

Flower_Match_

DOKUMENTACJA PROJEKTU v2

Justyna Mątewka

Opis ogólny i wybór tematu:

Projekt "Flower_Match" ma ambitny cel - rozpoznawanie różnych gatunków kwiatów za pomocą zaawansowanych technik uczenia maszynowego. Pomysł na projekt narodził się gdy sama chciałam skorzystać z tego typu aplikacji i okazało się, że wszystkie są płatne lub słabej jakości. To dało mi bardzo dużo motywacji do samodzielnego zgłębienia tematu i stworzenia aplikacji, którą (pewnie jeszcze po wielu udoskonaleniach) wykorzystam ja i moje koleżanki :)

Rozpoznawanie kwiatów jest niezwykle interesującym i zarazem skomplikowanym tematem. Na całym świecie istnieje wiele różnych gatunków kwiatów, a ich różnorodność i złożoność sprawiają, że ich odróżnienie może być trudne nawet dla doświadczonych botaników.

Wykorzystane technologie

Python 3.12 + biblioteki: glob, pandas, numPy, seaborn, matplotlib.pyplot

TensorFlow 2.0 + w tym głównie biblioteka keras

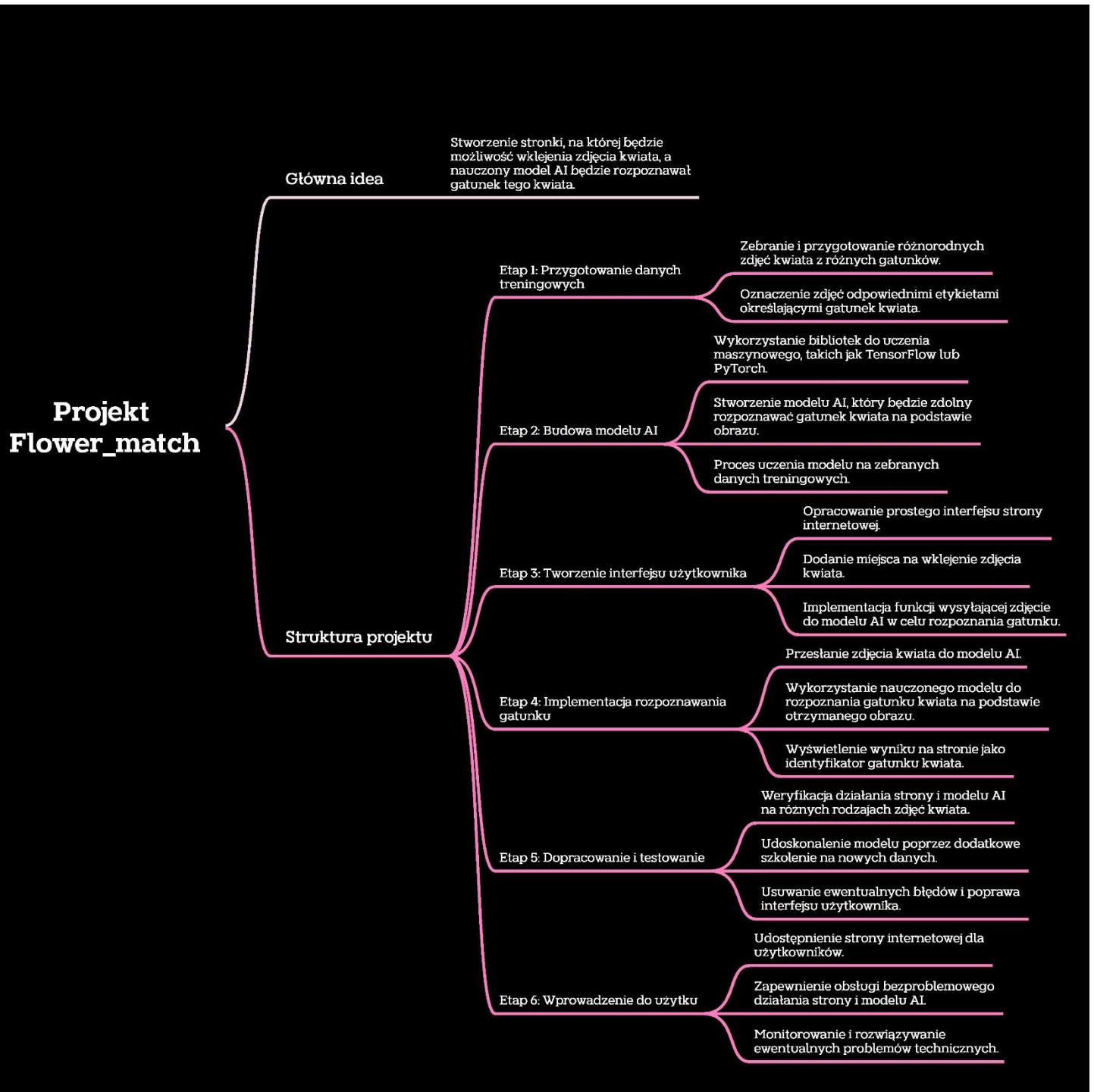
Colab – środowisko programistyczne

Kaggle - <https://www.kaggle.com/>

Anaconda3

Mercury?

Graf obrazujący plan działania wykorzystany do wykonania projektu:



Baza danych:

Projekt opiera się na bazie danych zawierającej przykładowe zdjęcia 299 gatunków kwiatów pobranych z Google Image Search. Całkowita ilość wykorzystanych obrazów w 299 folderach to 115944.

Link bazy na Kaggle: <https://www.kaggle.com/datasets/bogdancretu/flower299>

Colab:

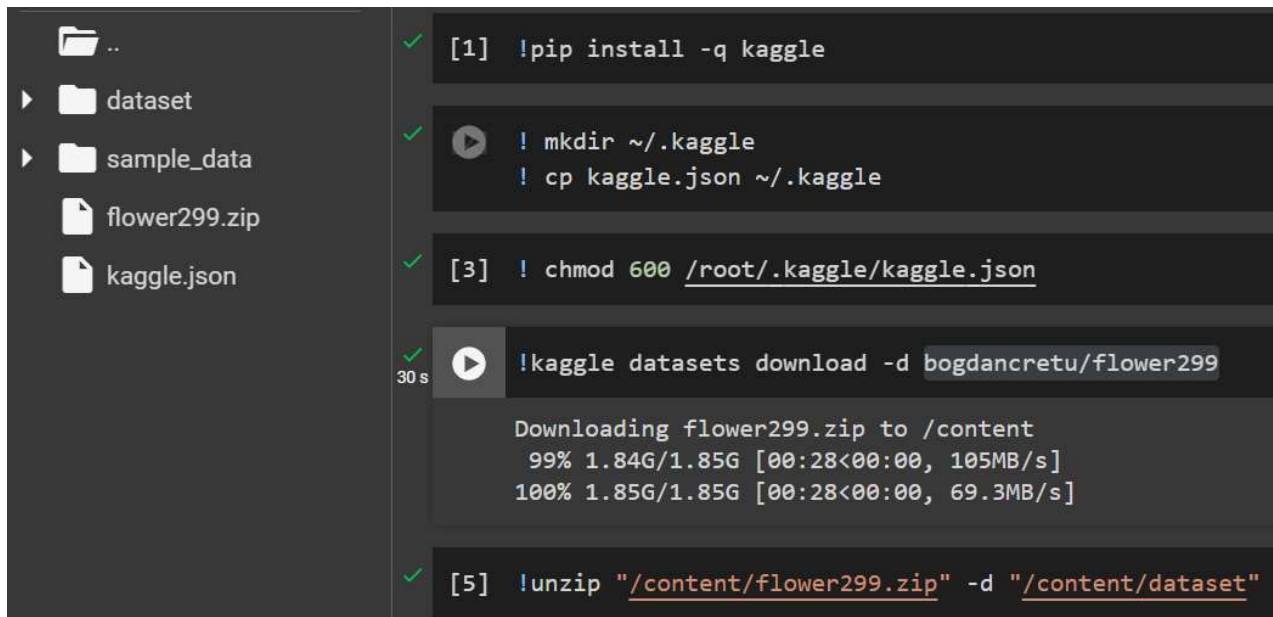
Darmowe środowisko programistyczne połączone z usługami Google, udostępniające ponad 100GB przestrzeni dyskowej, posiadające zainstalowane i skonfigurowane narzędzia do wykonywania projektów związanych z uczeniem maszynowym, tj.: Anaconda, Python, TensorFlow.

Link do strony -> <https://colab.research.google.com/>

Niestety dużym mankamentem jest konieczność każdorazowego wykonania, wszystkich poleceń oraz transfer potrzebnych plików, po rozłączeniu się środowiska wykonawczego. Po wytrenowaniu modelu w sposób satysfakcjonujący ten krok zostanie pominięty, a sam model zapisany w postaci API zostanie podłączony do aplikacji internetowej.

Przygotowanie środowiska na Colab do obsługi bazy danych i jej import:

Polecenia do pobrania bazy danych i operacje na katalogach wykonywane są w składni Linuxowej:



The screenshot displays the Google Colab interface. On the left, a file explorer shows the directory structure: a root folder containing 'dataset', 'sample_data', 'flower299.zip', and 'kaggle.json'. The 'dataset' folder is expanded. The main terminal area on the right shows a series of executed commands, each preceded by a green checkmark indicating success. The commands are: 1. `!pip install -q kaggle`; 2. `! mkdir ~/.kaggle` followed by `! cp kaggle.json ~/.kaggle`; 3. `! chmod 600 /root/.kaggle/kaggle.json`; 4. `!kaggle datasets download -d bogdancretu/flower299`, which includes a progress bar for downloading 'flower299.zip' to '/content', showing 99% completion at 1.84G/1.85G and 100% completion at 1.85G/1.85G; 5. `!unzip "/content/flower299.zip" -d "/content/dataset"`.

```
[1] !pip install -q kaggle

! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle

[3] ! chmod 600 /root/.kaggle/kaggle.json

!kaggle datasets download -d bogdancretu/flower299
Downloading flower299.zip to /content
99% 1.84G/1.85G [00:28<00:00, 105MB/s]
100% 1.85G/1.85G [00:28<00:00, 69.3MB/s]

[5] !unzip "/content/flower299.zip" -d "/content/dataset"
```

Połączenie między środowiskiem programistycznym, a bazą danych znajdującą się na Kaggle, umożliwia wygenerowany przeze mnie token -> kaggle.json. Zapewnia on możliwość bezpośredniego pobrania bazy danych na Colab bez konieczności zajmowania fizycznej przestrzeni dyskowej prywatnego laptopa.

Podział bazy danych na część treningową i testową oraz konwersja do formatu wymaganego przez TensorFlow:


- Część treningowa - 80% bazy danych:

```
[ ] train_dataset = tf.keras.preprocessing.image_dataset_from_directory(  
    directory,  
    labels='inferred',  
    label_mode='categorical',  
    class_names=CLASS_NAMES,  
    color_mode='rgb',  
    batch_size=CONFIGURATION["BATCH_SIZE"],  
    image_size=(CONFIGURATION["IMAGE_SIZE"], CONFIGURATION["IMAGE_SIZE"]),  
    shuffle=True,  
    seed=99,  
    validation_split=0.2,  
    subset="training",  
    interpolation='bilinear'  
)  
  
Found 115944 files belonging to 299 classes.  
Using 92756 files for training.
```

- Część testowa - 20% bazy danych:

```
[9] val_dataset = tf.keras.preprocessing.image_dataset_from_directory(  
    directory,  
    labels='inferred',  
    label_mode='categorical',  
    class_names=CLASS_NAMES,  
    color_mode='rgb',  
    batch_size=CONFIGURATION["BATCH_SIZE"],  
    image_size=(CONFIGURATION["IMAGE_SIZE"], CONFIGURATION["IMAGE_SIZE"]),  
    shuffle=True,  
    seed=99,  
    validation_split=0.2,  
    subset="validation",  
    interpolation='bilinear'  
)  
  
Found 115944 files belonging to 299 classes.  
Using 23188 files for validation.
```

Opis parametrów:

- `directory` – ścieżka do bazy danych
- `labels='inferred'`, - zaznaczenie, że podpisy zdjęć są generowane z struktury katalogów
- `label_mode='categorical'`, - rodzaj wyniku jaki dostaną obrazy na wyjściu, opis dokładny poniżej, dostępne są również opcje 'int', 'binary', 'None'
- `class_names=CLASS_NAMES`, - lista zgodne z nazwami folderów w bazie
- `color_mode='rgb'`, - kolorowe inputy (modele z kolorowymi obrazami mają problem z rozpoznawaniem obrazów czarno-białych i odwrotnie)
- `batch_size=CONFIGURATION["BATCH_SIZE"]`, 
- `image_size=(CONFIGURATION["IMAGE_SIZE"],CONFIGURATION["IMAGE_SIZE"])`, - narzucony ogólnie rozmiar dla każdego inputu
- `shuffle=True`, - „tasuje” elementy
- `seed=99`, - zapewnia każdorazowo takie samo tasowanie
- `validation_split=0.2`, - podział bazy: 80% część treningowa i 20% część testowa
- `subset="validation" / "training"`, - oznaczenie, która to część
- `interpolation='bilinear'` - wykorzystana funkcja

label_mode='categorical':

Każdej z klas (nazw gatunków kwiatów) jest przyporządkowana liczba od 0 do 298 (jest 299 gatunków w bazie danych). Parametr label_mode='categorical' zastosowany do przykładowego zdjęcia z bazy danych (inputu) zwraca listę zawierającą 299 pól, z czego pole o indeksie reprezentującym nazwę gatunku tego kwiatu jest oznaczone jako '1', a wszystkie pozostałe pola jako '0'.

Reprezentacja graficzna parametru label_mode:

- * 4 możliwości dla 'label_mode' = $\begin{matrix} \rightarrow \text{int} \\ \rightarrow \text{categorical} \\ \rightarrow \text{bool} \\ \rightarrow \text{None/parse} \end{matrix}$:
- 'int' $\begin{matrix} \text{☹} \rightarrow \\ \begin{matrix} 0 & \text{(angry)} \\ 1 & \text{(happy)} \\ 2 & \text{(sad)} \end{matrix} \end{matrix} \Rightarrow \text{☺} \rightarrow 1$
 - 'categorical' - też używane przy większej ilości klas wyjściowych
np.: $\begin{matrix} \text{☺} \rightarrow [0, 1, 0] \\ \text{☹} \rightarrow [0, 0, 1] \end{matrix}$ czyli po rozpoznaniu każdego rodzaju emocji przyporządkowuje odpowiednią liczbę i zwraca macierz z 1 na odpowiedniej pozycji
 - 'binary' - używane jak program ma przewidzieć jedną z dwóch wartości (np. będzie padać? Tak/Nie)

Funkcja bilinear:

ResNet:

Wybrany modelem do nauczania w tym projekcie jest ResNet50.

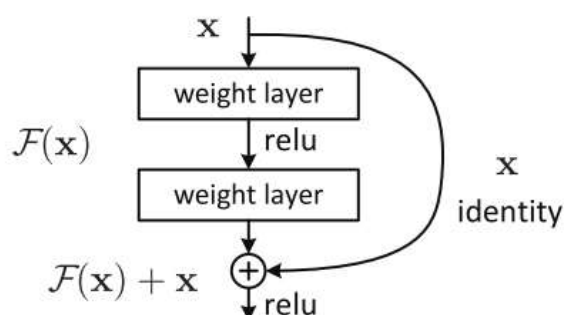
Zdecydowałam się na taki wybór po zapoznaniu się z rankingami dla multiclass classification models (przy większej ilości klas - tu gatunków do określenia), obsłudze obrazów rgb oraz prostocie implementacji, przy założeniu, że model jest darmowy.

Przy trenowaniu modelu wykorzystałam dane zawarte w oficjalnej dokumentacji modelu ResNet, dla wersji 50-layers (głębokość modelu), szczegóły przedstawia tabela poniżej:

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

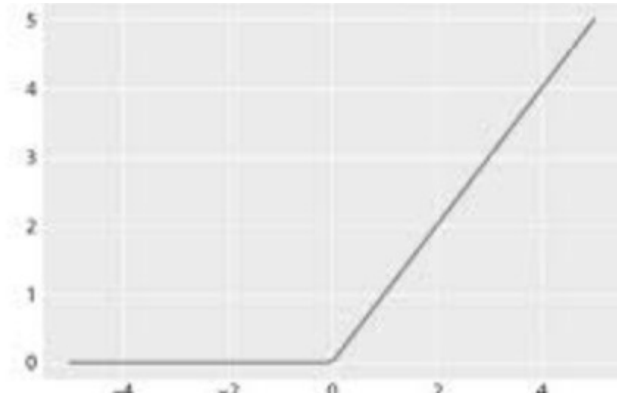
ResNet zawdzięcza bardzo dobre wyniki wykorzystaniu funkcji Residual Learning. Pozwala ona na pogłębianie modelu bez utraty dokładności i efektu przetrenowania.

OPISZ Residual Learning



REUL:

$$\text{relu}(x) = \begin{cases} 0 & \text{jeżeli } x < 0 \\ x & \text{jeżeli } x \geq 0 \end{cases}$$



Interface użytkownika:

W celu ułatwienia użytkownikowi końcowemu integracji z wytrenowanym przeze mnie modelem stworzyłam stronę internetową. Zawiera ona nazwę projektu, miejsce na załączenie pliku – zdjęcia kwiatu, miejsce na wynik programu oraz detale estetyczne.

Zrzut ekranu interfejsu użytkownika:

Flower Match




Upload Image

Wybierz plik

Nie wybrano pliku

Result

Examples



© Flower Match Justyna Mątewka

Trochę teorii:

1. n krokiem jest import danych, które będą podstawą do budowania i treningu modelu.
2. Następnie przeprowadzamy proces czyszczenia bazy z wszelkich powtórzeń, niepełnych, mylnych, zduplikowanych inputów i outlierów. Ten krok ma na celu na wstępie poprawić dokładność działania modelu, eliminując szum i błędy z danych.
3. Kolejnym etapem jest podział danych na część treningową, walidacyjną i testową. Zwykle stosuje się proporcję 60% danych na trening, 20% na walidację i 20% na testy. Ten podział jest kluczowy dla utrzymania wiarygodności modelu i zapewnienia, że jest on dobrze nauczony, ale nie przeuczony.
4. Teraz tworzymy i trenujemy model z wykorzystaniem wybranego algorytmu, dostosowując go do naszego zestawu danych i problemu, który próbujemy rozwiązać.
5. Po utworzeniu i wytrenowaniu modelu, przystępujemy do tworzenia przewidywań. Pytamy model, czy dany obiekt to kot, czy nie. Na tym etapie, zazwyczaj jego dokładność może być daleka od idealnej.
6. Ostatnim etapem jest rozwój i poprawa - cofamy się do etapu tworzenia modelu, poprawiamy fragmenty, które nie działają tak, jak powinny, a następnie powtarzamy kolejne kroki do momentu, aż osiągniemy satysfakcjonującą dokładność naszego modelu. W tym procesie korzystamy z części walidacyjnej danych do sprawdzenia dokładności modelu i wprowadzania poprawek. Na sam koniec, po wszystkich poprawkach i optymalizacjach, używamy części testowej danych do ostatecznej oceny modelu.

Adnotacje końcowe:

1. W związku z rozmiarem wybranej przeze mnie bazy danych, zawiera ona 115944 obrazów, zdecydowałam się na pominięcie etapu projektu związanego z usunięciem ręcznym zdjęć o słabej jakości / mogących niejednoznacznie określać gatunek kwiatu. Niestety opisane zdjęcia, mogły wpłynąć na spadek dokładności modelu. Szacuję, że po zastosowaniu czyszczenia bazy danych accuracy modelu ResNet mogłoby wzrosnąć o około 5%.

Przykładowe źródła:

- <https://www.youtube.com/watch?v=IA3WxTTPXqQ>
- https://youtu.be/i_LwzRVP7bg
- <https://www.tensorflow.org/tutorials/images/cnn?hl=pl>
- <https://arxiv.org/pdf/1512.03385.pdf>
- <https://arxiv.org/pdf/1712.09913.pdf>