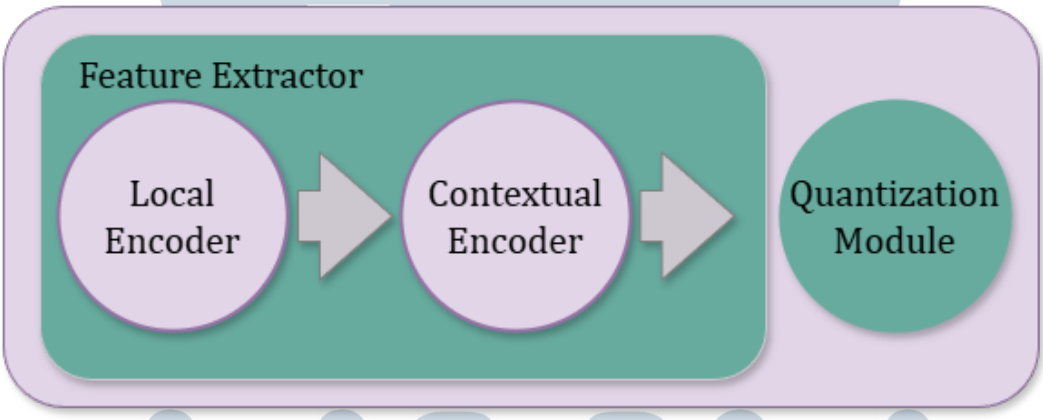
BAB 2
TINJAUAN PUSTAKA

2.1 Wav2vec 2.0

Wav2vec 2.0 merupakan sebuah kerangka kerja pembelajaran *self-supervised learning* yang mempelajari fitur ucapan dari audio mentah dengan cara melakukan *masking* pada sebagian fitur audio (*latent representation*) dan menyelesaikan tugas kontrasif.

Versi 2.0 dari wav2vec melalui tiga tahapan dalam alur pemrosesannya, seperti yang ditunjukkan pada Gambar 2.1:



Gambar 2.1. Pipeline wav2vec 2.0

Tahap pertama adalah *local encoder* (atau *feature encoder*), yang dibangun dari beberapa *convolutional block* untuk mengodekan *input* audio mentah menjadi sebuah urutan embedding. Setiap 20 ms, encoder menghasilkan satu embedding dengan *receptive field* sebesar 25 ms.

Tahap berikutnya adalah *contextual encoder*. Pada tahap ini, representasi dari *local encoder* digunakan sebagai input dan diteruskan menuju *context network* yang terdiri dari beberapa blok transformer, serta sebuah *convolutional layer* yang berfungsi sebagai *relative positional embedding* untuk mengodekan informasi posisi. *Contextual encoder* menghasilkan embedding yang *context aware*.

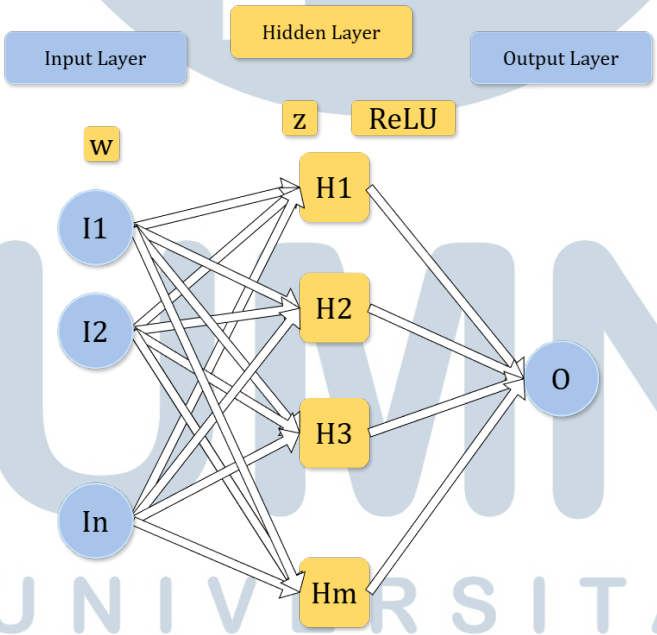
Tahap terakhir adalah modul *quantization* yang berfungsi untuk mendiskritisasi fitur audio kontinu yang dihasilkan dari representasi *local encoder*. Modul ini memiliki dua *codebook* yang masing-masing berisi 320 entri. Dari *codebook* tersebut, kode dipilih menggunakan metode Gumbel Softmax.

Kode-kode terpilih kemudian digabungkan, dan vektor hasil penggabungan tersebut dikenai transformasi linear sehingga menghasilkan representasi terkuantisasi dari keluaran *local encoder* yang digunakan dalam fungsi objektif.

Terdapat dua jenis model yang dirilis untuk penggunaan publik. Model pertama adalah model *base* yang memiliki embedding berdimensi 768 dengan 12 blok transformer dan masing-masing 8 *attention heads*. Model kedua adalah model *large* yang memiliki embedding berdimensi 1024 dengan 24 blok transformer dan masing-masing 16 *attention heads* [18], [19], [20].

2.2 Multilayer Perceptron (MLP)

Multilayer Perceptron (MLP) merupakan arsitektur jaringan saraf yang kuat dan umum digunakan untuk tugas klasifikasi. MLP terdiri dari *input layer* yang menerima data, satu atau lebih *hidden layer* yang menangkap pola-pola kompleks, serta *output layer* yang menghasilkan prediksi akhir [21].



Gambar 2.2. Arsitektur MLP

Pada dasarnya, MLP bekerja dengan menerapkan fungsi linear dan memperkenalkan non-linearitas melalui penggunaan activation function. Dengan cara ini, model mampu mempelajari hubungan dan pola yang terdapat dalam data. Seperti ditunjukkan pada Gambar 2.2, setiap neuron pada lapisan-lapisan tersebut

menerapkan bobot w pada input sehingga menghasilkan nilai *pre-activation* z [22], yang selanjutnya akan dikenai activation function.

$$Z = \sum(w \times x) + b \tag{2.1}$$

$$ReLU = \max(0, x) \tag{2.2}$$

Proses pembelajaran MLP melibatkan *feeding data* ke dalam *network*, evaluasi akurasi, serta penyesuaian bobot (*weight*) secara iteratif untuk meminimalkan perbedaan antara hasil prediksi dan nilai sebenarnya. Melalui proses optimisasi dan penyesuaian parameter yang berkelanjutan, MLP secara bertahap meningkatkan performanya dan menjadi semakin efektif dalam mengklasifikasikan data input [23]. Dalam konteks klasifikasi audio deephoax, MLP mempelajari representasi fitur suara untuk membedakan karakteristik audio asli dan audio hasil rekayasa.

2.3 SMOTE

SMOTE atau *Synthetic Minority Oversampling Technique* merupakan metode *oversampling* yang bertujuan untuk mengatasi ketidakseimbangan data dengan menambah jumlah sampel pada kelas minoritas. Berbeda dengan *random oversampling*, SMOTE menghasilkan sampel baru secara sintetis, sehingga lebih efektif dalam menghindari *overfitting* yang sering terjadi ketika *oversampling* dilakukan dengan cara menduplikasi sampel yang sudah ada[24].

SMOTE menggunakan algoritma *k-Nearest Neighbors* (k-NN) untuk mensintesis sampel baru berdasarkan tetangga terdekat dari kelas minoritas. Proses pembuatan sampel sintetis dilakukan melalui dua tahap. Tahap pertama adalah memilih sampel secara acak dari kelas minoritas, dan tahap kedua adalah menerapkan algoritma k-NN untuk menentukan lokasi pembentukan sampel sintetis.

Apabila jumlah sampel yang dibutuhkan lebih besar daripada jumlah sampel yang tersedia pada kelas minoritas, maka seluruh sampel pada kelas minoritas akan dipilih. Sebaliknya, jika jumlah yang dibutuhkan lebih kecil, maka hanya sebagian sampel yang dipilih secara acak. Setelah proses pemilihan, tetangga terdekat dari

sampel acak kelas minoritas yang memiliki hubungan linear dengan sampel terpilih akan ditentukan. Nilai selisih tersebut kemudian dikalikan dengan suatu bobot, dan hasilnya menjadi lokasi dari sampel sintetis yang baru. Secara matematis, representasi SMOTE dapat dijelaskan sebagai berikut:.

$$x_{\text{smote}} = x_i + (\hat{x}_i - x_i) \times \delta, \quad \delta \in [0,1], \quad i = 1,2,\ldots,k \tag{2.3}$$

x merupakan sampel dari kelas minoritas dan x adalah salah satu tetangga acak dari k tetangga terdekat milik x. Untuk memperoleh sampel sintetis x_smote, terlebih dahulu diidentifikasi k tetangga terdekat dari x, kemudian jarak antara x dan tetangganya dihitung dan dikalikan dengan nilai antara 0 dan 1. Proses ini diulang hingga jumlah sampel pada kelas minoritas dan kelas mayoritas menjadi seimbang [25], [26].

2.4 *Pooling*

Pooling adalah teknik yang digunakan untuk mengagregasi embedding token menjadi satu *unified embedding*. Teknik ini umum digunakan untuk mengurangi dimensi data, menyatukan ukuran tensor di seluruh sampel, serta mengurangi risiko *overfitting* sekaligus menekan biaya komputasi [27], [28]. Dengan merangkum fitur pada dimensi tertentu, *pooling* membantu menonjolkan pola tingkat tinggi sambil menekan variasi yang kurang informatif.

Mean Pooling adalah salah satu strategi *pooling* yang paling umum digunakan. Metode ini menghitung nilai rata-rata dari fitur-fitur di dalam sebuah jendela *pooling*, sehingga menghasilkan representasi ringkas yang tetap mempertahankan tren fitur secara keseluruhan. *Mean Pooling* secara matematis didefinisikan sebagai:

$$Pooling_{\text{mean}}(X) = \frac{1}{l}\sum_{i=1}^{l} x_i \tag{2.4}$$

di mana X merepresentasikan peta fitur masukan, l adalah jumlah elemen di dalam jendela *pooling*, dan x_i merepresentasikan elemen individual yang sedang diagregasi. Operasi ini menghasilkan satu vektor representatif dari sekumpulan *feature embedding*.

2.5 Binary Cross Entropy

Binary Cross Entropy (BCE) adalah *loss function* yang umum digunakan dalam *Machine Learning* untuk mengukur performa model klasifikasi yang outputnya ialah nilai diantara 0 dan 1. Pada permasalahan dimana klasifikasi dapat disederhanakan menjadi pilihan biner, BCE sangat efisien dalam menyelesaikan banyak masalah secara bersamaan [29]. Adapun secara matematis, BCE dapat didefinisikan sebagai berikut:

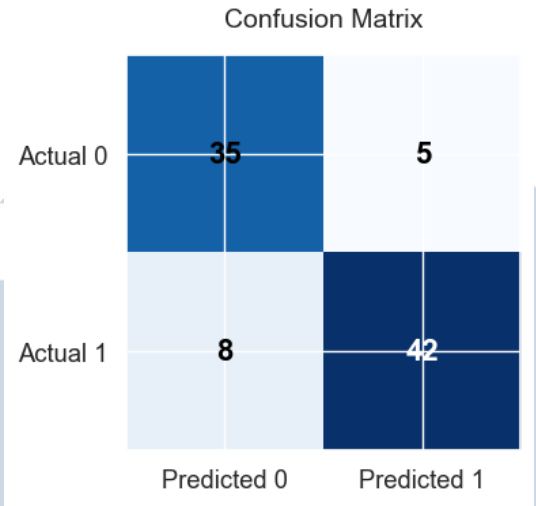
$$BCELoss = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p(y_i)) + (1-y_i)\log(1-p(y_i))) \tag{2.5}$$

di mana N adalah jumlah total sampel, y_i adalah label kebenaran (misalnya, 0 atau 1), dan $p(y_i)$ adalah probabilitas prediksi sampel termasuk dalam kelas 1 [30].

2.6 Matriks Evaluasi

Evaluasi model dilakukan untuk mengamati efisiensi pengklasifikasi MLP serta membandingkan kinerjanya dengan atau tanpa penggunaan wav2vec 2.0 sebagai *feature extractor.*

Untuk mengukur kualitas model secara kuantitatif, digunakan *confusion matrix* beserta beberapa metrik evaluasi. *Confusion matrix* merupakan alat kuantifikasi yang menggambarkan performa model pada satu pengujian. Metrik yang digunakan meliputi *True Negatives (TN), False Positives (FP), False Negatives (FN), dan True Positives (TP)*[31].



Gambar 2.3. Contoh dari *Confusion Matrix*

Confusion matrix menyoroti kemampuan model dalam mendeteksi kasus audio deephoax sekaligus menjaga tingkat kesalahan deteksi (*false alarm*) agar tetap terkendali. Hal ini sangat penting dalam konteks keamanan siber, karena deephoax yang tidak terdeteksi (*false negative*) menimbulkan risiko yang lebih besar dibandingkan dengan kesalahan peringatan sesekali.

Berdasarkan data yang dipetakan oleh *confusion matrix*, metrik evaluasi aktual dapat dihitung. Berikut adalah metrik evaluasi yang digunakan untuk menilai model ini [32]:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.6}$$

Akurasi (*accuracy*) mengukur tingkat ketepatan model secara keseluruhan pada seluruh hasil klasifikasi.

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

Presisi (*precision*) mengevaluasi seberapa banyak prediksi hoaks yang benar-benar merupakan hoaks (nilai FP rendah).

$$Recall = \frac{TP}{TP+FN} \tag{2.8}$$

Recall mengukur kemampuan model dalam mendeteksi hoaks yang sebenarnya (nilai FN rendah).

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (2.9)$$

F1-Score menyeimbangkan nilai presisi dan *recall*, dan sangat berguna ketika distribusi kelas tidak seimbang.