

1 Spis treści

| | | |
|-------------------|--|-----------|
| <small>2</small> | 1 Fizyka | 2 |
| <small>3</small> | 1.1 Plazma kwarkowo-gluonowa | 2 |
| <small>4</small> | 1.2 Dżety | 2 |
| <small>5</small> | 1.3 Identyfikacja dżetów b | 3 |
| <small>6</small> | 1.4 Eksperyment ALICE | 3 |
| <small>7</small> | 2 Uczenie maszynowe | 4 |
| <small>8</small> | 2.1 Wzmacniane drzewa decyzyjne | 4 |
| <small>9</small> | 2.2 Sieci neuronowe | 5 |
| <small>10</small> | 2.3 Dyskusja użycia dwóch algorytmów | 10 |
| <small>11</small> | 3 Dane | 11 |

1 Fizyka

1.1 Plazma kwarkowo-gluonowa

Plazma kwarkowo-gluonowa (ang. *quark-gluon plasma* – *QGP*) to stan materii, który istniał w pierwszych ułamkach sekund po Wielkim Wybuchu. Przewiduje się, że materia w takim stanie obecna jest także w jądrach gwiazd neutronowych [1].

Obecnie aby uzyskać dostęp do materii w stanie plazmy kwarkowo-gluonowej potrzebne są wysokoenergetyczne zderzenia cząstek. Powszechnie mówi się o niej w kontekście zderzeń ciężkich jonów, chociaż istnieją także prace doszukujące się obecności *QGP* w mniejszych systemach np. w zderzeniach proton-proton [2], [3].

QGP jest stanem o ekstremalnej gęstości i temperaturze. Jego cechą charakterystyczną jest obecność wolnych kwarków i gluonów (zbiorczo nazywanych partonami). W każdym innym stanie materii są one zawsze związane i tworzą hadrony. Zjawisko to, zwane uwięzieniem koloru (ang. *color confinement*) uniemożliwia obserwację cząstek obdarzonych ładunkiem kolorowym (a takimi są kwarki i gluony) w stanie wolnym. Wolne kwarki i gluony powstające w zderzeniach muszą zatem przejść przez proces hadronizacji, w którym rekombinują one ze spontanicznie wytwarzanymi nowymi partonami, tworząc hadrony. W wyniku tego procesu, z każdego partonu obecnego w początkowym etapie zderzenia może powstać wiele cząstek poruszających się podobnym kierunkiem, tworząc stożek z wierzchołkiem blisko punktu interakcji wiązek. Taki stożek skolimowanych cząstek nazywany jest dżetem cząstek.

1.2 Dżety

Przedstawiona powyżej definicja dżetu nie jest precyzyjna z punktu widzenia pracy eksperymentalnej. W detektorze obserwuje się tylko cząstki w stanie końcowym, nie jest znana natomiast ich historia (tj. parton, z którego powstały), nie jest zatem możliwe przyporządkowanie cząstki według jej pochodzenia. W związku z tym, konieczne jest użycie algorytmu klasteryzującego, dostającego na wejściu tylko obserwowalne eksperymentalnie cząstki. To jakie dżety zostaną zaobserwowane w danym zdarzeniu zależy od użytego algorytmu. Oznacza to, że precyzyjną definicję dżetu stanowi algorytm klasteryzujący wraz z zestawem parametrów. Obecnie najpowszechniej stosowanym algorytmem jest algorytm *anti-kt* [4].

Eksperymentalne ograniczenia związane z obserwacją tylko końcowego stanu oddziaływań nie występują w analizie danych z symulacji Monte Carlo (MC), gdzie ma się dostęp do pełnej informacji na temat historii każdej cząstki. Można pomyśleć, że daje to możliwość lepszej klasteryzacji dżetów. Rekonstrukcja przy użyciu informacji o partonach - matkach sprawia jednak, że definicja dżetu wykorzystana w badaniach danych symulacyjnych jest inna od tej wykorzystywanej w eksperymencie. Tracona jest przez to cecha odpowiedniości między obiektami nazywanymi dżetami w symulacji i w eksperymencie, która to cecha jest niewątpliwie jedną z podstawowych wymagań stawianych przed dobrą symulacją.

Dżety stanowią ważny element w badaniach plazmy kwarkowo-gluonowej. Dają one pośredni wgląd we właściwości *QGP* na podstawie jej wpływu na oddziałujące z nią partony. Przykładową obserwabłą mierzoną w zderzeniach ciężkich jonów jest czynnik modyfikacji jądrowej (ang. *nuclear modification factor*), który jest miarą strat energii przez parton przechodzący przez medium.

Oprócz globalnego wpływu medium na dżety, analizuje się także różnice między dżetami pochodzącymi z gluonów oraz kwarków o różnych zapachach (ang. *flavours*). Modele teoretyczne przewidują między innymi większe straty energii w wyniku interakcji z *QGP* dla dżetów gluonowych niż kwarkowych [5] oraz zależność strat energii od masy partonu [6] – w tym przypadku

57 precyzyjne pomiary rozróżniające typy dżetów pozwalają lepiej zrozumieć mechanizm odpo-
58 wiadający za straty energii przez partony. Zagadnienie rozpoznania z jakiego rodzaju partonu
59 powstał dany dżet, nazywane jest identyfikacją lub tagowaniem dżetu.

60 1.3 Identyfikacja dżetów b

61 Poza badaniami właściwości QGP , szczególne znaczenie ma identyfikacja dżetów pochodzących
62 z ciężkich kwarków: b i c . Są one ważnym elementem w poszukiwaniu łamania symetrii CP w
63 rozpadach hadronów B i D oraz innych sygnatur tzw. *Nowej Fizyki* wykraczającej poza ramy
64 Modelu Standardowego. Kwarki *piękne* pojawiają się także często w kanałach rozpadu cząstek
65 takich jak bozon Higgsa i kwark t .

66 Identyfikacja dżetów b jest sporym wyzwaniem ze względu na zdecydowanie częściej wystę-
67 pujące dżety lekkie, tj. powstałe z hadronizacji kwarków u, d, s oraz gluonów. Rozpoznawanie
68 dżetów b bazuje na charakterystycznych właściwościach hadronów zawierających kwark piękny:
69 relatywnie długim czasie życia oraz (w mniejszym stopniu) na ich pół-leptonowym rozpadach
70 o względnej częstości rozpadu w tym kanale (ang. *branching ratio*) na poziomie 10%.

71 **przegląd algorytmów używanych w identyfikacji b-jetów: TBD**

72 1.4 Eksperyment ALICE

73 TBD

74 2 Uczenie maszynowe

75 Uczenie maszynowe jest bardzo szerokim i obecnie dynamicznie się rozwijającym obszarem
76 wiedzy. Występuje w wielu odmianach łącząc w sobie w zależności od wariantu wiele dziedzin
77 takich jak matematyka (statystyka, algebra) informatyka (algorytmika, teoria informacji) a
78 także elementy robotyki i sterowania. Dziedzinami, w których jest najczęściej wykorzystywane
79 są min. widzenie maszynowe, przetwarzanie języka naturalnego, autonomiczne roboty i pojazdy,
80 systemy decyzyjno - eksperckie, optymalizacyjne oraz rekomendacyjne.

81 W tej pracy wykorzystywana jest gałąź uczenia maszynowego nazywana uczeniem nadzoro-
82 wanym lub "uczeniem z nauczycielem" (ang. *supervised learning*), gdzie uczenie występuje na
83 podstawie poprawnie oznaczonych przykładów. Terminami bliskoznacznymi dla tak rozumia-
84 nego uczenia maszynowego są uczenie statystyczne (ang. *statistical learning*) i rozpoznawanie
85 wzorców (ang. *pattern recognition*).

86 Problem identyfikacji dźwięków jest klasycznym przykładem zagadnienia klasyfikacji, gdzie
87 poprawna odpowiedź jest jedną ze skończonej ilości opcji (klas) w przeciwieństwie do regresji,
88 gdzie szukana odpowiedź algorytmu ma charakter ciągły.

89 Występuje wiele algorytmów uczenia maszynowego takich jak regresja liniowa i logistyczna,
90 drzewa decyzyjne i ich wariacje, maszyny wektorów wspierających, sztuczne sieci neuronowe
91 oraz wiele innych. Uczenie polega na znalezieniu pewnej funkcji dopasowującej do przyjmowa-
92 nego na wejściu zestawu (wektora) cech (zmiennych, kolumn) pewną odpowiedź (predykcję),
93 która minimalizuje zadaną funkcję straty. Jej rolę w przypadku regresji często pełni błąd śred-
94 niokwadratowy a w przypadku klasyfikacji np. entropia krzyżowa (ang. *cross entropy*)¹. Różne
95 algorytmy szukają przy tym funkcji dopasowującej należącej do różnych klas funkcji: przy-
96 kładowo klasyczne drzewa decyzyjne przeszukują tylko przestrzeń funkcji dających się opisać
97 skończonym zbiorem reguł "jeśli – to" (ang. *if – else*).

98 W pracy wykorzystane zostały dwa rodzaje algorytmów: wzmacniane drzewa decyzyjne oraz
99 sieci neuronowe.

100 2.1 Wzmacniane drzewa decyzyjne

101 Wzmacniane drzewa decyzyjne są jednym z rozwinięć klasycznego algorytmu drzewa decyzyj-
102 nego. Pojedyncze drzewo decyzyjne dzieli przestrzeń cech uczących przy pomocy prostopadłych
103 cięć, na mniejsze/większe niż zadana wartość w przypadku zmiennej ciągłej lub na należące/nie
104 należące do danej klasy w przypadku zmiennej kategorycznej. Każdy podział, nazywany wę-
105 złem, daje dwie gałęzie, które można dalej niezależnie dzielić aż do ostatniego poziomu (liści).
106 Kolejne podziały wybierane są tak, aby zbiory przykładów wpadające do poszczególnych gałęzi
107 były jak najbardziej jednorodne. Stosuje się różne miary nieporządku takie jak: *indeks Gini*
108 lub entropia.

109 Drzewa decyzyjne są często łączone w komitety klasyfikatorów (ang. *ensemble methods*).
110 Wiele "słabych" klasyfikatorów jest łączonych w jeden "silny" na dwa sposoby: *bagging* oraz
111 *boosting* (wzmacnianie), które są często ze sobą porównywane.

112 *Bagging* – w zastosowaniu dla drzew decyzyjnych nazywany algorytmem lasów losowych
113 (ang. *random forest*) - polega na wytrenowaniu wielu drzew, każdego na podstawie N przykła-
114 dów losowo wylosowanych z powtórzeniami spośród N -licznego zbioru treningowego. Dodatko-
115 wo, do uczenia każdego drzewa używa się tylko podzbioru wszystkich cech uczących. Końcową
116 predykcję algorytmu otrzymuje się poprzez "głosowanie" wszystkich drzew z odpowiednimi

¹ $J = -\frac{1}{N} \sum_{i=1}^N [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)]$, gdzie y_i to prawidłowa klasa i -tego przykładu a \hat{y}_i to predykcja algorytmu

117 wagami.

118 *Boosting* – wzmacniane drzewa decyzyjne (ang. *boosted decision trees*) – jest metodą po-
119 dobłą do *baggingu*. Główną różnicą jest zwiększanie wag przykładom uczącym, które przez po-
120 przednie drzewo zostały źle zaklasyfikowane – każde kolejne drzewo koncentruje się bardziej na
121 poprawie błędów poprzednich drzew. Widać tu kolejną ważną cechę odróżniającą obie metody:
122 *boosting* jest algorytmem sekwencyjnym podczas gdy *bagging* daje się trywialnie zrównoleglić
123 (każde drzewo trenowane jest w osobnym wątku).

124 Parametry i sposób trenowania drzew decyzyjnych na analizowanych danych

125 W niniejszej pracy wykorzystano wzmacniane drzewa decyzyjne zaimplementowane w wydaj-
126 nej bibliotece **XGBoost** [7]. Szybkość obliczeń jest bardzo ważna, gdyż oprócz komfortu pracy
127 z algorytmem, przekłada się na jakość otrzymanych wyników – krótszy czas obliczeń oznacza
128 możliwość przeprowadzenia większej ilości eksperymentów i lepsze dobranie parametrów oraz
129 danych. Implementacja wzmacnianych drzew decyzyjnych w **XGBoost** wykorzystuje wszystkie
130 rdzenie procesora, pomimo że sam algorytm ma charakter sekwencyjny – jest to możliwe dzięki
131 paralelizacji procesu tworzenia każdego drzewa (przed każdym podziałem konieczne jest sprawd-
132 zenie pewnej ilości możliwych zmiennych i wartości progowych i ten proces jest wykonywany
133 równolegle).

134 Dzięki szybkiemu uczeniu się algorytmu, możliwe było użycie kosztownego obliczeniowo au-
135 tomatycznego przeszukiwania przestrzeni parametrów przy pomocy przeszukiwania losowego
136 (ang. *random search*), które jest zwykle preferowane nad przeszukiwanie sieciowe [8]. W tym
137 celu cały zbiór danych dzielony był na dwie części: trenującą oraz testową (80/20%). Następ-
138 nie algorytm był trenowany i oceniany z użyciem trzy- lub pięciokrotnej walidacji krzyżowej
139 (ang. *cross-validation*) na zbiorze trenującym dla różnych zestawów parametrów. Model z naj-
140 lepszym wynikiem uzyskanym w walidacji krzyżowej był sprawdzany na zbiorze testowym.

141 Parametry optymalizowane w opisanym procesie to:

- 142 • *max_depth* – maksymalna głębokość każdego drzewa (niekoniecznie osiągnięta)
- 143 • *n_estimators* – liczba drzew
- 144 • *learning_rate* – parametr szybkości uczenia, komplementarny do *n_estimators*, w prak-
145 tyce można ustalić liczbę drzew i szukać optymalnej szybkości uczenia
- 146 • *subsample*, *colsample_bytree*, *colsample_bylevel* – parametry regularyzacyjne określają-
147 ce ułamek kolejno: danych użytych do trenowania każdego drzewa, kolumn użytych w
148 każdym drzewie (cechy losowane raz dla danego drzewa), kolumn użytych przy każdym
149 podziale (cechy losowane przy każdym podziale)
- 150 • γ – minimalny zysk w postaci zmniejszenia wartości funkcji straty konieczny do wykonania
151 podziału

152 2.2 Sieci neuronowe

153 Sieci neuronowe (ang. *neural networks* – *NN*) są szczególnym algorytmem uczenia maszyno-
154 wego. Występują w bardzo wielu odmianach i są wykorzystywane w rozwiązywaniu szerokiej
155 gamy problemów. Nawet bardzo pobieżny opis sieci neuronowych wymaga dużo więcej miejsca
156 niż może być temu poświęcone w tej pracy. Wprowadzenia do sieci neuronowych na różnym
157 poziomie zaawansowania można znaleźć m.in. w [REF]x100. Tu przedstawione zostaną wy-
158 łącznie wybrane zagadnienia mające ściślejszy związek z pracą. Używane mogą być terminy,
159 których znaczenie wyjaśniane jest w podanych źródłach.

160 W niniejszej pracy, wykorzystane zostały dwa rodzaje sieci neuronowych: sieci w pełni połą-
161 czone (ang. *fully connected NN – FC NN*), nazywane także wielowarstwowymi perceptronami
162 (ang. *multi-layer perceptron – MLP*) oraz sieci konwolucyjne (ang. *convolutional NN – Co-
163 nvNets, CNN*).

164 Sieci w pełni połączone

165 W nierekurencyjnych sieciach neuronowych (tylko takie są używane w tej pracy), informacja
166 jest przekazywana kolejno od warstw wejściowych, poprzez warstwy ukryte aż do wyjściowej. W
167 sieciach typu *FC* wszystkie warstwy składają się z identycznych neuronów – każdy neuron do-
168 staje na wejściu wektor, natomiast zwraca skalar – wartość pewnej zadanej, nieliniowej funkcji,
169 jako argument podając średnią ważoną z elementów wektora wejściowego. Wartości zwraca-
170 ne przez neurony w danej warstwie składają się na wektor wejściowy dla neuronów kolejnej
171 warstwy. Wejściami dla pierwszej warstwy są natomiast kolejne przykłady ze zbioru uczącego.
172 Trenowanie sieci neuronowych polega na zmienianiu wag (parametrów) w liczonej w każdym
173 neuronie średniej, każdy neuron posiada własny, niezależny zestaw wag.

174 Istnieje twierdzenie o sieciach neuronowych jako uniwersalnych aproksymatorach funkcji
175 (ang. *universal approximation theorem*) [9], mówiące, że już sieć neuronowa o jednej warstwie
176 ukrytej jest zdolna do przybliżenia dowolnej funkcji z dowolną dokładnością. Twierdzenie to
177 nie podaje niestety liczby potrzebnych neuronów a przede wszystkim – sposobu ich trenowania.
178 Trenowanie jest prostsze w przypadku zastosowania wielu warstw, które odpowiadają kolejnym
179 poziomą abstrakcji jednak nadal jest dużym wyzwaniem ze względu na fakt, że nawet stosun-
180 kowo niewielka sieć może posiadać bardzo dużą liczbę parametrów, przykładowo sieć o czterech
181 warstwach, w każdej po 128 neuronów ma ich ponad 65 tysięcy.

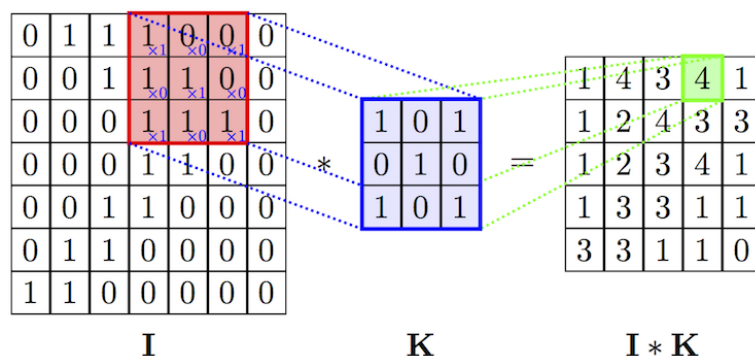
182 Sieci konwolucyjne

183 Jednym ze sposobów na ograniczenie liczby trenowanych parametrów jest użycie konwolucyj-
184 nych sieci neuronowych (bardziej poprawną choć rzadko używaną nazwą w języku polskim jest
185 sieć splotowa). Są one inspirowane połączeniami w korze wzrokowej zwierząt i wywodzą się z ba-
186 dań w obszarze widzenia komputerowego, gdzie liczby parametrów są szczególnie duże (wektor
187 wejściowy ma wymiar równy liczbie pikseli w obrazie), na takim przykładzie również najłatwiej
188 zrozumieć ich działanie.

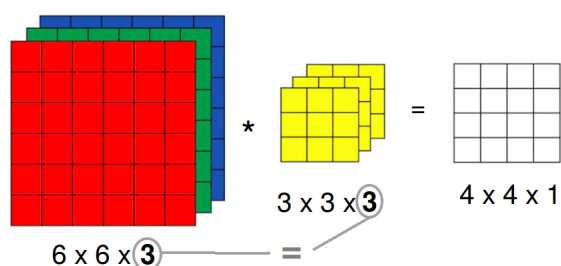
189 Sieci konwolucyjne różnią się od sieci typu *FC* tym, że część wag połączeń między warstwami
190 jest dzielona. Występuje w nich nowy rodzaj warstwy, nazywany warstwą konwolucyjną. Każda
191 jednostka w warstwie konwolucyjnej (filtr) ma pewną stałą (niewielką) liczbę wag. Połączenie z
192 dużym wejściem realizowane jest przez powielanie tych samych wag w połączeniach z kolejnymi
193 fragmentami wektora wejściowego (por. Rys. 1). Rezultatem działania filtra na macierz jest wy-
194 nik operacji splotu. Liczba parametrów przypadająca na każdy filtr jest równa jego rozmiarowi
195 i nie zależy od wielkości wektora wejściowego.

196 W przypadku gdy zamiast wejścia dwuwymiarowego (jak np. obraz czarno-biały), mamy do
197 czynienia z wejściem trójwymiarowym (np. trzeci wymiar to kolejne kolory w kodowaniu RGB),
198 filtry również muszą mieć trzy wymiary, przy czym rozmiar w ostatnim wymiarze musi być
199 równy rozmiarowi w tym kierunku wektora wejściowego. Wynik operacji splotu jest ponownie
200 dwuwymiarowy, gdyż filtr przesuwany jest tylko w dwóch pierwszych wymiarach. Trzeci wymiar
201 powstaje przez składanie kolejnych filtrów. Widać zatem, że również w przypadku gdy na
202 wejścia podawany jest obraz czarno-biały, filtry w kolejnych warstwach konwolucyjnych (oprócz
203 pierwszej) mają po trzy wymiary.

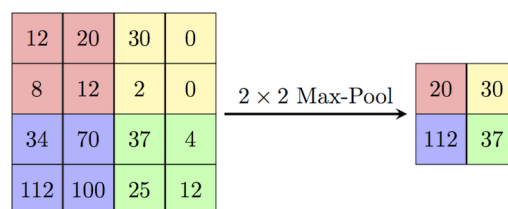
204 Oprócz warstw konwolucyjnych, w sieciach tego typu stosowane są także tzw. warstwy typu
205 *max-pooling*. Zasada jej działania jest bardzo prosta: wykonuje funkcję *maksimum* na zadanym



Rysunek 1: Schemat działania pojedynczego filtra z warstwy konwolucyjnej (operacja splotu).



Rysunek 2: Działanie pojedynczego filtra (3D) na wejście o trzech wymiarach.



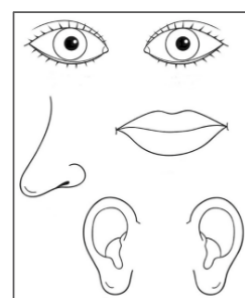
Rysunek 3: Działanie warstwy typu *max-pool*

206 fragmencie obrazu (por. Rys. 3). Ich rolą jest zmniejszanie rozmiaru przekazywanej w sieci
207 informacji.

208 Typowa architektura stosowana w przypadku sieci konwolucyjnych jest następująca: naj-
209 pierw warstwy konwolucyjne (pomiędzy nimi czasem warstwy typu *max-pool*), następnie wszyst-
210 kie filtry są rozwijane i składane w długi jednowymiarowy wektor, który przekazywany jest do
211 warstw typu *FC*. W przypadku problemu klasyfikacji, na końcu znajduje się jeszcze warstwa
212 typu *softmax* normalizująca wyjście z sieci do jedynki (por. Rys. 4).

213 Sieci konwolucyjne posiadają dwie właściwości odróżniające je od
214 *MLP*:

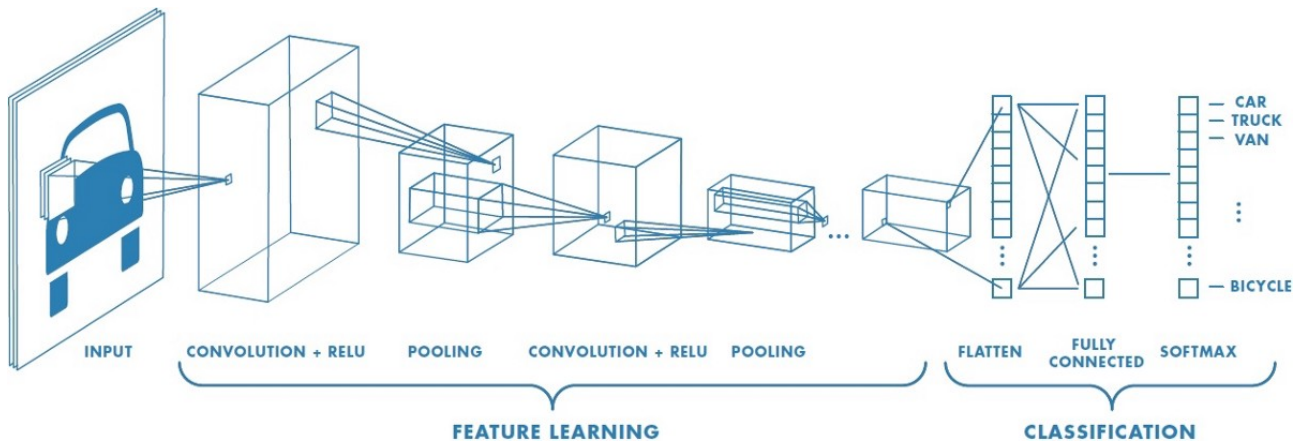
- 215 • niezmienniczość względem przesunięcia (ang. *translation invariance*) – głównie za sprawą dzielenia wag oraz obecności warstw
216 typu *max-pool*, położenie danej cechy na obrazie jest niemal
217 bez znaczenia (obraz po prawej stronie byłby rozpoznany ja-
218 ko twarz)
- 220 • lokalność połączeń – filtry obejmują tylko kilka sąsiednich pikseli
221 (tam zwykle występują najsilniejsze zależności) nie są w stanie
222 dostrzec cechy rozciągniętej na obszar większy od rozmiaru filtra



223 Hiperparametry i trenowanie sieci neuronowych na analizowanych danych

224 Parametry sieci, których wartości są określane przez projektanta sieci, takie jak liczba warstw
225 ukrytych są nazywane hiperparametrami (dla odróżnienia od parametrów - wag połączeń).

226 Testowane były trzy architektury: sieci w pełni połączone, sieci konwolucyjne oraz sieć
227 złożona z dwóch gałęzi, osobnych dla wtórnych wierzchołków i cząstek tworzących dżet (por.
228 Rozdz. 3) przedstawione schematycznie na Rys. 5. Do trenowania sieci wykorzystano wysoko-

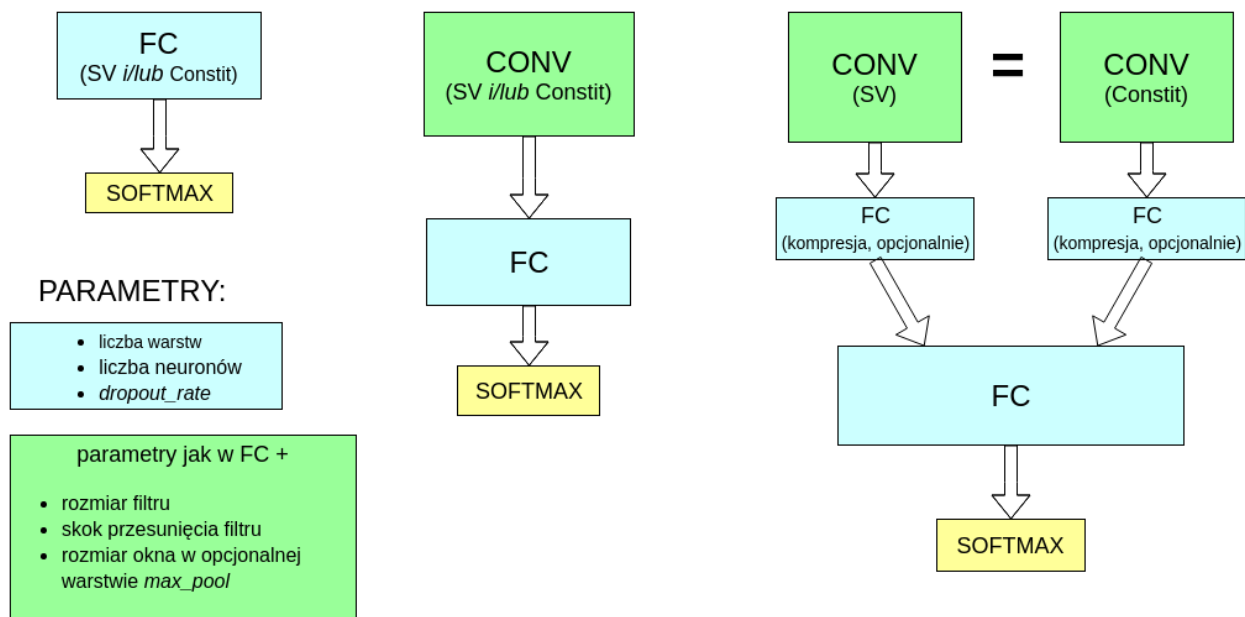


Rysunek 4: Typowa struktura stosowana w sieciach konwolucyjnych. Warstwy konwolucyjne mają za zadanie wydobywać cechy, na podstawie których późniejsze warstwy dokonują klasyfikacji. Widoczna jest charakterystyczna stopniowa zmiana rozmiaru przekazywanej macierzy: rozmiar poprzeczny maleje kosztem głębokości, co odpowiada rosnącej liczbie filtrów i malejącemu rozmiarowi obszaru po jakim są one przesuwane.

poziomową bibliotekę `Keras` [10] korzystającą z silnika obliczeniowego zaimplementowanego w `TensorFlow` [11].

Zestaw hiperparametrów definiujący działanie sieci w pełni połączonej:

- liczba warstw ukrytych
- liczba neuronów w każdej warstwie
- funkcja aktywacji – nieliniowa funkcja aplikowana przed zwróceniem wartości w każdym neuronie, najpopularniejsze to \tanh , $ReLU$ ($f(x) = \max(0, x)$) oraz funkcja sigmoidalna ($f(x) = \frac{1}{1+\exp(x)}$)
- algorytm optymalizacyjny – spadek gradientowy lub jego wariacje
- parametr szybkości uczenia i jego modyfikacje w trakcie uczenia
- liczba przykładów trenujących przetwarzanych w jednym kroku uczenia (ang. *batch_size*) – im większy tym szybsze jest trenowanie sieci (dzięki wydajnym operacjom macierzowym), natomiast może się to odbywać kosztem precyzji
- liczba epok uczenia – ile razy będzie pokazywany sieci każdy przykład
- opcjonalnie: warunki stopu (ang. *early stopping*) w razie osiągnięcia *plateau* (wysycenia procesu uczenia)
- opcjonalnie: regularyzacja przy pomocy różnych technik (zwykle konieczna)
- ponadto dla sieci konwolucyjnych:
 - liczba warstw konwolucyjnych i liczba filtrów w każdej warstwie
 - obecność lub brak warstw *max-pool* i rozmiar ich okna
 - rozmiar filtrów i długość skoku przy ich przesuwaniu



Rysunek 5: Schematyczne przedstawienie trzech testowanych rodzin architektur sieci. Każdy blok odpowiada kilku warstwom danego typu. Bloki warstw typu *FC* i opisane jako "kompresja" składały się z kilku neuronów i miały za zadanie zredukować całą informację z danej gałęzi do wektora kilku liczb. Obie gałęzie miały taką samą strukturę.

Same dwie pierwsze wielkości dają nieograniczoną liczbę konfiguracji. Czas trenowania sieci neuronowych jest rzędu wielkości większy niż drzew decyzyjnych, dlatego przyjęto szereg kroków mających na celu zmniejszenie przeszukiwanej przestrzeni hiperparametrów. Na podstawie wstępnych testów oraz różnych wskazówek dostępnych w literaturze przyjęto:

- *batch_size* zawsze równy 64 (inne testowane wartości: 16, 32, 128)
- za algorytm optymalizacyjny przyjęto algorytm o nazwie *Nadam* [12], tj. rozwinięcie algorytmu *Adam* [13] o parametr Nesterova (inne testowane to zwykły spadek gradientowy oraz *Adam*)
- funkcję aktywacji: *ReLU*
- liczba epok równa 100 lub 200 (lub mniej do testów), zrezygnowano z *early stopping*
- stałe w trakcie treningu wartości parametru szybkości uczenia
- spośród technik regularyzacyjnych testowano wyłącznie *dropout* [14] z parametrami 0.1, 0.2, 0.5
- kilka wybranych kombinacji dla zestawu parametrów: rozmiar filtra, długość skoku i rozmiar okna w warstwach *max-pool* – takie same w kolejnych warstwach
- liczby neuronów/filtrów w warstwach będące zawsze potęgami dwójki oraz stałą liczbę w kolejnych warstwach lub zmieniającą się o stały czynnik, np. 256-128-64, 128-128-128 lub 16-32-64
- liczba warstw *FC*: 2-8, konwolucyjnych 2-6

269 Nawet po przyjęciu powyższych uproszczeń nie sposób sprawdzić wszystkich możliwych
270 zestawów hiperparametrów, dlatego sposób ich dobierania w kolejnych testach był mocno em-
271 piryczny. Dostępne dane dzielone były na trzy zbiory: trenujący, walidacyjny i testowy. Wobec
272 braku warunków stopu, zbiór walidacyjny użyty był wyłącznie do porównywania różnych ze-
273 stawów parametrów, tak aby wynik testowy pozostał nieobciążony.

274 Zgodnie z zasadą ortogonalizacji działań, proces doboru hiperparametrów dzielono na dwie
275 części: najpierw starano się uzyskać jak najlepsze wyniki na zbiorze uczącym, a dopiero później
276 zmusić algorytm do lepszej generalizacji na zbiorze testowym przez zwiększoną regularyzację i
277 modyfikację parametru szybkości uczenia.

278 2.3 Dyskusja użycia dwóch algorytmów

279 Użycie więcej niż jednego algorytmu ma wiele zalet. Po pierwsze daje możliwość porównania
280 wyników. Pozwala to na oszacowanie błędu *Bayesowskiego* (najniższego możliwego do osiągnię-
281 cia przez jakikolwiek algorytm błędu). Jest to bardzo ważne w sytuacji, gdy nie dysponuje
282 się innym oszacowaniem tego błędu (w wielu problemach naturalnych dla człowieka jak roz-
283 poznawanie obiektów na obrazkach jest nim błąd ludzki lub też błąd popełniany przez zespół
284 ekspertów w bardziej zaawansowanych zastosowaniach).

285 Po drugie, wykorzystane zostały dwa algorytmy mocno różniące się w swojej naturze, co po-
286 zwala wykorzystać cechy każdego z nich w analizie: przykładowo sieci neuronowe dobrze radzą
287 sobie z nieustrukturyzowanymi danymi – potrafią tworzyć wysoko poziomowe cechy na podsta-
288 wie niskopoziomowego wejścia (np. położenia oka na zdjęciu twarzy na podstawie pixeli). Są
289 natomiast trudne w interpretacji i często traktowane są jako tzw. "czarne skrzynki" (ang. *black*
290 *box*). Oprócz tego, liczba możliwych konfiguracji sieci jest ogromna i przez to niemożliwe jest
291 stwierdzenie czy wykorzystane zostały pełne ich możliwości.

292 Z kolei drzewa decyzyjne posiadają stosunkowo niewielką liczbę parametrów, a ich trenowa-
293 nie jest bardzo szybkie co pozwala na ich ekstensywne przeszukiwanie i otrzymanie wyników,
294 które można uznać za optymalne dla tego algorytmu. Ponadto, w przypadku drzew istnieją
295 niewymagające dodatkowych obliczeń miary użyteczności poszczególnych zmiennych, co daje
296 wgląd w działanie algorytmu i poprawia intuicyjne zrozumienie jego predykcji.

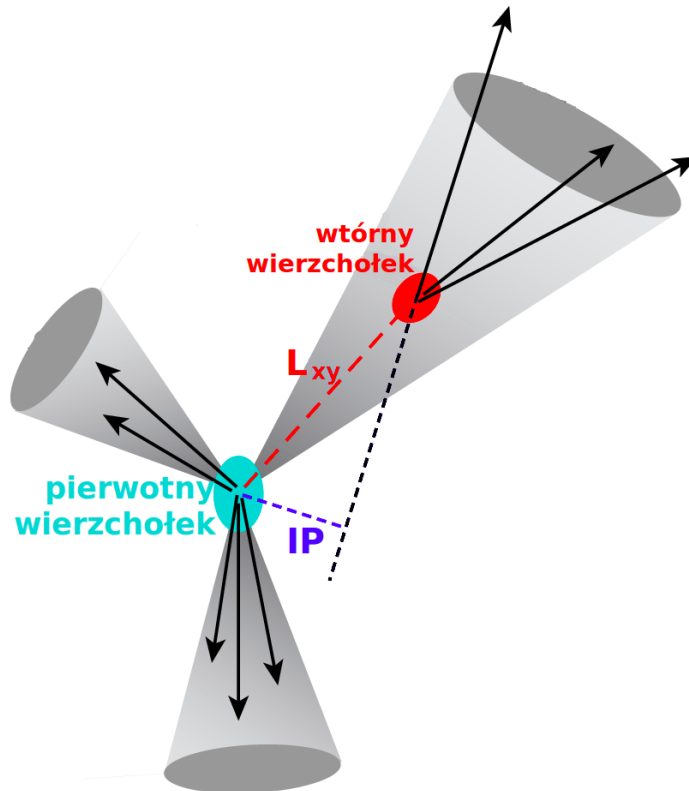
3 Dane

Dane użyte w analizie pochodzą z symulacji Monte Carlo zderzeń proton-proton przy energii w układzie środka masy równej $\sqrt{s} = 13$ TeV dostępnych na serwerach eksperymentu ALICE. W symulacjach zderzeń użyty był symulator `Pythia8` (*tune: Pythia8Jets_Monash2013*), następnie przy pomocy `Geant3` symulowany był transport cząstek i odpowiedź detektora ALICE.

Do rekonstrukcji dżetów wykorzystany został algorytm *anti-kt* z parametrem $R = 0.4$ zaimplementowany w pakiecie `FASTJET`. Dżetów poszukiwano tylko wyłącznie wśród cząstek naładowanych (ang. *charged jets*).

Do analizy wybrano dżety o p_T większym niż 15 GeV i mieszczących się w całości w akceptancji detektora *TPC*, tj. $|\eta| < 0.9$, co przy użytym parametrze rozmiaru dżetu $R = 0.4$, daje ograniczenie na pseudospieszość $|\eta| < 0.5$ dla osi dżetu.

Dla każdego dżetu obliczony został szereg wielkości, które można podzielić na zmienne na poziomie dżetu, związane z wtórnymi wierzchołkami oraz cząstkami tworzącymi dżet. Za potencjalne wtórne wierzchołki uznaje się wszystkie kombinacje trzech cząstek spełniających pewne dosyć luźne wymagania jak $p_T > 1\text{GeV}$ (rozważane są wyłącznie trzy-cząstkowe wtórne wierzchołki), stąd ich liczba może być dużo większa od liczby cząstek tworzących dżet.



Rysunek 6: Rysunek ilustrujący znaczenie używanych wielkości: L_{xy} oraz parametru zderzenia IP .

Lista używanych zmiennych:

- Zmienne na poziomie dżetu:

- η, ϕ – pseudospieszość (ang. *pseudorapidity*) i kąt azymutalny
- p_T – pęd poprzeczny dżetu

– masa dżetu

– powierzchnia dżetu – liczona w płaszczyźnie (η, ϕ) , do powierzchni dżetu zaliczany jest element w powierzchni w którym dodanie cząstki o nieskończenie małym pędzie poprzecznym sprawi, że zostanie ona zaliczona do tego dżetu [REF <https://arxiv.org/pdf/0707>]

– gęstość tła (w danym zdarzeniu)

– N_{SV} – liczba wtórnych wierzchołków

– $N_{Constit}$ – liczba cząstek tworzących dżet

• Zmienne opisujące cząstki tworzące dżet:

– η, ϕ – pseudopospieschność i kąt azymutalny cząstki względem osi dżetu

– p_T – pęd poprzeczny cząstki

– IP_D – rzut na kierunek poprzeczny wektora parametru zderzenia

– IP_Z – rzut na oś z wektora parametru zderzenia

• Zmienne opisujące wtórne wierzchołki:

– L_{xy} – odległość między pierwotnym a wtórnym wierzchołkiem (ang. *decay length*)

– σ_{Lxy} – niepewność wyznaczenia L_{xy}

– $\sigma_{vertex} = \sqrt{d_1^2 + d_2^2 + d_3^2}$ – rozrzut śladów (ang. *tracks*) wokół wtórnego wierzchołka, gdzie d_i to odległość najbliższego zbliżenia śladu / odległość najbliższego przelotu do wtórnego wierzchołka (ang. *distance of closest approach – DCA*)

– $M_{inv} = \sqrt{(E_1 + E_2 + E_3)^2 - (\vec{p}_1 + \vec{p}_2 + \vec{p}_3)^2}$ – masa niezmiennicza wierzchołka, gdzie E_i, p_i to energia i pęd i -tej cząstki tworzącej wierzchołek

– χ^2/Ndf dopasowania wtórnego wierzchołka

TBD: tabelaryczna postać, liczba SV i Nconstit, jak posortowane, wypileniania zerami korespondencja między użytymi danymi a algorytmami z sec. "identyfikacja dżetów b"

References

- [1] Anton Andronic. “An overview of the experimental study of quark-gluon matter in high-energy nucleus-nucleus collisions”. In: *Int. J. Mod. Phys. A* 29 (2014), p. 1430047. DOI: 10.1142/S0217751X14300476. arXiv: 1407.5003 [nucl-ex].
- [2] Vardan Khachatryan et al. “Evidence for collectivity in pp collisions at the LHC”. In: *Phys. Lett. B* 765 (2017), pp. 193–220. DOI: 10.1016/j.physletb.2016.12.009. arXiv: 1606.06198 [nucl-ex].
- [3] Jaroslav Adam et al. “Enhanced production of multi-strange hadrons in high-multiplicity proton-proton collisions”. In: *Nature Phys.* 13 (2017), pp. 535–539. DOI: 10.1038/nphys4111. arXiv: 1606.07424 [nucl-ex].
- [4] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. “The Anti-k(t) jet clustering algorithm”. In: *JHEP* 04 (2008), p. 063. DOI: 10.1088/1126-6708/2008/04/063. arXiv: 0802.1189 [hep-ph].
- [5] Carlos A. Salgado and Urs Achim Wiedemann. “Calculating quenching weights”. In: *Phys. Rev. D* 68 (2003), p. 014008. DOI: 10.1103/PhysRevD.68.014008. arXiv: hep-ph/0302184 [hep-ph].
- [6] Yuri L. Dokshitzer and D. E. Kharzeev. “Heavy quark colorimetry of QCD matter”. In: *Phys. Lett. B* 519 (2001), pp. 199–206. DOI: 10.1016/S0370-2693(01)01130-3. arXiv: hep-ph/0106202 [hep-ph].
- [7] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD ’16*. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: <http://doi.acm.org/10.1145/2939672.2939785>.
- [8] James Bergstra and Yoshua Bengio. “Random Search for Hyper-parameter Optimization”. In: *J. Mach. Learn. Res.* 13 (Feb. 2012), pp. 281–305. ISSN: 1532-4435. URL: <http://dl.acm.org/citation.cfm?id=2188385>. 2188395.
- [9] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080. DOI: [https://doi.org/10.1016/0893-6080\(91\)90009-T](https://doi.org/10.1016/0893-6080(91)90009-T). URL: <http://www.sciencedirect.com/science/article/pii/089360809190009T>.
- [10] François Chollet et al. *Keras*. <https://keras.io>. 2015.
- [11] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from [tensorflow.org](https://www.tensorflow.org/). 2015. URL: <https://www.tensorflow.org/>.
- [12] Timothy Dozat. *Incorporating Nesterov Momentum into Adam*. 2015. URL: http://cs229.stanford.edu/proj2015/054_report.pdf.
- [13] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: <http://arxiv.org/abs/1412.6980>.
- [14] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435. URL: <http://dl.acm.org/citation.cfm?id=2627435>. 2670313.