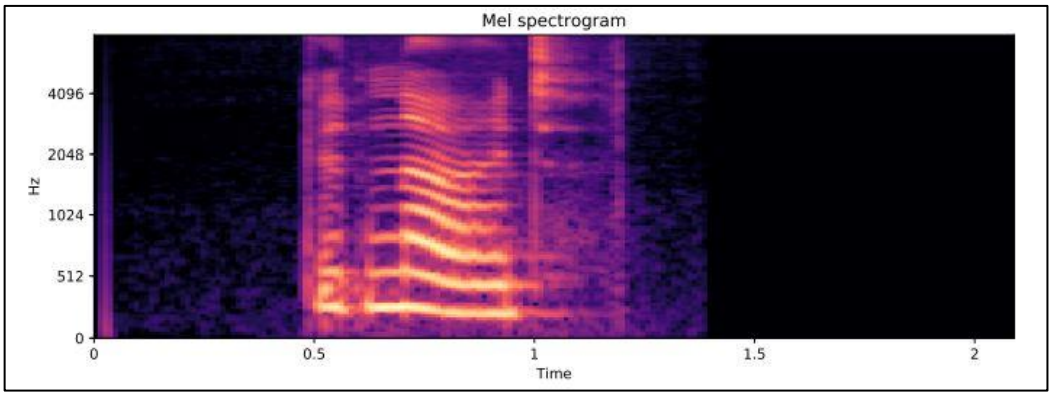
BAB 2

LANDASAN TEORI

2.1 Representasi Audio

Signal audio merupakan representasi umum pada data audio, peneliti berasumsi bahwa konten musik tersimpan dalam bentuk signal audio digital. Namun signal audio tidak menjadi pilihan yang populer dalam merepresentasikan data audio. Peneliti lebih memilih representasi 2D seperti *Short-Time Fourier Transform* (STFT) dan *Mel-Spectograms* dikarenakan pembelajaran pada *neural network* menggunakan signal audio membutuhkan *dataset* yang lebih besar (Choi, dkk., 2018).

STFT merupakan representasi frekuensi waktu dengan frekuensi pusat yang berjarak linear. Komputasi STFT lebih cepat dibandingkan representasi frekuensi waktu yang lain karena menggunakan *Fast Fourier Transform* (FFT). Namun STFT tidak menjadi representasi pilihan karena tidak cocoknya resolusi frekuensi sistem pendengaran manusia dan juga ukurannya tidak efisien. Sehingga dipilihlah representasi 2D *Mel-Spectrogram* karena telah dioptimalkan untuk persepsi pendengaran manusia dan juga ukurunnya yang lebih efisien (Choi, dkk., 2018). Contoh representasi *mel-spectrogram* dapat dilihat pada gambar 2.1



Gambar 2.1 Mel-Spectrogram

Sumber: (Eklund, 2019)

2.2 Artificial Neural Network (ANN)

ANN merupakan sebuah model arithmetical yang didesain menyerupai jaringan syaraf biologis pada otak manusia (Salah, dkk., 2018). Sistem ini bersifat adaptif karena dapat mengubah struktur dalam memecahkan masalah berdasarkan eksternal maupun internal (Noviando, dkk., 2016). struktur dari *neural network* terdiri dari unit atau *node* sebagai penyusunnya. *Node* dianalogikan seperti *neuron* yang merepresentasikan nilai skalar dan kumpulan dari *node* ini disebut dengan layer (Choi, dkk., 2018).

Layer-layer yang terdapat pada ANN pada umumnya dibagi ke dalam beberapa peranan seperti.

1. *Input* layer: seperti namanya, layer ini berfungsi dalam merepresentasikan data yang akan dimasukkan ke dalam jaringan.
2. *Hidden* layer: layer ini berfungsi untuk melakukan kalkulasi berdasarkan nilai *input* yang diberikan.
3. *Output* layer: layer ini merupakan layer yang terakhir sekaligus mengeluarkan hasil akhir dari berdasarkan kalkulasi yang dilakukan sebelumnya.

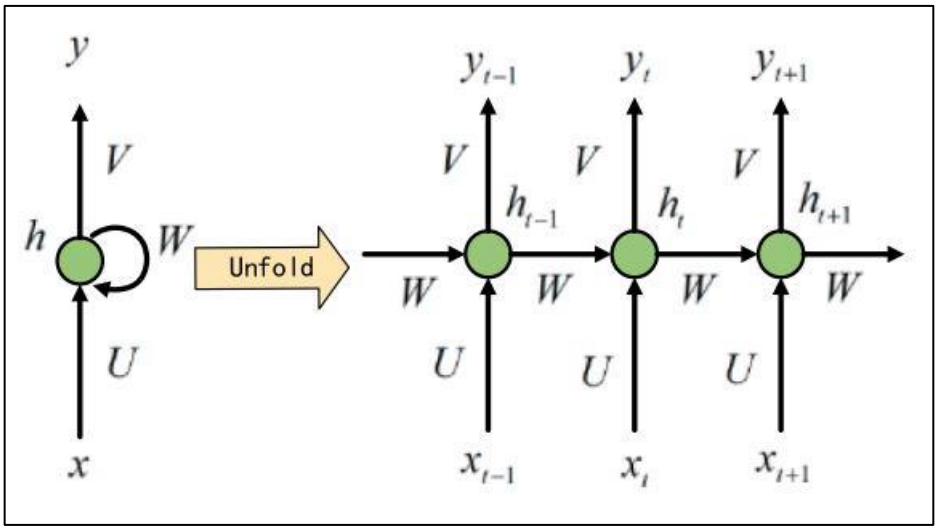
Cara kerja dari *neural network* dimulai dari data yang direpresentasikan melalui *input* layer yang saling terhubung dengan unit pada *hidden* layer dan nilai yang diberikan akan dikalkulasi. Kalkulasi pada unit di *hidden* layer dilakukan dengan persamaan (2, 1):

$$y = f(b + \sum_{i=1}^{n} w_i * x_i) \tag{2.1}$$

Setiap nilai *input* saling berasosiasi dengan nilai *weight* yang normalnya antara 0 dan 1. Penjumlahan menggunakan *input* tambahan berupa nilai **b** yaitu 1 yang merupakan bias pada unit. **w** merupakan *weight* dan **x** merupakan *input* masukkan (Nimbark, dkk., 2014). Kemudian setelah perhitungan, muncul sebuah konsep yang bernama *Activation Function*. fungsinya membuat *network* menjadi non-linear dan memungkinkan untuk mempelajari pola yang rumit (Choi, dkk., 2018). Hasil setelah masuk pada *Activation Function* akan menjadi *input* pada layer selanjutnya sampai pada layer *output*.

2.3 Deep Recurrent Neural Network (DRNN)

RNN adalah *neural network* dengan alur *feed-backward* yang memiliki *hidden* layer yang terhubung dengan dirinya sendiri (Nayyar, dkk., 2017). Konsep umumnya RNN melakukan komputasi yang sama pada setiap elemen yang berurutan dan bergantung dari hasil komputasi sebelumnya (Xu, dkk., 2018). Dengan kemampuannya tersebut, RNN memungkinkan untuk bekerja menggunakan data *sequential* karena bisa mempelajari korelasi antara perbedaan *time step* dengan baik, salah satunya yaitu dengan data musik dan juga memiliki kemampuan dalam manipulasi relasi jangka panjang (Song, dkk., 2018; Wu, dkk., 2018).



Gambar 2.2 Arsitektur RNN

Sumber: (Chen, dkk., 2019)

RNN secara umum bekerja dengan cara *input* data *sequential* yang masuk akan diproses menuju ke dalam *hidden state*. Setelah itu *hidden state* akan melakukan kalkulasi menggunakan persamaan (2.1).

$$h_t = f(Wh_{t-1} + Ux_t) \tag{2.2}$$

Hidden state dikalkulasi dengan menggunakan *hidden state* sebelumnya yang dilambangkan h_{t-1} dan *input* pada waktu sekarang yang dilambangkan dengan x_t. hasil kalkulasi kemudian dikonversi menggunakan *hyperbolic tangent activation function* sehingga nilai menjadi *non-linear* (Xu, dkk., 2018). Kemudian hasil *output* pada *hidden state* akan diteruskan ke layer yang lain seperti *output* layer dan juga hasil tersebut akan masuk kembali dalam *hidden state*. Hal inilah yang menjadi *memory* bagi RNN dari kalkulasi yang dilakukan sebelumnya untuk dipakai lagi pada kalkulasi di *hidden state* pada *input* selanjutnya.

Pada kasus tertentu dengan strukur RNN yang menggunakan *single* layer memberikan performa yang tidak memuaskan. Maka dari itu, struktur dari RNN dibuat lebih dalam yaitu dengan menambahkan beberapa layer. Hal ini bertujuan

untuk meningkatkan performa pada RNN dalam melakukan *feature extraction*. Caranya dengan menumpukkan *hidden* layer di atas satu sama lain yaitu memakai *hidden* layer dari lapisan terakhir sebagai *input* layer (Song, dkk., 2018).

2.4 Gated Recurrent Unit (GRU)

Semakin banyaknya layer yang digunakan pada RNN, mengantarkan pada sebuah permasalahan yaitu *vanishing gradient* dimana ketidakmampuan untuk melakukan optimisasi pada layer sebelumnya karena gradien yang semakin mengecil mendekati nol. Hal ini berdampak pada *neural network* menjadi tidak efisien lebih tepatnya *weight* dan *bias* pada layer-layer sebelumnya tidak terupdate pada setiap *training session*. Mengakibatkan layer pada *neural network* tidak belajar dan memiliki *short-term memory* karena pembelajaran yang dilakukan sebelumnya bisa dilupakan. Untuk mengatasi permasalahan ini, muncul sebuah solusi dengan menerapkan modul pemrosesan yang lebih canggih yaitu menggunakan *Gated Recurrent Unit* (GRU).

GRU merupakan sktruktur variasi dari *gated unit* yang bisa mempelajari relasi jangka panjang selama proses *training* (Song, dkk., 2018). GRU memperkenalkan *hidden unit* yang baru terinspirasi dari LSTM dimana GRU mengkombinasikan *forget gate* dan *input gate* menjadi *single update gate*, GRU juga mereduksi 4 *gate* pada LSTM menjadi 2 yang dinamai *update gate* dan *reset gate* sehingga model menjadi lebih simple (Chen, dkk., 2019). *Reset gate* memungkinkan untuk mengontrol seberapa banyak dari informasi yang lama untuk dilupakan. Sedangkan *update gate* memungkinkan untuk mengontrol seberapa banyak informasi yang sudah lewat dibutuhkan untuk disimpan lebih lama.

Reset gate dan *update gate* menggunakan sebuah perhitungan untuk menentukan bagaimana mereka melupakan informasi dan memperbaharui informasi yang disimpan dengan persamaan (3) dan (4):

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1}) \quad (2.3)$$

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1}) \quad (2.4)$$

Dengan z_t merupakan *update gate* dari waktu yang sekarang dan r_t adalah *reset gate* dari waktu yang sekarang. Sekilas rumus yang digunakan sama namun terdapat perbedaan pada *weight* yang digunakan dimana *W* dan *U* merupakan *weight* yang berasosiasi dengan masing-masing *gate* dan *index* pangkat mengindikasikan *weight* pada tiap *gate.* Kemudian x_t merupakan *input* yang masuk pada waktu yang sekarang dan h_{t-1} adalah *hidden state* yang menyimpan informasi pada waktu sebelumnya. Setelah melakukan perkalian dan penjumlahan, hasil akhirnya akan dikonversi menggunakan *non-linear activation function* sigmoid. Hasil akhir akan menentukan pada *reset gate* dan *update gate* jika hasil mendekati 1 maka *reset gate* tidak akan menghapus informasi sebelumnya dan *update gate* akan mempertahankan informasi yang sekarang dan jika mendekati 0 maka *reset gate* akan menghapus yang lama dan fokus pada yang sekarang kemudian *update gate* tidak akan mempertahankan informasi tersebut.

2.5 Data Augmentation

Pada umumnya, melatih model *deep learning* membutuhkan data yang sangat banyak untuk meningkatkan perkiraan statistik dalam menyelesaikan permasalahan. Namun mengumpulkan dan melabelkan data membutuhkan waktu dan tenaga yang sangat banyak. Untuk mengatasi permasalahan tersebut dengan

melakukan augmentasi data. Augmentasi dapat melakukan manipulasi data dengan membuat data tambahan dari yang sebelumnya sudah ada dan diubah menjadi bentuk yang lain. Penambahan data ini untuk mencegah model dari *overfitting* (Bhardwaj, 2017).

Beberapa teknik yang bisa diterapkan pada augmentasi audio adalah sebagai berikut.

1. *Pitch Scaling*

   Teknik augmentasi ini melakukan transformasi pada signal audio dengan mengubah *pitch* dari *input* audio tanpa mengubah durasinya.

2. *Time Stratching*

   Teknik augmentasi ini melakukan transformasi dengan mengubah durasi dari signal audio tanpa mengubah *pitch* dari audio tersebut.

3. *Gaussian Noise*

   Teknik augmentasi ini melakukan transformasi dengan menjumlahkan *noise* dengan signal asli audio. Penggunaan *Gaussian Noise* terbukti meningkatkan performa generalisasi pada masalah regresi dan klasifikasi (Eklund, 2019).

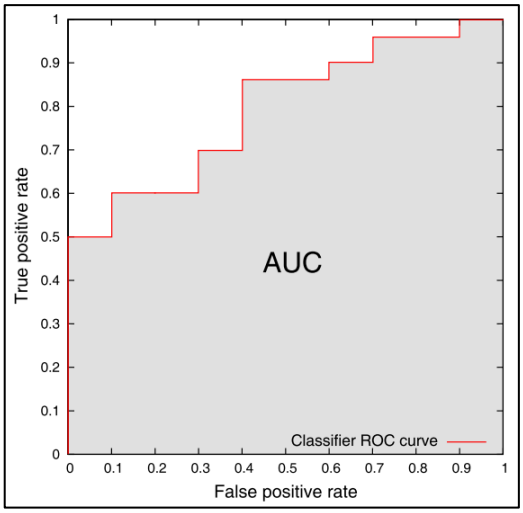
4. *Shift*

   Teknik augmentasi ini melakukan transformasi dengan menggeser signal audio lebih maju atau mundur.

2.6 Evaluasi

Hasil dari performa model DRNN akan dievaluasi menggunakan AUC-ROC dan AUC-PR score. ROC akan membandingkan performa *classifier* pada semua

distribusi kelas dengan membentuk kurva yang memvisualisasikan hubungan antara *true positive rate* atau *recall* dan *false positive rate* di semua *thresholds*.



Gambar 2.3 Kurva ROC

Sumber: (Brzezinski and Stefanowski, 2017)

Kemudian AUC akan mengartikan relasi yang terbentuk ke dalam satuan skalar dengan menghitung area di bawah kurva ROC (Brzezinski and Stefanowski, 2017). Untuk menghitung *true positive rate* dapat menggunakan persamaan (2.5).

$$TPR = \frac{TP}{TP+FN} \tag{2.5}$$

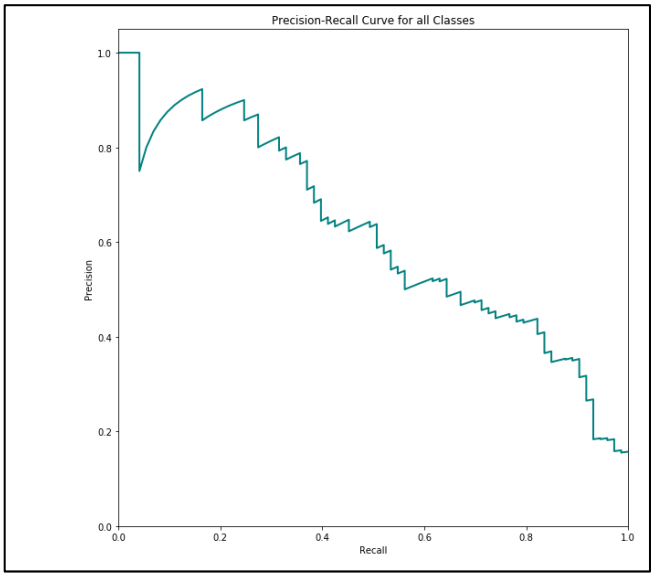
Untuk menghitung *false positive rate* dapat menggunakan persamaan (2.6).

$$FPR = \frac{FP}{TN+FP} \tag{2.6}$$

Perbedaannya dengan AUC-PR dimana evaluasi ini akan memetakan antara *positive predicted value* (PPV) atau *precision* dengan *True positive rate* (TPR) menjadi sebuah kurva. Kemudian hasil visualisasi kurva akan diringkas menjadi bentuk skalar dengan menghitung area di bawah kurva tadi di semua *threshold*.

Untuk menghitung *precision* dapat menggunakan persamaan (2.7)

$$Precision = \frac{TP}{TP+FP} \tag{2.7}$$



Gambar 2.4 Kurva precision-recall