MILESTONE A : FORWARD PROPAGATION
IF4072 PEMBELAJARAN MESIN LANJUT

Laporan

Diajukan untuk memenuhi tugas mata kuliah

IF4072 Pembelajaran Mesin Lanjut

Oleh

Christopher Chandrasaputra	13519074
Billy Julius	13519094
Akeyla Pradia Naufal	13519178

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

I. Penjelasan Kode Program

Berikut adalah penjelasan kode program yang dibuat sebagai hasil implementasi *forward propagation* di *milestone B* ini. Penjelasan kode program akan dibagi berdasarkan beberapa bagian dari program, yaitu *preprocessing image*, *layers*, dan *model*.

I.1. *Preprocessing Image*

Sebelum memulai proses prediksi atau pelatihan pada model, data yang akan digunakan perlu untuk disamakan guna menyesuaikan dengan bentuk *input* yang sudah ditentukan untuk model. Pada data yang diberikan, terdapat 40 gambar yang terdiri dari 20 gambar kucing dan 20 gambar anjing. Dapat diketahui dari data tersebut bahwa data yang diberikan belum memiliki ukuran yang seragam. Oleh karena itu, dibuat sebuah *class* yang ditujukan untuk membuka dan melakukan augmentasi (untuk keperluan *training*) yaitu *ImageConvert*.

ImageConvert merupakan *class* yang digunakan untuk melakukan konversi file gambar menjadi input yang diinginkan. Terdapat beberapa atribut dalam *ImageConvert* yang dapat digunakan untuk melakukan augmentasi yaitu *rotate* dan *rescale* yang masing-masing secara berurutan digunakan untuk merotasi gambar dan mengubah skala(atau nilai) warna dari *pixel*. Dengan salah satu *method* yang sudah diimplementasikan, yaitu *from_directory*, akan dihasilkan sebuah *class* untuk melakukan iterasi pada setiap gambar(data) yang ada dalam sebuah *directory*, kelas tersebut adalah *ImageDirectoryIterator*. *Method* tersebut juga akan secara otomatis melakukan encoding pada label yang ada di dalam dataset *directory*.

Hasil dari *method from_directory* yang dimiliki *class ImageConvert* adalah *class ImageDirectoryIterator*. Tujuan utama dari *class* ini adalah mempersiapkan gambar yang akan digunakan sebagai *input* tanpa harus menghabiskan memori untuk membuka seluruh *dataset* secara bersamaan. Cara kerja dari *class* ini adalah adanya iterasi fungsi *next(self)* pada *class*. Fungsi *next(self)* akan menyimpan index yang akan pada atribut kelas itu sendiri dan akan bergerak untuk memproses dan mengembalikan gambar yang ada di dalam atribut *shuffle_data_label*. Untuk data yang diproses dalam fungsi *next(self)* akan dilakukan tahapan *open*, *resize*, *pad*, *rotate*, *convert*, dan *rescale* secara berurutan. Hasil yang akan dikeluarkan dari fungsi *next(self)* adalah sebuah objek *dictionary* yang memiliki key *"label"* untuk value label dari data yang terpilih dan key *"data"* untuk data yang telah diproses dalam bentuk *numpy ndarray*.

I.2. Layers

Sebuah model neural network tentu perlu memiliki *layer* di dalamnya, baik hanya terdiri dari *input* dan *output layer*, atau terdiri dari *input*, *hidden*, dan *output layer* yang membuatnya menjadi sebuah model *deep learning*. Dalam tugas ini, terdapat beberapa *layer* yang diimplementasikan yaitu *Conv2D, InputLayer, Flatten,* dan *Dense*. Semua *layer* merupakan implementasi dari *abstract class Layer*.

Abstract class Layer merupakan kelas yang memiliki *attributes* dan *methods* yang harus diimplementasikan pada kelas-kelas yang menjadi anaknya. Untuk penjelasan *methods* dan *attributes* yang dimiliki *class* ini adalah sebagai berikut.

Tipe	Nama	Deskripsi
attribute	*name*	nama dari *layer*
attribute	*algorithm*	algoritma yang dipakai *layer*
attribute	*input*	riwayat *input* yang diterima *layer*
attribute	*output*	riwayat *output* yang diterima *layer*
attribute	*input_shape*	bentuk *input* yang diterima *layer*
attribute	*output_shape*	bentuk *output* yang diterima *layer*
method	*calculate*	melakukan kalkulasi pada input untuk mengeluarkan output
method	*update*	melakukan pembaruan bobot (jika ada)
method	*compile*	melakukan kompilasi untuk keperluan kompilasi model (untuk menentukan bentuk *output*)
method	*calculate_output_shape*	menghitung bentuk *output* berdasarkan bentuk *input* yang disimpan

Class Conv2D merupakan kelas yang digunakan untuk melakukan konvolusi gambar. Kelas ini memiliki tiga tahap dalam melakukan kalkulasi yaitu tahap konvolusi, deteksi, dan *pooling*. Terdapat beberapa tambahan atribut dan *method* pada kelas ini. Berikut adalah penjelasannya.

Tipe	Nama	Deskripsi
attribute	*num_of_filters*	Jumlah filter yang akan digunakan untuk melakukan

Tipe	Nama	Deskripsi
		proses konvolusi
attribute	*conv_kernel_size*	Ukuran kernel(filter) yang akan dipakai untuk proses konvolusi
attribute	*conv_padding_size*	Ukuran padding yang akan dipakai untuk proses konvolusi
attribute	*conv_stride*	Jarak pergeseran yang akan dilakukan untuk proses konvolusi
attribute	*conv_filters*	Menyimpan bobot(kernel) dari setiap filter dan bias
attribute	*conv_output_shape*	Menyimpan bentuk output dari proses konvolusi
attribute	*conv_output*	Menyimpan riwayat *output* dari proses konvolusi
attribute	*algorithm*	Algoritma fungsi detektor yang digunakan
attribute	*detector_output*	Menyimpan hasil perhitungan setelah proses deteksi
attribute	*pool_kernel_size*	Ukuran kernel(filter) yang akan dipakai untuk proses *pooling*
attribute	*pool_stride*	Jarak pergeseran yang akan dilakukan untuk proses konvolusi
attribute	*pool_mode*	Metode *pooling* yang digunakan
attribute	*deltas_wrt_filters*	Menyimpan gradien fungsi *error* terhadap filter
attribute	*deltas_wrt_inputs*	Menyimpan gradien fungsi *error* terhadap input
attribute	*delta_pools*	Menyimpan gradien fungsi *error* terhadap hasil *pooling*
attribute	*delta_detectors*	Menyimpan gradien fungsi *error* terhadap hasil proses deteksi
method	*convolve*	Melakukan proses konvolusi
method	*detect*	Melakukan proses deteksi
method	*pool*	Melakukan proses *pooling*
method	*generate_filters*	Menginisiasi bobot(kernel) dari setiap filter dan bias

Class Dense merupakan kelas *layer* yang digunakan pada *artificial neural network* pada umumnya. *Layer* ini biasanya digunakan sebagai *layer* output dengan mengubah fungsi aktivasinya untuk menyesuaikan label. Terdapat neuron-neuron yang menyimpan bobot untuk melakukan perhitungan nantinya. Pada implementasi tugas ini, bobot disimpan dalam bentuk matriks yang ditujukan untuk mempercepat perhitungan(inferensi) nantinya. Terdapat beberapa tambahan atribut dan *method* pada kelas ini. Berikut adalah penjelasannya.

Tipe	Nama	Deskripsi
attribute	*num_of_units*	Jumlah *unit* (*neuron*) yang ada di dalam *layer*
attribute	*weights*	Menyimpan bobot dari seluruh *unit* pada *layer* dalam bentuk matriks
attribute	*deltas_wrt_inputs*	Menyimpan gradien fungsi *error* terhadap input
method	*generate_weights*	Menginisiasi bobot dari setiap *unit* dalam *layer*

Class Flatten merupakan kelas *layer* yang digunakan untuk melakukan konversi multidimensi *tensor* menjadi *tensor* dengan satu dimensi. Kelas ini tidak memiliki tambahan *attributes* atau *methods* tetapi terdapat beberapa yang tidak diimplementasikan seperti *algorithm attribute* dan *update method* karena tidak adanya algoritma yang diperlukan serta bobot yang perlu diperbarui. Meskipun demikian, untuk mempermudah proses *backpropagation* nantinya, kedua hal tersebut tetap ada namun diberikan nilai *None* pada *attribute* dan dilakukan *pass* pada *method*.

Class InputLayer merupakan kelas *layer* yang digunakan untuk melakukan pengecekan bentuk data yang diberikan ke model. Sama seperti *Flatten*, *layer* ini tidak memiliki bobot sehingga tidak mengimplementasikan *update method* dan *algorithm attribute*.

I.3. Model

Untuk menyimpan seluruh *layer* dan membuat *layer-layer* tersebut dapat bekerja sama, terdapat sebuah kelas yaitu *Sequential*. Kelas *Sequential* ini bertujuan untuk menyalurkan data dari satu *layer* ke *layer* selanjutnya. Karena bentuk *input* dari setiap layer tergantung dari bentuk *output layer* sebelumnya, maka diperlukan penyampaian informasi tersebut oleh kelas ini. Berikut penjelasan *attributes* dan *methods* yang dimiliki oleh *class Sequential*.

Tipe	Nama	Deskripsi
attribute	*name*	Nama dari model

Tipe	Nama	Deskripsi
attribute	*layers*	List dari *layers* yang ada dalam model secara berurutan
attribute	*input_shape*	Bentuk *input* yang diterima *layer*
attribute	*output_shape*	Bentuk *output* yang diterima *layer*
attribute	*optimizer*	Optimasi yang digunakan untuk training
attribute	*metrics*	Metrik yang dipakai
method	*compile*	Melakukan kompilasi pada setiap *layer* dan menyampaikan informasi bentuk *output* dari hasil kompilasi satu *layer* ke *layer* selanjutnya
method	*fit*	Melakukan *training* pada model berdasarkan data dan label yang diberikan
method	*predict*	Melakukan prediksi pada model berdasarkan data yang diberikan
method	*save*	Menyimpan model ke dalam sebuah file
method	*load*	Membuka model dari sebuah file
method	*summary*	Menampilkan *layer-layer* beserta informasinya yang ada secara berurutan
method	*get_layer*	Mengembalikan *layer* berdasarkan indeks atau nama yang diberikan

II. Contoh Hasil Prediksi

Berikut adalah contoh hasil prediksi yang dihasilkan dengan menggunakan model *convolutional neural network* buatan penulis.

IMPORTS

```
import os
from classes.layers.Conv2D import Conv2D
from classes.layers.Dense import Dense
from classes.layers.Flatten import Flatten
from classes.layers.Input import InputLayer
from classes.models.Sequential import Sequential
from classes.utils.ImageConvert import ImageConvert
```

Pertama, dilakukan impor kelas yang sudah diimplementasikan.

DATA PREPROCESSING

```
ic = ImageConvert(
        rotate=30.,
        rescale=1./255.)
data_gen = ic.from_directory(os.path.join('.','data','test'), (256,256), mode='binary', color_mode='rgb')
```

Kemudian, data training dan tes akan dibangkitkan.

MODEL

Instantiating

```
model = Sequential([
        InputLayer(input_shape=(256,256,3)),
        Conv2D(2, (16, 16), activation='relu'),
        Flatten(),
        Dense(16, activation='relu'),
        Dense(1, activation='sigmoid')
    ])

model.compile()
```

Model yang digunakan terdiri dari *layers* berikut:

1. *Input layer*, yang menerima masukan berukuran 256 x 256 x 3
2. *Convolution layer* dengan kernel konvolusi berukuran 16 x 16 dan setelah itu terdapat *max pooling* berukuran 2 x 2
3. *Flatten layer*
4. *Dense layer* dengan 16 *hidden neuron* dan fungsi aktivasi ReLU
5. Layer output dengan fungsi aktivasi sigmoid

```python
# CULL DATA
data_gen.shuffle_data_label = data_gen.shuffle_data_label[:10]

# TEST
model.predict(data_gen)
```

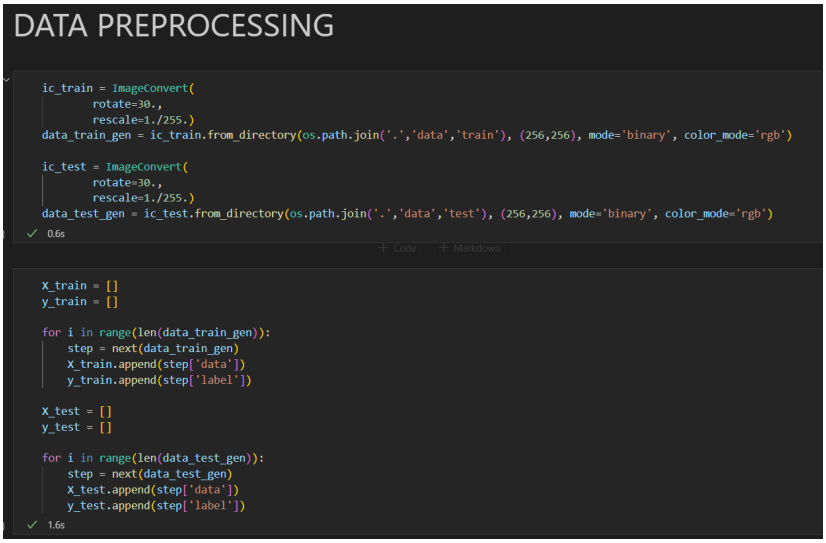
```
Step: 1/10
Step: 2/10
Step: 3/10
Step: 4/10
Step: 5/10
Step: 6/10
Step: 7/10
Step: 8/10
Step: 9/10
Step: 10/10

{'results': [array([1.]),
  array([1.]),
  array([1.]),
  array([1.]),
  array([1.]),
  array([1.]),
  array([1.]),
  array([1.]),
  array([1.]),
  array([1.])],
 'true_labels': [1, 1, 0, 0, 1, 0, 0, 0, 0, 0]}
```

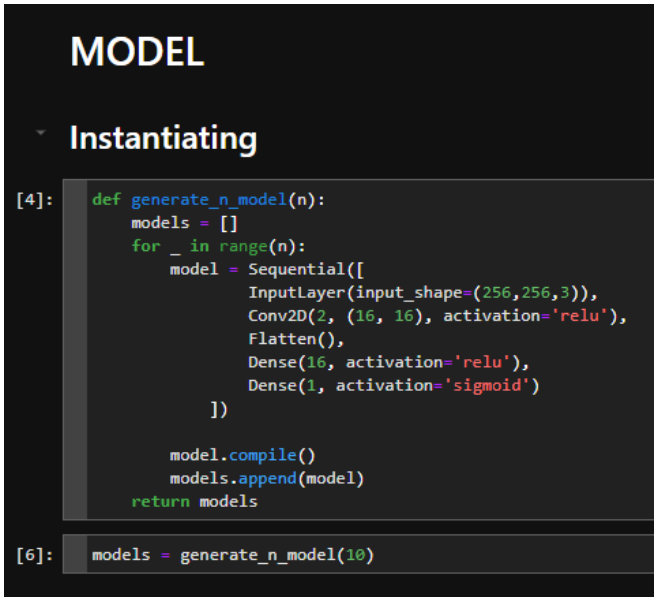
Prediksi yang dilakukan oleh model masih selalu bernilai 1 karena belum ada *backpropagation* yang dapat mengubah nilai bobot.

III. Hasil Eksperimen

Berikut adalah hasil eksperimen dengan menggunakan *model CNN* buatan kelompok penulis.



Dilakukan *data preprocessing* yang meliputi proses pemuatan gambar-gambar untuk *training* dan juga untuk *testing*. Dalam melakukan *preprocessing data*, dilakukan proses augmentasi gambar seperti melakukan *rescale* yang menormalisasikan nilai dari setiap *pixel* dan rotasi sebesar 30 derajat, mengingat jumlah data yang sangat sedikit untuk pembelajaran



Kemudian dilakukan instansiasi model *Sequential* sebanyak 10 model dan juga dilakukan *compile*. Instansiasi sebanyak 10 model ditujukan untuk melakukan proses *10-fold cross validation* nanti.

Training

```
[7]:
def cross_validation_split(data, label, n):
    data_split = [list(_) for _ in np.array_split(data, n)]
    label_split = [list(_) for _ in np.array_split(label, n)]

    train_split = []
    valid_split = []
    label_train_split = []
    label_valid_split = []

    for i in range(len(data_split)):
        # Separate validation k fold split
        for data in data_split[i]:
            valid_split.append(data)
        for label in label_split[i]:
            label_valid_split.append(label)
        # Union train k fold split
        temp_train_data = []
        for data in data_split[:i]:
            # for data in set:
            temp_train_data += list(data)
        for data in data_split[i+1:]:
            # for data in set:
            temp_train_data += list(data)
        train_split.append(temp_train_data)

        temp_train_label = []
        for set in label_split[:i]:
            for label in set:
                temp_train_label.append(label)
        for set in label_split[i+1:]:
            for label in set:
                temp_train_label.append(label)
        label_train_split.append(temp_train_label)

    return train_split, valid_split, label_train_split, label_valid_split

train_split, valid_split,\
    label_train_split, label_valid_split = cross_validation_split(X_train, y_train, 10)
```

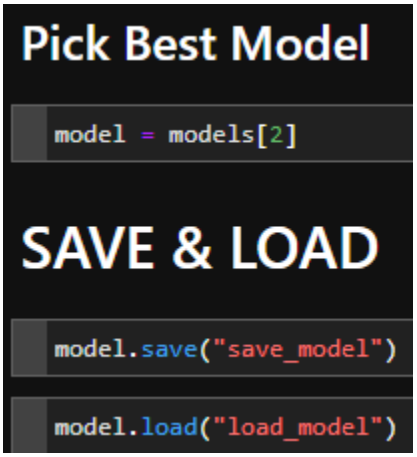
```
[*]:
for i in range(len(models)):
    models[i].fit(train_split[i], label_train_split[i], 2, 10, 0.2)
```

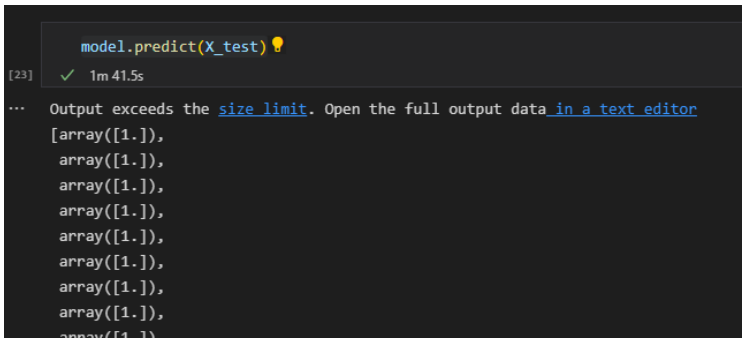
Setelah itu, dilakukan proses *training* dengan memanggil fungsi *fit* dari masing-masing *model* pada data test yang sudah dipisahkan menggunakan fungsi *cross_validation_split*. Masing-masing model akan dilatih menggunakan komposisi data yang berbeda karena metode *10-fold cross validation* digunakan.

Testing

```
# TEST
losses = []
for model in models:
    y_pred = model.predict(X_test)
    loss = BinaryCrossEntropy.BinaryCrossEntropy(y_test, y_pred)
    losses.append(np.average(loss))
```

Setelah proses *training* dilakukan *testing* dengan menggunakan fungsi *predict* pada masing-masing *model* untuk memprediksi *label* dari *test data (X_test)*. Dari hasil predict, dapat diketahui loss dari masing-masing model. Di akhir, dilakukan proses perhitungan *loss* dengan menggunakan *binary crossentropy*. Setelah itu, dilakukan proses *save* dan *load* model untuk menyimpan hasil model terbaik.





IV. Pembagian Tugas

Berikut adalah pembagian tugas yang dilakukan dalam pelaksanaan pengerjaan *milestone A*.

Nama	NIM	Pembagian Tugas
Christopher Chandrasaputra	13519074	● *Conv2D Layer* ● *Sequential Model* ● *Refactor* ● Laporan
Billy Julius	13519094	● *Preprocessing image* ● Laporan
Akeyla Pradia Naufal	13519178	● *Dense Layer* ● *Flatten Layer* ● Laporan

Berikut adalah pembagian tugas yang dilakukan dalam pelaksanaan pengerjaan *milestone B*.

Nama	NIM	Pembagian Tugas
Christopher Chandrasaputra	13519074	● *Refactor* proses konvolusi ● Merancang struktur file untuk model ● Implementasi metode *Save/Load Model* ● *10-fold Cross Validation* ● Laporan
Billy Julius	13519094	● *Backpropagation Dense Layer* ● *Connect Dense Backpropagation to Convolution Layer* ● Laporan
Akeyla Pradia Naufal	13519178	● *Backpropagation Conv2D Layer* ● Laporan