Dawid Witek - Semestr 5. Złożone systemy cyfrowe

# Temat projektu

# Rozpoznawanie rasy psów przy użyciu sieci neuronowych konwolucyjnych.

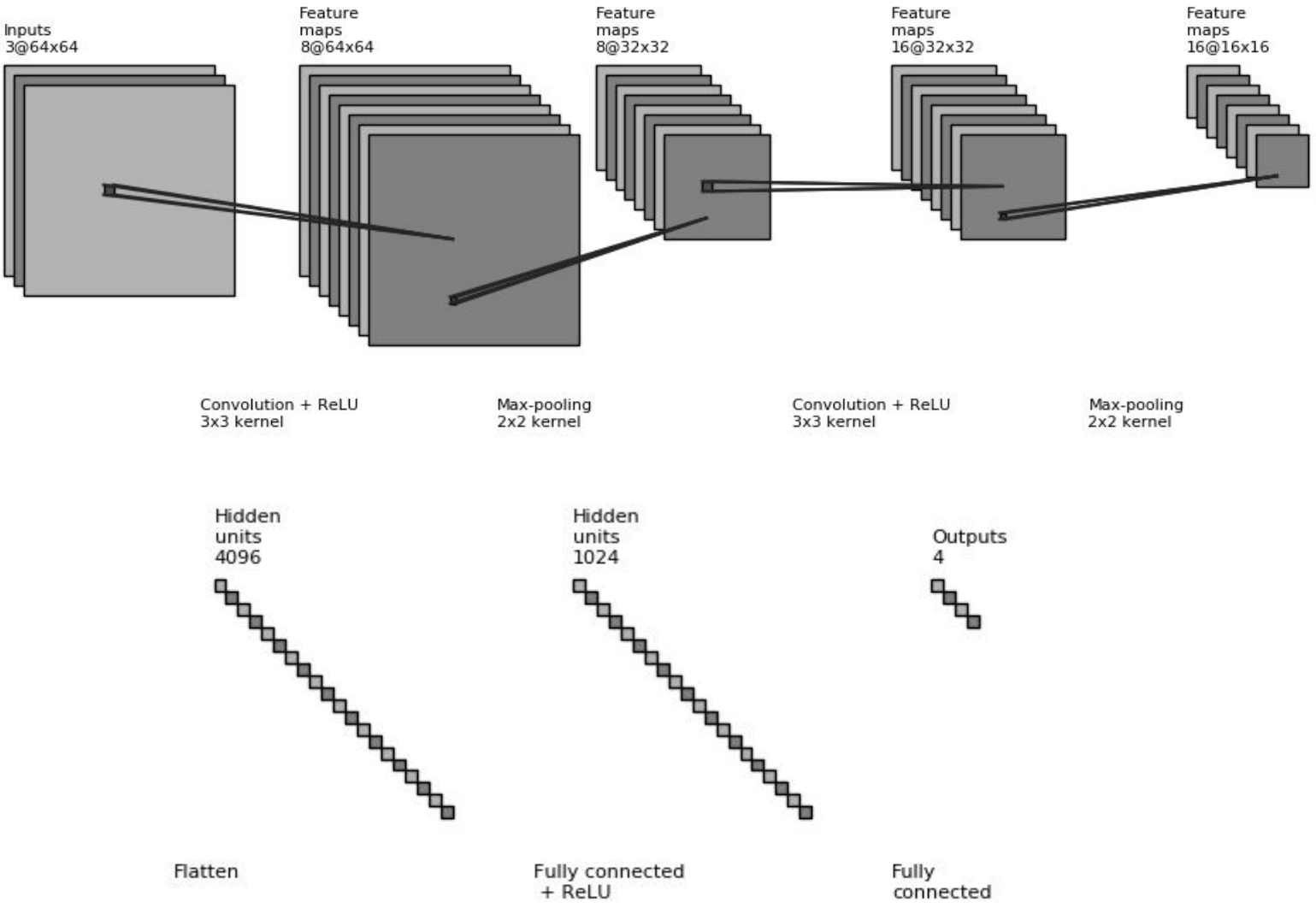
# Cel projektu

Zaprojektowanie i implementacja sieci neuronowej konwolucyjnej służącej do rozpoznawania rasy psów ze zdjęć, z wykorzystaniem Javy. Elementem niskopoziomowym w projekcie jest zaprojektowanie odpowiedniej architektury sieci neuronowej zdolnej do spełnienia wyżej wymienionego celu.

# Założenia

Pracę nad projektem rozpocznę od zrozumienia działania sieci neuronowych konwolucyjnych i zaplanowania architektury sieci. Kluczowym elementem w początkowej fazie projektu będzie również zdobycie odpowiednich zdjęć potrzebnych zarówno do wytrenowania sieci, jak również do sprawdzenia jej skuteczności w późniejszym czasie. Następnie zaimplementuję i sprawdzę poprawność działania zaproponowanej przeze mnie sieci neuronowej konwolucyjnej. Kolejnym krokiem będzie odpowiednia obróbka zdjęć oraz wytrenowanie sieci. Po uzyskaniu satysfakcjonujących wyników przez sieć parametry potrzebne do jej odtworzenia zostaną zapisane. Ostatnim elementem będzie przedstawienie skuteczności sieci w rozpoznowaniu rasy psów ze zdjęć zarówno tych na których sieć się uczyła, jak również osobnej partii nowych zdjęć.

# Schemat architektury sieci neuronowej konwolucyjnej.




Rysunek 1: Schemat planowanej architektury sieci neuronowej konwolucyjnej. Wykorzystano https://github.com/gwding/draw_convnet

# Postęp prac

## 22 październik - 5 listopad:

W trakcie tego okresu skupiłem się głównie na poznaniu i zrozumieniu zasady działania najpierw samej sieci neuronowej, a następnie konkretnie sieci neuronowej konwolucyjnej. W bibliografii zamieszczone są linki do stron z których korzystałem w celu zrozumienia tego problemu. Następnie zaplanowałem wstępnie jak bedzię wyglądać architektura mojej sieci i zacząłem jej implementacje od podstaw. W tym czasie napisałem algorytmy odpowiedzialne za: inicjalizację filtrów, wag i 'bias' odpowiednimi wartościami, konwolucję, ReLU, Max Pooling, warstwę Fully-Connected, Softmax, wyliczanie Cross Entropy Loss i L2 Regularization, jak również parę innych odpowiedzialnych za operacje na wektorach / macierzach lub odpowiednie skalowanie / zmienianie rozmiaru zmiennych. Niestety nie przetestowałem poprawności działania żadnego z nich. Zacząłem również powoli składać wyżej wymienione algorytmy w docelową sieć neuronową konwolucyjną.

W najbliższym czasie skupię się na jeszcze dokładniejszym zrozumieniu problemu oraz implementacji pozostałych algorytmów potrzebnych do prawidłowego działania sieci m.in. Backpropagation i Adam optimizer. Następnie przetestuję czy algorytmy działają poprawnie i złożę w całość wszystkie elementy architektury sieci. Kluczowym problemem będzie wybór tak zwanych 'hyperparameters', najprawdopodobiej wykorzystam wartości najbardziej popularne i polecane w literaturze.

## 5 listopad - 19 listopad:

Zrozumiałem i zaimplementowałem brakujące elementy sieci: backpropagation, adam optimizera (w wersji podstawowej, istnieją jego odmiany poprawiające jego jakość). Skleiłem całą sieć w całość, dodając przy okazji regularyzację przy pomocy dropout. Następnie ściągnałem przykładowe dane i zacząłem testować sieć. Po zaimplementowaniu odpowiedniego formatowania,labelowania i ładowania danych poprawiłem wszystkie błędy związane z implementacją oraz znalazłem zestaw hyperparametrów do testowania. Przy ich użyciu sprawdziłem działanie sieci dla bazowych problemów.

Sieć dochodzi praktycznie do 100% sprawności (co dowodzi jej poprawności) dla:

- jednego elementu
- dwóch elementów z różnych klas wynikowych
- 10 elementów, nawet pomimo przewagi jednej klasy wynikowej

Zauważyłem przy tym, że przy konfiguracji 1 duży batch, a 5 mniejszych, wydajniejszy jest 1 duży, ale to może być związane z małą ilością danych.

Obecnie czekam na dostęp do zdjęć ze strony http://www.image-net.org/. Planuję pobrać po około 1000 zdjęć dla każdej z czterach klas wynikowych i spróbować nauczyć sieć je rozpoznawać. Wydaje się, że w skończonym czasie sieć powinna poradzić sobie z tym zadaniem. W przypadku większych problemów albo nadzwyczajnego sukcesu, ilość danych i klas wynikowych może się zmienić.

## 19 listopad - 10 grudzień:

Dodałem zapisywanie i odczytywanie parametrów potrzebnych do odtworzenia sieci z pliku, jak również runtime shutdown odpowiedzialny za to, żeby zapis aktualnych parametrów zawsze wykonał się przed zamknięciem programu. Ponadto dodałem wielowątkowość do programu, od teraz każde zdjęcie jest analizowane na osobnym wątku co przyspiesza proces nauki. W samej sieci neuronowej zmieniłem zachowanie się learning rate'a, wraz z upływem czasu trenowania jest on odpowiednio skalowalny, aby sieć się nie przetrenowała.

W tym okresie powstały już pierwszy próby wizualizacji sieci neuronowej, odpowiednich filtrów i wag w celu wizualnego sprawdzenia poprawności implementacji.

## 10 grudzień - 7 styczeń:

Poprawiłem drobne błędy zauważone na zajęciach, które powodowały błędne działanie sieci. Implementacja sieci neuronowej została zakończona, sama sieć wytrenowana, wyniki jej działania zaprezentowane są poniżej. Ostatnie rzeczy, które zostały zaimplementowane były związane bardziej ze zbadaniem skuteczności sieci, zarówno dla znanych już zdjęć, jak i nowych.

# Skuteczność działania sieci neuronowej konwolucyjnej

Skuteczność działania sieci neuronowej konwolucyjnej testowałem w dwóch różnych wariantach, dla dwóch różnych danych. Wariant pierwszy polegał na prezentacji dokładnych wyników zaproponowanych przez sieć, podczas gdy drugi sprawdzał, dla jakiego procenta zdjęć, dobra kategoria zdjęcia jest proponowana przez sieć z największym prawdopodobieństwem. Pierwsze dane są to 4096 zdjęć na których sieć była trenowana, drugie dane są to nowe 368 zdjęć użytych do walidacji działania sieci.

Wyniki prezentują się następująco:

- dla danych, na których sieć była uczona

```
Training set probability: 67.32340797274495%
Golden probability: 67.41255374729384%
Poodle probability: 65.170157712520 42%
Doberman probability: 69.0066815421606%
Husky probability: 68.02772692355188%

Training set 0/1 probability: 83.984375%
Golden probability: 83.1848552338 53%
Poodle probability: 82.827406764960 97%
Doberman probability: 84.84231943031536%
Husky probability: 85.12241054613936%
```

Rysunek 2: Skuteczność sieci dla danych, na których była uczona.

- dla nowych danych

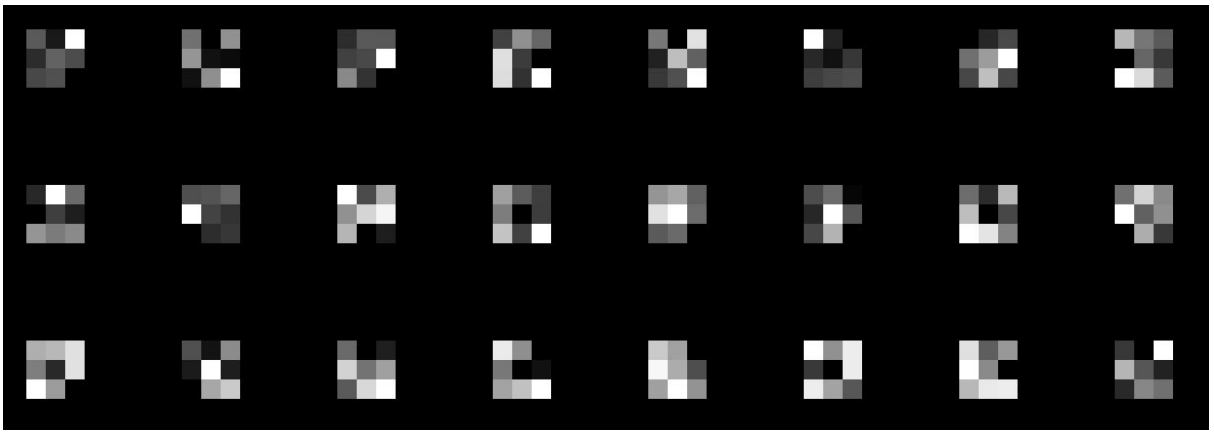
```
Validation set probability: 42.073034965036115%
Golden probability: 39.931007961841274%
Poodle probability: 32.744560095280 79%
Doberman probability: 41.03160877616803%
Husky probability: 54.37304536052656%

Validation set 0/1 probability: 47.55434782608695%
Golden probability: 40.659340659340 66%
Poodle probability: 39.784946236559136%
Doberman probability: 47.777777777777 78%
Husky probability: 61.702127659574465%
```

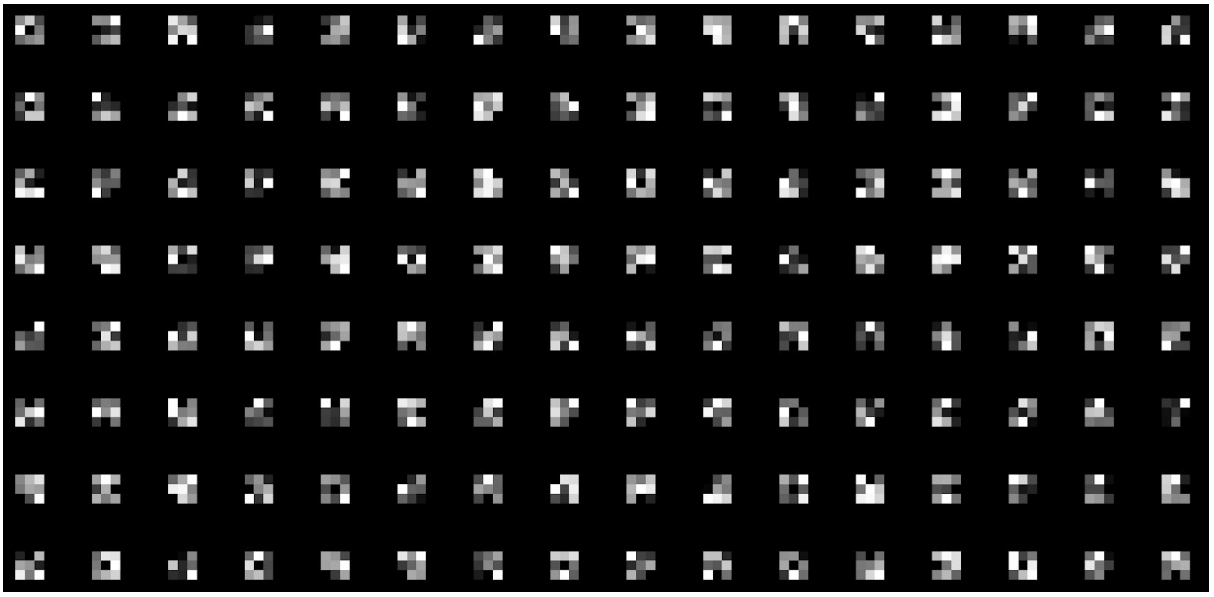
Rysunek 3: Skuteczność sieci dla nowych danych.

# Wizualizacja sieci neuronowej i jej działania

W sieci wykorzystuję dwa zestawy filtrów, ich wizualizacja może pokazać czy filtry mają odpowiednie wartości, czy nie są “martwe”. Przeskalowane wartości z filtrów wyglądają następująco:



Rysunek 4: Zestaw filtrów dla pierwszej warstwy konwolucyjnej.



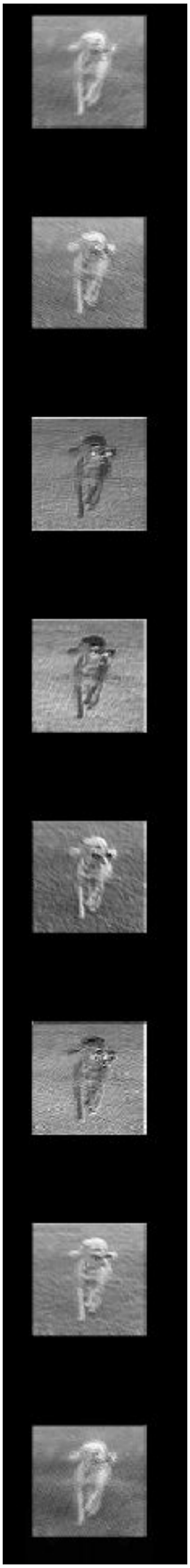
Rysunek 5: Zestaw filtrów dla drugiej warstwy konwolucyjnej.

Patrząc na filtry można zauważyć, że żaden z nich nie jest całkowicie czarny, co oznaczałoby, że jest “martwy”. Wizualizacja zestawów wag użytych w warstawach fully connected jest bezsensowna, ze względu na rodzaj tablicy w jakiej te wagi są przechowywane ( bardzo duża wysokość, mała szerokość ).

Przykład początkowych działań sieci:

- zdjęcie początkowe

- warstwa konwolucji



- warstwa ReLU

- warstwa Max Pooling

Ze względu na czytelność dalsze wizualizacje nie mają sensu w sprawozdaniu, jednakże wszystkie znajdują się w folderze z projektem.

# Podsumowanie

Skuteczność działania sieci nie jest najwyższa, jednakże biorąc pod uwagę wszystkie czynniki uważam, że jest ona wystarczająca. Problemem okazało się wykorzystanie w celu implementacji sieci neuronowej konwolucyjnej języka programowania Java, jak również moc obliczeniowa sprzętu wykorzystanego do trenowania sieci. Przy większej ilości czasu i mocy obliczeniowej sieć doszłaby do bardzo dobrej sprawności, czemu dowodzą testy dla mniejszej ilości danych. Zaproponowana architektura sieci jest poprawna.

W folderze z projektem w pliku “opis.txt” opisane są wszystkie foldery i pliki, jak również kluczowe klasy programu.

# Bibliografia


Część teoretyczna:

- https://en.wikipedia.org/wiki/Convolutional_neural_network
- https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
- https://www.youtube.com/watch?v=JB8T_zN7ZC0
- https://www.youtube.com/user/BrandonRohrer/videos
- https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi
- http://cs231n.github.io/
- https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv
- https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
- http://cs231n.github.io/understanding-cnn/

Część praktyczna:

- https://www.youtube.com/watch?v=45MbmHQ5iMY
- https://gluon.mxnet.io/chapter04_convolutional-neural-networks/cnn-scratch.html
- https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1
- http://cs231n.github.io/neural-networks-case-study/