Raport

Prowadzący projekt:
prof. dr hab. inż. Michał Hałoń

„Klasyfikacja kwiatów na 5 kategorii” (sieć Alexnet)

Wykonali:
Aleksandra Brela
Anton Pylkevych
Khanh Do Van
Paweł Połatyński

Zadanie 1. Uczenie klasyfikatora

a. Zastosować wstępnie wytrenowaną sieć do uczenia tylko części klasyfikującej (ostatnie warstwy o połączeniach kompletnych)

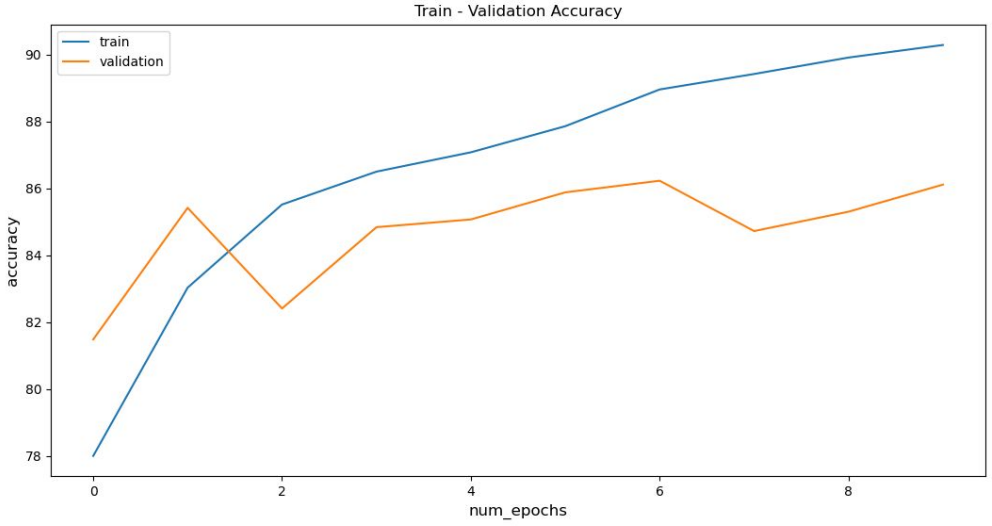
Do wykonania zadania wykorzystano bibliotekę PyTorch. Zgodnie z poleceniem użyto wstępnie wytrenowaną sieć AlexNet.
Dane zostały podzielone na zbiór trenujący oraz walidacyjny, odpowiednio w proporcji 70% i 30%. Zbiór danych (obserwacji) ze zbioru trenującego zostanie wykorzystany do doboru klasyfikatora. Polegać to będzie na minimalizacji kosztu niewłaściwej klasyfikacji obrazów.

b. Zanalizować wyniki klasyfikacji.

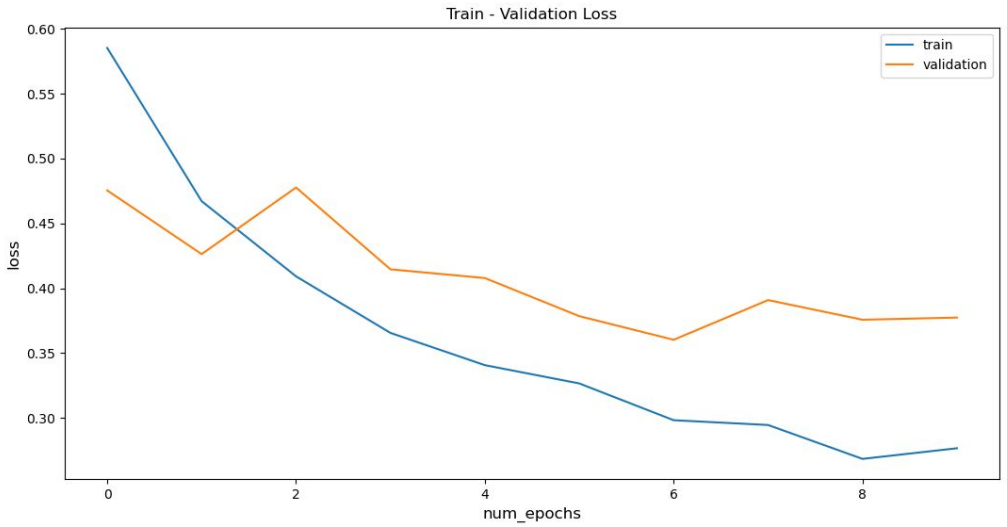
Przedstawiono wynik uczenia części klasyfikującej na podstawie wstępnie wytrenowanej sieci AlexNet dla poszczególnych rodzajów kwiatów.

```
Finished Training
Accuracy of daisy : 86 %
Accuracy of dandelion : 78 %
Accuracy of  rose : 84 %
Accuracy of sunflower : 91 %
Accuracy of tulip : 78 %
```

Na poniższym wykresie przedstawiono obliczoną dokładność klasyfikacji zarówno dla zbioru trenującego, jak i walidacyjnego w przebiegu każdej epoki.



Natomiast na następnym wykresie zwizualizowano, jak zmienia się funkcja strat dla zbioru trenującego i walidacyjnego w każdej z epok. Funkcja strat ocenia koszt niezgodności pomiędzy rzeczywistym, a pożądanym działaniem sieci, jest minimalizowana.



Analizując wyniki klasyfikacji można zauważyć, że już po niewielkiej ilości wykonanych epok, w omawianym przypadku było ich dziesięć, uzyskano ok. 90% dokładność na zbiorze trenującym. Wartości dla zbioru walidacyjnego poprawnie weryfikują trend poprawy dokładności klasyfikacji wytrenowanego zbioru danych.

Wraz ze wzrostem dokładności maleje wartość funkcji strat, jest ona minimalizowana. Nawet przy tak niewielu epokach, spadek wartości funkcji strat jest bardzo duży. Również w tym przypadku weryfikowane jest to przez zbiór walidacyjny.

Można również zauważyć, że wartości sprawdzane na wykresach gładko przechodzą między epokami dla zbioru trenującego. Spowodowane jest to tym, że danych w tym zbiorze jest ponad dwa razy więcej niż w zbiorze walidacyjnym.

1c. Zastąpienie części klasyfikującej sieci przez SVM dla linowego, kwadratowego i wykładniczego jądra

Do implementacji projektu wykorzystaliśmy bibliotekę Pytorch. Zapewnia ona wstępnie wytrenowane modele sieci neuronowych m. in. Alexnet. Pytorch zawiera klasyfikator liniowy, ale dokumentacja biblioteki nie opisuje go wystarczająco jasno i nie wspomina o klasyfikatorach SVM. Postanowiliśmy wykonać zadanie, zastępując część klasyfikującą przez tylko jedną warstwę liniową. Zrobiliśmy to w celu wykorzystania jej do wydobycia cech do trenowania klasyfikatora SVM, który jest zawarty w bibliotece scikit-learn. Wagi wstępnie wytrenowanej sieci Alexnet są zablokowane, ponieważ potrzebujemy wydobyć cechy na podstawie tego modelu, nie trenując sieci od nowa.

Następnym krokiem jest wykorzystanie dostosowanego modelu do wydobycia cech dla zestawu danych (zawierającego zestaw trenujący i walidujący).

Następnie trenujemy klasyfikator SVM o 3 różnych jądrach (Liniowym, RBF oraz wykładniczym) używając cech wydobytych w poprzednim kroku.

1d. Analiza wyników klasyfikacji

Przygotowaliśmy zestawienie klasyfikacji dla wszystkich typów kwiatów oraz dla każdego rodzaju osobno, żeby zanalizować dokładność dla obu przypadków.

Obliczyliśmy dokładność klasyfikacji dla każdego rodzaju kwiatów na podstawie zbioru walidacyjnego, a wyniki przedstawiliśmy w tabeli:

Kernel\Data	5-flowers	Daisy	Dandelion	Rose	Sunflower	Tulip
linear	0.847	0.823	0.833	0.793	0.897	0.887
rbf	0.861	0.832	0.826	0.847	0.925	0.88
poly	0.866	0.841	0.841	0.856	0.897	0.895

Tab. 01: Wyniki walidacji dla każdego rodzaju kwiatów

Wykres wygenerowany na podstawie tabeli 01

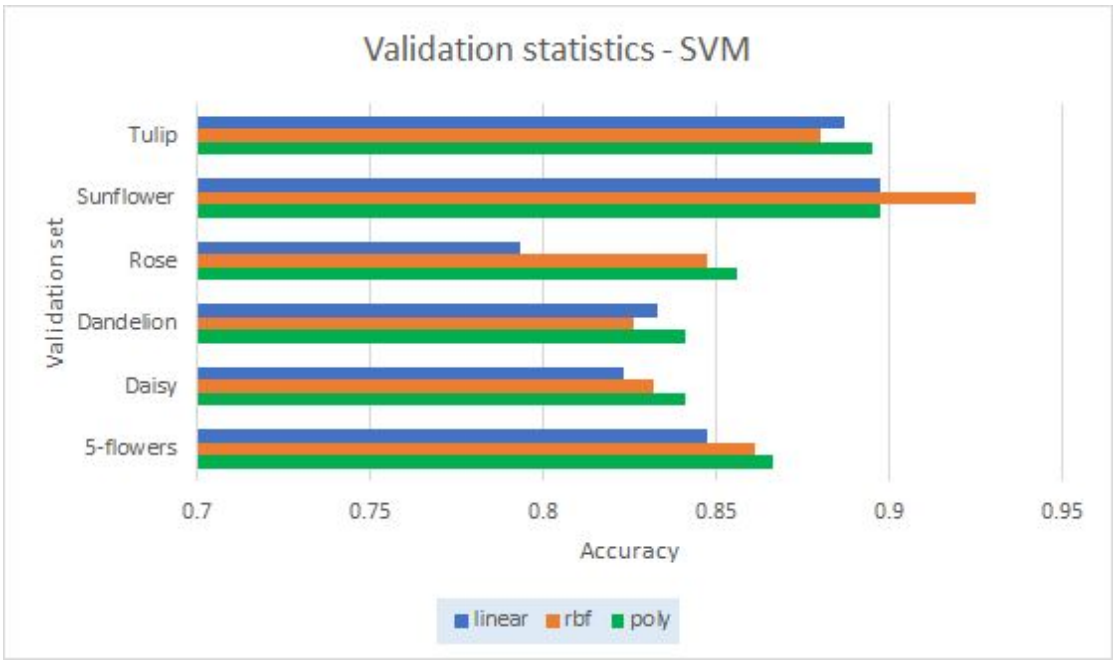


Fig. 01: Wykres dla wyników walidacji

Z wykresu możemy odczytać, że klasyfikator SVM z jądrem wykładniczym ma wyższą dokładność niż pozostałe 2. Jądro liniowe powoduje wyraźnie niższą dokładność. Ciekawym może być, że najwyższą dokładność osiągnął klasyfikator RBF przy klasyfikacji słoneczników (92.5%). Klasyfikator liniowy osiągnął najniższą wartość dokładności ze wszystkich pomiarów dla róż (79.3%).

Dla czytelniejszej prezentacji wyników zadania 1c oraz zadania 1a., stworzyliśmy tabelę oraz wykres porównujące wyniki uzyskane dla trenowania części klasyfikującej oraz zastąpienia jej przez SVM:

Kernel\Data	Daisy	Dandelion	Rose	Sunflower	Tulip
linear	0.82	0.83	0.79	0.90	0.89
rbf	0.83	0.83	0.85	0.93	0.88
poly	0.84	0.84	0.86	0.90	0.90
Pre-trained Alexnet	0.86	0.78	0.84	0.91	0.78

.Tab. 02: Wyniki dla klasyfikatora SVM oraz wstępnie wytrenowanej sieci Alexnet.

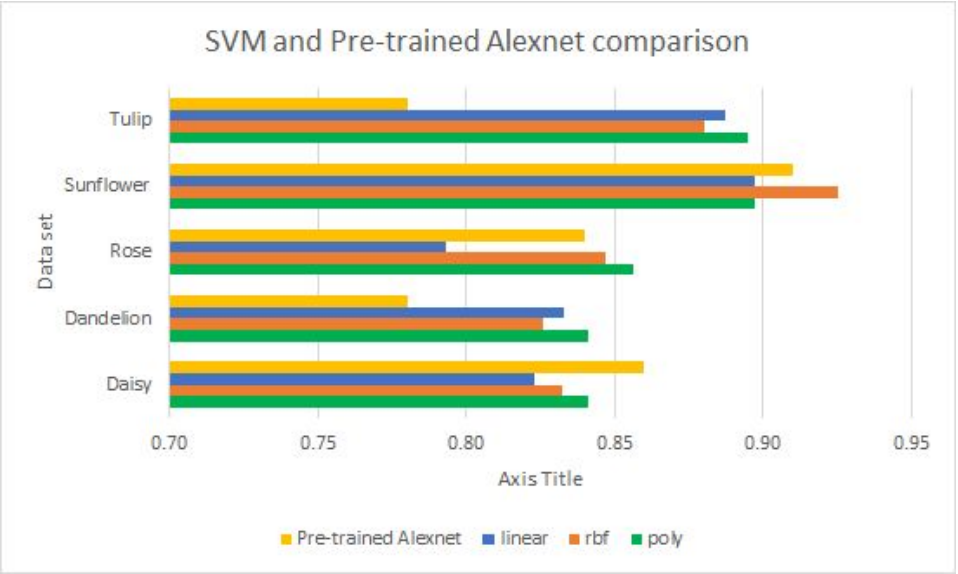


Fig. 02: Porównanie SVM oraz wstępnie wytrenowanej sieci Alexnet

Klasyfikator z zadania 1a uzyskał lepsze wyniki klasyfikowania stokrotek niż klasyfikator SVM(86%) W obu zadaniach uzyskaliśmy podobną skuteczność klasyfikowania słoneczników oraz róż, ale przy klasyfikacji tulipanów oraz mniszków lekarskich SVM wpłynął na znaczną poprawę wyników.

2. Uczenie sieci głębokiej

W zadaniu 2 skupiliśmy się na uczeniu głębokim sieci. Osiągnęliśmy to odblokowując odpowiednie warstwy modelu Alexnet wykorzystanego w zadaniu 1a. Najpierw odblokowaliśmy tylko ostatnią warstwę splotową (warstwa nr 10), następnie dwie ostatnie warstwy (7 oraz 10), wszystkie warstwy, a na koniec usunęliśmy z wytrenowanej sieci ostatnią warstwę splotową i ponowiliśmy uczenie. Strukturę sieci po naszych zmianach przedstawia zdjęcie poniżej:

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=5, bias=True)
  )
)
```

Wyniki trenowania sieci przedstawiliśmy w tabeli oraz na wykresach:

Uczenie \ Paramtery	dokładność (%	wartość funkcji strat (%
1 warstaw splotowa	87,5	6,2
2 warstwy spolotowe	87	6,2
cała sieć	82,4	10,4
sieć bez ostatniej warstwy splotowej	83,9	8,5

Tab. 03: Porównanie efektów uczenia sieci neuronowej dla różnych zakresów uczenia

Jak wynika z tabeli im większą część sieci neuronowej trenowaliśmy tym większą wartość miała funkcja strat, a niższą dokładność. Może to wynikać ze zbyt małej liczby obrazów jakie posłużyły do uczenia sieci oraz tego, że wstępnie wytrenowana sieć dostarczana przez bibliotekę PyTorch bardzo dobrze nadaje się do klasyfikacji naszych zdjęć kwiatów.

Usunięcie ostatniej warstwy splotowej z sieci nie wpłynęło na spadek dokładności. Po ponownym przeprowadzeniu procesu uczenia dokładność wzrosła.

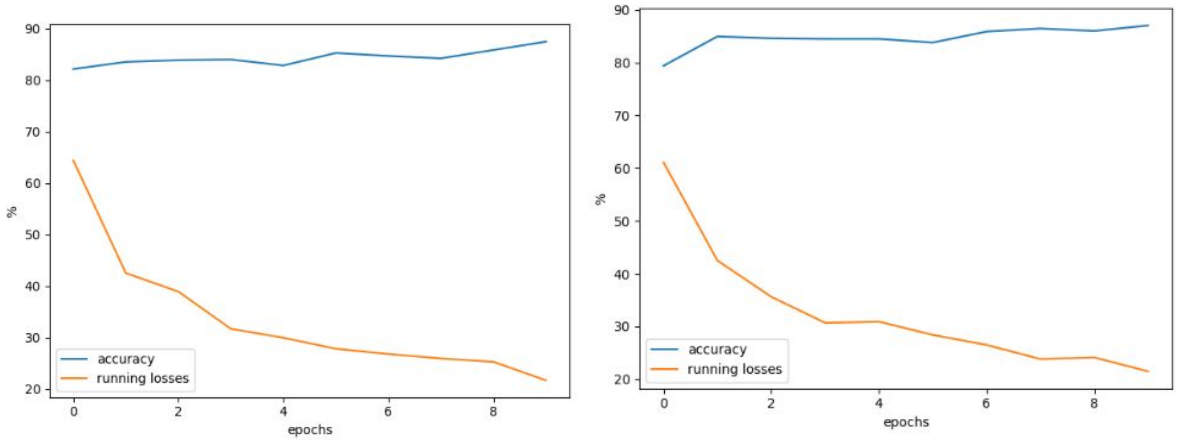


Fig. 03: Dokładność oraz wartość funkcji strat w procesie uczenia jednej warstwy splotowej (wykres po lewej) oraz dwóch warstw splotowych (wykres po prawej)

Jak widać już od pierwszego cyklu sieć ma bardzo wysoką dokładność, która z każdą kolejną nieznacznie wzrasta. Wartość funkcji strat natomiast bardzo szybko spada. W obu przypadkach sieć zachowuje się bardzo podobnie. Może to wskazywać na większy udział w klasyfikacji ostatniej warstwy splotowej.

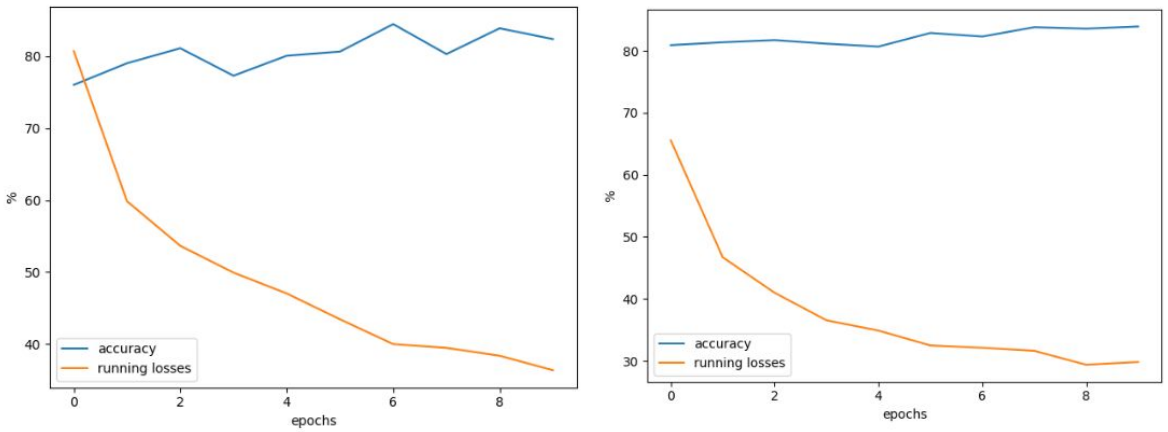


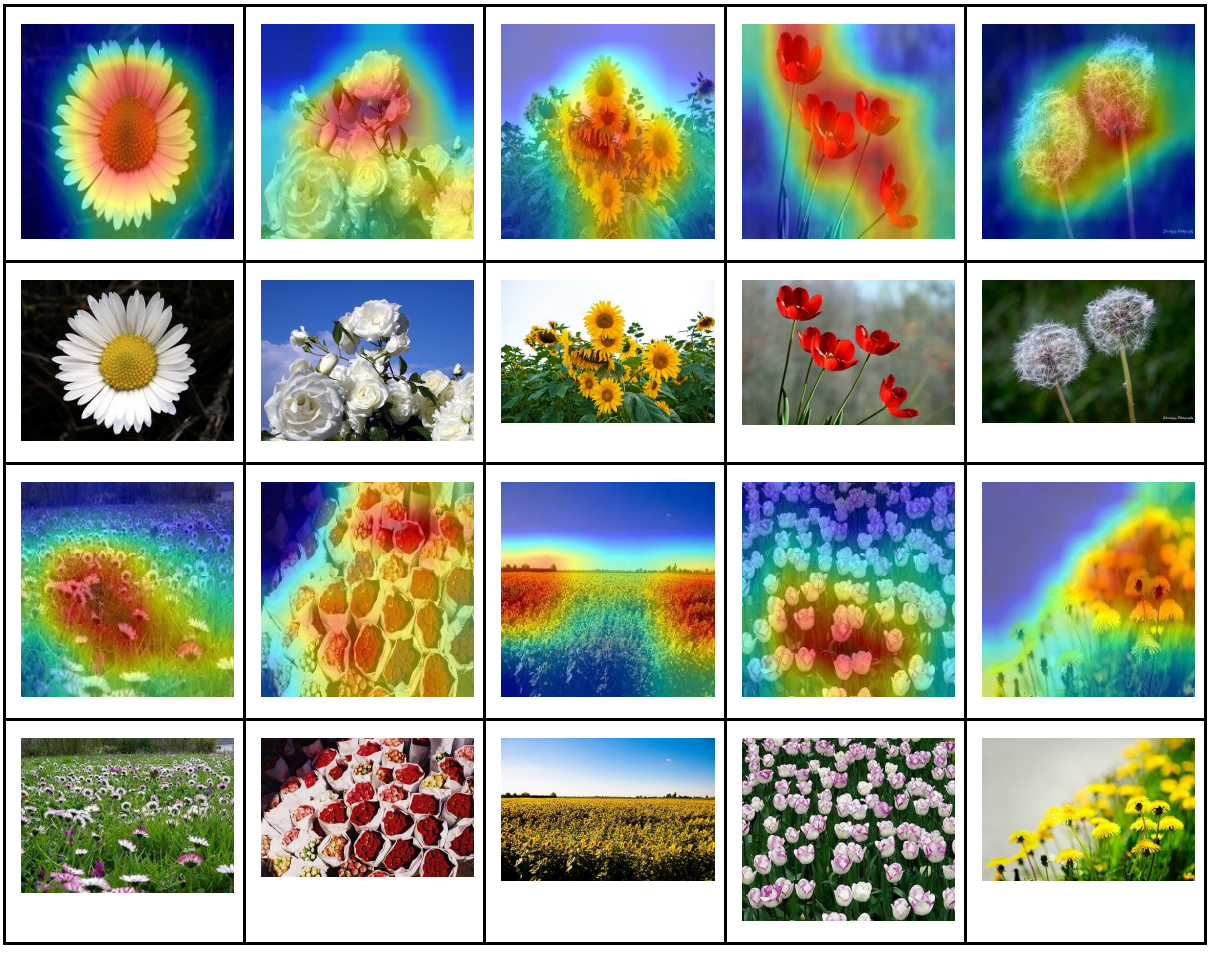
Fig. 04: Dokładność oraz wartość funkcji strat w procesie uczenia całej sieci (wykres po lewej) oraz tej samej sieci, ale z usuniętą ostatnią warstwą splotową(wykres po prawej)

Najbardziej spośród tych czterech wykresów wyróżnia się wykres przedstawiający proces uczenia się całej sieci. Podczas trenowania całej sieci otrzymaliśmy najniższą dokładność po pierwszej iteracji spośród badanych procesów uczenia. Dodatkowo podczas tego procesu uczenia początkowa wartość funkcji strat była znacznie wyższa niż w przypadku pozostałych procesów uczenia. Wydaje się to potwierdzać, że sieć była lepiej przygotowana do klasyfikacji na podstawie wcześniej wytrenowanych wag.

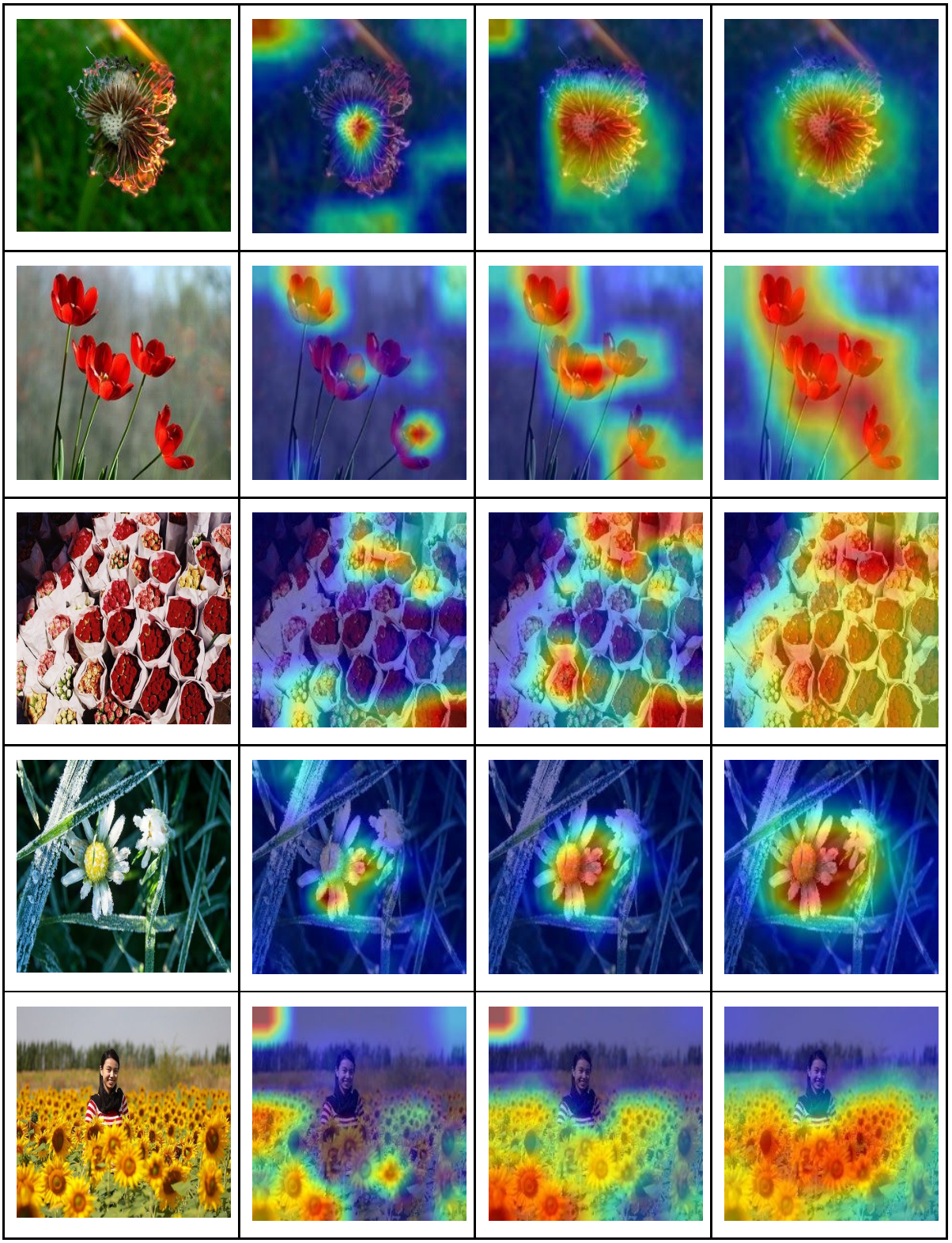
3.a. Wizualizacja obszarów uwagi sieci wytrenowanych z wykorzystaniem metod Class Activation Map (CAM).

Przed prezentacją wyników działania wizualizacji należy wyjaśnić czym jest metoda CAM (Class Activation Map).

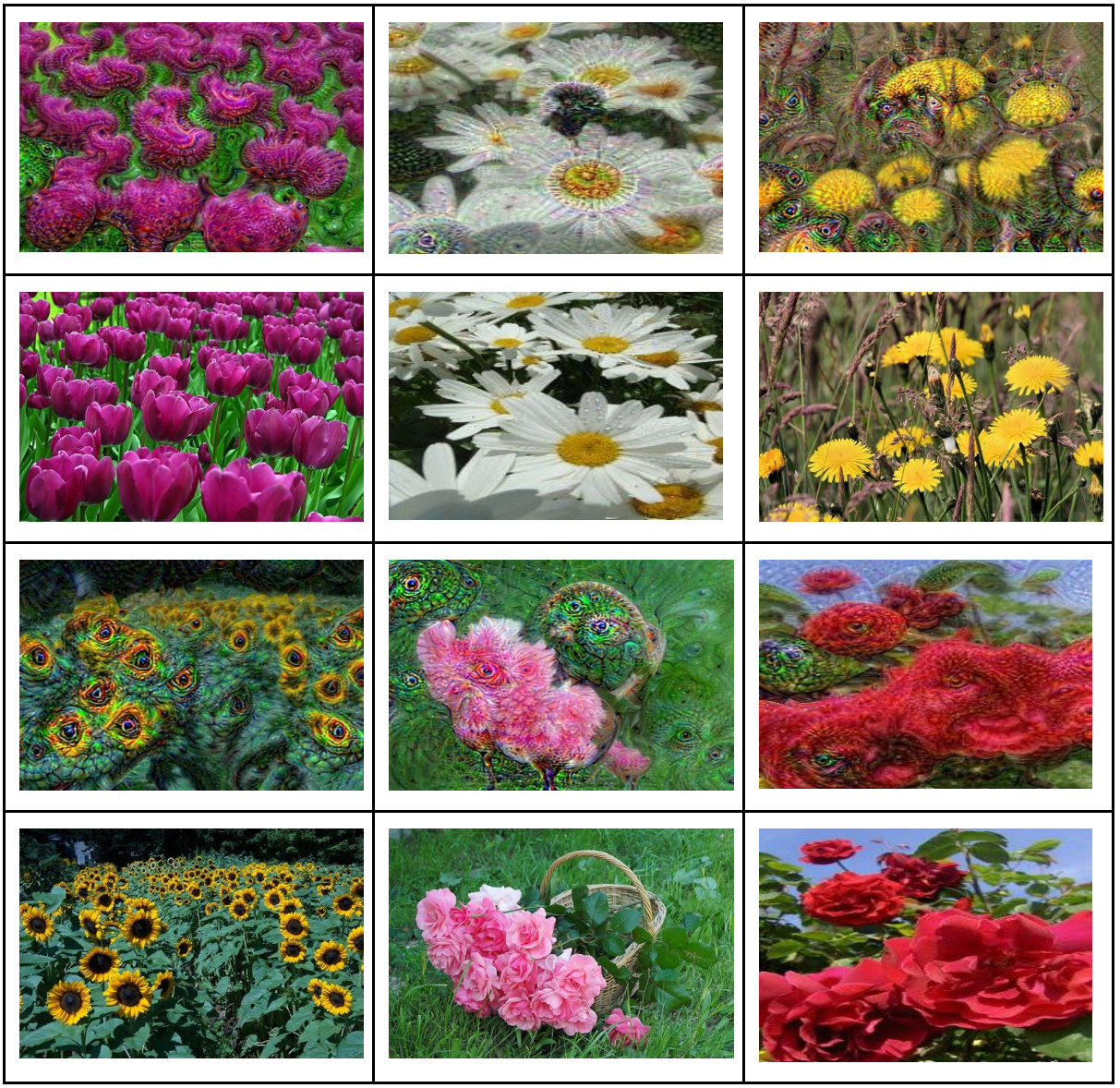
Model CNN składa się z wielu splotowych warstw i wykonuje globalne uśrednianie puli tuż przed końcową warstwą wyjściową. Aby uzyskać pożądany wynik, uzyskane funkcje są podawane do w pełni połączonej warstwy z aktywacją softmax. Dzięki rzutowaniu wag warstwy wyjściowej z powrotem na mapy splotowe pochodzące z ostatniej warstwy konwolucji można zidentyfikować ważność obszarów obrazu. Ta technika jest nazywana mapowaniem aktywacji klasy.



Oczywiście za pomocą CAM możemy wizualizować nie tylko jedną warstwę sieci, ale kilka, dzięki czemu możemy śledzić, jak ta sieć działa od wewnątrz. Poniżej znajduje się przykład serii kolejnych warstw:



3.b. Wizualizacja aktywacji wewnętrznych warstw sieci z wykorzystaniem techniki DeepDream.



Ocena narzędzi wykorzystanych w zadaniu:

W projekcie wykorzystaliśmy wiele bibliotek, ale tymi najważniejszymi był that Pytorch i Scikit-learn. Korzystaliśmy również z Google Colab, aby przyspieszyć proces uczenia się sieci.

Najważniejszym narzędziem wykorzystanym w projekcie jest biblioteka PyTorch. Pozwala ona w prosty sposób tworzyć, edytować już istniejące oraz trenować sztuczne sieci neuronowe. Zapewni również narzędzia, które można użyć do przetwarzania zbiorów danych. Dużym plusem jest również łatwy dostęp do krótkich kursów, które pozwalają zapoznać się z tym narzędziem. Pomocy w rozwiązaniu problemów można szukać również na forum, na którym na pytania programistów odpowiadają twórcy biblioteki oraz inni programiści.

Biblioteka scikit-learn posłużyła do realizacji zadania związanego z SVM. Jest to jedna z najpopularniejszych bibliotek języka Python wykorzystywana w data science oraz machine learning. Pozwala zrealizować większość zadań związanych z machine-learning.

Znacznym ułatwienie w pracach nad projektem było wykorzystanie Google Colab. Jest to platforma bardzo prosta w obsłudze (twórcy zapewnili również kilka tutoriali) oraz pozwala wykorzystać GPU do przyspieszenia procesu uczenia. Google Collab zawiera również narzędzia pozwalające na wizualizację przetwarzanych danych. Na podstawie własnych doświadczeń możemy polecić to narzędzie.

Github: https://github.com/APylkevych/SNR-Alexnet-flowers.git