Metody Głębokiego Uczenia
Projekt nr 3

Piotr Olesiejuk

Marzec 2019

1 Temat projektu

Celem projektu było napisanie sieci LSTM w celu klasyfikacji komend głosowych. Zdanie miało zostać zrealizowane przy wykorzystaniu zbioru dostępnego w serwisie kaggle.com pod nazwą "speech recognition challange".

2 Uproszczenia

Na wstępie należy zaznaczyć, że ze względu na problemy wynikłe w trakcie rozwijania projektu, dotyczące przede wszystkim wczytywania danych, zostało przyjęte pewne uproszenie, które jednak można nieznaczącym kosztem rozszerzyć do początkowych założeń projektowych. Późne rozwiązanie problemu wczytywania danych wymusiło ograniczenie zbioru uczącego do zbioru składającego się z klas: 'yes', 'no', 'up', 'down', 'left', 'right', 'on', 'off', 'stop', 'go', pomijając przy tym klasę 'unknown' i uzyskując bardziej zbalansowany zbiór. W kolejnej iteracji projektu, klasa 'unknown' powinna być dodana do zbioru uczącego, niemniej ten zabieg ograniczył rozmiar danych i przyspieszył proces uczenia, co pozwoliło przetestować poprawność funkcjonowania architektury.

3 Wstępny preprocessing danych i przygotowanie danych

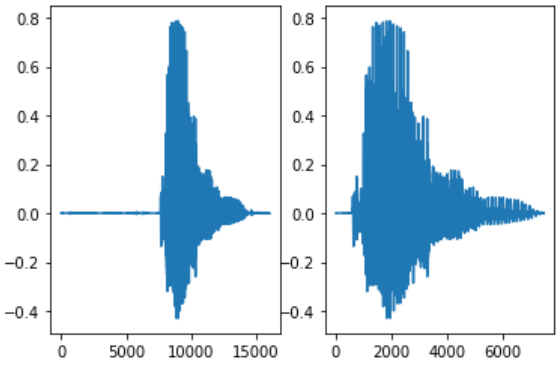
Pliki w stanie surowym to pliki audio w formacie .wav. Koniecznym było przetworzenie tych plików, do formatu, który może zostać podany jako wejście do sieci neuronowej. W tym celu zdecydowano o wyborze spektrogramów jako dobrego formatu treningowego. Rozważanymi formatami były również Mel Frequency Cepstrum Coefficients oraz Filter Bank Coefficients. W celu wygenerowania spektrogramów wykorzystano pakiet librosa. Testowana była również funkcja wavefile z pakietu scipy.io, niemniej w ostatecznym rozwiązaniu wykorzystano wspomnianą librosę. Problem z jakim należało się zmierzyć to nierówne długości próbek audio w zbiorze danych. Większość próbek była długości jednej sekundy, natomiast pojawiały się również próbki dużo krótsze. Koniecznym natomiast było ujednolicenie rozmiaru próbek treningowych w celu skutecznego podania ich na wejście do sieci neuronowej. Możliwymi do rozważenia scenariuszami są dwa następujące:

1. Generowanie spektrogramów oraz wypełnianie początkowej lub końcowej części spektrogramu zerami. Wtedy wszystkie próbki o długości 1s są normalnie generowane, natomiast przekształcenie spektrogramu jest realizowane na próbkach krótszych.
2. Generowanie spektrogramów i skalowanie ich przy zapisie do jednakowych rozmiarów.

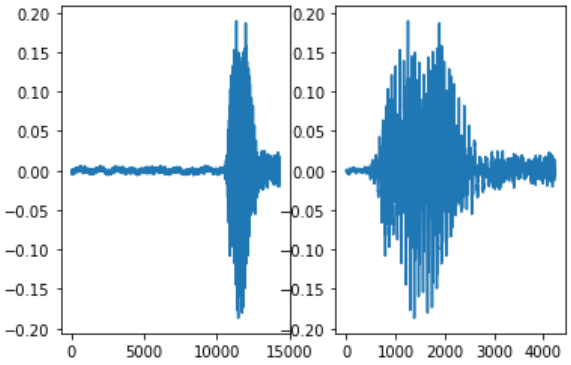
Pierwsze rozwiązanie wydaje się lepsze pod kątem sieci rekurencyjnych, natomiast drugie pod kątem sieci konwolucyjnych. W rozwiązaniu zostało wykorzystane podejście drugie. Mimo wykorzystania LSTM, motywacją tego wyboru było sprawdzenie jak sieć LSTM poradzi sobie z takim przekształceniem danych oraz porównanie tych wyników z wynikami z sieci konwolucyjnej.

3.1 Eliminacja ciszy

W procesie preprocesingu został wykonany zabieg odcięcia części sygnału na podstawie zadanej bramki. Miało to na celu wyodrębnienie właściwej części sygnału związanej z komendą głosową. Rozwiązanie wydaje się bardzo dobre do podejścia związanego ze skalowaniem spektrogramów, natomiast może być również pomocne w przypadku podejścia nr 1, opisanego wcześniej. Dzięki odcięciu ciszy, zera będą obecne zasadniczo jedynie z jednej strony spektrogramu co powinno wpłynąć pozytywnie na performance sieci. Poniżej prezentują się dwa przykładowe wykresy amplitudy sygnału od czasu przed oraz po przekształceniu.



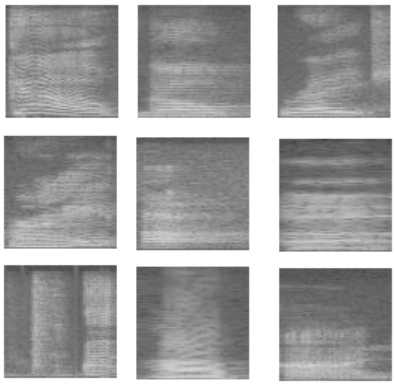
Rysunek 1: Usuwanie szumu



Rysunek 2: Usuwanie szumu

3.2 Generowanie spektrogramów

Do generowania spektrogramów wykorzystano wspomnianą wcześniej librosę oraz pakiet matplotlib. Aby konwertować obrazki z RGB do obrazków w skali szarości wykorzystano pakiet PILLOW. Poniżej prezentują się przykładowe wygenerowane spektrogramy, każdy o wymiarach 75 x 75 pikseli:



Rysunek 3: Przykładowe spektrogramy

4 Podział zbioru na część treningową i testową

Podobnie jak to było sugerowane na stronie internetowej kaggle.com, dokonano podziału na zbiór testowy oraz treningowy w taki sposób, aby w każdym ze zbiorów byli obecni inni speakerzy. Dzięki temu gwarantowane jest, że sieć uczy się w pełni na elementach odmiennych od tych na których następuje walidacja co dobrze symuluje rzeczywiste zastosowania rozpoznawania komend głosowych. Podziału zbioru dokonano również z zachowaniem proporcji w klasach, zatem odpowiednio w zbiorze treningowy znajduje się 80% elementów z każdej klasy, zaś reszta znajduje się w zbiorze testowym.

Aby podawać spektrogramy na wejście do sieci koniecznym było zapewnienie DataLoadera, który będzie ładował do pamięci kolejne batche danych. W początkowej fazie projektu starano się wykorzystać dołączony do biblioteki PyTorch ImageLoader. Jest to DataLoader, który wczytuje dane z odpowiednio ustrukturyzowanej formy danych zebranych w katalogach. Niemniej podeście to okazało się niewystarczające ze względu na duże czasy ładowania danych. Ciężko jest jednoznacznie określić czy duże czasy ładowania były spowodowane tylko tym elementem, natomiast, aby umożliwić szybkie ładowanie danych zostały poczynione następujące kroki:

1. ImageLoader został zastąpiony DataLoaderem ładującym własny Dataset
2. Rozmiary spektrogramów początkowo wynosiły 450 x 450 pikseli, zostały zredukowane do 75 x 75 pikseli
3. Dane początkowo były ładowane na dysk google, który następnie był mountowany na Google Colab. Zmieniono podejście na takie, że przeprocesowane wstępnie dane były pakowane i umieszczane na AWS S3 Bucket, z którego następnie były pobierane funkcją wget bezpośrednio w serwisie Google Colab

Wszelkie czynności związane ze wstępnym processingiem oraz podziałem zbiorów na część treningową oraz testową były wykonywane na lokalnym komputerze, natomiast trening sieci odbywał się na serwerze Google Colab z wykrorzystaniem GPU.

5 Modele

Zostały przetestowane dwa modele, jeden w pełni oparty o architekturę konwolucyjną oraz jeden wykorzystujący LSTM oraz częściowo konwolucję. Poniżej zostały przestawione poszczególne warstwy każdego z modeli:

5.1 Model LSTM

```
1: Conv2d(in_channels = 1, out_channels = 8,
          stride = 1, kernel_size = 5, padding = 1)
2: ReLU()
3: MaxPool2d(kernel_size = 2, stride = 2)
4: LSTM(input_size= 36 * 36 * 8, hidden_size= 300)
5: ReLU()
6: Linear(300, 84)
7: ReLU()
8: Dropout(p = 0.5)
9: Linear(84, number_of_classes)
```

5.2 Model Konwolucyjny

```
1: Conv2d(in_channels = 1, out_channels = 8,
          stride = 1, kernel_size = 5, padding = 1)
2: ReLU()
3: MaxPool2d(kernel_size = 2, stride = 2)
4: Conv2d(in_channels = 8, out_channels = 16,
          stride = 1, kernel_size = 5, padding = 0)
5: ReLU()
6: MaxPool2d(kernel_size = 2, stride = 2)
7: Linear(300, 84)
8: ReLU()
9: Dropout(p = 0.5)
10: Linear(84, number_of_classes)
```

6 Wyniki

W niniejszej sekcji zebrano wyniki jakie uzyskano dla poszczególnych modeli, zaagregowano takie informacje jak dokładność na zbiorach testowym i treningowym. W obu przypadkach sieci były testowane z takimi samymi następującymi parametrami:

1. Rozmiar mini-batch: 100
2. Wartość regularyzacji L2: 0.001
3. Learinig rate: 0.001
4. Optymalizator: Adam

Jedyną transformacją jaką wykonano na zbiorze danych była operacja .ToTensor(). Nie została zastosowana normalizacja, ale bardzo możliwe, że jej zastosowanie poprawiłoby uzyskane wyniki. Klasy do których odwołują się wykresy w dalszej części zostały zebrane w tabeli.

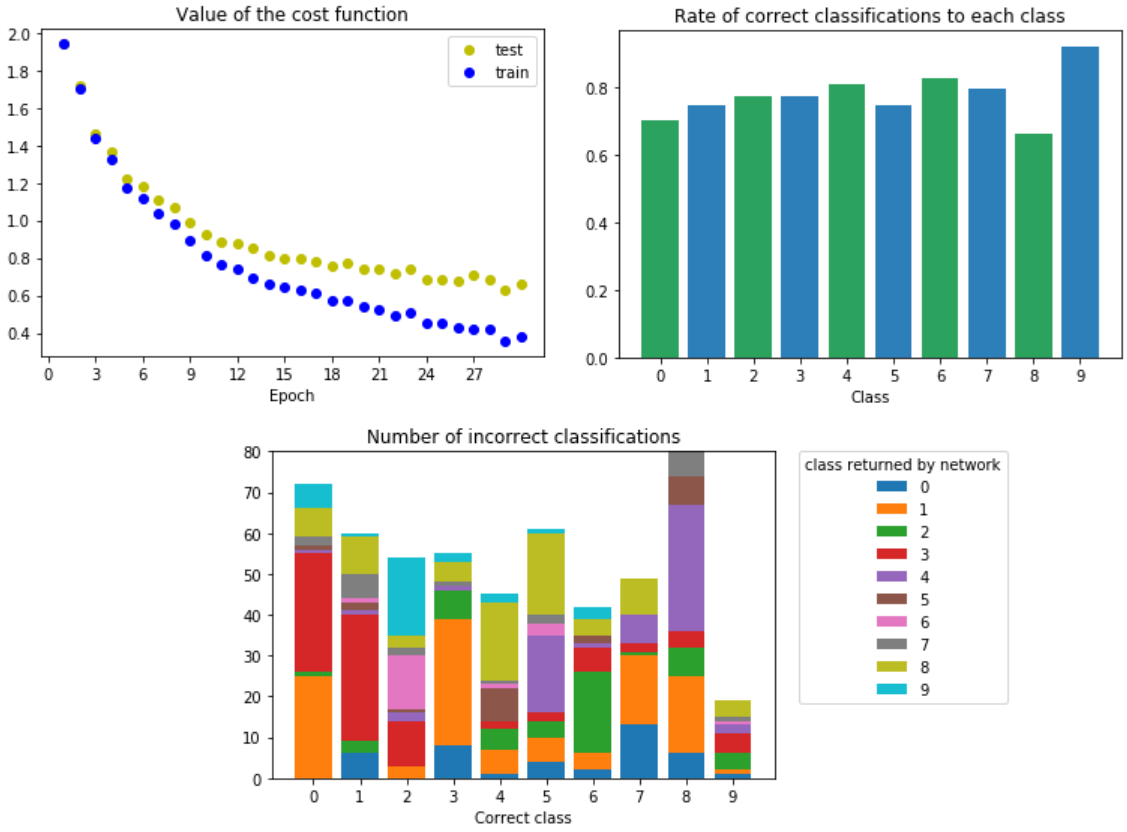
Klasa numerycznie	Klasa rzeczywista
0	down
1	go
2	left
3	no
4	off
5	on
6	right
7	stop
8	up
9	yes

6.1 Sieć LSTM

W poniższej tabeli zostały zebrane wartości skuteczności klasyfikacji sieci na zbiorze treningowym oraz testowym.

Skuteczność na zbiorze testowym	Skuteczność na zbiorze treningowym
77,49%	88,61%

Powyższe dane to wartości uzyskane po 30 epokach treningu, niemniej w 30 nastąpił spadek, ponieważ accuraccy na zbiorze testowym w 29 epoce wynosił 79,8%, co było najwyższą wartością podczas treningu. Poniżej zaprezentowano wykresy przedstawiające proces uczenia, a także ilość poprawnie oraz niepoprawnie sklasyfikowanych komend.



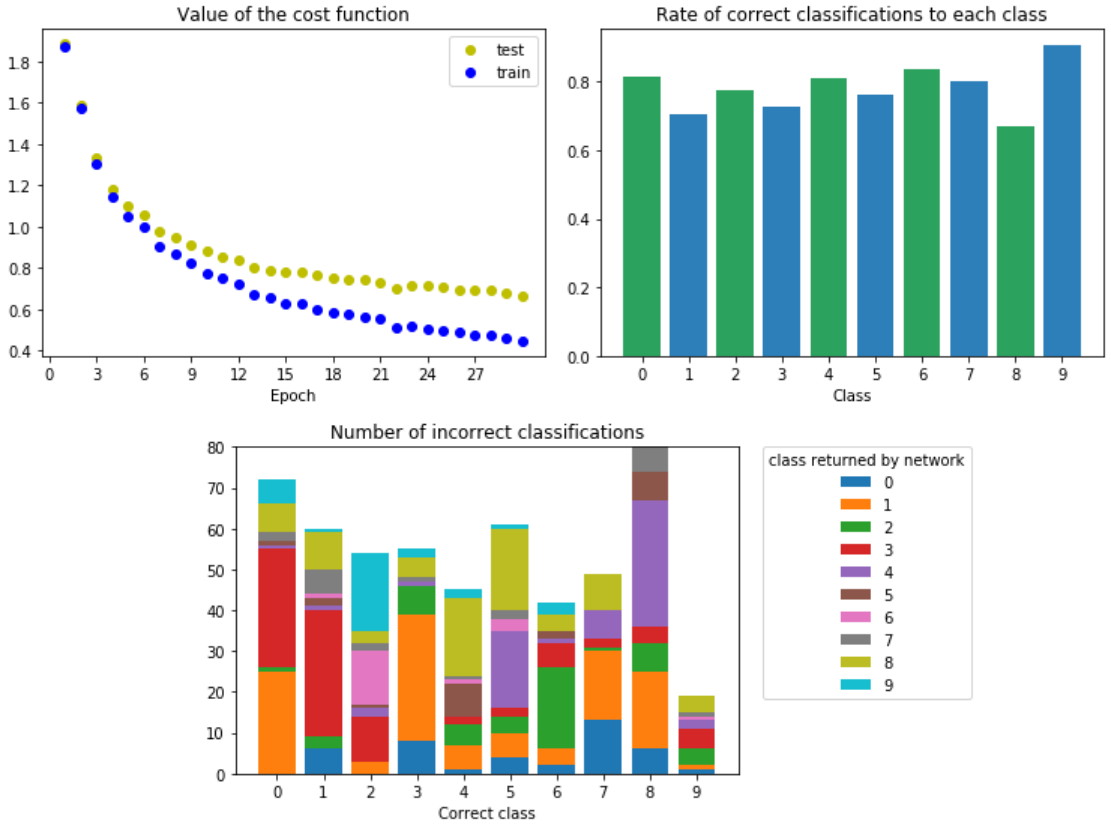
Rysunek 4: Podsumowanie modelu

6.2 Sieć Konwolucyjna

W poniższej tabeli zostały zebrane wartości skuteczności klasyfikacji sieci na zbiorze treningowym oraz testowym.

Skuteczność na zbiorze testowym	Skuteczność na zbiorze treningowym
78,1%	85,91%

Powyższe dane to wartości uzyskane podobnie jak w przypadku sieci LSTM po 30 epokach treningu. Poniżej zaprezentowano wykresy przedstawiające proces uczenia, a także ilość poprawnie oraz niepoprawnie sklasyfikowanych komend.



Rysunek 5: Podsumowanie modelu

7 Wnioski

Jak widać, zamiana warstw konwolucyjnej i maxpool na warstwę LSTM nie wpływa znacząco na zmianę wyników. Sieć LSTM nieco lepiej radzi sobie na zbiorze treningowym. Na zbiorze testowym jednak ciężko wyrazić jednoznaczną ocenę o wyższości jednego rozwiązania nad drugim, gdyż wyniki są bardzo zbliżone. Jak widać na wykresach, sieci najlepiej radziły sobie z rozpoznawaniem słowa "yes" natomiast najwięcej problemów miały ze słowem "up". W procesie usuwania szumu mogło się zdarzyć, że cały sygnał został odcięty, co sprawiło, że otrzymany spektrogram był pusty, mogło to wpłynąć w pewien sposób na wynik, gdyż tego typu rekordy nie były odfiltrowywane po preprocessingu. Projekt przyjmował nieco szersze założenie, mianowicie dodatkową klasę 'unknown', która byłaby klasą agregującą wszelkie klasy inne niż te, które brano tutaj pod uwagę. Niemniej, przekształcenie obecnego rozwiązania na rozwiązanie rozszerzone jest stosunkowo proste na podstawie załączonych do rozwiązania kodów. Zdecydowano się na redukcję tylko ze względu na przyspieszenie procesu uczenia.
Jako wniosek płynący również z samego użytkowania PyTorcha należy wspomnieć, że prawdopodobnie problematyczne staje się pisanie sieci konwolucyjnych i ich ewaluacja w systemie operacyjnym Windows. Przeniesienie tej samej sieci na Google Colab działa poprawnie, gdzie na Windowsie pojawiają się błędy.

8 Źródła

1. Wykorzystanie ImageFolder oraz pobieranie, rozpakowywanie danych na Google Colab: https://www.youtube.com/watch?v=5rW_PZI4B2It=420s

2. Preprocessing danych, w tym funkcja do odszumiania sygnału: https://www.youtube.com/watch?v=Z7YM-HAz-IYlist=PLhA3b2k8R3t2Ng1WW_7MiXeh1pfQJQi_P

3. Zapisywanie spektrogramów: https://stackoverflow.com/questions/47147146/save-spectrogram-only-content-without-axes-or-anything-else-to-a-file-using-m

4. Materiały zamieszczone w Kernels na stronie kaggle.com, które były związane ze Speech Recognition Challange.

5. Dokumentacja PyTorch

6. Dedykowany Dataset: https://github.com/utkuozbulak/pytorch-custom-dataset-examples/blob/master/src/custom_datasets.py