Projekt Kompetencyjny I

Raport z realizacji projektu studenckiego

GAN MUSIC

Autor	**Album**	**Ocena**				
		Prezentacja (0,1)	Raport (0,2)	Implementacja (0,5)	Wkład pracy (0,2)	**Końcowa**
Kateryna Tsarova	226451					
Kateryna Ocheretian	226448					

Prowadzący

Tomasz Jaworski, dr inż

Piotr Duch, dr inż.

Notatka prowadzącego:

Łódź, 10/02/2021

Spis treści

Wstęp

W dzisiejszych czasach w Internecie istnieje taki problem jak demonetyzacja wideo, jeżeli ktoś chciałby skorzystać z muzyki innych osób. Nie każdy jest kompozytorem, żeby móc tworzyć muzykę i nie każdy ma pieniądze, żeby móc pozwolić sobie kupić tą muzykę w firmach muzycznych. Dlatego powstał pomysł zbudowania sieci, która będzie generować muzykę, do wykorzystania na portalach Internetowych, takich jak YouTube i innych, lub dodania tych utworów do napisanych przez siebie gier.
Dlatego powstał taki pomysł na napisanie programu, który mógłby na podstawie zestawu piosenek stworzyć nową, podobną do całego zestawu. W tym projekcie zaimplementowano kilka modeli sieci, żeby znaleźć, który z nich lepiej daje sobie radę z wygenerowaniem utworów bardzo podobnych do utworów stworzonych przez człowieka.

Stan wiedzy

Na dany moment istnieje dużo różnych sieci do tworzenia muzyki, na przykład:
CNN, GAN,

Najbardziej znane modele sieci neuronowych do generowania muzyki zaproponowane zostały przez zespół Google Brain to modele MelodyRNN. Oni zaproponowali trzy modele oparte na RNN, w tym dwa warianty ze strukturami długoterminowymi. Kod źródłowy i modele są publiczne dostępnie dla wszystkich.

Muzyka od PI [1] to hierarchiczny model RNN, który wykorzystuje hierarchię powtarzających się warstw do generowania nie tylko melodii, ale także perkusji i akordów, co prowadzi do wielościeżkowych piosenki muzyki pop. Ten model ładnie pokazuje zdolność RNN do generowania wielu sekwencji jednocześnie. Ale wymaga to znajomości teorii muzyki, czego nie potrzebują inne biblioteki jak MidiNet [3].

DeepBach, zaproponowany przez Sony CSL, jest specjalnie zaprojektowany do tworzenia czterogłosowej muzyki chorałowej podobnej do muzyki kompozytora J. S. Bacha. Jest to model oparty na RNN, który umożliwia zrealizowanie ograniczeń zdefiniowanych przez użytkownika, takich jak rytm, nuty, partie i akordy.

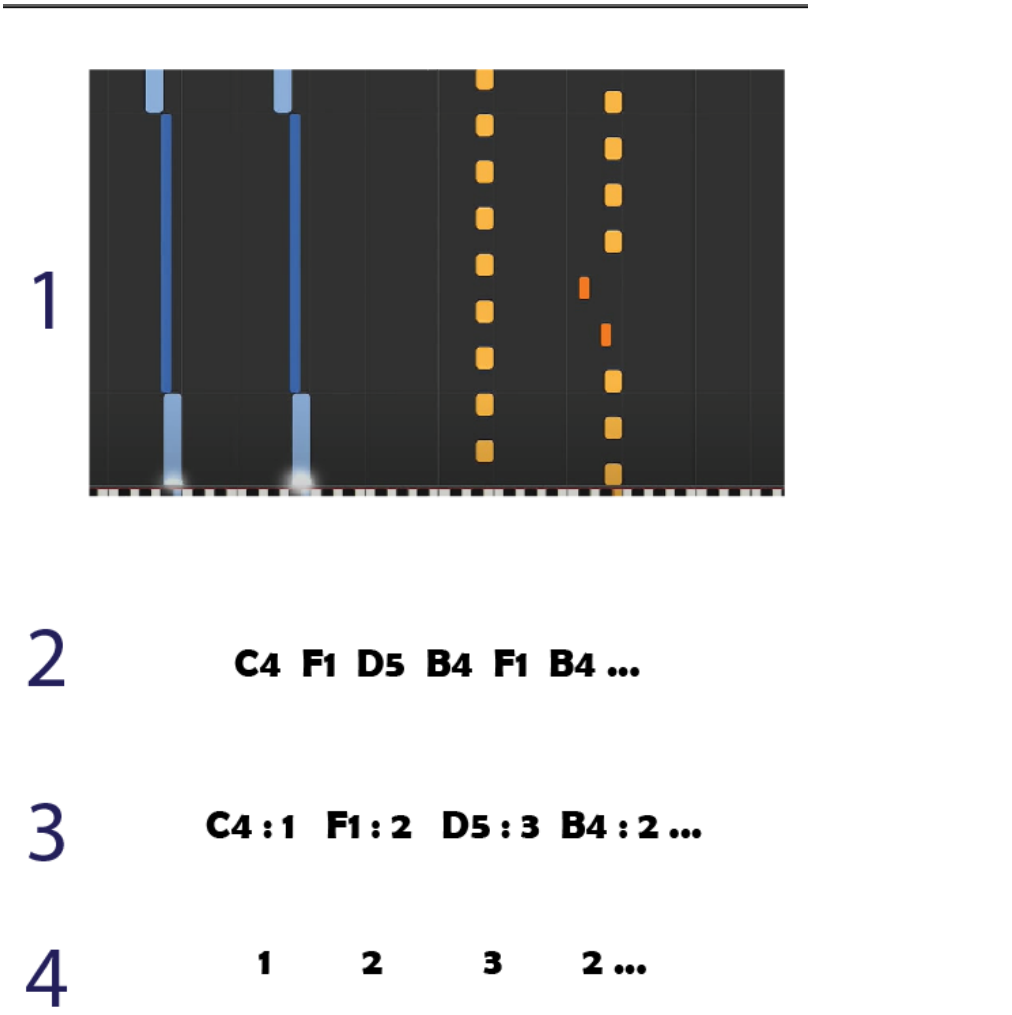
C-RNN-GAN jest jak dotąd jedynym modelem, który wykorzystuje GAN do generowania muzyki. Wykorzystuje on przypadkowe dźwięki jako dane wejściowe, jak i MidiNet, aby wygenerować różnorodne melodie.

W naszym projekcie są porównywane różne sieci GAN do tworzenia muzyki, w tym, gdzie generator to perceptron wielowarstwowy. Celem naszego projektu było znalezienie najlepszego GAN do tworzenia muzyki.

Opis projektu

Baza danych sieci to zestaw piosenek różnej tematyki: jest zestaw piosenek z szybkim tempem i wesołej melodii, oraz zestaw hymnów narodowych. Każdy zestaw zawiera po 200 piosenek.

Do programu utwory są podawane w formacie midi. Dalej za pomocą biblioteki Music21 nuty i odstępy czasowe nut, które wczytaliśmy w postaci stringów, były przekształcone w nuty w postaci liczbowej, które później podawane są na wejście sieci.



Rysunek 1. schemat przetwarzania muzyki

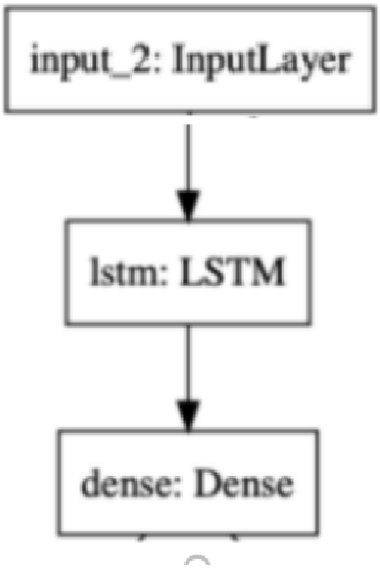
Przetwarzanie danych

Na rysunku 1 po kolei są zilustrowane etapy przetwarzania muzyki do postaci ciągów liczbowych, które później są przekazywane na wejście do sieci:
1 – mamy na wejściu plik midi,
2 – za pomocą biblioteki Music21 wczytana jest każda nuta w postaci stringu,
3 – definicja słownika, do którego wpisują się wszystkie unikalne nuty i wartość nut (jest unikalna dla każdej innej nuty),
4 – za pomocą słownika, kodowana jest sekwencja nut w postaci liczbowej, i potem ten ciąg liczb jest dzielony na sekwencje (po 100 nut).

Tak samo wygląda przetwarzanie danych wejściowych razem z odstępem czasowym dla sieci 3. Odwrotny algorytm przetwarzania danych jest wykonywany na danych wyjściowych z sieci na format plików midi.

Projekt składał się z realizowania, zaimplementowania i porównywania dwóch sieci GAN:

1 sieć - sieć z generatorem - perceptronem wielowarstwowym



Rysunek 2 Model dyskryminatora dla sieci 1

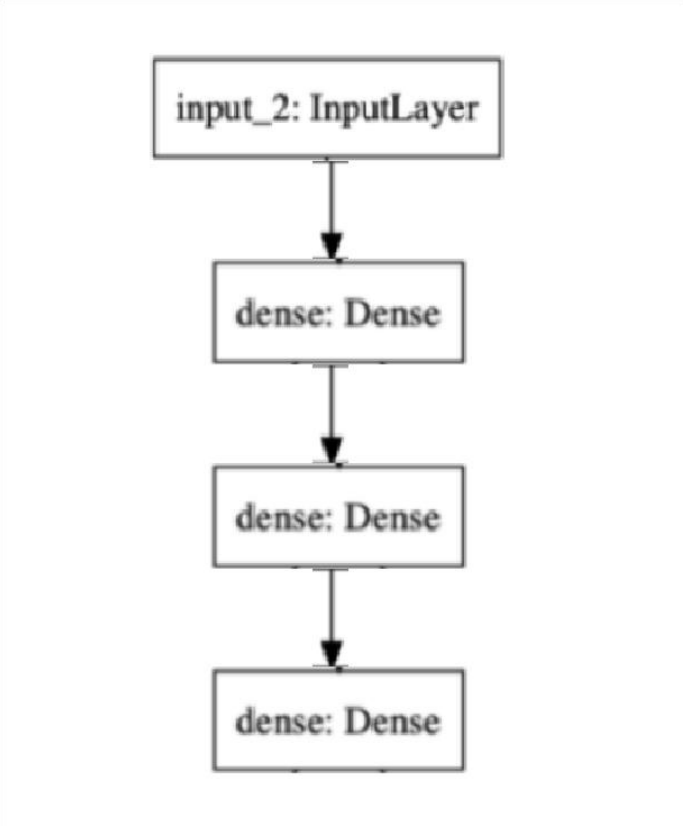
```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 100, 512)          1052672
_________________________________________________________________
bidirectional_2 (Bidirection (None, 1024)              4198400
_________________________________________________________________
dense_4 (Dense)              (None, 512)               524800
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               131328
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 257
=================================================================
Total params: 5,907,457
Trainable params: 5,907,457
Non-trainable params: 0
```

Rysunek 3. Warstwy dyskryminatora sieci 1

1) Dyskryminator (rys 2)
Jest on siecią LSTM.
Każda warstwa modelu ma dokładnie jeden tensor wejściowy i jeden tensor wyjściowy - jest to model sekwencyjny.
input_shape warstwy wejściowej ma rozmiar (None, 100)

None – *batch_size,*
100 – sekwencja nut w formacie liczbowym.

Warstwa wyjściowa dyskryminatora (rys 3) ma jeden neuron, który wskazuje czy utwór muzyczny był stworzony przez człowieka czy przez sieć. Mamy takie ukryte warstwy: *LSTM, Bidirectional LSTM, Dense, LeakyReLu.*
Do wytrenowania modelu była wykorzystana funkcja straty *binary_crossentropy*
Był także wykorzystany algorytm optymalizacji ADAM *(ang. adaptive moment estimation).*



Rysunek 4. Model generatora dla sieci 2

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 256)               256256
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
dense_8 (Dense)              (None, 512)               131584
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
batch_normalization_2 (Batch (None, 512)               2048
_________________________________________________________________
dense_9 (Dense)              (None, 1024)              525312
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
batch_normalization_3 (Batch (None, 1024)              4096
_________________________________________________________________
dense_10 (Dense)             (None, 100)               102500
_________________________________________________________________
reshape_1 (Reshape)          (None, 100, 1)            0
=================================================================
Total params: 1,022,820
Trainable params: 1,019,236
Non-trainable params: 3,584
```

Rysunek 5. Warstwy generatora sieci 1

2) Generator (rys 4)
Każda warstwa modelu ma dokładnie jeden tensor wejściowy i jeden tensor wyjściowy - jest to model sekwencyjny:
input_shape warstwy wejściowej ma rozmiar (None, 100)

None – *batch_size,*
100 – randomowe liczby.

Melodie wygenerowane przez daną sieć są bardzo podobne do siebie, co jest wielkim minusem, bo wszystko brzmi nudnie i nieciekawie.
Warstwa wyjściowa generatora (rys 5) ma rozmiar (100, 1) - na wyjściu sieć generuje sekwencje składającą się ze stu nut. Mamy takie ukryte warstwy: *Dense, LeakyReLu, BatchNormalization.*
Do wytrenowania modelu była wykorzystana funkcja straty *binary_crossentropy.*
Był także wykorzystany algorytm optymalizacji ADAM *(ang. adaptive moment estimation).*

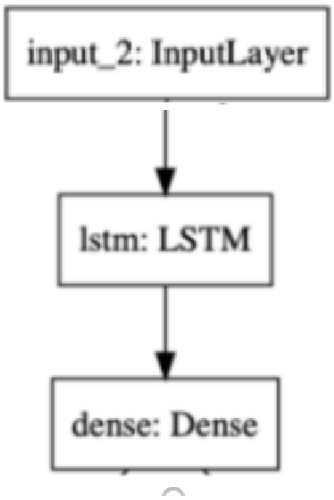
3) Uczenie sieci
Do trenowania modelu były wybrane takie parametry:
- *epoch* – 10000,
- *batch_size* – 32.

Czas uczenia się sieci – 1 godzina.
Dokładność sieci – 32%.

Wyniki końcowe dla dwóch zestawów utworów(szybkie piosenki i hymny) były prawie jednakowe, przez to że sieć źle uczy się i jest słabszą od sieci 2. W generowanych piosenkach odstępy czasowe między nutami były jednakowe

Dany model sieci nie zadowala naszych potrzeb i nie generuje muzyki podobnej do tej, którą tworzą ludzie, dlatego postanowiono nie rozwijać dalej tej sieci i spróbować napisać sieć LSTM.

2 sieć - sieć z generatorem - siecią LSTM

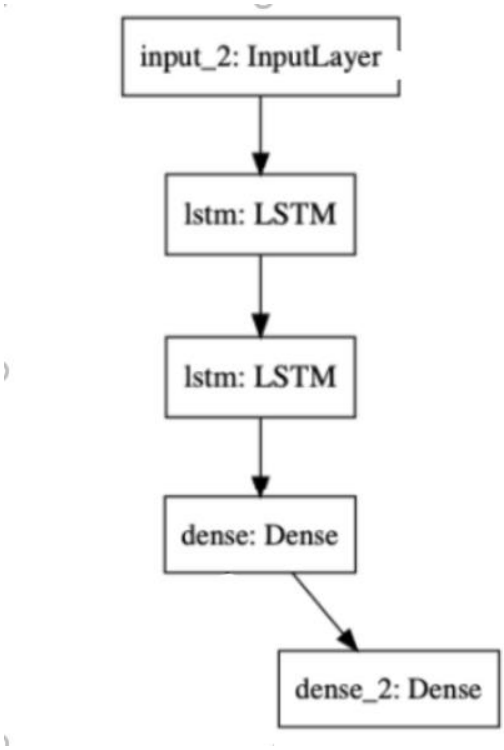


Rysunek 6. Model dyskryminatora dla sieci 2

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 100, 512)          1052672
_________________________________________________________________
bidirectional_2 (Bidirection (None, 1024)              4198400
_________________________________________________________________
dense_4 (Dense)              (None, 512)               524800
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               131328
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 257
=================================================================
Total params: 5,907,457
Trainable params: 5,907,457
Non-trainable params: 0
```

Rysunek 7. Warstwy dyskryminatora sieci 2

1) Dyskryminator (rys 6 i 7)
Jest taki sam jak dyskryminator sieci 1.



Rysunek 8. Model generatora dla sieci 2

Model: "sequential_12"

Layer (type)	Output Shape	Param #
bidirectional_6 (Bidirection	(None, 1000, 512)	612352
dropout_13 (Dropout)	(None, 1000, 512)	0
lstm_21 (LSTM)	(None, 1000, 128)	328192
dropout_14 (Dropout)	(None, 1000, 128)	0
lstm_22 (LSTM)	(None, 64)	49408
dropout_15 (Dropout)	(None, 64)	0
dense_41 (Dense)	(None, 256)	16640
leaky_re_lu_31 (LeakyReLU)	(None, 256)	0
batch_normalization_13 (Batc	(None, 256)	1024
dense_42 (Dense)	(None, 512)	131584
leaky_re_lu_32 (LeakyReLU)	(None, 512)	0
batch_normalization_14 (Batc	(None, 512)	2048
dense_43 (Dense)	(None, 1024)	525312
leaky_re_lu_33 (LeakyReLU)	(None, 1024)	0
batch_normalization_15 (Batc	(None, 1024)	4096
dense_44 (Dense)	(None, 100)	102500
reshape_5 (Reshape)	(None, 100, 1)	0

Total params: 1,773,156
Trainable params: 1,769,572
Non-trainable params: 3,584

Rysunek 9. Warstwy generatora sieci 2

2) Generator (rys 8)
Jest on siecią LSTM.
Każda warstwa modelu ma dokładnie jeden tensor wejściowy i jeden tensor wyjściowy - jest to model sekwencyjny:
input_shape warstwy wejściowej ma rozmiar (None, 1000),
None – *batch_size,*
1000 – randomowe liczby.
Warstwa wyjściowa generatora ma rozmiar (100, 1) - na wyjściu sieć generuje sekwencje składającą się ze stu nut. Mamy takie ukryte warstwy: *LSTM, Dropout, Bidirectional LSTM, Dense, LeakyReLu, BatchNormalization.*

Do wytrenowania modelu była wykorzystana funkcja straty *binary_crossentropy.*
Był także wykorzystany algorytm optymalizacji ADAM *(ang. adaptive moment estimation).*

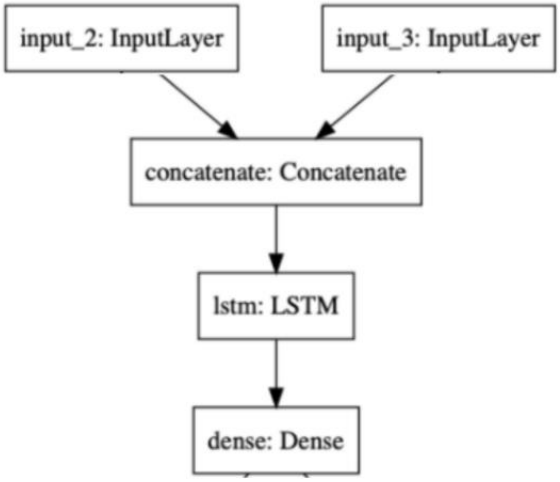
3) Uczenie sieci
Do trenowania modelu były wybrane takie parametry:
- *epoch* – 10000,
- *batch_size* – 32.

Czas uczenia się sieci – 8 godzin.
Dokładność sieci – 48%.

Melodie wygenerowane przez tą sieć brzmią żywiej i można odróżnić piosenki wygenerowane na podstawie zestawu z piosenkami szybkimi i wesołymi od piosenek wygenerowanych na dostawie zestawu hymnów. W pierwszych można wyraźnie usłyszeć wysokie nuty i szybkie tempo, a w drugich jest odpowiedni rytm podobny do rytmu w hymnach.

W tym modelu widzimy potencjał do dalszego rozwoju i postanowiliśmy dalej rozwijać ten model.

3 sieć - ulepszona sieć z generatorem - siecią LSTM



Rysunek 10. Model dyskryminatora dla sieci 3

Model: "model_1"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 100, 1)	0	
input_2 (InputLayer)	(None, 100, 1)	0	
concatenate_1 (Concatenate)	(None, 200, 1)	0	input_1[0][0] input_2[0][0]
lstm_1 (LSTM)	(None, 200, 512)	1052672	concatenate_1[0][0]
bidirectional_1 (Bidirectional)	(None, 1024)	4202496	lstm_1[0][0]
dense_1 (Dense)	(None, 512)	524800	bidirectional_1[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 512)	0	dense_1[0][0]
dense_2 (Dense)	(None, 512)	262656	leaky_re_lu_1[0][0]
leaky_re_lu_2 (LeakyReLU)	(None, 512)	0	dense_2[0][0]
dense_3 (Dense)	(None, 1)	513	leaky_re_lu_2[0][0]

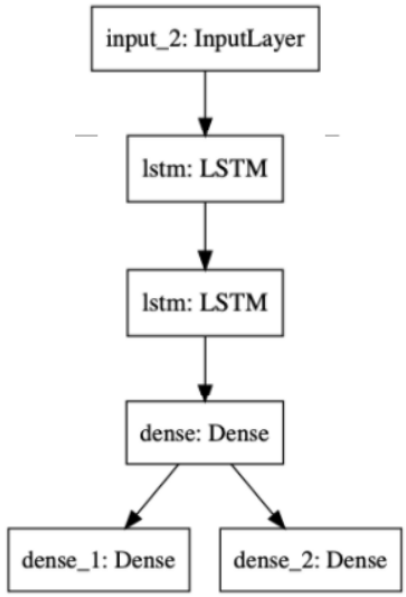
Total params: 6,043,137
Trainable params: 6,043,137
Non-trainable params: 0

Rysunek 11. Warstwy dyskryminatora sieci 3

1) Dyskryminator (rys 10)
Jest on siecią LSTM.
Model ma dwie warstwy wejściowe - osobno dla wprowadzenia do sieci nut i odstępu czasowego nut.
input_shape warstwy wejściowej ma rozmiar (None, 100):

None – *batch_size,*
100 – sekwencja nut.

Warstwa wyjściowa dyskryminatora (rys 11) ma jeden neuron, który wskazuje czy utwór muzyczny był stworzony przez człowieka lub przez sieć. Mamy takie ukryte warstwy: *LSTM, Bidirectional LSTM, Dense, LeakyReLu.*
Do wytrenowania modelu była wykorzystana funkcja straty *binary_crossentropy*
Był także wykorzystany algorytm optymalizacji ADAM *(ang. adaptive moment estimation)*



Rysunek 12. Warstwy dyskryminatora sieci 3

Model: "model_3"

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 100, 100)	0	
cu_dnnlstm_2 (CuDNNLSTM)	(None, 100, 512)	1257472	input_3[0][0]
dropout_1 (Dropout)	(None, 100, 512)	0	cu_dnnlstm_2[0][0]
batch_normalization_1 (BatchNor	(None, 100, 512)	2048	dropout_1[0][0]
cu_dnnlstm_3 (CuDNNLSTM)	(None, 256)	788480	batch_normalization_1[0][0]
dropout_2 (Dropout)	(None, 256)	0	cu_dnnlstm_3[0][0]
batch_normalization_2 (BatchNor	(None, 256)	1024	dropout_2[0][0]
dense_4 (Dense)	(None, 256)	65792	batch_normalization_2[0][0]
dense_5 (Dense)	(None, 100)	25700	dense_4[0][0]
dense_6 (Dense)	(None, 100)	25700	dense_4[0][0]
reshape_1 (Reshape)	(None, 100, 1)	0	dense_5[0][0]
reshape_2 (Reshape)	(None, 100, 1)	0	dense_6[0][0]

Total params: 2,166,216
Trainable params: 2,164,680
Non-trainable params: 1,536

Rysunek 13. Warstwy dyskryminatora sieci 3

2) Generator (rys 12)
Jest on siecią LSTM.
Model ma dwie warstwy wyjściowe - osobno dla generacji nut i odstępu czasowego nut.
input_shape warstwy wejściowej ma rozmiar (None, 100, 100):

None – *batch_size,*
(100, 100) – randomowe liczby.

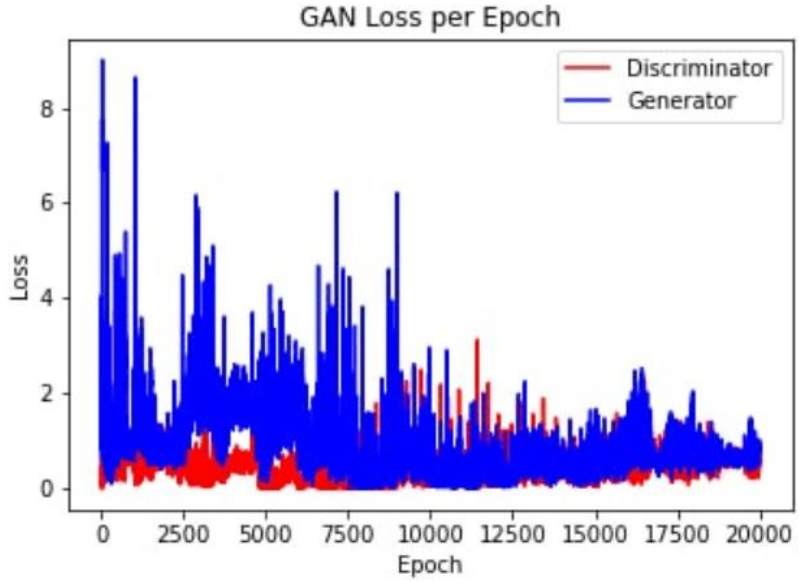
Warstwa wyjściowa generatora (rys 13) ma rozmiar (100, 1) - na wyjściu sieć generuje sekwencje składającą się ze stu nut. Mamy takie ukryte warstwy: *CuDNNLSTM, Dropout, Dense, BatchNormalization.*
Do wytrenowania modelu była wykorzystana funkcja straty *binary_crossentropy.*
Był także wykorzystany algorytm optymalizacji ADAM *(ang. adaptive moment estimation).*

3) Uczenie sieci
Do trenowania modelu były wybrane takie parametry:
- *epoch* – 30000,
- *batch_size* – 32.

Czas uczenia się sieci – 3 dni.
Dokładność sieci – 67%.

W tym modelu uzyskaliśmy najlepsze wyniki ze wszystkich sieci. W tym modelu również został dodany odstęp czasowy między nutami przez co muzyka brzmi bardziej głęboko i ciekawie. To jest nasz ostateczny model, który przedstawiamy i jesteśmy zadowoleni z wyników.

Z wykresu poniżej (rys 14) można zobaczyć to, że ze zwiększeniem epok, dana sieć uczy się lepiej i generuje lepsze wyniki. Największa liczba epok na której trenowałyśmy sieć to 30 000 epok i uczenie sieci trwało 3 dni.



Rysunek 14. Wykres strat dla generatora i dyskryminatora

Użyte technologie

W tym projekcie zostało użyte takie technologii jak język programowania Python i biblioteki TensorFlow i music21 w środowisku Jupiter.

Wnioski oraz doświadczenia

Podczas realizacji projektów udało się stworzyć dwa skuteczne modele sieci GAN i otrzymać wyniki w postaci muzyki z tych sieci. Po porównaniu wygenerowanych piosenek można stwierdzić, że najlepiej uczy się 3 sieć: generuje muzykę podobną do zestawu podawanego na wejście, zawiera różne odstępy czasowe między nutami i brzmi jako prawdziwa piosenka.

W zakończeniu tego projektu my zdobyłyśmy i pogłębiłyśmy wiedzę o sieciach neuronowych, rodzajach sieci GAN, i udało się otrzymać wyniki na różnych ilościach epok.

Praca była podzielona tak:

- Kateryna Tsarova stworzyła ii rozbudowywała sieć
- Kateryna Ocheretian przygotowywała muzykę która była inputem dla programu, napisała kod konwertujący tą muzykę na dane wejściowe do sieci i kod konwertujący wyniki sieci na utwory muzyczne

Literatura

Wykorzystywane strony internetowe

1. https://iapp.org/news/a/when-is-ai-pi-how-current-and-future-privacy-laws-implicate-ai-and-machine-learning/
2. https://arxiv.org/abs/1703.10847
3. https://en.wikipedia.org/wiki/Long_short-term_memory
4. https://en.wikipedia.org/wiki/Generative_adversarial_network
5. https://pl.wikipedia.org/wiki/Perceptron_wielowarstwowy