POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Informatyka

SPECJALNOŚĆ: Grafika i Systemy Multimedialne

PRACA DYPLOMOWA MAGISTERSKA

Detekcja manipulacji zawartości zdjęć przy pomocy metod uczenia głębokiego

Image manipulation detection using deep learning techniques

AUTOR:
Jarosław Ciołek-Żelechowski

PROMOTOR:
dr inż. Paweł Ksieniewicz

WROCŁAW 2020

SPIS TREŚCI

1. WSTĘP

O Leninie i falsyfikacji zdjęć

Fake News

Uczenie maszynowe na ratunek i zagładе(sieci GAN do generowania nowych twarzy)

Praca została podzielona na sześć rozdziałów w następujący sposób. Rozdział pierwszy jest wstępem wprowadzającym w tematykę pracy. Rozdział drugi dotyczy szeroko-pojmowanego uczenia maszynowego i jego elementów zastosowanych w niniejszej pracy. Rozdział trzeci przybliża naukowe osiągnięcia i prace o tematyce podobnej do opisywanej - detekcji manipulacji zdjęć. W rozdziale czwartym postawione zostają pewne założenia metodologiczne co do samej pracy, w tym te dotyczące wyboru zbiorów danych, sposobu przeprowadzania eksperymentów i sposobu ich oceny. Rozdział piąty przedstawia szczegółowe implementacje wybranych klasyfikatorów oraz uzyskane przez nie wyniki. Całość tego rozdziału zakończona jest zestawieniem i omówienie uzyskanych wyników. Szóstym, ostatnim rozdziałem pracy jest podsumowanie, które to w krótki sposób opisuje w jaki sposób zostały wypełnione założenia postawione niniejszej pracy, oraz kreśli w jaki sposób można by ją w dalszej części rozwijać.

Todo: Fancy it up Przeczytać i ogarnąć językowo

2. UCZENIE MASZYNOWE

Alan Turing w swojej pracy z 1950 roku *Computing Machinery and Inteligence*[38] zdefiniował pojęcie *obiekcji lady Lovelace*. Odnosiło się ono do krótkiej notatki[24] jaką lady Ada Lovelace poczyniła w 1843 roku podczas tłumaczenia na język angielski artykułu Luigi Menabrea[27], który to był streszczeniem wykładu Charlesa Babbage'a wygłoszonego w Turynie w 1841 roku. Wykład dotyczył projektu maszyny analitycznej której zadaniem było zautomatyzowanie niektórych obliczeń związanych z analizą matematyczną. Pojęcie to brzmi następująco:

Maszyna analityczna nie ma na celu zapoczątkowania czegokolwiek. Może wykonywać operacje, które możemy kazać jej przeprowadzać... Jej celem jest zwiększanie dostępności tego co umiemy już wykonać[38]

Turing, przywołał to pojęcie, zastanawiając się nad tym, czy komputery mogą się uczyć, tworzyć nowe rzeczy. Jak piszę on dalej w swoim artykule[38]: problem jest natury programistycznej i wymaga on stworzenia zupełnie innego, jak na tamte czasy, środowiska i sposobu pojmowania nauczania jako takiego. Wierzył jednak, że jest to możliwe.

Termin *Uczenie Maszynowe* po raz pierwszy pojawił się w 1959 roku w pracy naukowej autorstwa Arthura Samuela:

Uczenie maszynowe to dziedzina nauki dająca komputerom możliwość uczenia się bez konieczności ich jawnego programowania.[34]

Praca ta dotyczyła maszyny, która przez ok. 8 godzin *uczyła się* gry w warcaby znając jedynie jej zasady, posiadając pewnego rodzaju funkcję celu(zbijanie pionów przeciwnika), oraz macierz losowych liczb, której to zmiany i przekształcenia miały na celu reprezentować inne podejścia(nową wiedzę). Sprawdzianem sukcesu maszyny, było pokonanie jej twórcy.

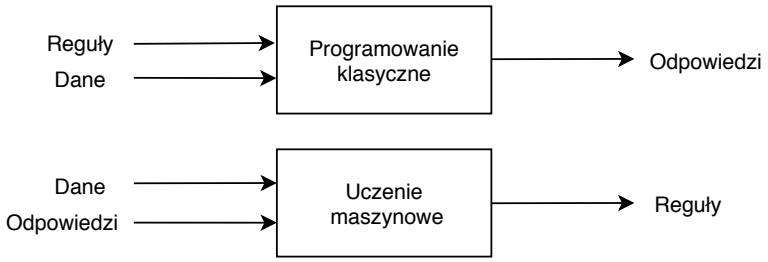
Bardziej techniczną definicję podał w 1997 roku Tom Mitchel, w rozdziale otwierającym swoją książki pt. *Machine Learning*:

Mówimy, że program komputerowy uczy się na podstawie doświadczenia E w odniesieniu do jakiegoś zadania T i pewnej miary wydajności P, jeśli jego wydajność (mierzona przez P) wobec zadania T wzrasta wraz z nabywaniem doświadczenia E.[28]

Tym samym uczenie maszynowe zostało sprowadzone do problemu, w którym to posiadamy trzy elementy T, P i E. W dalszej części Tom Mitchel podaje przykład powyżej definicji zrealizowanej dla gry w warcaby(wyraźny ukłon w stronę pracy Arthura Samuela[34]):
— Zadanie T: gra w warcaby,
— Miara Wydajności P: procent gier wygranych na oponentów,
— Doświadczenie E: granie partii przeciwko samemu sobie.

Tym samym można rozumieć uczenie maszynowe jako nowy paradygmat programowania. W programowaniu klasycznym, programista definiuje reguły według których program przetwarza dane wejściowe, generujący tym samym dane wyjściowe - odpowiedź pracy programu(patrz

rysunek 2.1). W przypadku uczenia maszynowego mamy sytuację w której programista wprowadza dane oraz odpowiedzi i oczekuje uzyskać od programu zestawu reguł, według których dane odpowiedzi zostały przypisane do konkretnych próbek. Reguły te, mają posłużyć w dalszej części do przetwarzania nowych danych.



Rys. 2.1: Uczenie maszynowe: nowy model programowania

Podsumowując, system uczenia maszynowego jest trenowany, a nie programowany w sposób jawny. Celem programisty jest przedstawienie mu odpowiedniej dużej ilości przykładów wyników, tak by sam system określił ich statystyczną strukturę, co w dalszej części pozwoli na ustalenie reguł umożliwiających automatyzację całego procesu. Ważne, by podkreślić tutaj znaczenie danych wejściowych, które to często określa się terminem *Big Data*[36].

2.1. BIG DATA

Popularyzację tego terminu przypisuję się do wykładu autorstwa Johna Mashey'a[25] jaki wygłosił w 1998 roku. Zauważył on, że wraz ze spadkiem cen nośników do przechowywania i gromadzenia danych, ilość zbieranych przez ludzkość informacji wzrasta z każdym rokiem. Co więcej, ilość tych danych sprawia, że ich analiza przestała być możliwa do realizacja w sposób inny niż automatyczny. Za kryterium, czy dany zbiór informacji można określić jako Big Data, często przywołuje się te podane przez Douglasa Laney'ego[21] i określane mianem *3V*, które to rozwija się jako rozmiar(ang. *volume*), różnorodność(ang. *variety*) i prędkość (ang. *velocity*). Oznaczają one kolejno:

— **rozmiar** - dane są zbyt duże by mieściły się na standardowych dyskach twardych,
— **różnorodność** - dane pochodzą z różnych źródeł, są niejednorodne i słabo ustrukturyzowane,
— **prędkość** - tempo napływu nowych danych jest znaczące, co w rezultacie utrudnia ich analizowanie.

Jednym z kluczowych zjawisk przyczyniającym się do procesu nagłego przyrostu ilości i źródeł danych jest Internet przedmiotów (ang. *Internet of Things*[16]). Według tej koncepcji, różnego typu urządzenia osadzone w urządzeniach codziennego użytku(np. odkurzacze, żarówki, instalacje grzewcze), czy też w maszynach przemysłowych, zbierają dane o otoczeniu, czy samym procesie w którym uczestniczą, a ponadto mają możliwość komunikacji pomiędzy sobą, ale również z jednostką centralną jeśli taka występuje. Wprowadza to nowy wymiar w możliwościach automatyzacji procesów i tworzy nową przestrzeń do tworzenia się złożonych, inteligentnych systemów.

2.2. RODZAJE SYSTEMÓW UCZENIA MASZYNOWEGO

Istnieje wiele metod podziału uczenia maszynowego na kategorię. Jedną z najpopularniejszych jest podział ze względu na sposób uczenia się oraz zadanie do wykonania[32]:
— Uczenie nadzorowane:
  — klasyfikacja,
  — regresja.
— Uczenie nienadzorowane:
  — klasteryzacja,
  — redukcja wymiarowości,
  — uczenie przy użyciu reguł asocjacyjnych.
— Uczenie ze wzmocnieniem.

W uczeniu z nadzorem(ang. *supervised learning*), którym zajmę się w poniżej pracy, dane trenujące przekazane algorytmowi zawierają dołączone do nich etykiety, czyli rozwiązania problemu. Celem algorytmu jest stworzenie funkcji(nazywanej też hipotezą[5]), która będzie maksymalizować swoją skuteczność względem zadanych kryteriów.

Na zbiór uczący Z_u składa się zbiór n wektorów, gdzie każdy z nich opisuje pojedynczy obiekt(patrz wzór 2.1):

$$Z_u = \{(x_1, y_1), ..., (x_n, y_n)\} \tag{2.1}$$

I tak w powyższym równaniu2.1) i-ty wektor oznaczony byłby jako x_i i odpowiadałaby mu etykieta oznaczona jako y_i. W zależności od typu danych przechowywanych pod etykietą będziemy mieć do czynienia z zadaniem klasyfikacji(etykieta należy do skończonego i przeliczalnego zbioru) lub regresji(wartości przyjmowane przez etykietę należą do przestrzeni ciągłej). Ważne żeby dodać że na wektor x_i może składać się m-liczb, gdzie każdą z tych liczb będziemy nazywać atrybutem lub cechą danego wektora(patrz wzór 2.2):

$$x_i = \{x_{1,m}, ..., x_{i,m}\} \tag{2.2}$$

Etykietą y_i, w tym rozumowaniu, oznaczamy prawdziwą wartość funkcji, którą to chcemy by nasz algorytm odwzorował. Algorytm ten, nazywamy również modelem i opisujemy go jako następującą funkcję f(patrz wzór 2.3):

$$f(x) : x \in X \mapsto y \in Y \tag{2.3}$$

Jak widać zadaniem powyższej funkcji jest przyporządkowanie wektorom wejściowym etykiet. Jej skuteczność jest mierzona przy użyciu wybranej przez nas metryki na zbiorze testowym. Zbiór testowy musi posiadać tą samą strukturę co zbiór trenujący. Można traktować metrykę M jako funkcję wyższego rzędu, której pierwszym argumentem jest sam model f, a drugim zbiór testowy Z_t. Dziedziną metryki jest zazwyczaj podzbiór liczb rzeczywistych z zakresu od 0 do 1(patrz wzór 2.4)

$$M(f, Z_t) : (f, Z_t) \mapsto m \in [0, 1] \tag{2.4}$$

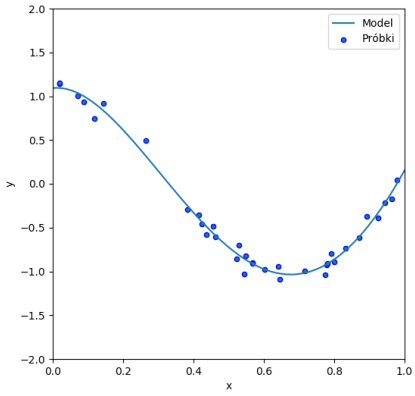
Przykładem metryk stosowanych w uczeniu maszynowym są między innymi dokładność, czułość, precyzja, a także miara *F* (ang. *F-score*), które omówione zostały szczegółowo w dalszej części rozdziału.

2.3. PROBLEM PRZEUCZANIA, NIEDOUCZANIA

Wykorzystanie osobnego zbioru do badania jakości modelu w uczeniu nadzorowanym związane jest bezpośrednio z takimi problemami jak nadmierne dopasowanie, przeuczenia(ang. *overfitting*) oraz niedouczenie (ang. *underfitting*)[12]. Zadaniem, które dobrze nadaje się do graficznej ilustracji powyższych problemów jest regresja liniowa. Jak już zostało opisane powyżej w problemie tym, etykiety przykładów są liczbami należącymi do przestrzeni ciągłej, a zadaniem algorytmu jest wyznaczenie funkcji f(patrz wzór 2.5):

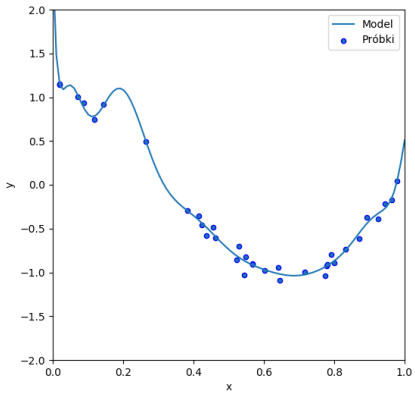
$$f(x) : x \mapsto y \in \mathbb{R} \tag{2.5}$$

Dodatkowo dla lepszej interpretacji graficznej każdy wektor w przestrzeni x będzie posiadał tylko jedną cechę. Przykład prawidłowego dopasowania wygląda następująco(patrz rysunek 2.2):



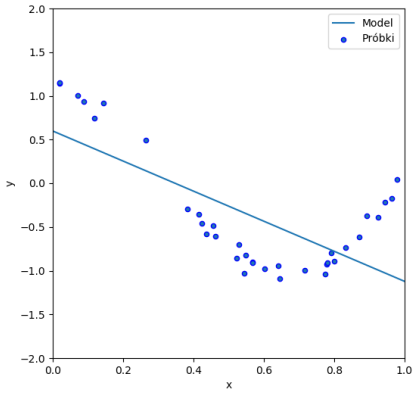
Rys. 2.2: Przykład prawidłowego dopasowania

Problem nadmiernego dopasowania cechuje się tym, że taki model zbyt mocno generuje charakterystykę dla danych trenujących, zawierając w niej również występujące tam szumy(patrz rysunek 2.3):



Rys. 2.3: Przykład nadmiernego dopasowania

Odwrotnym problem jest problem niedouczenia. Występuję on na przykład wtedy, kiedy zastosujemy zbyt prosty model w stosunku do poziomu skomplikowania zbioru danych. Wynik takiej operacji widoczny jest na rysunku 2.4



Rys. 2.4: Przykład niedostatecznego dopasowania

2.4. ZADANIE KLASYFIKACJI BINARNEJ

W niniejszej pracy będę zajmował się zadaniem należącym do problemów klasyfikacji binarnej, czyli takiej w której przestrzeń etykiet ogranicza się do dwóch elementów(patrz wzór 2.6):

$$y = \{0, 1\} \tag{2.6}$$

Klasy 0 i 1 w powyższym wzorze2.6 są przykładowe i bardziej niż ich wartość interesuje Nas liczność zbioru y. Tym samym mając zdefiniowaną przestrzeń możliwych wartości modelu można zdefiniować szereg miar, które są wykorzystywane do oceny jego wyników. Podstawowym elementem zadania ewaluacji modelu w zadaniu klasyfikacji binarnej jest macierz, tablica błędów widoczna w tabeli 2.1. Do jej skonstruowania potrzebujemy oczywiście przetestować działanie naszego modelu na zbiorze testowym Z_t. Poszczególnym reprezentantom tego zbioru, przypisywane są pozytywne lub negatywne etykiety, teraz w zależności od tego czy dany element zbioru x był faktycznie pozytywny czy negatywny można go wpisać w macierz błędów 2.1.

	Klasyfikacja pozytywna	**Klasyfikacja negatywna**
Stan pozytywny	Prawdziwie dodatnia (ang. *true positive*, TP)	Fałszywie ujemna (ang. *false negative*, FN)
Stan negatywny	Fałszywie dodatnia (ang. *false positive*, FP)	Prawdziwie ujemna (ang. *true negative*, TN)

Tabela 2.1: Tablica pomyłek, możliwe wyniki klasyfikacji binarnej

2.5. EWALUACJA MODELU - MIARY JAKOŚCI

Korzystając z opisanej powyżej macierzy błędów 2.1 w łatwy sposób można przedstawić definicję szeregu miar do oceny modelu, z których to skorzystałem w niniejszej pracy:
— **Dokładność** - procent poprawnych klasyfikacji, opisanych wzorem 2.7:

$$\mathbf{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.7}$$

— **Czułość** - stosunek prawidłowych wyników pozytywnych do sumy prawidłowych wyników pozytywnych oraz błędnych wyników negatywnych, który można rozumieć jako zdolność modelu do poprawnego etykietowania(patrz wzór 2.8),

$$\mathbf{recall} = \frac{TP}{TP + FN} \tag{2.8}$$

— **Precyzja** - stosunek prawidłowych wyników pozytywnych do sumy prawidłowych wyników pozytywnych oraz błędnych wyników pozytywnych, który można rozumieć jako zdolność modelu do niepoprawnej klasyfikacji próbek negatywnych jako pozytywne(patrz wzór 2.9),

$$\mathbf{precision} = \frac{TP}{TP + FP} \tag{2.9}$$

— **miara *F*** - będąca średnią ważoną z czułości i precyzji(patrz wzór 2.10).

$$\mathbf{F1} = \frac{2 * precision * recall}{precision + recall} \tag{2.10}$$

2.6. SPLOTOWE SIECI NEURONOWE

Splotowe, inaczej konwolucyjne, sieci neuronowe (ang. *CNN - convolutional neural networks*) stanowią wynik badań nad korą wzrokową i od 1980 roku są używane w zadaniach rozpoznawania obrazów[12]. Podstawową różnicą w stosunku do sieci neuronowych jest stosowanie wielowymiarowej operacji splotu, realizowanej za pomocą szeregu filtrów, których to parametry dobierane są podczas trenowania sieci. Pozwala to siecią konwolucyjnym na nie przetwarzanie obrazów piksel po pikselu, a raczej poprzez zauważanie ogólnych cech obrazu i budowaniu z nich nowych struktur.

W analizie matematycznej operacją splotu(reprezentowana jako $*$) dwóch funkcji f i g, jest trzecia funkcja s, patrz wzór 2.11:

$$\begin{aligned} s(t) = (f * g)(t) &= \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau = \\ &= \int_{-\infty}^{\infty} f(t - \tau) g(\tau) d\tau \end{aligned} \tag{2.11}$$

, a w przypadku operacji dyskretnych, patrz wzór 2.12:

$$s(t) = (f * g)(t) = \sum_{\tau=-\infty}^{\infty} f(\tau)(g - \tau) \tag{2.12}$$

W literaturze opisującej sieci neuronowych przyjęło się określać $f(\tau)$ jako wejście, $g(t-\tau)$ jako jądro lub filtr(z ang. *kernel*, *filter*), a wynik samej operacji jako mapę atrybutów lub aktywacji (z ang. *feature map*, *activation map*)[30]. Zadanie dla typowego problemu klasyfikacji obrazów, operującego na dwuwymiarowym obrazie wejściowym M przy pomocy filtra K z założeniem wykorzystania wielowymiarowego splotu $S(i, j)$, można sformułować następująco 2.13:

$$\begin{aligned} S(i, j) = (M * K)(i, j) &= \sum_k \sum_l M(k, l) K(i - k, j - l) \\ = (K * M)(i, j) &= \sum_k \sum_l M(i - k, j - l) K(k, l) \end{aligned} \tag{2.13}$$

2.7. UCZENIE GŁĘBOKIE

Budowa sieci splotowych, a dokładnie fakt, że nie wymagały one połączeń pomiędzy wszystkimi neuronami w każdej z warstw oraz fakt, że zastosowanie operacji splotu, pozwala na zmniejszenie wymiarów zdjęcia pozwoliło budować sieci o większej ilości warstw ukrytych [32]. Za pierwszą pracę z stosującą uczenie głębokie(funkcje splotu, wsteczną propagację) uważa się tą z 1989 roku, której autorzy stworzyli model rozpoznający cyfry zawarte w kodach pocztowych[22]. Sam model okazał się sukcesem, problemem był jednak czas uczenia - $\sim$3 dni. Dodatkowo w tamtym okresie na popularności zyskiwały metody jądrowe(maszyny wektorów nośnych) oraz wszelkiego rodzaju algorytmy oparte o drzewa decyzyjne [12]. Warto tutaj również dodać, że zbiory danych konstruowane w latach 80 i 90 ubiegłego wieku, były zazwyczaj małe w porównaniu do dzisiejszych standardów, co premiowało algorytmy oparte o manualną ekstrakcję cech [3].

Krokiem w kierunku popularyzacji podejścia splotowego i samej idei uczenia głębokiego była praca Hintona i Salakhutdinova z 2006 roku, w której to autorzy zaproponowali sposób budujący coraz głębszy model składający się z ograniczonych maszyn Boltzmanna [15]. Pokazali oni że czas od pracy LeCuna z 1989 roku [22] zapewnił potrzebny postęp sprzętowy co przedstawiło dotychczasowy problem długiego czasu trenowania sieci jako problem optymalizacyjny. Zgodnie z danymi podawanymi przez F. Cholleta [3] szybkość dostępnych dla standardowych użytkowników procesorów wzrosła pomiędzy latami 1990 a 2010 o około 5000 razy. Dodatkowo stworzenie i udostępnienie takich zbiorów danych jak ImageNet, zawierających powyżej 14 milionów tagowanych zdjęć wysokiej rozdzielczości i zbudowanie wokół niego konkursu ImageNet Large Scale Visual Recognition Challenge [33] sprawiło że praktycznie każdy jest w stanie przeprowadzić proste eksperymenty w oparciu o uczenie głębokie.

Dalsza demokratyzacja uczenia głębokiego w postaci serwisu *kaggle.com*, który to hostuje nie tylko zbiory danych, ale również konkursy na najlepsze modele, w połączeniu z łatwością w korzystaniu z technologii CUDA(z ang. *Compute Unified Device Architecture*) już nie tylko w języku C++, ale również Python - sprawia że odpalanie modelów sieci głębokich jest proste i równe szybkie co modeli opartych o klasyczne uczenie maszynowe[3].

2.7.1. Transfer Learning

Jedną z najpopularniejszych technik w uczeniu głębokim jest technika Transfer Learningu, która to pozwala na ponowne wykorzystanie gotowego i nauczonego modelu do rozpatrzeniu nowego problemu [12]. Wyobraźmy sobie przykład w którym posiadamy klasyfikator bardzo dobrze rozróżniający psy od kotów. My sami zaś chcemy rozwiązać zadanie rozpoznawania psów, kotów i ludzi. Możemy wykorzystując metodologie Transfer Learning skorzystać z modelu rozpoznającego psy i koty jako z bloczka budulcowego, tym samym upraszczając sobie zadanie, nie mówiąc już o tym że znacząco oszczędzamy na czasie.

Oprócz wykorzystania całego modelu, możemy również posłużyć się częścią architektury danego rozwiązania. Chodzi mi o sytuację w której zamrażamy możliwość uczenia dla wag z kilku pierwszych warstw jakiegoś modelu uczenia głębokiego. Warstwy te, jak ustaliliśmy już wcześniej, odpowiedzialne są na definiowanie bardzo prostych kształtów, zapamiętanie ich powinno *przenieść* tą własność na model docelowy którego nasz model jest jedynie częścią [32].

3. ANALIZA ISTNIEJĄCYCH METOD W OBRĘBIE DZIEDZINY

Badanie manipulacji zdjęć jest dość obszernym dziedziną w której to, co więcej, można wyróżnić szereg podejść i prób zrozumienia problemu. Stanem wiedzy o podejściach statystycznych, lub takich opartych o metadane zdjęcia jest książka *Photo Forensics*, autorstwa Hany Farida [10]. Podejścia oparte o analizę poszczególnych zdjęć bez używania uczenia maszynowego w większości opierają się o metaznaczniki zawarte w strukturze plików *.JPEG. Metaznaczniki te są zarówno wspierane po stronie aparatów cyfrowych jak i oprogramowania do edycji zdjęć. Dla zdjęć cyfrowych generowane są dane dotyczące modelu aparatu, jego ustawień(takich jak czas naświetlania, wartość przesłony czy czułość matrycy w ISO), daty wykonania zdjęcia, czy nawet współrzędne GPS miejsca w którym zdjęcie zostało zrobione. Dla programów do obróbki zdjęć generowane są informację o nazwie użytego programu i w zależności od rodzaju oprogramowania - część danych utworzonych przez aparat zostaje *wymazana*. Takie podejście sprawia, że do sprawdzenia czy dane zdjęcie zostało przerobione, wystarczy przeczytać jego metadane [10].

Innym podejściem, choć ciągle bazującym na metaznacznikach jest zbudowanie *odcisku palca* aparatu o zadanych ustawieniach i porównywaniu spreparowanych zdjęć do obrazów rozpatrywanych jako potencjalnie przerobione [37]. Problemów z zaproponowanym przez A. Swaminathan, M. Wu oraz K. J. Ray Liu jest kilka i sami wspominają o nich w swojej pracy [37]: całość wymaga zbudowania modelu poszczególnych aparatów, co znacząco zimniejsza możliwości skalowania aplikacji oraz dodatkowo, co stwierdzają sami autorzy, im więcej rozpatrywanych aparatów, tym mniejsza dokładność predykcji - aparaty tego samego producenta, będące nie daleko od siebie pod względem rodziny modelu produkują bardzo podobne *odciski palców*. Co więcej, do działania i samego treningu, opisywany model wymaga zdjęć z wypełnionymi znacznikami.

Inną rodziną podejść są te zorientowane na pomijanie metadanych, podział zdjęcia na mniejsze części, a następnie sprawdzanie ich właściwości względem siebie. Przykładem może być praca z 2015 roku autorstwa M. Goljan and J. Fridrich, w której opisują stworzony przez siebie wariant metody CFA(z ang. *color filter array*), nazwany przez siebie CRM(z ang. *color rich model*) [11]. Jego zadaniem jest stworzenie szeregu statystyk opisujących relacje pomiędzy kolorami pikseli w określonym obszarze. Następnie mając te informacje sprawdzane jest czy występują w obrazie nagłe przejścia w których zbierane statystyki wskazują na obiekt spoza oryginalnej przestrzeni. Takie podejście świetnie sprawdza się w sytuacji w której manipulacja polega na wstawieniu obcego elementu w zdjęcie. Problemem są oczywiście takie manipulacje, które ingerują w cały obszar zdjęcia, a mówiąc bardziej szczegółowo, w przestrzeń kolorów zdjęcia.

Podobny do opisanego powyżej jest pomysł pracy autorstwa D. Cozzolino, G. Poggi, L. Verdoliva z 2015 roku, z tą różnicą że zamiast przestrzeni barw autorzy sprawdzają szum występujący pomiędzy pikselami poszczególnych elementów zdjęcia [7]. Nagłe zmiany, lub pewne nieciągłości pozwalają autorom nie tylko stwierdzić czy zdjęcie zostało przerobione czy nie -

pozwala też zlokalizować miejsce w którym do takiej manipulacji potencjalnie doszło.

Innym podejściem bazującym na poprzednich ale jednak bardziej zwróconym w stronę uczenia głębokiego, była praca naukowców z USA z 2017 roku [1]. Podobnie jak poprzednicy, wykorzystali podział obrazu na mniejsze elementy z tą jednak różnicą że tym razem nie definiowali funkcji ekstrahującej z pomniejszych elementów cech - zamiast tego stanowiły one wejście klasyfikatora LSTM, którego pamięć została wykorzystana do przetworzenia pojedynczego zdjęcia. Z racji jednak że badali oni zmiany lokale natrafili na ograniczenia takie same jak inni uczeni [11] [7].

Kolejnym typem prób rozpoznawania czy dane zdjęcie zostało zmanipulowane czy nie jest wykorzystanie podwójnej kompresji JPEG - skorzystanie z algorytmu ELA(z ang. *Error Level Analysis*), co zostało opisane dość dokładnie w pracy N. Krawetza z 2007 roku[19]. Podczas zapisywania plików JPEG dochodzi do kompresji danych, domyślnie wartość kompresji jest ustawiona na 90%. Oznacza to że algorytm kompresji będzie przeglądał obszary wielkości 8x8 pikseli na zdjęciu w celu zaoszczędzenia $\sim 10\%$ danych. Ideą algorytmu ELA jest zapisanie zdjęcia podwójnie z kompresją pomiędzy $\sim 80\%$ - $\sim 90\%$, a następnie *odjęcie* od zdjęcia oryginalnego. Taki zabieg sprawi że dojdzie do wytworzeniu artefaktów kompresji - miejsc gdzie *poziom* uproszczenia jest różny dla elementów na zdjęciu. Tym samym będzie to oznaczało, że elementy te przed zastosowanie algorytmu ELA były różnej jakości, czyli nie należą do tego samego obrazka [23].

Jednym z ciekawszych i najbliższych mojej propozycji rozwiązania problemu jest praca z 2016 roku, która wykorzystuje autoencoder [39]. Ideą pracy autoencodera jest kompresja informacji wejściowej, a następnie próba odtworzenia jej w jak najlepszym stopniu. Okazuje się, że gdy zbadamy obraz oryginalny, skompresowany i odtworzony to wystąpią różnice w jakości odtwarzania w zależności od tego czy dany obszar był poddany manipulacji czy nie. Tym samym ponownie nie dość że dostaniemy wynik klasyfikacji jako nie manipulowane zdjęcie lub zmanipulowane, to jeszcze dostaniemy obszar który nasz model wytypował jako zmieniony względem oryginału.

4. ZAŁOŻENIA METODOLOGICZNE

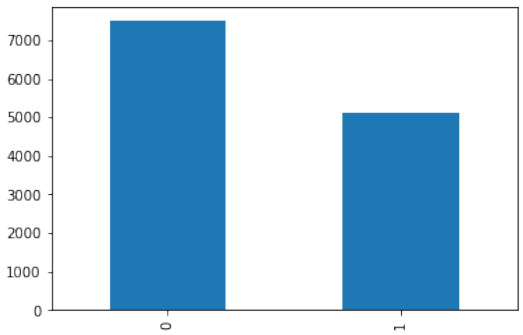
W poniższym rozdziale przedstawię elementy składowe jakie stanowiły części wykonanych eksperymentów.

4.1. WYBRANE ZBIORY DANYCH

W pracy posłużyłem się dwoma zbiorami danych:
— Casia image tampering detection evaluation database [9]
— The PS-Battles Dataset [14]

4.1.1. CASIA v.2

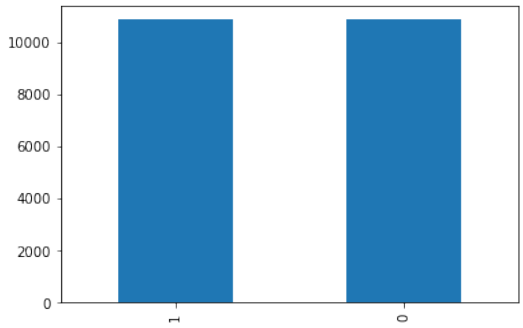
Opisywany zbiór posiada 12614 zdjęć, z czego 7941 z nich stanowi zdjęcia oryginalne, a kolejne 5123 to zdjęcia zmanipulowane(patrz rysunek 4.1). Zgodnie z artykułem [9] elementy zbioru różnią się wielkością od 320×240, aż do 800×600 pikseli. Manipulacja zdjęć jest wykonana poprzez wycinanie, kopiowanie i rozmazanie. Twórcy zbioru danych podzielili go na 9 kategorii: scena, zwierzę, architektura, postać, roślina, artykuł, natura, wnętrze i tekstura - na potrzeby pracy interesuje Nas tylko podział na dane prawdziwe i zmanipulowane. Dodatkowo zdjęcia zawarte w zbiorze zapisane są jako *.JPEG, *.BMP, *.TIFF.



Rys. 4.1: Ilość przedstawicieli danej klasy w zbiorze CASIA v.2

4.1.2. The PS-Battles Dataset

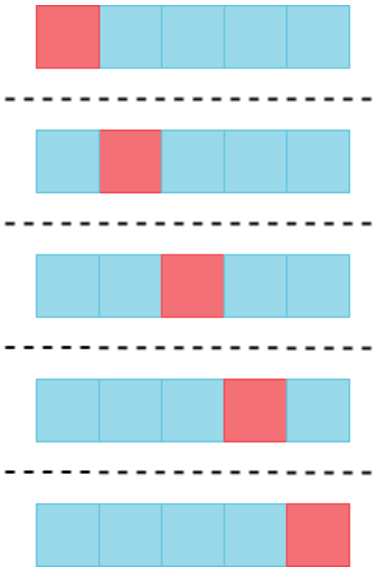
Powyższy zbiór jest liczniejszy - 21758 elementów, z czego po 10879 należy do każdej z dwóch klas(patrz rysunek 4.2). Formatem zdjęć jest *.JPEG lub *.PNG, a wielkość jest różna od 136×68, do aż 20000×12024 pikseli. Zbiór danych został stworzony przez społeczność forum internetowego *reddit.com/r/photoshopbattles* do której na moment pisania pracy należy $\sim 16,5$ mln użytkowników, którzy to codziennie dodają kolejne ~ 92 posty ze zdjęciami. Ideą tej społeczności jest wrzucanie zdjęć nieoczywistych, śmiesznych a następnie przerabianie ich na śmieszniejsze.



Rys. 4.2: Ilość przedstawicieli danej klasy w zbiorze PS-Battles

4.2. STRATYFIKOWANA K-KROTNA WALIDACJA KRZYŻOWA

Do przeprowadzenia wiarygodnych eksperymentów i wyliczeniu odpowiednich metryk pracy modeli skorzystałem z k-krotnej stratyfikowanej walidacji krzyżowej. W samej k-krotnej walidacji krzyżowej chodzi po prostu o podział zbioru Z na k równych elementów [12]. Następnie 1 z tych k zbiorów zostaje zbiorem testowym Z_t podczas gdy pozostałe $k-1$ zbiorów zostają zbiorem uczącym Z_u. Całe zadanie jest powtarzane k razy. Całość została pokazana na rysunku 4.3



Rys. 4.3: Przebieg 5-krotnej walidacji krzyżowej

Z racji jednak że w zbiorze testowym CASIA [9] ilość przedstawicieli klas nie jest taka sama - zdecydowałem się na stratyfikowaną wersję k-krotnej walidacji krzyżowej, oznacza to że podczas podziału na podzbiory operacja ta zachowuje oryginalny lub zbliżony do oryginalnego poziom niezbalansowania danych.

4.3. PAROWE TESTY STATYSTYCZNE

Do sprawdzenia czy po przeprowadzeniu eksperymentu na tych samych modelach ale np. różnych jądrach, różnice które otrzymałem w wartościach metryk są istotne statystycznie, skorzystałem z testu *T-Studenta* z poziomem ufności $\alpha = 0,05$ [2]. Obliczenia zostały przepro-

wadzone dla każdej z metryk, tj. dokładności, czułości, precyzji i miary *F*. Za hipotezę zerową przyjąłem, że pomiędzy klasyfikatorami nie ma istotnej różnicy statystycznej, co oznacza że kiedy wyliczona przez statystykę t, wartość $p - value$ będzie mniejsza lub równa α - odrzucę hipotezę zerową. W przeciwnym wypadku($\alpha > p - value$) przyjmę hipotezę zerową.

4.4. MASZYNA WEKTORÓW NOŚNYCH(*SVM*)

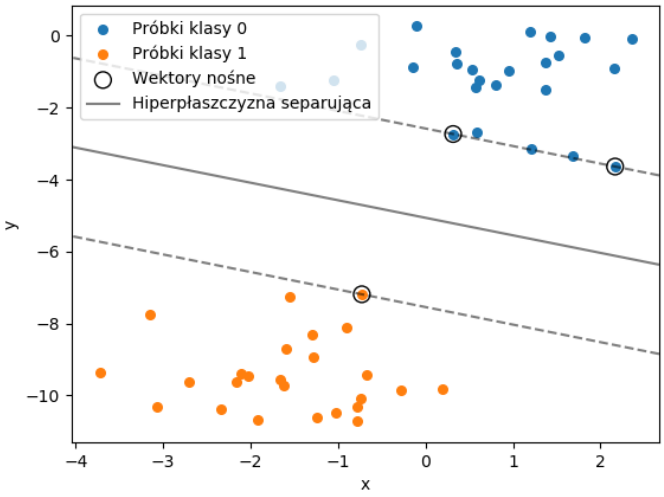
Jako bazowym klasyfikatorem binarnym posłużyłem się maszyną wektorów nośnych(ang. *Support Vector Machine - SVM*). Ideą tego algorytmu jest rzutowanie danej przestrzeni cech, na przestrzeń o większej ilości wymiarów. Taka operacja pozwala osiągnąć własność zwaną liniową separowalnością. Oznacza to nie mniej ni więcej, że istnieje możliwość rozdzielenia zbioru uczącego, tj. $(\vec{x}_1, y_1), ..., (\vec{x}_n, y_n)$ za pomocą hiperpłaszczyzny separującej, opisanej wzorem 4.1

$$\vec{w} * \vec{x} - b = 0 \tag{4.1}$$

, gdzie $\vec{w}$ jest wektorem normalnym dla tej płaszczyzny. Ponadto, wyliczane są wektory nośne, takie żeby spełniały własność opisaną wzorem 4.2

$$\begin{aligned} \vec{w} * \vec{x} - b &= 1 \\ \vec{w} * \vec{x} - b &= -1 \end{aligned} \tag{4.2}$$

Tym samym poszczególne próbki są rozdzielane ponad lub poniżej hiperpłaszczyzny separującej, co zostało zobrazowane na rysunku 4.4.

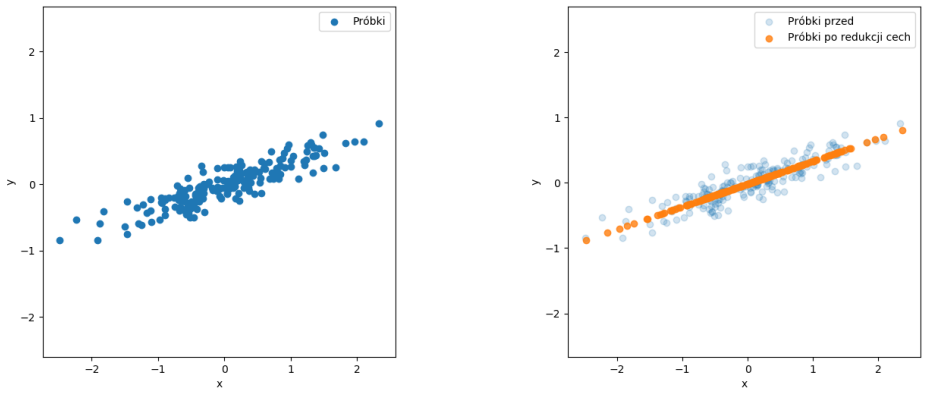


Rys. 4.4: Przykład klasyfikacji maszyny wektorów nośnych

4.5. ANALIZA GŁÓWNYCH SKŁADOWYCH

Dla uczącego zbioru danych Z_u, składającego się z n obserwacji, gdzie każda ma postać $k_1, ..., k_i$ można zdefiniować chmurę punktów, która im odpowiada. Oczywiście, mamy wtedy do czynienia z n punktów w przestrzeni i wymiarowej [20]. Na potrzeby przykładu można przyjąć że mamy do czynienia z $n = 50$ i $i = 2$. Celem analiza głównych składowych(z ang.*principal*

component analysis- PCA) jest taki obrót układu współrzędnych, aby maksymalizować wartość wariancji pierwszej współrzędnej, a następnie drugiej. Obrót ten, ma na celu skonstruowanie nowej przestrzeni, w której to początkowe współrzędne są najważniejsze. Graficzny przykład omawianej operacji widoczny jest na rysunku 4.5.



Rys. 4.5: Dane przed i po redukcji cech przy pomocy PCA(n_components=1)

4.6. VGG NET

Architektura splotowej sieci neuronowej VGG Net została zaproponowana na przełomie 2014 i 2015 roku, przez K. Simonyan i A. Zisserman [35]. Wykorzystywała ona 19 warstw oraz bardzo małe jak na te czasy filtry splotowe(3×3). Sam skok splotu, jak i jego wypełnienie(z ang. *padding*) wynosiło 1 piksel. Takie potraktowanie danych pozwoliło uzyskać mapy aktywacji posiadające takie same rozdzielczości co na wejściach warstw splotowych. Zabieg ten spowodował że wydłużanie łańcucha takich połączeń stało się bardzo proste.

Ważnym punktem tej architektury i towarzyszącej jej pracy naukowej [35] jest zauważenie przez autorów że moduł składający się z 3 warstw 3×3 ma efektywne pole postrzegania takie jak pojedyncza warstwa 7×7., przy używaniu dużo mniejszej liczby parametrów praz używaniu trzech nieliniowości zamiast jednej. Wniosek ten sprawił że w dalszych pracach w dziedzinie można zauważyć ewidentne zwrócenie się w stronę wykorzystywania większej ilości warstw z coraz mniejszymi wielkościami filtrów.

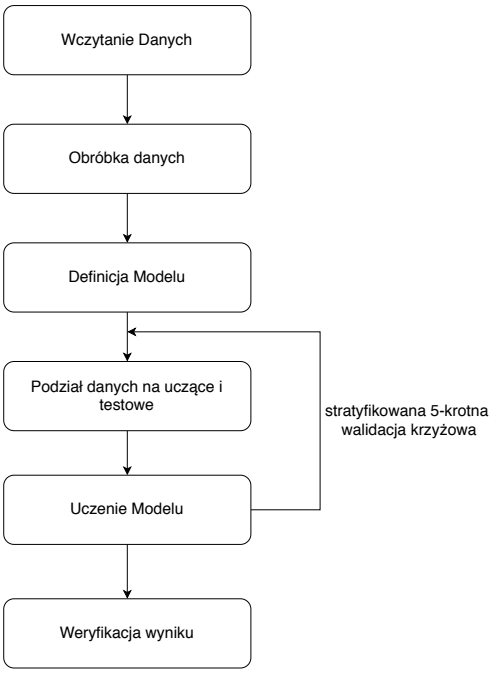
5. IMPLEMENTACJA I INTERPRETACJA WYNIKÓW BADAŃ

Do napisania pracy skorzystałem z języka skryptowego Python. Wynikało to z faktu że większość prac naukowych i badawczych jakie pisane są w obrębie dziedziny Uczenia Maszynowego jest tworzona właśnie w nim. Dodatkowo, stkorzystałem z szeregu otwarto-źródłowych rozwiązań, takich jak:

— *Keras*[4] - wysokopoziomowa biblioteka do trenowania sztucznych sieci neuronowych,
— *Sikit-Learn* [6] - biblioteka zawierająca zbiór algorytmów i narzędzi do walidacji wyników klasyfikacji,
— *NumPy* [29] - biblioteka służąca do wykonania obliczeń z zakresu analizy numerycznej i algebry liniowej,
— *Pandas* [26] - biblioteka służąca do manipulacji danych,
— *Matplotlib* [18] - biblioteka służąca do wizualizacji danych,
— *Jupyter notebook* [31] - środowisko do pracy, posiadające szereg ułatwień i możliwość kolejkowania zadań.

5.1. PROTOKÓŁ BADAWCZY

W niniejszej pracy przyjąłem następujący protokół badawczy - widoczny na rysunku 5.1. Wyróżnić można 3 główne etapy: przygotowanie danych i modelu(wczytanie danych, ich obróbka i definicja modelu), uczenie(Podział danych i same trenowanie modelu) oraz interpretacja i prezentacja wyniku.



Rys. 5.1: Protokół badawczy zastosowany w wykonanych eksperymentach

5.2. IMPLEMENTACJA ŚRODOWISKA WYKORZYSTUJĄCEGO MASZYNĘ WEKTORÓW NOŚNYCH

W pierwszej kolejności zdecydowałem się na wykorzystanie maszyny wektorów nośnych jako bazowego klasyfikatora. Idea ta jest spowodowana chęcią sprawdzenia jak klasyczne metody uczenia maszynowego sprawdzają się w porównaniu do podejścia splotowego, głębokiego.

5.2.1. Przygotowanie danych

```
for filename in A_folder_list:
  categories.append(0)
  filenames.append(dir_path + '/' + A_folder + '/' + filename)
for filename in B_folder_list:
  categories.append(1)
  filenames.append(dir_path + '/' + B_folder + '/' + filename)

df = pd.DataFrame({
  'filename': filenames,
  'category': categories
})
```

Listing 5.1: Stworzenie ramki danych **df**

Dane, a raczej pojedyncze zdjęcia zostają wczytane do Dataframe'u **df** 5.1, skąd to potem poddawane są procesowi ekstrakcji cech. Na sam proces składa 5 osobnych funkcji, z której każda z nich przetwarza oryginalne zdjęcie. Są to kolejno:

Momenty Hu

Jest to średnia ważona momentów zdjęcia opisana jako intensyfikacja pikselów w danym rejonie [17]. Pozwala to na zdefiniowanie translacji, skali i rotacji zdjęcia w odniesieniu do reszty. Opisane są wzorem 5.1, a sam kod widoczny jest na listingu 5.2

$$
\begin{aligned}
Hu_1 &= \eta_{20} + \eta_{02} \\
Hu_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
Hu_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{02})^2 \\
Hu_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{02})^2 \\
Hu_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} - \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] + \\
&\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] \\
Hu_6 &= (\eta_{20} - \eta_{02})\left[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
Hu_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] - \\
&\quad - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]
\end{aligned}
\tag{5.1}
$$

```
def ft_hu_moments(image):
  image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  hu_moments = cv2.HuMoments(cv2.moments(image)).flatten()
  hu_moments = scaler.fit_transform(hu_moments.reshape(-1, 1))
  hu_moments = hu_moments.flatten()
  return hu_moments
```

Listing 5.2: Kod momentów hu

Cechy Haralicka

Zaproponowany przez Roberta Haralicka w 1979 roku [13] zestaw cech pozuje na czarno-białej reprezentacji oryginalnego zdjęcia. Na jej podstawie generuje się tablicę współwystąpień oznaczaną GLCM(ang. *gray level co-occurence matrix*). Na jej podstawie wylicza się zestaw 13 cech opisanych następującymi wzorami 5.2 i kodem zawartym w listingu 5.3

$$
\begin{aligned}
f_1 &= \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} P(i,j)^2 & f_7 &= -\sum_{i=0}^{G-1}\sum_{j=0}^{G-1} P(i,j)\log(P(i,j)) \\
f_2 &= \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} P(i,j) * (i-j)^2 & f_8 &= \sqrt{\sum_{i=0}^{G-1}\sum_{j=0}^{G-1} P(i,j)^2} \\
f_3 &= \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} P(i,j) \left[\frac{(i-\mu_i)(j-\mu_j)}{\sqrt{(\delta_i^2)(\delta_j^2)}}\right] & f_9 &= \sum_{i=2}^{2N_g} i p_{x+y}(i) \\
f_4 &= \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} P(i,j) * |i-j| & f_{10} &= -\sum_{i=2}^{2N_g} p_{x+y}(i)\log(p_{x+y}(i)) \\
f_5 &= \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} \frac{P(i,j)}{1+(i+j)^2} & f_{11} &= \sum_{i=2}^{2N_g} (i-f_{10})^2 p_{x+y}(i) \\
f_6 &= \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} P(i,j) * (i-\mu)^2 & f_{12} &= Var\left[p_{x-y}\right] \\
 & & f_{13} &= -\sum_{i=0}^{N_g-1} p_{x-y}(i)\log(p_{x-y}(i))
\end{aligned}
\tag{5.2}
$$

```python
def ft_haralick(image):
  image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  haralick = mahotas.features.haralick(image).mean(axis=0)
  haralick = scaler.fit_transform(haralick.reshape(-1, 1))
  haralick = haralick.flatten()
  return haralick
```

Listing 5.3: kod przeliczania cechy Haralicka

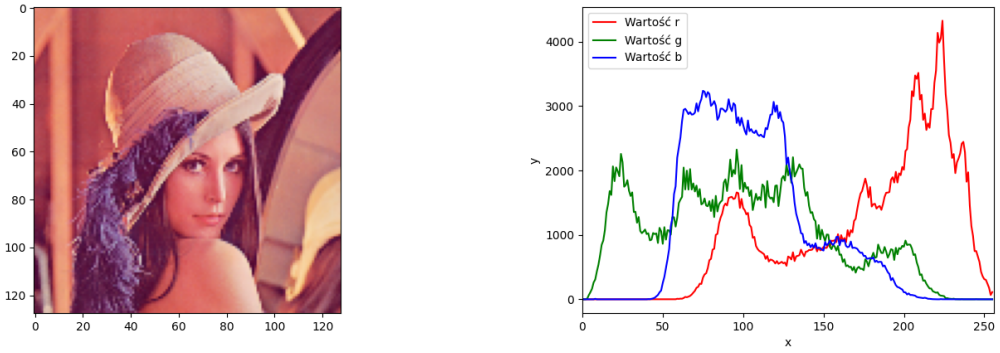
Histogram kolorów

Kolejną z rozpatrywanych cech jest histogram zdjęcia. Jest to prostu graficzny sposób przedstawiania rozkładu empirycznego cechy - dla zdjęcia będą to oczywiście rozkłady każdego z kolorów(r - dla czerwonego, g - dla zielonego i b - dla niebieskiego). Całość została graficznie pokazana na rysunku 5.2.

Jeśli chodzi o reprezentacje opisywanej cechy w postaci kodu, jest ona dostępna w listingu 5.4.

```python
def ft_histogram(image, mask=None):
  image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
  hist  = cv2.calcHist([image], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
  cv2.normalize(hist, hist)
  return hist.flatten()
```

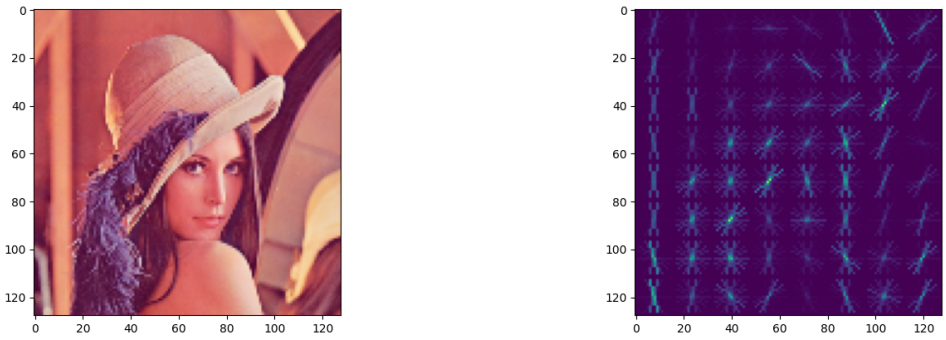
Listing 5.4: Histogram



Rys. 5.2: Zdjęcie i odpowiadający mu histogram

Histogram gradientów

Technika ta opracowana w 2005 roku [8] służy w większości do rozpoznawania sylwetek głównych obiektów na zdjęciach. Polega ona na przeglądaniu kwadratów na zdjęciu o wielkości 8×8 pikseli i staraniu się w nich o detekcje krawędzi, poprzez badanie wielkości różnic pomiędzy kolorami sąsiednich pikseli. Całość została graficznie przedstawiona na obrazku 5.3, a listing samej funkcji realizującej to zadanie jest widoczny 5.5.



Rys. 5.3: Zdjęcie oryginalne i po wykonaniu na nim histogramu gradientów

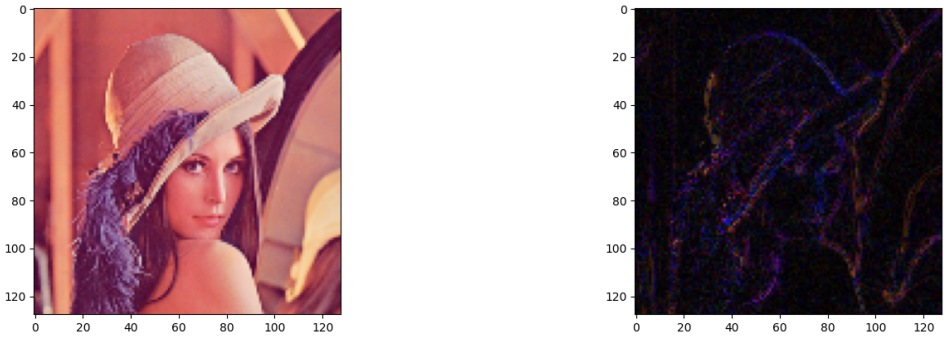
```
def ft_hog(image):
  image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  hog_features, hog_image = hog(image, block_norm='L2-Hys',
                                pixels_per_cell=(16, 16), cells_per_block=(1, 1),
                                visualize=True)
  return hog_features, hog_image
```

Listing 5.5: Histogram gradientów

Analiza poziomu błędu

Jest to funkcjonalność którą opisywałem już w rozdziale 3. Podsumowując korzystanie z algorytmu ELA(z ang. *Error Level Analysis*), opisanego w pracy N. Krawetza z 2007 roku[19] polega na wykorzystaniu właściwości zapisywania formatu *.JPEG w różnych jakościach. Ideą

algorytmu ELA, w wersji którą zaimplementowałem i wykorzystałem, jest zapisanie zdjęcia podwójnie z kompresją pomiędzy $\sim 85\%$ - $\sim 95\%$, a następnie *dodanie* różnic pomiędzy zdjęciem oryginalnym, a zdjęciem rozpatrywanym. Całość widoczna jest graficznie na rysunku 5.4, a kod został przedstawiony na listingu 5.6.



Rys. 5.4: Zdjęcie oryginalne i po wykonaniu na nim histogramu gradientów

```python
def ft_ela(image):
  image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
  ans = []
  for i in range(86, 96):
    im = Image.fromarray(image)
    im.save(f'tmp_{i}.jpg', 'JPEG', quality=i)
    resaved_im = Image.open(f'tmp_{i}.jpg')

    ela_im = ImageChops.difference(im, resaved_im)

    extrema = ela_im.getextrema()
    max_diff = max([ex[1] for ex in extrema])
    if max_diff == 0:
    max_diff = 1
    scale = 255.0 / max_diff
    ans.append(max_diff)

  ela_im = ImageEnhance.Brightness(ela_im).enhance(scale)
  ret = numpy.array([ans]).flatten() / 255

  return ret, ela_im
```

Listing 5.6: Analiza poziomu błędu

Podsumowując pojedyncze zdjęcie, o wielkości 128×128 po ekstrakcji cech przez opisane powyżej funkcję jest opisane przez zestaw 1118 cech. Udział poszczególnych cech widoczny jest w tabeli 5.1. Warto tutaj też dodać również ze opisane cechy są znormalizowane, tj. i zakres wynosi $[0, 1]$. Funkcję takie jak histogram kolorów, histogram gradientów i ELA normalizowane są niejako automatycznie(znany jest zakres możliwych do przyjęcia wartości), dla funkcji momentów hu i cech haralicka z racji że nie maja one zakresu rzuconych wyników, zastosowany został *MinMaxScaler*. Jego zasada jest dość prosta - znajduję w zadanym zakresie minimum i maksimum i resztę wartości w odniesieniu do tych.

	Ilość elementów	Procentowy udział w całości
Wielkość całego wektora	**1118**	**100%**
Momenty Hu	7	0.63%
Cechy Haralicka	13	1.16%
Histogram kolorów	512	45.80%
Histogram gradientów	576	51.52%
Analiza poziomu błędu	10	0.89%

Tabela 5.1: Udział poszczególnych funkcji ekstrakcji cech w konstrukcji wektora opisującego pojedyncze zdjęcie

Dodatkowo z racji dużej ilości danych opisujący pojedynczą próbkę, zastosowałem dekompozycję przy pomocy analizy głównych składowych [20], którą to dokładniej opisałem w rozdziale 2 niniejszej pracy. Redukcja PCA odbyła z 1118 elementów do 700, co daję redukcję o $\sim 37\%$.

5.2.2. Definicja i uczenie modeli

Na potrzeby maszyny wektorów nośnych przygotowane zostały 5 podstawowych wersji klasyfikatora SVM [12]: *SVM linear*, *SVM poly*, *SVM rbf*, *SVM sigmoid*(widoczne na listingu 5.7). Różnią się one tylko jądrem, a co za tym idzie - poszukiwaną funkcją na podstawie której budowana jest hiperpłaszczyzna separująca.

```
clfs = {
  "SVM linear": SVC(kernel='linear', probability=True, random_state=odp, verbose=True),
  "SVM poly": SVC(kernel='poly', probability=True, random_state=odp, verbose=True),
  "SVM rbf": SVC(kernel='rbf', probability=True, random_state=odp, verbose=True),
  "SVM sigmoid": SVC(kernel='sigmoid', probability=True, random_state=odp,verbose=True)
}
```

Listing 5.7: Możliwe klasyfikactory dla SVM'a

Same uczenie jest wykonywane zgodnie z zasadami 5-krotnej stratyfikowanej walidacji krzyżowej, opisanej szczegółowo w rozdziale 2 niniejszej pracy. Oprócz tego na każdym etapie liczone sa wszelkie zdefiniowane w rozdziale 2 miary określające prace modelu. Kod opisanych sytuacji widoczny jest na listingu 5.8.

```
for fold_id, (train_index, test_index) in enumerate(kf.split(features, labels)):
  for clf_idx, clf_name in enumerate(clfs):
    clf = clone(clfs[clf_name])
    clf.fit(features[train_index], labels[train_index])
    y_pred = clf.predict(features[test_index])

    accuracy, precision, recall, fscore = countStats(labels[test_index], y_pred)
    cm = confusion_matrix(labels[test_index], y_pred)
```

Listing 5.8: Trenowanie modelów

5.2.3. Interpretacja uzyskanych wyników

Poniżej przedstawiam wyniki uzyskane dla poszczególnych zbiorów danych jakie zostały użyte podczas pisania pracy.

Zbiór danych CASIA

Same wyniki są następujące 5.2. Podano wartość danej statystyki oraz dodatkowo odchylenie standardowe. Obie wartości sa liczone jako średnie z 5 wpisów:

Kernel	**Dokładność**	**Precyzja**	**Czułość**	**miara F**
linear	0.710 (0.01)	0.664 (0.01)	0.581 (0.01)	0.619 (0.01)
poly	0.647 (0.01)	0.618 (0.01)	0.343 (0.01)	0.441 (0.01)
rbf	0.725 (0.00)	0.669 (0.00)	0.638 (0.01)	0.653 (0.00)
sigmoid	0.662 (0.01)	0.593 (0.01)	0.538 (0.01)	0.564 (0.01)

Tabela 5.2: Wyniki szeregu miar w zależności od jądra, dla zbioru danych CASIA

Oprócz tego dla każdej z miar zostały wykonane parowe testy statystyczne w oparciu o statystykę t-studenta [2]. i cały szereg miar. Wyniki wyglądają następująco:

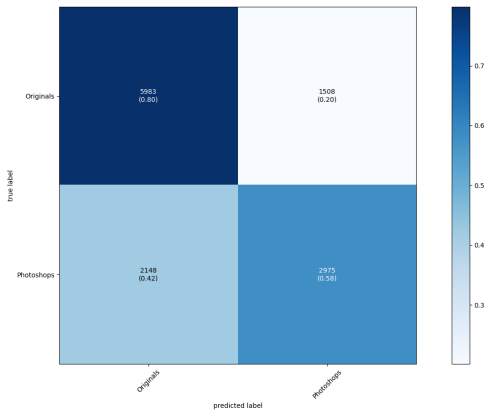
	Dokładność				**Precyzja**				**Czułość**				**miara F**			
	lin	**pol**	**rbf**	**sig**	**lin**	**pol**	**rbf**	**sig**	**lin**	**pol**	**rbf**	**sig**	**lin**	**pol**	**rbf**	**sig**
lin	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
pol	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
rbf	1	1	0	1	0	1	0	1	1	1	0	1	1	1	0	1
sig	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0

Tabela 5.3: Wynik działania dla wszystkich miar w oparciu o parowe testy statystyczne wykonane na zbiorze danych CASIA

Z analizy tablicy 5.3 wynika że najlepszy wynik dla zbioru danych CASIA osiągnął kernel **RBF**:

— Dokładność: **0.725 (0.00)**,
— Precyzja: **0.669 (0.00)**,
— Czułość: **0.638 (0.01)**,
— Miara F: **0.653 (0.00)**.

Tablica błędów prezentuję się następująco(rysunek 5.6):



Rys. 5.5: Tablica błędów dla SVM'a z jądrem RBF, trenowanego na zbiorze danych CASIA

Zbiór danych PS-Battles

Same wyniki są następujące 5.5. Podano wartość danej statystyki oraz dodatkowo odchylenie standardowe. Obie wartości sa liczone jako średnie z 5 wpisów:

Kernel	**Dokładność**	**Precyzja**	**Czułość**	**miara F**
linear	0.580 (0.01)	0.590 (0.01)	0.527 (0.01)	0.557 (0.01)
poly	0.482 (0.01)	0.468 (0.01)	0.264 (0.01)	0.337 (0.01)
rbf	0.529 (0.01)	0.530 (0.01)	0.504 (0.01)	0.517 (0.01)
sigmoid	0.541 (0.00)	0.543 (0.00)	0.527 (0.01)	0.535 (0.01)

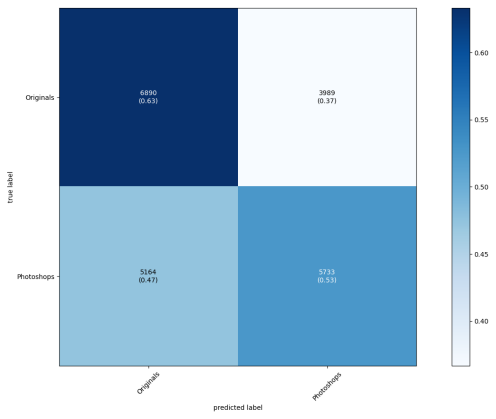
Tabela 5.4: Wyniki szeregu miar w zależności od jądra, dla zbioru danych PS-Battles

Oprócz tego dla każdej z miar zostały wykonane parowe testy statystyczne w oparciu o statystykę t-studenta [2]. i cały szereg miar. Wyniki wyglądają następująco:

	Dokładność				**Precyzja**				**Czułość**				**miara F**			
	lin	**pol**	**rbf**	**sig**	**lin**	**pol**	**rbf**	**sig**	**lin**	**pol**	**rbf**	**sig**	**lin**	**pol**	**rbf**	**sig**
lin	0	1	1	1	0	1	1	1	0	1	1	0	0	1	1	1
pol	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rbf	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
sig	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0

Tabela 5.5: Wynik działania dla wszystkich miar w oparciu o parowe testy statystyczne wykonane na zbiorze danych PS_Battles

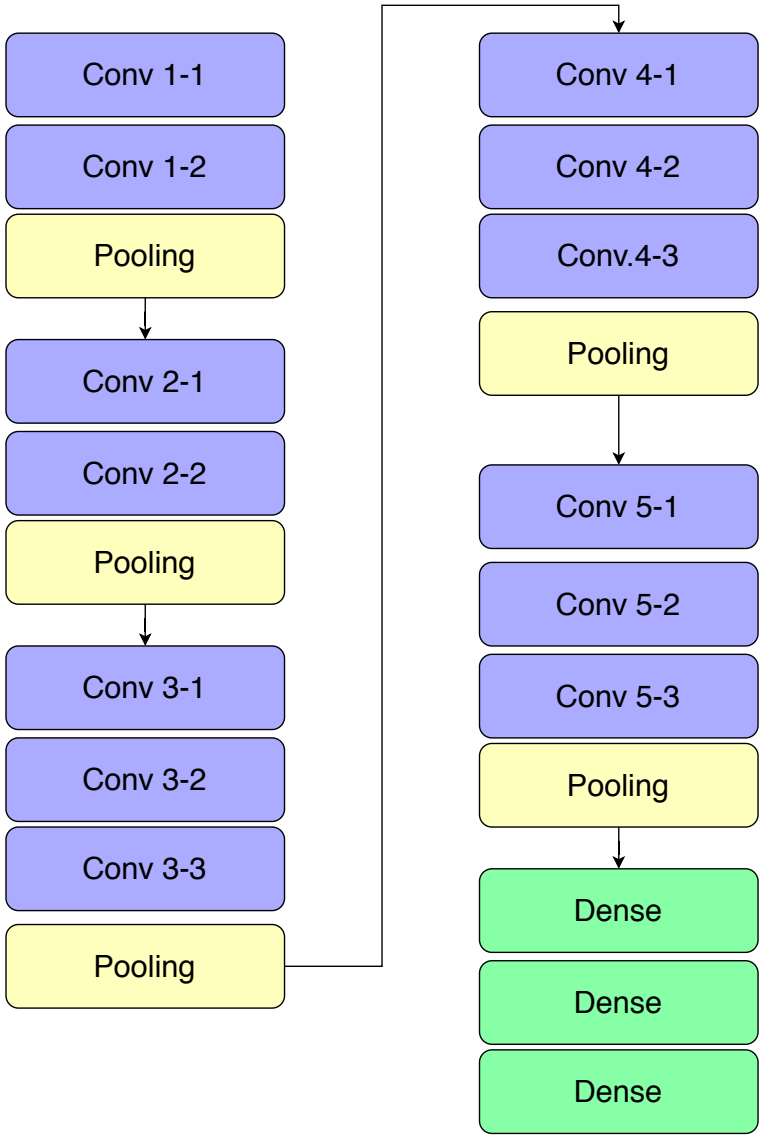
Z analizy tablicy 5.5 wynika że najlepszy wynik dla zbioru danych PS_Battles osiągnął kernel **linear**:
— Dokładność: **0.580 (0.00)**,
— Precyzja: **0.590 (0.00)**,
— Czułość: **0.527 (0.01)**,
— Miara F: **0.557 (0.00)**.
Tablica błędów prezentuję się następująco(rysunek 5.6):



Rys. 5.6: Tablica błędów dla SVM'a z jądrem RBF, trenowanego na zbiorze danych PS_Battles

5.3. IMPLEMENTACJA ŚRODOWISKA WYKORZYSTUJĄCEGO ISTNIEJĄCE ARCHITEKTURY SIECI

Do następnej części postanowiłem skorzystać z klasycznego modelu ucznia głębokiego, jakim jest VGG Net, wykorzystując w tym momencie technikę Transfer Learningu opisanego w rozdziale 2 . W dużym skrócie, polega ona na wykorzystaniu modelu o już nauczonych wagach, które to częściowo są zamrażane, przez co nie zmieniają się w procesie uczenia. Schemat modelu dostępny jest na rysunku 5.7.



Rys. 5.7: Schemat modelu VGG16

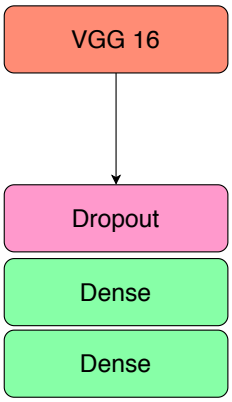
5.3.1. Przygotowanie danych

Dane w początkowej fazie swojej żywotności traktowane są dokładnie tak samo jak przy maszynie wektorów nośnych(zgodnie z listingiem 5.1). Tym razem jednak zamiast w dalszej części skupić się na kolejnych pięciu funkcjach odpowiedzialnych za ekstrakcje cech, na wejście sieci zawierającej VGG Net trafia wynik operacji algorytmy ELA na pojedynczym zdjęciu. Kod

tego rozwiązania dostępny jest na listingu 5.6 oraz w formie graficznej na rysunku 5.4. Warto tutaj zaznaczyć że wielkość zdjęcia została przyjęta na 150×150.

5.3.2. Definicja i uczenie modelu

Wykorzystałem w swojej pracy 16-warstwowy model VGG Netu - powodem było duża ilość parametrów, które wraz z kolejnymi wersjami(dodającymi kolejne warstwy), nie gwarantują dużo lepszych wyników [35]. Na jego podstawie zbudowałem model odpowiadający mojej ilości klas, który to składał się z warstwy VGG Netu, pojedynczej warstwy Dropout i dwóch warstw w pełni połączonych. Graficzną reprezentację zaproponowanego schematu można zobaczyć na rysunku 5.8.



Rys. 5.8: Schemat modelu zbudowanego o VGG16

Same uczenie opiera o szereg parametrów, tj. spodziewana ilość epok, funkcja aktywacji, czy funkcja optymalizująca wykorzystująca propagację wsteczną. I tak prezentują się one następująco 5.9:

```
epochs = 50
batch_size = 100
activation = 'relu'
loss_type = 'binary_crossentropy'
optimizer = RMSprop(lr=1e-5)
dropout = 0.25
```

Listing 5.9: Parametry pracy modelu VGG

Dodatkowo rónież skorzystałem z modułu odpowiedzi zwrotnych dostępnego w bibliotece *Keras* [4], który to pozwolił na zdefiniowanie trzech funkcji zarządzających przebiegiem uczenia. Są to:

— *EarlyStopping* - odpowiedzialny za zakończenie uczenia kiedy wartość testowa funkcji straty nie będzie wzrastać ani razu przez 10 epok. Pozwala to ograniczyć czas uczenia i zapobiega w pewnym zakresie występowaniu efektu przeuczenia modelu - patrz rozdział 2 niniejszej pracy,

— *ReduceLROnPlateau* - odpowiedzialny za monitorowanie wartości współczynnika nauki i w sytuacji kiedy wartość testowa funkcji straty nie zmaleje i zmniejszenie go o pewną wartość,

— *ModelCheckpoint* - zapis modelu, w sytuacji w której osiągnie najniższą wartość funkcji straty dla zbioru testowego.

I w listingu 5.10 prezentują się następująco:

```
EARLY_STOP_PATIENCE = 10
LEARNING_RATE_PATIENCE = 5

cb_early_stopper = EarlyStopping(monitor = 'val_loss', patience = EARLY_STOP_PATIENCE,
                                 verbose=0)
cb_checkpointer = ModelCheckpoint(filepath = 'best.h5', monitor = 'val_loss',
                                  save_best_only = True, verbose=0)
cb_learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss',
                               patience=LEARNING_RATE_PATIENCE,
                               verbose=0)
```

Listing 5.10: Funkcje sterujące procesem uczenia

5.3.3. Interpretacja uzyskanych wyników

Poniżej przedstawiam wyniki uzyskane dla poszczególnych zbiorów danych jakie zostały użyte podczas pisania pracy.

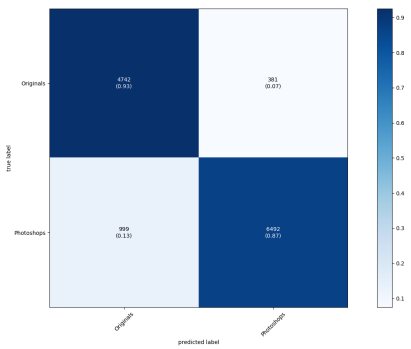
Zbiór danych CASIA

Same wyniki są następujące 5.6. Podano wartość danej statystyki oraz dodatkowo odchylenie standardowe. Obie wartości sa liczone jako średnie z 5 wpisów:

Kernel	**Dokładność**	**Precyzja**	**Czułość**	**miara F**
VGG	0.891 (0.01)	0.945 (0.01)	0.867 (0.01)	0.904 (0.01)

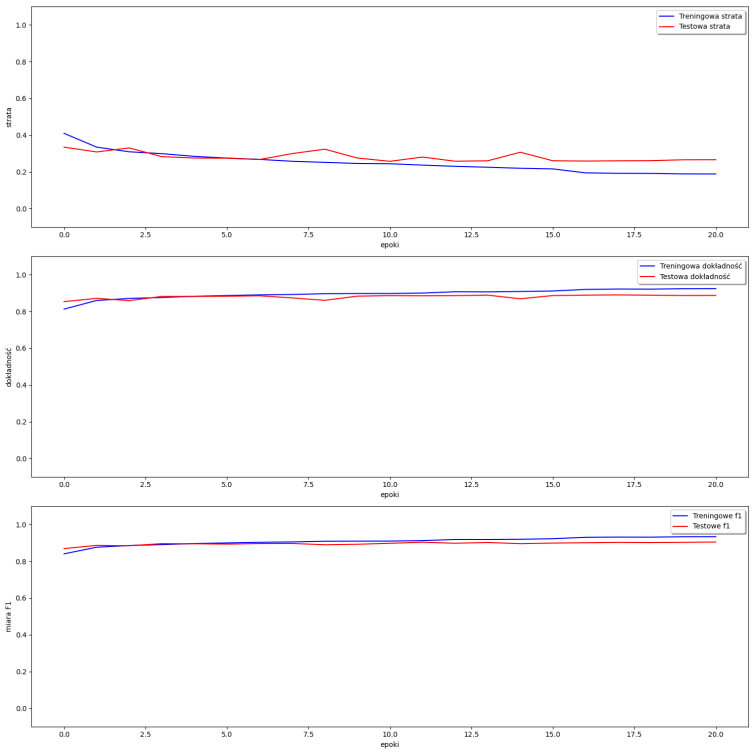
Tabela 5.6: Wyniki szeregu miar w zależności dla modelu opartego o VGG16, dla zbioru danych CASIA

Tablica błędów prezentuję się następująco(rysunek 5.9):



Rys. 5.9: Tablica błędów dla VGG16, trenowanego na zbiorze danych CASIA

Wartość funkcji uczących prezentuje się następująco(rysunek 5.10):



Rys. 5.10: Przebieg uczenia dla VGG16, trenowanego na zbiorze danych CASIA

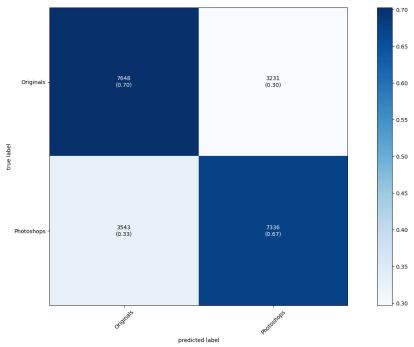
Zbiór danych PS_Battles

Same wyniki są następujące 5.7. Podano wartość danej statystyki oraz dodatkowo odchylenie standardowe. Obie wartości sa liczone jako średnie z 5 wpisów:

Kernel	**Dokładność**	**Precyzja**	**Czułość**	**miara F**
VGG	0.689 (0.01)	0.696 (0.01)	0.674 (0.02)	0.684 (0.00)

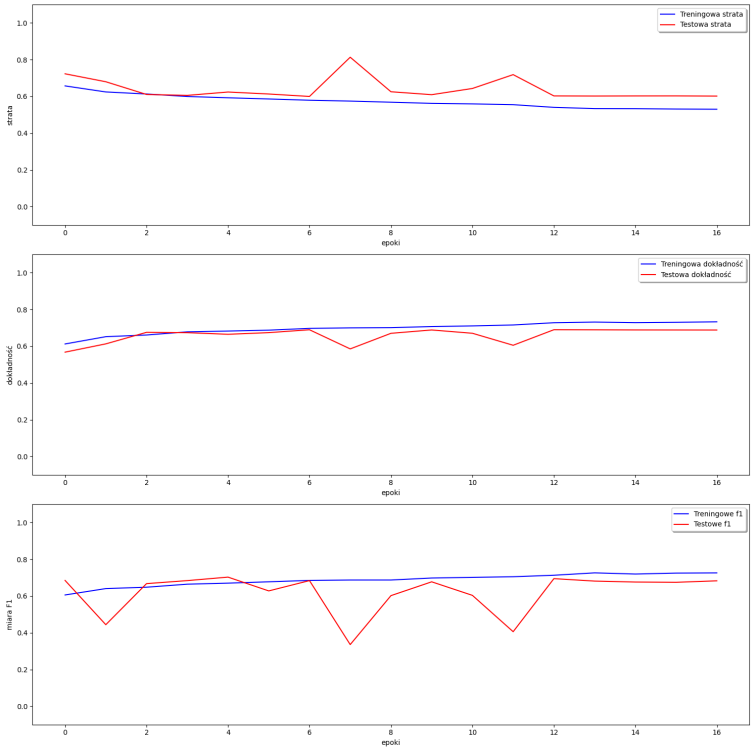
Tabela 5.7: Wyniki szeregu miar w zależności dla modelu opartego o VGG16, dla zbioru danych PS_Battles

Tablica błędów prezentuję się następująco(rysunek 5.11):



Rys. 5.11: Tablica błędów dla VGG16, trenowanego na zbiorze danych CASIA

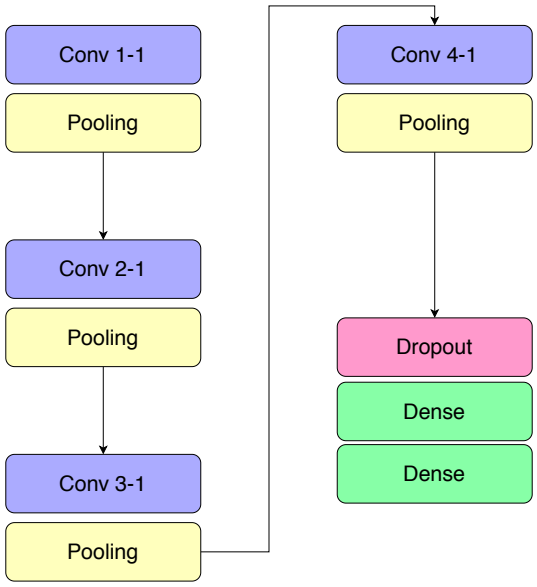
Wartość funkcji uczących prezentuje się następująco(rysunek 5.12):



Rys. 5.12: Przebieg uczenia dla VGG16, trenowanego na zbiorze danych PS_Battles

5.4. IMPLEMENTACJA AUTORSKIEJ METODY WYKORZYSTANIA UCZENIA GŁĘBOKIEGO W ZADANIU KLASYFIKACJI

Ostatnim klasyfikatorem na jaki się zdecydowałem jest autorska splotowa sieć neuronowa. Schemat używanego przez ze mnie modelu dostępny jest na rysunku 5.13



Rys. 5.13: Schemat autorskiego modelu CNN

5.4.1. Przygotowanie danych

Sam sposób przygotowania i obróbki danych jest bliźniaczy co na rozwiązaniu wykorzystującym sieć VGG Net, tj. nie jest wykonywana żadna ekstrakcja cech, a jedyne co jest robione to zmiana zdjęcia z oryginalnego na jego wersję przetworzoną przez algorytm ELA, którego kod dostępny jest na listingu 5.6, oraz w formacie graficznego przykładu na rysunku 5.4. Wielkość zdjęcia została ustalona na 128×128.

5.4.2. Definicja i uczenie modelu

Uczony model jest zgodny z jego graficzną reprezentacją na rysunku 5.13. Oczywiście podobnie jak w przypadku VGG Netu - mamy tu do czynienia z szeregiem parametrów sterujących, przedstawionych na listingu 5.11:

```
epochs = 50
batch_size = 100
activation = 'relu'
loss_type = 'binary_crossentropy'
optimizer = RMSprop(lr=1e-4)
dropout = 0.25
```

Listing 5.11: Parametry pracy modelu CNN

Idea trzech odpowiedzi zwrotnych sterujących procesem uczenia jest również tu obecna i bliźniacza jak do rozwiązania VGG Netowego, dostępnego na listningu 5.10.

5.4.3. Interpretacja uzyskanych wyników

Poniżej przedstawiam wyniki uzyskane dla poszczególnych zbiorów danych jakie zostały użyte podczas pisania pracy.

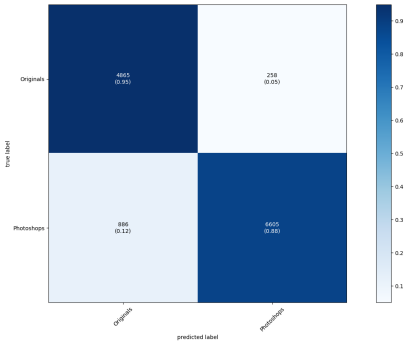
Zbiór danych CASIA

Same wyniki są następujące 5.8. Podano wartość danej statystyki oraz dodatkowo odchylenie standardowe. Obie wartości sa liczone jako średnie z 5 wpisów:

Kernel	**Dokładność**	**Precyzja**	**Czułość**	**miara F**
CNN	0.909 (0.01)	0.962 (0.01)	0.882 (0.01)	0.920 (0.01)

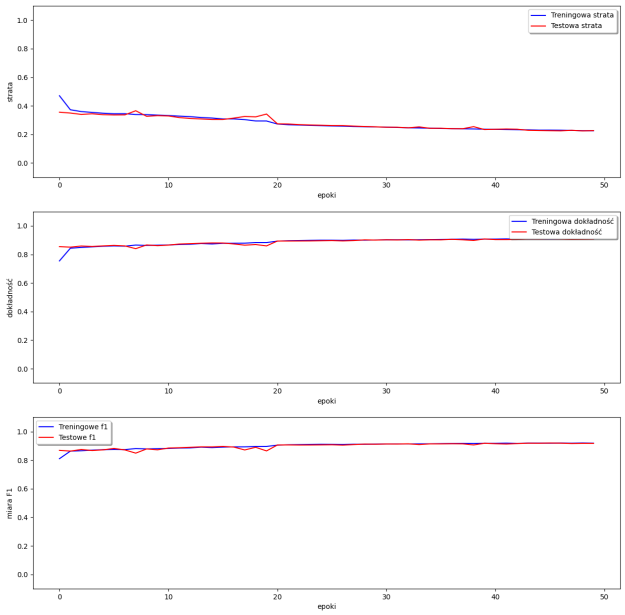
Tabela 5.8: Wyniki szeregu miar w zależności dla modelu opartego o autorskie rozwiązanie CNN, dla zbioru danych CASIA

Tablica błędów prezentuję się następująco(rysunek 5.14):



Rys. 5.14: Tablica błędów dla autorskiego rozwiązania CNN, trenowanego na zbiorze danych CASIA

Wartość funkcji uczących prezentuje się następująco(rysunek 5.15):



Rys. 5.15: Przebieg uczenia dla autorskiego rozwiązania CNN, trenowanego na zbiorze danych CASIA

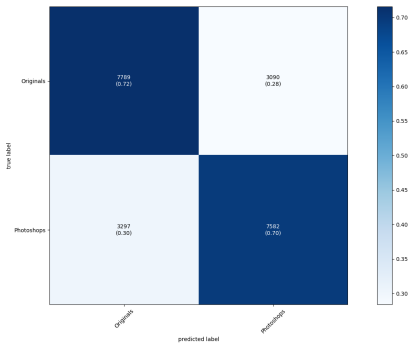
Zbiór danych PS_Battles

Same wyniki są następujące 5.9. Podano wartość danej statystyki oraz dodatkowo odchylenie standardowe. Obie wartości sa liczone jako średnie z 5 wpisów:

Kernel	Dokładność	Precyzja	Czułość	miara F
CNN	0.706 (0.01)	0.711 (0.01)	0.697 (0.01)	0.704 (0.01)

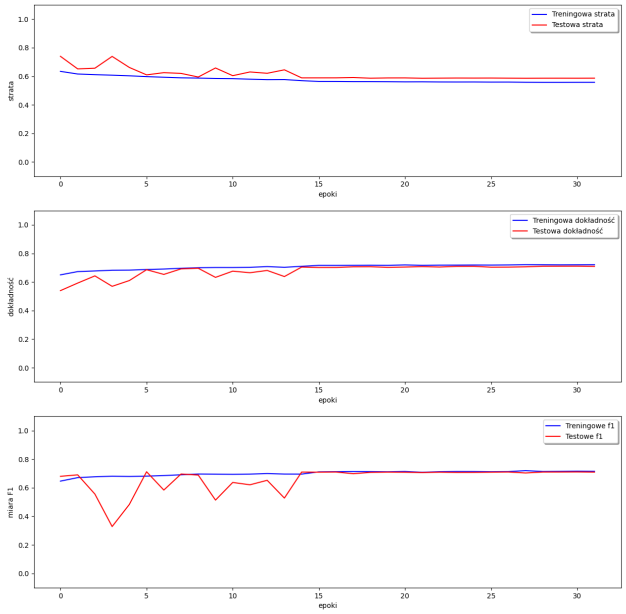
Tabela 5.9: Wyniki szeregu miar w zależności dla modelu opartego o autorskie rozwiązanie CNN, dla zbioru danych PS_Battles

Tablica błędów prezentuję się następująco(rysunek 5.16):



Rys. 5.16: Tablica błędów dla autorskiego rozwiązania CNN, trenowanego na zbiorze danych PS_Battles

Wartość funkcji uczących prezentuje się następująco(rysunek 5.17):



Rys. 5.17: Przebieg uczenia dla autorskiego rozwiązania CNN, trenowanego na zbiorze danych PS_Battles

W poniższej tabeli 5.10 przedstawiam rezultaty dla wszystkich mierzonych statystyk, wszystkich modeli i obu zbiorów danych:

	Kernel	**Dokładność**	**Precyzja**	**Czułość**	**Miara F**
CASIA	*SVM*	0.725 (0.00)	0.669 (0.00)	0.638 (0.01)	0.653 (0.00)
	VGG	0.891 (0.01)	0.945 (0.01)	0.867 (0.01)	0.904 (0.01)
	CNN	0.909 (0.01)	0.962 (0.01)	0.882 (0.01)	0.920 (0.01)
PS_Battles	*SVM*	0.580 (0.01)	0.590 (0.01)	0.527 (0.01)	0.557 (0.01)
	VGG	0.689 (0.01)	0.696 (0.02)	0.674 (0.02)	0.684 (0.00)
	CNN	0.706 (0.01)	0.711 (0.01)	0.697 (0.01)	0.704 (0.01)

Tabela 5.10: Zbiorcze wyniki wszystkich miar dla wszystkich policzonych modeli i zbiorów danych

Poniżej wyniki parowych testów statystycznych dla wszystkich modeli i miar. Tabela 5.11 przedstawia wyniki dla zbioru danych CASIA, a tabela 5.12 pokazuje wyniki dla zbioru danych PS_Battles.

	Dokładność			**Precyzja**			**Czułość**			**miara F**		
	S	**V**	**C**	**S**	**V**	**C**	**S**	**V**	**C**	**S**	**V**	**C**
SVM	0	0	0	0	0	0	0	0	0	0	0	0
VGG	1	0	0	1	0	0	1	0	0	1	0	0
CNN	1	1	0	1	1	0	1	0	0	1	1	0

Tabela 5.11: Wyniki parowego testu statystycznego dla zbioru danych Casia

	Dokładność			**Precyzja**			**Czułość**			**miara F**		
	S	**V**	**C**	**S**	**V**	**C**	**S**	**V**	**C**	**S**	**V**	**C**
SVM	0	0	0	0	0	0	0	0	0	0	0	0
VGG	1	0	0	1	0	0	1	0	0	1	0	0
CNN	1	1	0	1	0	0	1	0	0	1	1	0

Tabela 5.12: Wyniki parowego testu statystycznego dla zbioru danych PS_Battles

Jak widać najlepsze wyniki, które są statystycznie znaczące, osiągnęło moje autorskie rozwiązanie. Uzyskując kolejno $\sim 91\%$ dokładność dla zbioru CASIA i $\sim 71\%$ dokładności dla zbioru PS_Battles.

6. PODSUMOWANIE

BIBLIOGRAFIA

[1] Bappy, J.H., Roy-Chowdhury, A.K., Bunk, J., Nataraj, L., Manjunath, B., *Exploiting spatial structure for localizing manipulated image regions*, IEEE International Conference on Computer Vision. 2017.

[2] Billingsley, P., *Prawdopodobieństwo i miara* (Wydawnictwo Naukowe PWN, 2009).

[3] Chollet, F., *Deep Learning with Python* (Manning Publications, 2017).

[4] Chollet, F., *Dokumentacja techniczna biblioteki keras*. (dostęp 1.07.2020). `https://keras.io/`.

[5] Cichosz, P., *Systemy uczące się* (Wydawnictwo Naukowo-Techniczne, 2000).

[6] Cournapeau, D., *Dokumentacja techniczna biblioteki scikit-learn*. (dostęp 1.07.2020). `https://scikit-learn.org/`.

[7] Cozzolino, D., Poggi, G., Verdoliva, L., *Splicebuster: a new blind image splicing detector*, IEEE International Workshop on information forensics and security. 2015.

[8] Dalal, N., Triggs, B., *Histograms of oriented gradients for human detection* (IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005).

[9] Dong, J., Wang, W., Tan, T., *Casia image tampering detection evaluation database*, IEEE China Summit and International Conference on Signal and Information Processing. 2013.

[10] Farid, H., *Photo Forensics* (MIT Press, 2016).

[11] Goljan, M., Fridrich, J., *Cfa-aware features for steganalysis of color images*, Proceedings of SPIE - The International Society for Optical Engineering. 2015.

[12] Géron, A., *Hands-On Machine Learning with Scikit-Learn and TensorFlow* (O'Reilly Media, 2017).

[13] Haralick, R.M., *Statistical and structural approaches to texture*, Proceedings of the IEEE. 1979.

[14] Heller, S., Rossetto, L., Schuldt, H., *The ps-battles dataset - an image collection for image manipulation detection*, Computing Research Repository. 2018.

[15] Hinton, G., Salakhutdinov, R., *Reducing the dimensionality of data with neural networks*, Neural Computation. 2006.

[16] Holler, J., Tsiatsis, V., Mulligan, C., Avesand, S., Karnouskos, S., Boyle, D., *From machineto-machine to the internet of things: Introduction to a new age of intelligence*, Academic Press. 2014.

[17] Hu, M.K., *Visual pattern recognition by moment invariants*, IRE Transaction on information theory. 1962.

[18] Hunter, J.D., *Dokumentacja techniczna biblioteki matplotlib*. (dostęp 1.07.2020). `https://matplotlib.org/contents`.

[19] Krawetz, N., *A picture's worth...*, Hacker Factor Solutions. 2007.

[20] Krzanowski, W., *Principles of Multivariate Analysis* (OUP Oxford, 2000).

[21] Laney, D., *3d data management: Controlling data volume, velocity, and variety*, tech. rep., META Group. 2001.

[22] LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D., *Backpropagation applied to handwritten zip code recognition*, Neural Computation. 1989.

[23] Liu, Q., *Detection of misaligned cropping and recompression with the same quantization matrix and relevant forgery*, ACM workshop on Multimedia in forensics and intelligence. 2011.

[24] Lovelace, A., Menabrea, L.F., *Sketch of the analytical engine invented by charles babbage... with notes by the translator. translated by ada lovelace*, Scientific Memoirs. 1843.

[25] Mashey, J.R., *Big data... and the next wave of infrastress*, Slides from invited talk. 1998.

[26] McKinney, W., *Dokumentacja techniczna biblioteki pandas*. (dostęp 1.07.2020). `https://pandas.pydata.org/docs/`.

[27] Menabrea, L.F., *Sketch of the analytical engine invented by charles babbage*, Bibliothèque Universelle de Genève, No. 82. 1842.
[28] Mitchell, T.M., *Machine Learning* (McGraw-Hill, Inc., 1997).
[29] Oliphant, T., *Dokumentacja techniczna biblioteki numpy*. (dostęp 1.07.2020). `https://numpy.org/doc/stable/`.
[30] Piczak, K., *Klasyfikacja dźwięku za pomocą splotowych sieci neuronowych*, Prasa akademicka. 2018.
[31] Pérez, F., *Dokumentacja techniczna biblioteki jupyter notebook*. (dostęp 1.07.2020). `https://jupyter.org/documentation`.
[32] Raschka, S., *Python Machine Learning* (Packt Publishing, 2015).
[33] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., S, M., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L., *Imagenet large scale visual recognition challenge*, International Journal of Computer Vision. 2015.
[34] Samuel, A.L., *Some studies in machine learning using the game of checkers*, IBM Journal of Research and Development(Volume: 3, Issue: 3). 1959.
[35] Simonyan, K., Zisserman, A., *Very deep convolutional networks for large-scale image recognition*, Computing Research Repository. 2015.
[36] Singh, S., Singh, N., *Big data analytics*, Communication, Information Computing Technology (ICCICT). 2012.
[37] Swaminathan, A., Wu, M., Liu, K.J.R., *Digital image forensics via intrinsic fingerprints*, IEEE transactions on information forensics and security. 2008.
[38] Turing, A.M., *Computing machinery and inteligence*, Mind(Volume: LIX, Issue: 236). 1950.
[39] Zhang, Y., Goh, J., Win, L., Thing, V., *Image region forgery detection: A deep learning approach*, Proceedings of the Singapore Cyber-Security Conference. 2016.

SPIS RYSUNKÓW

SPIS TABEL

SPIS LISTINGÓW