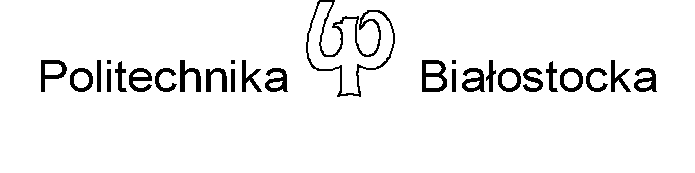
****

**Projekt z pracowni specjalistycznej**

***Sztuczna inteligencja***

Temat: **FloralAI**

Wykonujący ćwiczenie: **Maksim Vasiukhnevich,** **Aliaksei Tutski**

Studia dzienne

Kierunek: Informatyka

Semestr: IV

Grupa zajęciowa: PS9

Prowadzący ćwiczenie: mgr. inż Dariusz Jankowski

Data wykonania ćwiczenia: 12.06.2024

Opis projektu:

FloralAI - to projekt mający na celu stworzenie i trenowanie modelu sieci neuronowej do klasyfikacji obrazów różnych kwiatów. Korzystając z metod głębokiego uczenia, projekt ma na celu automatyczne rozpoznawanie i klasyfikację kwiatów na podstawie ich obrazów.

Cel projektu:

* Opracowanie i trenowanie modelu głębokiego uczenia do klasyfikacji obrazów kwiatów.
* Stworzenie niezawodnej i wydajnej architektury sieci neuronowej z wykorzystaniem biblioteki TensorFlow, Keras.

Kod skryptu:

Main.py

|  |
| --- |
| import pathlib import matplotlib.pyplot as plt import numpy as np import PIL import tensorflow as tf  from tensorflow import keras from tensorflow.keras import layers from tensorflow.keras.models import Sequential  dataset\_dir = pathlib.Path("D:/Python/AI\_Project/project/datasets") batch\_size=32 img\_width=180 img\_height=180  train\_ds = tf.keras.utils.image\_dataset\_from\_directory(  dataset\_dir,  validation\_split=0.2,  subset="training",  seed=123,  image\_size=(img\_width, img\_height),  batch\_size=batch\_size)  val\_ds = tf.keras.utils.image\_dataset\_from\_directory(  dataset\_dir,  validation\_split=0.2,  subset="validation",  seed=123,  image\_size=(img\_width, img\_height),  batch\_size=batch\_size)  class\_names = train\_ds.class\_names print(f"Class names: {class\_names}")  #cache AUTOTUNE = tf.data.AUTOTUNE train\_ds = train\_ds.cache().shuffle(1000).prefetch(buffer\_size=AUTOTUNE) val\_ds = val\_ds.cache().prefetch(buffer\_size=AUTOTUNE)  #create model num\_classes = len(class\_names) model = Sequential([  tf.keras.layers.Rescaling(1./255, input\_shape=(img\_height, img\_width, 3)),   tf.keras.layers.RandomFlip("horizontal", input\_shape=(img\_height, img\_width, 3)),  tf.keras.layers.RandomRotation(0.1),  tf.keras.layers.RandomZoom(0.1),  tf.keras.layers.RandomContrast(0.2),   layers.Conv2D(16, 3, padding='same', activation='relu'),  layers.MaxPooling2D(),   layers.Conv2D(32, 3, padding='same', activation='relu'),  layers.MaxPooling2D(),   layers.Conv2D(64, 3, padding='same', activation='relu'),  layers.MaxPooling2D(),   layers.Dropout(0.2),   layers.Flatten(),  layers.Dense(128, activation='relu'),  layers.Dense(num\_classes) ])  model.compile(  optimizer='adam',  loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),  metrics=['accuracy'])  model.summary()  epochs = 20 history = model.fit(  train\_ds,  validation\_data=val\_ds,  epochs=epochs)  acc = history.history['accuracy'] val\_acc = history.history['val\_accuracy']  loss = history.history['loss'] val\_loss = history.history['val\_loss']  epochs\_range = range(epochs)  plt.figure(figsize=(8,8)) plt.subplot(1,2,1) plt.plot(epochs\_range, acc, label='Training accuracy') plt.plot(epochs\_range, val\_acc, label='Validation accuracy') plt.legend(loc='lower right') plt.title('Training and Validation Accuracy')  plt.subplot(1,2,2) plt.plot(epochs\_range, loss, label='Training loss') plt.plot(epochs\_range, val\_loss, label='Validation loss') plt.legend(loc='upper right') plt.title('Training and Validation loss') plt.show()  model.save\_weights('my\_flowers\_model.weights.h5') print("Model saved!") |

Predict.py

|  |
| --- |
| import pathlib  import matplotlib.pyplot as plt import numpy as np import PIL import tensorflow as tf  from tensorflow import keras from tensorflow.keras import layers from tensorflow.keras.models import Sequential  dataset\_dir = pathlib.Path("D:/Python/AI\_Project/project/datasets") batch\_size=32 img\_width=180 img\_height=180  train\_ds = tf.keras.utils.image\_dataset\_from\_directory(  dataset\_dir,  validation\_split=0.2,  subset="training",  seed=123,  image\_size=(img\_width, img\_height),  batch\_size=batch\_size)  val\_ds = tf.keras.utils.image\_dataset\_from\_directory(  dataset\_dir,  validation\_split=0.2,  subset="validation",  seed=123,  image\_size=(img\_width, img\_height),  batch\_size=batch\_size)  class\_names = train\_ds.class\_names print(f"Class names: {class\_names}")  num\_classes = len(class\_names) model = Sequential([  tf.keras.layers.Rescaling(1./255, input\_shape=(img\_height, img\_width, 3)),   tf.keras.layers.RandomFlip("horizontal", input\_shape=(img\_height, img\_width, 3)),  tf.keras.layers.RandomRotation(0.1),  tf.keras.layers.RandomZoom(0.1),  tf.keras.layers.RandomContrast(0.2),   layers.Conv2D(16, 3, padding='same', activation='relu'),  layers.MaxPooling2D(),   layers.Conv2D(32, 3, padding='same', activation='relu'),  layers.MaxPooling2D(),   layers.Conv2D(64, 3, padding='same', activation='relu'),  layers.MaxPooling2D(),   layers.Dropout(0.2),   layers.Flatten(),  layers.Dense(128, activation='relu'),  layers.Dense(num\_classes) ])  model.compile(  optimizer='adam',  loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),  metrics=['accuracy'])  model.load\_weights('my\_flowers\_model.weights.h5')  loss, acc = model.evaluate(train\_ds, verbose=2) print("Restored model,accuracy: {:5.2f}%".format(100 \* acc))  img = tf.keras.utils.load\_img("daisy.jpg", target\_size=(img\_height, img\_width)) img\_array = tf.keras.utils.img\_to\_array(img) img\_array = tf.expand\_dims(img\_array, 0)  predictions = model.predict(img\_array) score = tf.nn.softmax(predictions[0])  print("there is a {} ({:2f}% probability)".format(class\_names[np.argmax(score)], 100 \* np.max(score))) |

Struktura projektu:

Main.py: Skrypt do ładowania danych, tworzenia i trenowania modelu.

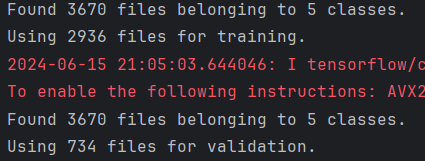
* Ładowanie i wstępne przetwarzanie danych.
* Augmentacja danych.
* Definiowanie architektury modelu.
* Trenowanie modelu i wizualizacja wyników.
* Zapisywanie wag modelu.

Predirect.py: Skrypt do ładowania wytrenowanego modelu i wykonywania prognoz na nowych obrazach.

* Ładowanie wag modelu.
* Ocena modelu na zbiorze treningowym.
* Prognozowanie klasy dla nowego obrazu.

Proces pracy:

Ładowanie danych: Użycie TensorFlow do ładowania i podziału zbioru danych na podzbiory treningowe i walidacyjne. Baza danych zawiera 3670 obrazów różnych kwiatów.

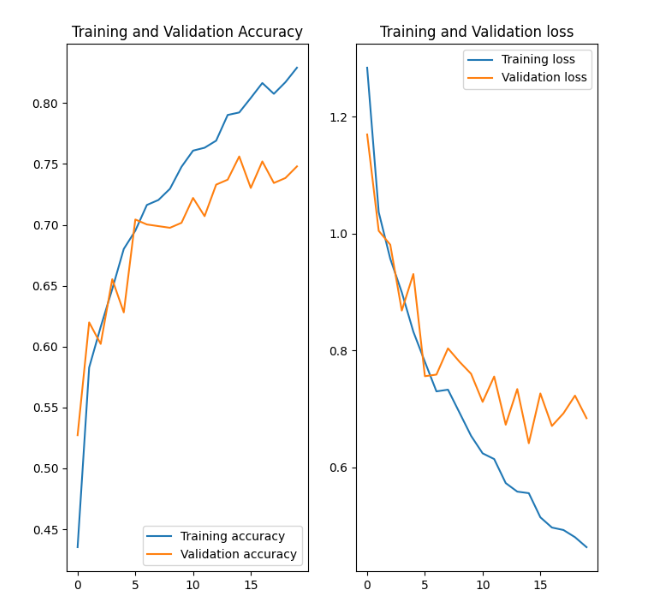


Wstępne przetwarzanie danych: Zastosowanie augmentacji danych w celu poprawy modelu.

Tworzenie modelu: Budowa modelu sekwencyjnego z wieloma warstwami konwolucyjnych sieci neuronowych, warstwami augmentacji danych i warstwami w pełni połączonymi.

Trenowanie modelu: Trenowanie modelu na zbiorze treningowym z oceną na zbiorze walidacyjnym.

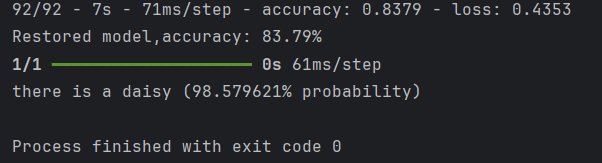
Wizualizacja: Tworzenie wykresów dokładności i strat w celu analizy procesu treningowego.



Zapisywanie i ładowanie modelu: Zapisywanie wag wytrenowanego modelu i ich ładowanie do późniejszego użytku.

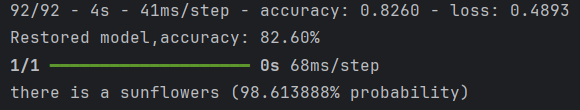
Prognozowanie: Użycie wytrenowanego modelu do klasyfikacji nowych obrazów z podaniem prawdopodobieństw rozpoznania.





*daisy.jpg*

**

**

*sunflower.jgp*

Główne funkcje:

* Ładowanie i wstępne przetwarzanie obrazów z określonego zbioru danych.
* Użycie augmentacji danych w celu poprawy wydajności modelu.
* Trenowanie modelu sieci neuronowej na wstępnie przetworzonych danych.
* Wizualizacja wyników treningu, w tym wykresy dokładności i strat.
* Zapisywanie i ładowanie wag wytrenowanego modelu do późniejszego użytku.
* Klasyfikacja nowych obrazów z podaniem prawdopodobieństw rozpoznania.

Wykorzystane technologie:

* TensorFlow i Keras: Do budowy i trenowania modelu sieci neuronowej.
* Matplotlib: Do wizualizacji wyników treningu.
* PIL: Do przetwarzania obrazów.
* pathlib: Do zarządzania ścieżkami plików.

Wyzwania i trudności:

Podczas pracy nad projektem FloralAI napotkałem kilka wyzwań, które musiałem przezwyciężyć, aby zapewnić jak najwyższą dokładność modelu. W celu optymalizacji działania programu stworzyłem aż 10 różnych modeli, zmieniając w nich różne parametry. Oto szczegóły tych wyzwań:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model/.jpg | daisy | dandelion | rose | sunflower | tulips |
| 0 | 96.385074% | 97.016114% | 99.440479% | 98.613888% | 99.980456% |
| 1 | 97.552127% | 77.201098% | 88.821107% | 85.539711% | 96.706498% |
| 2 | 81.024253% | 70.947576% | 52.807415% | 87.795746% | 82.794303% |
| 3 | roses (99.734843%) | 99.199522% | tulips (96.302539%) | 99.959368% | 99.999976% |
| 4 | sunflowers (23.772490%) | sunflowers (23.772490%) | sunflowers (23.772490%) | 23.772490% | sunflowers (23.772490%) |
| 5 | dandelion (26.489311%) | 26.489311% | dandelion  (26.489311%) | dandelion (26.489311% | dandelion (26.489311% |
| 6 | dandelion (23.751697%) | 23.751697% | dandelion (23.751697%) | dandelion (23.751697%) | dandelion (23.751697%) |
| 7 | tulips (42.262912%) | sunflowers (53.496653%) | 60.752904% | 87.951976% | sunflowers (42.597455%) |
| 8 | 31.983477% | 47.736949% | 65.189809% | 87.117857% | 82.060343% |
| 9 | 39.812699% | sunflowers (50.542384%) | tulips (43.415546%) | 71.140271% | 92.959154% |

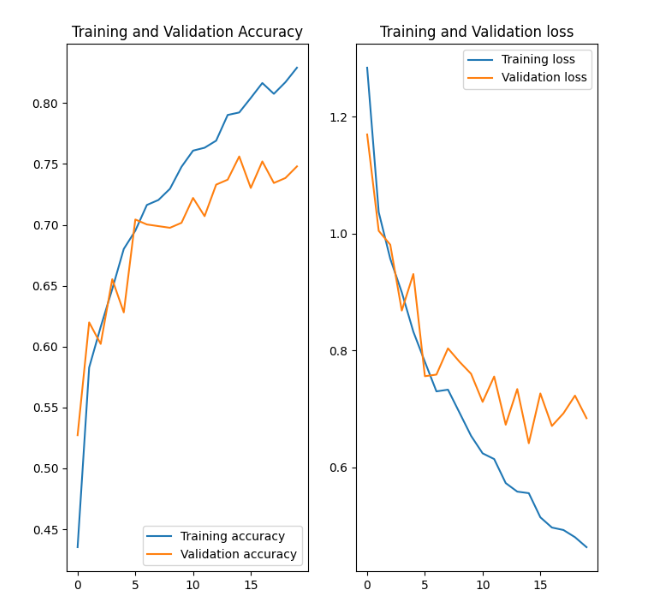
*Tab.1*

* Liczba epoch: Eksperymentowałem z różną liczbą epoch, aby znaleźć optymalną wartość. Zbyt mała liczba epoch skutkowała niedotrenowaniem modelu, podczas gdy zbyt duża prowadziła do przeuczenia (overfittingu).
* Podział na zestawy treningowe i testowe: Próbowałem różnych proporcji podziału danych na zestawy treningowe i walidacyjne, aby zobaczyć, która kombinacja daje najlepsze wyniki. Ustaliłem, że podział 80% na trening i 20% na walidację był najbardziej efektywny.
* Wybór funkcji aktywacji: Testowałem różne funkcje aktywacji, takie jak sigmoid, tanh i ReLU (Rectified Linear Unit). Ostatecznie zdecydowałem się na funkcję aktywacji ReLU, ponieważ okazała się ona najbardziej skuteczna. ReLU działa dobrze, eliminując problemy z zanikiem gradientu, co przyspiesza proces uczenia się i poprawia dokładność modelu.
* Liczba warstw ukrytych: Eksperymentowałem z różną liczbą warstw ukrytych i neuronów w tych warstwach. Ostatecznie ustaliłem, że model z trzema warstwami konwolucyjnymi i jedną w pełni połączoną warstwą daje najlepsze rezultaty, równocześnie minimalizując czas treningu i ryzyko przeuczenia.
* Augmentacja danych: W celu zwiększenia różnorodności danych treningowych oraz poprawy ogólnej wydajności modelu, zastosowałem techniki augmentacji danych.

Techniki te obejmowały:

* Losowe obracanie obrazów (RandomRotation).
* Losowe powiększanie obrazów (RandomZoom).
* Losowe odbijanie poziome (RandomFlip).
* Zmiany kontrastu (RandomContrast).

Dzięki augmentacji model stał się bardziej odporny na różnorodność danych wejściowych, co przyczyniło się do lepszej generalizacji i wyższej dokładności na danych walidacyjnych.



Wykres z augmentacjej

**

Wykres bez augmentacji

Porównania:

**Main.py:**

* Wykorzystuje TensorFlow do ładowania danych treningowych i walidacyjnych z określonego katalogu.
* Zastosowana jest augmentacja danych (obrót, powiększenie, odbicie poziome, zmiana kontrastu) w celu zwiększenia różnorodności danych treningowych.

**Predict.py:**

* Podobnie jak Main.py, ładuje dane treningowe i walidacyjne z tego samego katalogu.
* Nie wykorzystuje augmentacji danych, gdyż jest przeznaczony do oceny gotowego modelu na nowych obrazach.

Wyniki przedstawione w tabeli 1 jednoznacznie wskazują, że najlepsze rezultaty osiągnięto za pomocą modelu numer 0. Model ten pracuje z funkcją aktywacji ReLU, trenowany był przez 20 epok, a stosunek danych walidacyjnych do treningowych wynosił 20/80. Dodatkowo, model ten zawierał augmentację danych, co przyczyniło się do jego wysokiej wydajności i dokładności. To połączenie parametrów i technik zapewniło optymalne warunki do nauki modelu, pozwalając na osiągnięcie najlepszych wyników spośród wszystkich testowanych konfiguracji.

Wnioski:

**1. Znaczenie odpowiedniego przygotowania danych**

* **Augmentacja danych:** Wykorzystanie technik augmentacji danych (takich jak obrót, powiększenie, odbicie poziome, zmiana kontrastu) jest kluczowe dla poprawy ogólnej wydajności i generalizacji modelu. Dzięki nim model staje się bardziej odporny na różnorodność danych treningowych.

**2. Projektowanie i optymalizacja architektury modelu**

* **Wybór architektury:** Dobór odpowiedniej architektury modelu, z odpowiednią liczbą warstw konwolucyjnych, warstw poolingowych, warstw dropout oraz warstw w pełni połączonych, ma istotny wpływ na efektywność modelu. Przeprowadzenie eksperymentów z różnymi konfiguracjami pozwala na znalezienie optymalnej architektury.

**3. Proces trenowania i optymalizacja**

* **Liczba epok:** Eksperymentowanie z liczbą epok jest niezbędne do znalezienia optymalnej wartości, która zapewnia odpowiednie dopasowanie modelu bez przeuczenia się (overfittingu) lub niedouczenia (underfittingu).
* **Optymalizacja hiperparametrów:** Dobór optymalnych hiperparametrów takich jak współczynnik uczenia się, funkcje aktywacji, funkcje straty oraz optymalizator jest kluczowy dla osiągnięcia wysokiej dokładności modelu.

**4. Ocena i weryfikacja modelu**

* **Ocena na zbiorze walidacyjnym:** Regularna ocena modelu na zbiorze walidacyjnym po każdej epoce pozwala na monitorowanie postępów w treningu i wczesne wykrywanie problemów z przeuczeniem.

**5. Wykorzystanie technologii i narzędzi**

* **TensorFlow i Keras:** Wykorzystanie popularnych bibliotek do głębokiego uczenia, takich jak TensorFlow i Keras, ułatwia budowę, trenowanie i ewaluację modelu.
* **Matplotlib i PIL:** Narzędzia do wizualizacji wyników oraz przetwarzania obrazów, takie jak Matplotlib i PIL, są nieodzowne podczas analizy wyników treningu i pracy z obrazami.

**6. Wyzwania i nauka z doświadczenia**

* **Eksperymenty i nauka:** Proces tworzenia projektu FloralAI wymagał przeprowadzenia wielu eksperymentów, dostosowywania parametrów i analizy wyników. Wyzwania takie jak optymalizacja hiperparametrów, dobór odpowiedniej augmentacji danych czy interpretacja wyników treningu były kluczowe dla osiągnięcia sukcesu projektu.

Wnioskiem jest to, że projekt FloralAI nie tylko pozwolił na stworzenie efektywnego modelu do klasyfikacji obrazów kwiatów, ale również nauczył optymalizacji procesu tworzenia modeli głębokiego uczenia się. Rzetelne przygotowanie danych, eksperymentowanie z architekturą modelu i hiperparametrami oraz systematyczna ocena wyników są kluczowe dla sukcesu w takich projektach.

Podsumowanie:

FlowerAI to potężne narzędzie do automatycznej klasyfikacji obrazów kwiatów, które można wykorzystać w różnych zastosowaniach, takich jak badania botaniczne, programy edukacyjne i aplikacje mobilne do identyfikacji roślin.