|  |  |  |  |
| --- | --- | --- | --- |
|  | Politechnika Bydgoska im. J. J. Śniadeckich  Wydział Telekomunikacji,  Informatyki i Elektrotechniki  **Zakład Systemów Teleinformatycznych** | |  |
| **Przedmiot** | Zaawansowane Techniki Sztucznej Inteligencji | | |
| **Prowadzący** | prof. dr hab. inż. prof. PBŚ Piotr Cofta | | |
| **Temat** | *Project* | | |
| **Student** | Cezary Tytko | | |
| **Ocena** |  | **Data oddania spr.** |  |

Przed przystąpieniem do etapu drugiego mieliśmy podjąć decyzję z którego zbioru będziemy korzystali w kolejnych etapach, ze zbalansowanego (wszystkich klas jest po 4 tysiące) czy nie zbalansowanego. Zacznę od tego że nie rozumiem stwierdzenia że pierwotne dane podane w pierwszym etapie są złe i do niczego się nie nadają, dane są jakie są i trzeba się z tym pogodzić, rozumiem problem z nie zbalansowanymi danymi w uczeniu maszynowym, ale to nie wina danych i w takim przypadku należałoby umieć sobie z tym poradzić stosując np. metody augmentacji danych. Jeżeli planowałbym oprzeć swoje rozwiązanie o duży model sieci neuronowej, gdzie cała analiza pozostawiona jest tylko w rękach wytrenowanemu modelu to zdecydowania wolałbym nie utrudniać i wybrać zbalansowane dane, jednak rozwiązanie które chciałbym zastosować (przedstawione na końcu raportu), w którym trenowany model nie będzie dokonywał klasyfikacji całych obrazów do jednej z 26 klas (szczegóły na końcu) problem niezbalansowania tych klas nie będzie miał znaczenia, dlatego pozostanę przy danych pierwotnych.

Etap 2. – ustalenie minimalnego poziomu jakości (benchmark).

Przed przystąpieniem do rozwiązania problemu, należy określić co chcemy osiągnąć, co jest wymagane, a co jest tylko opcjonalne, określić minimum które nas zadowoli, aby wiedzieć kiedy osiągniemy zakładany efekt.

W problemach związanych ze sztuczna inteligencją stasuje się różne metryki do porównywania jakości modeli miedzy sobą jak i dają możliwość jasnego określenia jak model jest dobry, metryki te sprowadzają się do procentowego określenia jak model jest dobry względem ideału (jeżeli taki ideał możemy określić). W problemie klasyfikacji (taki jak w projekcie) stosuje się celność, mówiącą ile obserwacji jest poprawnie klasyfikowanych, model idealny zawsze zwracał by poprawną klasę, czyli miałby dokładność 100%.

Określenie minimalnego poziomu jakości, możemy dokonać dowolnie, ale nie znając i nie analizując wcześniej problemu, możemy pomylić się w jedną jak i drugą stronę, przeszacować albo nie doszacować co jesteśmy w stanie osiągnąć. W naszym problemie musimy przypisać jedną z 24 klas (suma liczby oczek na poprawnych kostkach). W takim problemie możemy zacząć od analizy modelu losowego, jeżeli naiwnie założymy, że wszystkich klas jest po równo, taki model uzyskiwałby 100% / liczba klas = 100 / 26 % ~= 4. Jednak kiedy analizowaliśmy dane w pierwszym etapie okazało się że klasy nie są równo liczne (są niezbalansowane) tylko klasa „18” jest najliczniejsza, taka informacja z perspektywy określenia minimum jest bardzo istotna, ponieważ możemy utworzyć model stały (zawsze zwraca tą samą klasę), w naszym przypadku klasę „18”, w takim założeniu dokładność modelu będziemy liczyć: liczba wystąpień najliczniejszej klasy / wszystkie obserwacje = 10025 / 100000 ~= 10%, jak widać taki model jest już 2,5 razy lepszy niż model losowy, a z perspektywy kosztów jest on darmowy. Aby nie generować kosztów, możemy również wykorzystać już istniejący model, w takim przypadku zmienimy tylko wejście i wyjście z modelu tak aby przystosować go do nowych danych, wykorzystałem tutaj model z projektu „Sztuczne sieci neuronowe”, był to model służący do klasyfikacji liczby oczek na kostce, na podstawie obrazka.

Obraz zawierający zrzut ekranu, tekst, diagram, Wykres

Opis wygenerowany automatycznie

Definicja modelu:

1. class CNN(nn.Module):

2.     def \_\_init\_\_(self):

3.         super(CNN, self).\_\_init\_\_()

4.         self.conv1 = nn.Conv2d(1, 32, kernel\_size=3, stride=1, padding=1)

5.         self.conv2 = nn.Conv2d(32, 64, kernel\_size=3, stride=1, padding=1)

6.         self.pool = nn.MaxPool2d(2, 2)

7.         self.fc1 = nn.Linear(64 \* 25 \* 25, 64)

8.         self.fc2 = nn.Linear(64, 26)

9.

10.     def forward(self, x):

11.         x = self.pool(torch.relu(self.conv1(x)))

12.         x = self.pool(torch.relu(self.conv2(x)))

13.         x = x.view(-1, 64 \* 25 \* 25)

14.         x = torch.relu(self.fc1(x))

15.         x = self.fc2(x)

16.         return x

17.

Model po dostosowaniu wejść i wyjść i trenowaniu przez 5 epok, na danych treningowych stanowiących 80% całości (20% przeznaczone na testowanie), uzyskał wynik 10, 52% dla danych testowych i 15, 95% na danych testowych (model się przeuczył)

Obraz zawierający tekst, Czcionka, zrzut ekranu, czarne

Opis wygenerowany automatycznie

Jest to wynik którego się spodziewałem, ponieważ nie zakładałem że model uzyska wynik lepszy niż model stały, a i tak się przeuczy, zakładałem właśnie że model będzie dążył do tego modelu stałego i wskazywał najczęściej „18” niezależnie od treści obrazu, dodatkowo analizując przykłady błędnie sklasyfikowanych obrazów, model zwracał wartości „18” i klas bliskich jak „17” i „19”, wynika to z tego, że model po 5 epokach nie stał się jeszcze modelem stałym, ale patrząc na rozkład klas z analizowany w poprzednim etapie, jest to coś czego nie tylko można była się spodziewać, ale należało się tego spodziewać i świadczy to o tym że model uczy się w sposób właściwy, ale brakuje mu narzędzi do polepszenia wyniku.

Podsumowując powyższe 3 podejścia wyznaczyłbym plan absolutnego minimum na 10% dokładności, jest to próg poniżej którego rozwiązanie jest błędne, wyniki powyżej można uznać już za mały sukces.

Analizując problem w kolejnym etapie planuje podzielić rozwiązanie problemu na etapy, zaczynając od segmentacji obrazu i wykrycia/ wyizolowania kostek z obrazu, sklasyfikować na poprawne i niepoprawne, klasyfikować liczbę oczek na każdej kostce, i na końcu zsumować liczbę na poprawnych kostkach danego obrazu.