**Laporan Tugas Besar II**

**IF4071 Pembelajaran Mesin**

# **Halaman Judul**

**FEED FORWARD NEURAL NETWORK**

oleh

13515021

13515057

13515063

Dewita Sonya T.

Erick Wijaya

Kezia Suhendra



PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2018

# **Daftar Isi**

[**Halaman Judul**](#_os8s3qz12w82) **1**

[**Daftar Isi**](#_p6qatjidrqhw) **2**

[**Source Code**](#_bko5sulcg9dk) **3**

[Classifier A - Feed Forward Neural Network](#_s1ihqi8n98iv) 3

[Classifier B - Keras](#_toh1uvfdoj98) 6

[Hold-out validation](#_280o2738zlf2) 7

[**Hasil Eksekusi**](#_wyt7ym1qum2f) **8**

[Tahap Preprocessing (Encoding & Split dataset)](#_chj2o1bmwvmx) 8

[Classifier A - Feed Forward Neural Network](#_qrzpufbgf5f3) 9

[Classifier B - Keras](#_layxm9jdgrk3) 9

[**Perbandingan Classifier**](#_54z0waiftoy8) **10**

[**Pembagian Tugas**](#_kqnqldabrh1n) **11**

# **Source Code**

Seluruh kode program *classifier* dapat diakses pada repositori github melalui pranala <https://github.com/tugas-itb-erick/feed-neural-network>. Berikut adalah potongan kode program untuk setiap algoritma beserta penjelasannya.

## **Classifier A - Feed Forward Neural Network**

File: *main.ipynb*

|  |
| --- |
| **class** **NNClassifier**:  **def** **\_\_init\_\_**(self, n\_nodes=[], lrate=0.05, momentum=0, batch\_size=1, max\_iter=100):  *# Checking parameter input*  **if** (len(n\_nodes) > self.\_\_MAX\_HIDDEN):  **raise** ValueError('Number of hidden layers cannot be greater than {}'.format(self.\_\_MAX\_HIDDEN))   **if** (**not** all(x > 0 **for** x **in** n\_nodes)):  **raise** ValueError('Number of nodes in a layer cannot be nonpositive')   **if** (batch\_size <= 0):  **raise** ValueError('Batch size cannot be nonpositive')   *# Setting parameter*  self.n\_nodes = n\_nodes  self.n\_hiddens = len(n\_nodes)  self.lrate = lrate  self.momentum = momentum  self.batch\_size = batch\_size  self.max\_iter = max\_iter  self.weights = []  self.prev\_weights = []   **@property**  **def** **\_\_MAX\_HIDDEN**(self):  **return** 10   **def** **\_\_stochastic\_gradient\_descend**(self, data, target):  **for** x, y **in** zip(data, target):  x = np.append(x, 1.)  values\_layers = self.\_\_feed\_forward(x)  errors\_layers = self.\_\_backward\_prop(y, values\_layers)  values\_layers.insert(0, x)   *# Update weight*  new\_weights = []  **for** ilayer, (weights\_per\_layer, prev\_weights\_per\_layer) **in** enumerate(zip(self.weights, self.prev\_weights)):  new\_weights\_per\_layer = []  **for** inode, (weight\_all, prev\_weight\_all) **in** enumerate(zip(weights\_per\_layer, prev\_weights\_per\_layer)):  new\_weight\_all = []  **for** iweight, (weight, prev\_weight) **in** enumerate(zip(weight\_all, prev\_weight\_all)):  new\_weight\_all.append(self.\_\_calculate\_weight(weight, prev\_weight,   values\_layers[ilayer][inode], errors\_layers[ilayer][iweight]))  new\_weights\_per\_layer.append(new\_weight\_all)  new\_weights.append(np.array(new\_weights\_per\_layer))  **for** i **in** range(len(new\_weights)):  **for** j **in** range(len(new\_weights[i])):  **for** k **in** range(len(new\_weights[i][j])):  self.prev\_weights[i][j][k] = new\_weights[i][j][k] - self.weights[i][j][k]  self.weights = new\_weights    **def** **\_\_feed\_forward**(self, x):  outputs = [x]  **for** weight **in** self.weights:  outputs.append(self.\_\_sigmoid(outputs[-1] @ weight))  **del** outputs[0]  **return** outputs    **def** **\_\_sigmoid**(self, x):  **return** 1 / (1 + np.exp(-x))   **def** **\_\_backward\_prop**(self, target, values\_layers):  n\_hiddens\_out\_layers = len(values\_layers)  errors\_layers = [**None**] \* n\_hiddens\_out\_layers  **for** i **in** range(n\_hiddens\_out\_layers-1, 0-1, -1):  errors = []  **if** i < n\_hiddens\_out\_layers-1: *# (hidden layer)*  **for** inode, output **in** enumerate(values\_layers[i]):  errors.append(self.\_\_hidden\_error(output, inode, i, errors\_layers))  **else**: *# i == n\_hiddens\_out\_layers-1 (output layer)*  **for** output **in** values\_layers[i]:  errors.append(self.\_\_output\_error(output, target))  errors\_layers[i] = np.array(errors)  **return** errors\_layers   **def** **\_\_output\_error**(self, output, target):  **return** output \* (1 - output) \* (target - output)    **def** **\_\_hidden\_error**(self, output, inode, index\_layer, errors\_layers):  index\_delta = index\_layer + 1  index\_weight = index\_layer + 1  sigma = 0  **for** i **in** range(0, len(self.weights[index\_weight][inode])):  sigma += self.weights[index\_weight][inode][i] \* errors\_layers[index\_delta][i]  **return** output \* (1 - output) \* sigma   **def** **\_\_calculate\_weight**(self, weight, prev\_weight, err, val):  **return** weight + self.momentum \* prev\_weight + self.lrate \* err \* val   **def** **fit**(self, data, target):  self.\_\_initialize\_weights(data)   **for** \_ **in** range(self.max\_iter):  *# Random shuffle data and target simultaneously*  p = np.random.permutation(data.shape[0])  data, target = data[p], target[p]   *# Do gradient descent per batch*  **for** i **in** range(0, data.shape[0], self.batch\_size):  index = list(range(i, i+self.batch\_size))  self.\_\_stochastic\_gradient\_descend(data[index], target[index])    **return** self   **def** **\_\_initialize\_weights**(self, data):  *# Initialize weights with random numbers*  n\_features = data.shape[1]  **if** (self.n\_hiddens > 0):  self.weights = [np.random.randn(n\_features + 1, self.n\_nodes[0])]  **for** i **in** range(1, self.n\_hiddens):  self.weights.append(np.random.randn(self.n\_nodes[i-1], self.n\_nodes[i]))  self.weights.append(np.random.randn(self.n\_nodes[self.n\_hiddens - 1], 1))  **else**:  self.weights = [np.random.randn(n\_features + 1, 1)]    *# Assume first prev\_weights be zeroes*  self.prev\_weights = deepcopy(self.weights)    **def** **predict**(self, data):  result = [self.\_\_feed\_forward(np.append(d, 1.))[-1][0] **for** d **in** data]  **return** [1 **if** r >= 0.5 **else** 0 **for** r **in** result] |

Kode program diatas adalah implementasi *classifier* Feed Forward Neural Network dengan algoritma *gradient descent.* Terdapat fungsi **fit** yang dapat digunakan untuk melakukan *fitting* terhadap data *training,* dan fungsi *predict* yang dapat digunakan untuk melakukan prediksi terhadap data *testing.* Fungsi **fit** menerima parameter berupa data yang akan di-*train* dan *label* dari data yang akan digunakan. Fungsi ini menggunakan *gradient descent* untuk melakukan *update* pada berat di setiap *layer*, *update* dilakukan sebanyak jumlah iterasi maksimal. Fungsi **predict** digunakan untuk memprediksi kelas dari data *training* yang menjadi masukan. Fungsi ini memasukkan data masukan ke dalam jaringan dan mengeluarkan hasilnya.

## **Classifier B - Keras**

File: *main.ipynb*

|  |
| --- |
| **from** tensorflow.keras **import** optimizers **from** tensorflow.keras.layers **import** Dense, Activation **from** tensorflow.keras.models **import** Sequential  **class** **NNKeras**():  **def** **\_\_init\_\_**(self, nnodes\_per\_hidden\_layer=[100], lrate=0.05, momentum=0, batch\_size=1):  self.nnodes\_per\_hidden\_layer = nnodes\_per\_hidden\_layer  self.lrate = lrate  self.momentum = momentum  self.batch\_size = batch\_size    **def** **fit**(self, data, labels, epochs=1):  """data: ndarray"""  n\_rows = len(data)  n\_attr = len(data[n\_rows-1])  self.model = Sequential()  *# First Hidden Layer*  self.model.add(Dense(units=self.nnodes\_per\_hidden\_layer[0], activation='sigmoid', input\_dim=n\_attr))  *# 2nd .. Last Hidden Layer*  **for** i **in** range(1, len(self.nnodes\_per\_hidden\_layer)):  self.model.add(Dense(units=self.nnodes\_per\_hidden\_layer[i], activation='sigmoid'))  *# Output Layer*  self.model.add(Dense(units=1, activation='sigmoid'))    sgd = optimizers.SGD(lr=self.lrate, momentum=self.momentum)  self.model.compile(optimizer=sgd, loss='mean\_squared\_error')  self.model.fit(data, labels, batch\_size=self.batch\_size, epochs=epochs, verbose=0)  **return** self   **def** **predict**(self, sample):  **return** self.model.predict\_classes(sample, batch\_size=self.batch\_size) |

Kode diatas adalah implementasi dari Keras untuk melakukan *training* data dengan menggunakan fungsi **fit** dan memprediksi kelas dari data baru dengan menggunakan fungsi **predict**. Fungsi **fit** menerima parameter berupa data yang akan di-*train*, *label* dari data yang akan digunakan serta banyaknya *epoch* yang akan dilakukan. Pada fungsi ini, digunakan **models.Sequential** untuk mengkonfigurasi model. *Optimizers* yang digunakan adalah *stochastic gradient descent* dengan parameter berupa *learning rate* dan momentum. Dipilih SGD sebagai *optimizers* agar menyesuaikan dengan *optimizers* yang digunakan pada *classifier* A. Fungsi **predict** digunakan untuk memprediksi kelas dari data yang baru di-*input*. Sedangkan *dense* digunakan untuk merepresentasikan setiap layer yang ada seperti *hidden layer* dan *output layer.* *Dense* yang terakhir akan dijadikan sebagai *output layer* dan sisanya sebagai *hidden layer*. *Units* yang ada pada *dense* merepresentasikan jumlah *node* yang ada pada setiap *layer*. Fungsi aktivasi yang digunakan adalah *sigmoid*. *Loss* melambangkan nilai dari *error* yang ada pada setiap *epoch*, semakin kecil nilai tersebut maka bobot dari data akan semakin konvergen.

## **Hold-out validation**

File: *main.ipynb*

|  |
| --- |
| *# Evaluate with validation data* model\_nn = NNClassifier(n\_nodes=[20,10], lrate=0.1, momentum=0.1, batch\_size=1, max\_iter=450).fit(  train\_df.values, train\_labels)  print("Test with train data:") print(" Predicted", model\_nn.predict(train\_df.values)) print(" Expected", train\_labels) print("Test with validation data:") print(" Predicted", model\_nn.predict(validation\_df.values)) print(" Expected", validation\_labels)  *# Evaluate with validation data* model\_keras = NNKeras(nnodes\_per\_hidden\_layer=[20,10], lrate=0.1, momentum=0.1, batch\_size=1).fit(  train\_df.values, train\_labels, epochs=450) print("Test with train data:") print(" Predicted", model\_keras.predict(train\_df.values).tolist()) print(" Expected", train\_labels) print("Test with validation data:") print(" Predicted", model\_keras.predict(validation\_df.values).tolist()) print(" Expected", validation\_labels) |

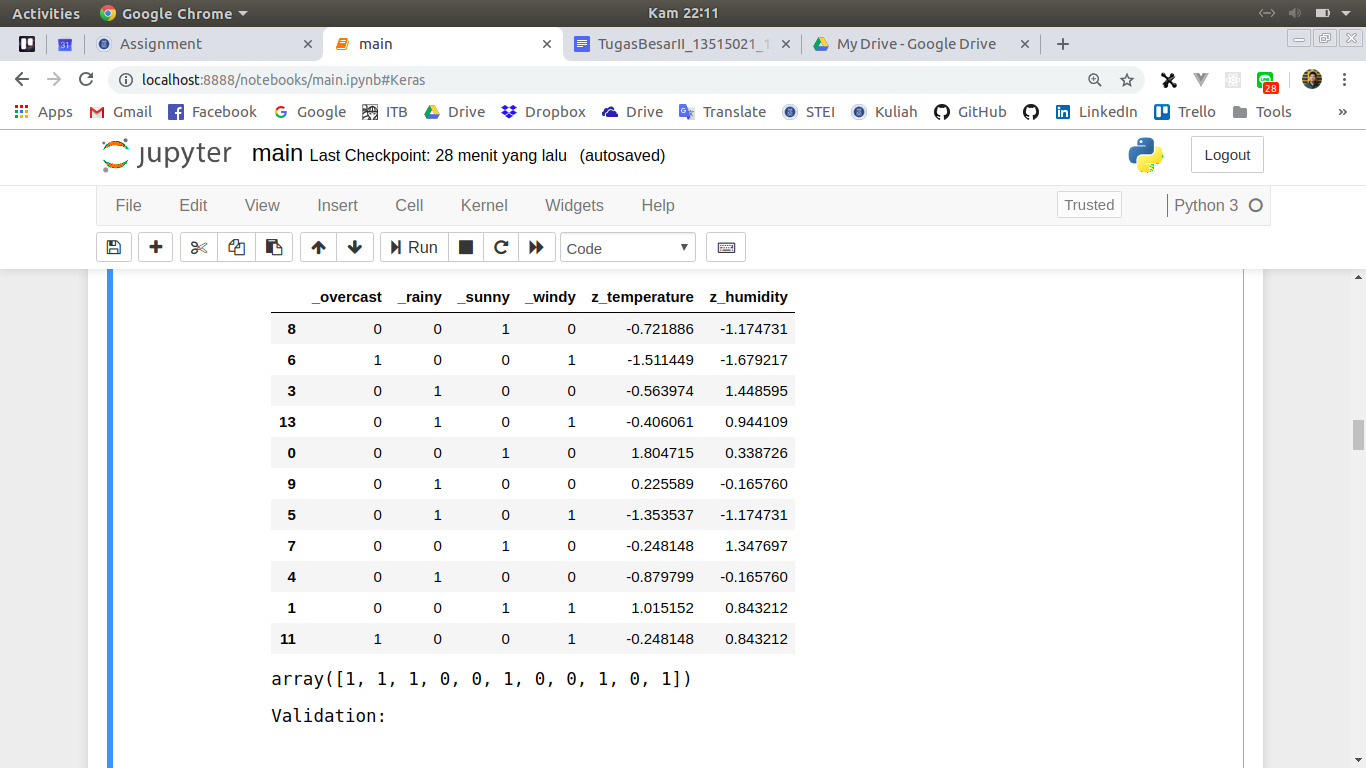
Kode program diatas adalah *training* yang dilakukan dengan menggunakan algoritma kedua classifier. Untuk melakukan hold-out, validasi dilakukan pada data validasi terlebih dulu. Setelah beberapa percobaan yang dilakukan kelompok kami, diperoleh bahwa jumlah *hidden layer* sebanyak 2 (20 unit dan 10 unit) dengan *learning rate* dan *momentum* bernilai 0.1 dan dengan *epoch* yang besar menghasilkan model yang paling baik.

# **Hasil Eksekusi**

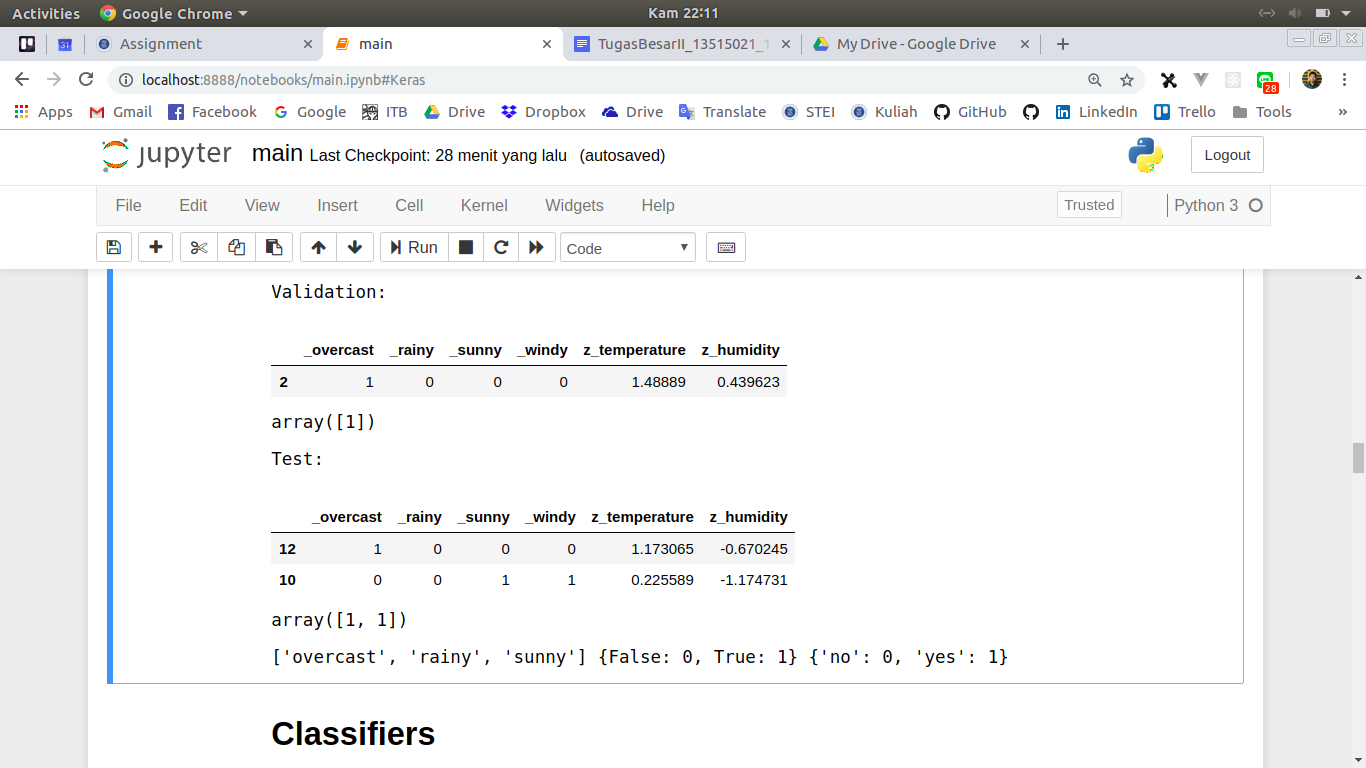
Hasik eksekusi program selengkapnya dapat dilihat pada repositori github melalui pranala <https://github.com/tugas-itb-erick/feed-neural-network>.

## **Tahap Preprocessing (Encoding & Split dataset)**

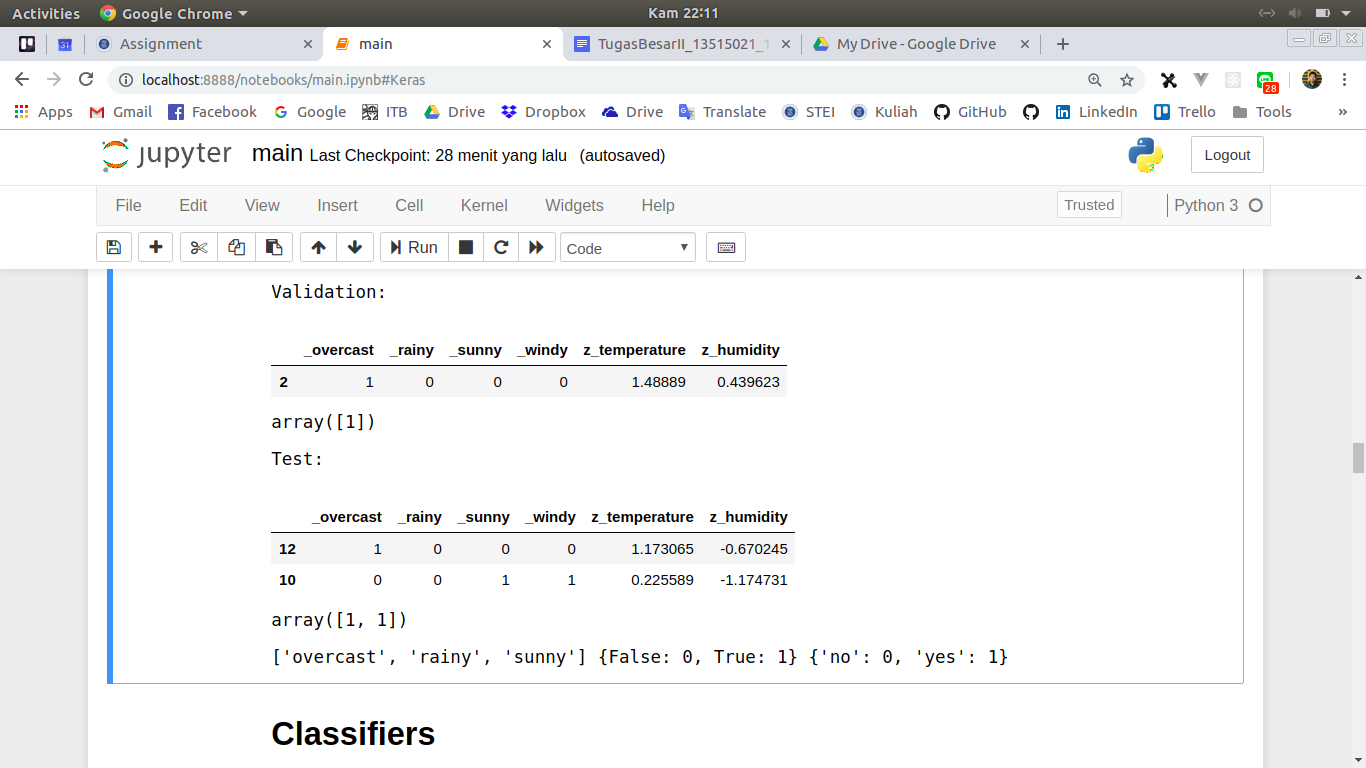
**Train data (80%):**



**Validation data (10%):**



**Test data (10%):**



## **Classifier A - Feed Forward Neural Network**

**Validasi Hold-out dengan data validasi:**

|  |
| --- |
| Test with train data:  Predicted [[1], [1], [1], [0], [0], [1], [0], [0], [1], [0], [1]]  Expected [1 1 1 0 0 1 0 0 1 0 1]  Test with validation data:  Predicted [[1]]  Expected [1] |

**Evaluasi akhir dengan data uji:**

|  |
| --- |
| Test with train+validation data:  Predicted [[1], [1], [1], [1], [1], [1], [0], [1], [0], [0], [1], [1]]  Expected [1 1 1 1 1 1 0 1 0 0 1 1]  Test with test data:  Predicted [[1], [1]]  Expected [1 1] |

## **Classifier B - Keras**

**Validasi Hold-out dengan data validasi:**

|  |
| --- |
| Test with train data:  Predicted [[1], [1], [1], [0], [0], [1], [0], [0], [1], [0], [1]]  Expected [1 1 1 0 0 1 0 0 1 0 1]  Test with validation data:  Predicted [[1]]  Expected [1] |

**Evaluasi akhir dengan data uji:**

|  |
| --- |
| Test with train+validation data:  Predicted [[1], [1], [1], [1], [1], [1], [0], [1], [0], [0], [1], [1]]  Expected [1 1 1 1 1 1 0 1 0 0 1 1]  Test with test data:  Predicted [[1], [1]]  Expected [1 1] |

# 

# **Perbandingan Classifier**

*Classifier* Adan *classifier* B menghasilkan hasil prediksi untuk data baru pada kelas yang sama. Hasil validasi *hold-out* dan evaluasi akhir dapat dilihat pada bagian hasil eksekusi (eksperimen). Hal ini dikarenakan, kedua *classifier* menggunakan *optimizers* yang sama yaitu *stochastic gradient descent* dan pada dasarnya kedua *classifier* akan selalu menghasilkan prediksi yang sama. Namun, untuk *classifier* B akan lebih mudah untuk digunakan karena hanya memanggil *library* keras milik Tensorflow. Dari segi waktu yang dibutuhkan saat melakukan eksekusi, *classifier* A masih kurang optimal karena tidak melakukan optimasi dan banyak terdapat operasi yang cukup boros, seperti *copy list* dan sebagainya.

# **Pembagian Tugas**

|  |  |  |
| --- | --- | --- |
| **Dewita Sonya T**  **13515021** | **Erick Wijaya**  **13515057** | **Kezia Suhendra**  **13515063** |
| Classifier A tahap Feed Forward, Debugging | Classifier A tahap Backward Propagation, Preprocessing | Classifier B (Keras) |