 1	2400	relu
Warstwa 2	3	sigmoid

batch_size	epochs	optimizer
32	70	adam

Test loss	test accuracy
0.785247814655304	0.6733089327812195

3 warstwy:

	liczba unitsów	funkcja aktywacji
Warstwa 1	2400	relu
Warstwa 2	1000	relu
Warstwa 3	3	sigmoid

batch_size	epochs	optimizer
32	70	adam

Test loss	test accuracy
0.3412716472148895	0.8867641496658325

7 warstw:

	liczba unitsów	funkcja aktywacji
Warstwa 1	2400	relu
Warstwa 2	1500	relu
Warstwa 3	800	relu
Warstwa 4	400	relu
Warstwa 5	150	relu
Warstwa 6	30	relu
Warstwa 7	3	sigmoid

batch_size	epochs	optimizer
32	70	adam

Test loss	test accuracy
0.6335867047309875	0.7367458939552307

Dla dwóch warstw oraz dla dużej liczby 7+, wyniki się pogarszają, dla 3 i 5 wyniki wyglądają podobnie, ale dla 5 są nieco lepsze, więc wybieramy model z 5 warstwami Optymalna liczba warstw jest taka sama jest przy analizie liczby unitsów zatem wyniki są takie same więc wykresy przedstawiające ten wynik znajdują się powyżej.

Funkcje aktywacji:

Przy użyciu funkcji relu w każdej warstwie:

Test accuracy : 0.3264840245246887
Test loss : 0.8170813336146876

Przy użyciu funkcji sigmoid w każdej warstwie:

Test accuracy : 0.6141552329063416
Test loss: 0.8400813341140747

Przy użyciu funkcji softmax w każdej warstwie:

Test accuracy : 0.34246575832366943
Test loss: 1.0984135866165161

Przy użyciu funkcji exponential w każdej warstwie:

Test accuracy : 0.3264840245246887
Test loss: 0.8255316841608142

Przy użyciu funkcji softplus w każdej warstwie:

Test accuracy : 0.7397260069847107
Test loss: 0.5505733489990234

Przy użyciu funkcji relu + softmax w ostatniej warstwie

Test accuracy : 0.9049360156059265
Test loss: 0.3323972702026367

Jedynie kombinacja funkcji aktywacji relu + softmax w ostatniej warstwie daje podobne wyniki co relu + sigmoid, ale w odróżnieniu do każdej widzianej przez nas pracy, gdzie używaną funkcją w ostatniej warstwie było softmax, w tym przypadku minimalnie, ale lepsze wyniki daje użycie funkcji sigmoid.
Optymalna kombinacja funkcji aktywacji jest taka zatem wyniki są takie same więc wykresy przedstawiające ten wynik znajdują się powyżej.

Sposób dobierania próby uczącej i testowej:

Dla 85% uczenia i 15% testowania:

Test accuracy : 0.8767641496658325
Test loss: 0.296743262632613

Dla 65% uczenia i 35% testowania:
Test accuracy : 0.8501562642663473
Test loss: 0.376485705232904

Oba wyniki są gorsze od tego uzyskiwanego dotychczas dla 75% uczenia i 25% testowania.
Optymalna kombinacja części uczenia i testowania jest taka sama, zatem wyniki są takie same więc wykresy przedstawiające ten wynik znajdują się powyżej

Batch_size(części zbioru uczącego) + Epchos(cykle trenowania):

Można zauważyć pewną zależność między tymi dwoma czynnikami (wielkość batch_size wpływa na wielkość każdego epoch) więc będą analizowane wspólnie

batch_size: 32 epochs: 70
Test accuracy: 0.8628884553909302
Test loss: 0.38039401173591614

batch_size: 64 epochs: 70
Test accuracy: 0.8894515538215637
Test loss: 0.2767857015132904

batch_size: 8 epochs: 70
Test accuracy: 0.42517938501804538
Test loss: 1.04123940117359161

batch_size: 100 epochs: 70
Test accuracy: 0.8975137066841125
Test loss: 0.35188719630241394

batch_size: 200 epochs: 70
Test accuracy:0.7458866834640503
Test Loss: 0.6402800679206848

batch_size: 100 epochs: 35

Test accuracy :0.8738573789596558
Test loss: 0.3393741846084595

batch_size: 100 epochs: 100

test accuracy :0.9099817442893982
test loss:0.3568240761756897

batch_size: 100 epochs: 180

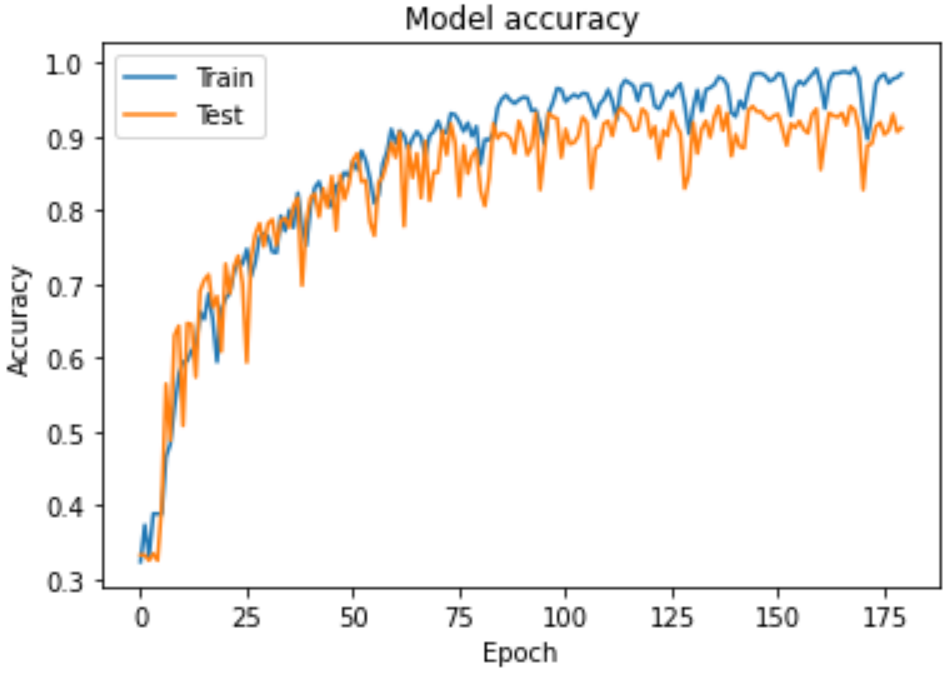
Test loss: 0.3547388029098511
Test accuracy: 0.9239670920372009

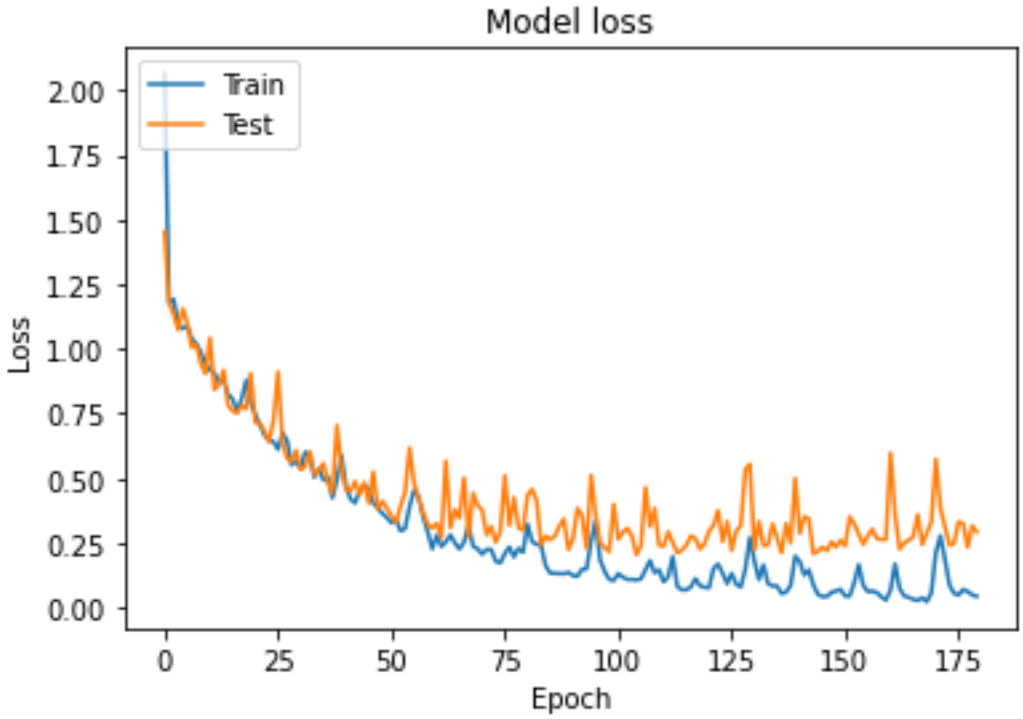
batch_size: 100 epochs: 400

Test loss: 0.4847388029098511
Test accuracy: 0.9265265274047852

Jeśli chodzi o batch_size to wartość w okolicach 100 wydaje się być najlepsza, natomiast jeśli chodzi o epochs, to kolejne uczenia dążą do 1, jednak całkowity wynik nie poprawia się znacząco, a wzrasta wartość test loss co świadczy o przeuczeniu modelu i wartość dla epochs = 180 wydaje się być odpowiednia uwzględniając głównie dokładność modelu, ale również rosnące błędy testowania.

Wykresy dla najlepszej kombinacji epchos i batch_size:





Metody Uczenia:

Adadelta:

Test loss: 0.795846700668335
Test accuracy: 0.6965265274047852

Adagrad:

Test loss: 0.40456414222717285
Test accuracy: 0.8592321872711182

Nadam:

Test loss: 0.740446765268335
Test accuracy: 0.6465166274047852

SGD:

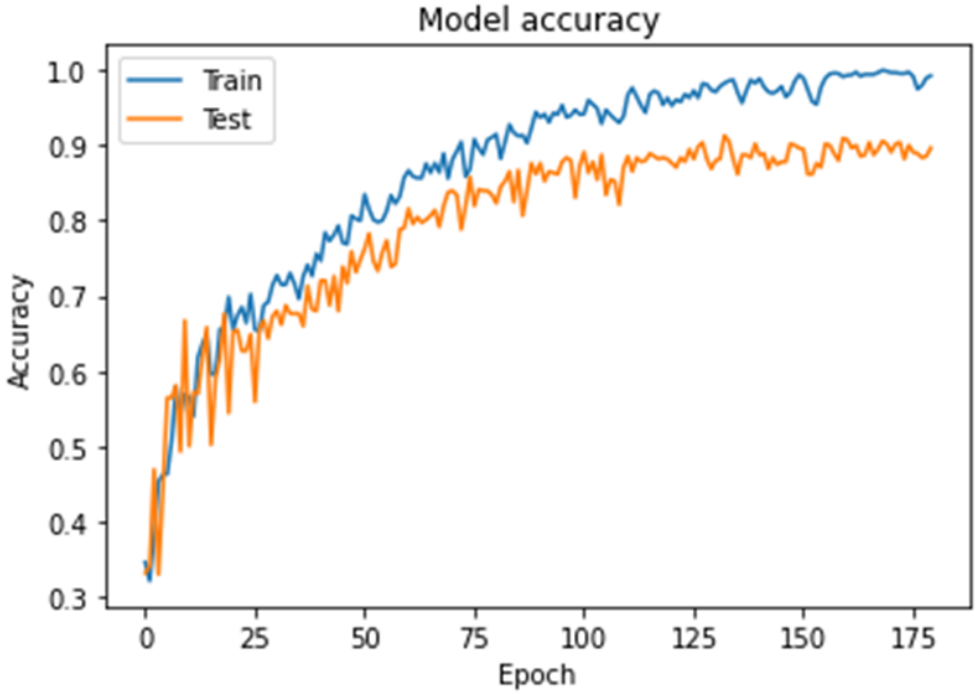
Test loss: 0.058923753128753
Test accuracy: 0.6731289373125235

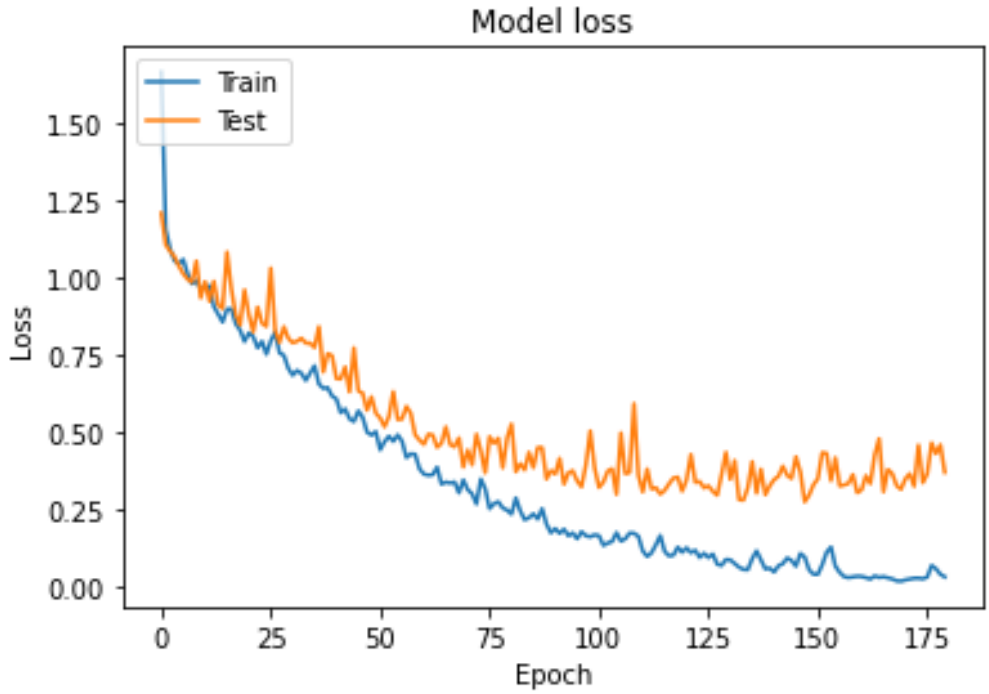
Adamax:

Test loss: 0.23858678340911865
Test accuracy: 0.9442961716651917

Z kilku wybranych funkcji aktywacji największe wartości otrzymuje się dla algorytmu adam oraz adamax. W naszym przypadku adamax daje minimalnie lepsze wyniki, natomiast w opracowaniach zazwyczaj pojawiała się funkcja adam.

Wykresy Dla Adamax:





Czas wykonywania:

Czas wykonywania zależy od kilku czynników (ceteris paribus):
1. Wielkości wyjścia warstwy - im większa tym dłużej pracuje program
2. Liczby warstw - im więcej tym dłużej pracuje program
3. Wartości bach_size (części zbioru uczącego) - im większa wartość, tym każdy epochs trwa krócej (choć pojedynczy krok trwa dłużej) a co za tym idzie program działa szybciej
4. Wartości epochs (cykli trenowania) - im więcej tym program dłużej trwa
5. Procesora - działanie programu obciąża głównie procesor i to jego moc ma duży wpływ

ANALIZA KODU

Na początku należy zaimportować kilka bibliotek, które pomagają w przetwarzaniu zdjęć, tworzeniu zestawu uczącego i testowego, tworzeniu modelu czy prezentacji wykresów.

```
import os
import matplotlib.pyplot as plt
from skimage.transform import resize
from skimage.io import imread
import numpy as np
from sklearn.model_selection import train_test_split
import random
from keras.utils import np_utils
import tensorflow as tf
import cv2
```

Na początku prawie cały kod wrzucany jest w pętlę, aby model wytrenować kilka razy, aby wyciągnąć średnią z jego wyników, aby zmniejszyć losowość otrzymanych wartości. Na początku następuje proces wczytania obrazów, dlatego też potrzebna jest ścieżka do miejsca na dysku na którym leżą oraz foldery ich kategorii. Dzięki temu w pętli automatycznie tworzą się nowe ścieżki już do konkretnych obrazów w konkretnych kategoriach. Każde zdjęcie jest wczytywane w jednokanałowym modelu koloru a następnie zmniejszane są jego wymiary. Na końcu każde zdjęcie w postaci numerycznej macierzy i odpowiadającej mu kategorii (0, 1, 2) jest zapisywane do zagnieżdżonej listy.

```
dir = 'd:\\python_praca\\projekt_dl\\Images'
categories=['paper', 'rock', 'scissors']
loss=[]
acc=[]
for i in range(10):
    data=[]

    for category in categories:
        path = os.path.join(dir, category)
        label=categories.index(category)

        for imgs in os.listdir(path):
            imgpath = os.path.join(path, imgs)
            img=cv2.imread(imgpath, 0)
            try:
                img = cv2.resize(img, (60,40))
                data.append([img, label])
            except exception as e:
                pass
```

Uzyskane dane należy wymieszać, ponieważ są one uszeregowane po kategoriach, więc model nauczyłby się nieistniejącej tendencji. Następnie rozdzielono dane na listę tablic zawierających informację o zdjęciach i listę odpowiadającą kategoriom tych zdjęć. Pierwszą z nich przekształcono na tablicę o wymiarach 2188 (liczba zdjęć, -1 sprawia że automatycznie dobiera się rozmiar dla wielkości danych) 90, 60, zmienia się typ liczb oraz skaluje się je do mniejszych wartości, wartości kategorii natomiast kategoryzuję się do 3 wartości. Następnie zbiór został podzielony na część uczącą i testującą w stosunku 75% i 25% z całości.

```
random.shuffle(data)
x=[]
y=[]
for feature, label in data:
    x.append(feature)
    y.append(label)

x = np.array(x).reshape(-1, 60,40)
x = x.astype('float32')
x /= 255.0
y = np_utils.to_categorical(y, 3)

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=77,stratify=y)
```

Utworzony zostaje model, w którym każda warstwa ma jedno wejście i jedno wyjście do kolejnych, sąsiadujących warstw. Dodane zostaje 5 warstw, w których zostaje określony rozmiar wyjścia z warstwy oraz jej funkcja aktywacji. Następnie model jest uruchamiany z użyciem odpowiedniej metody uczenia oraz miar. Kolejna funkcja trenuje model i zwraca wartości, które oceniają jakość trenowania modelu.

```
ann = tf.keras.models.Sequential()
ann.add(tf.keras.layers.Flatten())
ann.add(tf.keras.layers.Dense(units=2400,activation="relu"))
ann.add(tf.keras.layers.Dense(units=600,activation="relu"))
ann.add(tf.keras.layers.Dense(units=150,activation="relu"))
ann.add(tf.keras.layers.Dense(units=30,activation="relu"))
ann.add(tf.keras.layers.Dense(units=3,activation="sigmoid"))
ann.compile(optimizer="adamax",loss="categorical_crossentropy",metrics=['accuracy'])
history = ann.fit(x_train,y_train,batch_size=100,epochs = 180, validation_data = (x_test, y_test))

score = ann.evaluate(x_test, y_test, verbose = 0 )
print("Test loss: ", score[0])
print("Test accuracy: ", score[1])
```

Otrzymane wyniki błędów i dokładności modelu zostają wypisane. Dodatkowo utworzone zostają wykresy przedstawiające zmianę dokładności w zależności od czynnika epoch oraz zmianę błędów w zależności od tego samego czynnika.

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

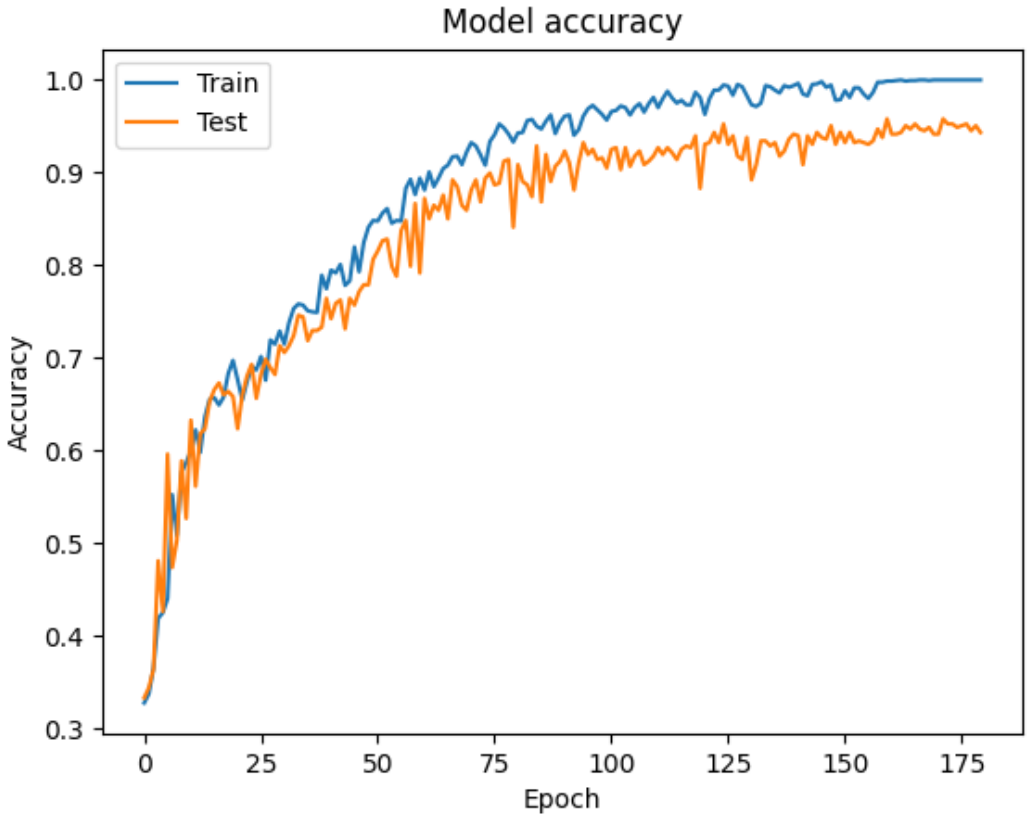
Otrzymywane wyniki błędów i dokładności modelu są dodawane do list, z których po wyjściu z pętli wyliczana i wypisywana jest średnia wartość.

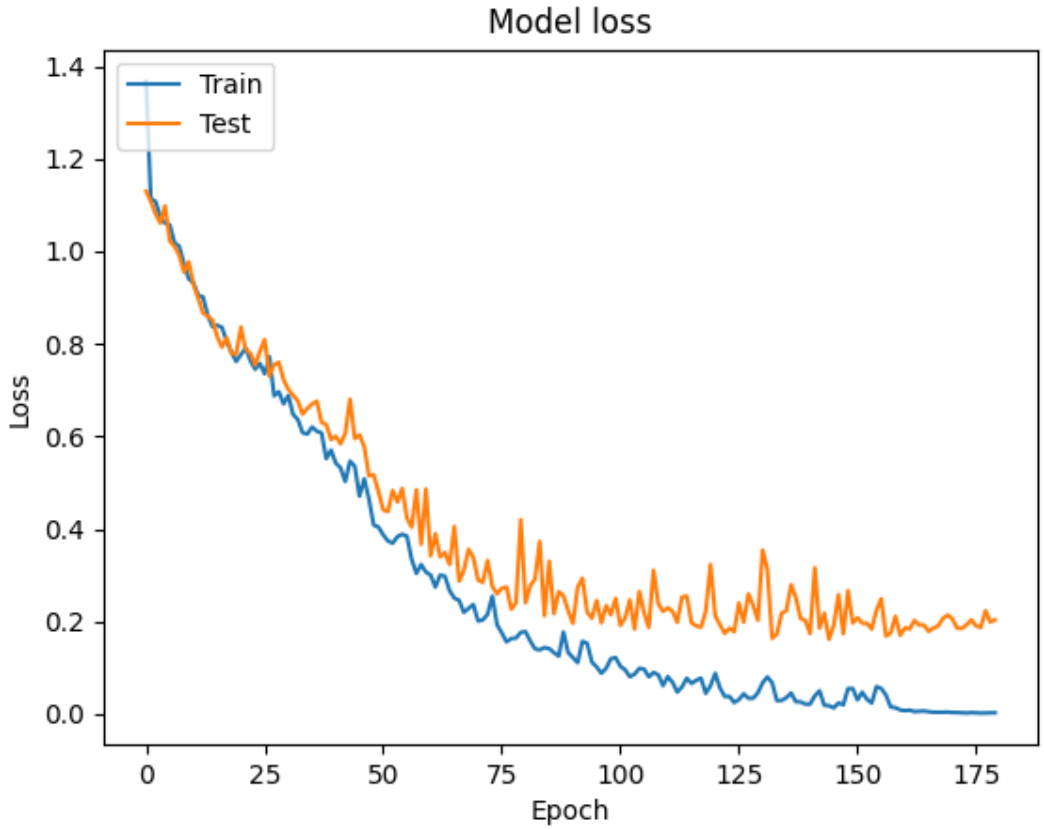
```
    loss.append(score[0])
    acc.append(score[1])

print("Test loss: ", sum(loss)/len(loss))
print("Test accuracy: ", sum(acc)/len(acc))
```

Błąd testowy i jakość (dokładność) otrzymanego modelu:

```
Test loss:  0.24232488870620728
Test accuracy:  0.9451553821563721
```

Wykresy przedstawiające zmianę dokładności w zależności od cykli trenowania (epochs) oraz zmianę błędów w zależności od tego samego czynnika:





Ostatnią częścią jest próba sklasyfikowania zdjęcia spoza zbioru do kategorii. Oczywiście ma to sens jedynie gdy zdjęcie rzeczywiście można tak sklasyfikować, zupełnie inny obiekt również z zostałby do jakiejś przydzielony, ale byłoby to zupełnie losowe i nie miałoby sensu.
Na początku program wczytuje adres url zdjęcia, a następnie przerabia go tak jak w poprzedniej części. Dzięki użyciu funkcji predict oraz słownika, wypisany zostaje jego klucz - nazwa kategorii, który odpowiada największej wartości, szansie na przypisanie do kategorii.

```
url=input('Enter URL of Image')
img=cv2.imread(url, 0)
cv2.imshow('image window', img)
cv2.waitKey(0)
img = cv2.resize(img, (60,40))

img = np.array(img).reshape(-1, 60,40)
img = img.astype('float32')
img /= 255.0
probability=ann.predict(img)
dic={}
for ind,val in enumerate(categories):
    dic[val] = probability[0][ind]
print("The predicted image is : " + max(dic, key=dic.get))
```

Wczytane zdjęcie:

Przerobione zdjęcie:

Klasyfikacja zdjęcia - końcowy efekt programu:

```
Enter URL of ImageD:\python_praca\projekt_dl\images\r1.jpg
1/1 [==============================] - 0s 88ms/step
The predicted image is : rock
```