Laporan Tugas Besar Convolutional Neural Network (Milestone B)

IF4074 Pembelajaran Mesin Lanjut

Forward Propagation

Disusun oleh:

Mahesa Lizardy	13520116
Bryan Amirul Husna	13520146

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

2023

Daftar Isi

Daftar Isi **2**
Pendahuluan **3**
A. CNN 3
B. Forward Propagation 4
C. Backpropagation 4
Implementasi **6**
A. Convolutional Layer 6
B. Dense 9
C. Flatten 11
D. Model 12
Pengujian **14**
A. Rancangan Pengujian 14
B. Hasil Prediksi 16
Lampiran 17
A. Pembagian Tugas 17

Pendahuluan

A. CNN

Convolutional Neural Network (CNN) adalah jenis arsitektur jaringan saraf tiruan yang secara khusus dirancang untuk memproses data grid, seperti gambar dan video. CNN telah menjadi bagian integral dari berbagai aplikasi dalam pengolahan citra, pengenalan pola, visi komputer, dan banyak lagi.

Keunggulan utama CNN terletak pada kemampuannya untuk secara otomatis mengekstraksi fitur-fitur penting dari data grid ini, yang memungkinkan model untuk memahami hierarki pola dan struktur dalam data visual. Selain itu, CNN dapat menangkap fitur-fitur spasial yang membuatnya sangat efektif dalam memproses informasi berdimensi tinggi. Selain itu, CNN juga dikenal dapat meningkatkan kemampuan generalisasi dan kecepatan dalam pemrosesan data. Berikut adalah beberapa komponen utama dalam arsitektur CNN:

1. *Convolution Layer*

   Layer ini adalah inti dari CNN. Mereka menggunakan operasi konvolusi untuk menerapkan filter atau kernel pada input gambar. Ini membantu dalam mengekstraksi fitur-fitur seperti tepi, garis, dan pola visual lainnya.

2. *Detector Layer*

   Layer ini digunakan untuk mengurangi dimensi spasial dari representasi yang dihasilkan oleh layer konvolusi. Max-pooling adalah jenis pooling yang paling umum, yang memilih nilai maksimum dari segmen-segmen data dalam grid. Juga terdapat Average poling yang menghitung rata–rata dari segmen-segmen data dalam grid.

3. *Pooling Layer*

   Sebelum memasuki lapisan fully connected, representasi dari lapisan-lapisan konvolusi dan pooling harus diubah menjadi vektor satu dimensi.

CNN telah membawa revolusi dalam bidang pengolahan citra dan visi komputer, serta digunakan dalam berbagai aplikasi seperti pengenalan wajah, deteksi objek, pengenalan tulisan tangan, dan banyak lagi. Mereka terus berkembang dengan peningkatan dalam arsitektur dan teknik pelatihan.

B. Forward Propagation

Forward propagation adalah salah satu langkah kunci dalam proses pelatihan dan penggunaan *Artificial Neural Network* (ANN), termasuk jugas *Convolutional Neural Network* (CNN). Forward propagation adalah proses di mana data input disebarkan melalui jaringan neural network untuk menghasilkan prediksi atau output yang digunakan dalam tugas tertentu.

Dalam CNN atau jaringan saraf lainnya, forward propagation adalah langkah pertama dalam memproses data. Ini melibatkan aliran data melalui berbagai lapisan jaringan, termasuk lapisan konvolusi, pooling, dan dense layers. Selama proses ini, data input mengalami serangkaian transformasi matematika, yang melibatkan penggabungan, konvolusi, penjumlahan bobot, dan fungsi aktivasi, sehingga mewakili informasi secara hierarkis dan abstrak. Output dari langkah forward propagation ini adalah prediksi atau representasi dari data input yang dapat digunakan untuk tugas seperti klasifikasi, deteksi objek, atau regresi.

C. Backpropagation

Backpropagation, atau disingkat sebagai "backprop," adalah salah satu algoritma utama dalam pembelajaran mesin yang digunakan untuk melatih *Artificial Neural Network* (ANN). Tujuan utama dari backpropagation adalah mengoptimalkan parameter dalam jaringan saraf sehingga jaringan dapat mempelajari pola-pola yang ada dalam data input.

Proses backpropagation melibatkan dua tahap utama: propagasi maju (forward propagation) dan propagasi mundur (backward propagation).

1. Propagasi Maju (Forward Propagation):

   Pada tahap ini, data input disebarkan melalui jaringan saraf, melalui berbagai lapisan (atau "neurons") hingga mencapai lapisan output. Setiap neuron dalam jaringan memiliki bobot (weights) dan bias (nilai tambahan). Input dikalikan dengan bobot dan ditambah dengan bias, kemudian hasilnya diolah menggunakan fungsi aktivasi. Fungsi aktivasi ini memberikan keluaran (output) dari neuron. Hasil output ini kemudian digunakan untuk menghitung nilai kesalahan (error) dengan membandingkan output prediksi dengan target yang seharusnya. Error ini akan memberikan gambaran seberapa "salah" prediksi jaringan terhadap data input tertentu.

2. Propagasi Mundur (Backward Propagation):

   Setelah error dihitung, langkah berikutnya adalah menghitung gradien dari error terhadap bobot dan bias dalam jaringan. Gradien ini memberikan informasi tentang seberapa besar dan arah perubahan yang harus dilakukan pada bobot dan bias untuk mengurangi kesalahan. Backpropagation menggunakan algoritma turunan untuk

menghitung gradien. Backpropagation akan mengukur seberapa sensitif output jaringan terhadap perubahan kecil dalam bobot dan bias. Gradien ini kemudian akan digunakan untuk memperbarui bobot dan bias dalam jaringan, dengan tujuan mengurangi kesalahan pada tahap propagasi maju berikutnya.

Proses ini diulang melalui beberapa iterasi (epoch) dengan dataset pelatihan, di mana jaringan secara bertahap mengoptimalkan bobot dan bias nya untuk membuat prediksi yang lebih akurat. Dengan meminimalkan kesalahan melalui backpropagation, jaringan saraf dapat belajar mewakili pola-pola yang kompleks dalam data, memungkinkan aplikasi seperti pengenalan gambar, bahasa alami, dan banyak lagi. Backpropagation merupakan landasan bagi banyak jenis arsitektur jaringan saraf, termasuk CNN (Convolutional Neural Networks) untuk pengenalan gambar dan RNN (Recurrent Neural Networks).

Implementasi

A. Convolutional Layer

Pada bagian ini merupakan implementasi dari Convolution, Detector, dan Pooling.

1. Convolution

   Implementasi Convolution terdapat pada class Convolution. Class ini memiliki beberapa fungsi utama sebagai berikut.

   a. __init__(self, input_size, padding_size, filter_size, num_filters, stride)

   Fungsi ini merupakan konstruktor kelas Convolution. Ini digunakan untuk menginisialisasi berbagai parameter yang diperlukan untuk lapisan konvolusi. Parameter pada inisialisasi kelas convolution adalah sebagai berikut

Parameter	**Description**
input_size	Ukuran input (panjang, lebar, jumlah saluran).
padding_size	Ukuran padding (jumlah sel yang ditambahkan di sekitar input).
filter_size	Ukuran filter konvolusi (panjang dan lebar).
num_filters	Jumlah filter yang akan digunakan.
stride	Jarak (langkah) antara filter yang digunakan saat menggeser melalui input.

   Selain inisialisasi parameter tersebut akan dilakukan juga inisialisasi ukuran output, inisialisasi filter dengan random, serta inisialisasi bias dengan nol.

   b. forward(self, input)

   Fungsi ini melakukan operasi konvolusi pada input yang diberikan. Pertama, jika padding diperlukan (jika padding_size lebih dari nol), maka input akan dipad dengan nilai nol. Kemudian, loop dilakukan melalui seluruh input dengan langkah sejauh stride untuk melakukan operasi konvolusi pada input dengan filter yang telah diinisialisasi. Hasil dari operasi konvolusi disimpan dalam matriks output. Hasil akhir adalah matriks output yang berisi hasil konvolusi dari input dengan filter-filter yang sesuai.

   c. backward(self, front_deltas, label, front_weights

   Fungsi backward pada kelas Convolution adalah bagian dari algoritma backpropagation yang digunakan dalam CNN. Fungsi ini bertanggung jawab untuk menghitung gradien dari fungsi kerugian (loss function) terhadap bobot (weights) dan bias lapisan konvolusi, serta mengembalikan nilai delta (gradien) yang akan

digunakan dalam lapisan sebelumnya selama proses backpropagation. Pada awal fungsi, derivatif dari fungsi aktivasi ReLU (Rectified Linear Unit) digunakan. Pertama akan dilakukan perhitungan nilai turunan relu dari net. Derivatif ini diperlukan dalam perhitungan gradien untuk lapisan konvolusi yang menggunakan aktivasi ReLU. Selanjutnya, Jika front_deltas tidak disediakan (bernilai None), berarti lapisan ini merupakan lapisan konvolusi terakhir dalam jaringan dan menerima input langsung dari data. Dalam hal ini, delta dihitung berdasarkan perbedaan antara nilai target (label) dan hasil output lapisan konvolusi (self.output). Delta ini kemudian dikalikan dengan hasil derivatif ReLU. Jika front_deltas disediakan, delta dihitung berdasarkan delta dari lapisan yang berada di depannya. Proses ini melibatkan konvolusi dari delta pada lapisan depan dengan bobot (front_weights). Delta ini juga dikalikan dengan derivatif ReLU. Selanjutnya dilakukan penanganan stride dan dilasi. Jika stride tidak sama dengan 1, maka dilakukan penghitungan yang melibatkan stride. Delta dihitung dengan memperhitungkan dilasi (dilated convolution) dari delta pada lapisan depan. Jika stride sama dengan 1, delta dihitung tanpa dilasi. Perhitungan Gradien dan Pembaruan Bobot: Gradien dari fungsi kerugian terhadap bobot (self.gradients) dihitung dengan menggunakan fungsi validconv antara input (setelah dilakukan padding jika diperlukan) dan delta yang sudah di rotate 180 derajat). Gradien ini kemudian ditambahkan pada gradien lapisan konvolusi. Pembaruan pada bias (self.bias) dihitung dengan menambahkan nilai negatif dari delta pada setiap filter. Gradien ini digunakan dalam proses optimisasi, seperti untuk memperbarui bobot dan bias lapisan konvolusi.

d. update_weights(self, learning_rate)

   Fungsi ini akan melakukan update bobot (filter dan bias) berdasarkan learning rate.

e. validconv(self, m1, m2)

   Fungsi ini menghitung konvolusi antara dua matriks m1 dan m2 menggunakan metode valid convolution. Prosesnya adalah sebagai berikut: m2 diputar 180 derajat menggunakan metode rotate180. Dimensi matriks input m1 dan m2 diambil. Jika m1 adalah matriks 3D (misalnya, dalam kasus gambar berwarna dengan saluran warna RGB), dimensi ketiga (saluran) juga diambil. Matriks output (result) diinisialisasi dengan nol, dengan ukuran yang sesuai berdasarkan dimensi m1 dan m2. Loop dilakukan melalui matriks input m1. Pada setiap iterasi, dilakukan pemotongan matriks m1 dengan ukuran yang sama dengan m2. Hasil konvolusi dihitung dengan mengalikan setiap elemen matriks potongan dengan elemen matriks m2, kemudian menjumlahkan hasilnya untuk mendapatkan nilai konvolusi. Nilai konvolusi disimpan di posisi yang sesuai dalam matriks output result.

f. fullconv(self, m1, m2):

Fungsi ini menghitung konvolusi antara dua matriks m1 dan m2 menggunakan metode full convolution. Prosesnya hampir sama dengan valid convolution, namun dengan perbedaan pada langkah-langkah berikut. m2 diputar 180 derajat menggunakan fungsi rotate180. Dimensi matriks input m1 dan m2 diambil. Jika m1 adalah matriks 3D (misalnya, gambar berwarna dengan saluran warna RGB), dimensi ketiga (saluran) juga diambil. Matriks output (result) diinisialisasi dengan nol, dengan ukuran yang diperoleh dari penjumlahan ukuran m1 dan m2 dikurangi satu. Loop dilakukan melalui matriks output result. Pada setiap iterasi, dilakukan pemotongan matriks m1 dan m2 dengan ukuran yang sesuai. Potongan matriks ini dikalikan elemen demi elemen, dan hasilnya dijumlahkan untuk mendapatkan nilai konvolusi. Nilai konvolusi disimpan di posisi yang sesuai dalam matriks output result.

g. rotate180(self, m)
   Fungsi ini memutar matriks m sebanyak 180 derajat.

h. relu(self, x)
   Fungsi ini menghitung nilai fungsi aktivasi ReLU dari x

i. relu_derivative(self, x):
   Fungsi ini menghitung nilai turunan fungsi aktivasi ReLU dari x

j. set_deltas(self, delta)
   Fungsi ini menginisialisasi nilai deltas dengan nilai delta yang diberikan. Bentuk dari deltas adalah ukuran output setelah proses konvolusi.

k. set_gradients(self, gradient)
   Fungsi ini menginisialisasi nilai gradients dengan nilai gradien yang diberikan. Bentuk dari gradients adalah jumlah filter, ukuran filter, dan jumlah saluran pada input.

l. getModel(self)
   Fungsi ini mengembalikan model konvolusi dalam bentuk dictionary. Model ini berisi kernel filter dan bias dalam bentuk list. Fungsi ini digunakan untuk menyimpan model konvolusi ke dalam bentuk json.

m. setFilter(self, filter)
   Fungsi ini memungkinkan Anda untuk mengatur filter konvolusi dengan filter yang diberikan sebagai parameter.

n. showModel(self):
   Fungsi yang digunakan untuk menampilkan informasi tentang lapisan konvolusi pada terminal

2. Detector

   Implementasi Detector terdapat pada class Convolution yaitu pada fungsi forward. Hasil dari operasi konvolusi sebelum disimpan ke matriks output dilakukan ungsi aktivasi ReLU diaplikasikan. Langkah ini dilakukan untuk mengurangi beban komputasi yang timbul jika lapisan Detector dijalankan secara terpisah.

3. Pooling

   Implementasi Pooling terdapat pada class Pooling. Class ini memiliki beberapa fungsi utama sebagai berikut.

   a. __init__(self, filter_size, stride, mode)

      Konstruktor kelas ini digunakan untuk menginisialisasi parameter operasi pooling, termasuk ukuran filter pooling, langkah (stride), dan mode (max atau average).

   b. forward(self, input)

      Fungsi forward digunakan untuk melakukan operasi *polling* pada input yang diberikan. Ini mengambil input dan menghasilkan output berdasarkan parameter yang diatur pada saat inisialisasi. Pertama, itu menghitung ukuran output berdasarkan ukuran input, ukuran filter, dan langkah yang telah diatur. Kemudian, itu membuat matriks output dengan ukuran yang sesuai. Loop dilakukan melalui input dengan langkah sejauh stride, dan untuk setiap jendela filter di dalam input, itu melakukan operasi pooling sesuai dengan mode yang telah ditentukan (max atau average). Hasil *polling* disimpan dalam output dan direturn.

   c. backward(self, front_deltas, label, front_weights)

      Fungsi backward pada pooling terdapat dua yaitu untuk *average pooling* dan *max pooling*. Pada *max pooling*

   d. getModel(self)

      Fungsi ini mengembalikan deskripsi model pooling dalam bentuk dictionary. Fungsi ini digunakan untuk menyimpan model konvolusi ke dalam bentuk json.

   e. showModel(self)

      Fungsi ini digunakan untuk menampilkan informasi tentang lapisan pooling.

B. Dense

Implementasi Dense terdapat pada class Dense. Class ini memiliki beberapa fungsi utama sebagai berikut.

a. __init__(self, num_units, activation_function, input_size = None)

Fungsi ini merupakan konstruktor kelas Dense. Fungsi ini akan menginisialisasi objek lapisan dengan parameter seperti jumlah unit (neuron), fungsi aktivasi yang digunakan, dan ukuran input jika sudah diketahui. Fungsi ini juga menginisialisasi bobot (weights) dan bias dengan nilai-nilai acak. Selain itu, Fungsi init akan menginisialisasi variabel yang digunakan untuk keperluan backprop seperti nilai input terakhir, net, output, delta, dan gradien.

b. forward(self, input_data)

Fungsi ini akan melakukan operasi forward propagation. Fungsi akan mengambil input data dan mengalokasikan bobot dan bias jika belum diinisialisasi sebelumnya. Kemudian, melakukan perkalian matriks antara input_data dan bobot. Perkalian tersebut akan ditambahkan dengan bias, hasilnya akan disimpan di variabel net untuk keperluan backprop. Berdasarkan fungsi aktivasi yang dipilih (ReLU atau sigmoid) akan dihitung nilai nya berdasarkan hasil sebelumnya untuk disimpan di variabel output dan direturn.

c. backward(self,front_deltas, label, front_weights)

Fungsi ini menghitung gradien untuk lapisan neuron. Jika front_deltas adalah None, ini menunjukkan bahwa ini adalah *output layer*, dan gradien dihitung berdasarkan perbedaan antara label dan output saat ini, dikalikan dengan turunan fungsi aktivasi (ReLU atau sigmoid). Jika front_deltas tidak None, ini menunjukkan bahwa ini adalah *hidden layer*, dan gradien dihitung dengan mengakumulasikan front deltas dan front weights untuk setiap neuron, lalu dikalikan dengan turunan fungsi aktivasi. Gradien diupdate dalam variabel gradients untuk digunakan dalam *update* berikutnya.

d. update_weights(self,front_deltas, learning_rate)

Fungsi ini mengupdate bobot dan bias lapisan neuron berdasarkan gradien yang dihitung. Bobot dan bias diperbarui menggunakan nilai gradien, dikalikan dengan learning_rate. Setelah *update* bobot, gradien di reset menjadi 0, dan nilai *update* bias juga di reset.

e. set_deltas(self, delta)

Fungsi ini menginisialisasi nilai gradien (deltas) dengan nilai delta. Deltas digunakan saat menghitung gradien pada lapisan output.

f. set_gradients(self, gradient)

Fungsi ini menginisialisasi nilai gradien (self.gradients) dengan nilai gradient. Gradien digunakan saat menghitung pembaruan bobot dalam proses pelatihan.

g. relu(self, x)

Fungsi ini menerapkan fungsi aktivasi ReLU (Rectified Linear Unit) pada input x. ReLU mengembalikan nilai maksimum antara 0 dan input.

h. sigmoid(self, x)

Fungsi ini menerapkan fungsi aktivasi sigmoid pada input x, yang menghasilkan nilai antara 0 dan 1.

i. relu_derivative(self, x)

Fungsi ini menghitung nilai turunan fungsi aktivasi ReLU (Rectified Linear Unit) pada input x. turunan ReLU adalah 1 jika x > 0, dan 0 jika x <= 0..

j. sigmoid_derivative(self, x)

Fungsi ini menghitung nilai turunan fungsi aktivasi sigmoid pada input x. Turunan sigmoid adalah nilai sigmoid dari x dikali dengan 1 dikurangi nilai sigmoid dari x.

k. getModel(self)

Fungsi ini mengembalikan representasi model dalam bentuk kamus (dictionary) yang mencakup tipe lapisan (dense), kernel (bobot), bias, dan jenis fungsi aktivasi dalam bentuk json.

l. setModel(self)

Fungsi ini digunakan untuk load data dari file json yang diberikan.

m. setWeights(self, weights)

Fungsi ini memungkinkan pengguna untuk mengatur bobot lapisan dengan bobot yang diberikan sebagai argumen.

n. setBias(self, bias)

Fungsi ini memungkinkan pengguna untuk mengatur bias lapisan dengan nilai bias yang diberikan sebagai argumen.

o. showModel(self, input=1)

Fungsi ini mencetak tampilan sederhana untuk lapisan Dense, menampilkan jumlah unit dalam lapisan, serta jumlah parameter (bobot dan bias) yang ada dalam lapisan ini.

C. Flatten

Implementasi Flatten terdapat pada class Flatten. Class ini memiliki beberapa fungsi utama sebagai berikut.

a. __init__(self)

Ini adalah konstruktor kelas Flatten. Pada konstruktor Ini tidak melakukan apa-apa selain menginisialisasi objek.

b. forward(self, input_data)

Fungsi ini digunakan untuk melakukan operasi forward propagation. Ia mengambil data input yang berupa dimensi berapapun kemudian mengubahnya menjadi satu dimensi dengan cara "melipat" atau "flattening". Ini berarti semua elemen dari input awal akan disusun secara berurutan menjadi satu vektor satu dimensi. Hasil vektor ini adalah representasi data yang akan diteruskan ke lapisan Dense.

c. backward(self, input_data)

Fungsi ini digunakan untuk melakukan operasi backward propagation. Weight dan deltas akan diubah bentuknya ke dalam bentuk input.

d. getModel(self)

Fungsi ini mengembalikan representasi model dalam bentuk kamus (dictionary) yang mencakup tipe lapisan (flatten) dalam bentuk json.

e. showModel(self, input=1)

Fungsi ini mencetak tampilan sederhana untuk lapisan Flatten.

D. Model

Implementasi Model terdapat pada class Model. Class ini memiliki beberapa fungsi utama sebagai berikut.

a. __init__(self)

Fungsi Ini adalah konstruktor kelas Model. Ini menginisialisasi objek dengan sebuah daftar (list) yang akan digunakan untuk menyimpan layer-layer dalam jaringan.

b. predict(self, X)

Fungsi ini digunakan untuk melakukan prediksi. Fungsi menerima X yang merupakan daftar nilai x yang ingin diprediksi, dan menghasilkan suatu daftar nilai prediksi yang bersesuaian dengan tiap x.

c. train_network(self, train_data, label_data, batch_size, lr=0.01, epochs=200)

Fungsi ini digunakan untuk melatih artificial neural network. Fungsi ini akan mengambil data pelatihan (data train dan data label), ukuran batch (batch_size), learning rate (lr), dan jumlah epoch (epochs) sebagai argumen. Selama pelatihan, ia membagi data pelatihan ke dalam batch sesuai dengan batch_size, dan kemudian melakukan proses forward propagation untuk setiap sampel dalam batch menggunakan lapisan-lapisan yang telah ditambahkan ke jaringan. Ini juga menghitung akurasi selama pelatihan dan mencetaknya. Ini mengulangi proses pelatihan sebanyak jumlah epochs.

d. fit(self, X, y, epochs, batch_size, learning_rate):

Fungsi ini Fungsi fit pertama-tama memeriksa apakah X dan y tidak None dan memiliki panjang yang sama. Jika tidak, itu menghasilkan kesalahan. Fungsi ini melakukan pelatihan selama sejumlah epochs yang diberikan. Setiap epoch adalah satu putaran melalui seluruh data pelatihan. Data pelatihan dibagi ke dalam batch dengan ukuran batch_size. Setiap batch diambil dari data pelatihan untuk pelatihan mini-batch. Setiap batch melewati seluruh jaringan menggunakan langkah-langkah "forward propagation". Setiap lapisan (self.layers)

menerapkan operasi forward pada inputnya dan meneruskan hasilnya ke lapisan berikutnya. Setelah tahap forward propagation, langkah "backward propagation" dilakukan untuk menghitung gradien melalui jaringan. Gradien ini digunakan untuk memperbarui bobot lapisan dengan metode gradien turun. Pada setiap lapisan, fungsi backward dipanggil, dan gradien serta deltas dihitung dan disimpan untuk lapisan berikutnya. Setelah selesai menghitung gradien untuk seluruh batch, bobot dan bias diperbarui dengan metode update_weights. Setelah itu, akan dihitung akurasi pada akhir setiap epoch.

e. add(self, layer)

   Fungsi ini digunakan untuk menambahkan layer baru ke dalam jaringan. Layer-layer ini harus telah diinisialisasi sebelumnya.

f. saveModel(self)

   Fungsi ini digunakan untuk melakukan pembacaan model yang disimpan di dalam file json. Fungsi ini akan membaca layer-layer yang terdapat pada model beserta parameter dari layer tersebut.

g. loadModel(self)

   Fungsi ini digunakan untuk menyimpan model jaringan ke dalam file JSON. Ini menciptakan repr

h. showModel(self)

   Fungsi ini mencetak tampilan struktur model jaringan dengan mencetak informasi tentang setiap lapisan, termasuk tipe lapisan, bentuk keluaran (output shape), dan jumlah parameter yang digunakan.

Pengujian

A. Rancangan Pengujian

Pengujian dilakukan pada Panda or Bear Dataset. Dataset tersebut terdiri atas gambar-gambar RGB berukuran 256×256 piksel. Digunakan arsitektur yang terdiri atas satu convolutional layer, satu pooling layer, satu flatten layer, dan satu dense layer. Rincian parameternya terdapat pada gambar di bawah. Tiap weights di tiap layer diinisialisasi secara random. Hasilnya kemudian dibandingkan dengan model neural network yang sama yang dihasilkan menggunakan pustaka Keras, dengan weight dan bias dari model pustaka sendiri diekspor ke model Keras sehingga menghasilkan model yang identik.

Gambar contoh Panda or Panda Dataset

1. Pembelajaran dengan skema split train 90% dan test 10%

Berikut merupakan pembelajaran dengan skema split train 90% dan test 10 %

```
# Buat model, dengan weights bernilai random
samples = next(train_images)
input_shape = samples[0][0].shape

model = Model()
model.add(Convolution(input_size=input_shape, padding_size=1, filter_size=(3, 3, 3), num_filters=2, stride=1))
model.add(Flatten())
model.add(Dense(num_units=7, activation_function="relu"))
model.add(Dense(num_units=1, activation_function="sigmoid"))
```

Gambar struktur model dan parameternya untuk prediksi

2. Pembelajaran dengan skema 10-fold cross validation

```
# Merge inputs and targets
inputs = np.concatenate((X_train, X_test), axis=0)
targets = np.concatenate((y_train, y_test), axis=0)
print(len(inputs))
print(len(targets))
```

```
# K-fold Cross Validation model evaluation
fold_no = 1

for train, test in kfold.split(inputs, targets):
    # Generate a print
    print('------------------------------------------------------------------------')
    print(f'Training for fold {fold_no} ...')

    # Fit data to model
    history = model.fit(inputs[train], targets[train],
                epochs=2, batch_size=4)
```

```
model.fit(X_train, y_train, epochs=10, batch_size=4)
```

```
===== Epoch 1 =====
Accuracy: 0.375
===== Epoch 2 =====
Accuracy: 0.0
===== Epoch 3 =====
Accuracy: 0.0
===== Epoch 4 =====
Accuracy: 0.0
```

Gambar contoh pelatihan, dengan memanggil model.fit

3. Menyimpan Model

```
model.saveModel("model.json")
```

Save Model dapat dilakukan dengan memanggil fungsi model.saveModel()

4. load Model

```
A = model.loadModel("model.json")
```

Load Model dapat dilakukan dengan memanggil fungsi model = Model.loadModel()

B. Hasil Prediksi

Prediksi kemudian dilakukan untuk lima gambar pertama dari dataset pengujian. Hasilnya terdapat pada gambar-gambar di bawah.

```
# Prediksi
y_p = model.predict(X_test[0:5])
print(y_p)
```

```
[[9.89365197e-22]
 [2.47779341e-19]
 [9.92786342e-18]
 [5.11667884e-22]
 [1.73297854e-20]]
```

```
# Kesimpulan prediksi
y_predicted = np.array([0. if y < 0.5 else 1. for y in y_p])
print("Actual y:", y_test[:5])
print("Predicted y:", y_predicted)
```

```
Actual y: [0. 0. 0. 0. 0.]
Predicted y: [0. 0. 0. 0. 0.]
```

Gambar hasil prediksi dengan pustaka sendiri

Lampiran

A. Pembagian Tugas

NIM	Nama	Pembagian Tugas
13520116	Mahesa Lizardy	Forward Propagation, Model, test Model, Laporan
13520146	Bryan Amirul Husna	Backward Propagation, Model, test Model, Laporan