

1 Spis treści

2	1 Fizyka	2
3	1.1 Plazma kwarkowo-gluonowa	2
4	1.2 Dżety	2
5	1.3 Identyfikacja dżetów b	3
6	1.4 Eksperyment ALICE	3
7	2 Uczenie maszynowe	4
8	2.1 Wzmacniane drzewa decyzyjne	4
9	2.2 Sieci neuronowe	5
10	2.3 Dyskusja użycia dwóch algorytmów	10
11	3 Dane	11
12	4 Analiza	13

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

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

61 1.3 Identyfikacja dżetów b

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

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

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

73 1.4 Eksperyment ALICE

74 TBD

75 2 Uczenie maszynowe

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

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

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

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

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

101 2.1 Wzmacniane drzewa decyzyjne

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

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

113 *Bagging* – w zastosowaniu dla drzew decyzyjnych nazywany algorytmem lasów losowych
114 (ang. *random forest*) - polega na wytrenowaniu wielu drzew, każdego na podstawie N przykła-
115 dów losowo wylosowanych z powtórzeniami spośród N -licznego zbioru treningowego. Dodatko-
116 wo, do uczenia każdego drzewa używa się tylko podzbioru wszystkich cech uczących. Końcową
117 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

118 wagami.

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

125 Parametry i sposób trenowania drzew decyzyjnych na analizowanych danych

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

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

142 Parametry optymalizowane w opisanym procesie to:

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

153 2.2 Sieci neuronowe

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

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

165 Sieci w pełni połączone

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

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

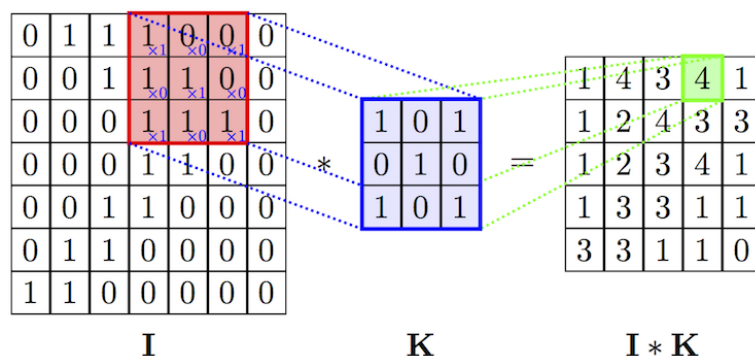
183 Sieci konwolucyjne

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

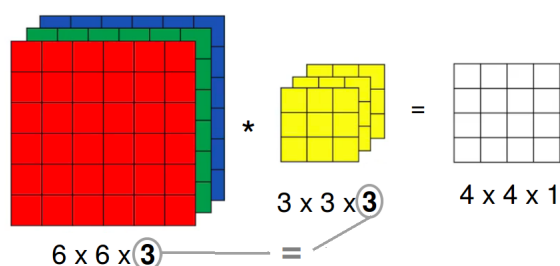
190 Sieci konwolucyjne różnią się od sieci typu *FC* tym, że część wag połączeń między warstwami
191 jest dzielona. Występuje w nich nowy rodzaj warstwy, nazywany warstwą konwolucyjną. Każda
192 jednostka w warstwie konwolucyjnej (filtr) ma pewną stałą (niewielką) liczbę wag. Połączenie z
193 dużym wejściem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

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

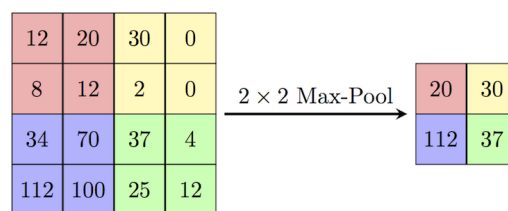
205 Oprócz warstw konwolucyjnych, w sieciach tego typu stosowane są także tzw. warstwy typu
206 *max-pooling*. Zasada jej działania jest bardzo prosta: wykonuje funkcję *maksimum* na zadanym



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



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



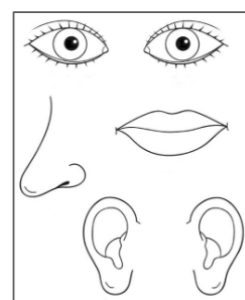
Rysunek 3: Działanie warstwy typu *max-pool*

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

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

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

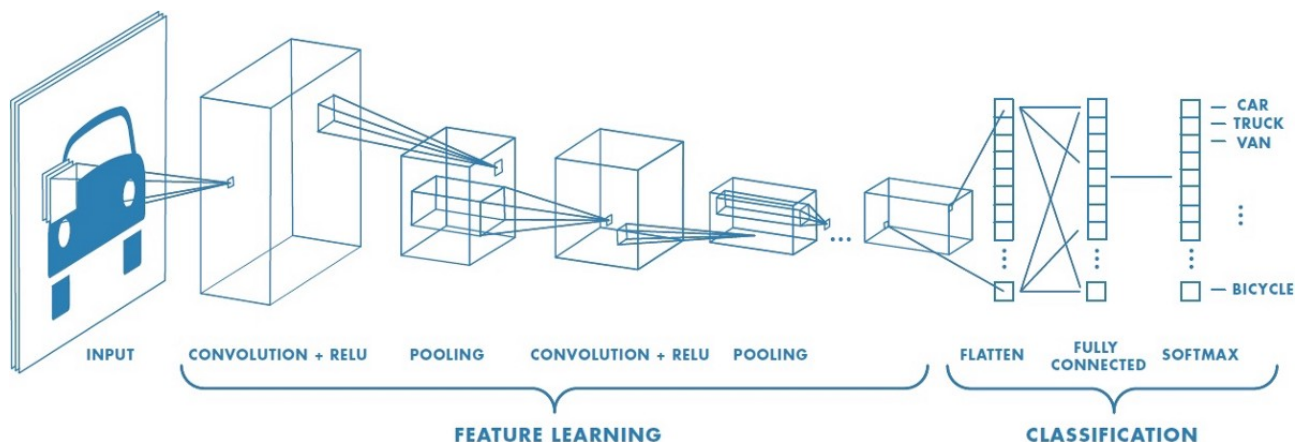
- 216 • niezmienniczość względem przesunięcia (ang. *translation invariance*) – głównie za sprawą dzielenia wag oraz obecności warstw
217 typu *max-pool*, położenie danej cechy na obrazie jest niemal
218 bez znaczenia (obraz po prawej stronie byłby rozpoznany ja-
219 ko twarz)
- 221 • lokalność połączeń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



224 Hiperparametry i trenowanie sieci neuronowych na analizowanych danych

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

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

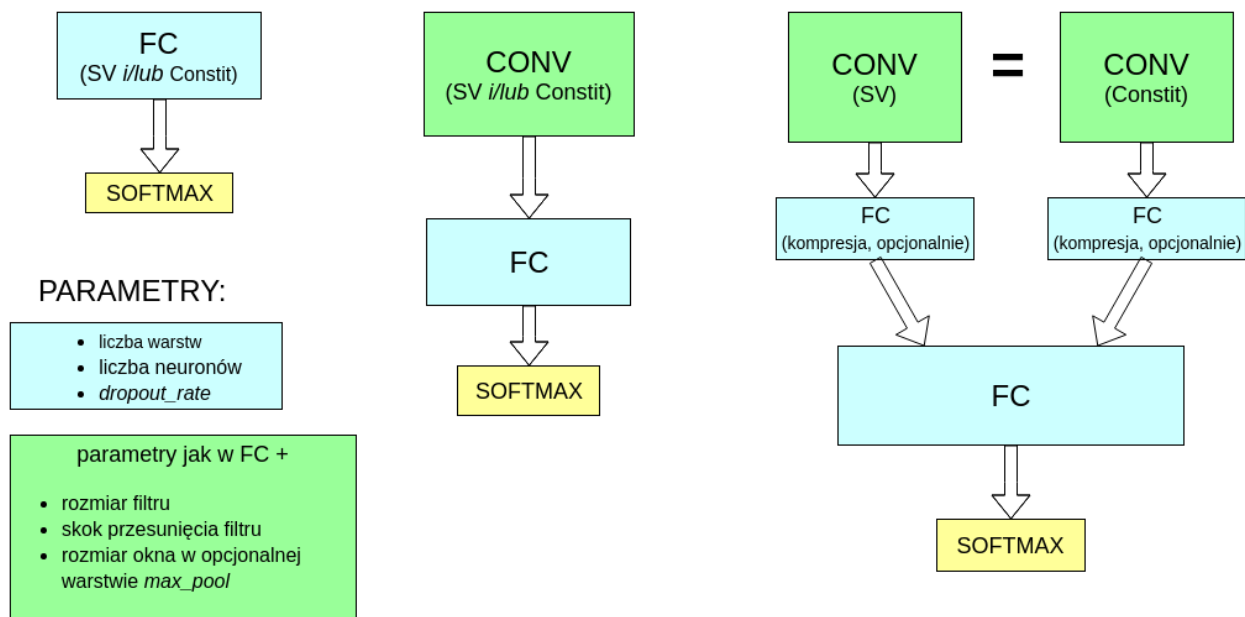


Rysunek 4: Typowa struktura stosowana w sieciach konwolucyjnych. Warstwy konwolucyjne mają za zadanie wydobywać cechy, na podstawie których późniejsze warstwy dokonują klasyfikacji. Widoczna jest charakterystyczna stopniowa zmiana rozmiaru przekazywanej macierzy: rozmiar poprzeczny maleje kosztem głębokości,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

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

232 Zestaw hiperparametrów definiujący działanie sieci w pełni połączonej:

- 233 • liczba warstw ukrytych
- 234 • liczba neuronów w każdej warstwie
- 235 • funkcja aktywacji – nieliniowa funkcja aplikowana przed zwróceniem wartości w każdym
236 neuronie, najpopularniejsze to \tanh , $ReLU$ ($f(x) = \max(0, x)$) oraz funkcja sigmoidalna
237 ($f(x) = \frac{1}{1+\exp(x)}$)
- 238 • algorytm optymalizacyjny – spadek gradientowy lub jego wariacje
- 239 • parametr szybkości uczenia i jego modyfikacje w trakcie uczenia
- 240 • liczba przykładów trenujących przetwarzanych w jednym kroku uczenia (ang. *batch_size*)
241 – im większy tym szybsze jest trenowanie sieci (dzięki wydajnym operacjom macierzowym), natomiast może się to odbywać kosztem precyzji
- 242
- 243 • liczba epok uczenia – ile razy będzie pokazywany sieci każdy przykład
- 244 • opcjonalnie: warunki stopu (ang. *early stopping*) w razie osiągnięcia *plateau* (wysycenia
245 procesu uczenia)
- 246 • opcjonalnie: regularyzacja przy pomocy różnych technik (zwykle konieczna)
- 247 • ponadto dla sieci konwolucyjnych:
 - 248 – liczba warstw konwolucyjnych i liczba filtrów w każdej warstwie
 - 249 – obecność lub brak warstw *max-pool* i rozmiar ich okna
 - 250 – 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

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

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

279 2.3 Dyskusja użycia dwóch algorytmów

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

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

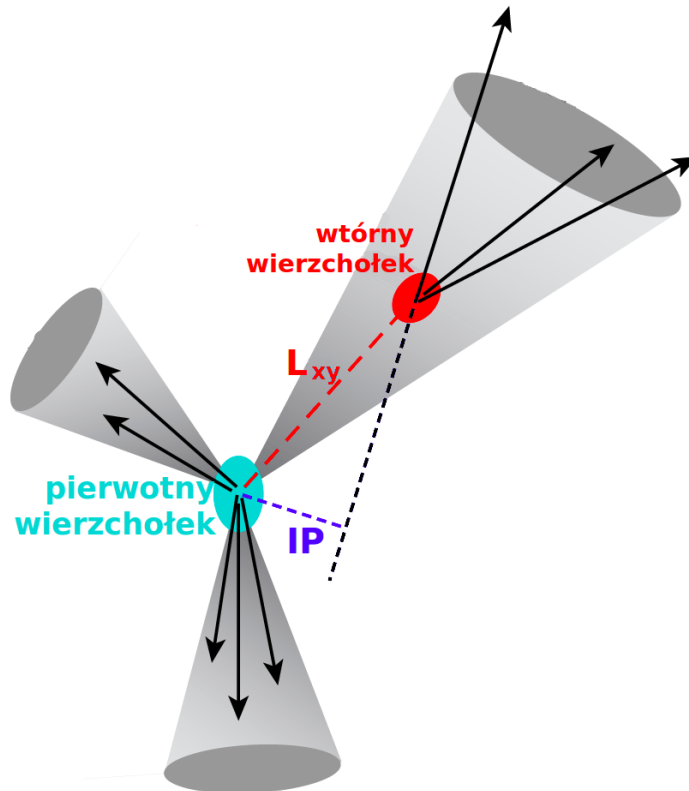
293 Z kolei drzewa decyzyjne posiadają stosunkowo niewielką liczbę parametrów, a ich trenowa-
294 nie jest bardzo szybkie co pozwala na ich ekstensywne przeszukiwanie i otrzymanie wyników,
295 które można uznać za optymalne dla tego algorytmu. Ponadto, w przypadku drzew istnieją
296 niewymagające dodatkowych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

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

303 Do rekonstrukcji dżetów wykorzystany został algorytm *anti-kt* z parametrem $R = 0.4$ za-
 304 implementowany w pakiecie **FASTJET**. Dżetów poszukiwano tylko wyłącznie wśród cząstek na-
 305 ładowanych (ang. *charged jets*).

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

309 Dla każdego dżetu obliczony został szereg wielkości, które można podzielić na zmienne
 310 na poziomie dżetu, związane z wtórnymi wierzchołkami oraz cząstkami tworzącymi dżet. Za
 311 potencjalne wtórne wierzchołki uznaje się wszystkie kombinacje trzech cząstek spełniających
 312 pewne dosyć luźne wymagania jak $p_T > 1\text{GeV}$ (rozważane są wyłącznie trzy-cząstkowe wtórne
 313 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 .

314 Lista używanych zmiennych:

315 • Zmienne na poziomie dżetu:

- 316 – η, ϕ – pseudospieszość (ang. *pseudorapidity*) i kąt azymutalny
- 317 – 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, wypieniania zerami korespondencja między użytymi danymi a algorytmami z sec. "identyfikacja dżetów b"

342 Dobór metryki

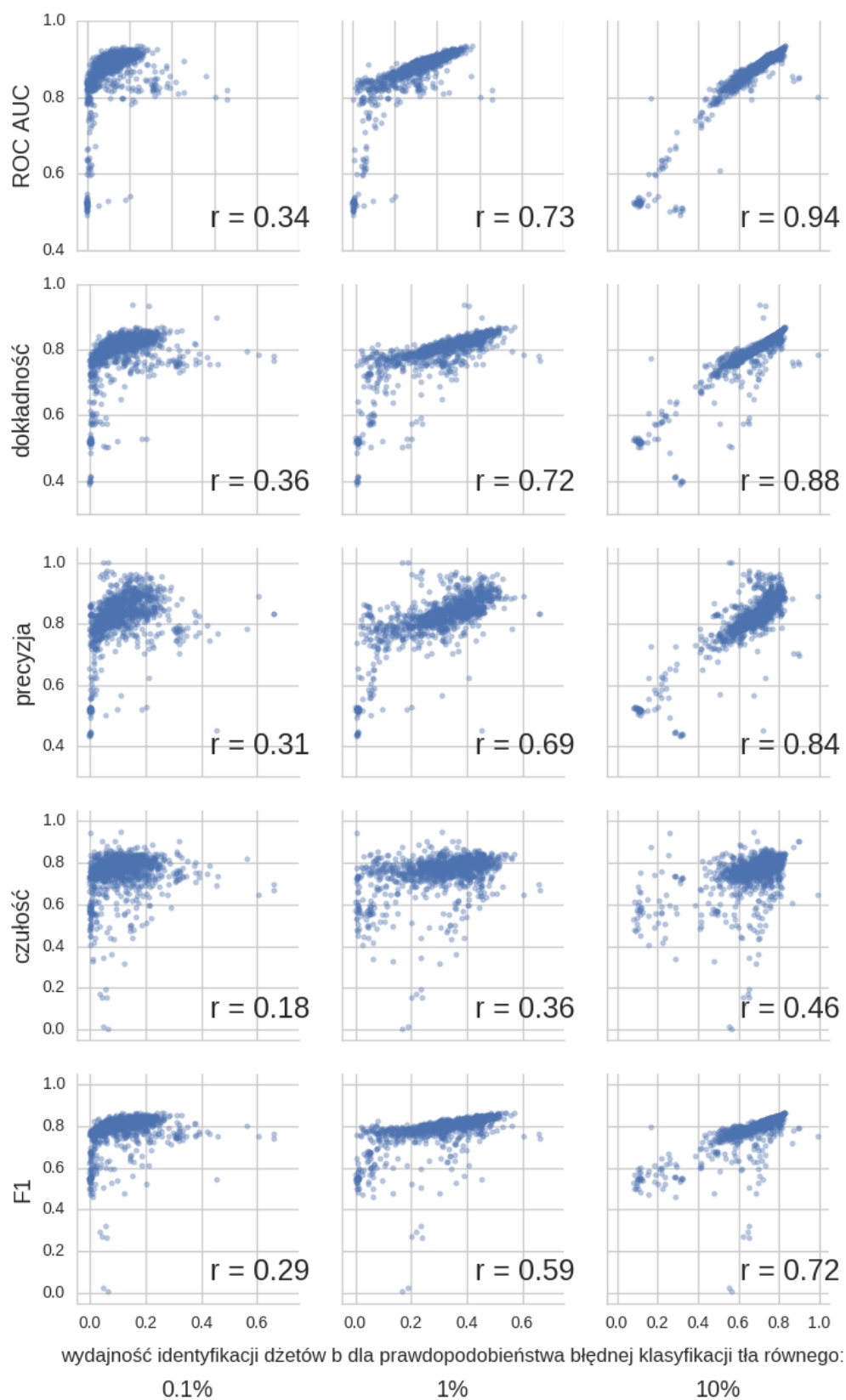
343 Bardzo ważnym elementem w trenowaniu algorytmów uczenia maszynowego jest dobór odpo-
 344 wiedniej metryki – klasycznym złym przykładem jest używanie dokładności (ang. *accuracy*) do
 345 oceniania klasyfikacji binarnej w przypadku dużego niezrównoważenia klas – algorytm przewi-
 346 dujący zawsze klasę większościową może osiągnąć dużą wartość dokładności będąc jednocześnie
 347 bardzo słabym modelem.

348 Kilka najczęściej używanych metryk wymieniono w Tab. ?? . Używanie i porównywanie
 349 kilku miar efektywności jest często niepraktyczne dlatego dobrze jest wybrać jedną metrykę.
 350 Przy jej wyborze należy kierować się potencjalnymi zastosowaniami modelu. W tym przypadku
 351 są to analizy fizyczne, które mogą mieć różne wymagania dotyczące czystości i liczebności
 352 otrzymywanych próbek a co za tym idzie, preferować inne punkty pracy zdefiniowane jako pary
 353 liczb: wydajność poprawnej klasyfikacji dżetów b (ang. *tagging efficiency = true positive rate*
 354 $= recall$), i ułamek niepoprawnie zaklasyfikowanych przypadków tła (ang. *mistagging rate =*
 355 *false positive rate*).

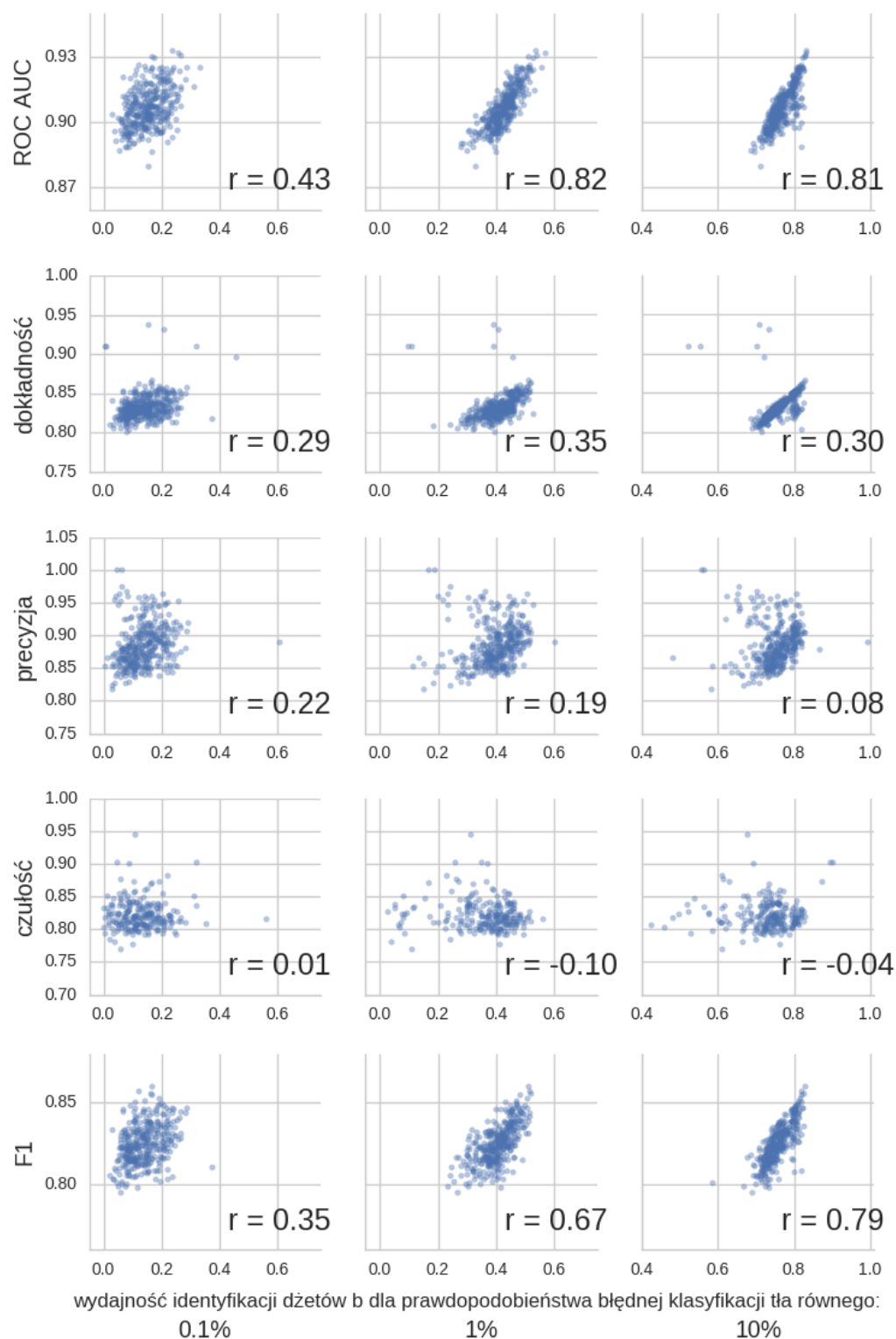
356 Naturalnym wyborem wydaje się pole pod powierzchnią krzywej ROC (ang. *ROC Area*
 357 *Under Curve – ROC AUC*). Potencjalną przeszkodą może być zakres rozsądnych wartości
 358 prawdopodobieństwa błędnej klasyfikacji przypadków tła: dżety b stanowią tylko kilka procent
 359 liczby wszystkich dżetów, zatem z punktu widzenia analizy dopuszczalne będą punkty pracy
 360 zapewniające wydajność identyfikacji dżetów b ok. 10 – 100 razy większą niż częstość niepo-
 361 prawnego zaklasyfikowania przypadków tła. Oznacza to, że zdecydowana większość punktów
 362 pracy znajdujących się na krzywej ROC jest nie do zaakceptowania – interesujące są tylko te o
 363 najniższych częstościach.

364 Aby ilościowo porównywać różne algorytmy wprowadzone zostaną trzy punkty pracy: o
 365 prawdopodobieństwie błędnej klasyfikacji tła równej 0.1%, 1% oraz 10%. Na Rys. 7 przedsta-
 366 wione zostały zależności poszczególnych metryk od wydajności identyfikacji dżetów b w tych
 367 trzech punktach pracy. Każdy punkt odpowiada jednemu eksperymentowi (dla dowolnego al-
 368 gorytmu) przeprowadzonemu w trakcie przygotowywania analizy. Daje to pogląd, używanie
 369 której metryki zapewni jednocześnie wysokie wartości wydajności na identyfikację dżetów b w
 370 wybranych punktach pracy. Najwyższe korelacje występują dla punktu pracy o najwyższym
 371 prawdopodobieństwie błędnej klasyfikacji tła – jak będzie to pokazane później wyniki dla tego
 372 punktu pracy są najbardziej stabilne. Spośród analizowanych metryk najwyższe wartości współ-
 373 czynnika Pearsona otrzymano dla pola pod powierzchnią krzywej ROC, dosyć wysokie również
 374 dla dokładności i precyzji. Widać, że wartości czułości są najsłabiej skorelowane z wydajnością
 375 identyfikacji dżetów b – jest zrozumiałe, że są one dużo niższe niż dla precyzji: wartości czułości
 376 maleją gdy błędnie klasyfikowane są dżety b stanowiące sygnał, podczas gdy wartości precyzji
 377 maleją, gdy błędnie klasyfikowane są przypadki tła. Z tych dwóch błędów, drugi jest bardziej
 378 kosztowny, gdyż błędna klasyfikacja nawet niewielkiej części tła znacznie pogarsza czystość
 379 otrzymywanej próbki.

380 Na Rys. 8 przedstawiono te same wykresy, ale tym razem wybrano tylko 25% najlepszych
 381 wartości dla każdej metryki – te punkty są bardziej znaczące, gdyż ostatecznie modele z ekspe-
 382 rymentów dających najlepsze wyniki będą używane. Dla tych wykresów otrzymano zdecydowa-
 383 ną dominację *ROC AUC* – wybór modeli dających najwyższe pole pod powierzchnią krzywej
 384 ROC zapewnia jednocześnie otrzymanie wysokich wartości wydajności identyfikacji dżetów b
 385 dla wybranych punktów pracy.



Rysunek 7: Zależność podstawowych metryk od wydajności identyfikacji dżetów b dla punktów pracy o prawdopodobieństwie błędnej klasyfikacji tła równej 0.1%, 1% oraz 10%. Dla każdego wykresu przedstawiono współczynnik korelacji r Pearsona.



Rysunek 8: Rysunek podobny do Rys. 7, ale przedstawione zostały tylko punkty odpowiadające eksperymentom o wartościach metryki będących w górnym kwartylu wartości danej metryki dla wszystkich eksperymentów.

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. 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>.