Głębokie uczenie i inteligencja obliczeniowa
Automatyka i Robotyka, II stopień
2021/2022

Skład zespołu:

- Roman Dembrovskyi
- Piotr Kula
- Mateusz Sołtys
- Filip Żmijewski

Opiekun: dr hab. inż. Joanna Kwiecień

Spis treści

1. Wstęp

Rozwój technologii i chęć usprawnienia każdej z form współczesnego życia powoduje, iż inżynierowie i programiści poszukują rozwiązań, które pozwolą zautomatyzować pewne czynności co pozwoli na wykluczenie czynnika ludzkiego. Takie rozwiązania mogą powodować zwiększenie bezpieczeństwa, nieomylność i pozwolić na sprawdzanie decyzji człowieka.

2. Cel i problem badania

Celem niniejszego projektu jest opracowanie i oprogramowanie sieci neuronowej, której celem jest rozpoznawanie stylu muzycznego na podstawie 30sekundowych sampli utworu. Aby rozwiązać zadanie wykonane zostały następujące kroki:

- stworzenie playlisty na spotify,

- stworzenie aplikacji pobierającej playlisty ze spotify (wykorzystanie spotify API),

- skrócenie utworów muzycznych do podanych wcześniej 30s,

- wygenerowanie spektogramu,

- stworzenie DataSet,

- utworzenie modelu,

- trening sieci,

- badanie działania algorytmu.

Powyższe kroki zostały zrealizowane z wykrozystaniem języka programowania Python, bibliotek takich jak numpy oraz tensorflow, które jest częścią Keras API dodatkowo w celu posługiwania się plikami dźwiękowymi użyteczna okazała się biblioteka opensoundscape.

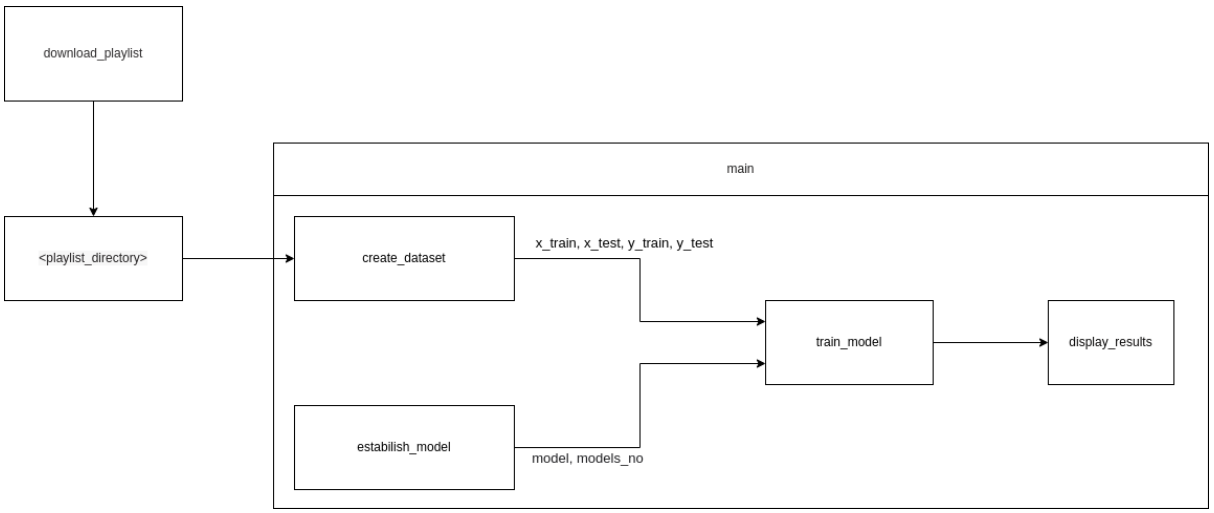
3. Propozycja rozwiązania

Podjętym sposobem rozwiązania zostało stworzenie sieci neuronowej konwolucyjnej. Sieć została zaimplementowana w środowisku Python z użyciem biblioteki Keras. Keras jest powszechnie stosowaną biblioteką do uczenia maszynowego, opartą na open-sourcowej platformie TensorFlow 2.

Używając modelu Sequential z biblioteki Keras utworzono dwuwarstwową sieć konwolucyjną z dodatkową warstwą DropOut o częstotliwości 0.3. Następnie przetrenowano ją na liście utworów.

Jako dane do aplikacji wykorzystano playlistę Dance Party ze Spotify, zawierającą 150 różnych utworów. W prawdzie długości utworów były zróżnicowane, jednakże w czasie działania programu przeprowadzane jest skracanie długości próbek do jednolitej długości (30 sekund).

Zdecydowano się na napisanie następującej struktury przepływu danych. Jako, że pliki zawierające utwory mają dość pokaźne rozmiary postanowiono wykorzystać do ich przechowywania oddzielny folder. Z tego powodu przed uruchomieniem głównej części programu należy użyć wbudowanej funkcji pobierającej pliki. Przetwarzanie danych przedstawiono na poniższym schemacie:



4. Aplikacja

Stworzona aplikacja została zaprogramowana tak, aby jej wykorzystanie było jak najbardziej uniwersalne. Po uruchomieniu jesteśmy proszeni o podanie linku do wykorzystywanej playlisty, a następnie algorytm przechodzi do pobrania playlisty. Wykorzystane zostało tutaj opensourcowe spotify API spotify-dl. Jest to pakiet który po podaniu hiperłącza zbiera metadane utworu ze spotify i następnie używa pakietu yt-dlp do pobrania jej z serwisu YouTube. Dzięki temu uzyskano dostęp do wszystkich utworów i możliwe jest ich dowolne przetwarzanie. Jako, że pakiet ten nie jest częścią aplikacji, przed rozpoczęciem korzystania z niej należy go doinstalować.

Następnie pobrane utwory zostają skrócone do odpowiednich długości. Zdecydowaliśmy się na 30s jak końcowo pokażą testy był to dobry wybór. Z 30 sekundowych sampli kolejna funkcja generuje spektrogramy, a w końcowym kroku generowany jest pełen dataset, który następnie nasz algorytm wykorzystuje do trenowania. Proces trenowania zajął w przypadku naszych danych około 2 godzin. Po tym procesie nastąpił etap faktycznego badania sieci i jej jakości, a otrzymane przez nas wyniki były bardzo obiecujące. Całość procesu testowania została opisana w kolejnym rozdziale.

Opis funkcjonalności:

Najpierw zostały stworzone funkcje *download_playlist* oraz *create_dataset* , realizujące odpowiednio pobieranie playlist pożądanych styli z platformy Spotify i stworzenie z nich zbiorów danych, umożliwiających uczenie się i testowanie sieci neuronowej.

Dalej została zaimplementowana funkcja *establish_model*, odpowiadająca za stworzenie modeli sieci. Przy jej implementacji został użyty sekwencyjny szablon modeli z biblioteki Keras z odpowiednimi dopasowaniami Flatten, Dropout, Dense i funkcji Softmax.

Kolejna funkcja *train_model*, jak wynika z nazwy, odpowiadała za trenowanie stworzonej modeli. Do kompilacji modeli zostały użyte optymizator SGD oraz marker strat *binary cross-entropy* (binarnej entropii krzyżowej).

Została zaimplementowana funkcja, niezbędna do reprezentacji wyników, generująca spektrogramy.

Również zostały zaimplementowane “quality of life” funkcje, które modyfikowały długość kawałków muzycznych do wspomnianych wcześniej 30 sekund oraz zmieniały nazwy plików do ich łatwiejszej analizy i użycia.

Na sam koniec została zaimplementowana część programu main(), która obejmowała działanie znaczącej części programu i zwracała otrzymane wyniki:

```
def main():
  models_set = [
    [16, 8], [32, 16], [64, 32],
    [16, 8, 4], [32, 16, 8], [64, 32, 16],
  ]

  pooling_set = [
    [3, 3], [3, 3], [3, 3],
    [3, 2, 2], [3, 2, 2], [3, 2, 2],
  ]
  folder_no = 6

  general_dir = "D:\\uni\\deepLearning\\dataset_2"
  x_train, x_test, y_train, y_test = create_dataset(general_dir)
  for count, s in enumerate(models_set):
    mdl, mdl_no = establish_model(s, pooling_set[count], 3)
    history = train_model(mdl, x_train, y_train, mdl_no, folder_no, lr=0.0001)
    model_filepath = f"D:\\uni\\deepLearning\\models\\models_{folder_no}\\{mdl_no}\\"
    train_acc = mdl.evaluate(x_train, y_train)
    test_acc = mdl.evaluate(x_test, y_test)
    predictions = mdl.predict(x_test)
    # conf_matrix = confusion_matrix(y_test.argmax(axis=1), predictions.argmax(axis=1))
    with open(model_filepath+'eval.txt', 'w', encoding='utf-8') as f:
      f.write(f"Train acc: {train_acc}\n"
          f"Test acc: {test_acc}\n"
          f"y_test:\n{y_test}\n\n"
          f"predictions:\n{predictions}")
    plt.plot(history.history['accuracy'], label='train')
    plt.plot(history.history['val_accuracy'], label='validation')
    plt.legend()
    plt.savefig(f'{model_filepath}val_graph.png')
    plt.close()
  print("fin")

if __name__ == "__main__":
  main()
  general_dir = "D:\\uni\\deepLearning\\dataset_2"
  x_train, x_test, y_train, y_test = create_dataset(general_dir)
  model_folder_dir = 'D:\\uni\\deepLearning\\models\\models_4'
  model_dir = 'model_32_16\\311-0.0950032-0.1710008.hdf5'
  pred, actual_y = load_n_predict(f'{model_folder_dir}\\{model_dir}', x_test, y_test)
  with open('predicions.txt', 'w', encoding='utf-8') as f:
    f.write(f'predictions:\n'
        f'{pred}\n\n'
        f'actual y:\n'
        f'{actual_y}')
  pass
```

5. Wyniki, warstwy i badanie jakości sieci

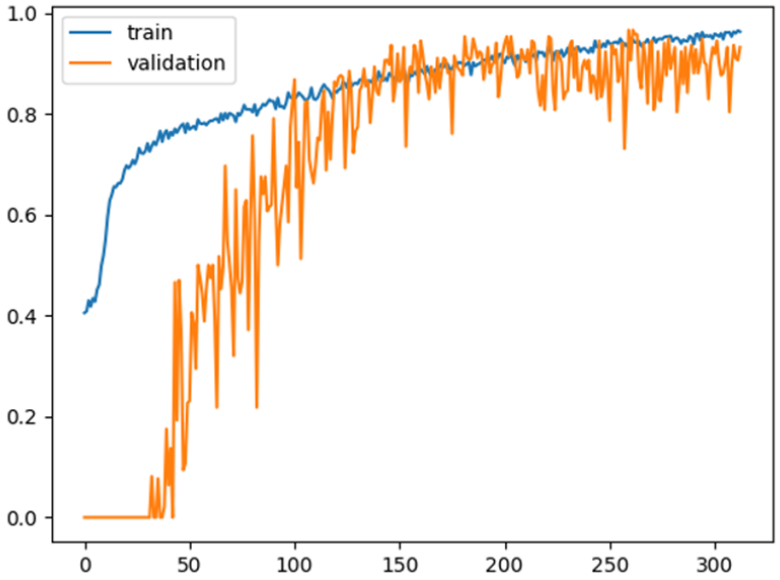
Pierwszą warstwą została użyta warstwa konwolucyjna 2D . Warstwa ta tworzy jądro konwolucji, które jest konwertowane z danymi wejściowymi warstwy w celu utworzenia tensora danych wyjściowych. Jeśli zmienna biasowa ma wartość True, tworzony jest wektor stronniczości, który jest dodawany do danych wyjściowych.

Kolejna warstwa MaxPooling2D zmniejsza rozdzielczość sieci poprzez zmniejszenie próbkowania danych wejściowych wzdłuż jej wymiarów przestrzennych (wysokość i szerokość), pobierając maksymalną wartość z okna danych wejściowych (o rozmiarze określonym przez pool_size) dla każdego kanału danych wejściowych. Okno jest przesuwane o kolejne kroki wzdłuż każdego wymiaru.

Na sam koniec jest używana warstwa dropout, która zmniejsza liniowość sieci, tym samym naprawiając błąd liniowości systemu. Warstwa dropout losowo ustawia jednostki wejściowe na 0 z częstotliwością równą zmiennej rate w każdym kroku w czasie szkolenia, co pomaga uniknąć przepełnienia. Wejścia nie ustawione na 0 są skalowane w górę o 1/(1 - współczynnik) w taki sposób, że suma wszystkich wejść pozostaje niezmieniona.

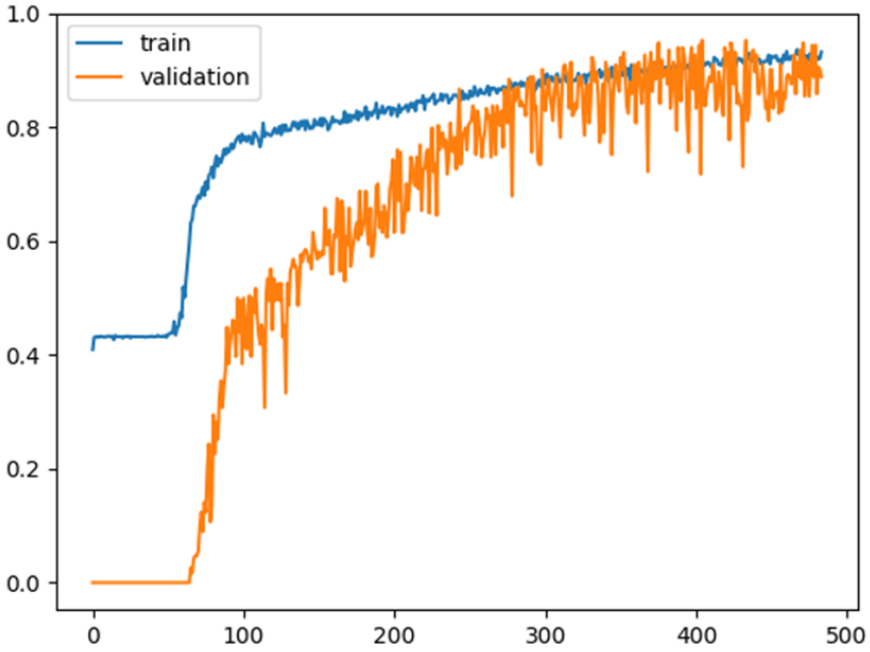
Należy zauważyć, że warstwa droput ma zastosowanie tylko wtedy, gdy dla opcji Trening wybrano wartość Prawda, co oznacza, że podczas wnioskowania nie są odrzucane żadne wartości.

Przeprowadzone testy pozwoliły nam na uzyskanie poniższych wykresów, które prezentują w jaki sposób kolejne utwory podczas trenowania miały wpływ na osiągniętą jakość algorytmu.



Rys. X Wyniki uzyskane dla modelu 2 warstwowego

Jak możemy zauważyć dla modelu 2 warstwowego następuję widoczny wzrost wraz z ilością trenowanych piosenek oraz ich walidacja.



Rys. X Wyniki uzyskane dla modelu 3 warstwowego

Sytacja w ogólnej mierze przybiera podobny obrót sprawy przy algorytmie 3 warstwowym, aczkolwiek tutaj potrzeba nieco większej ilości piosenek do osiągnięcia oczekiwanego rezultatu.

Jako najlepsze rezultaty uzyskane w trakcie walidacji uzyskaliśmy 72% i 87% kolejno dal 2 i 3 warstwowych modeli.

6. Instrukcja użytkownika

1. Muzyka

W celu prawidłowego rozpoczęcia pracy z programem należy pobrać odpowiednie kolejki muzyczne(inczej playlisty) z platformy muzycznej Spotify albo dowolnego innego serwisu, umożliwiającego pobranie żądanych utworów muzycznych w formacie MP3. Zostaną dalej użyte do uczenia systemu oraz dalszych badań zdolności sieci.

2. Puszczenie programu

Po sukcesywnym pobraniu niezbędnych źródeł kolejnym krokiem będzie puszczenie programu. Wszystkie konieczne zmienne są wskazane domyślnie, więc nie potrzebują zmian. Natomiast aby sterować procesem uczenia lub zmieniać kształt wyników, odpowiedne zmienne mogą zostać zmienione.

3. Otzymanie wyniku

Na sam koniec, po sukcesywnym trenowaniu modelu oraz po puszczeniu właściwych testów program generuje wyniki, reprezentowanę tabelą błędów oraz wyliczoną dokładnością *accuracy* do wszystkich badanych sieci neuronowych.

7. Wnioski

Już na pierwszy rzut oka widać, iż otrzymane przez nas wyniki powinny być co najmniej satysfakcjonujące i sieć spełnia swoje pierwotne założenia. Dokładniejsza analiza otrzymanych wyników pozwala stwierdzić, iż wykreowana w trakcie projektu sieć neuronowa radzi sobie w ponadprzeciętny sposób z rozpoznawaniem gatunków muzycznych. Porównując skuteczność do książkowych rezultatów jest ona nawet lepsza niż w opracowaniach. Sytuacja ta może wynikać z faktu, iż trenowane oraz badane były jedynie 3 gatunki muzyczne co w pewien sposób zmniejsza wybór i prawdopodobieństwo błędu. Przykład ten jasno wskazuje na to, iż sieci neuronowe bardzo dobrze radzą sobie z wykrywaniem gatunków muzycznych, co może znajdować swoje zastosowanie w wielu dziedzinach nie tylko tych związanych z rozwojem muzyki, ale także gier czy nawet przemysłu.