# TUGAS 1

# ADVANCE DEEP LEARNING

**NIM : 14230018**

**NAMA : ILHAM MAULANA**

SourceCode : https://colab.research.google.com/drive/1K6xZbYdRAE7_QhEOlhq4zNqBAiZXI_XJ?usp=sharing

## A. Dataset

Dataset ini adalah bagian dari proyek CUMIDA (CUtting-edge MIcroarray DAta analysis), yang merupakan sumber data yang disediakan oleh Laboratório de Genômica Computacional (LabGC) di Departamento de Genética da Universidade Federal do Rio Grande do Sul (UFRGS), Brazil. Dataset yang Anda tunjukkan adalah bagian dari dataset Prostate_GSE6919_U95B, yang merupakan hasil dari studi mikroarray pada kanker prostat.

https://sbcb.inf.ufrgs.br/data/cumida/Genes/Prostate/GSE6919_U95B/Prostate_GSE6919_U95B.csv

| TYPE | GSE | GPL PLATFORM | SAMPLES | GENES | CLASSES | |
|---|---|---|---|---|---|---|
| Prostate | 6919_U95B | 92 | 124 | 12621 | 2 | Download |

| ZEROR | SVM | MLP | DT | NB | RF | HC | KNN | K-MEANS |
|---|---|---|---|---|---|---|---|---|
| 0.52 | 0.68 | 0.62 | 0.6 | 0.71 | 0.67 | 0.51 | 0.56 | 0.54 |

| | samples | type | 41880_at | 41881_at | 41882_at | 41883_at | 41884_at | 41885_at | 41886_r_at | 41887_at | ... | AFFX-ThrX-3_at | AFFX-ThrX-5_at | AFFX-ThrX-M_at | AFFX-TrpnX-3_at | AFFX-TrpnX-5_at |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | GSM152992.CEL | primary_prostate_tumor | 2.414076 | 4.113824 | 2.035911 | 3.102248 | 2.115578 | 1.775455 | 6.107839 | 2.160168 | ... | 2.955998 | 2.910953 | 2.095267 | 1.617076 | 2.060144 |
| 1 | GSM152993.CEL | primary_prostate_tumor | 2.385157 | 4.078664 | 2.123064 | 3.087631 | 2.254190 | 1.815183 | 5.708878 | 2.134447 | ... | 3.196521 | 2.975412 | 2.249950 | 1.757867 | 2.352185 |
| 2 | GSM152994.CEL | primary_prostate_tumor | 2.295522 | 4.085505 | 2.144344 | 3.071539 | 2.229422 | 1.985899 | 5.679248 | 2.100443 | ... | 2.929904 | 2.857025 | 2.047436 | 1.625339 | 2.065674 |
| 3 | GSM152995.CEL | primary_prostate_tumor | 2.260478 | 4.466391 | 2.206410 | 3.505265 | 2.605014 | 1.887307 | 5.935039 | 2.261295 | ... | 3.578538 | 3.420946 | 2.736342 | 1.940826 | 2.713500 |
| 4 | GSM152996.CEL | primary_prostate_tumor | 2.229731 | 4.291435 | 2.506255 | 3.220628 | 2.404673 | 1.886664 | 5.965917 | 2.274317 | ... | 3.558184 | 3.083316 | 2.562048 | 1.923414 | 2.775842 |
| 5 | GSM152997.CEL | primary_prostate_tumor | 2.323719 | 4.042794 | 2.219460 | 3.030761 | 2.539964 | 1.836586 | 5.970438 | 1.880250 | ... | 3.442099 | 3.239850 | 2.373556 | 1.973410 | 2.563835 |
| 6 | GSM152998.CEL | primary_prostate_tumor | 2.557032 | 4.196903 | 2.122405 | 3.495034 | 2.353686 | 2.100858 | 6.338756 | 2.019285 | ... | 3.542867 | 3.178389 | 2.413796 | 2.038962 | 2.723099 |
| 7 | GSM152999.CEL | primary_prostate_tumor | 2.441765 | 4.148545 | 1.968985 | 3.230116 | 2.240704 | 1.896051 | 6.166129 | 2.012576 | ... | 3.363998 | 3.314865 | 2.345718 | 1.927122 | 2.600874 |
| 8 | GSM153000.CEL | primary_prostate_tumor | 2.604707 | 4.234341 | 2.167552 | 3.394769 | 2.294088 | 2.089270 | 6.190621 | 2.030608 | ... | 3.703331 | 3.263648 | 2.769345 | 2.183191 | 2.621211 |
| 9 | GSM153001.CEL | primary_prostate_tumor | 2.364670 | 4.183144 | 1.866217 | 3.169024 | 2.371755 | 1.983816 | 5.748455 | 2.142535 | ... | 3.077640 | 3.048728 | 2.230446 | 1.928804 | 2.466[illegible] |

10 rows × 12622 columns

1. Membangun 4 DNN dengan variasi

## a. Hidden layer 1: 1000 neurons, ReLU activation

```
def build_model_one_hidden():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(1000, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # Hidden layer with 1000 neurons and
ReLU activation
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')           # Output layer with sigmoid activation
    ])
    return model
```

## b. Hidden layer 2: 500 neurons, ReLU activation

```
def build_model_two_hidden():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(500, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # 1st hidden layer with 500 neurons
and ReLU activation
        tf.keras.layers.Dense(500, activation='relu'),
# 2nd hidden layer with 500 neurons and ReLU activation
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')          # Output layer with sigmoid activation
    ])
    return model
```

c. Hidden layer 3: 250 neurons, ReLU activation

```
def build_model_three_hidden():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(250, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # First hidden layer with 250 neurons
and ReLU activation
        tf.keras.layers.Dense(250, activation='relu'),
# Second hidden layer with 250 neurons and ReLU activation
        tf.keras.layers.Dense(250, activation='relu'),
# Third hidden layer with 250 neurons and ReLU activation
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model
```

d. Hidden layer 4: 100 neurons, ReLU activation

```
def build_model_four_hidden():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(100, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # First hidden layer with 100 neurons
and ReLU activation
        tf.keras.layers.Dense(100, activation='relu'),
# Second hidden layer with 100 neurons and ReLU activation
        tf.keras.layers.Dense(100, activation='relu'),
# Third hidden layer with 100 neurons and ReLU activation
        tf.keras.layers.Dense(100, activation='relu'),
# Fourth hidden layer with 100 neurons and ReLU activation
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model
```

e. Output Layer: (jlh kelas) neuron, Sigmoid activation, Epoch 100, batch size 10, Optimizer: SGD

```
model_one_hidden_layer = build_model_one_hidden()
model_two_hidden_layer = build_model_two_hidden()
model_three_hidden_layer = build_model_three_hidden()
model_four_hidden_layer = build_model_four_hidden()
```

```
model_one_hidden_layer.compile(optimizer='sgd',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_one_hidden_layer.fit(X_train_scaled, y_train, epochs=100,
batch_size=10, validation_data=(X_val_scaled, y_val))

model_two_hidden_layer.compile(optimizer='sgd',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_two_hidden_layer.fit(X_train_scaled, y_train, epochs=100,
batch_size=10, validation_data=(X_val_scaled, y_val))

model_three_hidden_layer.compile(optimizer='sgd',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_three_hidden_layer.fit(X_train_scaled, y_train, epochs=100,
batch_size=10, validation_data=(X_val_scaled, y_val))

model_four_hidden_layer.compile(optimizer='sgd',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_four_hidden_layer.fit(X_train_scaled, y_train, epochs=100,
batch_size=10, validation_data=(X_val_scaled, y_val))
```

## f. Hasilnya

| | Optimizer | Hidden Layers | Validation Accuracy | Test Accuracy |
|---|---|---|---|---|
| 0 | SGD | 1 | 0.75 | 0.64 |
| 1 | SGD | 2 | 0.75 | 0.64 |
| 2 | SGD | 3 | 0.70 | 0.68 |
| 3 | SGD | 4 | 0.70 | 0.64 |

Hasil terbaik test accuracy menggunakan SGD Optimizer adalah pada **layer3** dengan Test Accuracy **0.68**

## 2. Menganti Optimizer

### a. Adam

```
model_one_hidden_layer.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_one_hidden_layer.fit(X_train_scaled, y_train, epochs=100,
batch_size=10, validation_data=(X_val_scaled, y_val))
```

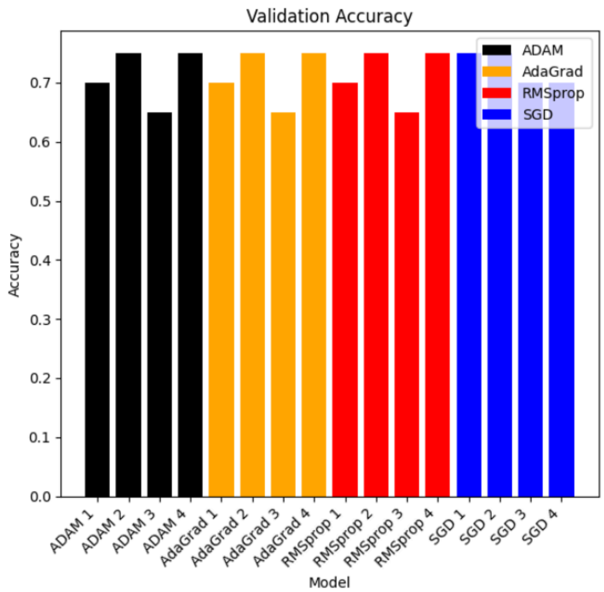
### b. AdaGrad

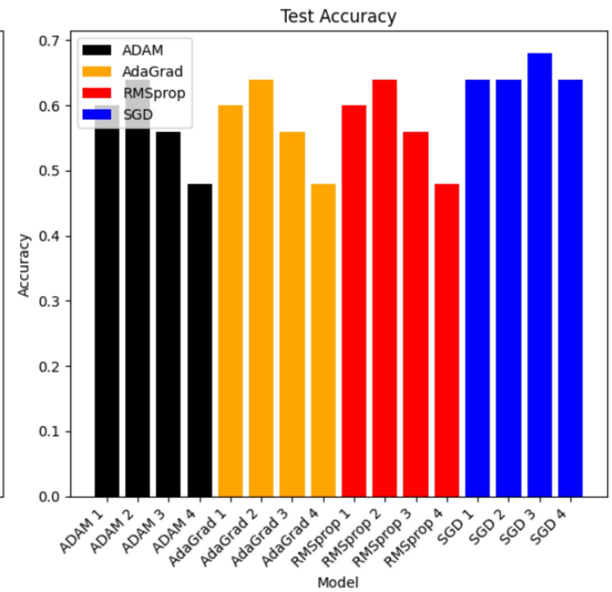
```
optimizer_adagrad = tf.keras.optimizers.Adagrad()
model_one_hidden_layer.compile(optimizer=optimizer_adagrad,
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_one_hidden_layer.fit(X_train_scaled, y_train, epochs=100,
batch_size=10, validation_data=(X_val_scaled, y_val))
```

### c. RMSProp

```
optimizer_rmsprop = tf.keras.optimizers.RMSprop()
model_one_hidden_layer.compile(optimizer=optimizer_rmsprop,
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_one_hidden_layer.fit(X_train_scaled, y_train, epochs=100,
batch_size=10, validation_data=(X_val_scaled, y_val))
```

## d. Hasilnya




| | Optimizer | Hidden Layers | Validation Accuracy | Test Accuracy |
|---|---|---|---|---|
| 0 | SGD | 1 | 0.75 | 0.64 |
| 1 | SGD | 2 | 0.75 | 0.64 |
| 2 | SGD | 3 | 0.70 | 0.68 |
| 3 | SGD | 4 | 0.70 | 0.64 |
| 4 | ADAM | 1 | 0.70 | 0.60 |
| 5 | ADAM | 2 | 0.75 | 0.64 |
| 6 | ADAM | 3 | 0.65 | 0.56 |
| 7 | ADAM | 4 | 0.75 | 0.48 |
| 8 | AdaGrad | 1 | 0.70 | 0.60 |
| 9 | AdaGrad | 2 | 0.75 | 0.64 |
| 10 | AdaGrad | 3 | 0.65 | 0.56 |
| 11 | AdaGrad | 4 | 0.75 | 0.48 |
| 12 | RMSprop | 1 | 0.70 | 0.60 |
| 13 | RMSprop | 2 | 0.75 | 0.64 |
| 14 | RMSprop | 3 | 0.65 | 0.56 |
| 15 | RMSprop | 4 | 0.75 | 0.48 |

Hasil terbaik test accuracy yaitu menggunakan SGD Optimizer adalah pada **layer3**

Dengan Validation Accuracy **0.70 dan** Test Accuracy **0.68**

## 3. Menambahkan Dropout 50%

### a. Di salah satu layer

```
def build_model_three_hidden_with_dropout():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(250, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # First hidden layer with 250 neurons
and ReLU activation
        tf.keras.layers.Dense(250, activation='relu'),
# Second hidden layer with 250 neurons and ReLU activation
        tf.keras.layers.Dense(250, activation='relu'),
# Third hidden layer with 250 neurons and ReLU activation
        tf.keras.layers.Dropout(0.5),  # Dropout layer with dropout rate of 50%
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model
```

### b. Di semua layer

```
# Function to build the DNN model with one hidden layer
def build_model_one_hidden_with_dropout():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(1000, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # Hidden layer with 1000 neurons and
ReLU activation
        tf.keras.layers.Dropout(0.5),  # Dropout layer with dropout rate of 50%
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')          # Output layer with sigmoid activation
    ])
    return model

# Function to build the DNN model with two hidden layer
def build_model_two_hidden_with_dropout():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(500, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # 1st hidden layer with 500 neurons
and ReLU activation
        tf.keras.layers.Dense(500, activation='relu'),
# 2nd hidden layer with 500 neurons and ReLU activation
```

```
        tf.keras.layers.Dropout(0.5),  # Dropout layer with dropout rate of 50%
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model

# Function to build the DNN model with three hidden layer
def build_model_three_hidden_with_dropout():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(250, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # First hidden layer with 250 neurons
and ReLU activation
        tf.keras.layers.Dense(250, activation='relu'),
# Second hidden layer with 250 neurons and ReLU activation
        tf.keras.layers.Dense(250, activation='relu'),
# Third hidden layer with 250 neurons and ReLU activation
        tf.keras.layers.Dropout(0.5),  # Dropout layer with dropout rate of 50%
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model

# Function to build the DNN model with four hidden layer
def build_model_four_hidden_with_dropout():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(100, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # First hidden layer with 100 neurons
and ReLU activation
        tf.keras.layers.Dense(100, activation='relu'),
# Second hidden layer with 100 neurons and ReLU activation
        tf.keras.layers.Dense(100, activation='relu'),
# Third hidden layer with 100 neurons and ReLU activation
        tf.keras.layers.Dense(100, activation='relu'),
# Fourth hidden layer with 100 neurons and ReLU activation
        tf.keras.layers.Dropout(0.5),  # Dropout layer with dropout rate of 50%
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model
```

## c. Hasil

Hasil terbaik test accuracy yaitu tetap pada SGD Optimizer adalah pada **layer3** Dengan Validation Accuracy **0.70 dan** Test Accuracy **0.68**

| | Optimizer | Hidden Layers | Validation Accuracy | Test Accuracy |
|---|---|---|---|---|
| 0 | SGD | 1 | 0.75 | 0.64 |
| 1 | SGD | 2 | 0.75 | 0.64 |
| 2 | SGD | 3 | 0.70 | 0.68 |
| 3 | SGD | 4 | 0.70 | 0.64 |
| 4 | ADAM | 1 | 0.70 | 0.60 |
| 5 | ADAM | 2 | 0.75 | 0.64 |
| 6 | ADAM | 3 | 0.65 | 0.56 |
| 7 | ADAM | 4 | 0.75 | 0.48 |
| 8 | AdaGrad | 1 | 0.70 | 0.60 |
| 9 | AdaGrad | 2 | 0.75 | 0.64 |
| 10 | AdaGrad | 3 | 0.65 | 0.56 |
| 11 | AdaGrad | 4 | 0.75 | 0.48 |
| 12 | RMSprop | 1 | 0.70 | 0.60 |
| 13 | RMSprop | 2 | 0.75 | 0.64 |
| 14 | RMSprop | 3 | 0.65 | 0.56 |
| 15 | RMSprop | 4 | 0.75 | 0.48 |
| 16 | SGD+D | 1 | 0.70 | 0.60 |
| 17 | SGD+D | 2 | 0.75 | 0.60 |
| 18 | SGD+D | 3 | 0.75 | 0.56 |
| 19 | SGD+D | 4 | 0.80 | 0.52 |

Hasil validation accuracy naik menjadi **0.80** pada SGD Optimizer layer 4 dengan menambahkan dropout 50%.