

**IMPLEMENTASI *DEEP LEARNING* MENGGUNAKAN METODE
CONVOLUTIONAL NEURAL NETWORK UNTUK KLASIFIKASI
GAMBAR**

(Studi Kasus: Klasifikasi Gambar Pada Tanaman Anggrek Bulan Putih, Anggrek
Dendrobium, dan Anggrek Ekor Tupai)

TUGAS AKHIR



Disusun Oleh :

Rahayu Kia Sandi Cahaya Putri

14 611 018

**PROGRAM STUDI STATISTIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS ISLAM INDONESIA
YOGYAKARTA
2018**

**IMPLEMENTASI *DEEP LEARNING* MENGGUNAKAN METODE
CONVOLUTIONAL NEURAL NETWORK UNTUK KLASIFIKASI
GAMBAR**

(Studi Kasus: Klasifikasi Gambar Pada Tanaman Anggrek Bulan Putih, Anggrek
Dendrobium, dan Anggrek Ekor Tupai)

TUGAS AKHIR

Diajukan Sebagai Salah Satu Syarat Untuk Memperoleh Gelar Sarjana
Jurusan Statistika



Disusun Oleh :

Rahayu Kia Sandi Cahaya Putri

14 611 018

**PROGRAM STUDI STATISTIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS ISLAM INDONESIA
YOGYAKARTA
2018**

HALAMAN PERSETUJUAN DOSEN PEMBIMBING

TUGAS AKHIR

Judul : Implementasi *Deep Learning* Menggunakan
Convolutional Neural Network untuk Klasifikasi
Gambar (Studi Kasus: Klasifikasi Gambar Pada
Tanaman Anggrek Bulan Putih, Anggrek
Dendrobium, dan Anggrek Ekor Tupai)

Nama Mahasiswa : Rahayu Kia Sandi Cahaya Putri

Nomor Mahasiswa : 14611018

**TUGAS AKHIR INI TELAH DIPERIKSA DAN DISETUJUI UNTUK
DIUJIKAN**

Yogyakarta, 22 Maret 2018

Pembimbing,



(Dr. RB. Fajriya Hakim, S.Si., M.Si.)

HALAMAN PENGESAHAN

TUGAS AKHIR

Implementasi *Deep Learning* Menggunakan Metode *Convolutional Neural Network* Untuk Klasifikasi Gambar

(Studi Kasus: **Klasifikasi Gambar Pada Tanaman Anggrek** Bulan Putih, Anggrek Dendrobium, dan Anggrek Ekor Tupai)

Nama Mahasiswa : Rahayu Kia Sandi Cahaya Putri

Nomor Mahasiswa : 14611018

**TUGAS AKHIR INI TELAH DIUJIKAN
PADA TANGGAL 18 APRIL 2018**

Nama Penguji :

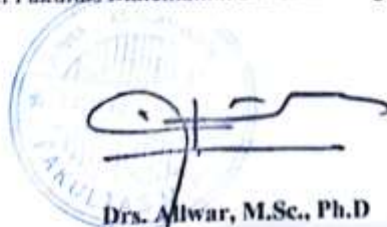
1. Andrie Pasca Hendradewa, S.T., M.T
2. Ayundyah Kesumawati, S.Si., M.Si
3. Dr. RB Fajriya Hakim, S.Si., M.Si

Tanda Tangan



Mengetahui,

Dekan Fakultas Matematika dan Ilmu Pengetahuan Alam



Drs. Alwar, M.Sc., Ph.D

KATA PENGANTAR



Assalamu'alaikum Wr. Wb

Alhamdulillahirabbil'aalamiin, puji syukur kehadiran Allah SWT yang telah melimpahkan rahmat, petunjuk, dan kemudahan sehingga atas ridho-Nya Tugas Akhir ini dapat terselesaikan. Shalawat serta salam tercurah kepada junjungan kita Nabi Muhammad SAW beserta keluarga dan para pengikut-pengikutnya sampai akhir zaman.

Tugas akhir yang berjudul “**Implementasi Deep Learning Menggunakan Metode Convolutional Neural Network Untuk Klasifikasi Gambar**” ini disusun sebagai hasil proses pembelajaran yang telah peneliti dapatkan selama melakukan proses pembelajaran di Jurusan Statistika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Islam Indonesia.

Selama proses menyusun Tugas Akhir ini, peneliti telah banyak mendapatkan bantuan dari berbagai pihak. Untuk itu, pada kesempatan ini peneliti bermaksud menyampaikan ucapan terimakasih kepada :

1. Drs. Allwar, M.Sc., Ph.D selaku dekan Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Islam Indonesia.
2. Dr. RB. Fajriya Hakim, S.Si., M.Si selaku ketua Jurusan Statistika FMIPA UII, yang sekaligus menjadi dosen pembimbing yang tidak hentinya membimbing dan memberi masukan sampai terselesaikannya tugas akhir ini.
3. Dosen-dosen Statistika UII yang telah mendedikasikan ilmunya kepada peneliti.
4. Kedua Orang Tua serta seluruh keluarga besar yang selalu memberikan dukungan, doa dan motivasi dalam penyusunan tugas akhir ini.
5. Sahabat-sahabat seperjuangan “Tim Deep Learning” dalam satu bimbingan Tugas Akhir

6. Teman-teman Statistika UII 2014, khususnya teman-teman kelas A terimakasih untuk kebersamaan, bantuan, dan pengalamannya selama kuliah di Statistika.
7. Teman-teman KKN unit 7 angkatan 55 Bang Erda, Luthfi, Kiki, Desi, Octa, Raras, Nanang, Gharby
8. Rizaldi Zakaria, terimakasih atas segala dukungan, doa dan bantuannya dalam menyelesaikan segala urusan selama menyelesaikan tugas akhir ini.
9. Semua pihak yang tidak dapat peneliti sebutkan satu persatu, terimakasih atas bantuannya.

Peneliti menyadari sepenuhnya bahwa tugas akhir ini masih jauh dari kata sempurna, untuk itu segala kritik dan saran yang sifatnya membangun selalu peneliti harapkan. Semoga tugas akhir ini dapat bermanfaat bagi peneliti khususnya dan bagi semua yang membutuhkan.

Akhir kata, semoga Allah SWT selalu melimpahkan rahmat serta hidayah-Nya kepada kita semua, Aaamiin yaa robbal ‘aalamiin.

Wassalamu’alaikum Wr. Wb.

Yogyakarta, 22 Maret 2018

Peneliti

DAFTAR ISI

HALAMAN PERSETUJUAN DOSEN PEMBIMBING Error! Bookmark not defined.

HALAMAN PENGESAHAN iv

KATA PENGANTAR..... v

DAFTAR ISI..... vii

DAFTAR TABEL ix

DAFTAR GAMBAR..... x

DAFTAR LAMPIRAN xi

PERNYATAAN BEBAS PLAGIARISME xii

BAB I PENDAHULUAN..... 1

1.1. Latar Belakang 1

1.2. Rumusan Masalah 2

1.3. Jenis Penelitian dan Metode Analisis 2

1.4. Tujuan Penelitian..... 3

1.5. Sistematika Penulisan..... 3

BAB II TINJAUAN PUSTAKA..... 5

BAB III LANDASAN TEORI..... 9

3.1. Citra Digital 9

3.2. Klasifikasi Citra..... 9

3.3. Bunga Anggrek..... 10

3.4. Pengertian Pengolahan Citra Digital (*Image Processing*)..... 12

3.5. Citra Warna (*RGB*) 12

3.6. *Machine Learning* 13

3.7.	<i>Deep Learning</i>	15
3.8.	<i>Convolutional Neural Network</i>	22
3.9.	Operasi Konvolusi	24
3.11.	Aktivasi <i>ReLU</i>	27
3.12.	<i>Fully-Connected Layer</i>	27
3.13.	<i>Dropout Regularization</i>	28
3.14.	<i>Softmax Classifier</i>	28
3.15.	<i>Crossentropy Loss Function</i>	29
3.16.	<i>Stochastic Gradient Descent</i>	29
3.17.	<i>Max Norm Constraint</i>	30
3.18.	<i>Confusion Matriks</i>	31
3.19.	<i>R-Studio</i>	32
3.20.	<i>Keras</i>	33
BAB IV METODE PENELITIAN		35
BAB V HASIL DAN PEMBAHASAN		40
5.1.	Pengumpulan Data Citra	40
5.2.	Histogram Citra	42
5.3.	<i>Preprocessing</i> Citra	42
5.4.	Pengolahan Citra	44
BAB VI KESIMPULAN DAN SARAN		51
6.1.	Kesimpulan.....	51
6.2.	Saran	51
DAFTAR PUSTAKA		53
LAMPIRAN		56

DAFTAR TABEL

Tabel 1 Tabel Pemilihan Metode	18
Tabel 2 Model <i>Confusion Matriks</i>	34
Tabel 3 Tabel Definisi Operasional Penelitian.....	37
Tabel 4 Hasil Klasifikasi pada Data <i>Training</i>	48
Tabel 5 Hasil Klasifikasi pada Data <i>Testing</i>	49
Tabel 6 Hasil Klasifikasi untuk Data <i>Training</i>	50
Tabel 7 Hasil Klasifikasi untuk Data <i>Testing</i>	51

DAFTAR GAMBAR

Gambar 3. 1 Bunga Anggrek Bulan Putih	12
Gambar 3. 2 Bunga Anggrek Dendrobium.....	12
Gambar 3. 3 Anggrek Ekor Tupai	13
Gambar 3. 4 Ruang Warna dari RGB.....	15
Gambar 3. 5 Perbandingan Pemrograman Tradisional pada <i>Machine Learning</i>	16
Gambar 3. 6 <i>Layer-layer</i> pada <i>Deep Learning</i>	19
Gambar 3. 7 <i>Perceptron</i> dengan d buah input.....	19
Gambar 3. 8 Contoh Jaringan CNN.....	23
Gambar 3. 9 Operasi pada Konvolusi.....	26
Gambar 3. 10 Operasi pada <i>Max Pooling</i>	28
Gambar 3. 11 Jaringan Syaraf pada Pengaplikasian <i>Dropout</i>	30
Gambar 4. 1 Alur Penelitian	39
Gambar 5. 1 <i>Input</i> Citra.....	43
Gambar 5. 2 Matrik Piksel pada Citra	43
Gambar 5. 3 Histogram Citra	44
Gambar 5. 4 Pengubahan Ukuran Piksel Citra pada Data <i>Training</i>	45
Gambar 5. 5 Pengubahan Ukuran Piksel pada Citra Data <i>Testing</i>	45
Gambar 5. 6 Kombinasi Citra untuk Data <i>Training</i>	46
Gambar 5. 7 Kombinasi Citra untuk Data <i>Testing</i>	46
Gambar 5. 8 Pelabelan untuk Data <i>Training</i> dan Data <i>Testing</i>	47
Gambar 5. 9 Model <i>Convolutional Neural Network</i>	47
Gambar 5. 10 Fit Model	48

DAFTAR LAMPIRAN

Lampiran 1 Obyek Citra Tanaman Bunga Anggrek 32 piksel x 32 piksel	56
Lampiran 2 <i>Syntax</i> untuk Analisis CNN	57
Lampiran 3 Hasil Proses Akurasi dengan <i>Epochs</i> 500	65
Lampiran 4 Hasil Evaluasi Training dan Prediksi <i>Test Data</i>	90
Lampiran 5 Hasil Evaluasi <i>Training Detail</i>	91
Lampiran 6 Hasil Prediksi <i>Test Data</i>	96

PERNYATAAN BEBAS PLAGIARISME

Dengan ini, peneliti menyatakan bahwa dalam Tugas Akhir ini tidak terdapat karya yang sebelumnya pernah diajukan untuk memperoleh gelar kesarjanaan di suatu perguruan tinggi dan sepanjang pengetahuan peneliti juga tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan orang lain, kecuali yang di acu dalam naskah ini dan disebutkan dalam daftar pustaka.

Yogyakarta, 22 Maret 2018



Peneliti

INTISARI

Indonesia kaya akan keanekaragaman hayati, kesuburan tanah dan iklim tropis yang membuat Indonesia memiliki banyak jenis dan macam tanaman yang tumbuh, termasuk tanaman hias. Tanaman hias contohnya anggrek banyak digemari penghobi tanaman hias untuk dibudidayakan dan dilestarikan. Masih banyak masyarakat yang sulit untuk melakukan klasifikasi atau mengetahui macam jenis tanaman hias terutama anggrek. Hal ini, menjadi masalah seiring ketidaktahuan jika terus dilanjutkan maka tidak akan ada lagi yang melestarikan. Solusinya dapat dengan teknologi terbaru yaitu dengan deep learning metode convolutional neural network untuk klasifikasi gambar (image recognition) dengan melakukan training menggunakan metode feedforward dan backpropagation. Terakhir yaitu, tahap klasifikasi dengan metode feedforward dengan bobot data dan bias yang diperbarui. Hasil uji coba dari klasifikasi objek citra pada citra tanaman anggrek bulan putih, dendrobium dan ekor tupai mampu mencapai nilai akurasi 83%, recall 80% dan presisi 89%. Artinya, klasifikasi sudah terbentuk dengan baik. Untuk hasil perbandingan dengan jumlah objek data training yang berbeda jumlahnya tetap mempunyai akurasi yang sama yaitu 83%.

Kata Kunci : *Deep Learning, Convolutional Neural Network, Klasifikasi Tanaman.*

ABSTRACT

Indonesia is rich in biodiversity, soil fertility and a tropical climate that makes Indonesia has many types and kinds of plant growing, including ornamental plants. For example many ornamental plants Orchid hobbyists ornamental plants popular for cultivated and preserved. There are still many communities that are hard to do or know the classification of types of ornamental plant especially orchids. This, to be a problem as ignorance if continue then there will be no longer the preserve. The solution may be with the latest technology with the deep learning method of convolutional neural network to classification of images (image recognition) by conducting training using the method of feedforward and backpropagation. The last stage, namely classification by method of feedforward with weights data and bias are updated. Trial results of classification of image objects on the image of the white moon Orchid plant, dendrobium and squirrels are able to reach a value of 83% accuracy, precision and recall 80% 89%. This means that the classification is already well formed. For the results of the comparison with the amount of training data objects of different fixed amount has the same accuracy that is 83%.

Keywords : *Deep Learning, Convolutional Neural Network, Plants Classification.*

BAB I

PENDAHULUAN

1.1. Latar Belakang

Alam Indonesia yang kaya akan keanekaragaman hayati, tanah yang subur, dan iklim tropis membuat banyak jenis dan macam tumbuhan dapat tumbuh di negara ini. Berdasarkan data FAO pada tahun 2010 mengenai hutan dunia yang termasuk di dalamnya hutan Indonesia, secara keseluruhan menyimpan 289 gigaton karbon dan memegang peranan penting dalam menjaga kestabilan iklim di dunia.

Luas dan didukung oleh keanekaragaman hayati menjadikan Indonesia banyak memiliki jenis tumbuhan yang hanya dimiliki di Indonesia. Banyak jenis bunga yang berasal dari hutan Indonesia yang salah satunya Bunga Anggrek. Jenis bunga Anggrek di dunia ini ada berkisar 5000-6000 jenis ada di tanah di Indonesia, dimana 29 jenis di antaranya termasuk dalam kategori langka. Maka dari keistimewaan itu, bunga anggrek termasuk tanaman yang dilindungi dalam Peraturan Pemerintah Nomor 7 Tahun 1999 tentang pengawetan tumbuhan dan satwa.

Banyak masyarakat yang masih belum mengetahui apa saja jenis dan macam tanaman bunga yang ada di Nusantara ini, dan dikarenakan terdapat beberapa macam jenis bunga yang hanya mampu tumbuh atau hidup di beberapa bagian wilayah saja. Dengan persoalan tersebut, diharapkan para petani atau pekebun melakukan penanaman dalam skala besar sebagai langkah untuk melestarikan keanekaragaman hayati ini. Penanaman skala besar ini nanti juga akan menjadi permasalahan baru jika tidak ditambah dengan teknologi untuk membantu dalam penanaman dalam skala besar tersebut. Maka dari itu, harus diimbangi dengan suatu teknologi menggunakan teknik *deep learning* dengan metode *Convolutional Neural Network* yang mampu digunakan sebagai solusi untuk mengklasifikasikan macam-macam jenis tanaman bunga sebagai langkah untuk pendeteksian tanaman bunga.

Sebagai percobaan pengklasifikasian bunga anggrek dengan 3 bunga anggrek yang ada pada 3 kategori bunga anggrek yang banyak ditanam atau dibudidayakan oleh petani dan ditanam di tanah Indonesia. Tanaman anggrek tersebut yaitu yaitu jenis anggrek bulan putih, anggrek dendrobium, dan anggrek ekor tupai. Dengan melakukan klasifikasi dan pendeteksian objek gambar ini menggunakan CNN sehingga dapat mengetahui kategori jenis dan macam tanaman bunga, yang gunanya agar orang-orang pada umumnya yang tidak mengetahui jenis dan bunga tersebut dapat mengetahui jenis dan tanaman bunga apa yang ada pada gambar sebagai pengetahuan baru. Selain dapat mengetahui jenis dan tanaman bunga apa pada gambar tersebut, masyarakat juga mampu mengetahui bagaimana ciri, detil dari jenis bunga tersebut, pupuk apa yang digunakan pada tanaman bunga tersebut, dan bagaimana karakteristik yang dimiliki jika dikembangkan menjadi aplikasi yang bermanfaat bagi ilmu pengetahuan ke depannya.

1.2. Rumusan Masalah

Berdasarkan latar belakang masalah di atas, maka dapat dirumuskan masalah sebagai berikut :

1. Bagaimana hasil pendeteksian objek citra konsep *Deep Learning* dengan menggunakan CNN untuk permasalahan klasifikasi citra gambar objek yang digunakan; anggrek bulan putih, anggrek dendrobium, dan anggrek ekor tupai?
2. Seberapa tinggi tingkat keberhasilan atau akurasi dari hasil klasifikasi yang dihasilkan dengan metode CNN?
3. Apakah terdapat perubahan hasil akurasi yang didapatkan untuk data *testing* jika dilakukan perbaikan kualitas gambar dengan mengganti data *training* dari 140 data citra menjadi 90 data citra?

1.3. Jenis Penelitian dan Metode Analisis

Jenis penelitian ini merupakan penelitian aplikatif menggunakan *software R* dengan menggunakan konsep *deep learning* dan metode *Convolutional Neural Network* yang bertujuan agar kompiuter mampu mengklasifikasikan objek citra

tanaman bunga anggrek bulan putih, anggrek dendrobium, dan anggrek ekor tupai dengan tingkat akurasi yang juga tinggi.

1.4. Tujuan Penelitian

Tujuan yang akan dicapai dalam penelitian ini antara lain :

1. Mengimplementasikan *Deep Learning* dengan menggunakan CNN (*Convolutional Neural Network*) untuk mengklasifikasi citra gambar bunga anggrek dengan 3 kategori jenis; anggrek bulan putih, anggrek dendrobium, dan anggrek ekor tupai.
2. Mengetahui tingkat keberhasilan metode CNN dalam melakukan klasifikasi objek citra bunga anggrek dengan hasil tingkat akurasi yang didapatkan.
3. Mengetahui pengaruh jika dilakukan pengurangan terhadap jumlah data objek citra pada data *training*.

1.5. Sistematika Penulisan

Sistematika penulisan tugas akhir ini secara garis besar dapat diuraikan sebagai berikut :

BAB I : PENDAHULUAN

Bab ini menjelaskan mengenai latar belakang masalah dalam penelitian ini, perumusan masalah yang digunakan dalam penelitian, batasan masalah, tujuan penelitian, dan sistematika penulisan dalam penyusunan tugas akhir ini.

BAB II : TINJAUAN PUSTAKA

Bab 2 ini berisikan tinjauan atas penelitian peneliti-peneliti terdahulu, dan untuk membandingkan dan menimbang seberapa benar dan penting penelitian yang akan dilakukan.

BAB III : LANDASAN TEORI

Bab 3 ini akan dijelaskan mengenai teori-teori yang berhubungan dengan penelitian yang akan digunakan sebagai dasar untuk menyelesaikan permasalahan yang diangkat pada penelitian ini,

yaitu konsep *Machine Learning*, Algoritma *Deep Learning*, dan dataset yang digunakan dalam penelitian.

BAB IV : METODOLOGI PENELITIAN

Bab 4 ini menjelaskan tentang alat dan bahan yang digunakan dalam penelitian termasuk pengambilan dataset/objek yang digunakan dalam analisis, kemudian juga mengenai langkah-langkah dalam menganalisis penelitian.

BAB V : HASIL DAN PEMBAHASAN

Bab 5 ini dijelaskan mengenai hasil dari analisis yang dilakukan dan pembahasan dari hasil yang didapatkan yaitu berupa hasil model jaringan dari dataset yang sudah dilatih dengan program. Hasil tersebut berupa waktu yang diperlukan untuk pelatihan dan akurasi yang didapat dari pelatihan serta validasi model dari jaringan data yang berada diluar dataset.

BAB VI : KESIMPULAN DAN SARAN

Bab 6 ini berisikan mengenai kesimpulan dari hasil penelitian yang telah didapatkan dari hasil keseluruhan yang telah dilakukan, serta saran untuk peneliti selanjutnya yang akan mengembangkan penelitian ini.

BAB II

TINJAUAN PUSTAKA

Banyak pengembangan sistem yang mendalami *Computer Vision* seperti *Face Recognition*, *Image Recognition* maupun pengenalan pola tertentu. Sistem seperti ini, terbukti secara kongkrit menjadi sebuah cara yang dapat mempermudah pekerjaan di banyak bidang. Implementasi *Deep Learning* pada permasalahan tersebut sangatlah tepat dan efektif. Penggunaan teknik *Deep Learning* dengan metode *Convolutional Neural Network* pertama kali berhasil diaplikasikan oleh Yann LeCun pada tahun 1998 (LeCun, Bottou, Bengio, & Haffner, 1998). Pada penelitian ini, LeCun mengemukakan bahwa metode CNN untuk mengenal tulisan tangan untuk keperluan pembacaan pada suatu dokumen. Hasil yang didapat dari penelitian tersebut menunjukkan nilai akurasi yang cukup tinggi hingga mencapai *test error* sebesar 1,7%.

Penerapan metode *Convolutional Neural Network* dapat dikembangkan dari sisi arsitektur dan banyaknya lapisan yang digunakan pada jaringan. Penggunaan arsitektur yang benar akan sangat baik untuk klasifikasi citra dalam berbagai macam kategori. Contohnya seperti pada dataset *ImageNet* yang memiliki sebanyak lebih dari 1000 kategori. Pada tahun 2012, teknik *Deep Learning* dengan metode CNN dipopulerkan dengan arsitektur AlexNet yang diuji dengan dataset *ImageNet* (Krizhevsky, Sutskever, & Hinton, 2012). Arsitektur yang dibuat oleh Alex Krizhevsky menunjukkan hasil yang sangat signifikan pada *testing set* dengan *test error* sebesar 17%. Hasil tersebut dinilai sudah sangat luar biasa karena citra pada dataset yang digunakan sangatlah kompleks dan banyak.

Kedalaman jaringan merupakan komponen penting untuk kinerja yang baik untuk pengenalan citra. Lebih dalam sebuah arsitektur jaringan maka akan lebih banyak lapisan yang digunakan. Jaringan yang dibuat pada penelitian Karen Simonyan dan Andrew Zisserman menggunakan lapisan konvolusi sebanyak 16 hingga 19 lapisan dan jaringan tersebut menunjukkan performa sangat baik

dengan akurasi yang lebih besar dibanding dengan arsitektur yang sudah dikemukakan sebelumnya (Simonyan & Zisserman, 2015).

Penelitian mengenai klasifikasi menggunakan *deep learning* dengan judul “Penggunaan *Deep Learning* untuk Prediksi *Churn* pada Jaringan Telekomunikasi *Mobile*” yaitu prediksi *churn* mengenai turun naiknya jumlah pelanggan pada jaringan telekomunikasi selular, yang dipecahkan dengan menggunakan arsitektur jaringan *Multilayer Perceptron*, dan implementasi sistem dengan *Autoencoder*. Dari perhitungan menggunakan sistem dengan *F-Measure* didapatkan nilai akurasi 81,35% untuk data *training*, dan 83,12% untuk data *testing*. (Abdillah, 2016).

Algoritma pembelajaran yang diimplementasikan dalam penelitian ini yaitu menggunakan salah satu arsitektur *deep learning* yaitu *Convolutional Neural Network* (CNN) yang digunakan untuk memecahkan masalah dalam melakukan klasifikasi data karena mempunyai tingkat akurasi yang tinggi. Penerapan dengan menggunakan metode ini pernah diterapkan oleh I Wayan Suartika E. P., dkk (2016) dalam penelitiannya “Klasifikasi Citra Menggunakan *Convolutional Neural Network* (CNN) pada Caltech 101” menerapkan salah satu metode *deep learning* yaitu dengan menggunakan CNN untuk mengklasifikasi citra objek gambar. Dalam metode penelitian yang digunakan dalam algoritma CNN terdapat 3 lapisan yang diimplementasikan, yaitu *convolutional layer*, *subsampling layer*, dan *fully connected layer*. Metode dalam penelitiannya terlebih dahulu dilakukan proses pengolahan pada data citra untuk memfokuskan objek yang akan diklasifikasi dengan metode *wrapping* dan *cropping*. Selanjutnya, melakukan proses *training* dan *testing* dengan menggunakan algoritma CNN. Hasil uji coba dari klasifikasi citra objek tingkat *confusion* yang berbeda pada basis data Caltech 101 menghasilkan nilai akurasi sebesar 20% hingga 50%.

Mulyana (2014) dalam penelitiannya berjudul “Identifikasi Jenis Bunga Anggrek Menggunakan Pengolahan Citra Digital dengan Metode KNN” meneliti bunga anggrek dengan pengidentifikasian melalui ekstraksi dengan metode

Principal Component Analysis (PCA), dan warna diekstrak dengan perhitungan rata-rata RGB (*Red-Green-Blue*) kemudian diklasifikasikan dengan KNN dan menghasilkan nilai akurasi 71,25%. Akurasi dapat menurun saat citra dipengaruhi kondisi akuisisi yang berbeda dari data latih sistem. Data latih dari ekstraksi metode PCA tidak mewakili citra bunga anggrek dengan mempengaruhi jarak, sudut, ketinggian, dan rotasi akuisisi citra. Sehingga, hasil klasifikasi KNN menjadi kurang tepat.

Yuzhen Lu (2016) dalam penelitiannya “*Food Image Recognition by Using Convolutional Neural Network*” menggunakan dataset 5822 gambar dengan 10 kategori dan 5 lapisan CNN yang digunakan pada klasifikasi gambar. Model pertama yang digunakan menggunakan SVM “*The-bag-of-feature*” (BoF) dengan tingkat akurasi 56%, ketika menggunakan CNN lebih baik akurasinya menjadi 74%. Kemudian, dilakukan data latih sehingga menjadi signifikan lebih dari 90% akurasinya. Parameter yang banyak akan meningkatkan akurasinya.

Rismiyati (2016) dalam penelitiannya “*Implementasi Convolutional Neural Network untuk Sortasi Mutu Salak Eskpor Berbasis Citra Digital*” menggunakan metode *deep learning* dengan CNN untuk melakukan proses ekstraksi fitur citra salak dan klasifikasi yang mampu membedakan salak yang lolos ekspor dan tidak, dan juga dilakukan dengan menggunakan data latih yang beragam untuk melihat pengaruh terhadap akurasi. Pengujian yang dilakukan dengan metode *stratified cross validation* untuk mengukur akurasi berdasarkan *confusion matrix*. Hasil penelitiannya yaitu bahwa akurasi terbaik terletak pada *learning rate* 0,0001, satu lapisan konvolusi dengan 15 filter dengan ukuran 3 x 3 x 3, dan jumlah neuron tersembunyi 100. Sehingga akurasi yang didapatkan yaitu 81,5%.

BAB III

LANDASAN TEORI

3.1. Citra Digital

Citra *digital* adalah citra elektronik yang diambil dari beberapa jenis dokumen, yaitu berupa foto, buku, ataupun video dan suara. Proses yang digunakan untuk merubah citra analog menjadi suatu citra digital disebut sebagai proses digitasi. Digitasi adalah proses dimana mengubah suatu gambar, teks, atau suara yang berasal dari benda dapat dilihat ke dalam data elektronik dan dapat disimpan serta diproses untuk keperluan yang lainnya.

Citra *digital* merupakan sebuah representasi numerik (mayoritas biner) dari gambar 2 dimensi. Sebuah gambar dapat didefinisikan sebagai fungsi 2 dimensi $f(x,y)$ di mana x dan y merupakan titik koordinat bidang datar, dan harga dari fungsi f dari setiap pasangan titik koordinat (x,y) yang disebut dengan intensitas atau level keabuan (*grey level*) dari suatu gambar. Ketika nilai titik x,y dan nilai intensitas f terbatas dengan nilai diskrit, maka gambar tersebut akan dapat dikatakan sebagai sebuah citra digital (Gonzales, Rafael, & Woods, 2002).

3.2. Klasifikasi Citra

Klasifikasi citra adalah sebuah pekerjaan untuk memasukkan citra dan menempatkan ke dalam suatu kategori. Ini merupakan salah satu dari permasalahan yang ada pada *Computer Vision* yang dapat disederhanakan dan memiliki berbagai macam aplikasinya. Salah satu aplikasi dalam klasifikasi citra adalah pengklasifikasian nama tempat pada suatu citra.

Setiap citra yang di *input* pada *training set data* diberikan label atau penamaan. Saat klasifikasi, label atau penamaan tersebut akan menjadi perbandingan dengan hasil hipotesis yang diberikan oleh model pembelajaran dan akan menghasilkan nilai *error*. Klasifikasi yang terawasi ini bisa sangat efektif dan akurat dalam mengklasifikasikan citra tempat maupun objek lainnya. Banyak

metode dan algoritma yang dapat mendukung proses klasifikasi yang terawasi terutama dengan teknik *Deep Learning*.

3.3. Bunga Anggrek

Bunga jenis anggrek (latin: *Orchidaceae*) adalah satu jenis tumbuhan berbunga dengan anggota jenis terbanyak. Jenis-jenisnya tersebar luas dari daerah tropika basah hingga wilayah sirkumpolar, walaupun sebagian besar anggotanya ditemukan di daerah tropika. Anggrek di daerah beriklim sedang biasanya hidup di tanah dan membentuk umbi sebagai cara beradaptasi dengan musim dingin. Organ-organnya yang cenderung tebal dan berdaging “sukulen” membuatnya tahan menghadapi tekanan ketersediaan air. Anggrek epifit dapat hidup dari embun dan udara lembab.

a. Anggrek Bulan Putih

Anggrek bulan (*phalaenopsis amabilis*) atau puspa pesona adalah salah satu bunga nasional Indonesia yang ditemukan pertama kali oleh seorang ahli botani Belanda, Dr. C. L. Blume. Anggrek bulan termasuk ke dalam tanaman anggrek monopodial yang menyukai sedikit cahaya matahari sebagai penopang hidupnya. Daunnya berwarna hijau dengan bentuk memanjang. Akar-akarnya berwarna putih dan berbentuk bulat memanjang. Akar-akarnya berwarna putih dan berbentuk bulat memanjang serta terasa berdaging. Bunganya memiliki sedikit keharuman dan waktu mekar yang lama serta dapat tumbuh hingga diameter 10 cm lebih (Wikipedia, 2018).



Gambar 3. 1 Bunga Anggrek Bulan Putih

b. Anggrek Dendrobium

Anggrek dendrobium merupakan jenis anggrek epifit yang dapat digunakan sebagai tanaman hias dalam ruangan ataupun untuk taman. Dendrobium memiliki banyak warna, dan bentuk serta aroma yang khas. Bentuk bunga anggrek dendrobium memiliki sepal yang bentuknya hampir menyerupai segitiga, dasarnya bersatu dengan kaki kolom untuk membentuk taji. Petal biasanya lebih tipis dari sepal dan bibirnya berbelah (Wikipedia, 2018).



Gambar 3. 2 Bunga Anggrek Dendrobium

c. Anggrek Ekor Tupai

Anggrek ekor tupai (*Rhynchosylis violacea*) memiliki panjang keseluruhan bunganya dapat mencapai 20-30 cm dengan ukuran bunga sekitar 2cm. Warna bunga biasanya didominasi warna putih dan merah jambu keunguan. Rumpunan bunga dapat mencapai puluhan bunga dalam satu rumpun (Suzanna, 2015).



Gambar 3. 3 Anggrek Ekor Tupai

3.4. Pengertian Pengolahan Citra Digital (*Image Processing*)

Sebuah gambar disebut dengan citra digital bila gambar yang diperoleh dari hasil proses pada kamera, komputer, mesin *scan*, atau perangkat elektronik lainnya. Pengolahan pada citra digital diproses oleh komputer dengan menggunakan fungsi algoritma.

Pada \pengolahan citra ini, tujuannya agar citra yang mengalami gangguan lebih mudah direpresentasikan (baik oleh manumur maupun mesin) dengan melakukan manipulasi menjadi citra yang lain yang kualitasnya lebih baik. Pada umumnya, operasi-operasi pada pengolahan citra diterapkan pada citra bila (Jain A. , 1989)

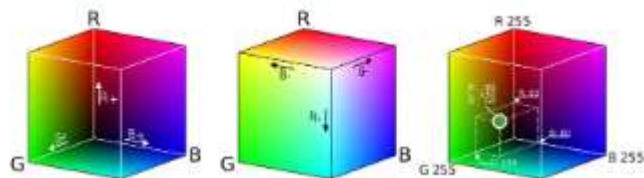
- a. Perbaikan atau memodifikasi citra perlu dilakukan untuk meningkatkan kualitas penampakan atau menonjolkan beberapa aspek informasi yang terkandung di dalam citra,
- b. Elemen pada citra perlu dikelompokkan, dicocokkan, dan diukur,
- c. Sebagian citra perlu digabungkan dengan bagian citra yang lain.

Agar dapat diolah dengan mesin komputer digital, maka suatu citra harus direpresentasikan secara numerik dengan nilai-nilai diskrit. Representasi citra dan fungsi kontinu menjadi nilai-nilai diskrit disebut digitalisasi. Citra yang dihasilkan inilah yang disebut Citra Digital. Pada umumnya citra digital berbentuk empat persegi panjang, dan dimensi ukurannya dinyatakan sebagai tinggi x lebar (Munir, 2004).

3.5. Citra Warna (RGB)

Citra warna atau biasa dikenal dengan citra RGB adalah suatu model warna aditif yang terdiri dari tiga buah warna dasar yaitu warna merah, warna hijau, dan warna biru. Sedangkan, untuk warna lain dibentuk dari hasil kombinasi ketiga warna dasar (merah, hijau, dan biru). Model warna RGB yaitu warna dasar yang dapat dibedakan dengan sel kerucut dari mata manusia berdasarkan teori trikomatik.

Citra warna adalah hasil dari tumpukan tiga matriks dengan masing-masing matriks merepresentasikan nilai masing-masing dari ketiga warna dasar (merah, hijau, dan biru) di setiap piksel, hingga setiap piksel saling berkaitan dengan tiga nilai (Sianipar, Mangiri, & Wiryajati, 2013). Setiap piksel pada warna citra mewakili warna kombinasi dari ketiga warna dasar dengan melakukan pengubahan nilai HSV (*Hue, Saturation, Values*) supaya dapat menghasilkan bermacam-macam warna. Jika setiap komponen warna menggunakan 8 bit (nilainya berkisar antara 0 sampai dengan 255). Maka, kemungkinan kombinasi warna yang dapat ditampilkan yaitu $2^8 \times 2^8 \times 2^8 = 2^{24}$ atau dengan kata lain terdapat lebih dari 16 juta warna yang dihasilkan. Untuk sebuah warna dapat direpresentasikan sebagai vektor tiga dimensi. Misalkan suatu vektor mempunyai sebuah warna yang ditulis sebagai $r = (x, y, z)$, dan kemudian komponen x , y , dan z tersebut digantikan menjadi merah, hijau, dan biru. Contoh pada **gambar 3.4** yaitu sebuah warna ditulis sebagai RGB (83, 150, 60) di mana warna tersebut memiliki nilai $R = 83$, $G = 150$, dan $B = 60$. Sehingga, warna tersebut dapat dilihat dengan mata lebih cenderung ke warna hijau. Sedangkan, warna hitam ditulis sebagai RGB (0, 0, 0) dan putih ditulis sebagai RGB (255, 255, 255).



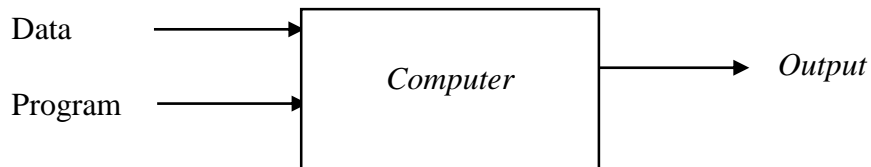
Gambar 3. 4 Ruang Warna dari RGB

3.6. *Machine Learning*

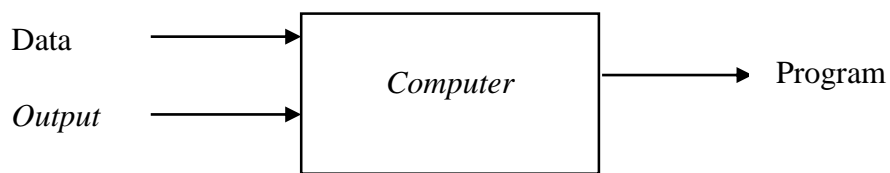
Machine Learning merupakan rangkaian teknik yang mampu membantu untuk menangani dan memprediksi data yang sangat besar dengan cara mempresentasikan data besar tersebut menggunakan algoritma pembelajaran. *Machine Learning* dapat membuat komputer memprogram diri mereka sendiri. Pada dasarnya, *Machine Learning* membiarkan data untuk melakukan pekerjaannya sendiri. Berikut merupakan gambaran umum untuk *Machine*

Learning dibandingkan dengan pemrograman yang lainnya (tradisional) tanpa metode ini (Danukusumo, 2017)

Traditional Programming



Machine Learning



Gambar 3. 5 Perbandingan Pemrograman Tradisional dengan *Machine Learning*

Dari **gambar 3.5** dapat diketahui bahwa pemrograman dengan cara tradisional data dan program komputer untuk menghasilkan suatu *output*. Sedangkan, dengan menggunakan teknik *Machine Learning* data dan *output* dijalankan dengan komputer untuk membuat sebuah program.

Banyak algoritma menggunakan *Machine Learning* yang dikembangkan pada setiap tahunnya. Setiap algoritma pembelajaran mesin memiliki tiga komponen yang penting, yaitu :

a. Representasi

Bagaimana mempresentasikan pengetahuan. Contohnya : *Decision tree*, *Neural Network*, *Support Vector Machine* dan lain-lain.

b. Evaluasi

Cara mengevaluasi prediksi dan hipotesis. Contohnya: *Mean Squared Error* (MSE), *Cost Function* dan lain-lain.

c. Optimasi

Cara program dari model dihasilkan dan proses pencarian parameter terbaik. Misalnya *Convex Optimization* dan *Gradient Descent*.

Terdapat 5 langkah dasar yang digunakan untuk melakukan tugas *Machine Learning* dan tugas ini sangat penting dalam mempersiapkan solusi untuk segala bentuk permasalahan dalam *machine learning* maupun *deep learning*:

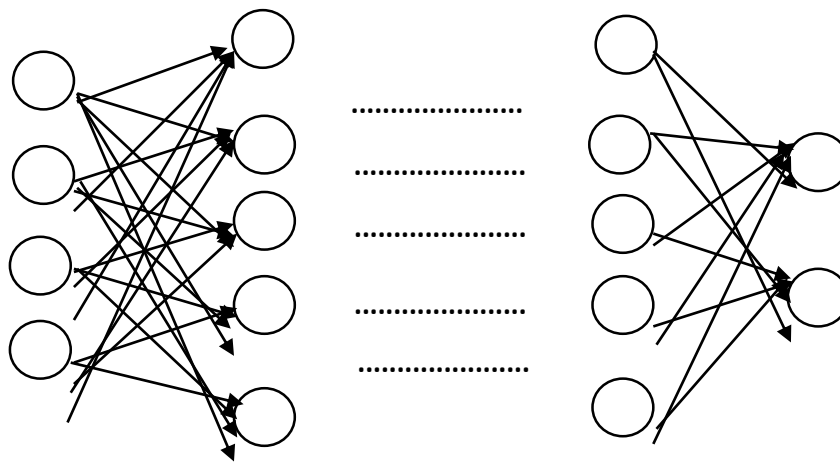
1. Mengumpulkan data : terbentuk dari simpanan pada beberapa file yang berisikan data yang dapat diperelajari komputer, dan setelah itu dipisah antara fitur masukan (*input*) dan keluaran (*output*).
2. Mempersiapkan data : penentuan kualitas data pada *preprocessing data* sehingga hasil yang didapat juga optimal/baik.
3. Melatih sebuah model : langkah ini dilakukan dengan pemilihan algoritma dan representasi data yang tepat dalam bentuk model.
4. Mengevaluasi model : menguji keakuratan hasil berdasarkan bagian *test dataset*.
5. Meningkatkan kinerja : langkah ini melibatkan pemilihan *hyperparameter* untuk meningkatkan efisiensi dan biasanya menggunakan *cross-validation*.

3.7. Deep Learning

Deep Learning merupakan cabang ilmu dari *Machine Learning* yang berbasis Jaringan Syaraf Tiruan (JST) atau dapat dikatakan perkembangan dari JST yang mengajarkan komputer untuk dapat melakukan tindakan yang dianggap alami oleh manusia. Misalnya yaitu belajar dari contoh. Dalam *Deep Learning*, sebuah komputer dapat belajar mengklasifikasi secara langsung dari gambar, suara, teks, atau video sekalipun. Sebuah komputer seperti dilatih dengan menggunakan data set berlabel dan jumlahnya sangat besar yang kemudian dapat mengubah nilai piksel dari sebuah gambar menjadi suatu representasi internal atau *feature vector* yang dimana pengklasifikasiannya dapat digunakan untuk mendeteksi atau mengklasifikasi pola pada masukan *input* (LeCun, Bengio, & Hinton, 2015).

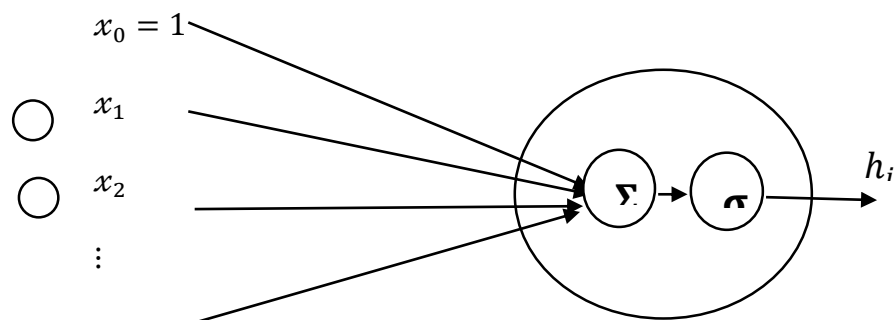
Metode *deep learning* adalah metode pembelahan dengan beberapa tingkat representasi, dimana representasi dapat membentuk arsitektur jaringan syaraf yang mempunyai banyak layer (lapisan). Lapisan pada *deep learning*

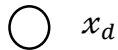
terbagi menjadi tiga bagian yaitu, *input layer*, *hidden layer*, dan *output layer*. Pada *hidden layer* dapat dibuat banyak lapis atau berlapis-lapis untuk menemukan komposisi algoritma yang tepat agar dapat meminimalisir *error* pada *output*. Semakin banyak *layer* yang dihasilkan, maka akan semakin kecil *error* yang di *Input Layer* dan *Hidden Layer 1* dan *Hidden Layer 2* *Output Layer 2* tinggi.



Gambar 3. 6 Layer-layer pada *Deep Learning*

Pada **gambar 3.6** menggambarkan hasil layer-layer pada *deep learning* yang memiliki $p + 2$ layer (p *hidden layer*, 1 *input layer*, dan 1 *output layer*). Lingkaran berwarna hitam menggambarkan *neuron*. Setiap lapisan *hidden layer* mempunyai satu atau lebih neuron. Neuron-neuron ini yang akan terhubung langsung dengan neuron lain pada *layer* berikutnya. Koneksi atau hubungan antar neuron hanya akan terjadi di antara 2 buah *layer* (*input* dan *output*) tidak ada koneksi atau hubungan pada *layer* yang sama walaupun jika secara teknis bisa saja dibuat dan juga *fully connected*.





Gambar 3.7 *Perceptron* dengan d buah *input*

Pada sebuah sistem yang terdiri dari sebuah neuron beserta *input* dan *output* nya disebut sebagai *perceptron*. Seperti pada **gambar 3.7** sebuah *neuron* diperbesar disimbolkan sebagai h_j yang menerima d buah *input* $x_1, \dots, x_d \in R$ yang berasal dari data apapun *output* lapisan sebelumnya. Sedangkan, x_0 tidak dianggap sebagai *input* (sebagai *dummy*) dan selalu bernilai 1. Variabel-variabel $w_{1j}, \dots, w_{dj} \in R$ merupakan bobot/*weights* dari koneksi *input* ke *neuron* h_j dan w_{0j} dinamakan sebagai bias.

Bobot adalah koneksi antar lapisan yang berupa nilai yang menentukan fungsi *input-output* dari mesin. Bobot adalah parameter penyesuaian yang diatur oleh mesin untuk mengukur kesalahan antara nilai *output* dan pola nilai yang diinginkan pada pembelajaran. Sehingga, nilai bobot inilah yang diatur mesin untuk mengurangi kesalahan yang mungkin terjadi. Dalam sistem *deep learning*, kemungkinan ada ratusan juta bobot yang dapat diatur. Untuk menyesuaikan vektor bobot dengan benar, algoritma menghitung nilai gradien vektor untuk setiap bobot berdasarkan jumlah kesalahan yang meningkat atau menurun jika bobot meningkat dalam jumlah kecil (LeCun, Bengio, & Hinton, 2015).

Operasi untuk sebuah *perceptron* yaitu merupakan dua buah operasi terpisah yaitu operasi *linear* \sum (2.1) dan operasi non-linear/aktivasi σ (2.2). Operasi linear dengan nilai bobot yang ada, kemudian hasil komputasi dari operasi linear akan ditransformasi menggunakan operasi non-linear yang kemudian disebut sebagai fungsi aktivasi.

$$z_j = \sum_{i=0}^d w_{ij} x_i \quad \dots\dots\dots (2.1)$$

$$h_j = \sigma(z_j) \quad \dots\dots\dots (2.2)$$

Pada operasi aktivasi non-linear $\sigma : R \rightarrow R$ terdapat berbagai macam fungsi aktivasi yang dapat diimplementasikan pada *hidden neuron*. Didalam penelitian ini digunakan salah satu fungsi aktivasi yaitu *softmax layer*. *Softmax*

layer digunakan apabila pada permasalahan *multiclass classification*, *output layer* biasanya memiliki lebih dari satu neuron. Misalnya, $a = [a_1, \dots, a_m]^T$ merupakan sebuah vektor dengan m buah elemen, *softmax* dapat didefinisikan sebagai berikut :

$$\sigma(a_j) = \frac{\exp(a_j)}{\sum_{k=1}^m \exp(a_k)} \dots\dots\dots (2.3)$$

Dapat dilakukan pengecekan untuk $\sum_{j=1}^m \sigma(a_j) = 1$ untuk *softmax*.

Deep learning memungkinkan komputasi model yang terdiri dari beberapa *processing layer* untuk mempelajari representasi data dengan macam-macam tingkat abstraksi. Metode ini telah memperbaiki *state-of-the-art* dalam pengenalan suara (*speech recognition*), pengenalan objek visual (*visual object recognition*), deteksi objek (*object detection*) dan banyak penemuan lainnya seperti penemuan obat dan genomik. *Deep learning* menemukan struktur yang rumit dan sulit dalam kumpulan data yang sangat besar dengan menggunakan algoritma *backpropagation* untuk menunjukkan bagaimana sebuah mesin harus mengubah parameter internal yang digunakan untuk menghitung representasi pada setiap lapisan dari representasi pada lapisan sebelumnya (LeCun, Bengio, & Hinton, 2015).

Beberapa algoritma yang menerapkan konsep *deep learning* antara lain *Deep Convolutional Neural Network* (DCNN) untuk melakukan klasifikasi gambar. *Deep Belief Network* – *Deep Neural Network* (DBN-DNN) untuk pengenalan suara, *Recurrent Neural Network* (RNN) untuk menerjemahkan bahasa. *Query-Oriented* untuk meringkas multi dokumen, *Conditional Restricted Boltzman Machine* (RBM) untuk melakukan prediksi *Drug-Target Interaction* (DTI), dan *Deep Belief Network* (DBN) untuk prediksi data sesuai waktu.

Tabel 1 Tabel Pemilihan Metode

Metode		Kelebihan		Kekurangan	
K-Nearest	Neighbour	KNN	mempunyai	- KNN	perlu

(KNN)	<p>beberapa kelebihan bahwa metode ini tangguh untuk melakukan training data yang <i>noisy</i> dan efektif apabila data latihnya besar.</p>	<p>menentukan nilai dari parameter K (jumlah dari tetangga terdekat)</p> <ul style="list-style-type: none"> - Pembelajaran berdasarkan jarak tidak jelas mengenai jenis jarak apa yang harus digunakan dan atribut mana yang harus digunakan untuk mendapatkan hasil yang terbaik.
Klasifikasi Naive Bayes	<ul style="list-style-type: none"> - Mampu bekerja tangguh terhadap data – data yang terisolasi yang biasanya data <i>outliner</i>. - Dapat menangani nilai atribut yang salah dengan mengabaikan data latih selama proses pembangunan model dan prediksi. 	<ul style="list-style-type: none"> - Tidak berlaku jika probabilitas kondisionalnya adalah nol, apabila nol maka probabilitas prediksi akan nol juga. - Diasumsikan variabel bebas

	<ul style="list-style-type: none"> - Atribut mempunyai korelasi bisa mendegradasi kinerja klasifikasi Naive Bayes karena asumsi independensi atribut sudah tidak ada. 	
Convolutional Neural Network (CNN)	<ul style="list-style-type: none"> - <i>Layer</i> dapat ditentukan oleh peneliti. - Banyaknya <i>hidden layer</i> membuat hasil klasifikasi menjadi lebih baik tingkat akurasi. - Dilakukan pelabelan terlebih dahulu untuk data latih sehingga pada data uji hasilnya akan menjadi lebih akurat. 	<ul style="list-style-type: none"> - CNN hanya dapat digunakan pada data yang memiliki struktur dua dimensi seperti citra dan suara
Support Vector Machine (SVM)	<ul style="list-style-type: none"> - Generalisasi : Kemampuan suatu metode untuk mengklasifikasikan suatu <i>pattern</i>, yang tidak termasuk data yang dipakai dalam data latihnya. - <i>Curse of</i> 	<ul style="list-style-type: none"> - Sulit dipakai dalam skala besar (jumlah sampel) - SVM dikembangkan untuk klasifikasi dua kelas.

	<p><i>dimensionality</i> :</p> <p>Masalah yang dihadapi suatu metode <i>pattern recognition</i> dalam mengestimasi parameter (jumlah <i>hidden neuron</i> pada <i>neural network</i>, <i>stopping criteria</i> dalam proses pembelajaran, karena data sampel yang relatif sedikit.</p> <p>- <i>Feasibility</i> : SVM dapat diimplementasikan dengan mudah, karena dapat dirumuskan dalam <i>QP problem</i>.</p>	
--	---	--

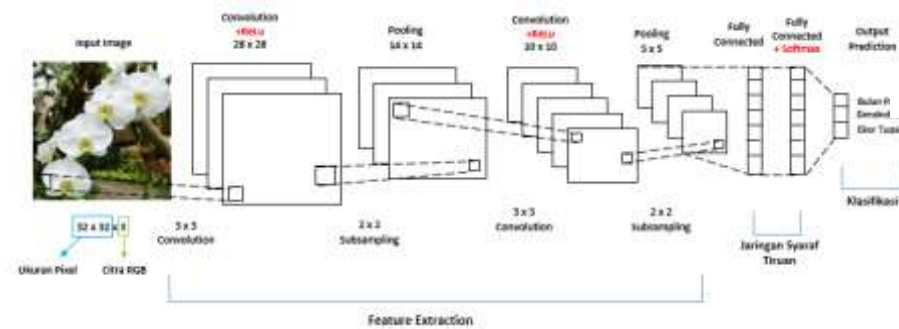
Dari perbandingan metode pada **tabel 1**, maka peneliti memilih menggunakan metode *convolutional neural network* (CNN) karena metode tersebut kelebihanannya yaitu dapat digunakan untuk data yang berjumlah besar, dapat sesuai ketentuan peneliti untuk membentuk model untuk banyaknya lapisan *layer* yang digunakan, kemudian juga dikarenakan termasuk pada *supervised learning* artinya analisis dilakukan dengan melatih data *training* sehingga hasil yang didapatkan untuk klasifikasi akan semakin akurat.

3.8. *Convolutional Neural Network*

Convolutional Neural Network (CNN/ConvNet) merupakan salah satu algoritma dari *deep learning* yang merupakan hasil pengembangan dari *Multilayer Perceptron* (MLP) yang dirancang untuk melakukan olah data menjadi bentuk dua dimensi, misalnya yaitu: gambar atau suara. CNN digunakan untuk melakukan klasifikasi data yang berlabel dengan menggunakan metode *supervised learning* yang cara kerjanya dari *supervised learning* adalah terdapat data yang dilatih dan terdapat variabel yang ditargetkan sehingga tujuan dari metode ini yaitu mengelompokkan suatu data ke data yang sudah ada.

CNN sering digunakan untuk mengenali benda atas pemandangan dan melakukan deteksi dan melakukan segmentasi objek. CNN belajar langsung melalui data citra, sehingga dapat menghilangkan ekstraksi ciri dengan cara manual. Penelitian awal yang menjadi dasar penemuan ini yaitu pertama kali dilakukan oleh Hubel dan Wiesel yang melakukan penelitian *visual cortex* pada indera penglihatan kucing. *Visual cortex* pada hewan sangat *powerful* kemampuannya dalam sistem pemrosesan visual yang pernah ada. Sehingga, banyak penelitian yang terinspirasi oleh cara kerjanya dan menghasilkan banyak model-model baru yang beberapa diantaranya yaitu, *Neocognitron*, HMAX, LeNet-5, dan AlexNet.

CNN juga merupakan syaraf yang dikhususkan untuk memproses data yang memiliki struktur kotak (*grid*). Sebagai contoh yaitu berupa citra dua dimensi. Nama konvolusi merupakan operasi dari aljabar linear yang mengalikan matriks dari filter pada citra yang akan diproses. Proses ini disebut dengan lapisan konvolusi dan merupakan salah satu jenis dari banyak lapisan yang bisa dimiliki dalam suatu jaringan. Meskipun begitu, lapisan konvolusi ini merupakan lapisan utama yang paling penting digunakan. Jenis lapisan yang lain yang biasa digunakan adalah *Pooling Layer*, yakni lapisan yang digunakan untuk mengambil suatu nilai maksimal atau nilai rata-rata dari bagian-bagian lapisan piksel pada citra. Berikut merupakan gambaran umum arsitektur *Convolution Net* :



Gambar 3. 8 Contoh Jaringan CNN

Pada **gambar 3.8** yaitu di setiap lapisan *input* yang dimasukkan mempunyai susunan neuron 3 dimensi, yaitu lebar, tinggi, dan kedalaman). Lebar dan tinggi yaitu ukuran lapisan, sedangkan untuk ke dalam yaitu mengacu pada jumlah lapisan. Setiap besaran yang didapat terhantung dari hasil fitrasi dari lapisan sebelumnya dan banyaknya filter yang digunakan. Model jaringan seperti ini sudah terbukti efektif dalam menangani permasalahan klasifikasi citra. Sebuah CNN mampu memiliki puluhan hingga ratusan lapisan yang masing-masing lapisan mempelajari deteksi berbagai gambar. Pengolahan citra diterapkan pada setiap citra latih pada resolusi yang berbeda, dan *output* dari masing-masing data gambar yang diolah dan digunakan sebagai *input* ke lapisan berikutnya. Pengolahan citra dapat dimulai sebagai fitur yang sederhana, seperti ukuran kecerahan dan tepi atau meningkatkan kekompleksan pada fitur secara unik untuk menentukan objek sesuai ketebalan lapisan (MathWorks, 2018).

Secara umum tipe lapisan CNN dibagi menjadi dua bagian, yaitu :

a. Layer Ekstraksi Fitur (*feature extraction layer*)

Gambar yang letaknya ada di awal arsitektur yang tersusun atas beberapa lapisan dan di setiap susunan lapisannya atas neuron yang terkoneksi pada daerah lokal (*local region*) dari lapisan sebelumnya. Lapisan pada jenis pertama yaitu adalah *convolutional layer* dan lapisan kedua adalah *pooling layer*. Pada setiap lapisan diberlakukan fungsi aktivasi dengan posisinya yang berselang-seling antara jenis pertama dan jenis kedua. Lapisan ini

menerima *input* gambar secara langsung dan memprosesnya sampai menghasilkan *output* berupa vektor untuk diolah di lapisan berikutnya.

b. Layer Klasifikasi (*classification layer*)

Layer ini tersusun atas beberapa lapisan yang di setiap lapisan tersusun atas neuron yang terkoneksi secara penuh (*fully connected*) dengan lapisan yang lainnya. *Layer* ini menerima *input* dari hasil *output layer* ekstraksi fitur gambar berupa vektor yang kemudian di transformasikan seperti pada *Multi Neural Network* dengan tambahan beberapa *hidden layer*. Hasil *output* berupa akurasi kelas untuk klasifikasi.

Dengan ini, CNN merupakan metode untuk melakukan transformasi gambar asli lapisan per lapisan dari nilai piksel gambar ke dalam nilai skoring kelas untuk klasifikasi. Setiap lapisan ada yang memiliki *hyperparameter* dan ada yang tidak memiliki parameter (bobot dan bias pada neuron).

3.9. Operasi Konvolusi

Operasi konvolusi merupakan operasi dua fungsi argumen yang bernilai nyata (Goodfellow, Bengio, & Courville, 2016). Operasi ini menerapkan fungsi keluaran (*output*) sebagai *Feature Maps* dari masukan (*input*) citra. *Input* dan *output* ini dapat dilihat sebagai dua argumen yang mempunyai nilai riil. Secara khusus, operasi konvolusi dapat ditulis dengan rumus berikut :

$$s(t) = (x * w)(t) \dots \dots \dots (3.1)$$

Fungsi $s(t)$ memberikan *output* tunggal yaitu *Feature Maps*, argumen pertama yaitu *input* yang merupakan x dan argumen kedua w sebagai *kernel* atau *filter*. Jika dilihat *input* sebagai citra dua dimensi, maka dapat diasumsikan bahwa t sebagai piksel dan menggantinya dengan i dan j . Maka dari itu, operasi untuk konvolusi ke dalam *input* lebih dari satu dimensi dapat ditulis sebagai berikut :

$$S(i, j) = (K * I)(i, j) = \sum_{m=1}^m \sum_{n=1}^n I(i - m, j - n) K(m, n) \dots \dots \dots (3.2)$$

Pada persamaan 3.2 adalah perhitungan dasar dalam operasi konvolusi dimana i dan j adalah piksel dari citra. Perhitungannya bersifat kumulatif dan

muncul saat K sebagai *kernel*, I sebagai *input* dan *kernel* yang dapat dibalik relatif terhadap *input*. Sebagai alternatif, operasi konvolusi dapat dilihat sebagai perkalian matriks antara citra masukan dan *kernel* yang dimana keluarannya dapat dihitung dengan *dot product*.

Peneliti dapat menentukan volume *output* dari masing-masing lapisan dengan *hyperparameters*. *Hyperparameters* yang digunakan dalam persamaan 3.3 digunakan untuk menghitung berapa banyak *neuron* aktivasi dalam sekali keluaran (*output*).

$$(W - F + 2P)/S + 1 \dots\dots\dots (3.3)$$

Dari persamaa 3.3 yaitu perhitungan untuk menghitung ukuran spasial dari volume *output* dimana *hyperparameter* yang dipakai yaitu ukuran volume (W), *filter* (F), *stride* yang diterapkan (S) dan jumlah *padding* nol yang digunakan (P). *Stride* adalah nilai yang digunakan untuk menggeser *filter* melalui *input* citra. Sedangkan, *zero padding* adalah nilai yang digunakan untuk menenmpatkan angka nol di sekitar border citra.

Dalam pengolahan citra, konvolusi berarti mengaplikasikan sebuah *kernel* (kotak kuning) pada citra di semua *offset* yang memungkinkan seperti pada **gambar 3.9** :

4	3	4	1	1	1	0	0
2	4	3	0	1	1	1	0
2			0	0	1	1	1
			0	0	1	1	0
			0	1	1	0	0
<i>Image</i>			<i>Convolved Feature</i>				

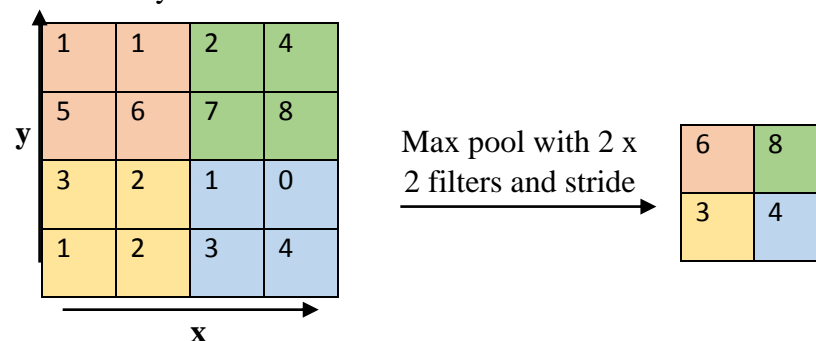
Gambar 3. 9 Operasi pada Konvolusi

Kotak hijau secara keseluruhan adalah citra yang akan dilakukan konvolusi. *Kernel* bergerak dari sudut kiri atas ke kanan bawah. Sehingga hasil

konvolusi dari citra tersebut dapat dilihat dari gambar di sebelah kanan. Tujuannya dilakukan konvolusi pada data citra yaitu untuk mengekstraksi fitur dari citra *input*. Konvolusi akan menghasilkan transformasi *linear* dari data *input* sesuai informasi spasial pada data. Bobot pada lapisan tersebut mengspesifikasi *kernel* konvolusi yang digunakan, sehingga kernel konvolusi dapat dilatih berdasarkan *input* pada CNN.

3.10. Pooling Layer

Pooling layer adalah lapisan fungsi untuk *Feature Maps* sebagai masukan dan mengolahnya dengan berbagai operasi statistik berdasarkan nilai piksel terdekat. Pada model CNN, lapisan *Pooling* biasanya disisipkan secara teratur setelah beberapa lapisan konvolusi. Lapisan *pooling* yang dimasukkan di antara lapisan konvolusi secara berturut-turut dalam susunan arsitektur model CNN dapat secara progresif mengurangi ukuran volume *output* pada *Feature Maps*, sehingga dapat mengurangi jumlah parameter dan perhitungan di jaringan, dan untuk mengendalikan *overfitting*. Hal penting dalam pembuatan model CNN adalah dengan memilih banyak jenis lapisan *pooling* yang dalam hal ini dapat menguntungkan kinerja model (Lee, Gallagher, & Tu, 2015). Lapisan *pooling* bekerja di setiap susunan *Feature Maps* dan mengurangi ukurannya. Bentuk lapisan *pooling* yang paling umum adalah dengan menggunakan *filter* berukuran 2 x 2 yang di aplikasikan dengan langkah sebanyak 2 dan kemudian beroperasi pada setiap irisan dari *input*. Bentuk seperti ini akan mengurangi *Feature Maps* hingga 75% dari ukuran aslinya.



Gambar 3. 10 Operasi pada *Max Pooling*

Lapisan pada *pooling* akan berjalan pada setiap irisan kedalaman volume *input* secara bergantian. Pada **gambar 3.10** lapisan *pooling* menggunakan salah satu operasi maksimal (*max pooling*) yang merupakan operasi yang paling umum. **Gambar 3.9** menunjukkan operasi dengan langkah 2 dan ukuran *filter* 2 x 2. Dari ukuran *input* 4 x 4, pada masing-masing 4 angka pada *input* operasi mengambil nilai maksimalnya dan membuat ukuran *ouput* baru menjadi 2 x 2.

Penggunaan *pooling layer* pada CNN bertujuan untuk mereduksi ukuran citra sehingga dapat dengan mudah digantikan dengan sebuah *convolution layer* dengan *stride* yang sama dengan *pooling layer* yang bersangkutan (Springenberg, Dosovitskiy, Brox, & Riedmiller, 2015).

3.11. Aktivasi ReLu

Aktivasi ReLu (*Rectified Linear Unit*) adalah lapisan aktivasi pada model CNN yang mengaplikasikan fungsi $f(x) = \max(0, x)$ yang artinya fungsi ini melakukan *Thresholding* dengan nilai nol terhadap nilai piksel pada *input* citra. Aktivasi ini membuat seluruh nilai piksel yang bernilai kurang dari nol pada suatu citra akan dijadikan 0.

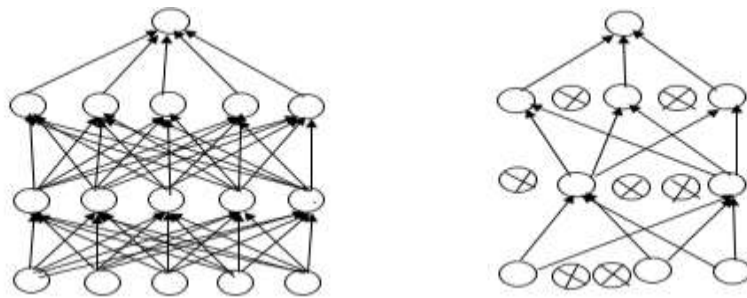
3.12. Fully-Connected Layer

Lapisan *fully-connected* merupakan lapisan dimana seluruh *neuron* aktivasi dari lapisan sebelumnya terhubung dengan *neuron* pada lapisan selanjutnya seperti halnya jaringan syaraf tiruan biasa. Setiap aktivasi dari lapisan sebelumnya perlu dirubah menjadi data satu dimensi sebelum dapat dihubungkan ke semua *neuron* di lapisan *fully-connected*. Lapisan *fully-connected* biasanya digunakan pada metode *Multilayer Perceptron* dan gunanya untuk mengolah data sehingga dapat diklasifikasikan.

Perbedaan antara lapisan *fully-connected* dan lapisan konvolusi biasa yaitu *neuron* di lapisan konvolusi hanya terhubungan ke daerah tertentu pada *input*, sementara lapisan *fully-connected* memiliki *neuron* yang secara keseluruhan akan terhubung. Namun, kedua lapisan tersebut masih mengoperasikan *dot product*, sehingga fungsinya tidak begitu berbeda.

3.13. Dropout Regularization

Dropout yaitu merupakan teknik regularisasi jaringan syaraf dimana beberapa akan dipilih secara *random* dan tidak dipakai selama data latih. *Neuron-neuron* ini dibuang juga secara *random*. Hal ini, artinya bahwa kontribusi *neuron* yang dibuang akan dihentikan sementara jaringan dan bobot baru juga tidak diterapkan pada *neuron* pada saat melakukan *backpropagation*.



(a) *Neural Network Biasa*

(b) Setelah dilakukan *dropout*

Gambar 3. 11 Jaringan Syaraf Sebelum dan Sesudah Mengaplikasikan *Dropout*

Sumber : cs231n.github.io

Gambar 3.11 merupakan jaringan syaraf biasa (a) dengan 2 lapisan yang tersembunyi. Sedangkan, untuk bagian (b) yaitu jaringan syaraf yang sudah diaplikasikan teknik regularisasi *dropout* yang terdapat beberapa *neuron* aktivasi yang tidak terpakai lagi. Teknik ini sangat mudah diterapkan pada model CNN sehingga akan berdampak pada performa model dalam melatih serta mengurangi *overfitting* (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014).

3.14. Softmax Classifier

Softmax classifier adalah bentuk lain dari algoritma regresi logistik yang dapat digunakan untuk melakukan klasifikasi lebih dari dua kelas. Standar dari klasifikasi yang biasanya dilakukan oleh algoritma dari regresi logistik adalah tugas untuk klasifikasi kelas biner. Pada *softmax* mempunyai bentuk persamaan sebagai berikut :

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \quad (3.4)$$

Pada persamaan 3.4 untuk notasi f_j menunjukkan hasil fungsi untuk setiap elemen ke- j pada vektor *output* kelas. Argumen z merupakan hipotesis yang diberikan oleh model pelatihan supaya dapat diklasifikasikan oleh fungsi *softmax*.

Softmax juga memberikan hasil yang lebih intuitif dan memiliki hasil interpretasi probabilistik yang lebih baik dibandingkan dengan algoritma klasifikasi lainnya. *Softmax* memungkinkan peneliti untuk menghitung nilai probabilitas untuk semua label. Hasil dari label yang ada, akan diambil sebuah vektor nilai yang mempunyai nilai riil dan merubahnya menjadi vektor dengan nilai antara nol dan satu. Jika semua hasil dijumlah maka akan bernilai satu.

3.15. *Crossentropy Loss Function*

Lost function atau *cost function* adalah fungsi yang menggambarkan kerugian yang berkaitan dengan semua kemungkinan yang dihasilkan oleh model. *Loss function* bekerja ketika model pembelajaran memberikan kesalahan yang harus diperhatikan. *Loss function* yang baik yaitu dimana fungsi yang menghasilkan *error*/kesalahan yang diharapkan paling rendah.

Jika suatu model memiliki kelas yang cukup banyak, perlu adanya cara untuk mengukur perbedaan antara probabilitas hasil hipotesis dengan probabilitas kebenaran yang asli, dan selama pelatihan banyak algoritma yang dapat menyesuaikan parameter sehingga perbedaan ini diminimalkan. *Crossentropy* adalah pilihan yang masuk akal.

Gambaran umum algoritma ini yaitu untuk meminimalkan kemungkinan log negatif dari *dataset*, yang merupakan ukuran langsung dari hasil prediksi model.

3.16. *Stochastic Gradient Descent*

Gradient Descent merupakan salah satu algoritma yang paling populer dalam melakukan optimasi pada model jaringan syaraf tiruan. Algoritma ini adalah cara yang paling sering dipakai dalam berbagai macam model pembelajaran. Ketika akan melatih sebuah model, akan dibutuhkan sebuah *loss function* yang dapat memungkinkan peneliti untuk mengukur kualitas dari setiap

bobot atau parameter tertentu. Tujuan dari pengoptimalan ini yaitu untuk menentukan parameter manakah yang mampu meminimalkan *loss function* (Ruder, 2017).

Gradient Descent bekerja dengan meminimalkan fungsi $J(\theta)$ yang mempunyai parameter θ dengan memperbarui parameter ke suatu arah yang menurun. *Gradient descent* mempunyai *learning rate* (η) yang digunakan untuk menentukan langkah yang akan diambil untuk mencapai pada titik minimum. Hal ini, dapat digambarkan bahwa suatu objek akan seperti menuruni sebuah bukit dengan langkah tersebut sehingga mencapai pada bagian lembah (titik minimum).

Stochastic Gradient Descent (SGD) merupakan metode *gradient descent* yang melakukan *update* parameter untuk setiap data pelatihan $x(i)$ dan label $y(i)$ dan mempunyai persamaan dasar berikut :

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)}) \quad (3.5)$$

SGD seringkali melakukan *update*/pembaruan dengan varians yang tinggi, sehingga menyebabkan fungsi objektif meningkat secara tidak beraturan. Di satu sisi, hal ini dapat membuat *loss function* melompat ke titik minimal yang baru dan mempunyai potensi untuk melompat ke nilai minimum yang tidak pasti. Namun, hal ini dapat dicegah dengan mengurangi nilai *learning rate*, dan hasil SGD akan menuruni *loss function* ke titik minimum dengan optimal.

3.17. *Max Norm Constraint*

Max norm constraint merupakan salah satu bentuk lain dari regularisasi yang mempunyai fungsi untuk mengubah bobot pada vektor untuk setiap *neuron* dan dapat digunakan untuk mengoptimisasi gradien. Saat melakukan pelatihan regularisasi ini melakukan *update* parameter pada optimasi biasa, dan kemudian menerapkan suatu batasan yang dapat merubah vektor bobot. Penggunaan regularisasi ini dapat menunjukkan perbaikan dari hasil model yang sudah dilatih.

Regularisasi ini merupakan cara yang lebih baik untuk membatasi kompleksitas model secara eksplisit merubah parameter di setiap iterasi jika nilai parameter melebihi ambang batas kewajaran.

3.18. *Confusion Matriks*

Confusion matriks adalah sebuah metode yang digunakan untuk menghitung nilai akurasi pada konsep data mining. Evaluasi dengan *confusion* matriks menghasilkan nilai akurasi, presisi, dan *recall*. Nilai akurasi dalam klasifikasi adalah presentase ketepatan *record data* yang diklasifikasikan dengan tepat setelah dilakukan pengujian pada hasil klasifikasi. Presisi atau *confidence* adalah nilai proporsi dari kasus yang diprediksi positif yang juga positif benar pada data sebenarnya. *Recall* atau *confidence* merupakan proporsi kasus yang diprediksi positif yang sebenarnya diprediksi positif secara tepat (Mayadewi & Rosely, 2015).

Tabel 2 Model *Confusion* Matriks

<i>Correct Classification</i>	<i>Classified as</i>	
	+	-
+	<i>True Positives (A)</i>	<i>False Negatives (B)</i>
-	<i>False Positives (C)</i>	<i>True Negatives (D)</i>

Pada **tabel 1** adalah perhitungan akurasi dengan tabel *confusion* matriks adalah sebagai berikut :

$$\text{Akurasi} = (A+D)/(A+B+C+D) \dots\dots\dots (3.6)$$

Presisi adalah hasil rasio item relevan yang dipilih terhadap seluruh item yang terpilih. Presisi dapat diartikan sebagai kecocokan antara permintaan informasi dan jawaban terhadap permintaan tersebut. Rumus dari presisi adalah :

$$\text{Presisi} = A/(C+A) \dots\dots\dots (3.7)$$

Recall adalah rasio dari *item* relevan yang dipilih terhadap total jumlah *item* yang relevan dan tersedia. *Recall* dapat dihitung dengan rumus berikut :

$$Recall = A/(A+D) \dots\dots\dots (3.8)$$

Presisi dan *recall* dapat diberi nilai angka dengan menggunakan nilai presentase (1-100%) atau menggunakan bilangan 0-1. Sistem rekomendasi akan dianggap baik jika nilai presisi dan *recall*nya tinggi.

Kurva ROC menunjukkan bahwa nilai akurasi dalam membandingkan klasifikasi secara visual. ROC menginterpretasikan *confusion* matriks. ROC adalah grafik dua dimensi dengan *false positive* sebagai garis horizontal dan *true positive* sebagai garis vertikal. AUC (*the area under curve*) dihitung untuk mengukur perbedaan performansi metode yang digunakan. ROC mempunyai tingkat nilai diagnosa sebagai berikut :

- a. Nilai akurasi bernilai 0,90 – 1,00 artinya *excellent classification*
- b. Nilai akurasi bernilai 0,80 – 0,90 artinya *good classification*
- c. Nilai akurasi bernilai 0,70 – 0,80 artinya *fair classification*
- d. Nilai akurasi bernilai 0,60 – 0,70 artinya *poor classification*
- e. Nilai akurasi bernilai 0,50 – 0,60 artinya *failure*

3.19. R-Studio

R-Studio merupakan *software open source* yang terintegrasi dengan R, bahasa pemrograman yang digunakan yaitu untuk komputasi statistik dan grafis. R-studi ditemukan oleh JJ. Allaire, pencipta bahasa pemrograman ColdFusion. Hadley Wickham adalah kepala ilmuwan di *R-Studio*. *R-Studio* tersedia dalam dua edisi, yaitu : *Rstudio Desktop*, tempat program aplikasi dijalankan di dalam *desktop* biasa, dan *R-Studio Server* yang digunakan untuk mengakses *R-Studio* menggunakan *browser web* saat sedang berjalan di *server Linux*. *R-Studio desktop* tersedia untuk *Windows*, *MacOS*, dan *Linux*.

R-Studio merupakan perkembangan dari *R-Programming* yang juga menyediakan berbagai teknik dalam statistika yaitu permodelan liner dan non linier, uji statistik klasik, analisis deret waktu, klasifikasi, klasterisasi, grafik, dan lain sebagainya. Kelebihan R ini adalah fasilitas grafiknya yang menghasilkan grafik dengan kualitas publikasi yang dapat memuat simbol matematika. R

memiliki format dokumentasi seperti *LaTex*, yang digunakan untuk menyediakan dokumentasi yang lengkap, baik dalam berbagai format, maupun secara cetakan (Wikipedia, 2017).

3.20. Keras

Keras adalah jaringan syaraf tiruan tingkat tinggi yang dikembangkan dengan fokus dan memungkinkan eksperimen dengan cepat. Keras memiliki beberapa fitur utama:

- a. Memungkinkan kode yang sama dijalankan di CPU atau di GPU, tanpa hambatan.
- b. *User-friendly* API yang memudahkan pembuatan *prototype* model pembelajaran dengan cepat.
- c. *Built-in* mendukung jaringan konvolusi (untuk penglihatan komputer), jaringan berulang (dalam proses sekuensial), dan kombinasi keduanya.
- d. Mendukung arsitektur jaringan yang sewenang-wenang: model multi *input* atau multi *output*, berbagi lapisan, berbagi model, dan sebagainya. Artinya, Keras sesuai untuk membangun model pembelajaran, dari jaringan memori ke mesin turing syaraf.

3.21. Fatkhun Batch Image Downloader

Fatkhun batch image downloader adalah sebuah ekstensi atau fitur tambahan yang ada pada *software* Google Chrome. *Software* tambahan tersebut digunakan untuk mengunduh seluruh gambar dalam 1 kali klik dalam pencarian gambar, sehingga pengunduh dapat mengunduh berdasarkan keinginannya.

Selain dapat mengunduh sesuai dengan keinginan. *Software* ini juga dapat digunakan untuk menyaring *link* atau menyaring resolusi gambar sesuai kebutuhan pengunduh.

BAB IV

METODE PENELITIAN

4.1. Populasi dan Sampel

Populasi dalam penelitian ini digunakan gambar atau objek tanaman yaitu bunga anggrek. Gambar bunga anggrek yang digunakan yaitu 3 jenis anggrek dengan jenis anggrek bulan putih, anggrek dendrobium, dan anggrek ekor tupai. Sedangkan, untuk sampel yang digunakan untuk 3 jenis tersebut di masing-masing 3 jenis anggrek diambil 100 gambar. Kemudian, dilakukan penyortiran.

4.2. Variabel dan Definisi Operasional

Definisi operasional variabel penelitian merupakan penjelasan dari masing-masing variabel yang digunakan dalam penelitian terhadap indikator-indikator yang membentuknya. Definisi operasional penelitian ini dapat dilihat pada **tabel 2** :

No	Variabel	Kode	Definisi Operasional
1	Anggrek Bulan Putih	'bp(nomor).jpg' ,	Citra anggrek bulan putih sebanyak 30 citra sampel untuk data <i>training</i> , dan 5 citra sampel untuk data <i>testing</i>
2	Anggrek Dendrobium	'd(nomor).jpg'	Citra anggrek dendrobium sebanyak 30 citra sampel untuk data <i>training</i> , dan 5 citra sampel untuk data <i>testing</i>
3	Anggrek Ekor Tupai	'et(nomor).jpg' ,	Citra anggrek ekor tupai sebanyak 30 citra sampel untuk data <i>training</i> , dan 5 citra sampel untuk data <i>testing</i>

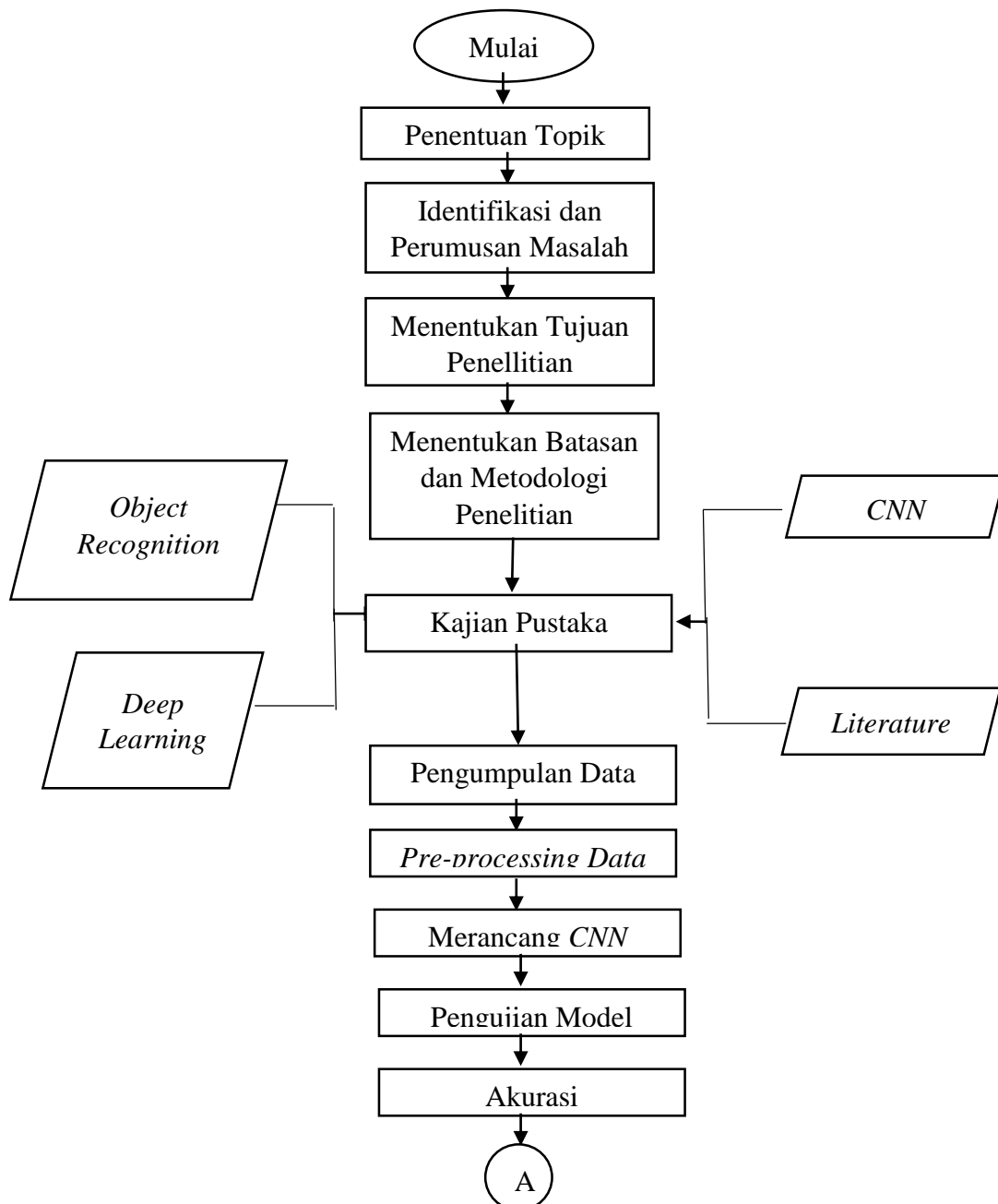
Tabel 1 Tabel Definisi Operasional Penelitian

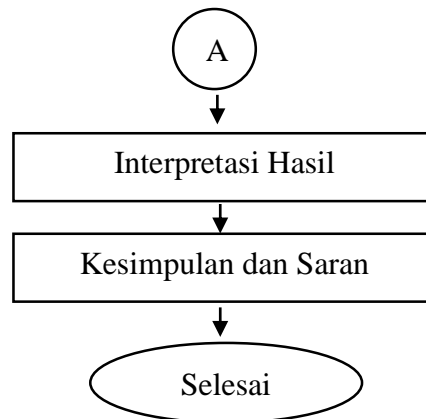
4.3. Teknik Sampling

Teknik sampling yang digunakan³⁴ itu *purposive sampling* yaitu sampel yang sudah dipilih sesuai dengan tujuan penelitian. Peneliti memilih sampel gambar untuk data *training* dan data *testing*.

4.4. Alat dan Cara Organisasi Data

Alat dan cara organisasi data ditampilkan dalam diagram alir pada **gambar 4.1** :





Gambar 4. 1 Alur Penelitian

Pada **gambar 4.1** adalah gambar alur penelitian yang digunakan dalam penelitian yang jika diringkas dari seluruh tahapan yaitu hanya akan menjadi 5 tahapan, yaitu tahapan pendahuluan, tahapan kajian pustaka, tahapan pengumpulan dan pengolahan data, interpretasi hasil, serta kesimpulan dan saran.

Tahap pertama, yaitu dalam tahapan pendahuluan yang artinya tahapan ini dalam diagram alur yaitu dari mulai, kemudian menentukan topik dari penelitian yang akan dilakukan oleh peneliti sebagai dasar utama penelitian, kemudian identifikasi dan rumusan masalah dari penelitian ini, menentukan tujuan utama penelitian ini, dan menentukan batasan permasalahan dan metodologi penelitian yang digunakan oleh peneliti.

Tahap kedua, yaitu tahapan kajian pustaka yaitu sebagai referensi atau landasan digunakan peneliti untuk melakukan penelitian baik itu dari penelitian yang sama sebelumnya pernah diperlukan, atau mengembangkan penelitian yang sudah ada. Dalam melakukan kajian pustaka, hal-hal yang digunakan peneliti sebagai kajian pustaka yaitu mengenai *object recognition*, *deep learning*, *convolutional neural network*, *machine learning*, dan lain sebagainya yang berkaitan dengan penelitian ini.

Tahap ketiga, yaitu tahapan pengumpulan dan pengolahan data. Pengumpulan data yang dilakukan dalam penelitian ini yaitu menggunakan pengunduhan citra melalui *google* dan menentukan banyak citra yang diunduh, dan yang akan digunakan dalam melakukan analisis. Setelah dilakukan pengumpulan baik *data training* maupun *data testing*, dilakukan pengolahan data yaitu *image pre-processing* yaitu dengan cara mengubah citra dari warna menjadi citra *grayscale* (hitam putih) dan mengubah ukuran dari citra dengan mengubah ukuran piksel citra sesuai yang ditentukan peneliti. Setelah diubah menjadi *grayscale* dan mengubah ukuran piksel citra, dilakukan perancangan CNN dengan melatih data dalam pembelajaran program sehingga program mampu mendeteksi objek dengan membentuk *layer*, *pooling*, *dropout* untuk mendapatkan hasil deteksi objek sesuai dengan tujuan penelitian. Setelah mampu mendeteksi objek yang digunakan, dilanjutkan dengan pengujian model yang didapat dari hasil pengolahan citra yang telah dilakukan. Setelah model didapatkan, dilakukan pengecekan terhadap hasil akurasi dari model yang telah didapatkan mengenai kemampuan program dalam melakukan pendeteksian objek yang digunakan.

Tahap keempat, yaitu tahap interpretasi hasil yang didapatkan dari seluruh pengolahan data yang dilakukan dalam penelitian.

Tahap kelima, yaitu masuk ke dalam kesimpulan dari keseluruhan hasil deteksi objek dengan CNN tentang model yang didapatkan dan nilai tingkat akurasi yang didapatkan, dan saran yang digunakan untuk peneliti selanjutnya untuk mengembangkan penelitian ini, dan lain sebagainya. Kemudian, tahapan selesai dilakukan.

BAB V

HASIL DAN PEMBAHASAN

Pada bab ini akan interpretasikan hasil dari klasifikasi gambar dengan 3 kategori objek citra, yaitu citra bunga anggrek bulan putih, citra bunga anggrek dendrobium, dan citra bunga anggrek ekor tupai dengan menggunakan teknik *deep learning* dengan salah satu metode yang digunakan *convolutional neural network* (CNN). Hasil dan pembahasan akan dibagi menjadi 4 bagian, yaitu pengumpulan data citra, histogram citra, *preprocessing* citra, dan pengolahan citra.

5.1. Pengumpulan Data Citra

Pengumpulan data citra dilakukan dengan menggunakan aplikasi tambahan dalam *google* yaitu *Fatkun Image Downloader*. Aplikasi *Fatkun Image Downloader* ini mampu sekaligus mengunduh citra dalam jumlah besar. Citra yang diunduh dalam penelitian ini yaitu 3 kategori untuk bunga anggrek, yaitu: anggrek bulan putih sebanyak 40 citra, anggrek dendrobium sebanyak 40 citra, dan anggrek ekor tupai sebanyak 40 citra, dengan keseluruhan citra diambil 90 citra untuk data *training*, dan 30 citra untuk data *testing*.

Citra yang telah diunduh, kemudian diberikan penamaan citra yang disesuaikan dengan kategori bunga anggrek yang telah ditentukan. Bunga anggrek bulan putih diberi nama 'bp(nomor).jpg' untuk citra kategori bunga anggrek bulan putih, bunga anggrek dendrobium diberi nama 'd(nomor).jpg' untuk citra kategori bunga dendrobium, dan bunga anggrek ekor tupai diberi nama 'et(nomor).jpg' untuk citra kategori bunga anggrek ekor tupai dalam satu *file* penyimpanan.

Citra yang telah diberi penamaan, kemudian di *input* ke dalam program *software R*, dengan *input* 140 citra untuk data *training* pertama dan 90 data *training* ke dua dan *input* 30 citra untuk data *testing* untuk percobaan pertama dan kedua sama.

Data yang telah *diinput* akan dapat terbaca dalam program, berikut merupakan contoh hasil *input* data yang contoh salah satunya citra bunga anggrek bulan putih untuk *input* citra pertama ke dalam program :

```
> display(train[[1]])
```



Gambar 5. 1 *Input* Citra

Hasil citra yang telah *diinput* akan menghasilkan matriks piksel pada setiap masing-masing citra, dengan contoh pada citra **gambar 5.1** memiliki matriks piksel :

```
> print(train[[1]])
Image
  colorMode      : Color
  storage.mode   : double
  dim            : 259 194 3
  frames.total   : 3
  frames.render  : 1

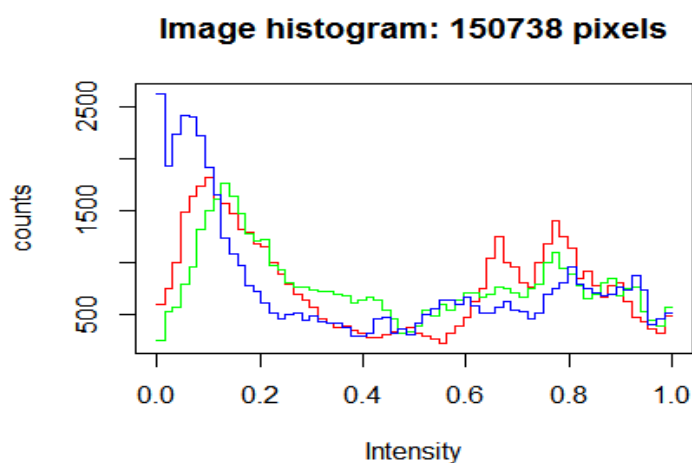
imageData(object)[1:5,1:6,1]
      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[1,] 0.6470588 0.6470588 0.6470588 0.6549020 0.6549020 0.6666667
[2,] 0.6470588 0.6470588 0.6509804 0.6549020 0.6627451 0.6666667
[3,] 0.6509804 0.6509804 0.6549020 0.6627451 0.6666667 0.6705882
[4,] 0.6549020 0.6549020 0.6627451 0.6666667 0.6705882 0.6705882
[5,] 0.6627451 0.6627451 0.6666667 0.6705882 0.6705882 0.6823529
```

Gambar 5. 2 Matrik Piksel pada Citra

Pada **gambar 5.2** gambar *input* dari **gambar 5.1** mempunyai dimensi dengan lebar citra 259 piksel x 194 piksel untuk tinggi citra dan memiliki 3 *channel* RGB (warna). Matriks yang dihasilkan membentuk gambar sesuai *detail* pada piksel yang dihasilkan.

5.2. Histogram Citra

Histogram citra digunakan untuk melihat intensitas cahaya atau penerangan pada sekitar warna merah, hijau, dan biru (RGB) untuk citra yang di *input* dikarenakan citra *input* berwarna dan diwakili dengan RGB. Untuk warna merah, hijau dan biru menuju ke arah kiri dengan intensitas dibawah 0,2 yang artinya tidak terlalu cerah untuk warna RGB dengan objek citra 150.738 piksel dari hasil perkalian 259 piksel x 194 piksel x 3 *channel* RGB.



Gambar 5. 3 Histogram Citra

5.3. Preprocessing Citra

Tahap *preprocessing* citra dilakukan dengan melakukan perubahan ukuran piksel pada citra (*resize image*) dari keseluruhan citra yang mempunyai ukuran piksel berbeda-beda di setiap citra, agar mempermudah dalam melakukan analisis lebih baik dilakukan penyamaan ukuran piksel, baik untuk data *training* maupun data *testing*.

```
> str(train)
List of 140
 $ :Formal class 'Image' [package "EBImage"] with 2 slots
  .. ..@ .Data      : num [1:259, 1:194, 1:3] 0.647 0.647 0.651 0.655 0.663 ...
  .. ..@ colormode: int 2
 $ :Formal class 'Image' [package "EBImage"] with 2 slots
  .. ..@ .Data      : num [1:259, 1:194, 1:3] 0.0549 0.0471 0.0353 0.0275 0.0235 ...
  .. ..@ colormode: int 2
 $ :Formal class 'Image' [package "EBImage"] with 2 slots
  .. ..@ .Data      : num [1:265, 1:190, 1:3] 0.0706 0.102 0.1608 0.2275 0.2941 ...
  .. ..@ colormode: int 2
 $ :Formal class 'Image' [package "EBImage"] with 2 slots
  .. ..@ .Data      : num [1:194, 1:259, 1:3] 0.141 0.145 0.149 0.145 0.133 ...
  .. ..@ colormode: int 2
```



```
> #Resize&Combine
> str(train)
num [1:140, 1:32, 1:32, 1:3] 0.667 0 0.1301 0.0611 1 ...
```

Gambar 5. 4 Pengubahan Ukuran Piksel Citra pada *Data Training*

```
> str(test)
List of 30
 $ :Formal class 'Image' [package "EBImage"] with 2 slots
 .. ..@ .Data      : num [1:275, 1:183, 1:3] 0.569 0.627 0.576 0.369 0.165 ...
 .. ..@ colormode: int 2
 $ :Formal class 'Image' [package "EBImage"] with 2 slots
 .. ..@ .Data      : num [1:194, 1:259, 1:3] 0.286 0.286 0.29 0.29 0.294 ...
 .. ..@ colormode: int 2
 $ :Formal class 'Image' [package "EBImage"] with 2 slots
 .. ..@ .Data      : num [1:238, 1:211, 1:3] 0.235 0.239 0.251 0.263 0.278 ...
 .. ..@ colormode: int 2
> str(test)
num [1:30, 1:32, 1:32, 1:3] 0.2 0.288 0.266 0.165 0.16 ...
```

Gambar 5. 5 Pengubahan Ukuran Piksel pada Citra *Data Testing*

Pada **gambar 5.4** dan **gambar 5.5** merupakan hasil pengubahan ukuran piksel untuk data *training* dan data *testing* menjadi satu ukuran piksel yaitu 32 piksel x 32 piksel untuk masing-masing citra. Dimensi yang digunakan digunakan ukuran 32 piksel x 32 piksel agar saat dilakukan klasifikasi tidak memerlukan konvolusi dan *pooling* yang terlalu banyak sampai dengan hasil klasifikasi nantinya.

Ukuran dimensi *input* ini juga dilakukan secara seragam agar semua ukuran dimensi sama, dikarenakan untuk memudahkan *processing* citra lebih mudah dilakukan dan diproses melalui ukuran dimensi yang sama di setiap citranya. Setiap citra yang masuk ke dalam proses konvolusi dan *pooling* yang sama dengan ukuran 32 piksel x 32 piksel, jika berbeda maka tidak dapat diproses ke dalam proses konvolusi dan *pooling* karena dengan dimensi yang berbeda maka perubahan pada jumlah setiap node tidak akan sama, sehingga konvolusi dan *pooling* yang digunakan untuk mengolah citra juga berbeda.

Setelah keseluruhan dilakukan pengubahan ukuran, masing-masing citra pada data *training* dan data *testing* dikombinasikan menjadi satu citra untuk data *training* dan satu citra untuk data *testing*.



Gambar 5. 6 Kombinasi Citra untuk *Data Training*



Gambar 5. 7 Kombinasi Citra untuk *Data Testing*

Hasil kombinasi citra di masing-masing data *training* dan data *testing* berurutan berdasarkan kategorinya yaitu bulan putih dari data 1 hingga data bulan putih yang terakhir dilanjutkan dengan anggrek dendrobium dari data 1 hingga terakhir, begitu pula dengan anggrek ekor tupai sesuai dengan pelabelan pada saat *input* data citra. Pada saat *input* citra juga diperhatikan di masing-masing citra secara manual satu-persatu apakah termasuk dalam bunga tersebut atau bukan, karena hal tersebut dapat mempengaruhi hasil klasifikasi dan tingkat akurasi yang didapatkan.

5.4. Pengolahan Citra

Pengolahan citra yang dilakukan dengan melatih data dengan data *training* dengan label sesuai dengan kategori citra, dalam melakukan pelabelan untuk data

convolution untuk lapisan 1 dengan kernel 3 x 3 dan *filter* 32 lapisan menggunakan aktivasi 'relu' untuk mengambil *nodes* paling tinggi dan meneruskannya untuk *convolution* berikutnya, sehingga *output* gambar menjadi dimensi 30 x 30 x 3 RGB, sehingga keseluruhan parameternya menjadi $((3 \times 3 \times 3) + 1 \text{ bias}) \times 32 \text{ filter} = 896 \text{ parameter}$.

Kemudian, dilanjutkan dengan melakukan *convoluion* untuk lapisan 2 menggunakan *filter* 32 lapisan dan kernel 3 x 3 menggunakan aktivasi 'relu' sehingga *output* dimensi menjadi 28 x 28 x 32 *channel*, sehingga keseluruhan parameternya menjadi $((3 \times 3 \times 32) + 1 \text{ bias}) \times 32 \text{ filter} = 9.248 \text{ parameter}$.

Setelah dilakukan *convolution* 1 dan 2, maka dilanjutkan dengan melakukan proses *pooling* sehingga parameter tidak perlu dihitung karena setiap 2 kotak atau 2 piksel x 2 piksel menjadi 1 piksel x 1 piksel sehingga dari *input shape* 28 piksel x 28 piksel menjadi 14 piksel x 14 piksel.

Proses setelah *pooling*, dilanjutkan dengan proses *dropout* dimana dengan batas nilai *rate* = 0,01 dan untuk nilai yang kurang dari 0,01 maka tidak akan dilanjutkan ke dalam proses selanjutnya untuk didalam nilai pikselnya. Setelah dilakukan proses *dropout* maka dilanjutkan dengan proses *convolution* kembali yaitu dengan *input shape* 14 x 14 x 32 *channel* dengan *filter* 64 lapisan dan kernel 3 x 3 menghasilkan *output* berdimensi 12 x 12 x 64 *channel*, sehingga keseluruhan parameternya menjadi $((3 \times 3 \times 32) + 1 \text{ bias}) \times 64 = 18.496 \text{ parameter}$.

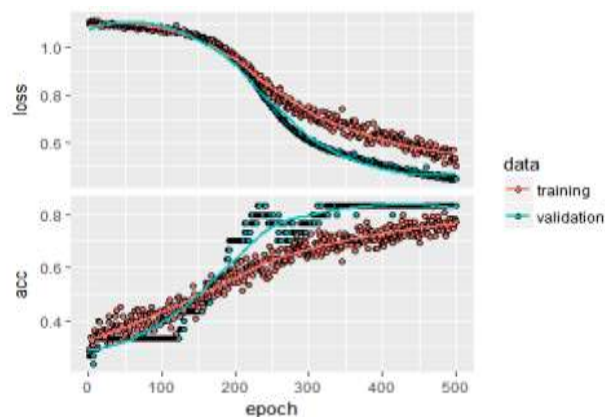
Kemudian, dilakukan kembali proses *convolution* ke 4 dengan *input shape* 12 x 12 x 32 *channel* RGB menggunakan kernel 3 x 3 dan filter sebanyak 64 lapisan menghasilkan *output* berdimensi 10 x 10 x 64 *channel*, sehingga keseluruhan parameternya menjadi $((3 \times 3 \times 64) + 1 \text{ bias}) \times 64 = 36.928 \text{ parameter}$.

Setelah dilakukan *convolution* 2 kali kemudian dinjutkan untuk proses *pooling* yang kedua yaitu dengan *input shape* 10 x 10 menjadi 5 x 5 dan tidak ada parameter pada proses ini. Kemudian dilanjutkan dengan proses *dropout* dan dihasilkan sama dengan hasil pada *pooling*. Setelah *pooling* dan *dropout* maka akan terbentuk *layer flatten* yaitu mengubah menjadi nilai vektor dengan 64

channel dan ukuran 5×5 menjadi bentuk vektor maka tinggal dikalikan menjadi $(5 \times 5 \times 64) = 1.600$, sehingga vektor yang terbentuk ada 1.600 nilai yang masuk ke jaringan syaraf tiruan dimana setiap nilai dari *layer flatten* berasal dari hasil proses konvolusi (*nodes*).

Kemudian, hasil *layer dense* dengan menetapkan sebanyak 256 unit. Hasil vektor sebanyak 1600 nilai akan masuk satu-persatu ke dalam 256 unit maka akan menghasilkan $(1600 \times 256) + 256$ bias = 409.856 parameter. Kemudian, dilakukan *dropout* untuk 256 unit tadi yang nantinya akan diproses kembali untuk *layer dense* sebanyak 3 klasifikasi, sehingga parameter yang akan dihasilkan menjadi $(256 \times 3) + 3$ bias = 771 parameter. Total parameter keseluruhan yang dikerjakan oleh *deep learning* adalah 476.195 perhitungan.

Model arsitektur pada **gambar 5.9** yang akan membentuk proses dan *output* yang akan didapatkan dalam penelitian ini. Setelah terbentuk model, dilakukan *fit model* untuk menentukan nilai akurasi dari data *training* dan data *testing* yang akan ditampilkan dengan hasil berikut :



Gambar 5. 10 *Fit Model*

Hasil pada **gambar 5.10**, peneliti menggunakan *epoch* atau iterasi sebanyak 500 iterasi, yang hasilnya bahwa data mendapatkan hasil akurasi sebesar 81% untuk data *training*. Dari data *training* yang telah diklasifikasikan dan diketahui nilai akurasinya, sehingga terbentuk hasil dalam *confusion matrix* untuk data *training* :

Tabel 2 Hasil Klasifikasi pada Data *Training*

	Bulan Putih	Dendrobium	Ekor Tupai
Bulan Putih	45	2	6
Dendrobium	0	34	7
Ekor Tupai	5	7	34

Hasil klasifikasi yang didapatkan pada **tabel 3**, terbentuk menjadi sebuah *confusion matrix* yang artinya bahwa data yang hasil klasifikasi tidak seluruhnya mampu terbaca sama oleh program sesuai dengan kategori. Pada objek citra bulan putih hanya 45 citra yang mampu terdeteksi kategorinya dari 50 objek citra yang diujikan, objek citra dendrobium sebanyak 34 objek citra dari 43 objek citra yang diujikan, dan objek citra anggrek ekor tupai sebanyak 34 objek citra dari 47 objek citra yang diujikan pada data *training*.

Hasil nilai prediksi pada data *training* yang benar menghasilkan nilai yang cukup besar dibandingkan dengan prediksi data *training* yang salah, maka dapat disimpulkan perhitungan dengan arsitektur jaringan dapat dikatakan baik dengan mendapatkan nilai *precision* sebesar 85%, nilai *recall* sebesar 90% dan nilai akurasi sebesar 81%.

Sedangkan, pada data *testing* program mampu mengklasifikasikan objek citra sebesar 83% dari keseluruhan data *testing* yang diklasifikasikan. Sehingga, didapatkan hasil klasifikasi yang terbentuk dalam *confusion matrix* :

Tabel 3 Hasil Klasifikasi pada Data *Testing*

	Bulan Putih	Dendrobium	Ekor Tupai
Bulan Putih	8	0	1
Dendrobium	0	9	1

Ekor Tupai	2	1	8
-------------------	---	---	---

Hasil klasifikasi yang didapatkan pada **tabel 4** memberikan penjelasan bahwa pada objek citra bulan putih hanya 8 citra yang mampu terdeteksi kategorinya dari 10 objek citra yang diujikan, objek citra dendrobium hanya 9 objek citra dari 10 objek citra yang diujikan, dan objek citra anggrek ekor tupai hanya 8 objek citra dari 10 objek citra yang diujikan.

Hasil dari nilai prediksi pada data *testing* yang benar menghasilkan nilai yang cukup besar dibandingkan dengan prediksi data *testing* yang salah, maka dapat disimpulkan perhitungan dengan arsitektur jaringan dapat dikatakan baik dengan mendapatkan nilai *precision* sebesar 89%, nilai *recall* sebesar 80% dan nilai akurasi sebesar 84%.

Tingkat akurasi 84% artinya kinerja sistem terhadap metode yang digunakan mampu mengerjakan dengan tingkat persentase sebesar 84% dari 100%. Sistem mampu mengklasifikasikan dengan tepat data citra dengan baik sebesar 84%, kemudian sisanya artinya 16% sistem tidak mampu mengklasifikasikan dengan tepat.

Untuk membandingkan hasil akurasi dengan jumlah data yang berbeda, peneliti menggunakan jumlah data objek citra sebanyak 90 data *training*, dan 30 data *testing* dengan hasil akurasi untuk data *training* 87% dengan *confusion matrix* yang dihasilkan :

Tabel 4 Hasil Klasifikasi untuk Data *Training*

	Bulan Putih	Dendrobium	Ekor Tupai
Bulan Putih	27	0	2
Dendrobium	0	27	4
Ekor Tupai	3	3	24

Hasil klasifikasi **tabel 5** untuk data *training* objek data 90 objek citra didapatkan hasil bahwa 27 objek citra anggrek bulan putih terdeteksi benar dari 30 objek citra, untuk objek citra anggrek dendrobium terdeteksi 27 objek citra benar dari 30 objek citra, dan 24 objek citra anggrek ekor tupai dari 30 objek citra. Dari keseluruhan didapatkan nilai presisi sebesar 93%, recall sebesar 90%, dan akurasi sebesar 87%.

Sedangkan, untuk hasil klasifikasi dengan data *testing* untuk 30 objek data hasil akurasinya sama dengan data *testing* sebelumnya yaitu sebesar 83%, terbukti dengan nilai hasil *confusion matrix* :

Tabel 5 Hasil Klasifikasi untuk Data *Testing*

	Bulan Putih	Dendrobium	Ekor Tupai
Bulan Putih	8	0	1
Dendrobium	0	9	1
Ekor Tupai	2	1	8

Hasil **tabel 6** membuktikan bahwa hasil klasifikasi untuk data *testing* untuk data *training* 90 hasilnya sama dengan data *testing* untuk data *training* 140, dengan akurasi 84%, presisi 89%, dan recall 80%.

Tingkat akurasi 84% artinya kinerja sistem terhadap metode yang digunakan mampu mengerjakan dengan tingkat persentase sebesar 84% dari 100%. Sistem mampu mengklasifikasikan dengan tepat data citra dengan baik sebesar 84%, kemudian sisanya artinya 16% sistem tidak mampu mengklasifikasikan dengan tepat.

BAB VI

KESIMPULAN DAN SARAN

6.1. Kesimpulan

Berdasarkan penelitian dan hasil pengujian metode *Convolutional Neural Network* untuk klasifikasi tanaman bunga anggrek bulan putih, anggrek dendrobium, dan anggrek ekor tupai, maka dapat diperoleh kesimpulan sebagai berikut :

1. Hasil pendeteksian yang didapatkan dari klasifikasi menggunakan CNN pada data *testing* yaitu, bunga anggrek bulan putih dapat terklasifikasi dengan tepat dengan prediksi sebanyak 8 obyek citra, untuk bunga anggrek dendrobium terklasifikasi sebanyak 9 obyek citra, dan bunga ekor tupai terklasifikasi sebanyak 8 obyek citra.
2. Hasil klasifikasi yang didapatkan mempunyai tingkat akurasi yang baik yaitu sebesar 84% untuk tingkat akurasi, presisi 89% , dan recall 80%.
3. Untuk data *training* yang dirubah menjadi 90 obyek citra untuk dianalisis dan data *testing* tetap sama akurasinya dengan data *training* yang sebelumnya yaitu tingkat akurasinya sebesar 83%. Artinya, tidak ada perubahan.

6.2. Saran

Saran yang dapat diberikan untuk penelitian selanjutnya, khususnya yang memanfaatkan metode *Convolutional Neural Network* (CNN) adalah sebagai berikut :

1. Analisis berikutnya bisa dilakukan dengan menggunakan data *training* yang lebih banyak agar dapat menaikkan nilai akurasi.
2. Alangkah baiknya jika dilakukan penyortiran citra yang akan digunakan untuk konvolusi, proporsi citra yang diperhatikan dimana dilihat dengan kualitas masing-masing gambar, semakin detil proporsi

citra atau semakin baik kualitas citra yang digunakan maka dapat meningkatkan hasil akurasi untuk klasifikasi.

3. *Input* citra dapat dilakukan dengan ukuran dimensi yang lebih besar dari 32 x 32 piksel sehingga nantinya *hidden layer* yang digunakan akan semakin banyak dan akan meningkatkan kemampuan klasifikasi lebih baik lagi karena semakin banyak *hidden layer* yang dibuat maka akan semakin baik klasifikasi yang dilakukan.

DAFTAR PUSTAKA

- Abdillah, F. (2016). Penggunaan Deep Learning untuk Prediksi Churn pada Jaringan Telekomunikasi Mobile. *e-Proceeding of Engineering: Vol.3 No.2*, 3882-3885.
- Danukusumo, K. P. (2017). *Implementasi Deep Learning Menggunakan Convolutional Neural Network untuk Klasifikasi Citra Candi Berbasis GPU*. Yogyakarta: Fakultas Teknologi Industri Universitas Atma Jaya.
- Felzenszwalb, P. F. (2009). Object Detection with Discriminatively Trained Part Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627-1645.
- Gonzales, Rafael, C., & Woods, R. E. (2002). *Digital Image Processing*. New Jersey: Prentice-Hall, Inc.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning (Adaptive Computation and Machine Learning Series)*. The MIT Press ISBN:0262035618 9780262035613.
- Jain, A. (1989). *Fundamentals of Digital Image Processing*. Prentice-Hall International.
- Jain, R., Kasturi, R., & Schunck, B. G. (1995). Object Recognition. *Machine Vision*, 459-491.
- Jain, R., Kasturi, R., & Schunk, B. (1995). Object Recognition. *Machine Vision*, 459-491.
- Latharani T.R., M. K. (2011). Various Object Recognition Techniques For Computer Vision. *Journal of Analysis and Computation Vol.7, No.1*, 39-47.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015, Mei 27). *Deep Learning*. Retrieved from Nature International Journal of Science: doi:10.1038/nature14539
- Lee, C.-Y., Gallagher, P. W., & Tu, Z. (2015). Generalizing Pooling Functions in Convolutional Neural Networks : Mixed, Gated, and Tree. *Neural and Evolutionary Computing*, arXiv:1509.08985v2 [stat.ML].
- Lu, Y. (2016). Food Image Recognition by Using Convolutional Neural Networks (CNNs). *Computer Vision and Pattern Recognition*, 1-4.

- MathWorks. (2018, Februari 15). Retrieved from <https://www.mathworks.com/help/images/ref/graycoprops.html?searchHighlight=>
- Mayadewi, P., & Rosely, E. (2015). Prediksi Nilai Proyek Akhir Mahasiswa Menggunakan Algoritma Klasifikasi Data Mining. *Seminar Nasional Sistem Informasi Indonesia*, 331-332.
- Mulyana, S. B. (2014). *Identifikasi Jenis Bunga Anggrek Menggunakan Pengolahan Citra Digital dengan Metode KNN*. Bandung: Universitas Telkom .
- Munir, R. (2004). *Pengolahan Citra Digital dengan Pendekatan Algoritmik*. Bandung: Informatika Bandung.
- Riesenhuber, M., & Poggio, T. (1990). Models of Object Recognition. *Nature Neuroscience*, 3:1199-1204. Retrieved from Nature Neuroscience.
- Rismiyati. (2016). *Implementasi Convolutional Neural Network untuk Sortasi Mutu Salak Ekspor Berbasis Citra Digital*. Yogyakarta: Universitas Gadjah Mada.
- Ruder, S. (2017). An Overview of Gradient Descent Optimization Algorithms. *Learning Computer Science*, arXiv:1609.04747.
- Sianipar, R., Mangiri, H., & Wiryajati, L. (2013). *MATLAB untuk Pemrosesan Citra Digital*.
- Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *Computer Vision and Pattern Recognition*, p.2-3 <https://arxiv.org/abs/1409.1556v6>.
- Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2015). Striving for Simplicity: The All Convolutional Net. *Computer Science*, arXiv:1412.6806.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 1929-1968.
- Suzanna, M. (2015, Februari 18). *Anggrek Ekor Tupai (Rhynchostylis violacea)*. Retrieved from Pusat Konservasi Tumbuhan Kebun Raya Bogor: <http://www.krbogor.lipi.go.id/id/Anggrek-Ekor-Tupai-Rhynchostylis-violacea.html>
- Szeliski, R. (2011). *Computer Vision : Algorithms and Applications Computers*. Washington USA: <https://doi.org/10.1007/978>.

Wikipedia. (2017, November 28). *Mawar*. Retrieved from Wikipedia:
<https://id.wikipedia.org/wiki/Mawar>

Wikipedia. (2017, April 27). *Orchidaceae*. Retrieved from Wikipedia:
<https://id.wikipedia.org/wiki/Orchidaceae>

Wikipedia. (2017, Oktober 4). *R (Bahasa Pemrograman)*. Retrieved from
Wikipedia: [https://id.wikipedia.org/wiki/R_\(bahasa_pemrograman\)](https://id.wikipedia.org/wiki/R_(bahasa_pemrograman))

Wikipedia. (2018, Januaro 18). *Anggrek Bulan*. Retrieved from Wikipedia :
https://id.wikipedia.org/wiki/Anggrek_bulan

Wikipedia. (2018, Januari 18). *Dendrobium*. Retrieved from Wikipedia:
<https://id.wikipedia.org/wiki/Dendrobium>

LAMPIRAN

Lampiran 1 Obyek Citra Tanaman Bunga Anggrek 32 piksel x 32 piksel

Data Training



Data Testing



Lampiran 2 *Syntax* untuk Analisis CNN

```
#Convolutional Neural Networks

#Load Packages
library(keras)
library(EBImage)

#Read Images
setwd('D://SKRIPSI KIA')
pic1<-
c('bp1.jpg','bp2.jpg','bp3.jpg','bp4.jpg','bp5.jpg','bp6.jpg',
  ', 'bp7.jpg','bp8.jpg','bp9.jpg','bp10.jpg',

  'bp11.jpg','bp12.jpg','bp13.jpg','bp14.jpg','bp15.jpg','bp16
  .jpg','bp17.jpg','bp18.jpg','bp19.jpg','bp20.jpg',

  'bp21.jpg','bp22.jpg','bp23.jpg','bp24.jpg','bp25.jpg','bp26
  .jpg','bp27.jpg','bp28.jpg','bp29.jpg','bp30.jpg',

  'bp51.jpg','bp53.jpg','bp55.jpg','bp56.jpg','bp57.jpg','bp59
  .jpg','bp61.jpg','bp64.jpg','bp65.jpg','bp67.jpg',

  'bp69.jpg','bp71.jpg','bp73.jpg','bp74.jpg','bp76.jpg','bp80
  .jpg','bp84.jpg','bp86.jpg','bp87.jpg','bp91.jpg',

  'd1.jpg','d2.jpg','d3.jpg','d4.jpg','d5.jpg','d6.jpg','d7.jp
  g','d8.jpg','d9.jpg','d10.jpg',

  'd11.jpg','d12.jpg','d13.jpg','d14.jpg','d15.jpg','d16.jpg',
  'd17.jpg','d18.jpg','d19.jpg','d20.jpg',

  'd21.jpg','d22.jpg','d23.jpg','d24.jpg','d25.jpg','d26.jpg',
  'd27.jpg','d28.jpg','d29.jpg','d30.jpg',
```

```

'd31.jpg', 'd35.jpg', 'd36.jpg', 'd37.jpg', 'd39.jpg', 'd40.jpg',
'd41.jpg', 'd43.jpg', 'd47.jpg', 'd50.jpg',

      'd92.jpg', 'd97.jpg', 'd99.jpg',

'et1.jpg', 'et2.jpg', 'et3.jpg', 'et4.jpg', 'et5.jpg', 'et6.jpg',
'et7.jpg', 'et8.jpg', 'et9.jpg', 'et10.jpg',

'et11.jpg', 'et12.jpg', 'et13.jpg', 'et14.jpg', 'et15.jpg', 'et16
.jpg', 'et17.jpg', 'et18.jpg', 'et19.jpg', 'et20.jpg',

'et21.jpg', 'et22.jpg', 'et23.jpg', 'et24.jpg', 'et25.jpg', 'et26
.jpg', 'et27.jpg', 'et28.jpg', 'et29.jpg', 'et30.jpg',

'et55.jpg', 'et56.jpg', 'et57.jpg', 'et58.jpg', 'et77.jpg', 'et81
.jpg', 'et83.jpg', 'et80.jpg', 'et84.jpg', 'et85.jpg',

'et86.jpg', 'et87.jpg', 'et88.jpg', 'et89.jpg', 'et90.jpg', 'et93
.jpg', 'et94.jpg')

train<-list()

for(i in 1:140){train[[i]]<-readImage(pic1[i])}

pic2<-
c('bp41.jpg', 'bp42.jpg', 'bp43.jpg', 'bp44.jpg', 'bp45.jpg', 'bp
46.jpg', 'bp47.jpg', 'bp48.jpg', 'bp49.jpg', 'bp50.jpg',

'd41.jpg', 'd42.jpg', 'd43.jpg', 'd44.jpg', 'd45.jpg', 'd46.jpg',
'd47.jpg', 'd48.jpg', 'd49.jpg', 'd50.jpg',

'et41.jpg', 'et42.jpg', 'et43.jpg', 'et44.jpg', 'et45.jpg', 'et46
.jpg', 'et47.jpg', 'et48.jpg', 'et49.jpg', 'et50.jpg')

test<-list()

for(i in 1:30){test[[i]]<-readImage(pic2[i])}

print(train[[1]])

hist(train[[1]])

```



```

#Explore
summary(train[[1]])
display(train[[1]])
plot(train[[1]])
display(train[[1]])
hist(train[[1]])

par(mfrow=c(14,10))
for(i in 1:140) plot(train[[i]])
par(mfrow=c(1,1))

#Resize&Combine
str(train)
for(i in 1:140){train[[i]]<-resize(train[[i]],32,32)}
str(test)
for(i in 1:30){test[[i]]<-resize(test[[i]],32,32)}
train<-combine(train)
x<-tile(train, 10)
display(x, title='Pictures')

test<-combine(test)
y<-tile(test, 10)
display(y, title="Pics")
str(train)

#Reorder dimension
train<-aperm(train, c(4, 1, 2, 3))
test<-aperm(test, c(4, 1, 2, 3))
str(train)

```

```

str(test)

#Response
trainy<-c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
          1, 1, 1,
          2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
          2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
          2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
          2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
          2, 2, 2, 2, 2, 2, 2, 2, 2, 2)

testy<-c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         2, 2, 2, 2, 2, 2, 2, 2, 2, 2)

#One hot encoding
trainLabels<-to_categorical(trainy)
testLabels<-to_categorical(testy)

trainLabels
testLabels

#Model
model<-keras_model_sequential()

```

```

model %>%
  layer_conv_2d(filters = 32,
                kernel_size = c(3,3),
                activation = 'relu',
                input_shape = c(32,32,3)) %>%
  layer_conv_2d(filters = 32,
                kernel_size = c(3,3),
                activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_dropout(rate=0.25) %>%
  layer_conv_2d(filters=64,
                kernel_size = c(3,3),
                activation = 'relu') %>%
  layer_conv_2d(filters = 64,
                kernel_size = c(3,3),
                activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_dropout(rate=0.25)%>%
  layer_flatten() %>%
  layer_dense(units=128, activation = 'relu') %>%
  layer_dropout(rate=0.25) %>%
  layer_dense(units=3,activation = 'softmax') %>%

compile(loss='categorical_crossentropy',
        optimizer=optimizer_sgd(lr=0.001,
                                decay = 1e-6,
                                momentum = 0.9,
                                nesterov = T),
        metrics = c('accuracy'))

```

```

summary(model)

#Fit Model
history<-model %>%
  fit(train,
       trainLabels,
       epochs = 500,
       batch_size = 200,
       validation_split = 0.2)
plot(history)
dev.off()

#Evaluation&Prediction-train data
model %>% evaluate(train, trainLabels)
pred<-model %>% predict_classes(train)
table(Predicted = pred, Actual = trainy)

prob<-model %>% predict_proba(train)
cbind(prob, Predicted_class = pred, Actual = trainy)

#Evaluation&Prediction-test data
model %>% evaluate(test, testLabels)
pred<-model %>% predict_classes(test)
table(Predicted = pred, Actual = testy)

prob<-model %>% predict_proba(test)
cbind(prob, Predicted_class = pred, Actual = testy)

#Model (2)
model<-keras_model_sequential()

```

```

model %>%
  layer_conv_2d(filters = 32,
                kernel_size = c(3,3),
                activation = 'relu',
                input_shape = c(32,32,3)) %>%
  layer_conv_2d(filters = 32,
                kernel_size = c(3,3),
                activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_dropout(rate=0.25) %>%
  layer_conv_2d(filters=64,
                kernel_size = c(3,3),
                activation = 'relu') %>%
  layer_conv_2d(filters = 64,
                kernel_size = c(3,3),
                activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_dropout(rate=0.25) %>%
  layer_flatten() %>%
  layer_dense(units=256, activation = 'relu') %>%
  layer_dropout(rate=0.25) %>%
  layer_dense(units=3, activation = 'softmax') %>%

  compile(loss='categorical_crossentropy',
          optimizer=optimizer_sgd(lr=0.001,
                                  decay = 1e-6,
                                  momentum = 0.9,
                                  nesterov = T),
          metrics = c('accuracy'))

summary(model)

```

```
#Fit Model (2)
history<-model%>%
  fit(train,
      trainLabels,
      epochs = 500,
      batch_size = 200,
      validation_split = 0.2,
      validation_data = list(test, testLabels))
plot(history)

#Evaluation&Prediction-train data (2)
model %>% evaluate(train, trainLabels)
pred<-model %>% predict_classes(train)
table(Predicted = pred, Actual = trainy)

prob<-model %>% predict_proba(train)
cbind(prob, Predicted_class = pred, Actual = trainy)

#Evaluation&Prediction-test data (2)
model %>% evaluate(test, testLabels)
pred<-model %>% predict_classes(test)
table(Predicted = pred, Actual = testy)

prob<-model %>% predict_proba(test)
cbind(prob, Predicted_class = pred, Actual = testy)
```

Lampiran 3 Hasil Proses Akurasi dengan *Epochs* 500

```

Epoch 1/500
140/140 [=====] - 5s 37ms/step - loss: 1
.1113 - acc: 0.2857 - val_loss: 1.1013 - val_acc: 0.2667
Epoch 2/500
140/140 [=====] - 2s 13ms/step - loss: 1
.1058 - acc: 0.2929 - val_loss: 1.1012 - val_acc: 0.2667
Epoch 3/500
140/140 [=====] - 2s 14ms/step - loss: 1
.1004 - acc: 0.3357 - val_loss: 1.1010 - val_acc: 0.2667
Epoch 4/500
140/140 [=====] - 2s 13ms/step - loss: 1
.1147 - acc: 0.3071 - val_loss: 1.1007 - val_acc: 0.2667
Epoch 5/500
140/140 [=====] - 2s 14ms/step - loss: 1
.1041 - acc: 0.3214 - val_loss: 1.1005 - val_acc: 0.2667
Epoch 6/500
140/140 [=====] - 2s 14ms/step - loss: 1
.1117 - acc: 0.3071 - val_loss: 1.1003 - val_acc: 0.2667
Epoch 7/500
140/140 [=====] - 2s 14ms/step - loss: 1
.1058 - acc: 0.3429 - val_loss: 1.1001 - val_acc: 0.2333
Epoch 8/500
140/140 [=====] - 2s 15ms/step - loss: 1
.0947 - acc: 0.3643 - val_loss: 1.0999 - val_acc: 0.3000
Epoch 9/500
140/140 [=====] - 2s 14ms/step - loss: 1
.1106 - acc: 0.2929 - val_loss: 1.0997 - val_acc: 0.3000
Epoch 10/500
140/140 [=====] - 2s 14ms/step - loss: 1
.1051 - acc: 0.3357 - val_loss: 1.0995 - val_acc: 0.3333
Epoch 11/500
140/140 [=====] - 2s 14ms/step - loss: 1
.1154 - acc: 0.2929 - val_loss: 1.0993 - val_acc: 0.3333
Epoch 12/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0970 - acc: 0.3500 - val_loss: 1.0991 - val_acc: 0.3000
Epoch 13/500
140/140 [=====] - 2s 13ms/step - loss: 1
.1038 - acc: 0.3571 - val_loss: 1.0990 - val_acc: 0.4000
Epoch 14/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0948 - acc: 0.3500 - val_loss: 1.0989 - val_acc: 0.4000
Epoch 15/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0897 - acc: 0.3929 - val_loss: 1.0987 - val_acc: 0.3333
Epoch 16/500
140/140 [=====] - 2s 14ms/step - loss: 1
.1033 - acc: 0.3500 - val_loss: 1.0986 - val_acc: 0.3333
Epoch 17/500
140/140 [=====] - 2s 13ms/step - loss: 1
.1021 - acc: 0.3571 - val_loss: 1.0985 - val_acc: 0.3333
Epoch 18/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0948 - acc: 0.3429 - val_loss: 1.0984 - val_acc: 0.3333
Epoch 19/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0947 - acc: 0.3429 - val_loss: 1.0983 - val_acc: 0.3333
Epoch 20/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0982 - acc: 0.3357 - val_loss: 1.0982 - val_acc: 0.3667
Epoch 21/500

```

```

140/140 [=====] - 2s 13ms/step - loss: 1
.0917 - acc: 0.3500 - val_loss: 1.0981 - val_acc: 0.3333
Epoch 22/500
140/140 [=====] - 2s 13ms/step - loss: 1
.1012 - acc: 0.3286 - val_loss: 1.0981 - val_acc: 0.3333
Epoch 23/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0986 - acc: 0.3357 - val_loss: 1.0980 - val_acc: 0.3333
Epoch 24/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0933 - acc: 0.3643 - val_loss: 1.0979 - val_acc: 0.3333
Epoch 25/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0989 - acc: 0.3143 - val_loss: 1.0978 - val_acc: 0.3333
Epoch 26/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0959 - acc: 0.3643 - val_loss: 1.0978 - val_acc: 0.3333
Epoch 27/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0869 - acc: 0.3929 - val_loss: 1.0977 - val_acc: 0.3333
Epoch 28/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0870 - acc: 0.4143 - val_loss: 1.0977 - val_acc: 0.3333
Epoch 29/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0980 - acc: 0.3571 - val_loss: 1.0976 - val_acc: 0.3333
Epoch 30/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0954 - acc: 0.3143 - val_loss: 1.0976 - val_acc: 0.3333
Epoch 31/500
140/140 [=====] - 2s 13ms/step - loss: 1
.1017 - acc: 0.3500 - val_loss: 1.0975 - val_acc: 0.3333
Epoch 32/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0983 - acc: 0.3643 - val_loss: 1.0975 - val_acc: 0.3333
Epoch 33/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0931 - acc: 0.3857 - val_loss: 1.0974 - val_acc: 0.3333
Epoch 34/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0982 - acc: 0.3643 - val_loss: 1.0973 - val_acc: 0.3333
Epoch 35/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0882 - acc: 0.3643 - val_loss: 1.0972 - val_acc: 0.3333
Epoch 36/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0901 - acc: 0.4143 - val_loss: 1.0971 - val_acc: 0.3333
Epoch 37/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0909 - acc: 0.3786 - val_loss: 1.0970 - val_acc: 0.3333
Epoch 38/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0891 - acc: 0.4357 - val_loss: 1.0969 - val_acc: 0.3333
Epoch 39/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0984 - acc: 0.3786 - val_loss: 1.0968 - val_acc: 0.3333
Epoch 40/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0932 - acc: 0.3857 - val_loss: 1.0967 - val_acc: 0.3333
Epoch 41/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0952 - acc: 0.3571 - val_loss: 1.0965 - val_acc: 0.3333
Epoch 42/500

```



```

140/140 [=====] - 2s 13ms/step - loss: 1
.0931 - acc: 0.3714 - val_loss: 1.0964 - val_acc: 0.3333
Epoch 43/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0944 - acc: 0.3786 - val_loss: 1.0963 - val_acc: 0.3333
Epoch 44/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0849 - acc: 0.4500 - val_loss: 1.0961 - val_acc: 0.3333
Epoch 45/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0963 - acc: 0.3286 - val_loss: 1.0960 - val_acc: 0.3333
Epoch 46/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0931 - acc: 0.3786 - val_loss: 1.0958 - val_acc: 0.3333
Epoch 47/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0886 - acc: 0.4143 - val_loss: 1.0956 - val_acc: 0.3333
Epoch 48/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0955 - acc: 0.3571 - val_loss: 1.0954 - val_acc: 0.3333
Epoch 49/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0812 - acc: 0.3357 - val_loss: 1.0953 - val_acc: 0.3333
Epoch 50/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0906 - acc: 0.3500 - val_loss: 1.0951 - val_acc: 0.3333
Epoch 51/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0910 - acc: 0.3714 - val_loss: 1.0949 - val_acc: 0.3333
Epoch 52/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0860 - acc: 0.3929 - val_loss: 1.0947 - val_acc: 0.3333
Epoch 53/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0907 - acc: 0.3929 - val_loss: 1.0946 - val_acc: 0.3333
Epoch 54/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0945 - acc: 0.4000 - val_loss: 1.0944 - val_acc: 0.3333
Epoch 55/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0975 - acc: 0.3643 - val_loss: 1.0942 - val_acc: 0.3333
Epoch 56/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0893 - acc: 0.3714 - val_loss: 1.0940 - val_acc: 0.3333
Epoch 57/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0894 - acc: 0.3929 - val_loss: 1.0938 - val_acc: 0.3333
Epoch 58/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0839 - acc: 0.4214 - val_loss: 1.0936 - val_acc: 0.3333
Epoch 59/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0824 - acc: 0.4071 - val_loss: 1.0934 - val_acc: 0.3333
Epoch 60/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0842 - acc: 0.3429 - val_loss: 1.0931 - val_acc: 0.3333
Epoch 61/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0883 - acc: 0.4000 - val_loss: 1.0929 - val_acc: 0.3333
Epoch 62/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0862 - acc: 0.4357 - val_loss: 1.0927 - val_acc: 0.3333
Epoch 63/500

```

```

140/140 [=====] - 2s 13ms/step - loss: 1
.0848 - acc: 0.3857 - val_loss: 1.0925 - val_acc: 0.3333
Epoch 64/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0861 - acc: 0.4000 - val_loss: 1.0923 - val_acc: 0.3333
Epoch 65/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0842 - acc: 0.4000 - val_loss: 1.0921 - val_acc: 0.3333
Epoch 66/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0767 - acc: 0.4429 - val_loss: 1.0919 - val_acc: 0.3333
Epoch 67/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0957 - acc: 0.3643 - val_loss: 1.0916 - val_acc: 0.3333
Epoch 68/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0844 - acc: 0.4357 - val_loss: 1.0914 - val_acc: 0.3333
Epoch 69/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0863 - acc: 0.3786 - val_loss: 1.0912 - val_acc: 0.3333
Epoch 70/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0772 - acc: 0.4214 - val_loss: 1.0910 - val_acc: 0.3333
Epoch 71/500
140/140 [=====] - 2s 15ms/step - loss: 1
.0787 - acc: 0.4000 - val_loss: 1.0908 - val_acc: 0.3333
Epoch 72/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0839 - acc: 0.4071 - val_loss: 1.0905 - val_acc: 0.3333
Epoch 73/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0999 - acc: 0.3500 - val_loss: 1.0903 - val_acc: 0.3333
Epoch 74/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0845 - acc: 0.4429 - val_loss: 1.0901 - val_acc: 0.3333
Epoch 75/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0847 - acc: 0.4214 - val_loss: 1.0898 - val_acc: 0.3333
Epoch 76/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0863 - acc: 0.3857 - val_loss: 1.0895 - val_acc: 0.3333
Epoch 77/500
140/140 [=====] - 3s 20ms/step - loss: 1
.0895 - acc: 0.3571 - val_loss: 1.0892 - val_acc: 0.3333
Epoch 78/500
140/140 [=====] - 3s 21ms/step - loss: 1
.0950 - acc: 0.3714 - val_loss: 1.0889 - val_acc: 0.3333
Epoch 79/500
140/140 [=====] - 3s 21ms/step - loss: 1
.0833 - acc: 0.4214 - val_loss: 1.0886 - val_acc: 0.3333
Epoch 80/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0837 - acc: 0.3857 - val_loss: 1.0882 - val_acc: 0.3333
Epoch 81/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0759 - acc: 0.4071 - val_loss: 1.0879 - val_acc: 0.3333
Epoch 82/500
140/140 [=====] - 2s 16ms/step - loss: 1
.0734 - acc: 0.4286 - val_loss: 1.0876 - val_acc: 0.3333
Epoch 83/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0905 - acc: 0.3929 - val_loss: 1.0873 - val_acc: 0.3333
Epoch 84/500

```

```

140/140 [=====] - 2s 13ms/step - loss: 1
.0743 - acc: 0.4286 - val_loss: 1.0870 - val_acc: 0.3333
Epoch 85/500
140/140 [=====] - 3s 21ms/step - loss: 1
.0918 - acc: 0.3643 - val_loss: 1.0867 - val_acc: 0.3333
Epoch 86/500
140/140 [=====] - 2s 15ms/step - loss: 1
.0771 - acc: 0.4500 - val_loss: 1.0864 - val_acc: 0.3333
Epoch 87/500
140/140 [=====] - 2s 15ms/step - loss: 1
.0709 - acc: 0.4286 - val_loss: 1.0861 - val_acc: 0.3333
Epoch 88/500
140/140 [=====] - 2s 16ms/step - loss: 1
.0733 - acc: 0.3857 - val_loss: 1.0858 - val_acc: 0.3333
Epoch 89/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0853 - acc: 0.3643 - val_loss: 1.0856 - val_acc: 0.3333
Epoch 90/500
140/140 [=====] - 2s 17ms/step - loss: 1
.0840 - acc: 0.3929 - val_loss: 1.0853 - val_acc: 0.3333
Epoch 91/500
140/140 [=====] - 2s 12ms/step - loss: 1
.0807 - acc: 0.4286 - val_loss: 1.0850 - val_acc: 0.3333
Epoch 92/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0886 - acc: 0.3857 - val_loss: 1.0847 - val_acc: 0.3333
Epoch 93/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0829 - acc: 0.3929 - val_loss: 1.0844 - val_acc: 0.3333
Epoch 94/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0788 - acc: 0.4286 - val_loss: 1.0840 - val_acc: 0.3333
Epoch 95/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0733 - acc: 0.4286 - val_loss: 1.0837 - val_acc: 0.3333
Epoch 96/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0808 - acc: 0.4071 - val_loss: 1.0833 - val_acc: 0.3333
Epoch 97/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0753 - acc: 0.4214 - val_loss: 1.0830 - val_acc: 0.3333
Epoch 98/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0670 - acc: 0.4357 - val_loss: 1.0825 - val_acc: 0.3333
Epoch 99/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0869 - acc: 0.3857 - val_loss: 1.0822 - val_acc: 0.3333
Epoch 100/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0726 - acc: 0.4000 - val_loss: 1.0818 - val_acc: 0.3333
Epoch 101/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0750 - acc: 0.4000 - val_loss: 1.0814 - val_acc: 0.3333
Epoch 102/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0728 - acc: 0.4214 - val_loss: 1.0810 - val_acc: 0.3333
Epoch 103/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0744 - acc: 0.4429 - val_loss: 1.0806 - val_acc: 0.3333
Epoch 104/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0797 - acc: 0.4214 - val_loss: 1.0802 - val_acc: 0.3333
Epoch 105/500

```

```

140/140 [=====] - 2s 13ms/step - loss: 1
.0784 - acc: 0.4214 - val_loss: 1.0797 - val_acc: 0.3333
Epoch 106/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0604 - acc: 0.4357 - val_loss: 1.0793 - val_acc: 0.3333
Epoch 107/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0843 - acc: 0.4000 - val_loss: 1.0788 - val_acc: 0.3333
Epoch 108/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0720 - acc: 0.4000 - val_loss: 1.0784 - val_acc: 0.3333
Epoch 109/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0633 - acc: 0.4643 - val_loss: 1.0779 - val_acc: 0.3333
Epoch 110/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0789 - acc: 0.4286 - val_loss: 1.0773 - val_acc: 0.3333
Epoch 111/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0664 - acc: 0.4786 - val_loss: 1.0768 - val_acc: 0.3333
Epoch 112/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0781 - acc: 0.4071 - val_loss: 1.0762 - val_acc: 0.3333
Epoch 113/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0784 - acc: 0.3714 - val_loss: 1.0757 - val_acc: 0.3333
Epoch 114/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0667 - acc: 0.4357 - val_loss: 1.0752 - val_acc: 0.3333
Epoch 115/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0743 - acc: 0.4071 - val_loss: 1.0746 - val_acc: 0.3333
Epoch 116/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0630 - acc: 0.4571 - val_loss: 1.0740 - val_acc: 0.3333
Epoch 117/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0686 - acc: 0.4429 - val_loss: 1.0734 - val_acc: 0.3333
Epoch 118/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0716 - acc: 0.4429 - val_loss: 1.0729 - val_acc: 0.3333
Epoch 119/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0577 - acc: 0.4571 - val_loss: 1.0723 - val_acc: 0.3333
Epoch 120/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0963 - acc: 0.3500 - val_loss: 1.0717 - val_acc: 0.3333
Epoch 121/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0627 - acc: 0.4857 - val_loss: 1.0711 - val_acc: 0.3333
Epoch 122/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0650 - acc: 0.4357 - val_loss: 1.0705 - val_acc: 0.3333
Epoch 123/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0641 - acc: 0.4929 - val_loss: 1.0699 - val_acc: 0.3333
Epoch 124/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0641 - acc: 0.4143 - val_loss: 1.0692 - val_acc: 0.3667
Epoch 125/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0626 - acc: 0.4786 - val_loss: 1.0686 - val_acc: 0.3667
Epoch 126/500

```

```

140/140 [=====] - 2s 13ms/step - loss: 1
.0619 - acc: 0.4643 - val_loss: 1.0678 - val_acc: 0.3667
Epoch 127/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0609 - acc: 0.4571 - val_loss: 1.0671 - val_acc: 0.3667
Epoch 128/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0620 - acc: 0.4714 - val_loss: 1.0664 - val_acc: 0.3667
Epoch 129/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0769 - acc: 0.4286 - val_loss: 1.0657 - val_acc: 0.3667
Epoch 130/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0611 - acc: 0.4357 - val_loss: 1.0649 - val_acc: 0.4000
Epoch 131/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0559 - acc: 0.4786 - val_loss: 1.0642 - val_acc: 0.4333
Epoch 132/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0533 - acc: 0.4714 - val_loss: 1.0634 - val_acc: 0.4333
Epoch 133/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0520 - acc: 0.4571 - val_loss: 1.0625 - val_acc: 0.4333
Epoch 134/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0641 - acc: 0.4643 - val_loss: 1.0618 - val_acc: 0.4333
Epoch 135/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0610 - acc: 0.4571 - val_loss: 1.0609 - val_acc: 0.4333
Epoch 136/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0523 - acc: 0.4571 - val_loss: 1.0600 - val_acc: 0.4333
Epoch 137/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0541 - acc: 0.4714 - val_loss: 1.0591 - val_acc: 0.4333
Epoch 138/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0537 - acc: 0.4786 - val_loss: 1.0582 - val_acc: 0.4333
Epoch 139/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0536 - acc: 0.4643 - val_loss: 1.0573 - val_acc: 0.4333
Epoch 140/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0503 - acc: 0.4643 - val_loss: 1.0564 - val_acc: 0.4333
Epoch 141/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0442 - acc: 0.4786 - val_loss: 1.0555 - val_acc: 0.4333
Epoch 142/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0452 - acc: 0.5286 - val_loss: 1.0546 - val_acc: 0.4333
Epoch 143/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0484 - acc: 0.4571 - val_loss: 1.0537 - val_acc: 0.4333
Epoch 144/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0453 - acc: 0.5000 - val_loss: 1.0528 - val_acc: 0.4333
Epoch 145/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0526 - acc: 0.4357 - val_loss: 1.0518 - val_acc: 0.4333
Epoch 146/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0582 - acc: 0.4071 - val_loss: 1.0509 - val_acc: 0.4333
Epoch 147/500

```

```

140/140 [=====] - 2s 13ms/step - loss: 1
.0473 - acc: 0.4357 - val_loss: 1.0499 - val_acc: 0.4333
Epoch 148/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0513 - acc: 0.4714 - val_loss: 1.0489 - val_acc: 0.4333
Epoch 149/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0403 - acc: 0.4929 - val_loss: 1.0479 - val_acc: 0.4333
Epoch 150/500
140/140 [=====] - 2s 12ms/step - loss: 1
.0459 - acc: 0.4643 - val_loss: 1.0468 - val_acc: 0.4333
Epoch 151/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0403 - acc: 0.4857 - val_loss: 1.0457 - val_acc: 0.4333
Epoch 152/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0367 - acc: 0.5000 - val_loss: 1.0445 - val_acc: 0.4333
Epoch 153/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0441 - acc: 0.4714 - val_loss: 1.0434 - val_acc: 0.4333
Epoch 154/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0351 - acc: 0.4643 - val_loss: 1.0422 - val_acc: 0.4333
Epoch 155/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0454 - acc: 0.4857 - val_loss: 1.0409 - val_acc: 0.4333
Epoch 156/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0467 - acc: 0.4500 - val_loss: 1.0397 - val_acc: 0.4667
Epoch 157/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0404 - acc: 0.5000 - val_loss: 1.0384 - val_acc: 0.4667
Epoch 158/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0358 - acc: 0.5071 - val_loss: 1.0371 - val_acc: 0.4667
Epoch 159/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0193 - acc: 0.5429 - val_loss: 1.0358 - val_acc: 0.4667
Epoch 160/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0320 - acc: 0.5214 - val_loss: 1.0344 - val_acc: 0.4667
Epoch 161/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0369 - acc: 0.5357 - val_loss: 1.0331 - val_acc: 0.4667
Epoch 162/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0167 - acc: 0.5000 - val_loss: 1.0317 - val_acc: 0.4667
Epoch 163/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0364 - acc: 0.5071 - val_loss: 1.0303 - val_acc: 0.4667
Epoch 164/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0162 - acc: 0.5429 - val_loss: 1.0288 - val_acc: 0.4667
Epoch 165/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0286 - acc: 0.5357 - val_loss: 1.0274 - val_acc: 0.4667
Epoch 166/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0203 - acc: 0.4929 - val_loss: 1.0259 - val_acc: 0.5000
Epoch 167/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0239 - acc: 0.5714 - val_loss: 1.0244 - val_acc: 0.5000
Epoch 168/500

```

```

140/140 [=====] - 2s 13ms/step - loss: 1
.0349 - acc: 0.5071 - val_loss: 1.0229 - val_acc: 0.5333
Epoch 169/500
140/140 [=====] - 2s 15ms/step - loss: 1
.0108 - acc: 0.5357 - val_loss: 1.0213 - val_acc: 0.5333
Epoch 170/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0140 - acc: 0.4929 - val_loss: 1.0197 - val_acc: 0.5333
Epoch 171/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0154 - acc: 0.5286 - val_loss: 1.0179 - val_acc: 0.5333
Epoch 172/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0275 - acc: 0.5286 - val_loss: 1.0163 - val_acc: 0.5333
Epoch 173/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0205 - acc: 0.5571 - val_loss: 1.0145 - val_acc: 0.5333
Epoch 174/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9980 - acc: 0.5500 - val_loss: 1.0127 - val_acc: 0.5333
Epoch 175/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0107 - acc: 0.5357 - val_loss: 1.0108 - val_acc: 0.5333
Epoch 176/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0203 - acc: 0.5000 - val_loss: 1.0089 - val_acc: 0.5333
Epoch 177/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0093 - acc: 0.5500 - val_loss: 1.0069 - val_acc: 0.5667
Epoch 178/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0142 - acc: 0.5143 - val_loss: 1.0049 - val_acc: 0.5667
Epoch 179/500
140/140 [=====] - 2s 13ms/step - loss: 1
.0053 - acc: 0.5214 - val_loss: 1.0029 - val_acc: 0.5667
Epoch 180/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0005 - acc: 0.5214 - val_loss: 1.0008 - val_acc: 0.5667
Epoch 181/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0068 - acc: 0.5643 - val_loss: 0.9987 - val_acc: 0.5667
Epoch 182/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0014 - acc: 0.5643 - val_loss: 0.9965 - val_acc: 0.5667
Epoch 183/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9927 - acc: 0.5286 - val_loss: 0.9944 - val_acc: 0.5667
Epoch 184/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9999 - acc: 0.5714 - val_loss: 0.9922 - val_acc: 0.5667
Epoch 185/500
140/140 [=====] - 2s 13ms/step - loss: 0
.9997 - acc: 0.5000 - val_loss: 0.9899 - val_acc: 0.5667
Epoch 186/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9916 - acc: 0.5643 - val_loss: 0.9875 - val_acc: 0.5667
Epoch 187/500
140/140 [=====] - 2s 14ms/step - loss: 1
.0032 - acc: 0.5286 - val_loss: 0.9850 - val_acc: 0.5667
Epoch 188/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9799 - acc: 0.5571 - val_loss: 0.9825 - val_acc: 0.6333
Epoch 189/500

```

```

140/140 [=====] - 2s 15ms/step - loss: 0
.9826 - acc: 0.5714 - val_loss: 0.9800 - val_acc: 0.6333
Epoch 190/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9851 - acc: 0.5214 - val_loss: 0.9773 - val_acc: 0.6667
Epoch 191/500
140/140 [=====] - 2s 13ms/step - loss: 0
.9837 - acc: 0.5857 - val_loss: 0.9747 - val_acc: 0.6667
Epoch 192/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9747 - acc: 0.5571 - val_loss: 0.9720 - val_acc: 0.7000
Epoch 193/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9877 - acc: 0.5429 - val_loss: 0.9694 - val_acc: 0.7000
Epoch 194/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9685 - acc: 0.5929 - val_loss: 0.9666 - val_acc: 0.7000
Epoch 195/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9710 - acc: 0.5929 - val_loss: 0.9638 - val_acc: 0.7000
Epoch 196/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9750 - acc: 0.5786 - val_loss: 0.9610 - val_acc: 0.7000
Epoch 197/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9747 - acc: 0.5500 - val_loss: 0.9581 - val_acc: 0.7000
Epoch 198/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9664 - acc: 0.5929 - val_loss: 0.9551 - val_acc: 0.7000
Epoch 199/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9605 - acc: 0.5429 - val_loss: 0.9520 - val_acc: 0.7000
Epoch 200/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9509 - acc: 0.5643 - val_loss: 0.9489 - val_acc: 0.7000
Epoch 201/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9558 - acc: 0.5786 - val_loss: 0.9456 - val_acc: 0.7000
Epoch 202/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9605 - acc: 0.5571 - val_loss: 0.9424 - val_acc: 0.7000
Epoch 203/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9354 - acc: 0.6143 - val_loss: 0.9389 - val_acc: 0.7000
Epoch 204/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9482 - acc: 0.5429 - val_loss: 0.9355 - val_acc: 0.7000
Epoch 205/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9494 - acc: 0.5786 - val_loss: 0.9320 - val_acc: 0.7000
Epoch 206/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9390 - acc: 0.6286 - val_loss: 0.9284 - val_acc: 0.7000
Epoch 207/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9346 - acc: 0.6214 - val_loss: 0.9248 - val_acc: 0.7000
Epoch 208/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9363 - acc: 0.6214 - val_loss: 0.9212 - val_acc: 0.7000
Epoch 209/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9468 - acc: 0.5143 - val_loss: 0.9174 - val_acc: 0.7000
Epoch 210/500

```



```

140/140 [=====] - 2s 14ms/step - loss: 0
.9436 - acc: 0.5929 - val_loss: 0.9136 - val_acc: 0.7000
Epoch 211/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9430 - acc: 0.5500 - val_loss: 0.9099 - val_acc: 0.7000
Epoch 212/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9349 - acc: 0.5643 - val_loss: 0.9062 - val_acc: 0.7333
Epoch 213/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9182 - acc: 0.5786 - val_loss: 0.9026 - val_acc: 0.7667
Epoch 214/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9097 - acc: 0.6357 - val_loss: 0.8985 - val_acc: 0.7667
Epoch 215/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9190 - acc: 0.6000 - val_loss: 0.8943 - val_acc: 0.7333
Epoch 216/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9237 - acc: 0.5857 - val_loss: 0.8901 - val_acc: 0.7000
Epoch 217/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9197 - acc: 0.6000 - val_loss: 0.8861 - val_acc: 0.7000
Epoch 218/500
140/140 [=====] - 2s 13ms/step - loss: 0
.9260 - acc: 0.5357 - val_loss: 0.8821 - val_acc: 0.7000
Epoch 219/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9008 - acc: 0.6143 - val_loss: 0.8780 - val_acc: 0.7333
Epoch 220/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9124 - acc: 0.6000 - val_loss: 0.8742 - val_acc: 0.7333
Epoch 221/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9114 - acc: 0.6071 - val_loss: 0.8704 - val_acc: 0.7667
Epoch 222/500
140/140 [=====] - 2s 14ms/step - loss: 0
.9103 - acc: 0.5786 - val_loss: 0.8665 - val_acc: 0.8000
Epoch 223/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8888 - acc: 0.6143 - val_loss: 0.8622 - val_acc: 0.8000
Epoch 224/500
140/140 [=====] - 2s 13ms/step - loss: 0
.8990 - acc: 0.5714 - val_loss: 0.8580 - val_acc: 0.7667
Epoch 225/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8894 - acc: 0.6071 - val_loss: 0.8536 - val_acc: 0.7667
Epoch 226/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8711 - acc: 0.6429 - val_loss: 0.8492 - val_acc: 0.7667
Epoch 227/500
140/140 [=====] - 2s 13ms/step - loss: 0
.8910 - acc: 0.6286 - val_loss: 0.8445 - val_acc: 0.7667
Epoch 228/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8750 - acc: 0.6143 - val_loss: 0.8402 - val_acc: 0.7667
Epoch 229/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8776 - acc: 0.6000 - val_loss: 0.8359 - val_acc: 0.8000
Epoch 230/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8539 - acc: 0.6143 - val_loss: 0.8315 - val_acc: 0.8000
Epoch 231/500

```

```

140/140 [=====] - 2s 14ms/step - loss: 0
.8707 - acc: 0.5929 - val_loss: 0.8277 - val_acc: 0.8333
Epoch 232/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8527 - acc: 0.6000 - val_loss: 0.8248 - val_acc: 0.8333
Epoch 233/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8616 - acc: 0.6071 - val_loss: 0.8210 - val_acc: 0.8000
Epoch 234/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8813 - acc: 0.5857 - val_loss: 0.8172 - val_acc: 0.8000
Epoch 235/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8634 - acc: 0.5857 - val_loss: 0.8120 - val_acc: 0.8000
Epoch 236/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8495 - acc: 0.6214 - val_loss: 0.8070 - val_acc: 0.8000
Epoch 237/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8550 - acc: 0.5929 - val_loss: 0.8014 - val_acc: 0.8000
Epoch 238/500
140/140 [=====] - 2s 13ms/step - loss: 0
.8551 - acc: 0.5714 - val_loss: 0.7964 - val_acc: 0.8000
Epoch 239/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8429 - acc: 0.6500 - val_loss: 0.7914 - val_acc: 0.8333
Epoch 240/500
140/140 [=====] - 2s 13ms/step - loss: 0
.8555 - acc: 0.5786 - val_loss: 0.7868 - val_acc: 0.8333
Epoch 241/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8370 - acc: 0.6286 - val_loss: 0.7825 - val_acc: 0.8333
Epoch 242/500
140/140 [=====] - 2s 13ms/step - loss: 0
.8656 - acc: 0.5571 - val_loss: 0.7787 - val_acc: 0.8000
Epoch 243/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8399 - acc: 0.5929 - val_loss: 0.7756 - val_acc: 0.8000
Epoch 244/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8563 - acc: 0.5786 - val_loss: 0.7722 - val_acc: 0.7667
Epoch 245/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8109 - acc: 0.6357 - val_loss: 0.7690 - val_acc: 0.8000
Epoch 246/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8464 - acc: 0.5643 - val_loss: 0.7640 - val_acc: 0.7667
Epoch 247/500
140/140 [=====] - 2s 15ms/step - loss: 0
.7921 - acc: 0.6786 - val_loss: 0.7599 - val_acc: 0.8000
Epoch 248/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8276 - acc: 0.6357 - val_loss: 0.7562 - val_acc: 0.8000
Epoch 249/500
140/140 [=====] - 2s 13ms/step - loss: 0
.8304 - acc: 0.6143 - val_loss: 0.7509 - val_acc: 0.7667
Epoch 250/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8234 - acc: 0.6643 - val_loss: 0.7465 - val_acc: 0.7667
Epoch 251/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8231 - acc: 0.6357 - val_loss: 0.7427 - val_acc: 0.8000
Epoch 252/500

```

```

140/140 [=====] - 2s 14ms/step - loss: 0
.7908 - acc: 0.6357 - val_loss: 0.7407 - val_acc: 0.7667
Epoch 253/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8180 - acc: 0.5857 - val_loss: 0.7377 - val_acc: 0.7667
Epoch 254/500
140/140 [=====] - 2s 15ms/step - loss: 0
.8053 - acc: 0.5857 - val_loss: 0.7350 - val_acc: 0.7667
Epoch 255/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8162 - acc: 0.6286 - val_loss: 0.7293 - val_acc: 0.7667
Epoch 256/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7759 - acc: 0.6071 - val_loss: 0.7242 - val_acc: 0.7000
Epoch 257/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7614 - acc: 0.6357 - val_loss: 0.7204 - val_acc: 0.7000
Epoch 258/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7846 - acc: 0.6643 - val_loss: 0.7133 - val_acc: 0.8000
Epoch 259/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7779 - acc: 0.6357 - val_loss: 0.7112 - val_acc: 0.7333
Epoch 260/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8282 - acc: 0.6214 - val_loss: 0.7091 - val_acc: 0.7000
Epoch 261/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8196 - acc: 0.6214 - val_loss: 0.7077 - val_acc: 0.7667
Epoch 262/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7730 - acc: 0.6714 - val_loss: 0.7041 - val_acc: 0.7667
Epoch 263/500
140/140 [=====] - 2s 14ms/step - loss: 0
.8055 - acc: 0.6357 - val_loss: 0.6985 - val_acc: 0.7000
Epoch 264/500
140/140 [=====] - 2s 13ms/step - loss: 0
.7864 - acc: 0.6143 - val_loss: 0.6981 - val_acc: 0.7667
Epoch 265/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7732 - acc: 0.6286 - val_loss: 0.6956 - val_acc: 0.7667
Epoch 266/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7859 - acc: 0.6214 - val_loss: 0.6945 - val_acc: 0.7667
Epoch 267/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7691 - acc: 0.6214 - val_loss: 0.6895 - val_acc: 0.7667
Epoch 268/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7946 - acc: 0.6286 - val_loss: 0.6851 - val_acc: 0.7333
Epoch 269/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7458 - acc: 0.6714 - val_loss: 0.6789 - val_acc: 0.7000
Epoch 270/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7842 - acc: 0.6429 - val_loss: 0.6790 - val_acc: 0.7333
Epoch 271/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7424 - acc: 0.6643 - val_loss: 0.6738 - val_acc: 0.7000
Epoch 272/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7682 - acc: 0.6714 - val_loss: 0.6700 - val_acc: 0.7000
Epoch 273/500

```

```

140/140 [=====] - 2s 14ms/step - loss: 0
.7479 - acc: 0.6714 - val_loss: 0.6680 - val_acc: 0.7000
Epoch 274/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7711 - acc: 0.6214 - val_loss: 0.6640 - val_acc: 0.7000
Epoch 275/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7840 - acc: 0.6357 - val_loss: 0.6617 - val_acc: 0.7000
Epoch 276/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7647 - acc: 0.6500 - val_loss: 0.6564 - val_acc: 0.7333
Epoch 277/500
140/140 [=====] - 2s 13ms/step - loss: 0
.7745 - acc: 0.6286 - val_loss: 0.6616 - val_acc: 0.7667
Epoch 278/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7455 - acc: 0.6786 - val_loss: 0.6583 - val_acc: 0.7667
Epoch 279/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7582 - acc: 0.6857 - val_loss: 0.6512 - val_acc: 0.7333
Epoch 280/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7236 - acc: 0.6929 - val_loss: 0.6496 - val_acc: 0.7333
Epoch 281/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7424 - acc: 0.6643 - val_loss: 0.6470 - val_acc: 0.7333
Epoch 282/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7418 - acc: 0.6714 - val_loss: 0.6422 - val_acc: 0.7000
Epoch 283/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7413 - acc: 0.7214 - val_loss: 0.6404 - val_acc: 0.7333
Epoch 284/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7203 - acc: 0.6714 - val_loss: 0.6368 - val_acc: 0.7000
Epoch 285/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7656 - acc: 0.6643 - val_loss: 0.6341 - val_acc: 0.7000
Epoch 286/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7568 - acc: 0.6500 - val_loss: 0.6359 - val_acc: 0.7333
Epoch 287/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7403 - acc: 0.6714 - val_loss: 0.6335 - val_acc: 0.7333
Epoch 288/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7050 - acc: 0.7000 - val_loss: 0.6333 - val_acc: 0.7667
Epoch 289/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7286 - acc: 0.6857 - val_loss: 0.6319 - val_acc: 0.7667
Epoch 290/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7339 - acc: 0.6714 - val_loss: 0.6265 - val_acc: 0.7333
Epoch 291/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7453 - acc: 0.6786 - val_loss: 0.6223 - val_acc: 0.7333
Epoch 292/500
140/140 [=====] - 2s 13ms/step - loss: 0
.7129 - acc: 0.6786 - val_loss: 0.6264 - val_acc: 0.7667
Epoch 293/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7299 - acc: 0.6500 - val_loss: 0.6159 - val_acc: 0.7667
Epoch 294/500

```

```

140/140 [=====] - 2s 14ms/step - loss: 0
.7248 - acc: 0.7000 - val_loss: 0.6169 - val_acc: 0.7333
Epoch 295/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7103 - acc: 0.6929 - val_loss: 0.6191 - val_acc: 0.7667
Epoch 296/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7283 - acc: 0.7143 - val_loss: 0.6128 - val_acc: 0.7333
Epoch 297/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7453 - acc: 0.6500 - val_loss: 0.6091 - val_acc: 0.7667
Epoch 298/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7390 - acc: 0.6357 - val_loss: 0.6134 - val_acc: 0.7667
Epoch 299/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7365 - acc: 0.6429 - val_loss: 0.6109 - val_acc: 0.7667
Epoch 300/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7503 - acc: 0.6714 - val_loss: 0.6080 - val_acc: 0.7667
Epoch 301/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7192 - acc: 0.6500 - val_loss: 0.6049 - val_acc: 0.7667
Epoch 302/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7070 - acc: 0.7286 - val_loss: 0.6013 - val_acc: 0.7667
Epoch 303/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6914 - acc: 0.6571 - val_loss: 0.6030 - val_acc: 0.7667
Epoch 304/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7379 - acc: 0.6571 - val_loss: 0.5997 - val_acc: 0.8000
Epoch 305/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7232 - acc: 0.6643 - val_loss: 0.5972 - val_acc: 0.8000
Epoch 306/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7435 - acc: 0.6429 - val_loss: 0.5966 - val_acc: 0.8000
Epoch 307/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7042 - acc: 0.6643 - val_loss: 0.6105 - val_acc: 0.7667
Epoch 308/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7140 - acc: 0.6929 - val_loss: 0.5993 - val_acc: 0.7667
Epoch 309/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7095 - acc: 0.6429 - val_loss: 0.5940 - val_acc: 0.8000
Epoch 310/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6970 - acc: 0.6714 - val_loss: 0.5920 - val_acc: 0.8000
Epoch 311/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7237 - acc: 0.6929 - val_loss: 0.5891 - val_acc: 0.8333
Epoch 312/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6859 - acc: 0.6857 - val_loss: 0.5901 - val_acc: 0.8000
Epoch 313/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7264 - acc: 0.6643 - val_loss: 0.5954 - val_acc: 0.7667
Epoch 314/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6816 - acc: 0.7357 - val_loss: 0.5929 - val_acc: 0.8000
Epoch 315/500

```

```

140/140 [=====] - 2s 14ms/step - loss: 0
.6839 - acc: 0.7214 - val_loss: 0.5812 - val_acc: 0.8000
Epoch 316/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7000 - acc: 0.6929 - val_loss: 0.5842 - val_acc: 0.8000
Epoch 317/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6864 - acc: 0.6429 - val_loss: 0.5796 - val_acc: 0.8333
Epoch 318/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6800 - acc: 0.7071 - val_loss: 0.5775 - val_acc: 0.8000
Epoch 319/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7295 - acc: 0.6571 - val_loss: 0.5833 - val_acc: 0.8000
Epoch 320/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6952 - acc: 0.6714 - val_loss: 0.5843 - val_acc: 0.8000
Epoch 321/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6865 - acc: 0.7071 - val_loss: 0.5788 - val_acc: 0.8000
Epoch 322/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6739 - acc: 0.7286 - val_loss: 0.5808 - val_acc: 0.8000
Epoch 323/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6995 - acc: 0.6786 - val_loss: 0.5769 - val_acc: 0.8000
Epoch 324/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7283 - acc: 0.6571 - val_loss: 0.5733 - val_acc: 0.8333
Epoch 325/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7161 - acc: 0.6929 - val_loss: 0.5745 - val_acc: 0.8333
Epoch 326/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6676 - acc: 0.7214 - val_loss: 0.5665 - val_acc: 0.8000
Epoch 327/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7224 - acc: 0.6714 - val_loss: 0.5680 - val_acc: 0.8333
Epoch 328/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6924 - acc: 0.6929 - val_loss: 0.5734 - val_acc: 0.8333
Epoch 329/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6602 - acc: 0.7071 - val_loss: 0.5735 - val_acc: 0.8333
Epoch 330/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6810 - acc: 0.7071 - val_loss: 0.5743 - val_acc: 0.8333
Epoch 331/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6893 - acc: 0.7214 - val_loss: 0.5659 - val_acc: 0.8333
Epoch 332/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6801 - acc: 0.6714 - val_loss: 0.5627 - val_acc: 0.8333
Epoch 333/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6466 - acc: 0.7071 - val_loss: 0.5640 - val_acc: 0.8333
Epoch 334/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7177 - acc: 0.6857 - val_loss: 0.5659 - val_acc: 0.8333
Epoch 335/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6656 - acc: 0.7214 - val_loss: 0.5588 - val_acc: 0.8333
Epoch 336/500

```

```

140/140 [=====] - 2s 14ms/step - loss: 0
.6980 - acc: 0.6786 - val_loss: 0.5605 - val_acc: 0.8333
Epoch 337/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6960 - acc: 0.6929 - val_loss: 0.5597 - val_acc: 0.8333
Epoch 338/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7122 - acc: 0.6643 - val_loss: 0.5535 - val_acc: 0.8000
Epoch 339/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6788 - acc: 0.6786 - val_loss: 0.5540 - val_acc: 0.8333
Epoch 340/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6932 - acc: 0.6857 - val_loss: 0.5537 - val_acc: 0.8333
Epoch 341/500
140/140 [=====] - 2s 16ms/step - loss: 0
.6886 - acc: 0.6857 - val_loss: 0.5662 - val_acc: 0.8333
Epoch 342/500
140/140 [=====] - 2s 16ms/step - loss: 0
.6886 - acc: 0.7286 - val_loss: 0.5658 - val_acc: 0.8333
Epoch 343/500
140/140 [=====] - 2s 15ms/step - loss: 0
.6897 - acc: 0.6714 - val_loss: 0.5512 - val_acc: 0.8333
Epoch 344/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6633 - acc: 0.6786 - val_loss: 0.5536 - val_acc: 0.8333
Epoch 345/500
140/140 [=====] - 2s 14ms/step - loss: 0
.7432 - acc: 0.6214 - val_loss: 0.5495 - val_acc: 0.8333
Epoch 346/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6678 - acc: 0.7143 - val_loss: 0.5523 - val_acc: 0.8333
Epoch 347/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6403 - acc: 0.7071 - val_loss: 0.5538 - val_acc: 0.8333
Epoch 348/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6807 - acc: 0.7286 - val_loss: 0.5516 - val_acc: 0.8333
Epoch 349/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6858 - acc: 0.7000 - val_loss: 0.5468 - val_acc: 0.8333
Epoch 350/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6766 - acc: 0.6857 - val_loss: 0.5456 - val_acc: 0.8333
Epoch 351/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6731 - acc: 0.7000 - val_loss: 0.5448 - val_acc: 0.8333
Epoch 352/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6286 - acc: 0.7214 - val_loss: 0.5450 - val_acc: 0.8333
Epoch 353/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6713 - acc: 0.7000 - val_loss: 0.5457 - val_acc: 0.8333
Epoch 354/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6711 - acc: 0.7000 - val_loss: 0.5446 - val_acc: 0.8333
Epoch 355/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6228 - acc: 0.7143 - val_loss: 0.5487 - val_acc: 0.8333
Epoch 356/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6671 - acc: 0.7071 - val_loss: 0.5445 - val_acc: 0.8333
Epoch 357/500

```

```

140/140 [=====] - 2s 13ms/step - loss: 0
.6607 - acc: 0.6857 - val_loss: 0.5362 - val_acc: 0.8333
Epoch 358/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6711 - acc: 0.7143 - val_loss: 0.5363 - val_acc: 0.8333
Epoch 359/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6919 - acc: 0.7000 - val_loss: 0.5380 - val_acc: 0.8333
Epoch 360/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6374 - acc: 0.7143 - val_loss: 0.5352 - val_acc: 0.8333
Epoch 361/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6912 - acc: 0.7071 - val_loss: 0.5578 - val_acc: 0.8333
Epoch 362/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6800 - acc: 0.6643 - val_loss: 0.5360 - val_acc: 0.8333
Epoch 363/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6905 - acc: 0.7000 - val_loss: 0.5440 - val_acc: 0.8333
Epoch 364/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6342 - acc: 0.7000 - val_loss: 0.5289 - val_acc: 0.8000
Epoch 365/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6474 - acc: 0.7071 - val_loss: 0.5293 - val_acc: 0.8333
Epoch 366/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6690 - acc: 0.6857 - val_loss: 0.5298 - val_acc: 0.8333
Epoch 367/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6358 - acc: 0.7500 - val_loss: 0.5307 - val_acc: 0.8333
Epoch 368/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6170 - acc: 0.7357 - val_loss: 0.5258 - val_acc: 0.8333
Epoch 369/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6259 - acc: 0.7571 - val_loss: 0.5350 - val_acc: 0.8333
Epoch 370/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6588 - acc: 0.7071 - val_loss: 0.5334 - val_acc: 0.8333
Epoch 371/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6352 - acc: 0.7429 - val_loss: 0.5317 - val_acc: 0.8333
Epoch 372/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6450 - acc: 0.7000 - val_loss: 0.5272 - val_acc: 0.8333
Epoch 373/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6451 - acc: 0.6929 - val_loss: 0.5288 - val_acc: 0.8333
Epoch 374/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6644 - acc: 0.7071 - val_loss: 0.5316 - val_acc: 0.8333
Epoch 375/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6710 - acc: 0.6929 - val_loss: 0.5211 - val_acc: 0.8333
Epoch 376/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6616 - acc: 0.7357 - val_loss: 0.5233 - val_acc: 0.8333
Epoch 377/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6655 - acc: 0.7286 - val_loss: 0.5271 - val_acc: 0.8333
Epoch 378/500

```



```

140/140 [=====] - 2s 14ms/step - loss: 0
.6313 - acc: 0.7000 - val_loss: 0.5280 - val_acc: 0.8333
Epoch 379/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5937 - acc: 0.7714 - val_loss: 0.5209 - val_acc: 0.8333
Epoch 380/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6154 - acc: 0.7500 - val_loss: 0.5244 - val_acc: 0.8333
Epoch 381/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6787 - acc: 0.6714 - val_loss: 0.5253 - val_acc: 0.8333
Epoch 382/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6191 - acc: 0.7214 - val_loss: 0.5155 - val_acc: 0.8333
Epoch 383/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6068 - acc: 0.7071 - val_loss: 0.5177 - val_acc: 0.8333
Epoch 384/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6130 - acc: 0.7429 - val_loss: 0.5159 - val_acc: 0.8333
Epoch 385/500
140/140 [=====] - 2s 14ms/step - loss: 0
.5954 - acc: 0.7357 - val_loss: 0.5233 - val_acc: 0.8333
Epoch 386/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6363 - acc: 0.7000 - val_loss: 0.5159 - val_acc: 0.8333
Epoch 387/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6834 - acc: 0.7071 - val_loss: 0.5246 - val_acc: 0.8333
Epoch 388/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6226 - acc: 0.7143 - val_loss: 0.5190 - val_acc: 0.8333
Epoch 389/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6353 - acc: 0.7000 - val_loss: 0.5078 - val_acc: 0.8333
Epoch 390/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6308 - acc: 0.6714 - val_loss: 0.5118 - val_acc: 0.8333
Epoch 391/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6420 - acc: 0.7286 - val_loss: 0.5144 - val_acc: 0.8333
Epoch 392/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6609 - acc: 0.6714 - val_loss: 0.5111 - val_acc: 0.8333
Epoch 393/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6327 - acc: 0.7357 - val_loss: 0.5157 - val_acc: 0.8333
Epoch 394/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6110 - acc: 0.7286 - val_loss: 0.5115 - val_acc: 0.8333
Epoch 395/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6352 - acc: 0.7214 - val_loss: 0.5035 - val_acc: 0.8333
Epoch 396/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6264 - acc: 0.7143 - val_loss: 0.5033 - val_acc: 0.8333
Epoch 397/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6426 - acc: 0.7000 - val_loss: 0.5096 - val_acc: 0.8333
Epoch 398/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6028 - acc: 0.7429 - val_loss: 0.5095 - val_acc: 0.8333
Epoch 399/500

```

```

140/140 [=====] - 2s 13ms/step - loss: 0
.5977 - acc: 0.6857 - val_loss: 0.5080 - val_acc: 0.8333
Epoch 400/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6432 - acc: 0.6929 - val_loss: 0.4994 - val_acc: 0.8333
Epoch 401/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6400 - acc: 0.6929 - val_loss: 0.5079 - val_acc: 0.8333
Epoch 402/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6174 - acc: 0.6786 - val_loss: 0.5022 - val_acc: 0.8333
Epoch 403/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6343 - acc: 0.7000 - val_loss: 0.5011 - val_acc: 0.8333
Epoch 404/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6427 - acc: 0.7500 - val_loss: 0.5058 - val_acc: 0.8333
Epoch 405/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6407 - acc: 0.7000 - val_loss: 0.4976 - val_acc: 0.8333
Epoch 406/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5949 - acc: 0.7214 - val_loss: 0.4964 - val_acc: 0.8333
Epoch 407/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6156 - acc: 0.7500 - val_loss: 0.5083 - val_acc: 0.8333
Epoch 408/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6104 - acc: 0.7214 - val_loss: 0.4963 - val_acc: 0.8333
Epoch 409/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6263 - acc: 0.7286 - val_loss: 0.4950 - val_acc: 0.8333
Epoch 410/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6070 - acc: 0.7643 - val_loss: 0.4967 - val_acc: 0.8333
Epoch 411/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6186 - acc: 0.7071 - val_loss: 0.4964 - val_acc: 0.8333
Epoch 412/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6441 - acc: 0.7071 - val_loss: 0.5044 - val_acc: 0.8333
Epoch 413/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6224 - acc: 0.7000 - val_loss: 0.4893 - val_acc: 0.8000
Epoch 414/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6251 - acc: 0.6929 - val_loss: 0.4966 - val_acc: 0.8333
Epoch 415/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6367 - acc: 0.7286 - val_loss: 0.4995 - val_acc: 0.8333
Epoch 416/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6152 - acc: 0.7500 - val_loss: 0.4907 - val_acc: 0.8333
Epoch 417/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6090 - acc: 0.7357 - val_loss: 0.4915 - val_acc: 0.8333
Epoch 418/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5964 - acc: 0.7571 - val_loss: 0.4891 - val_acc: 0.8333
Epoch 419/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6096 - acc: 0.7357 - val_loss: 0.4976 - val_acc: 0.8333
Epoch 420/500

```

```

140/140 [=====] - 2s 13ms/step - loss: 0
.6139 - acc: 0.7071 - val_loss: 0.4886 - val_acc: 0.8333
Epoch 421/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6008 - acc: 0.7429 - val_loss: 0.4874 - val_acc: 0.8333
Epoch 422/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5921 - acc: 0.7571 - val_loss: 0.4904 - val_acc: 0.8333
Epoch 423/500
140/140 [=====] - 2s 14ms/step - loss: 0
.6038 - acc: 0.7143 - val_loss: 0.4933 - val_acc: 0.8333
Epoch 424/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6316 - acc: 0.7214 - val_loss: 0.4879 - val_acc: 0.8333
Epoch 425/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6116 - acc: 0.7357 - val_loss: 0.4822 - val_acc: 0.8333
Epoch 426/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5976 - acc: 0.7786 - val_loss: 0.4874 - val_acc: 0.8333
Epoch 427/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6391 - acc: 0.7143 - val_loss: 0.4836 - val_acc: 0.8333
Epoch 428/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5791 - acc: 0.8000 - val_loss: 0.4830 - val_acc: 0.8333
Epoch 429/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5983 - acc: 0.7214 - val_loss: 0.4870 - val_acc: 0.8333
Epoch 430/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6119 - acc: 0.7357 - val_loss: 0.4803 - val_acc: 0.8333
Epoch 431/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5801 - acc: 0.7500 - val_loss: 0.4877 - val_acc: 0.8333
Epoch 432/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5635 - acc: 0.7500 - val_loss: 0.4800 - val_acc: 0.8333
Epoch 433/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5886 - acc: 0.7571 - val_loss: 0.4919 - val_acc: 0.8333
Epoch 434/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6071 - acc: 0.7714 - val_loss: 0.4756 - val_acc: 0.8333
Epoch 435/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5788 - acc: 0.7571 - val_loss: 0.4763 - val_acc: 0.8333
Epoch 436/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6022 - acc: 0.7429 - val_loss: 0.4777 - val_acc: 0.8333
Epoch 437/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5941 - acc: 0.7214 - val_loss: 0.4740 - val_acc: 0.8333
Epoch 438/500
140/140 [=====] - 2s 14ms/step - loss: 0
.5801 - acc: 0.7714 - val_loss: 0.4752 - val_acc: 0.8333
Epoch 439/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5884 - acc: 0.7571 - val_loss: 0.4894 - val_acc: 0.8333
Epoch 440/500
140/140 [=====] - 2s 14ms/step - loss: 0
.5892 - acc: 0.7214 - val_loss: 0.4795 - val_acc: 0.8333
Epoch 441/500

```

```

140/140 [=====] - 2s 13ms/step - loss: 0
.5705 - acc: 0.7571 - val_loss: 0.4770 - val_acc: 0.8333
Epoch 442/500
140/140 [=====] - 2s 14ms/step - loss: 0
.5955 - acc: 0.7286 - val_loss: 0.4736 - val_acc: 0.8333
Epoch 443/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5485 - acc: 0.7714 - val_loss: 0.4699 - val_acc: 0.8333
Epoch 444/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5858 - acc: 0.7571 - val_loss: 0.4848 - val_acc: 0.8333
Epoch 445/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5786 - acc: 0.7357 - val_loss: 0.4707 - val_acc: 0.8333
Epoch 446/500
140/140 [=====] - 2s 14ms/step - loss: 0
.5656 - acc: 0.7500 - val_loss: 0.4740 - val_acc: 0.8333
Epoch 447/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6187 - acc: 0.7143 - val_loss: 0.4784 - val_acc: 0.8333
Epoch 448/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5524 - acc: 0.7286 - val_loss: 0.4662 - val_acc: 0.8333
Epoch 449/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5891 - acc: 0.7143 - val_loss: 0.4683 - val_acc: 0.8333
Epoch 450/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5728 - acc: 0.7500 - val_loss: 0.4704 - val_acc: 0.8333
Epoch 451/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6010 - acc: 0.6929 - val_loss: 0.4699 - val_acc: 0.8333
Epoch 452/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6007 - acc: 0.7286 - val_loss: 0.4718 - val_acc: 0.8333
Epoch 453/500
140/140 [=====] - 2s 13ms/step - loss: 0
.6112 - acc: 0.7429 - val_loss: 0.4657 - val_acc: 0.8333
Epoch 454/500
140/140 [=====] - 2s 14ms/step - loss: 0
.5967 - acc: 0.7429 - val_loss: 0.4639 - val_acc: 0.8333
Epoch 455/500
140/140 [=====] - 2s 14ms/step - loss: 0
.5582 - acc: 0.7643 - val_loss: 0.4725 - val_acc: 0.8333
Epoch 456/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5673 - acc: 0.7286 - val_loss: 0.4650 - val_acc: 0.8333
Epoch 457/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5560 - acc: 0.7929 - val_loss: 0.4622 - val_acc: 0.8333
Epoch 458/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5589 - acc: 0.7571 - val_loss: 0.4666 - val_acc: 0.8333
Epoch 459/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5852 - acc: 0.7357 - val_loss: 0.4604 - val_acc: 0.8333
Epoch 460/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5804 - acc: 0.7500 - val_loss: 0.4983 - val_acc: 0.8333
Epoch 461/500
140/140 [=====] - 2s 14ms/step - loss: 0
.5764 - acc: 0.7286 - val_loss: 0.4602 - val_acc: 0.8333
Epoch 462/500

```

```

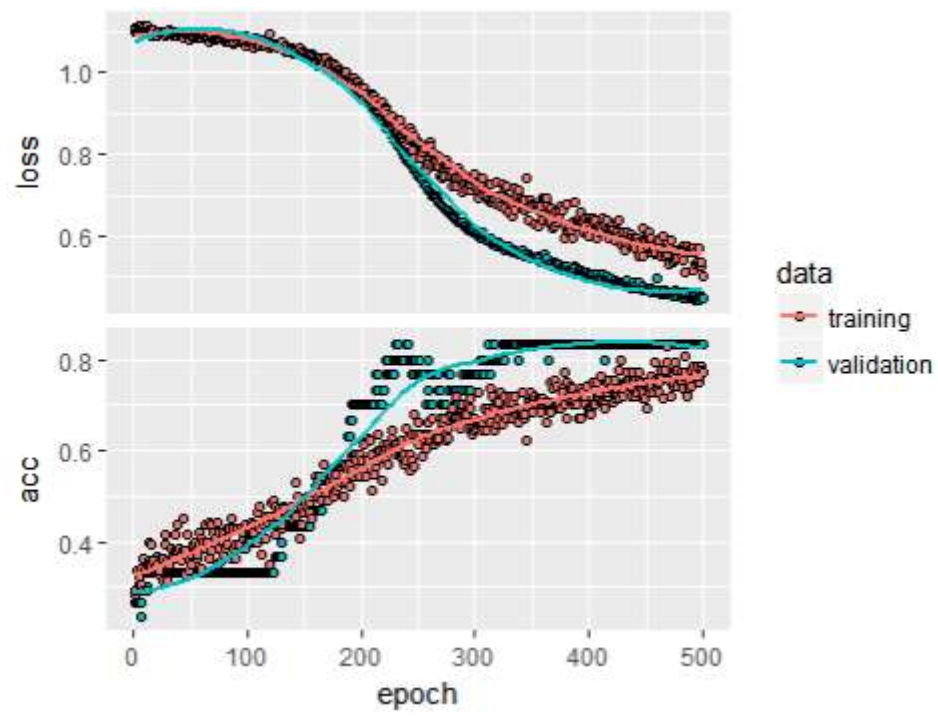
140/140 [=====] - 2s 14ms/step - loss: 0
.5923 - acc: 0.7500 - val_loss: 0.4653 - val_acc: 0.8333
Epoch 463/500
140/140 [=====] - 2s 14ms/step - loss: 0
.5549 - acc: 0.7643 - val_loss: 0.4598 - val_acc: 0.8333
Epoch 464/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5530 - acc: 0.7786 - val_loss: 0.4614 - val_acc: 0.8333
Epoch 465/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5982 - acc: 0.7143 - val_loss: 0.4627 - val_acc: 0.8333
Epoch 466/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5538 - acc: 0.7714 - val_loss: 0.4594 - val_acc: 0.8333
Epoch 467/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5487 - acc: 0.7786 - val_loss: 0.4666 - val_acc: 0.8333
Epoch 468/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5536 - acc: 0.8286 - val_loss: 0.4602 - val_acc: 0.8333
Epoch 469/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5716 - acc: 0.7929 - val_loss: 0.4565 - val_acc: 0.8333
Epoch 470/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5755 - acc: 0.7571 - val_loss: 0.4601 - val_acc: 0.8333
Epoch 471/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5299 - acc: 0.7571 - val_loss: 0.4552 - val_acc: 0.8333
Epoch 472/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5908 - acc: 0.7214 - val_loss: 0.4583 - val_acc: 0.8333
Epoch 473/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5374 - acc: 0.7429 - val_loss: 0.4622 - val_acc: 0.8333
Epoch 474/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5345 - acc: 0.7500 - val_loss: 0.4548 - val_acc: 0.8333
Epoch 475/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5394 - acc: 0.7929 - val_loss: 0.4652 - val_acc: 0.8333
Epoch 476/500
140/140 [=====] - 2s 14ms/step - loss: 0
.5636 - acc: 0.7786 - val_loss: 0.4556 - val_acc: 0.8333
Epoch 477/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5488 - acc: 0.7786 - val_loss: 0.4632 - val_acc: 0.8333
Epoch 478/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5445 - acc: 0.7429 - val_loss: 0.4511 - val_acc: 0.8333
Epoch 479/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5653 - acc: 0.7571 - val_loss: 0.4614 - val_acc: 0.8333
Epoch 480/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5536 - acc: 0.7571 - val_loss: 0.4566 - val_acc: 0.8333
Epoch 481/500
140/140 [=====] - 2s 14ms/step - loss: 0
.5643 - acc: 0.7714 - val_loss: 0.4517 - val_acc: 0.8333
Epoch 482/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5756 - acc: 0.7357 - val_loss: 0.4523 - val_acc: 0.8333
Epoch 483/500

```

```

140/140 [=====] - 2s 13ms/step - loss: 0
.5417 - acc: 0.7643 - val_loss: 0.4501 - val_acc: 0.8333
Epoch 484/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5125 - acc: 0.8071 - val_loss: 0.4604 - val_acc: 0.8333
Epoch 485/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5692 - acc: 0.7714 - val_loss: 0.4540 - val_acc: 0.8333
Epoch 486/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5731 - acc: 0.7214 - val_loss: 0.4478 - val_acc: 0.8333
Epoch 487/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5637 - acc: 0.7357 - val_loss: 0.4539 - val_acc: 0.8333
Epoch 488/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5590 - acc: 0.7500 - val_loss: 0.4535 - val_acc: 0.8333
Epoch 489/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5680 - acc: 0.7786 - val_loss: 0.4488 - val_acc: 0.8333
Epoch 490/500
140/140 [=====] - 2s 14ms/step - loss: 0
.5689 - acc: 0.7429 - val_loss: 0.4470 - val_acc: 0.8333
Epoch 491/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5691 - acc: 0.7857 - val_loss: 0.4488 - val_acc: 0.8333
Epoch 492/500
140/140 [=====] - 2s 14ms/step - loss: 0
.5151 - acc: 0.7857 - val_loss: 0.4462 - val_acc: 0.8333
Epoch 493/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5297 - acc: 0.7714 - val_loss: 0.4518 - val_acc: 0.8333
Epoch 494/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5581 - acc: 0.7714 - val_loss: 0.4446 - val_acc: 0.8333
Epoch 495/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5369 - acc: 0.7857 - val_loss: 0.4450 - val_acc: 0.8333
Epoch 496/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5679 - acc: 0.7500 - val_loss: 0.4555 - val_acc: 0.8333
Epoch 497/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5307 - acc: 0.7643 - val_loss: 0.4428 - val_acc: 0.8333
Epoch 498/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5244 - acc: 0.7857 - val_loss: 0.4531 - val_acc: 0.8333
Epoch 499/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5354 - acc: 0.7786 - val_loss: 0.4457 - val_acc: 0.8333
Epoch 500/500
140/140 [=====] - 2s 13ms/step - loss: 0
.5024 - acc: 0.7714 - val_loss: 0.4465 - val_acc: 0.8333

```



Lampiran 4 Hasil Evaluasi *Training* dan Prediksi *Test Data*

```

> #Evaluation&Prediction-test data (2)
> model %>% evaluate(test, testLabels)
30/30 [=====] - 0s 3ms/step
$loss
[1] 0.4465474

$acc
[1] 0.8333333

> pred<-model %>% predict_classes(test)
> table(Predicted = pred, Actual = testy)
      Actual
Predicted 0 1 2
         0 8 0 1
         1 0 9 1
         2 2 1 8

```


Lampiran 5 Hasil Evaluasi *Training Detail*

```
> prob<-model %>% predict_proba(train)
> cbind(prob, Predicted_class = pred, Actual = trainy)
```

			Predicted_class	Actual	
1					
0	[1,]	9.749415e-01	1.243251e-03	0.023815345	0
0	[2,]	8.369360e-01	1.713085e-02	0.145933151	0
0	[3,]	9.024103e-01	2.220752e-03	0.095368922	0
0	[4,]	9.042892e-01	2.371815e-02	0.071992621	0
0	[5,]	7.419530e-01	2.372354e-02	0.234323516	0
0	[6,]	7.501240e-01	7.969567e-02	0.170180321	0
0	[7,]	9.894910e-01	7.969728e-04	0.009712070	0
0	[8,]	9.160223e-01	1.227097e-02	0.071706697	0
0	[9,]	9.982775e-01	1.644039e-05	0.001706020	0
0	[10,]	3.825173e-01	1.353637e-01	0.482118994	2
0	[11,]	7.247431e-01	3.409719e-02	0.241159782	0
0	[12,]	7.867573e-01	1.010293e-02	0.203139752	0
0	[13,]	9.680935e-01	2.052046e-03	0.029854484	0
0	[14,]	9.067530e-01	2.811313e-02	0.065133832	0
0	[15,]	9.973659e-01	4.186110e-05	0.002592288	0
0	[16,]	9.067786e-01	6.528348e-03	0.086693004	0
0	[17,]	9.943370e-01	6.061178e-05	0.005602343	0
0	[18,]	7.442153e-01	1.245517e-02	0.243329525	0
0	[19,]	5.256305e-01	2.037601e-01	0.270609289	0
0	[20,]	2.504455e-01	1.742173e-01	0.575337231	2
0	[21,]	6.600259e-01	4.076191e-02	0.299212158	0
0	[22,]	9.582579e-01	7.048340e-03	0.034693815	0
0	[23,]	9.979609e-01	2.055604e-05	0.002018467	0
0	[24,]	9.769741e-01	2.975851e-03	0.020050015	0
0	[25,]	8.948011e-02	8.924604e-02	0.821273863	2

0	[26,]	9.754099e-01	2.441966e-03	0.022148149	0
0	[27,]	6.804641e-01	3.982793e-03	0.315553159	0
0	[28,]	6.600850e-01	9.513600e-02	0.244778991	0
0	[29,]	9.811357e-01	2.096075e-04	0.018654788	0
0	[30,]	9.961707e-01	5.434976e-05	0.003774957	0
0	[31,]	6.303394e-01	1.758053e-01	0.193855196	0
0	[32,]	8.271883e-01	6.052515e-02	0.112286650	0
0	[33,]	9.983211e-01	1.426944e-05	0.001664637	0
0	[34,]	9.456188e-01	2.654085e-03	0.051727168	0
0	[35,]	6.623717e-01	8.450084e-02	0.253127426	0
0	[36,]	8.064240e-01	5.745157e-03	0.187830895	0
0	[37,]	9.286107e-01	2.951339e-03	0.068437934	0
0	[38,]	9.429491e-01	7.979628e-03	0.049071260	0
0	[39,]	2.924923e-01	2.517797e-01	0.455727994	2
0	[40,]	9.394515e-01	2.460010e-03	0.058088608	0
0	[41,]	9.775428e-01	7.469648e-04	0.021710299	0
0	[42,]	5.098598e-01	2.280040e-01	0.262136251	0
0	[43,]	1.107538e-01	3.665368e-01	0.522709310	2
0	[44,]	9.129112e-01	1.103841e-02	0.076050438	0
0	[45,]	9.733303e-01	1.181192e-03	0.025488503	0
0	[46,]	5.053042e-01	6.146280e-02	0.433232993	0
0	[47,]	9.937202e-01	1.339551e-04	0.006145874	0
0	[48,]	6.159221e-01	2.444871e-02	0.359629065	0
0	[49,]	9.424865e-01	4.008380e-03	0.053505089	0
0	[50,]	9.442376e-01	9.922624e-03	0.045839678	0
1	[51,]	4.161474e-01	3.032802e-01	0.280572444	0
1	[52,]	1.639605e-02	7.718451e-01	0.211758867	1
1	[53,]	3.061354e-01	2.501216e-01	0.443742901	2

1	[54,]	7.467548e-04	8.839905e-01	0.115262784	1
1	[55,]	1.515774e-02	6.462099e-01	0.338632345	1
1	[56,]	1.006806e-01	3.460731e-01	0.553246319	2
1	[57,]	1.232136e-02	5.368215e-01	0.450857103	1
1	[58,]	5.476414e-01	1.936938e-01	0.258664757	0
1	[59,]	7.386885e-04	8.869485e-01	0.112312838	1
1	[60,]	1.509213e-03	9.020197e-01	0.096471123	1
1	[61,]	1.035784e-02	7.370777e-01	0.252564430	1
1	[62,]	9.569742e-02	5.958460e-01	0.308456630	1
1	[63,]	1.005614e-02	7.769889e-01	0.212954909	1
1	[64,]	2.822599e-01	2.290623e-01	0.488677800	2
1	[65,]	1.876073e-04	9.291224e-01	0.070689969	1
1	[66,]	1.206284e-02	6.686607e-01	0.319276422	1
1	[67,]	8.335602e-03	6.632900e-01	0.328374445	1
1	[68,]	2.959657e-03	9.132673e-01	0.083773136	1
1	[69,]	1.466092e-01	2.046677e-01	0.648723125	2
1	[70,]	4.302526e-01	7.576707e-02	0.493980229	2
1	[71,]	7.816939e-03	7.804496e-01	0.211733446	1
1	[72,]	2.083961e-02	7.374367e-01	0.241723731	1
1	[73,]	4.412608e-07	9.951614e-01	0.004838244	1
1	[74,]	1.506598e-04	8.855380e-01	0.114311345	1
1	[75,]	7.342530e-02	5.050650e-01	0.421509683	1
1	[76,]	4.703663e-02	4.149095e-01	0.538053930	2
1	[77,]	1.195694e-04	9.746200e-01	0.025260486	1
1	[78,]	9.798242e-03	6.842576e-01	0.305944145	1
1	[79,]	1.532953e-02	7.444385e-01	0.240231931	1
1	[80,]	1.507456e-04	9.495990e-01	0.050250251	1
1	[81,]	6.871385e-04	8.860239e-01	0.113288902	1

	[82,]	3.531028e-03	8.348312e-01	0.161637664	1
1					
	[83,]	2.817989e-02	4.283787e-01	0.543441474	2
1					
	[84,]	1.506598e-04	8.855380e-01	0.114311345	1
1					
	[85,]	6.871385e-04	8.860239e-01	0.113288902	1
1					
	[86,]	1.059294e-03	8.611335e-01	0.137807235	1
1					
	[87,]	1.177170e-03	9.252229e-01	0.073599800	1
1					
	[88,]	1.409522e-04	9.375179e-01	0.062341232	1
1					
	[89,]	6.363948e-04	9.313914e-01	0.067972220	1
1					
	[90,]	1.085017e-02	6.850621e-01	0.304087758	1
1					
	[91,]	1.876073e-04	9.291224e-01	0.070689969	1
1					
	[92,]	6.363948e-04	9.313914e-01	0.067972220	1
1					
	[93,]	7.779765e-04	9.238419e-01	0.075380206	1
1					
	[94,]	2.605938e-01	2.899857e-01	0.449420512	2
2					
	[95,]	4.439209e-01	6.987539e-02	0.486203671	2
2					
	[96,]	1.656154e-01	2.905164e-01	0.543868184	2
2					
	[97,]	2.261494e-01	8.946697e-02	0.684383631	2
2					
	[98,]	3.355549e-02	4.040903e-01	0.562354267	2
2					
	[99,]	5.638266e-01	3.362340e-02	0.402549982	0
2					
	[100,]	5.368095e-01	4.133911e-02	0.421851397	0
2					
	[101,]	6.561050e-02	3.129559e-01	0.621433616	2
2					
	[102,]	1.535763e-02	6.652749e-01	0.319367409	1
2					
	[103,]	2.173062e-01	2.187757e-01	0.563918054	2
2					
	[104,]	2.938817e-01	1.046516e-01	0.601466715	2
2					
	[105,]	1.324122e-01	2.854643e-01	0.582123399	2
2					
	[106,]	9.548149e-02	4.254839e-01	0.479034573	2
2					
	[107,]	6.818548e-04	8.365801e-01	0.162738070	1
2					
	[108,]	1.570742e-01	1.800853e-01	0.662840486	2
2					
	[109,]	3.900071e-02	1.873138e-01	0.773685515	2
2					

[110,]	4.402051e-02	4.764528e-01	0.479526699	2
2				
[111,]	1.364420e-01	3.114581e-01	0.552099884	2
2				
[112,]	1.619290e-01	1.122847e-01	0.725786328	2
2				
[113,]	5.938051e-02	2.095297e-01	0.731089711	2
2				
[114,]	8.399723e-02	1.674172e-01	0.748585582	2
2				
[115,]	1.848857e-01	1.797480e-01	0.635366321	2
2				
[116,]	7.504911e-01	1.175949e-02	0.237749428	0
2				
[117,]	2.308785e-01	2.315070e-01	0.537614465	2
2				
[118,]	1.806072e-01	1.419978e-01	0.677394927	2
2				
[119,]	8.189851e-03	5.975603e-01	0.394249797	1
2				
[120,]	1.781886e-01	3.424082e-01	0.479403228	2
2				
[121,]	1.049469e-01	2.789368e-01	0.616116166	2
2				
[122,]	2.013358e-01	1.893178e-01	0.609346449	2
2				
[123,]	3.322794e-02	2.436650e-01	0.723107040	2
2				
[124,]	2.595935e-01	1.420505e-01	0.598356068	2
2				
[125,]	1.403176e-01	2.942523e-01	0.565430105	2
2				
[126,]	4.576334e-02	8.793399e-02	0.866302669	2
2				
[127,]	7.775248e-01	6.187344e-03	0.216287926	0
2				
[128,]	6.472647e-01	3.187427e-02	0.320861071	0
2				
[129,]	2.513829e-01	8.629085e-02	0.662326276	2
2				
[130,]	8.081568e-01	1.898311e-02	0.172860146	0
2				
[131,]	1.331061e-01	3.981791e-01	0.468714833	2
2				
[132,]	6.226999e-02	3.146082e-01	0.623121798	2
2				
[133,]	3.640975e-02	4.113305e-01	0.552259743	2
2				
[134,]	7.066783e-04	8.292962e-01	0.169997126	1
2				
[135,]	1.760343e-01	1.290297e-01	0.694935977	2
2				
[136,]	5.632473e-02	5.801076e-01	0.363567740	1
2				
[137,]	8.812612e-02	4.652376e-01	0.446636289	1
2				

```
[138,] 4.455536e-02 1.074457e-01 0.847998917 2
2
[139,] 4.032075e-02 3.888989e-01 0.570780277 2
2
[140,] 2.699653e-03 6.465419e-01 0.350758463 1
2
```

Lampiran 6 Hasil Prediksi *Test Data*

```
> prob<-model %>% predict_proba(test)
> cbind(prob, Predicted_class = pred, Actual = testy)
```

				Predicted_class	Actual
[1,]	0.7067659497	0.032280438	0.26095352	0	0
[2,]	0.8861650229	0.032321226	0.08151382	0	0
[3,]	0.9676832557	0.007247581	0.02506922	0	0
[4,]	0.9053021669	0.013564077	0.08113374	0	0
[5,]	0.2246435732	0.087057516	0.68829900	2	0
[6,]	0.8717578650	0.009741879	0.11850017	0	0
[7,]	0.8797222972	0.024098493	0.09617919	0	0
[8,]	0.8989261389	0.002609324	0.09846448	0	0
[9,]	0.8606190681	0.023563169	0.11581782	0	0
[10,]	0.1998324543	0.036913026	0.76325446	2	0
[11,]	0.0011771697	0.925222933	0.07359980	1	1
[12,]	0.0103578428	0.737077713	0.25256443	1	1
[13,]	0.0001409522	0.937517881	0.06234123	1	1
[14,]	0.0029931683	0.762084723	0.23492214	1	1
[15,]	0.0002607608	0.937425792	0.06231349	1	1
[16,]	0.0027067617	0.839968801	0.15732440	1	1
[17,]	0.0006363948	0.931391358	0.06797222	1	1
[18,]	0.0832910985	0.564199269	0.35250959	1	1
[19,]	0.1093633920	0.336402893	0.55423367	2	1
[20,]	0.0108501669	0.685062051	0.30408776	1	1
[21,]	0.0822613090	0.337435186	0.58030355	2	2
[22,]	0.0526664034	0.318629503	0.62870407	2	2
[23,]	0.6661136150	0.035578847	0.29830757	0	2
[24,]	0.2026267797	0.129059449	0.66831374	2	2
[25,]	0.3312938213	0.132766381	0.53593981	2	2
[26,]	0.1860550642	0.194167003	0.61977798	2	2
[27,]	0.0652833804	0.247027099	0.68768954	2	2
[28,]	0.0698613971	0.386234164	0.54390436	2	2
[29,]	0.0334858149	0.618921041	0.34759319	1	2
[30,]	0.1057612523	0.181264803	0.71297389	2	2