LAPORAN TUGAS BESAR II
Mata Kuliah Pembelajaran Mesin

Disusun oleh:
Arbrian Satria Hananda (1301190455), Ryan Abdurohman (1301191171)

A. Formulasi Masalah

Diberikan dataset mengenai data pelanggan di *dealer*, kita diminta mengklasifikasikan pelanggan apakah pelanggan tertarik untuk membeli kendaraan baru atau tidak. Pelanggan yang tertarik diberi label 1 dan sebaliknya jika tidak tertarik diberi label 0. Permasalahan ini termasuk dalam *binary classification problem.* Permasalahan ini merupakan permasalahan klasik untuk menentukan/memprediksi suatu data dengan jumlah pilihannya ada dua yaitu satu dan nol.

Catatan: Algoritma utama yaitu Neural Network dibuat from scratch

B. Eksplorasi dan Persiapan Data

1. Memuat dataset. Data akan dimuat dalam bentuk pandas DataFrame supaya memudahkan dalam pengolahan data.

```
[ ]  path = "/content/drive/MyDrive/Semester 5/Pembelajaran Mesin/Tugas II"

[ ]  dftrain = pd.read_csv(path+"/kendaraan_train.csv")
     dftest = pd.read_csv(path+"/kendaraan_test.csv")
```

2. Melihat cuplikan data. Data yang dimuat akan dilihat gambaran besarnya terlebih dahulu untuk dilakukan langkah selanjutnya.

```
[ ]  df = dftrain.copy()
```

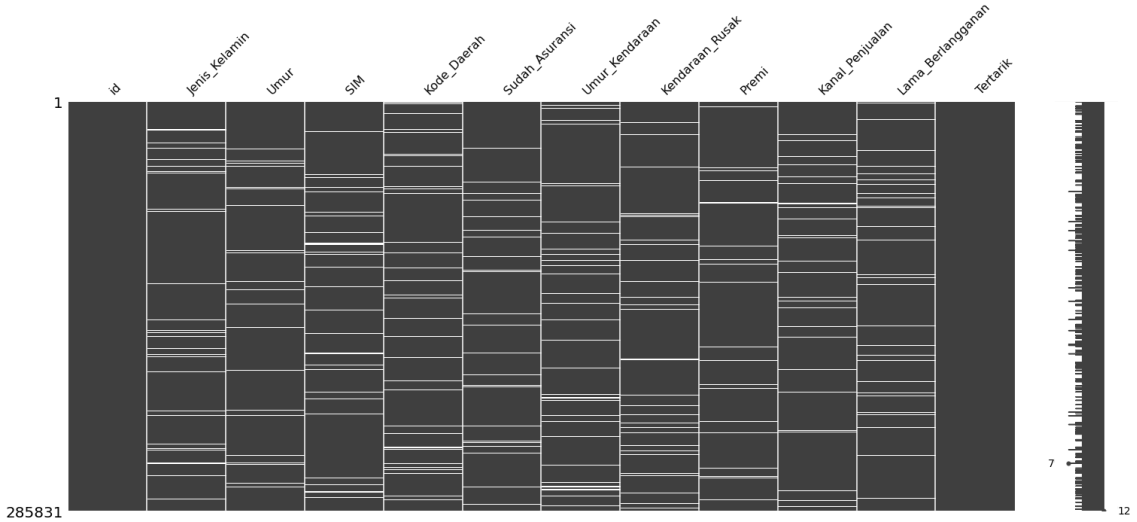
```
df.head()
```

	id	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan	Kendaraan_Rusak	Premi	Kanal_Penjualan	Lama_Berlangganan	Tertarik
1	1	Wanita	30.0	1.0	33.0	1.0	< 1 Tahun	Tidak	28029.0	152.0	97.0	0
2	2	Pria	48.0	1.0	39.0	0.0	> 2 Tahun	Pernah	25800.0	29.0	158.0	0
3	3	NaN	21.0	1.0	46.0	1.0	< 1 Tahun	Tidak	32733.0	160.0	119.0	0
4	4	Wanita	58.0	1.0	48.0	0.0	1-2 Tahun	Tidak	2630.0	124.0	63.0	0
5	5	Pria	50.0	1.0	35.0	0.0	> 2 Tahun	NaN	34857.0	88.0	194.0	0

3. Melihat informasi pada data. Dilakukan untuk mengetahui secara spesifik karakteristik dari data. Dapat kita lihat bahwa ada 285831 baris dengan 12 kolom, namun data yang Non-Null tidak setara dengan jumlah baris data, sehingga bisa kita dapatkan intuisi bahwa banyak data yang bernilai NaN atau kosong.

```
[ ]  df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 285831 entries, 0 to 285830
Data columns (total 12 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   id                 285831 non-null  int64
 1   Jenis_Kelamin      271391 non-null  object
 2   Umur               271617 non-null  float64
 3   SIM                271427 non-null  float64
 4   Kode_Daerah        271525 non-null  float64
 5   Sudah_Asuransi     271602 non-null  float64
 6   Umur_Kendaraan     271556 non-null  object
 7   Kendaraan_Rusak    271643 non-null  object
 8   Premi              271262 non-null  float64
 9   Kanal_Penjualan    271532 non-null  float64
 10  Lama_Berlangganan  271839 non-null  float64
 11  Tertarik           285831 non-null  int64
dtypes: float64(7), int64(2), object(3)
memory usage: 26.2+ MB
```

4. Melihat persebaran missing value. Dari informasi sebelumnya kita melihat banyak sekali kemunculan missing value/NaN. Untuk melihat gambaran persebarannya, kita bisa melakukan plotting menggunakan pustaka missingno. Bisa kita lihat bahwa persebarannya merata, tidak berkumpul di satu kolom. Sehingga kita perlu mempertimbangkan jumlah dari *missing value* itu sendiri di setiap kolomnya.



5. Melihat jumlah missing value pada setiap kolomnya, hal ini dilakukan untuk mendapatkan informasi yang akurat terhadap jumlah data yang kosong pada setiap kolomnya. Bisa kita lihat bahwa rata-rata ada data yang kosong sebanyak 14000-an di setiap kolom, namun pada langkah sebelumnya data kosong ini pun tersebar secara acak

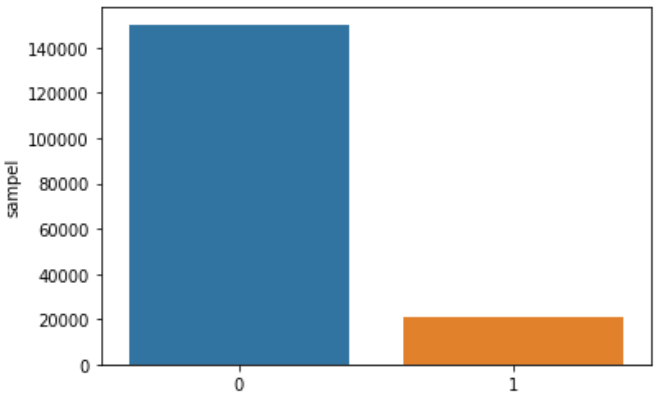
di setiap kolomnya, sehingga setiap baris bisa saja ada satu feature yang kosong atau lebih.

```
df.isna().sum()

id                    0
Jenis_Kelamin     14440
Umur              14214
SIM               14404
Kode_Daerah       14306
Sudah_Asuransi    14229
Umur_Kendaraan    14275
Kendaraan_Rusak   14188
Premi             14569
Kanal_Penjualan   14299
Lama_Berlangganan 13992
Tertarik              0
dtype: int64
```

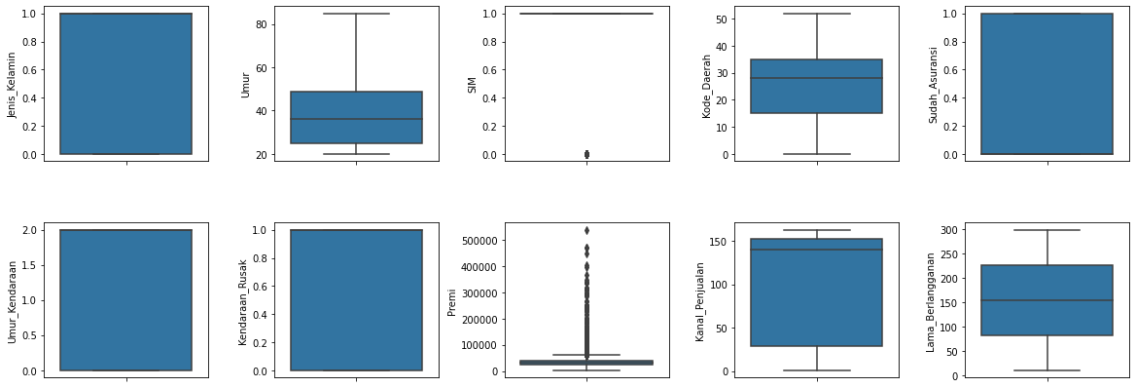
6. Drop baris yang berisi missing value. Pada tahap sebelumnya kita bisa lihat bahwa persebaran missing value begitu renggang, sehingga baris yang tidak terdapat missing value masih cukup untuk digunakan pada training.

```
[ ] df = df.dropna()

[ ] print("Jumlah data setelah di-drop sebanyak : "+str(df.shape[0]))

    Jumlah data setelah di-drop sebanyak : 171068

[ ] print("Baris yang di-drop sebanyak : "+ str(int(100*(285831-171068)/285831)) + " persen")

    Kolom yang di-drop sebanyak : 40 persen
```

7. Melihat persebaran kelas dari label. Bisa kita lihat bahwa data yang kita gunakan sejauh ini mayoritas memiliki label kelas 0 atau tidak tertarik. Untuk itu, penanganan terhadap data yang tidak seimbang ini perlu dilakukan suatu teknik supaya model yang dihasilkan bisa lebih baik dalam mengklasifikasi setiap kelasnya



8. Deteksi anomali/outlier pada data, kita lakukan visualisasi menggunakan boxplot dengan pustaka seaborn. Hal ini dilakukan untuk melihat distribusi suatu data dan melihat secara kasar apakah ada outlier.



Pada data yang kita olah, tidak terdapat outlier. Walaupun data premi dan SIM pada boxplot menunjukkan banyak titik-titik di luar plot, namun kita pahami bersama bahwa premi merupakan jumlah premi yang harus dibayarkan per tahun begitu pula kepemilikan SIM. Tentu saja bisa terjadi banyak data yang distribusinya beda, karena memang bisa saja setiap orang memiliki banyak produk asuransi yang berbeda, bahkan perbedaannya bisa signifikan, begitu pula dengan SIM, bisa saja memang orang tersebut tidak memiliki SIM. Oleh karena itu, bisa dinyatakan bahwa karakteristik data premi dan SIM itu sendiri di dunia nyata merupakan sesuatu yang bukan anomali/outlier.

9. Data Splitting. Data training yang digunakan akan di-split menjadi 9:1 dengan catatan "stratify" atau persebaran kelas baik di data training maupun validasi sama. Hal ini dilakukan dengan bantuan pustaka scikit-learn.

```
[ ] dfw_enc_ = dfw_enc.drop("Tertarik", axis=1)

    set_seed(42)
    xt, xte, yt, yte = train_test_split(dfw_enc_, dfw_enc["Tertarik"], test_size=0.1, random_state=42, stratify=dfw_enc["Tertarik"])

[ ] xt["Tertarik"] = yt

[ ] xte["Tertarik"] = yte
```

10. Target Encoder. Terakhir, data yang ada akan di-encode menggunakan Target Encoder dengan bantuan category_encoders. Pada dasarnya, setiap data pada masing-masing kolom akan direpresentasikan dengan peluang kemunculannya terhadap label 1. Misal, jika data pada kolom jenis kelamin terdapat:

Jenis Kelamin	Label 0	Label 1	Peluang Label 1
Pria	3	2	0.4
Wanita	2	3	0.6

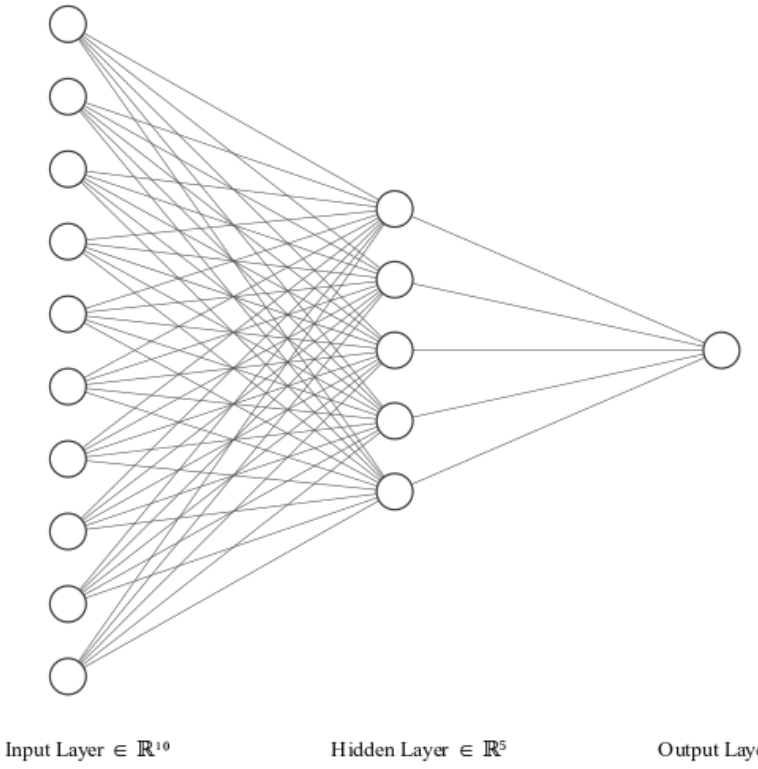
Kolom peluang pada label 1 akan menjadi representasi data yang ada. Kita tidak perlu melakukan scaling lagi pada data karena hasil dari peluang pasti range nya di antara nol dan satu. Berikut sampel data yang sudah melalui encoding:

x_val

	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan	Kendaraan_Rusak	Premi	Kanal_Penjualan	Lama_Berlangganan	Tertarik
0	0.451185	0.180804	0.500165	0.693656	0.676152	0.591941	0.693890	0.500000	0.097198	0.392185	0
1	0.451185	0.143065	0.500165	0.417901	0.006215	0.591941	0.006579	0.856260	0.097198	0.403822	0
2	0.451185	0.518990	0.500165	0.560677	0.676152	0.358731	0.693890	0.598201	0.612856	0.520679	1
3	0.451185	0.381401	0.500165	0.240496	0.676152	0.358731	0.693890	0.500000	0.612856	0.490868	0
4	0.451185	0.676347	0.500165	0.319861	0.676152	0.358731	0.693890	0.749996	0.612856	0.403822	1
...	...	...	...	...	...	...	...	...	...	...	...

C. Pemodelan

Pemodelan yang akan dilakukan menggunakan Artificial Neural Network. Pada eksperimen pertama ini, akan digunakan 2-Layer Neural Network dengan unit yang berbeda-beda setiap hidden layernya. Berikut contoh arsitekturnya:



Pada eksperimen kali ini, akan hidden layer akan diubah banyak unitnya sebanyak 2, 3, 4, dan 5. Dimensi Input layer mengikuti jumlah feature pada data yaitu 10. Untuk fungsi aktivasi pada hidden layer, kami menggunakan ReLU:

$$f(x) = \max(x, ax)$$

Lalu, untuk output layer, kami menggunakan sigmoid:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Inisialisasi Parameter

Pada eksperimen pertama ini, parameter akan diinisialisasi secara random lalu discale dengan 0.01. Pada setiap sejumlah L-layer, input feature kita ada 10 dan misal menggunakan data yang sudah dipreproses sebanyak 137121, maka ukuran matriks untuk setiap layernya sebagai berikut:

	Shape of W	Shape of b	Activation	Shape of Activation
Layer 1	$(n^{[1]}, 10)$	$(n^{[1]}, 1)$	$Z^{[1]} = W^{[1]}X + b^{[1]}$	$(n^{[1]}, 137121)$
Layer 2	$(n^{[2]}, n^{[1]})$	$(n^{[2]}, 1)$	$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$	$(n^{[2]}, 137121)$
⋮	⋮	⋮	⋮	⋮
Layer L-1	$(n^{[L-1]}, n^{[L-2]})$	$(n^{[L-1]}, 1)$	$Z^{[L-1]} = W^{[L-1]}A^{[L-2]} + b^{[L-1]}$	$(n^{[L-1]}, 137121)$
Layer L	$(n^{[L]}, n^{[L-1]})$	$(n^{[L]}, 1)$	$Z^{[L]} = W^{[L]}A^{[L-1]} + b^{[L]}$	$(n^{[L]}, 137121)$

Forward Propagation

Pada forward propagation, untuk setiap layernya akan dihitung menggunakan operasi perkalian matriks lalu diaktivasi menggunakan fungsi aktivasi sesuai yang sudah disebutkan di atas. Setiap layer direpresentasikan dengan satu matriks karena kami mengambil benefit vectorization dengan struktur data tensor.

$$A^{[L]} = \sigma(Z^{[L]}) = \sigma(W^{[L]}A^{[L-1]} + b^{[L]})$$

Cost Function

Perhitungan cost function dilakukan dengan menghitung rata-rata cross-entropy loss untuk semua sampel dengan rumus berikut:

$$-\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\log\left(a^{[L](i)}\right) + (1 - y^{(i)})\log\left(1 - a^{[L](i)}\right))$$

Backward Propagation

Pertama, kita akan menghitung turunan dari cost function terhadap fungsi aktivasi sigmoid (pada layer terakhir) misal kita notasikan dengan da[l] di mana l adalah layer ke-l. Lalu, untuk setiap fungsi aktivasi yang ada di antara layer, akan dihitung turunannya terhadap unit/fungsi linear. Misal kita notasikan dengan dZ:

$$dZ^{[l]} = dA^{[l]} * g'(Z^{[l]})$$

g'(Z[l]) merupakan turunan dari fungsi aktivasi yang digunakan. Lalu akan dihitung pula untuk setiap layernya turunan untuk setiap bobotnya dengan menghitung turunan parsial dari cost function terhadap setiap bobotnya:

$$dW^{[l]} = \frac{\partial \mathcal{J}}{\partial W^{[l]}} = \frac{1}{m} dZ^{[l]} A^{[l-1]T}$$

$$db^{[l]} = \frac{\partial \mathcal{J}}{\partial b^{[l]}} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[l](i)}$$

$$dA^{[l-1]} = \frac{\partial \mathcal{L}}{\partial A^{[l-1]}} = W^{[l]T} dZ^{[l]}$$

Update Parameter

Pada eksperimen pertama dilakukan update parameter menggunakan gradient descent dengan menghitung:

$$W^{[l]} = W^{[l]} - \alpha \; dW^{[l]}$$
$$b^{[l]} = b^{[l]} - \alpha \; db^{[l]}$$

Hyperparameter

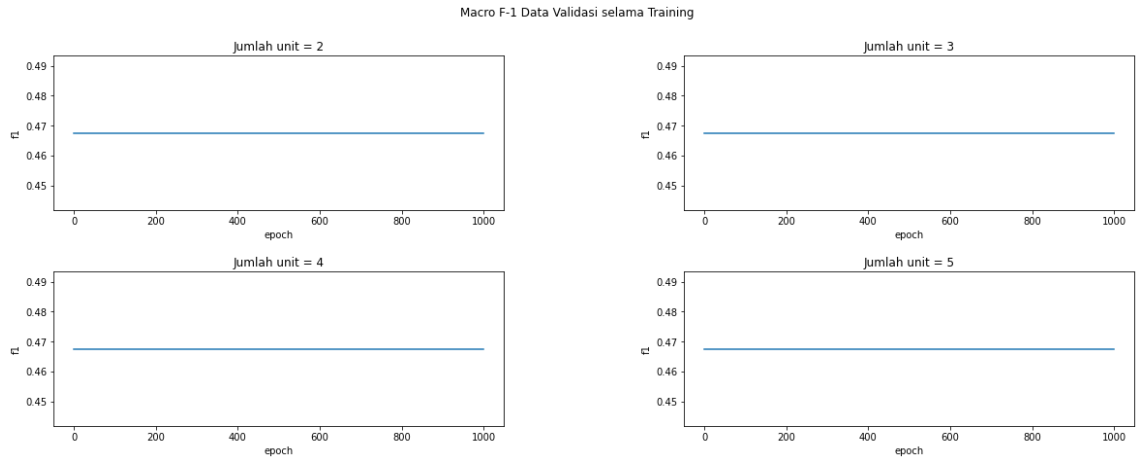
Untuk eksperimen pertama kami menggunakan arsitektur yang sudah disebutkan sebelumnya dengan proses training sebanyak 1000 epoch namun melalui tahap early stopping yaitu training akan berhenti sebanyak patience yang ditentukan. Semua eksperimen menggunakan patience sebanyak 100. Learning rate yang digunakan di semua eksperimen adalah 0.0007.

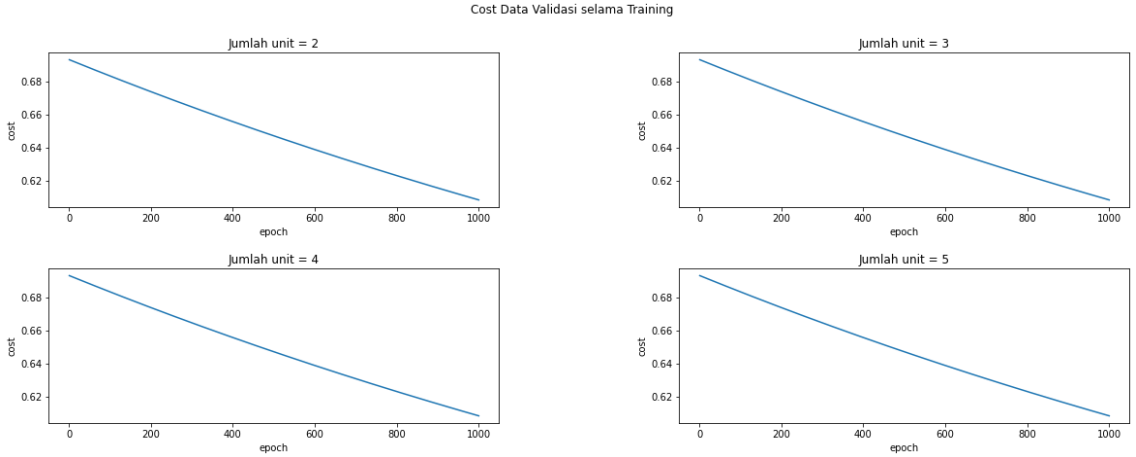
D. Evaluasi

Metrik Performansi

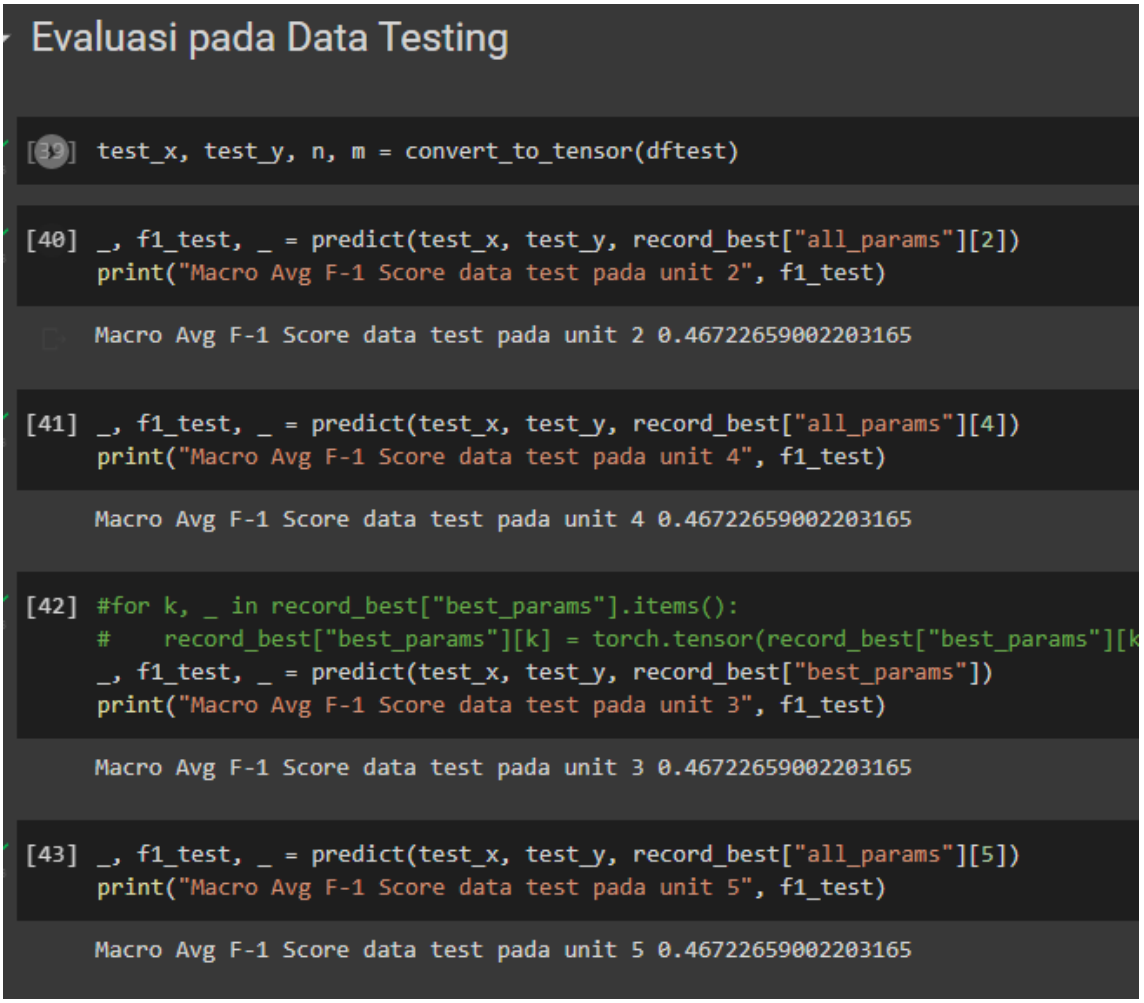
Untuk metrik performansi menggunakan macro avarage F-1 Score, karena datanya yang tidak seimbang sehingga perlu dihitung rerata F-1 score untuk setiap label.

Berikut hasil dari proses training pada eksperimen I:





	2	3	4	5
f1	0.467636	0.467636	0.467636	0.467636
val_cost	0.608529	0.608525	0.608528	0.608528
jumlah_epoch	1000.000000	1000.000000	1000.000000	1000.000000
f1_training	0.467639	0.467639	0.467639	0.467639
cost_training	0.608526	0.608522	0.608526	0.608526



Pada eksperimen pertama dapat dilihat bahwa performansinya masih dibawah 50%, artinya kita akan melakukan eksperimen pada Neural Network yang lebih banyak layernya.

E. Eksperimen

Eksperimen II: Deep Neural Network dengan Optimisasi dan Regularisasi

1. He Initialization

Pada eksperimen pertama parameter diinisialisasi secara random lalu discale dengan 0.01. Kami menyadari bahwa inisialisasi parameter pada Neural Network merupakan salah satu hal yang paling krusial. Maka, kami menggunakan teknik He Initialization yang pada dasarnya nilai random yang dihasilkan akan dikali dengan akar dari 2 dibagi dimensi pada layer sebelumnya

2. Random Mini-batch

Untuk mempercepat iterasi tiap epochnya serta mengurangi cost memory, maka data training akan dibagi ke dalam mini-batch tertentu. Untuk eksperimen kali ini data training akan dibagi sebanyak 2048 setiap batchnya.

3. Adaptive Moment Estimation

Pada eksperimen pertama, kami menggunakan gradient descent sebagai optimizer. Ketika menggunakan arsitektur deep neural network, maka diperlukan optimizer yang lebih cepat dan akurat. ADAM atau adaptive moment estimation pada dasarnya akan menghitung v atau exponentially weighting average pada bobot yang kita gunakan. Lalu discale dengan s yang pada dasarnya merupakan mean square dari exponentially weighting average pada bobot. Keduanya dihitung juga versi koreksinya supaya pada saat inisialisasi optimizer tidak mendapatkan bias yang besar

$$\begin{cases} v_{dW^{[l]}} = \beta_1 v_{dW^{[l]}} + (1-\beta_1)\frac{\partial \mathcal{J}}{\partial W^{[l]}} \\ v_{dW^{[l]}}^{corrected} = \frac{v_{dW^{[l]}}}{1-(\beta_1)^t} \\ s_{dW^{[l]}} = \beta_2 s_{dW^{[l]}} + (1-\beta_2)(\frac{\partial \mathcal{J}}{\partial W^{[l]}})^2 \\ s_{dW^{[l]}}^{corrected} = \frac{s_{dW^{[l]}}}{1-(\beta_2)^t} \\ W^{[l]} = W^{[l]} - \alpha \frac{v_{dW^{[l]}}^{corrected}}{\sqrt{s_{dW^{[l]}}^{corrected}}+\varepsilon} \end{cases}$$

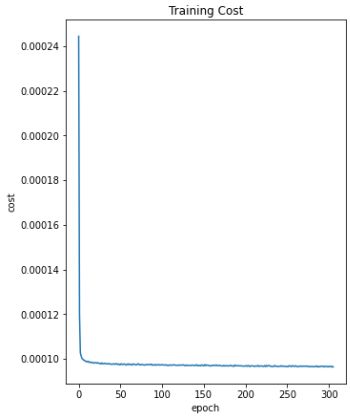
Parameter ADAM yang digunakan menggunakan default pada papernya yaitu beta1 = 0.9, beta2 = 0.999, dan epsilon = 1e-8.

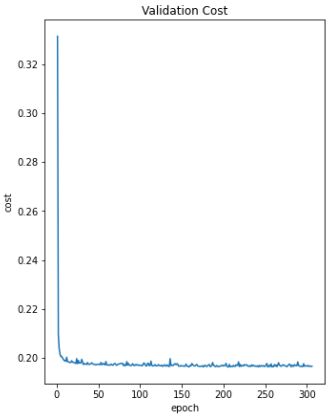
Evaluasi

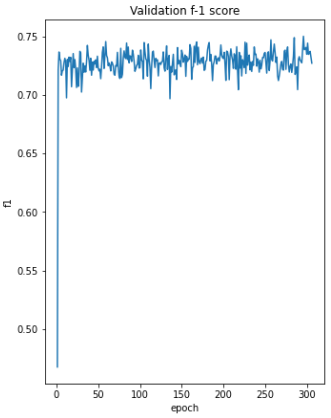
```
[52] pd.DataFrame(hist["hasil_eksperimen"])
```

	experimen_1	experimen_2	experimen_3	experimen_4
arsitektur	10->128->64->32->3->1	10->512->256->128->64->32->3->1	10->1024->128->512->32->3->1	10->81->27->3->1
f1_training	0.728602	0.467639	0.732274	0.732194
f1_validasi	0.732632	0.467636	0.735585	0.732492
f1_testing	0.557989	0.467227	0.561288	0.558642
cost_training	0.195913	0.370054	0.195724	0.195314
cost_validasi	0.19628	0.370076	0.196529	0.196889
cost_testing	1.00616	0.372928	0.909272	0.514715
jumlah_epoch	206	95	47	675
durasi_training	9.3524 menit	6.5812 menit	4.8276 menit	21.8119 menit

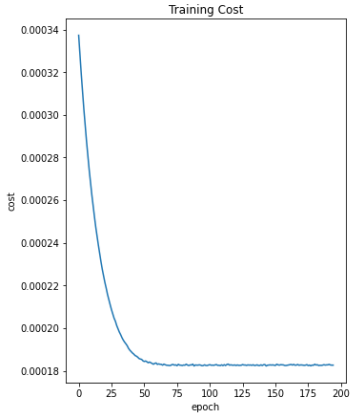
experimen_1: 10->128->64->32->3->1

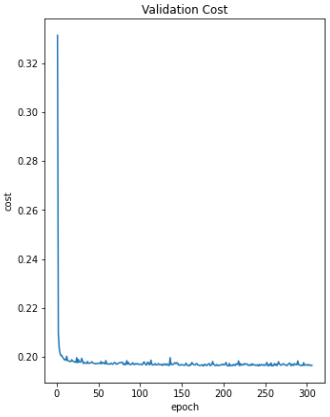


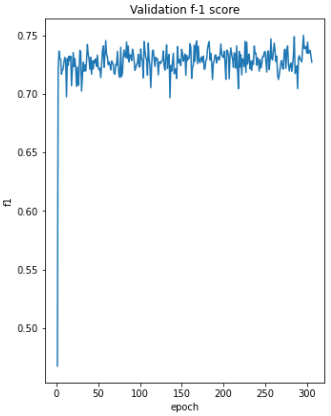




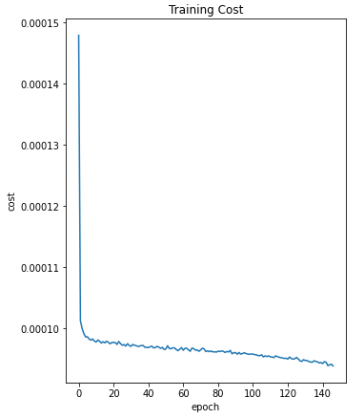
experimen_2: 10->512->256->128->64->32->3->1

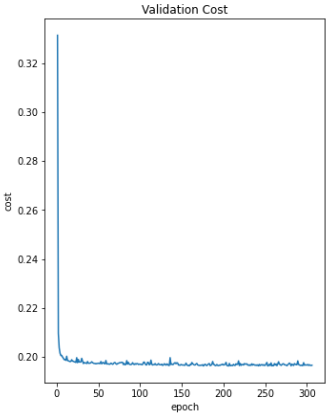


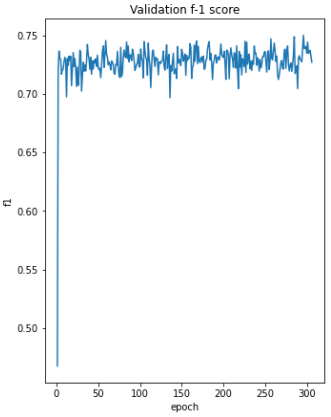


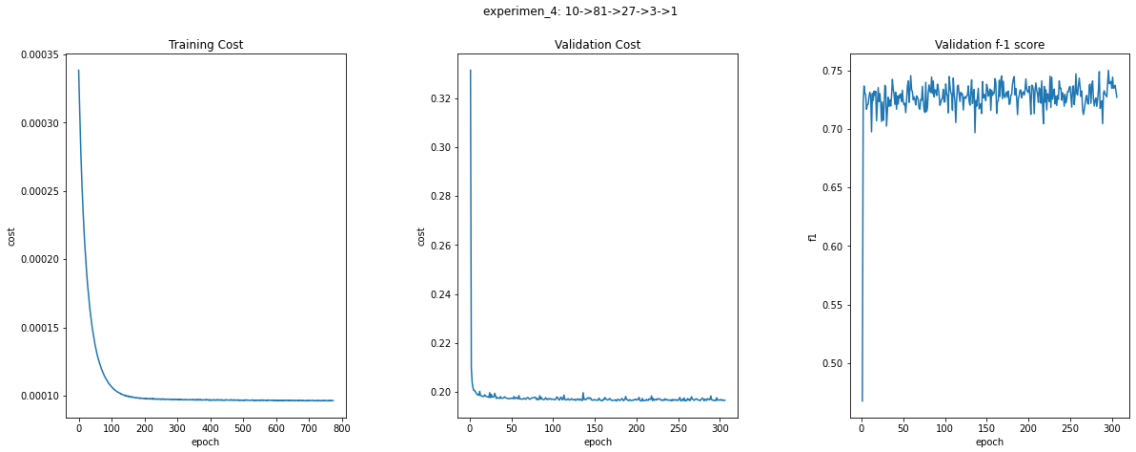


experimen_3: 10->1024->128->512->32->3->1









Pada eksperimen ini dicoba dengan berbagai model yang menggunakan banyak unit dan layer, bisa kita lihat bahwa performansi model yang dibangun sudah melebih 50% pada data testing. Dapat kita lihat juga bahwa tidak terjadi overfitting pada arsitektur yang dibangun (skor F-1 nya hampir sama). Model yang dibangun juga bisa dibilang mencapai konvergen dengan melihat plot training cost yang sudah mulai mendekati minimum.

F. Simpulan

Simpulan pada eksperimen kali ini, bahwa model yang dibangun sudah dapat melakukan klasifikasi pada data yang diberikan dengan macro average F-1 score terbaik pada 56%.

LAMPIRAN

Data hasil preprocessing ada pada folder: https://drive.google.com/drive/folders/19e9XsRBbT6XzM__Tzw-0d-URcLEoJn5G?usp=sharing

Link presentasi: https://youtu.be/mCY1C_NY1Cg