

MPUM - Projekt

Jędrzej Hodor



Spis treści

1 Wstęp	3
1.1 Charakterystyka danych - Animals3	3
1.2 Charakterystyka danych - Fruits	4
2 Omówienie teoretyczne	4
2.1 Konwolucyjne sieci neuronowe	4
2.1.1 Warstwa konwolucji	5
2.1.2 Warstwa Poolingu	6
2.1.3 Warstwa całkowicie połączona	6
2.1.4 Warstwa aktywacji	6
2.1.5 Funkcja straty	6
2.2 Tensorflow	6
2.3 Augmentacja	7
2.4 Dropout	9
2.5 Głosowania	9
2.6 Metoda k najbliższych sąsiadów	9
3 Opis modeli	9
3.1 myCNN	9
3.1.1 Flat, lr= 0.1	9
3.1.2 Flat, lr= 0.01	9
3.1.3 Simple	9
3.1.4 Complex	9
3.2 tfCNN	10
3.2.1 Pure	10
3.2.2 aug + drop	10
3.3 Głosowania	10
3.4 kNN	10
4 Omówienie empiryczne	10
4.1 myCNN - lr	10
4.2 Dodanie konwolucji	12
4.3 Tensorflow	14
4.4 CNN - podsumowanie	14
4.5 Głosowania	17
4.6 kNN	19
5 Obserwacje i Ciekawostki	20
5.1 Pomyłki	20
5.2 Najpewniejsze	21
6 Podsumowanie	22
6.1 Wyniki	22
6.2 Omówienie wyników	22
7 Źródła	22

1 Wstęp

Projekt skupia się na problemie rozpoznawania obrazów. Używam kilku algorytmów do klasyfikacji obrazów z dwóch zbiorów danych:

- [Animals3](#)
- [Fruits](#)

Wszystko najpierw zostało przestestowane na zbiorze Animals3, następnie niektóre testy były powtórzone na Fruits.

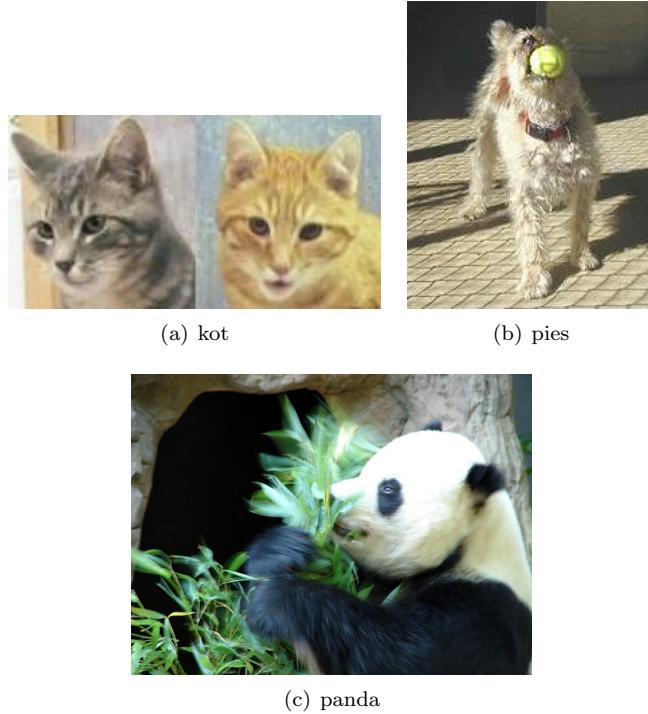
1.1 Charakterystyka danych - Animals3

Mamy obrazy różnych rozmiarów trzech gatunków zwierząt w różnych sytuacjach. Na obrazach widać różne części zwierząt, więc ich sklasyfikowanie nie powinno być trywialnym zadaniem. Jest to dość mały zbiór danych, ale z powodów wydajnościowo-czasowych postanowiłem skupić się właśnie na klasyfikacji małych zbiorów danych. Podzieliłem dane na trzy grupy:

- treningowe: $3 \cdot 700 = 2100$,
- walidacyjne: $3 \cdot 100 = 300$,
- testowe: $3 \cdot 200 = 600$.

Pobieżnie przeglądając dane można wysnuć hipotezę, że pandy będzie najłatwiej rozpoznać ze względu na ich charakterystyczne umaszczenie. W dalszej części analizy okaże się być to celnym spostrzeżeniem.

Przed użyciem modeli sieci neuronowych, które napisalem sam bez bibliotek zmniejszam wszystkie obrazki do rozmiaru 28×28 , aby nie trwało to wieki. Sieć korzystająca z tensorflow przetwarza obrazki rozmiaru 150×150 .



Rysunek 1: Przykłady obrazków ze zbioru Animals3

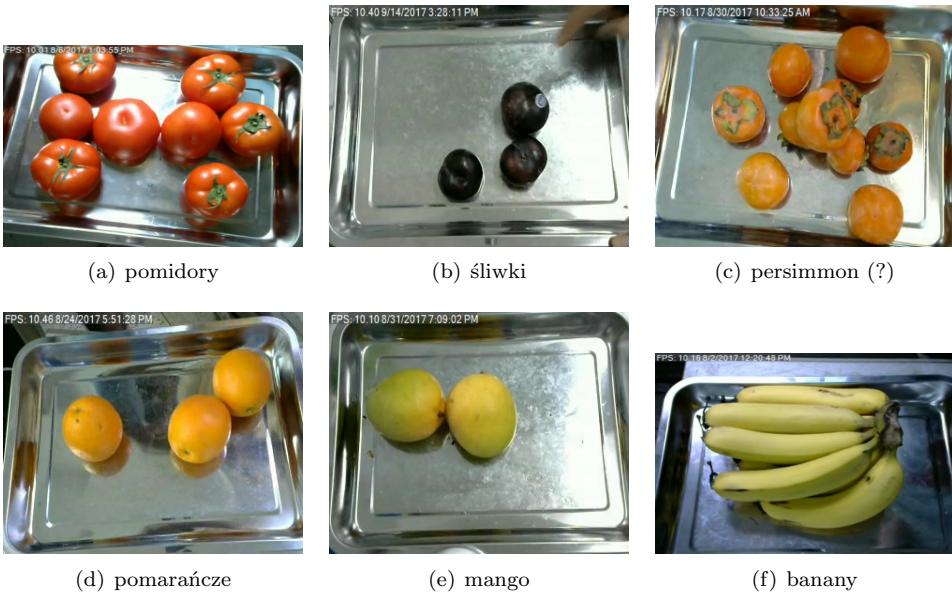
1.2 Charakterystyka danych - Fruits

Jest to ogromny zbiór zdjęć owoców na tacy. Jednak na projekt mamy tylko pół semestru, więc wybrałem kilka kategorii i pewne liczby przedstawicieli. Obrazki są dużo bardziej "przygotowane" pod uczenie maszynowe. Mamy dużo ujęć z różnych stron z różnym oświetleniem na dane owoce, ale jednak w tym samym settingu.

- treningowe: $6 \cdot 700 = 4200$,
- walidacyjne: $6 \cdot 100 = 600$,
- testowe: $6 \cdot 200 = 1200$.

Warto zwrócić uwagę na to, że wizualnie patrząc to co najbardziej wyróżnia dane to kolor owoców, więc można przypuszczać, że liniowe modele poradzą sobie lepiej niż przy pierwszym zbiorze danych.

Przed użyciem modeli sieci neuronowych, które napisałem sam bez bibliotek zmniejszam wszystkie obrazki do rozmiaru 28×28 , aby nie trwało to wieki. Sieć korzystająca z tensorflow przetwarza obrazki rozmiaru 150×150 .



Rysunek 2: Przykłady obrazków ze zbioru Fruits

2 Omówienie teoretyczne

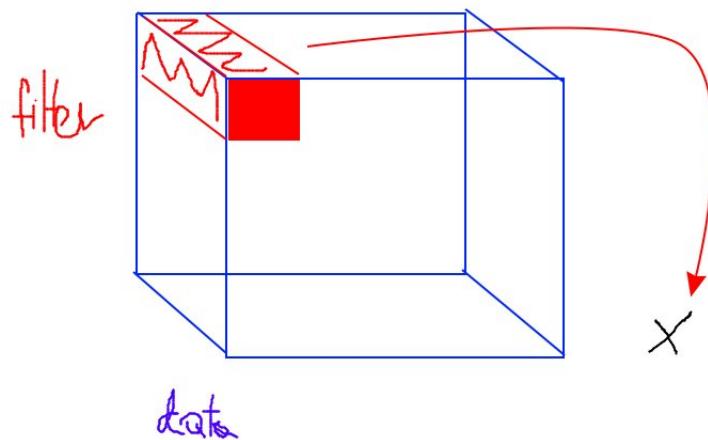
2.1 Konwolucyjne sieci neuronowe

Konwolucyjne sieci neuronowe tak jak zwykłe sieci neuronowe składają się z warstw. Jednak ze względu na szczegółowość danych jakimi są obrazy te warstwy wyglądają zwykle inaczej niż normalnie. Po pierwsze zastanówmy się dlaczego zwykła sieć neuronowa może nie być najlepszym narzędziem do przetwarzania obrazów. Mając obraz, cechami, które najprościej wyekstrahować są oczywiście piksele. Tutaj pojawia się pierwsza przeszkoła. Pikseli zwykłe jest bardzo dużo, zwykle każdy ma 3 wymiary (rgb). Co już dla małego obrazka 150×150 daje 67500 cech co przy głębszej sieci może być czasowo zabójcze. Kolejny powód, czemu to nie jest najlepszy pomysł to taki, że pojedynczy piksel tak naprawdę nie przekazuje zbyt wiele informacji o obrazku. Jest on zbyt mały jego częścią. Powiedzmy, że mamy kota i psa, oba w tym samym pokoju i w tym samym kolorze niech zajmują $\frac{1}{3}$ zdjęcia. Wtedy pozostała część (większość) to będzie podłoga, która będzie podobna i może zaważyć na tym, że sieć uzna obrazki za dużo bardziej podobne niż na przykład dwa zdjęcia kota w różnych środowiskach. Dlatego w konwolucyjnych

sieciach chcemy łączyć piksele w większe grupy i sprawiać, aby ”łącznie” dawały jakieś ogólniejsze cechy. Głównym filarem takich modeli jest warstwa konwolucji.

2.1.1 Warstwa konwolucji

Mamy trójwymiarowe wejście, na początku będzie to obrazek (plus rgb, stąd trzeci wymiar), później już przetworzony obrazek przez początkowe warstwy sieci. Powiedzmy, że jest to $h \times w \times d$. Ustalamy parametry F i N , gdzie F to rozmiar filtra, a N to ich liczba. Filtr jest pewną macierzą parametrów o odpowiednim rozmiarze. Zwykle będzie to: $F \times F \times d$. Czyli innymi słowy jest to taka kostka, którą przykładamy do macierzy danych. Każdy jeden filtr generuje jeden ”plaster” danych wyjściowych. Czyli jeśli N będzie duże to nasze dane zrobią się głębsze.



Rysunek 3: Działanie jednego filtru

Omówmy to dokładniej. Powiedzmy, że mamy filtr \mathcal{F} , który wpasuje się jak przyłożymy go w miejscu s (jak na Rysunku 3). Wtedy niech element filtru v odpowiada $f(v)$ w macierzy danych. Mamy:

$$x = \sum_{v \in \mathcal{F}} v \cdot f(v)$$

Jak z tego tworzymy wyjście? Przykładamy filtr w kolejnych wartościach s idąc od lewej do prawej, potem do rzędu niżej i tak dalej. Formalnie jeśli D to dane, a X to wyjście to mamy:

$$X[i, j, k] = \mathcal{F}_k(s := (i, j, 1)).$$

Gdzie \mathcal{F}_k to k -ty filtr (spośród N). Można stosować filtry z ”przerwą” tzn:

$$X[i, j, k] = \mathcal{F}_k(s := (2i, 2j, 1)).$$

Widzimy, że faktycznie każdy plaster X odpowiada za działanie danego filtru. Tak jak pisałem wyżej, każdy filtr jest zbiorem parametrów, które optymalizujemy w trakcie działania sieci. Domyslnie chcemy, aby filtry wyciągnęły z danych bardziej ogólne cechy. Zwykle używa się małych filtrów tzn. $F \in [2, 10]$.

2.1.2 Warstwa Poolingu

Kolejną ważną częścią architektur sieci neuronowych jest tzw. warstwa poolingu. Chodzi o to, aby zmniejszyć aktualnie przetwarzane dane i równocześnie można w ten sposób przeciwdziałać przeuczeniu. Gdybyśmy cały czas operowali na "wielkich" danych to byłoby to bardzo łatwe. Na każdym "plastrze" danych przykładamy taki jakby filtr, który bierze po prostu maksimum i przenosi dalej. Parametrem w tej warstwie jest wielkość kwadratu, z którego chcemy brać maksimum P . Jeśli $P = 2$ to:

$$h \times w \times d \mapsto \frac{h}{2} \times \frac{w}{2} \times d.$$

Tak jak przy zwykłych filtrach można używać nakładających się obszarów, z których bierzemy maksimum lub rozłącznych, wedle uznania.

Zauważmy, że ta warstwa nie ma żadnych parametrów, jest czysto mechaniczna.

2.1.3 Warstwa całkowicie połączona

Jest to element, który występuje w zwykłych sieciach. Polega po prostu na przemnożeniu skalarnie przez macierz parametrów tych samych rozmiarów co dane (chodzi o przemnożenie odpowiednio punktowo, a następnie dodanie do siebie wartości). Oczywiście to co napisałem samo w sobie byłoby bez sensu. Dokładniej mówiąc robimy to dla każdej cechy otrzymując wektor cech, na którym leżą pewne wagi. Najlepiej by było, aby po fazie treningu największa waga leżała na klasie, w której faktycznie dany obraz jest.

Warto mieć te cechy odpowiednio przeskalowane. Dlatego korzystamy z funkcji **softmax**, która wyraża się wzorem:

$$w(k) = \frac{e^{z_k}}{\sum_j e^{z_j}}.$$

Gdzie z_j to wyjścia odpowiadające odpowiednim cechom.

2.1.4 Warstwa aktywacji

Do aktywacji używamy klasycznej funkcji ReLu, czyli:

$$\max(x, 0).$$

2.1.5 Funkcja straty

Funkcją straty, której używam wszędzie jest **categorical crossentropy**, czyli:

$$\sum_i y_i \log(w).$$

Gdzie w to output z softmaxu, a y_i to wektor z samymi zerami i jedną jedynką w odpowiednim miejscu.

2.2 Tensorflow

Skonstruowałem też sieć przy pomocy biblioteki tensorflow. W ten sposób można dużo szybciej i efektywniej wykonać obliczenia. Ilość czasu zmniejsza się radykalnie.

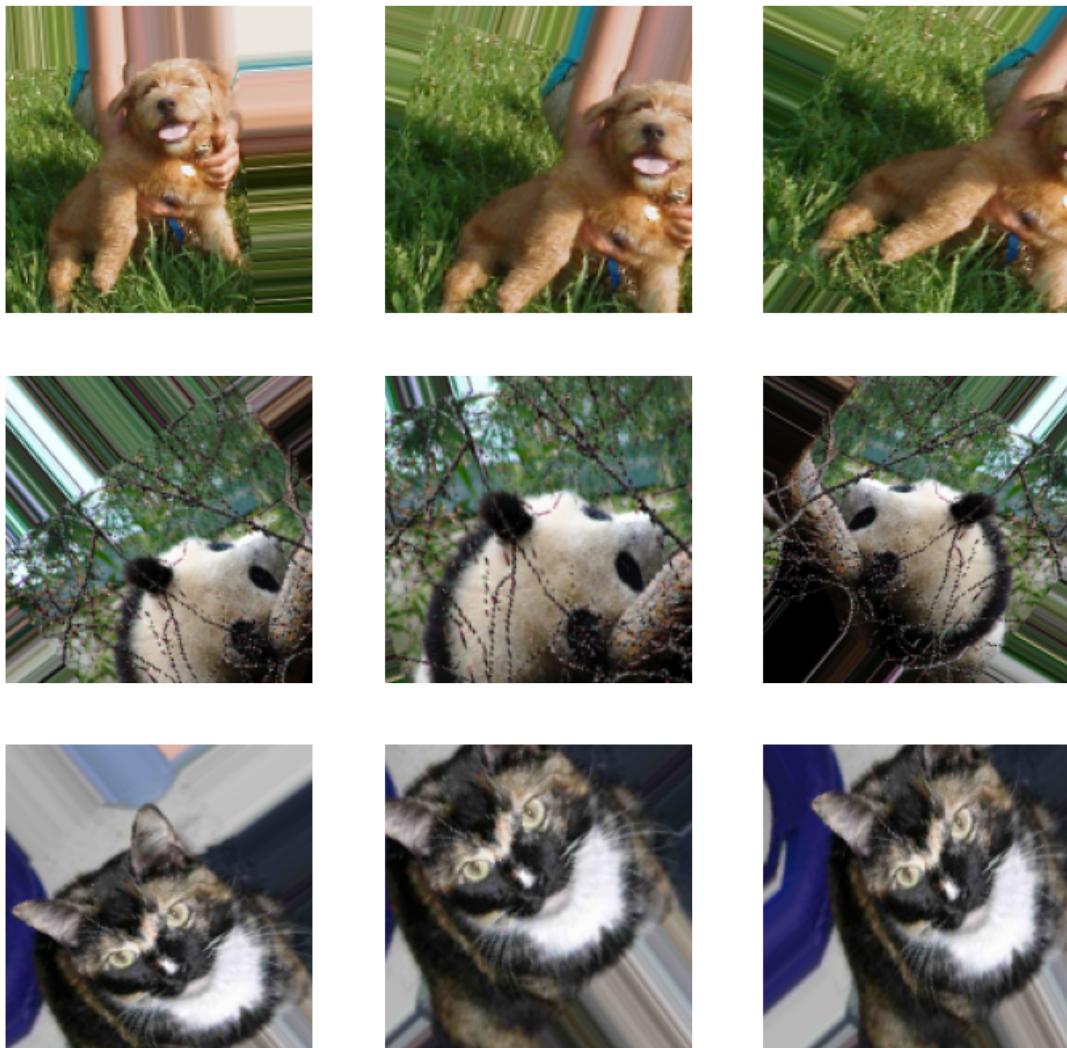
Jedyną teoretyczną różnicą jest użyty optymalizator. W moim modelu używałem zwykłego gradientu. Tutaj używam optymalizatora **Adam**. Jest to nowoczesny algorytm optymalizacyjny, którego główną zaletą jest elastyczne dostosowywanie learning rate do parametrów. Użyłem go ponieważ jest aktualnie najlepszym empirycznie optymalizatorem.

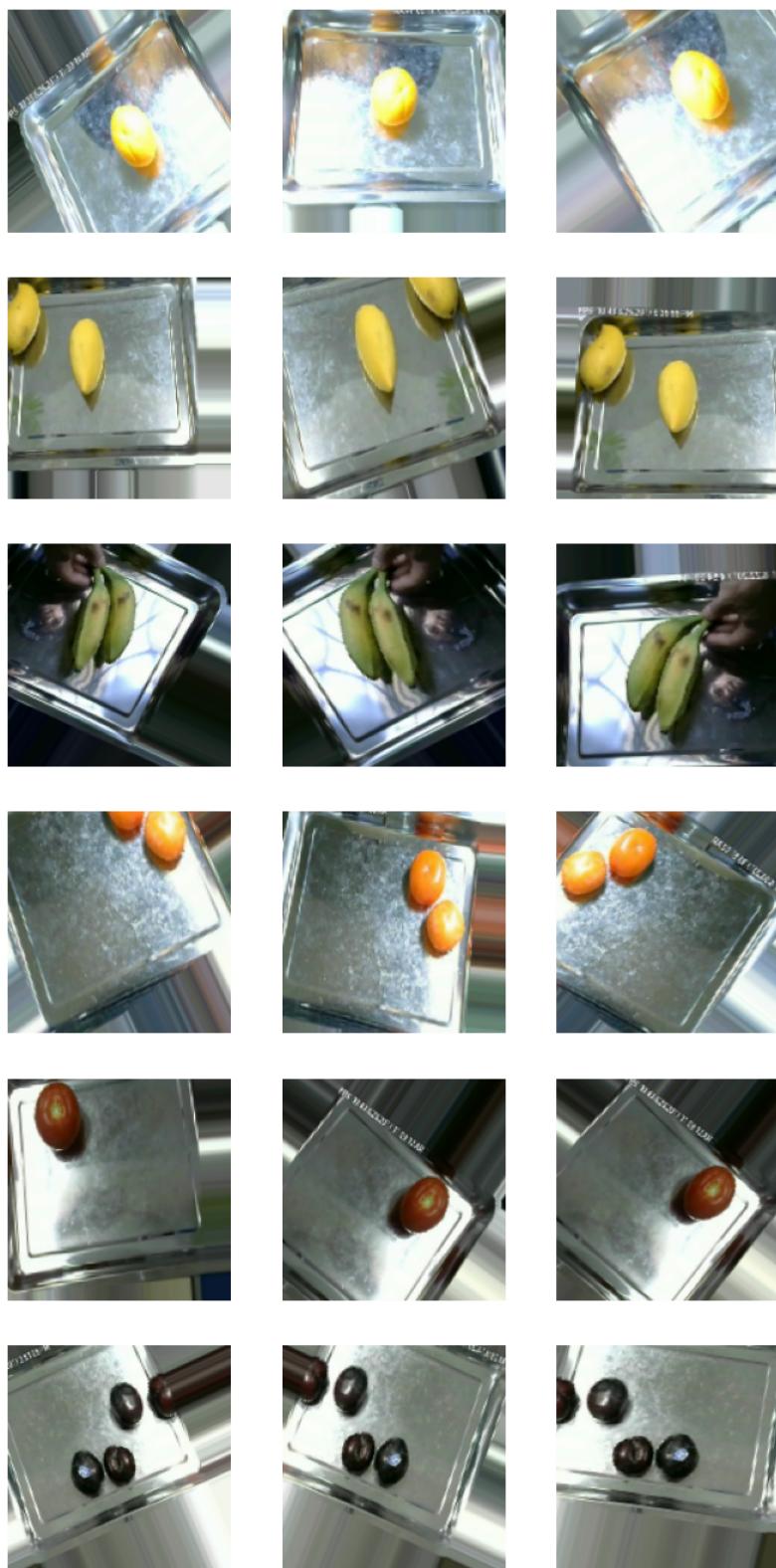
2.3 Augmentacja

Gdy operujemy na małych danych jesteśmy bardzo narażeni na przeuczenie. Na szczęście biblioteki pythona oferują bardzo ciekawą opcję jaką jest augmentacja danych. Jest to wydłużanie, rozmywanie, wyginanie, obracanie itd. danych. Innymi słowy, jest trochę tak, jakbyśmy rozmnażali dane. Sprawia to, że dużo trudniej o przeuczenie.

W szczegółach sieć w trakcie treningu nigdy nie widzi dwa razy tego samego obrazka, gdyż za każdym razem działa augmentacja.

Poniżej zamieszczam przykłady jak działa ten proces.





2.4 Dropout

Jest to mała modyfikacja do warstwy całkowicie połączonej. Polega na tym, że mamy parametr p , zwykle $p = \frac{1}{2}$. I dla każdego pola w warstwie całkowicie połączonej losujemy (z prawdopodobieństwem p), czy będziemy brać pod uwagę to pole przy ostatecznej sumie, czy nie. Czyli jak sama nazwa wskazuje dropout. Jak każda dodana losowość przeciwdziała to przeuczeniu.

2.5 Głosowania

Wiadomo, że co dwie głowy to nie jedna, więc może też co dwie sieci to nie jedna? Uznałem, że wypróbuję metodę "głosowania", to znaczy, kilka różnych sieci będzie z odpowiednimi wagami głosować, do której klasy należy obiekt. Nie binarnie, ale swoją pewnością. To znaczy, dodaję do siebie przeskalowane wektory wyjścia z architektury sieci, przemnażam przez odpowiednią wagę i patrzę jaki jest argmax.

2.6 Metoda k najbliższych sąsiadów

Była omawiana na wykładzie, więc pozwolę sobie pominąć tutaj opis teoretyczny.

3 Opis modeli

3.1 myCNN

Tak nazwałem modele oparte na mojej własnej implementacji konwolucyjnej sieci neuronowej. Wytrenowałem cztery rodzaje takich modeli, tu omówię ich architektury i parametry. We wszystkich użyłem obrazków przeskalowanych do rozmiarów 28×28 ze względów wydajnościowych.

3.1.1 Flat, lr= 0.1

Jest to sieć neuronowa z jedną warstwą w pełni połączoną, więc tak naprawdę jest to algorytm liniowy.

$$\rightarrow \text{FC} \rightarrow$$

Learning Rate jest równy 0.1 co jest dość dużą wartością (patrz porównanie lr w następnej sekcji). Trenowana przez 100 epok.

3.1.2 Flat, lr= 0.01

Model różni się od poprzedniego tylko learning rate

3.1.3 Simple

Prosta nietrywialna architektura:

$$\rightarrow \text{Conv}(N16, F3) \rightarrow \text{Pool}(2) \rightarrow \text{FC} \rightarrow$$

Również 100 epok, learning rate pozostawiłem "lepszy", czyli 0.01.

3.1.4 Complex

Dużo bardziej skomplikowana architektura, która przy tych danych i implementacji nie była zbyt wydajna:

$$\rightarrow \text{Conv}(N16, F3) \rightarrow \text{Pool}(2) \rightarrow \text{Conv}(N16, F2) \rightarrow \text{Pool}(3) \rightarrow \text{Conv}(N16, F2) \rightarrow \text{FC} \rightarrow$$

Tutaj, aby uzyskać sensowne wyniki pozwoliłem na 250 epok. Tę sieć wytrenowałem tylko na jednym zbiorze.

3.2 tfCNN

Sieci korzystające z tensorflow. Dzięki wydajności biblioteki obrazki są rozmiaru 150×150 , stosujemy zdecydowanie więcej filtrów. (Liczba parametrów to: 9497126.) Obie wersje trenujemy 30 epok.

3.2.1 Pure

Tak jak wspominałem wcześniej użyłem optymalizera Adam i cross-entropy loss. Architektura jest następująca:

$$\rightarrow \text{Conv}(N16, F3) \rightarrow \text{Pool}(2) \rightarrow \text{Conv}(N32, F3) \rightarrow \text{Pool}(2) \rightarrow \text{Conv}(N64, F3) \rightarrow \text{Pool}(2) \rightarrow \text{FC} \rightarrow$$

3.2.2 aug + drop

Jedyne zmiany to augmentacja danych i dropout przed ostatnią pełną warstwą.

3.3 Głosowania

Tę technikę testowałem tylko na zbiorze Animals3. Dokładniejsze informacje w następnej sekcji.

3.4 kNN

Testuję dla parametru $k \in [1, 50]$ w dwóch wariantach wielkości obrazków. Albo 30×30 , albo 60×60 .

4 Omówienie empiryczne

4.1 myCNN - lr

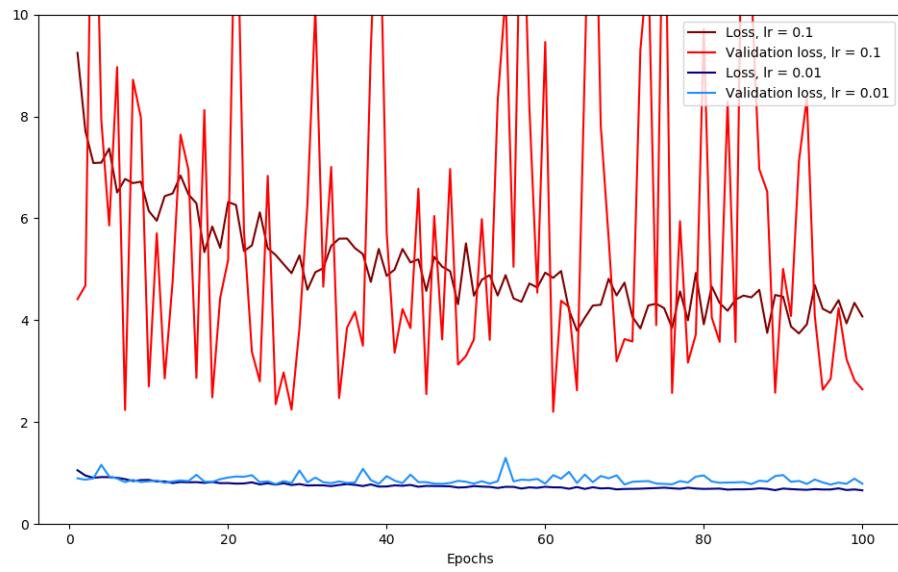
Porównam modele myCNN Flat z różnym learning rate. Można się spodziewać, że, gdy learning rate jest za duży, to funkcja celu, jak i również dokładność na zbiorze walidacyjnym będzie szaleć. Dokładnie to można zobaczyć na poniższych wykresach (Rysunek 4). W obu przypadkach niebieskie linie są zdecydowanie bardziej płaskie i doznają mniej fluktuacji. Widać zdecydowaną różnicę między zbiorami danych. Prawdopodobnie wynika to z dużo większej "liniowości" owoców.

Oczywiście to zachowanie ma matematyczne uzasadnienie. Jeśli za każdym razem odejmujemy większą część obliczonego gradientu, to większa szansa, że przeskoczymy jakiś punkt minimalny.

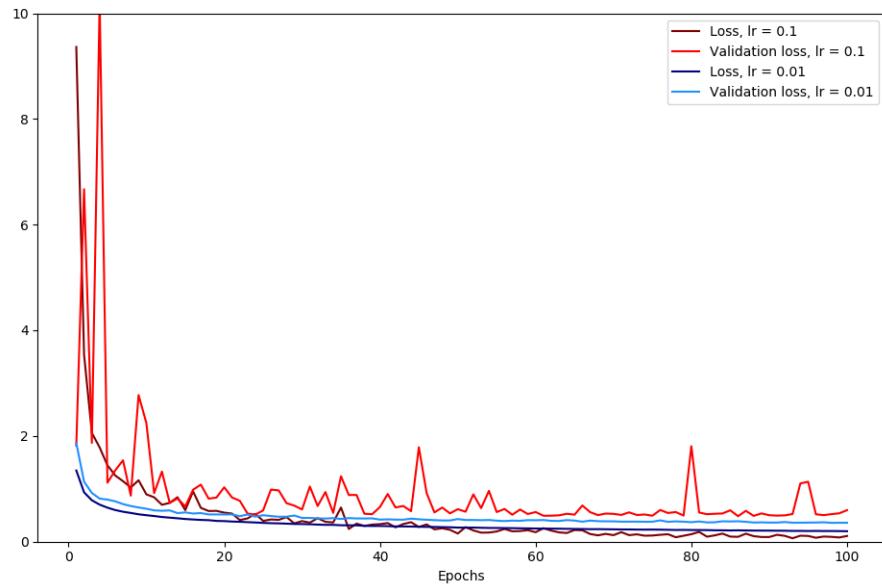
Mimo tych fluktuacji w ostatecznym teście na zbiorze testowym oba podejścia mają bardzo podobne wyniki:

- Animals3: 57.38% vs. 57.67%,
- Fruits: 88.92% vs. 87.92%.

Zgodnie z przewidywaniami już liniowy model radzi sobie bardzo dobrze z rozpoznawaniem owoców. Zwierzęta wypadają dużo gorzej (mimo mniejszej liczby klas), jednak i tak lepiej niż bym się spodziewał. (Pamiętajmy, że domyślny losowy wynik to 33.33%).



(a) Animals3



(b) Fruits

Rysunek 4: Porównanie ”płaskich” CNN

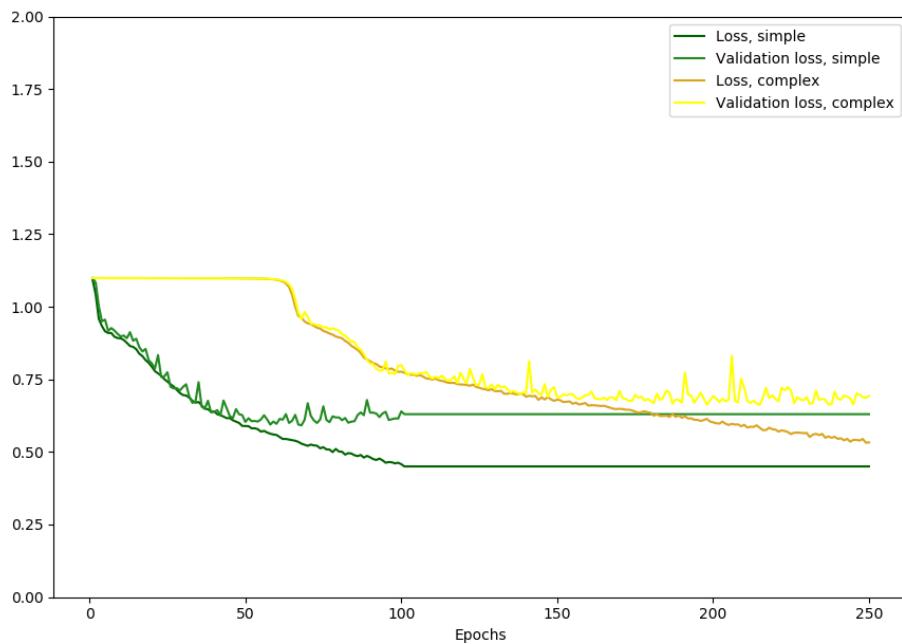
4.2 Dodanie konwolucji

Co było do przewidzenia dodanie jakiejkolwiek architektury od razu poprawia sytuację, na wykresach (Rysunek 6) widzimy poprawę w optymalizacji funkcji błędu w obu przypadkach. Dla ścisłości porównujemy wyniki myCNN - Flat, lr= 0.01 i myCNN - Simple. Ma to swoje pokrycie w wynikach końcowych:

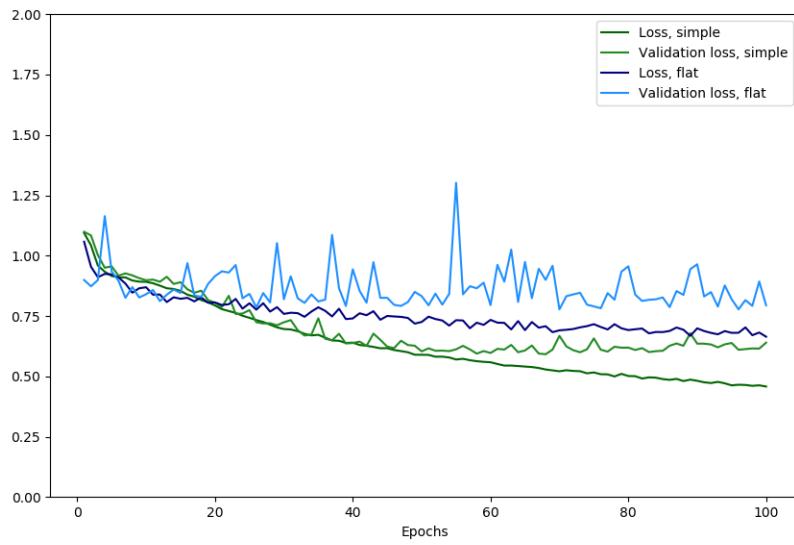
- Animals3: (F:) 57.67% vs. (S:) 68.33%,
- Fruits: (F:) 87.92% vs. (S:) 94.75%.

W przypadku owoców dostajemy już bardzo dobre wyniki i można się domyślać, że bardziej skomplikowane architektury nie poprawią dużo. Przy zwierzętach nasza sieć się dopiero rozpędza. Jednak niestety okazuje się, że myCNN - Complex nie działa zbyt dobrze i potrzebuje bardzo dużo czasu, aby w ogóle doścignąć sieć Simple. Prawdopodobnie obrazki są zbyt małe, aby taka architektura miała sens. Na wykresie 5 zamieszczam wykresy funkcji błędu dla odpowiednich sieci (zbiór Animals3).

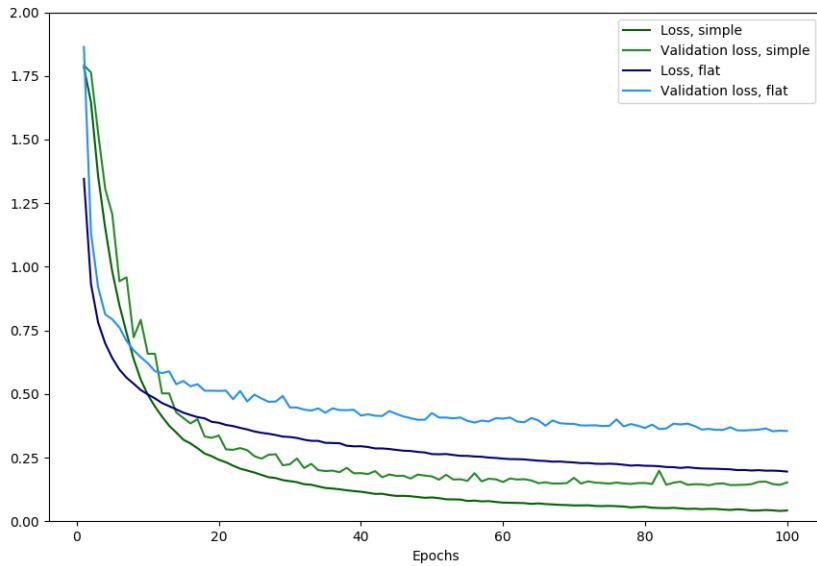
Jeśli chodzi o końcowy wynik to w trakcie jednej z pierwszych prób sieci Complex udało się przekroczyć 70%, jednak dla ścisłości zamieszczam wynik z końcowej próby, który był równy 64.5%. Niestety jest to ciężko powtarzalny eksperyment, gdyż taka sieć trenuje się parę godzin.



Rysunek 5: Simple vs. Complex



(a) Animals3



(b) Fruits

Rysunek 6: Flat vs. Simple

4.3 Tensorflow

Popatrzmy na wykresy 7. Na pierwszym widać bardzo szybkie i duże przeuczenie sieci bez augmentacji. Sugeruje to gwałtownie wzrastająca funkcja błędu na zbiorze walidacyjnym przy bardzo małych wartościach na zbiorze treningowym. Na drugim nie zachodzi takie zjawisko. Prawdopodobnie wynika to z moich początkowych obserwacji, że zbiór owoców jest mniej podatny na przeuczenia.

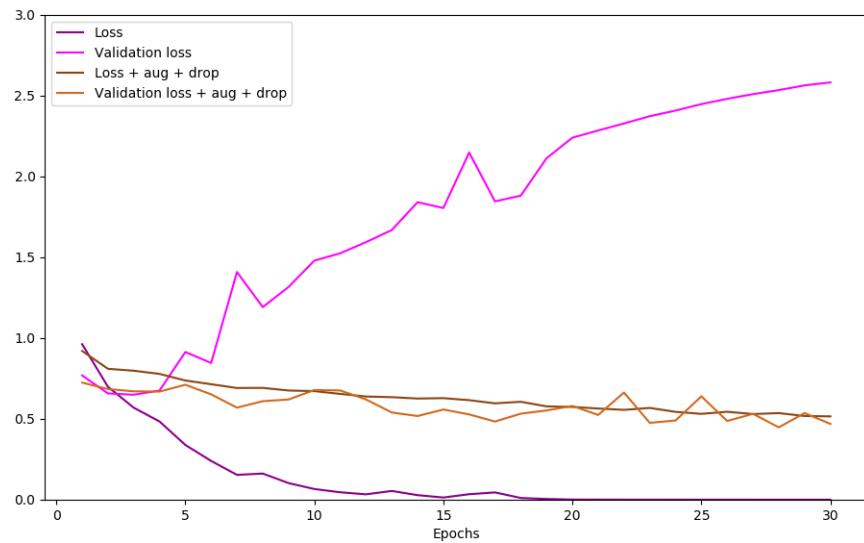
Ciekawym zjawiskiem jest lekko wzrastająca funkcja błędu na zbiorze treningowym przy ulepszonym modelu. Jednak ma to swoje wyjaśnienie, pamiętajmy, że zbiór treningowy nigdy nie widzi tych samych danych dwa razy. (Wzrost sam w sobie jest pewnie przypadkowy, bardziej chodzi o to, że funkcja nie maleje). Jeżeli jeszcze nie jesteśmy przekonani co do mocy augmentacji i dropoutu to popatrzmy na finalne wyniki:

- Animals3: 70.67% vs. 81.01%,
- Fruits: 94.75% vs. 99.17%.

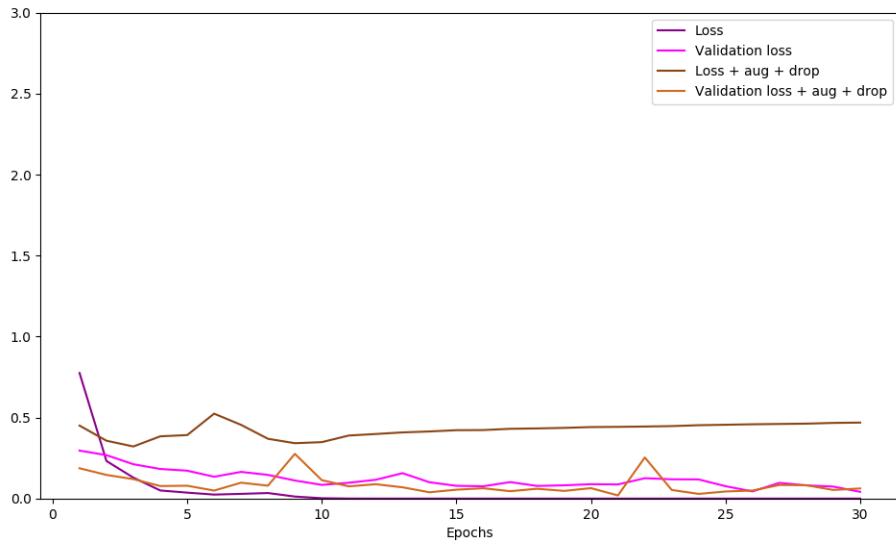
Oczywiście na korzyść ulepszonego modelu w obu przypadkach. Co ciekawe zauważmy, że sieci Pure nie są dużo lepsze niż Simple, a teoretycznie mają dostęp do dużo większej liczby danych (mam na myśli rozmiar obrazków). Prawdopodobnie chodzi o przeuczenie.

4.4 CNN - podsumowanie

Na koniec popatrzmy na wykresy podsumowanych dokładności na zbiorach walidacyjnych w trakcie uczenia (8). Raczej nie widać tu żadnych nowych wniosków, ale można potwierdzić wszystkie poprzednie. Na przykład na obu zbiorach widać większe fluktuacje przy większym learning rate.

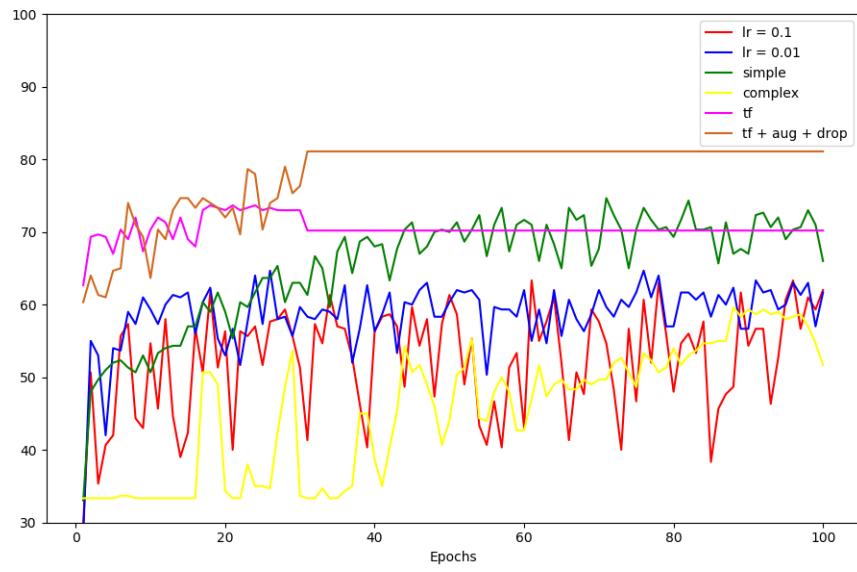


(a) Animals3

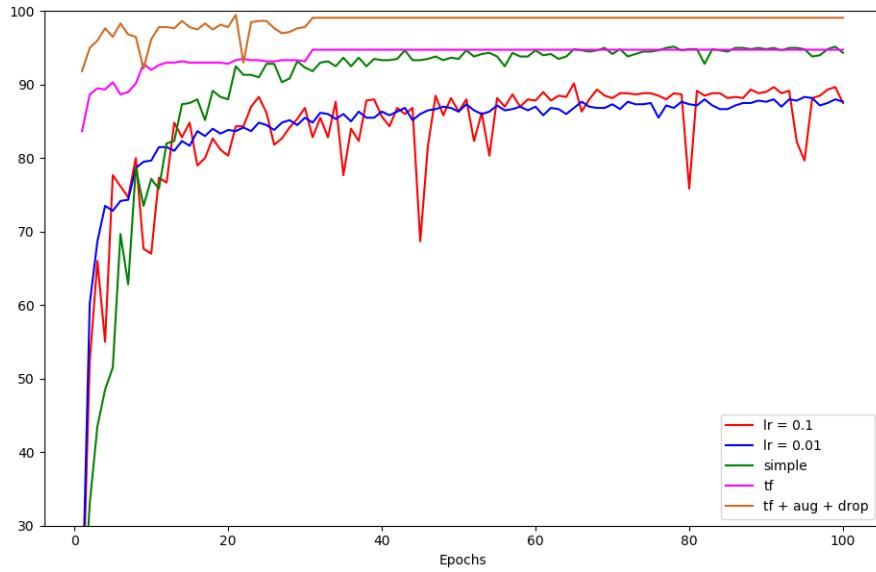


(b) Fruits

Rysunek 7: Pure vs. aug + drop



(a) Animals3



(b) Fruits

Rysunek 8: Validation Accuracy

4.5 Głosowania

Do głosowań wybrałem sieci własnej implementacji, tzn:

- myCNN - Flat lr= 0.01,
- myCNN - Simple,
- myCNN - Complex.

Decyzja była podyktowana głównie powodami technicznymi. Prowadziłem testy tylko na zbiorze Animals3. Główny problem polega na tym jakie wagi dobrać. Na pewno chcemy, aby najlepsza sieć (tutaj: Simple) miała największe prawo głosu. Jak zważyć pozostałe? Po kilku niezdarnych próbach postanowiłem sprawdzić wszystkie możliwości. To znaczy dla obu pozostałych sieci przetestować współczynniki z przedziału [0, 1] co 0.1. Wyniki przedstawia Rysunek 9.

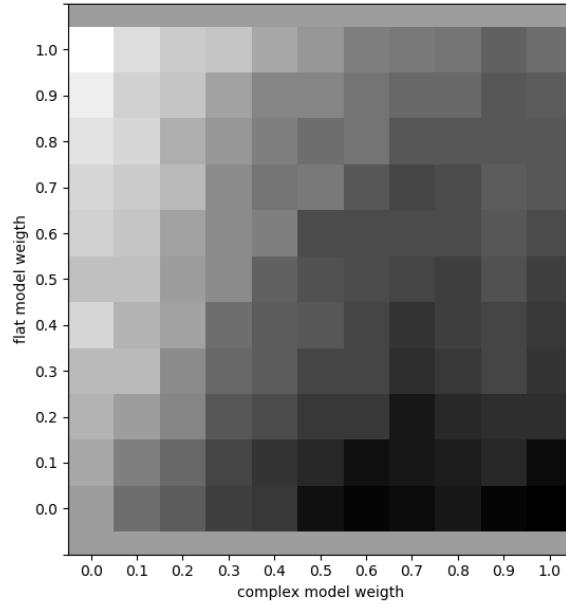
Im ciemniejsza kratka, tym lepszy wynik. Wynik sieci Simple (68.33%) oznacza kolor na dolnym i górnym obramowaniu. Więc już na oko widać, że ta metoda przynosi efekty, jest niewiele wyników gorszych niż wynik sieci Simple. A nawet te gorsze, szczególnie w lewym górnym rogu nie są aż takie złe. Najgorszy wynik to 65.5%. Jednak skupmy się na najlepszych wynikach. Widzimy, że sieć Flat nie pomaga, najlepsze wyniki mamy, gdy jej współczynnik jest mały, najlepiej 0. A co ciekawe najlepszy wynik to ten w prawym dolnym rogu, 72.8%, (czyli zdecydowanie lepszy niż Simple). Jest on wynikiem głosowania przy wagach:

- Flat - 0.0,
- Simple - 1.0,
- Complex - 1.0.

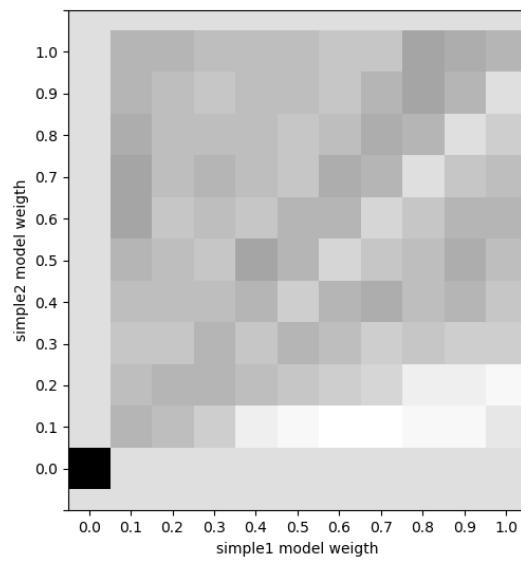
Czyli mimo tego, że Complex miało wynik "tylko" niecałe 65% to jest pomocne i podnosi ogólny wynik o prawie 5%.

Idąc tym tokiem rozumowania uznałem, że w takim razie może dwie sieci Simple sobie poradzą najlepiej. Wytrenowałem drugi model Simple o bardzo podobnej skuteczności, jednak jak widać na Rysunku 10 (lewy dolny róg to wartość 72.8% dało to mierne efekty. Najlepszy wynik przy dość losowych wagach to 69.5%, a często wyniki były jeszcze gorsze niż oryginalnie.

Prawdopodobnie wynika to z tego, że modele były zbyt "podobne" co sprawiło, że nie dawały sobie żadnych nowych informacji i jeśli czegoś nie wiedziały, to dalej nie wiedziały.



Rysunek 9: Voting Flat, Simple, Complex

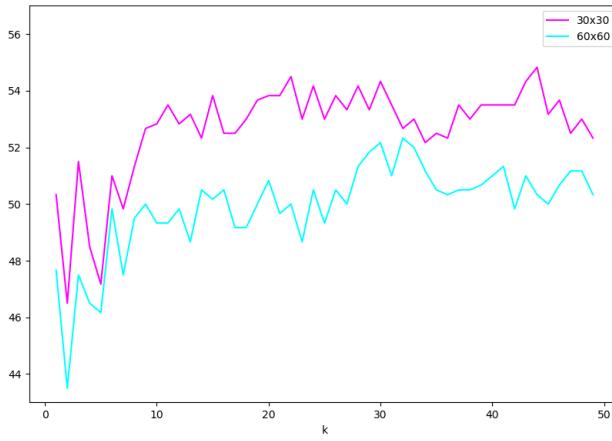


Rysunek 10: Voting Simple, Simple

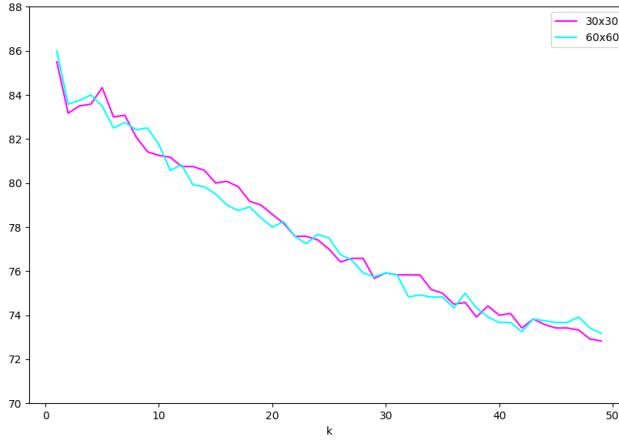
4.6 kNN

Na Rysunku 11 przedstawiłem wyniki jakie miał algorytm kNN dla różnych parametrów k . Po pierwsze widzimy, że przy zwierzętach lepiej sprawdzają się małe rozmiary obrazków. Może to wynikać ze "skondensowania" cech tak samo jak ma to miejsce przy konwolucji. Widzimy też, że dokładność raczej rośnie wraz z k , osiągając szczyt w 42. Może to oznaczać, że więcej cech nawet te mniej wyraźne wpływają na dokładność.

Przy owocach po pierwsze widzimy, że zaczynamy z dość wysokiego wyniku (porównywalnego do myCNN - Flat), a później już jest tylko gorzej. Wydaje się, że może to oznaczać, że mamy jakąś jedną cechę obrazka, która jest bardzo ważna i grupuje obrazki, a jak zaczniemy się rozdrabniać i patrzeć na więcej sąsiadów to zaczynają się pojawiać takie cechy jak oświetlenie, ustawnienie itp. Jest to uczenie bez nadzoru, więc nie mamy żadnego feedbacku i algorytm zaczyna brać pod uwagę cechy, których nie powinien. I tak bardzo wysoką skuteczność kNN na początku można uznać za zaskakującą.



(a) Animals3



(b) Fruits

Rysunek 11: kNN Results

5 Obserwacje i Ciekawostki

W ostatniej sekcji opiszę kilka ciekawostek i obserwacji.

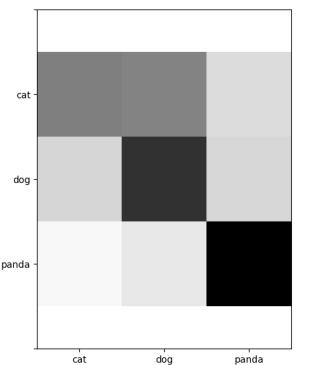
5.1 Pomyłki

Dla sieci Simple i najlepszych kNN-ów narysowałem (12) wykresy przedstawiające jak algorytmy odgadywali dane testowe na obu zbiorach danych.

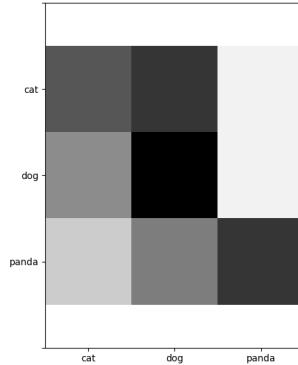
W Simple widzimy, że w zbiorze Animals3 z pandami nie było dużego problemu, natomiast koty i psy były ze sobą dużo częściej mylone. Tak jak pisałem na początku wynika to z charakterystycznego umaszczenia pand.

Sytuacja jest ciekawsza przy owocach. Po pierwsze aż 199 śliwek i 198 pomidorów zostało odgadniętych poprawnie. Banany były często mylone z mango i pomarańcze z persimonami. W prosty sposób widać zależność dotyczącą kolorów. Zaskakujące może być, że to mango i banany były najbardziej problematyczne. Przeglądając obrazki może się odnieść wrażenie, że największy problem będzie z pomarańczami i persimonami.

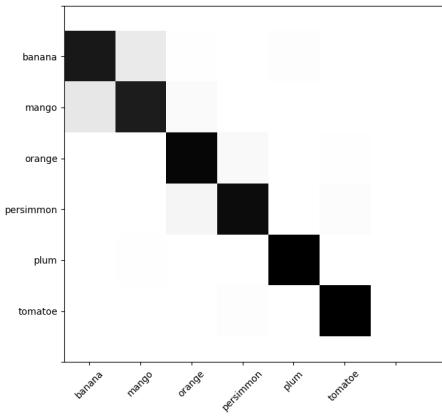
Jeśli chodzi o kNN to przestaje być widać przewagę pand. Wszystkie 200 śliwek było dobrze rozpoznane, jednak dokładność pomidorów spadła do tylko 159. Mimo, że dzieje się dużo dziwnych rzeczy, nadal wygląda na to, że największy problem jest z bananami i mango.



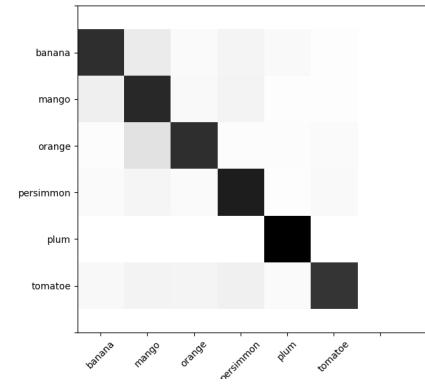
(a) Animals3 - Simple



(b) Animals3 - kNN



(c) Fruits - Simple



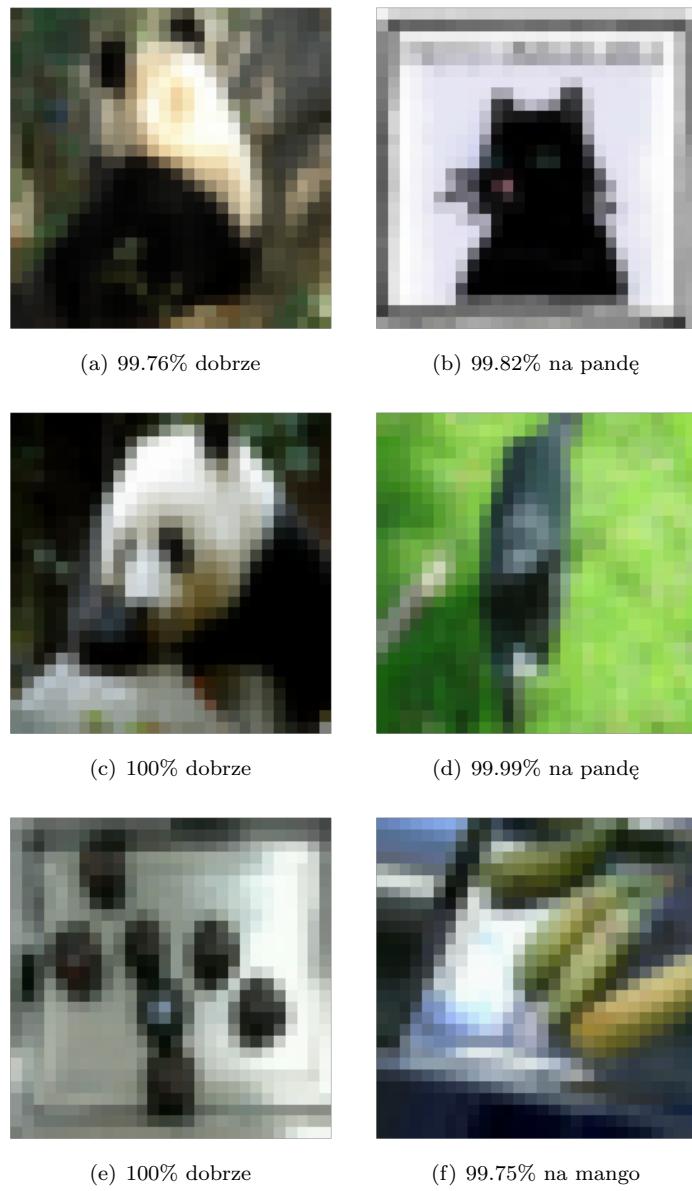
(d) Fruits - kNN

Rysunek 12: Accuracy by class

5.2 Najpewniejsze

Dla różnych sieci sprawdziłem, których z testowych obrazków były najpewniejsze, w dwóch wariantach: pewne - poprawne i pewne - niepoprawne. Co to znaczy najpewniejsze? Chodzi o ten przeskalowany wynik tuż po użyciu softmaxa. Czyli innymi słowy jak bardzo sieć jest pewna swojej decyzji. Na Rysunku 13 widzimy wyniki sprawdzenia. Co ciekawe obrazek (f) był najpewniejszym złym we wszystkich badanych sieciach.

Widzimy też, że na wszystkich najpewniejszych obrazkach przy zwierzętach głos szedł na pandę. Może to sugerować, że faktycznie cecha pandy, o której już wiele razy wspominałem jest bardzo "mocna". Dodatkowo popatrzmy na obrazek (d). Można zgadywać, że zielone tło i czarny kształt na środku sprawiły, że głos został w tak pewny sposób oddany na pandę. Potwierdza to jedynie, że wszystko ma tu jakiś sens i da się解释. :)



Rysunek 13: Najpewniejsze

6 Podsumowanie

6.1 Wyniki

Wyniki posortowane względem kolumny Animals3.

Model	Animals3	Fruits
tfCNN - aug + drop	81.01%	99.17%
myCNN - Simple + Complex	72.83%	-%
tfCNN - Pure	70.67%	94.75%
myCNN - Simple + Simple	69.50%	-%
myCNN - Simple	68.33%	94.75%
myCNN - Complex	64.50%	-%
myCNN - Flat, lr= 0.01	57.67%	87.92%
myCNN - Flat, lr= 0.1	57.38%	88.92%
kNN - best performance	54.83%	86.00%

Tabela 1: Wyniki

Zdaję sobie sprawę, że trochę używam zbioru testowego jako walidacyjnego, jednak uznałem, że nie ma to większego znaczenia, bo nie są to żadne zawody i "optymalizację" hiperparametrów, które wynikają są nieznaczne.

6.2 Omówienie wyników

Co wynika z powyższych wyników? Głównie wyższość bibliotek, które już są zaimplementowane nad "zwykłe" implementacje. Wynika też oczywiście, że konwolucje same w sobie są bardzo mocne jeśli chodzi o rozpoznawanie obrazków i algorytmy liniowe lub bardzo proste (jak kNN) zostawiają daleko w tyle. Myślę, że ciekawe jest też to, że głosowanie już nawet dwóch sieci może dać o tyle lepsze wyniki, ale tylko o ile te sieci są istotnie różne.

7 Źródła

Aby przyswoić teorię korzystałem głównie z **tego kursu online**. Jeśli chodzi o obsługę tensorflow to głównie wzorowałem się na **tym tutorialu**.