PENGENALAN JENIS HAMA SERANGGA MENGGUNAKAN ALGORITMA *CONVOLUTIONAL NEURAL NETWORK* DENGAN ARSITEKTUR MOBILENETV2

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Adhi Setiawan
NIM: 175150219111002

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2021

PENGESAHAN

PENGENALAN JENIS HAMA SERANGGA MENGGUNAKAN ALGORITMA CONVOLUTIONAL NEURAL NETWORK DENGAN ARSITEKTUR MOBILENETV2

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Adhi Setiawan
NIM: 175150219111002

Skripsi ini telah diuji dan dinyatakan lulus pada
19 Juli 2021
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing 1	Dosen Pembimbing 2
Dr. Eng. Novanto Yudistira, S.Kom., M.Sc.	Randy Cahya Wihandika, S.ST., M.Kom.
NIK: 2012018311101001	NIK: 2014058802061001

Mengetahui
Ketua Jurusan Teknik Informatika

Achmad Basuki, S.T., M.MG., Ph.D.
NIP: 19741118 200312 1 002

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar referensi.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 8 Juni 2021

Adhi Setiawan

NIM: 175150219111002

ABSTRAK

Adhi Setiawan, Pengenalan Jenis Hama Serangga Menggunakan Algoritma Convolutional Neural Network Dengan Arsitektur MobileNetV2

Pembimbing: Novanto Yudistira, Dr. Eng., S.Kom., M.Sc. dan Randy Cahya Wihandika, S.ST., M.Kom.

Serangga merupakan jenis hewan dengan populasi terbanyak di bumi. Dalam bidang pertanian, serangga sering disebut sebagai hama karena bersifat sebagai predator dan *parasitoid* pada tanaman. Jumlah populasi hama yang terlalu banyak akan merusak tanaman, oleh karena itu perlu dilakukan pencegahan agar populasi hama ini dapat terkontrol. Namun setiap jenis hama memiliki pencegahan yang berbeda-beda, banyaknya jenis hama yang ada juga akan menyulitkan untuk identifikasi jenis hama apa yang menyerang. Pembuatan sistem kecerdasan buatan berbasiskan pengenalan citra hama dengan algoritma CNN dan arsitektur MobileNetV2, dirasa akan sangat membantu dalam proses identifikasi hama tersebut agar pencegahan yang tepat bisa dilakukan. Dalam melakukan pelatihan pada algoritma CNN ini, digunakan *dataset* IP102 yang terdiri dari 75.000 citra serangga dan terbagi menjadi 102 jenis serangga yang berbeda-beda. Teknik pelatihan yang digunakan adalah dengan konfigurasi *optimizer* AdamW dengan nilai *learning rate* awal yaitu 0,0001 dan penambahan *dropout* sebesar 0,2 pada *layer linear*. Pada penelitian ini dilakukan beberapa pengujian yaitu menggunakan *dynamic learning rate*, penambahan *cutmix*, dan penggunaan *sparse regularization*. Hasil akurasi tertinggi sebesar 0,7132 didapatkan dari penggabungan ketiga metode tersebut. Penggunaan *dynamic learning rate* akan membantu proses penurunan gradien menjadi perlahan saat mendekati *global minima*. Penambahan *cutmix* juga membuat model bisa belajar lebih banyak dari data yang berbeda-beda, meskipun terjadi sedikit *underfitting* namun dari pengujian pada data uji menunjukkan hasil yang lebih baik. *Sparse regularization* juga berpengaruh karena akan membuat nilai keluaran dari fungsi aktivasi tidak terlalu jauh dari nol.

Kata kunci: pengenalan jenis hama, CNN, MobileNetV2, *Dynamic Learning Rate*, *CutMix*, *Sparse Regularization*

ABSTRACT

Adhi Setiawan, *Insect Pest Recognition Using Convolutional Neural Network Algorithm with MobileNetV2 Architecture*.

***Supervisors*: Novanto Yudistira, Dr. Eng., S.Kom., M.Sc. and Randy Cahya Wihandika, S.ST., M.Kom.**

Insects are the most common type of animal on earth. In agriculture, insects are often referred to as pests because they are predators and parasitoids of plants. The number of pest populations that are too much will damage the plants, therefore it is necessary to take precautions so that these pest populations can be controlled. However, each type of pest has different prevention, the many types of pests that exist will also make it challenging to identify what types of pest attack. Creating an artificial intelligence system based on pest image recognition with the CNN algorithm and the MobileNetV2 architecture is felt that will be very helpful in the process of identifying these pests so that proper prevention can be done. In conducting training on the CNN algorithm using the IP102 dataset consisted of 75,000 insect images and was divided into 102 different types of insects. The training technique used is the AdamW optimizer configuration with the initial learning rate value of 0.0001 and dropout of 0.2 on the linear layer. Several experiments were carried out in this study, using dynamic learning rate, adding cutmix, and using sparse regularization. The highest accuracy result of 0.7132 was obtained from the combination of the three methods. The use of a dynamic learning rate will help the gradient descent process be slower as it approaches the global minima. The addition of a cutmix also makes the model able to learn more from different data. Although there is a little underfitting, but the data testing shows better results. Sparse regularization also affects because it will make the output value of the activation function not too far from zero.

***Keyword**: insect pest recognition, CNN, MobileNetV2, Dynamic Learning Rate, CutMix, Sparse Regularization*

DAFTAR ISI

PENGESAHAN ii
PERNYATAAN ORISINALITAS iii
PRAKATA iv
ABSTRAK vi
ABSTRACT vii
DAFTAR ISI viii
DAFTAR TABEL xi
DAFTAR GAMBAR xii
BAB 1 PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Tujuan 2
1.4 Manfaat 3
1.5 Batasan Masalah 3
1.6 Sistematika Pembahasan 3
BAB 2 LANDASAN KEPUSTAKAAN 5
2.1 Landasan Kepustakaan 5
2.2 *Dataset* Jenis Hama 6
2.3 *Deep Learning* 6
2.3.1 *Convolutional Neural Network* (CNN) 7
2.3.2 MobileNetV2 8
2.3.3 *Batch Normalization* 10
2.3.4 Augmentasi Data 11
2.3.5 Fungsi Aktivasi 12
2.3.6 Fungsi *Loss* 13
2.3.7 *Sparse Regularization* 13
BAB 3 METODOLOGI PENELITIAN 15
3.1 Tipe Penelitian 15
3.2 Strategi dan Rancangan Penelitian 15
3.2.1 Strategi Penelitian 15

3.2.2 Lokasi Penelitian 15
3.2.3 Metode Pengumpulan Data 15
3.2.4 Metode Analisis Hasil 15
3.2.5 Peralatan Pendukung 16
3.2.6 Metode Penelitian 16
3.2.7 Augmentasi Data 16
3.2.8 Pelatihan dan Pengujian 17
BAB 4 Perancangan 18
4.1 Perancangan Algoritma 18
4.1.1 Diagram Alir Augmentasi Citra 19
4.1.2 Diagram Alir Membangun Arsitektur 20
4.1.3 Diagram Alir Pelatihan Model 23
4.1.4 Diagram Alir Prediksi atau Validasi 24
4.2 Perhitungan Manual 25
4.2.1 Propagasi Maju 26
4.2.2 Perhitungan Akurasi 34
4.2.3 Perhitungan Nilai *Loss* 34
4.2.4 Propagasi Balik 34
4.3 Perancangan Pengujian 36
4.3.1 Perancangan Pengujian Pengaruh *Dynamic Learning Rate* Terhadap Akurasi 36
4.3.2 Perancangan Pengujian Pengaruh Metode Augmentasi Data Terhadap Akurasi 36
4.3.3 Perancangan Pengujian Pengaruh *Sparse Regularization* Terhadap Akurasi 37
BAB 5 implementasi 38
5.1 Implementasi Perhitungan Manual 38
5.2 Implementasi Pemuat Dataset 41
5.2.1 Implementasi Struktur Dataset 41
5.2.2 Implementasi Pemuat Dataset (*Dataloader*) 41
5.3 Implementasi Fungsi Loss 42
5.4 Implementasi Optimizer 43
5.5 Implementasi *Learning Rate Scheduler* 45

5.6 Implementasi *Sparse Regularization* 46
BAB 6 Pengujian dan analisis hasil 47
6.1 Pengujian Pengaruh *Dynamic Learning Rate* Terhadap Akurasi 47
6.2 Pengujian Pengaruh Metode *CutMix* Terhadap Akurasi 48
6.3 Pengujian Pengaruh *Sparse Regularization* Terhadap Akurasi..... 50
6.4 Perbandingan Hasil Pengujian Dengan Penelitian Terdahulu 52
BAB 7 Kesimpulan 54
7.1 Kesimpulan..... 54
7.2 Saran 54
DAFTAR REFERENSI 55

DAFTAR TABEL

Tabel 3.1 Spesifikasi Perangkat Keras 16
Tabel 3.2 Spesifikasi Perangkat Lunak 16
Tabel 4.1 Pengaruh *Dynamic Learning Rate* Terhadap Akurasi..... 36
Tabel 4.2 Pengaruh Penambahan *CutMix* Terhadap Akurasi 37
Tabel 4.3 Pengaruh *Sparse Regularization* Terhadap Akurasi 37
Tabel 6.1 Hasil Pengujian Pengaruh *Dynamic Learning Rate* Terhadap Akurasi .. 47
Tabel 6.2 Hasil Pengujian Pengaruh Metode *CutMix* Terhadap Akurasi 49
Tabel 6.3 Hasil Pengujian Pengaruh *Sparse Regularization* Terhadap Akurasi 50
Tabel 6.4 Hasil Akurasi Perbandingan Dengan Penelitian Terdahulu..... 53

DAFTAR GAMBAR

Gambar 2.1 *Dataset* IP102 6
Gambar 2.2 Ilustrasi Arsitektur Jaringan Saraf Tiruan Sederhana 7
Gambar 2.3 Ilustrasi Arsitektur Jaringan Saraf Pada *Deep Learning* 7
Gambar 2.4 Arsitektur *Convolutional Neural Network* (CNN) 8
Gambar 2.5 Illustrasi Proses Pada CNN 8
Gambar 2.6 Perbedaan Proses Konvolusi Standar dan Konvolusi Pada MobileNetV1 9
Gambar 2.7 *Convolutional Block* Pada MobileNetV2 10
Gambar 2.8 Illustrasi *Random Rotation* 11
Gambar 2.9 Illustrasi *Random Resize Crop* 12
Gambar 2.10 Illustrasi *CutMix* 12
Gambar 2.11 Ilustrasi Fungsi Aktivasi ReLU6 13
Gambar 3.1 Alur Metode Penelitian 16
Gambar 4.1 Diagram Alir Sistem 18
Gambar 4.2 Diagram Alir Augmentasi Citra 19
Gambar 4.3 Diagram Alir Pembangunan Arsitektur 20
Gambar 4.4 Diagram Alir Pembangunan Lapisan *Convolutional Block* 21
Gambar 4.5 Diagram Alir Pembangunan Lapisan *Inverted Residual* 22
Gambar 4.6 Diagram Alir Pelatihan Model 23
Gambar 4.7 Diagram Alir Prediksi atau Validasi 24
Gambar 4.8 Data Masukkan *Channel* 1 25
Gambar 4.9 Data Masukkan *Channel* 2 25
Gambar 4.10 Data Masukkan *Channel* 3 25
Gambar 4.11 Arsitektur Jaringan Perhitungan Manual 26
Gambar 4.12 Bobot Filter Konvolusi 1 (*Channel* 1) 26
Gambar 4.13 Bobot Filter Konvolusi 1 (*Channel* 2) 26
Gambar 4.14 Bobot Filter Konvolusi 1 (*Channel* 3) 26
Gambar 4.15 Perhitungan Konvolusi Input Channel 1 27
Gambar 4.16 Perhitungan Konvolusi Bobot Filter Channel 1 27
Gambar 4.17 Hasil Konvolusi Channel 1 27

Gambar 4.18 *Feature Map* Hasil Konvolusi 1 27
Gambar 4.19 Hasil Perhitungan *Mini Batch Mean* dan *Variance* 28
Gambar 4.20 Hasil *Normalize Feature Map* Konvolusi Satu 28
Gambar 4.21 Hasil Propagasi Maju Pada Lapisan *Batch Normalization* 28
Gambar 4.22 Hasil Propagasi Maju Pada Lapisan ReLU6 28
Gambar 4.23 Bobot Filter Lapisan *Expansion* (3 filter 1 *Channel*) 29
Gambar 4.24 *Feature Map* Hasil Konvolusi Lapisan *Expansion* (*Channel* 1) 29
Gambar 4.25 *Feature Map* Hasil Konvolusi Lapisan *Expansion* (*Channel* 2) 29
Gambar 4.26 *Feature Map* Hasil Konvolusi Lapisan *Expansion* (*Channel* 3) 29
Gambar 4.27 Hasil *Batch Normalization* Lapisan *Expansion* (*Channel* 1) 29
Gambar 4.28 Hasil *Batch Normalization* Lapisan *Expansion* (*Channel* 2) 30
Gambar 4.29 Hasil *Batch Normalization* Lapisan *Expansion* (*Channel* 3) 30
Gambar 4.30 Hasil Aktivasi Lapisan *Expansion* (*Channel* 1) 30
Gambar 4.31 Hasil Aktivasi Lapisan *Expansion* (*Channel* 2) 30
Gambar 4.32 Hasil Aktivasi Lapisan *Expansion* (*Channel* 3) 30
Gambar 4.33 Bobot Pada Filter Lapisan *Depthwise* (*Channel* 1) 30
Gambar 4.34 Bobot Pada Filter Lapisan *Depthwise* (*Channel* 2) 31
Gambar 4.35 Bobot Pada Filter Lapisan *Depthwise* (*Channel* 3) 31
Gambar 4.36 *Feature Map* Hasil *Depthwise* Konvolusi (*Channel* 1) 31
Gambar 4.37 *Feature Map* Hasil *Depthwise* Konvolusi (*Channel* 2) 31
Gambar 4.38 *Feature Map* Hasil *Depthwise* Konvolusi (*Channel* 3) 31
Gambar 4.39 Hasil *Batch Normalization* Lapisan *Depthwise* (*Channel* 1) 31
Gambar 4.40 Hasil *Batch Normalization* Lapisan *Depthwise* (*Channel* 2) 32
Gambar 4.41 Hasil *Batch Normalization* Lapisan *Depthwise* (*Channel* 3) 32
Gambar 4.42 Hasil Aktivasi ReLU6 (*Channel* 1) 32
Gambar 4.43 Hasil Aktivasi ReLU6 (*Channel* 2) 32
Gambar 4.44 Hasil Aktivasi ReLU6 (*Channel* 3) 32
Gambar 4.45 Bobot Filter Lapisan *Projection* (1 Filter 3 *Channel*) 32
Gambar 4.46 *Feature Map* Hasil Konvolusi Lapisan *Projection* 33
Gambar 4.47 Hasil *Batch Normalization* Lapisan *Projection* 33
Gambar 4.48 Hasil Penambahan *Skip Connection* 33
Gambar 4.49 Bobot Lapisan Linear 33

Gambar 4.50 Hasil Operasi Lapisan Linear 33
Gambar 4.51 Hasil Aktivasi *Softmax* Lapisan Linear 34
Gambar 4.52 Gradien Loss Terhadap Output Softmax (dLoss/dOutSoftmax) 34
Gambar 4.53 Gradien Input Softmax Terhadap Output Softmax (dOutSoftmax/dInSoftmax) 35
Gambar 4.54 Gradien Input Softmax Terhadap Bobot Linear (dOutSoftmax/dWeightLinear) 35
Gambar 4.55 Gradien Loss Terhadap Bobot Linear (dLoss/dWeight) 1 35
Gambar 4.56 Gradien Loss Terhadap Bobot Linear (dLoss/dWeight) 2 35
Gambar 4.57 Gradien Loss Terhadap Bobot Linear (dLoss/dWeight) 3 35
Gambar 4.58 Hasil Pembaruan Bobot 1 35
Gambar 4.59 Hasil Pembaruan Bobot 2 36
Gambar 4.60 Hasil Pembaruan Bobot 3 36
Gambar 6.1 Grafik Hasil Pengujian MobileNetV2 48
Gambar 6.2 Grafik Hasil Pengujian MobileNetV2 dan *Dynamic Learning Rate* ... 48
Gambar 6.3 Grafik Hasil Pengujian Penggunaan *CutMix* 49
Gambar 6.4 Tanaman Yang Mendominasi Pada Citra 50
Gambar 6.5 Grafik Hasil Pengujian MobileNetV2 + *Sparse* 51
Gambar 6.6 Grafik Hasil Pengujian MobileNetV2 + *Sparse* + *Dynamic LR* 51
Gambar 6.7 Grafik Hasil Pengujian MobileNetV2 + *Sparse* + *Weight Decay* + *Dynamic Learning Rate* 51
Gambar 6.8 Grafik Hasil Pengujian MobileNetV2 + *Sparse* + *Cutmix* + *Dynamic Learning Rate* 52

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Serangga merupakan jenis hewan dengan populasi terbanyak di bumi. Menurut beberapa penelitian, jumlah spesies serangga hampir bernilai 80% dari keseluruhan hewan yang ada di bumi (Meilin dan Nasamsir, 2016), hal ini membuktikan bahwa serangga memang merupakan salah satu jenis hewan yang terbanyak dengan kemampuan adaptasi yang tinggi. Dalam bidang pertanian serangga sering disebut sebagai hama (Rothschild, 1981), karena sebagian besar serangga pada pertanian bersifat sebagai predator dan *parasitoid* (Westerkamp dan Gottsberger, 2000). Populasi hama yang banyak dapat merusak tanaman. Dalam kasus yang lebih besar pada sebuah lahan, tentu hal ini akan sangat merugikan apabila populasi dari hama tersebut tidak bisa terkontrol dengan baik. Biasanya para petani sudah menyiapkan berbagai rencana untuk mengatasi serangan hama tersebut, salah satunya adalah dengan penggunaan insektisida. Insektisida terbukti bisa mengatasi bertambahnya populasi hama yang ada pada sebuah tanaman.

Permasalahan baru yang muncul adalah tidak semua hama akan bisa terbunuh dengan insektisida, terdapat jenis insektisida yang hanya berfungsi untuk beberapa jenis hama saja. Apabila terdapat suatu jenis hama baru yang belum dikenali namun langsung diberikan insektisida yang kurang tepat, maka hama tersebut tidak akan mati dan bisa jadi penggunaan insektisida tersebut justru akan berdampak lain pada tanaman. Jumlah insektisida yang berlebihan juga tidak baik untuk lingkungan (Djojosumarto, 2008), oleh karena itu perlu pemilihan yang tepat dalam penggunaan insektisida bagi jenis hama tertentu agar tidak sampai salah penggunaan. Banyaknya jenis hama juga akan menyulitkan untuk melakukan identifikasi jenis hama yang menyerang suatu tanaman, karena tidak semua orang tau jenis hama. Keadaan ini juga akan menyulitkan untuk menentukan insektisida apa yang cocok digunakan pada hama tersebut serta tidak sampai merusak lingkungan.

Permasalahan tersebut bisa diatasi apabila para petani atau pemilik tanaman mampu mengenali jenis hama yang menyerang. Sehingga mereka akan bisa menentukan langkah apa yang seharusnya mereka ambil ketika terdapat jenis hama tertentu. Penggunaan insektisida juga bisa dikontrol karena hama yang ada sudah diketahui jenisnya, lalu bisa menentukan insektisida jenis apa yang akan digunakan untuk membasmi hama tersebut dan tidak sampai menggunakan secara berlebihan. Dengan adanya solusi tersebut, pembuatan sebuah sistem kecerdasan buatan yang mampu mengenali berbagai macam jenis hama dirasa akan sangat membantu dalam pengidentifikasian jenis hama yang menyerang. Salah satu sistem kecerdasan buatan yang bisa dibuat adalah sebuah pengenalan citra hama menggunakan salah satu algoritma *Deep Learning* yaitu *Convolutional Neural Network* (CNN). Penggunaan algoritma *deep learning* ini dirasa cocok dalam kasus pengenalan jenis hama, karena pada prosesnya bisa menjadi lebih

efektif dalam ekstraksi fitur serta secara spesifik penggunaan deep learning memang dikhususkan untuk data yang tidak terstruktur seperti citra.

Convolutional Neural Network (CNN) merupakan sebuah algoritma yang mirip dengan *Artificial Neural Network* (ANN) tradisional karena terdiri dari *neuron* yang dioptimalkan sendiri melalui pembelajaran (O'Shea dan Nash, 2015). Perbedaan mendasar antara CNN dan ANN adalah pada data yang digunakan dalam proses pembelajaran, CNN lebih cocok apabila data yang digunakan menggunakan data *image*. Dalam CNN dibagi menjadi 2 bagian yaitu *feature extraction* dan *fully connected layer*, sebuah data *image* akan di lakukan ekstraksi fitur terlebih dahulu pada bagian *feature extraction* kemudian hasil ekstrasi fitur tersebut akan dilakukan klasifikasi pada *fully connected layer*. Pada penelitian ini menggunakan salah satu arsitektur CNN yaitu MobileNetV2. MobileNetV2 merupakan suatu arsitektur pengembangan dari arsitektur sebelumnya yaitu MobileNetV1. Hasil pengembangan ini menghasilkan jenis arsitektur CNN yang ringan serta hemat memori dalam melakukan komputasi (Sandler *et al.*, 2018). Penggunaan arsitektur MobileNetV2 ini diharapkan bisa melebihi akurasi dari penelitian sebelumnya yang mengunakan arsitektur VGG-16, ResNet-50, dan lain-lain. Serta nantinya penggunaan MobileNetV2 bisa diterapkan pada *smartphone* atau *drone* agar bisa melakukan pengenalan hama pada perangkat dengan memori yang kecil serta bisa memudahkan petani ketika ingin mencari tahu suatu jenis hama menggunakan *smartphone* atau *drone*. Selain itu, akan dilakukan penelitian lebih lanjut dengan penambahan beberapa metode agar mencapai hasil akurasi yang maksimal meskipun menggunakan memori yang kecil dengan arsitektur ini.

1.2 Rumusan Masalah

Dari permasalahan yang dipaparkan pada latar belakang, maka dapat dirumuskan menjadi sebagai berikut:

1. Bagaimana implementasi MobileNetV2 pada *dataset Insect Pest* (IP102)?
2. Bagaimana teknik pelatihan MobileNetV2 yang menghasilkan akurasi tertinggi pada *dataset Insect Pest* (IP102)?
3. Berapakah hasil akurasi dari penggunaan arsitektur MobileNetV2 pada *dataset Insect Pest*(IP102)?

1.3 Tujuan

Dari rumusan masalah yang telah dituliskan diatas, maka bisa menjadi tujuan sebagai berikut:

1. Mengetahui implementasi dari MobileNetV2 pada *dataset Insect Pest* (IP102).
2. Mengetahui teknik pelatihan MobileNetV2 yang menghasilkan akurasi tertinggi pada *dataset Insect Pest* (IP102).
3. Mengetahui hasil akurasi dari penggunaan arsitektur MobileNetV2 pada *dataset Insect Pest* (IP102).

1.4 Manfaat

Dengan adanya penelitian ini diharapkan dapat memberikan manfaat sebagai berikut:

1. Manfaat bagi penulis adalah dapat menambah wawasan dan pengetahuan pada bidang pengenalan jenis hama.
2. Manfaat bagi keilmuan yaitu memberikan kontribusi terhadap keilmuan dalam bidang agrikultural yaitu hama dan penyakit tanaman.
3. Manfaat bagi pembaca adalah hasil penelitian ini diharapkan dapat dijadikan sebagai referensi dalam penelitian selanjutnya dalam bidang yang pengenalan jenis hama atau penggunaan arsitektur yang sama.

1.5 Batasan Masalah

Pada penelitian ini diperlukan sebuah batasan masalah agar penelitian lebih terfokus, adapun batasan masalah tersebut yaitu:

1. *Dataset* yang digunakan adalah *dataset* IP102 yang terdiri dari 102 kelas.
2. Kelas yang ada pada dataset terbatas hanya 102 jenis nama latin hama serangga, diantaranya adalah *Rice Leaf Roller, Brown Plant Hopper, Wireworm*, dan lain-lain.
3. Arsitektur yang digunakan adalah arsitektur yang ringan serta cepat dalam melakukan komputasi.

1.6 Sistematika Pembahasan

Pada penelitian ini, sistematika pembahasan penelitian disusun sebagai berikut:

BAB 1 : PENDAHULUAN

Bab pendahuluan dibahas mengenai latar belakang, rumusan masalah, tujuan penelitian, batasan masalah, serta sistematika pembahasan penelitian "Pengenalan Jenis Hama Serangga Menggunakan Algoritma *Convolutional Neural Network* Dengan Arsitektur Mobilenetv2".

BAB 2 : LANDASAN KEPUSTAKAAN

Bab landasan kepustakaan dibahas mengenai landasan kepustakaan dan kajian teori yang dilakukan serta dikumpulkan oleh para peneliti sebelumnya yang mendasari topik dan penelitian ini.

BAB 3 : METODOLOGI PENELITIAN

Bab metodologi penelitian dibahas mengenai tahapan yang dilakukan pada penelitian ini agar dapat dibuat ulang pada penelitian-penelitian yang akan datang.

BAB 4 : PERANCANGAN

Bab perancangan berisi mengenai pemaparan dari perancangan algoritma serta perhitungan manual argoritma yang digunakan dalam penelitian.

BAB 5 : IMPLEMENTASI

Bab implementasi berisikan mengenai implementasi dari penelitian dan terdiri dari proses pelatihan serta pengujian dalam bentuk kode program.

BAB 6 : PENGUJIAN DAN ANALISIS HASIL

Bab ini berisi bahasan mengenai pengujian dan analisis hasil dari implementasi pada bab sebelumnya. Proses pengujian dilakukan dengan skenario yang sudah dirancang pada bab perancangan.

BAB 7 : PENUTUP

Bab ini dibahas mengenai kesimpulan yang didapatkan dari hasil pengujian dan saran dari kekuarangan yang ditemukan saat penelitian.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Landasan Kepustakaan

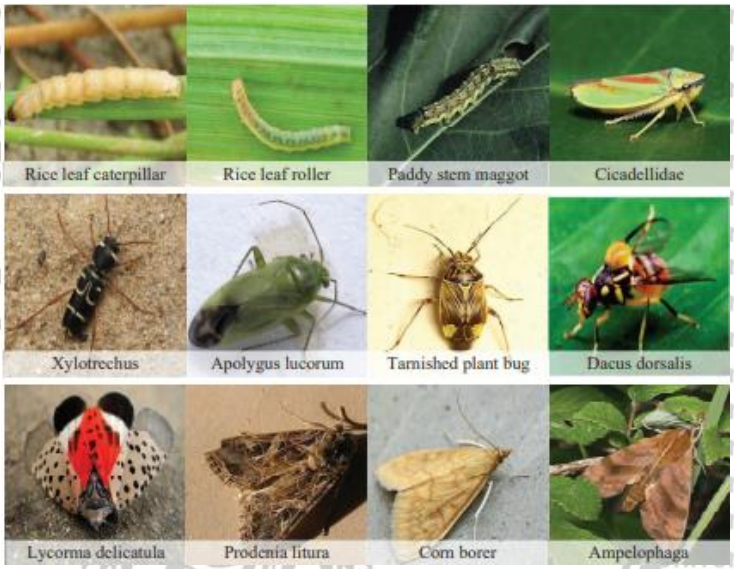
Sebelum memulai penelitian ini, ditemukan beberapa penelitian terdahulu yang membahas hal yang sama namun dalam studi kasus yang berbeda. Penelitian tersebut dijadikan referensi atau rujukan utama dalam melakukan penelitian ini. Penelitian dari Wu *et al.*, (2019) dibahas mengenai pengumpulan *dataset* hama serangga berskala besar. *Dataset* tersebut berisi 102 kelas dengan total *image* sebanyak 75.000 dan diberi nama *dataset* IP102 (Wu *et al.*, 2019). *Dataset* inilah yang digunakan dalam penelitian ini untuk melakukan pengenalan jenis hama serangga menggunakan arsitektur MobileNetV2.

Selanjutnya penelitian yang dijadikan rujukan adalah milik Nanni (2019). Pada penelitian tersebut dilakukan pengujian pada dua *dataset* dan salah satunya adalah *dataset* IP102, sama dengan yang digunakan pada penelitian ini. Pada penelitian tersebut diuji kemampuan dari lima arsitektur CNN yang berbeda yaitu AlexNet, GoogLeNet, ShuffleNet, MobileNetV2, dan DenseNet. Pada penelitian tersebut juga ditambahkan *saliency method* sebagai *input* data (Nanni, Maguolo dan Pancino, 2019). Ayan (2020) juga melakukan penelitian mengenai klasifikasi hama menggunakan dua *dataset* yang berbeda dan salah satunya adalah dataset IP102. Pada penelitian tersebut menguji beberapa arsitektur yang berbeda yaitu VGG-16, VGG-19, ResNet-50, Inception-V3, Xception, MobileNet, dan SqueezeNet. Dari pengujian beberapa arsitektur tersebut tiga diantaranya memiliki performa yang bagus, yaitu Inception-V3, Xception, dan MobileNet. Dari ketiga arsitektur tersebut kemudian digabungkan untuk membuat sebuah *ensemble* model yang diberi nama GAEnsemble (Ayan, Erbay dan Varçın, 2020).

Penelitian berikutnya yang dijadikan rujukan adalah milik Krizhevsky (2017). Pada penelitian tersebut dibahas model *Deep* CNN yang digunakan untuk 1,2 juta data *image* beresolusi tinggi dari ImageNet dan membaginya ke dalam 1000 kelas yang berbeda. Pada penelitian tersebut menggunakan sebuah metode *regularization* yang baru-baru ini ditemukan yaitu "*dropout*" untuk mengurangi *overfitting* dan terbukti sangat efektif (Krizhevsky, Sutskever dan Hinton, 2017). Penelitian selanjutnya adalah penelitian oleh Yudistira (2020). Pada penelitian tersebut digunakan arsitektur CNN yang ringan dalam melakukan komputasi namun memiliki hasil yang akurat untuk deteksi. Penelitian tersebut menguji kemampuan dari ShuffleNet, EfficientNet, dan ResNet-50 karena memiliki nilai parameter yang lebih kecil dibandingkan arsitektur CNN pada umumnya seperti VGGNet dan FullConv (Yudistira, Widodo dan Rahayudi, 2020).

2.2 *Dataset* Jenis Hama

Dataset yang digunakan dalam penelitian ini adalah *dataset* yang diberi nama IP102. *Dataset* ini dikumpulkan oleh Wu *et al.*, (2019), dengan total data kurang lebih sebanyak 75,000 citra hama dan terbagi kedalam 102 label yang berbeda. Seratus dua label yang ada tersebut merupakan kumpulan label jenis hama serangga yang ada di dunia. Dataset tersebut sudah terbagi dalam data latih sebanyak 45,095, data validasi sebanyak 7,508, dan data uji sebanyak 22,619 (Wu *et al.*, 2019). Dataset didapatkan dengan melakukan request langsung melalui link berikut https://github.com/xpwu95/IP102. Isi sebagian kecil dari data tersebut dapat dilihat pada Gambar 2.1.

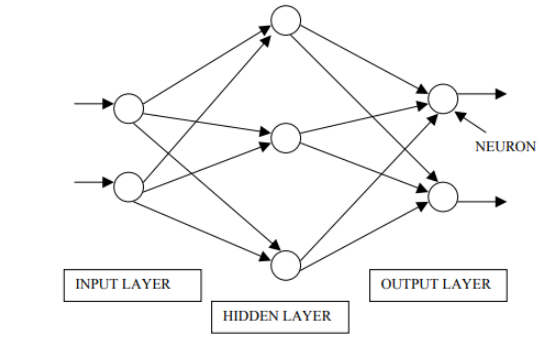


Gambar 2.1 *Dataset* IP102

Sumber: (Wu *et al.*, 2019)

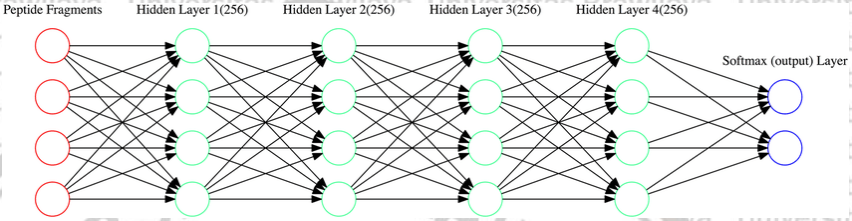
2.3 *Deep Learning*

Deep learning merupakan bidang yang merupakan bagian dari *machine learning*, bisa dikatakan bahwa *deep learning* adalah *subset* dari *machine learning*. *Deep learning* terinspirasi dari konsep jaringan saraf pada manusia yang memiliki banyak *neuron*, dimana setiap *neuron* tersebut berfungsi untuk memproses suatu informasi. Sama halnya dengan jaringan saraf pada manusia, pada *deep learning* lebih dikenal dengan jaringan saraf tiruan atau *artificial neural network*, pada jaringan saraf tiruan juga memiliki banyak *neuron* dan data-data yang ada akan dimasukkan ke dalam *neuron* tersebut untuk menghasilkan informasi yang bisa diolah. Jaringan saraf tiruan menggunakan suatu data untuk dijadikan sebagai *input neuron* dan memprosesnya kedalam sebuah *hidden layer* lalu kemudian menghasilkan sebuah *output* dari data yang telah diproses pada *hidden layer*. Sebuah jaringan saraf yang standar memiliki struktur *neuron* yang sederhana, namun pada *deep learning* susunan arsitektur jaringan saraf akan menjadi sangat kompleks dan dalam (Schmidhuber, 2015). Illustrasi perbedaan jaringan saraf tiruan dan deep learning dapat dilihat pada Gambar 2.2 dan Gambar 2.3.



Gambar 2.2 Ilustrasi Arsitektur Jaringan Saraf Tiruan Sederhana

Sumber: (Babu dan Shailesh, 2014)



Gambar 2.3 Ilustrasi Arsitektur Jaringan Saraf Pada *Deep Learning*

Sumber: (Wang *et al.*, 2017)

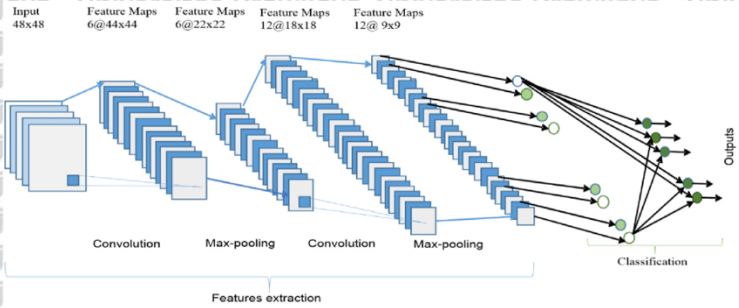
2.3.1 *Convolutional Neural Network* (CNN)

Convolutional neural network (CNN) merupakan salah satu metode yang ada pada *deep learning*. CNN termasuk pada *deep learning* karena kedalaman jaringannya. Algoritma *Convolutional Neural Network* (CNN) mirip dengan *Artificial Neural Network* (ANN) tradisional karena terdiri dari *neuron* yang dioptimalkan sendiri melalui pembelajaran (O'Shea dan Nash, 2015), perbedaan utamanya adalah CNN banyak diimplementasikan pada data citra atau *image*. CNN terdiri dari tiga *layer* atau biasa disebut dengan *building blocks* yaitu: konvolusi, *pooling*, dan *fully connected layer*. Dua yang pertama adalah konvolusi dan *pooling* yang akan melakukan ekstraksi fitur dan yang ketiga adalah *fully connected layer* yang akan digunakan untuk proses klasifikasi (Yamashita *et al.*, 2018).

Layer konvolusi berperan penting dalam CNN, karena pada *layer* ini dilakukan ekstraksi fitur yang akan membantu komputasi dilakukan oleh mesin. Berbeda dengan pada *machine learning* tradisional dimana proses ekstraksi fitur dilakukan oleh manusia atau biasa disebut *feature engineering*, pada CNN proses ini akan dilakukan oleh mesin secara otomatis. Persamaan yang digunakan pada konvolusi terdapat pada persamaan 2.1.

$$G(i,j) = (I * K)_{i,j} = \sum_m \sum_n i(i-m, j-n) K(m,n) \quad (2.1)$$

Keterangan:
G = Merupakan hasil konvolusi
K = *Kernel* yang akan digunakan
I = Data *input* (nilai *pixel* pada citra)



Gambar 2.4 Arsitektur *Convolutional Neural Network* (CNN)

Sumber: (Alom *et al.*, 2019)

Proses perhitungan pada CNN dilakukan dengan persamaan 2.1 dan pada prosesnya menggunakan langkah seperti pada Gambar 2.5. Bagian yang berwarna oranye merupakan filter konvolusi yang akan digeser ke sebelah kanan sampai pada kesemua pixel pada citra. Kemudian hasil konvolusi ditunjukkan pada warna merah di sisi kanan. Proses konvolusi akan terus dilakukan sampai akhir dari bagian pixel pada citra, seperti pada Gambar 2.5 disisi kanan.

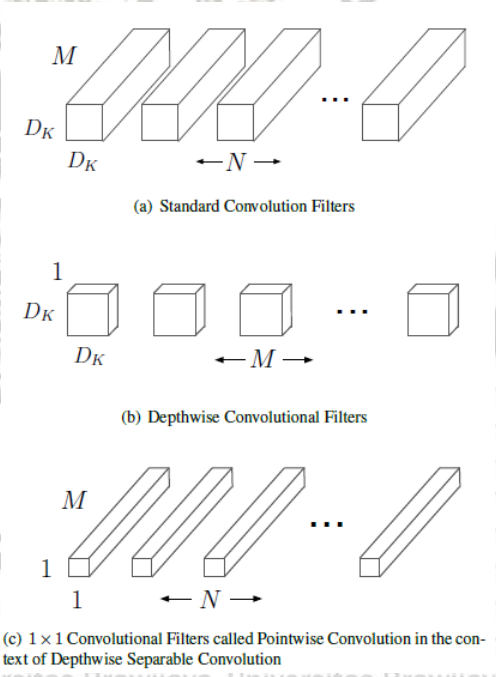
Gambar 2.5 Illustrasi Proses Pada CNN

Sumber: (Bansari, 2019)

2.3.2 MobileNetV2

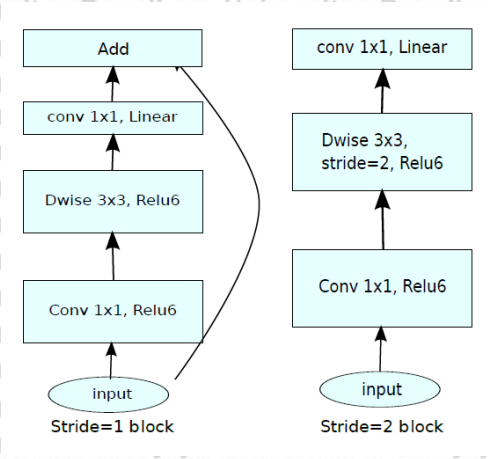
MobileNetV2 merupakan salah satu arsitektur dari CNN yang memiliki komputasi yang ringan. MobileNetV2 adalah arsitektur yang dikembangkan dari arsitektur sebelumnya yaitu MobileNetV1. Berbeda dengan arsitektur yang lain, kebanyakan arsitektur tersebut membutuhkan sumber daya dan komputasi yang besar untuk melakukan pemrosesan data, arsitektur MobileNet ini memang secara khusus dirancang untuk mobile dan dengan sumber daya yang terbatas (Sandler *et al.*, 2018). Versi awal MobileNet adalah MobileNetV1, pada versi awal ini Howard et al., (2017) menemukan jenis konvolusi yang dinamakan *depthwise separable convolution*. *Depthwise separable convolution* merupakan proses pembagian konvolusi pada arsitektur MobileNet. *Depthwise separable convolution* dibagi manjadi dua bagian yaitu *depthwise convolution* dan *pointwise convolution*. Pada prosesnya, *depthwise convolution* akan melakukan konvolusi pada setiap *channel* pada sebuah citra dan akan menghasilkan tiga *channel*. Selanjutnya hasil dari konvolusi tersebut akan disatukan lagi pada proses

pointwise convolution. Pada akhirnya *depthwise separable convolution* terbukti dapat membuat komputasi menjadi lebih ringan (Howard *et al.*, 2017). Pengembangan dari MobileNetV1 yaitu MobileNetV2 menambahkan hal baru yaitu *inverted residual* dan *linear bottleneck*. *Linear bottleneck* digunakan pada *depthwise separable convolution* untuk mengurangi jumlah fitur secara signifikan tanpa mengurangi kualitas fitur yang penting. Perbedaan mendasar dari MobileNet versi satu dan dua terdapat pada *convolutional block*. Pada versi satu hanya terdapat *depthwise separable convolution,* namun pada versi dua menambahkan *expansion layer* sebelum proses *depthwise separable convolution.* Fungsi dari *expansion layer* adalah untuk memperbesar jumlah *channel* sebelum dimasukkan pada *depthwise separable convolution*. Terdapat perubahan juga pada *pointwise convolution*, pada versi dua *pointwise convolution* digantikan oleh *projection layer*. Penggunaan *projection layer* ini akan mengurangi ukuran *channel* yang ada menjadi semakin kecil agar menjadi lebih ringan dalam proses komputasi. *Projection layer* biasa disebut juga dengan *bottleneck layer*, karena sifatnya yang melakukan pengurangan jumlah fitur. Penambahan *layer* pada MobileNetV2 ini akan membantu meningkatkan kemampuan model menjadi lebih efisien (Sandler *et al.*, 2018).



Gambar 2.6 Perbedaan Proses Konvolusi Standar dan Konvolusi Pada MobileNetV1

Sumber: (Howard *et al.*, 2017)



Gambar 2.7 *Convolutional Block* Pada MobileNetV2

Sumber: (Sandler *et al.*, 2018)

2.3.3 *Batch Normalization*

Batch normalization merupakan salah satu teknik dalam *deep learning* yang digunakan untuk melakukan normalisasi. *Batch normalization* ditempatkan pada layer jaringan dan pada banyak kasus berhasil meningkatkan akurasi dan mempercepat proses pelatihan. Kecenderungan untuk membuat akurasi meningkat dan mempercepat proses pelatihan tersebut, membuat *batch normalization* menjadi salah satu teknik yang sering digunakan dalam *deep learning* (Bjorck *et al.*, 2018). Pada penelitian milik Sergey Iofee (2015), terbukti bahwa *batch normalization* membuat hasil yang lebih baik dari klasifikasi citra pada ImageNet serta menggunakan proses pelatihan *(training step)* yang lebih sedikit (Ioffe dan Szegedy, 2015). Proses pada batch normalization memiliki empat tahap, yang pertama adalah perhitungan *mini batch mean*, kemudian dilanjutkan dengan perhitungan *mini batch variance*, lalu tahap selanjutnya adalah proses normalisasi, dan yang terakhir adalah proses *scale* dan *shift*. Proses tersebut dituliskan pada persamaan 2.2, 2.3, 2.4, dan 2.5.

$$\mu B = \frac{1}{m} \sum_{i=1}^{m} x_i \quad (2.2)$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu B)^2 \quad (2.3)$$

$$\widehat{x_i} = \frac{x_i - \mu B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (2.4)$$

$$y_i = \gamma \widehat{x_i} + \beta \quad (2.5)$$

Keterangan:

μB = *Mini Batch Mean*

σ_B^2 = *Mini Batch Variance*

m = Jumlah data pada setiap *mini batch*

$\hat{x}_i$ = Normalisasi

ϵ = Epsilon

γ = Gamma

β = Beta

y_i = *Scale* dan *Shift*

2.3.4 Augmentasi Data

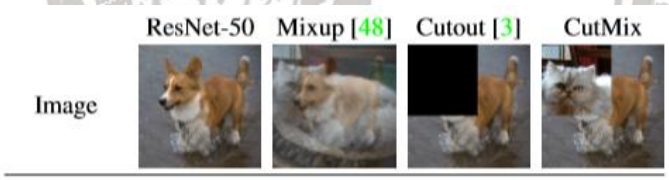
Augmentasi data adalah suatu proses memodifikasi sebuah data citra sehingga mesin akan mendeteksi gambar yang telah dimodifikasi tersebut adalah gambar yang berbeda. Hal ini tentunya akan menambah data dan membuat mesin belajar lebih banyak dari gambar atau citra yang berbeda-beda. Augmentasi data sangat layak dilakukan pada data citra atau *image*, karena bisa meningkatkan kualitas dari data. Augmentasi data terdiri dari beberapa metode, ada beberapa augmentasi data yang sering digunakan yaitu *horizontal flip*, *vertical flip*, *random crop*, *zoom in*, *image rotation*, dan lain-lain. Tentunya penerapan berbagai metode augmentasi data ini akan mengatasi permasalahan umum pada *deep learning* yaitu *data hungry*, karena masing-masing dari citra atau *image* yang ada akan di buat data baru sesuai metode augmentasinya. Namun perlu diketahui juga bahwa tidak semua data *image* atau citra harus dilakukan augmentasi dengan semua metode. Ada beberapa *image* atau citra yang harusnya tidak dilakukan augmentasi data dengan metode tertentu, contohnya *image* gedung, jika dilakukan *horizontal flip* atau *vertical flip* maka hasil dari Gedung tersebut sudah tidak sesuai dengan gambar asli, jadi untuk contoh kasus tersebut seharusnya tidak menggunakan *horizontal flip* atau *vertical flip*. Beberapa contoh hasil augmentasi data ditunjukkan pada Gambar 2.8 dan Gambar 2.9.

Gambar 2.8 Illustrasi *Random Rotation*

Gambar 2.9 Illustrasi *Random Resize Crop*

2.3.4.1 CutMix

Cutmix merupakan salah satu metode augmentasi data yang pada prosesnya melakukan penghapusan suatu bagian pada citra dan menggantikan dengan bagian citra yang lain. *Cutmix* akan menghasilkan satu diantara dua citra. *Cutmix* diusulkan untuk menggantikan metode *dropout*, dimana pada *dropout* menghilangkan suatu bagian dari citra dan menggantikannya dengan warna hitam. Hal ini tentunya akan menghilangkan informasi dari citra tersebut jika bagian yang hilang terlalu banyak, oleh karena itu daripada menghilangkan bagian pada citra lebih baik mengganti bagian yang hilang tersebut dengan citra yang lain agar suatu model bisa lebih fokus pada objek (Yun *et al.*, 2019).



Gambar 2.10 Illustrasi *CutMix*

Sumber: (Yun *et al.*, 2019)

2.3.5 Fungsi Aktivasi

Fungsi aktivasi merupakan suatu fungsi pada jaringan saraf tiruan yang digunakan untuk melakukan pemrosesan dari data *input* sehingga akan menghasilkan suatu *output*. Suatu data akan masuk melalui *input layer* dan ketika masuk ke *layer* berikutnya maka akan diproses dan dilakukan aktivasi menggunakan fungsi aktivasi, begitu pula pada *layer-layer* selanjutnya. Ada banyak jenis fungsi aktivasi, namun yang paling sering digunakan pada *deep learning* adalah ReLU, Softmax, sigmoid, dan Tanh.

2.3.5.1 *Rectified Linear Unit 6* (ReLU6)

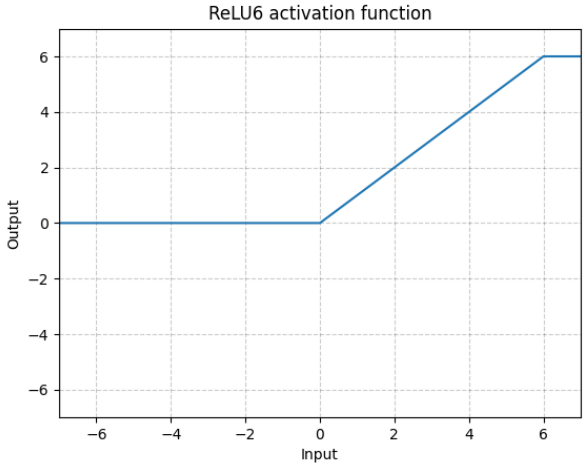
ReLU6 adalah modifikasi dari aktivasi sebelumnya yaitu ReLU. Pada ReLU6 menggunakan batas maksimal suatu nilai keluaran yaitu 6. Berbeda dengan ReLU yang tidak memberikan batas maksimal dan hal ini akan membuat nilai keluaran

aktivasi semakin meningkat yang menyebabkan sebuah jaringan saraf semakin berat. ReLU6 digunakan ketika kita ingin tetap mempertahankan sebuah jaringan saraf tetap ringan dan hemat memori. Persamaan pada fungsi aktivasi ReLU6 dituliskan pada persamaan 2.6 dan grafik fungsi aktivasi ReLU terdapat pada Gambar 2.11.

$$R(x) = \min(\max(0, x), 6) \quad (2.6)$$

Keterangan:

x = masukan dari fungsi aktivasi



Gambar 2.11 Ilustrasi Fungsi Aktivasi ReLU6

Sumber: (PyTorch, 2019)

2.3.6 Fungsi *Loss*

Fungsi *loss* merupakan suatu fungsi yang digunakan untuk menghitung selisih besarnya nilai prediksi dan nilai sebenarnya pada sebuah data. Pada klasifikasi, fungsi *loss* yang sering digunakan adalah *cross entropy*. Persamaan dari fungsi *loss cross entropy* dapat dilihat pada persamaan 2.7.

$$L = \sum_j y^{(j)} log\sigma(o)^{(j)} \quad (2.7)$$

Keterangan:

L = Fungsi *Loss*

y = kelas target

o = keluaran model

j = data pada setiap label

2.3.7 *Sparse Regularization*

Sparse regularization atau sering disebut juga *sparness* adalah salah satu jenis metode regularisasi. *Sparness* pertama kali ditemukan untuk mencegah peningkatan jumlah parameter dari sebuah model. Ada beberapa jenis *sparness*,

yang pertama adalah *sparness* pada bagian bobot yang sering dikenal dengan nama *weight decay*, *sparness* pada keluaran atau *output hidden unit*, dan yang terakhir adalah *sparness* pada input fungsi aktivasi ReLU yang diusulkan oleh Ide dan Kurita (2017). Penggunaan *sparness* pada input fungsi aktivasi ReLU dapat mencegah peningkatan nilai yang terlalu berlebihan pada *output* fungsi aktivasi ReLU. Peningkatan nilai yang berlebihan akan menyebabkan *overfitting,* oleh karena itu perlu dilakukan pencegahan kenaikan nilai yang terlalu berlebihan sehingga dapat memberikan efek *batch normalization* dan meningkatkan kemampuan generalisasi pada model (Ide dan Kurita, 2017). Persamaan yang digunakan untuk penambahan *sparse regularization* atau *sparness* pada *input* ReLU dapat dilihat pada persamaan 2.8

$$E = L + \lambda \sum_k S(hk) \quad (2.8)$$

Keterangan:

E = Error total / Loss total

L = Fungsi Loss

λ = Nilai Lambda

k = nilai setiap output fungsi aktivasi

S(hk) = *Output* Fungsi Aktivasi

BAB 3 METODOLOGI PENELITIAN

3.1 Tipe Penelitian

Penelitian ini merupakan penelitian bertipe non implementatif analitik. Penelitian tipe ini dilakukan untuk menjelaskan hubungan antar elemen dalam objek yang sedang diteliti. Kegiatan pada penelitian mengutamakan penggalian informasi pada hal yang sedang diteliti dan digunakan untuk mengidentifikasi elemen-elemen penting pada objek. Pada kasus penelitian ini meneliti penggunaan arsitektur MobileNetV2 pada citra hama dan dilakukan identifikasi terhadap hasil yang didapatkan agar mencapai hasil akurasi yang maksimal ketika melakukan pelatihan menggunakan arsitektur MobileNetV2 pada citra hama.

3.2 Strategi dan Rancangan Penelitian

3.2.1 Strategi Penelitian

Penelitian ini menggunakan strategi studi literatur dalam mencari sumber terdahulu yang relevan dan studi eksperimen untuk melakukan peningkatan akurasi dari model yang digunakan. Eksperimen bisa dilakukan dengan berbagai cara seperti penggunaan augmentasi data dan jenis fungsi aktivasi tertentu, bisa juga dilakukan dengan menerapkan *transfer learning* atau yang lainnya.

3.2.2 Lokasi Penelitian

Penelitian ini berlokasikan di Fakultas Ilmu Komputer Universitas Brawijaya.

3.2.3 Metode Pengumpulan Data

Pada penelitian ini, data dikumpulkan dengan cara melakukan permintaan data pada Xiaoping Wu, seorang peneliti yang melakukan pengumpulan dataset ini dan dicantumkan pada papernya dan diberi nama dataset IP102 (Wu *et al.*, 2019). Dataset ini terdiri dari kurang lebih 75,000 citra dan dibagi menjadi 102 kelas. Seluruh data citra tersebut dipecah menjadi data latih, data uji, dan data validasi. Sampel dari dataset tersebut ada di landasan kepustakaan pada Gambar 2.1

3.2.4 Metode Analisis Hasil

Penelitian ini melakukan analisis terhadap hasil dari pelatihan yang dilakukan menggunakan arsitektur MobileNetV2. Selain dari pelatihan, pengujian juga dilakukan menggunakan beberapa *hyperparameter* yang ada untuk mendapatkan hasil akurasi terbaik dan komputasi yang ringan.

3.2.5 Peralatan Pendukung

Penelitian ini menggunakan *tools* berupa perangkat keras seperti yang ditunjukkan pada Tabel 3.1 dan perangkat lunak yang ditunjukkan pada Tabel 3.2.

Tabel 3.1 Spesifikasi Perangkat Keras

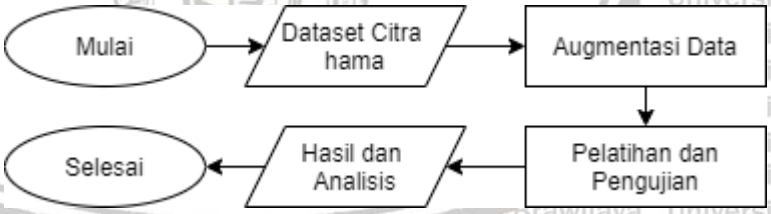
Komponen	Keterangan
CPU	Intel Core i3-2310M
GPU	Intel HD Graphic 3000
RAM	6 GB DDR3-665.1 MHz
Media Penyimpanan	SSD SATA 2.5 500GB

Tabel 3.2 Spesifikasi Perangkat Lunak

Jenis	Keterangan
Sistem Operasi	Windows 10
Bahasa Pemrograman	Python
IDE	Google Colaboratory
Pustaka Pemrograman	Pytorch

3.2.6 Metode Penelitian

Penelitian ini menggunakan arsitektur MobileNetV2 pada klasifikasi citra hama. Alur penelitian dapat dilihat pada Gambar 3.1



Gambar 3.1 Alur Metode Penelitian

3.2.7 Augmentasi Data

Dataset yang sudah dikumpulkan berjumlah kurang lebih 75.000 data. Jumlah ini masih bisa ditingkatkan lagi menggunakan augmentasi data. Augmentasi data pada citra membuat setiap citra memiliki beberapa versi yang sudah diaugmentasi, hal ini membuat jumlah data semakin bertambah.

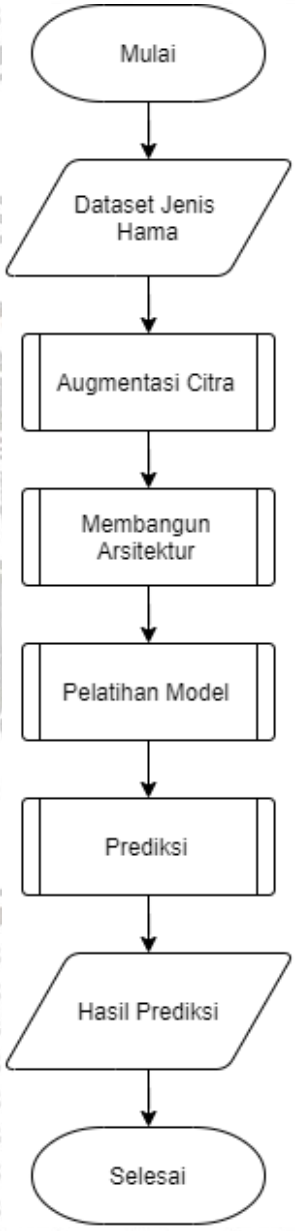
3.2.8 Pelatihan dan Pengujian

Dataset yang sudah dilakukan augmentasi selanjutnya akan dilakukan proses *training* atau pelatihan menggunakan arsitektur MobileNetV2 pada CNN dan dilakukan proses klasifikasi. Proses pengujian dilakukan setelah proses pelatihan selesai, kemudian diberikan sebuah data baru yang tidak ada pada data latih dan akan dilakukan perhitungan akurasi dari data uji tersebut.

BAB 4 PERANCANGAN

4.1 Perancangan Algoritma

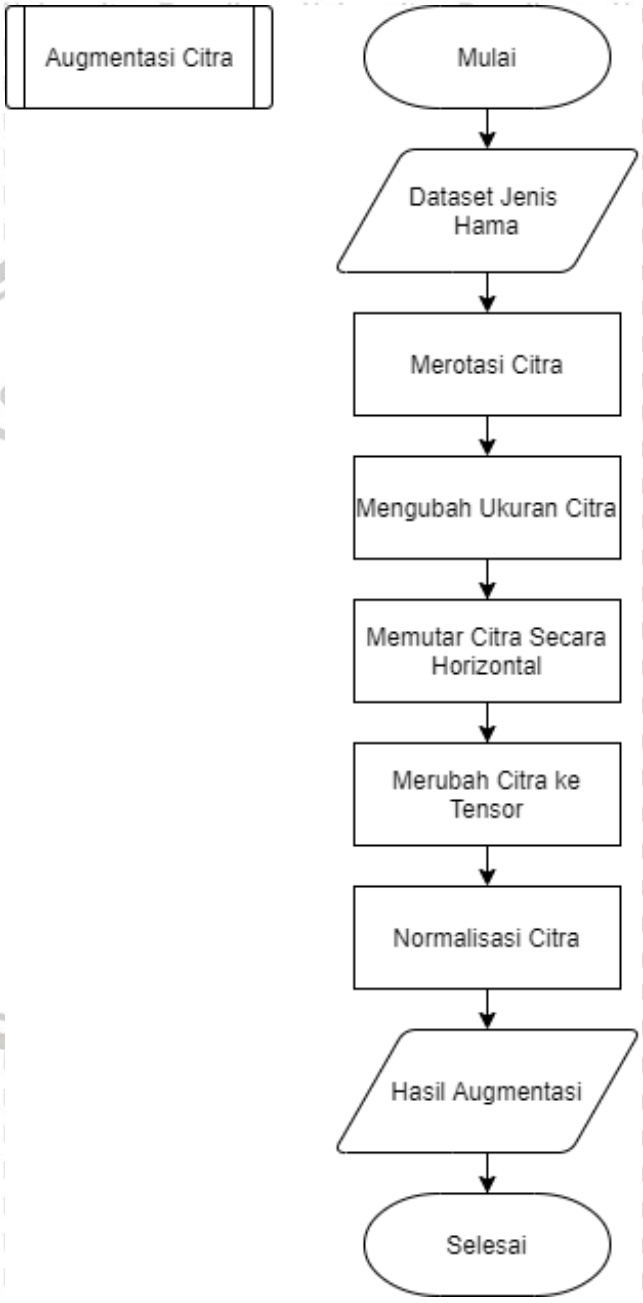
Perancangan algoritma pengenalan jenis hama ini dimulai dengan memasukkan *dataset* jenis hama yang sudah disediakan ke dalam sebuah model atau arsitektur. Sebelum dimasukkan pada arsitektur, *dataset* terlebih dahulu dilakukan augmentasi lalu dilanjutkan dimasukkan kedalam arsitektur. Diagram alir dari algoritma pengenalan jenis hama ditunjukkan pada Gambar 4.1



Gambar 4.1 Diagram Alir Sistem

4.1.1 Diagram Alir Augmentasi Citra

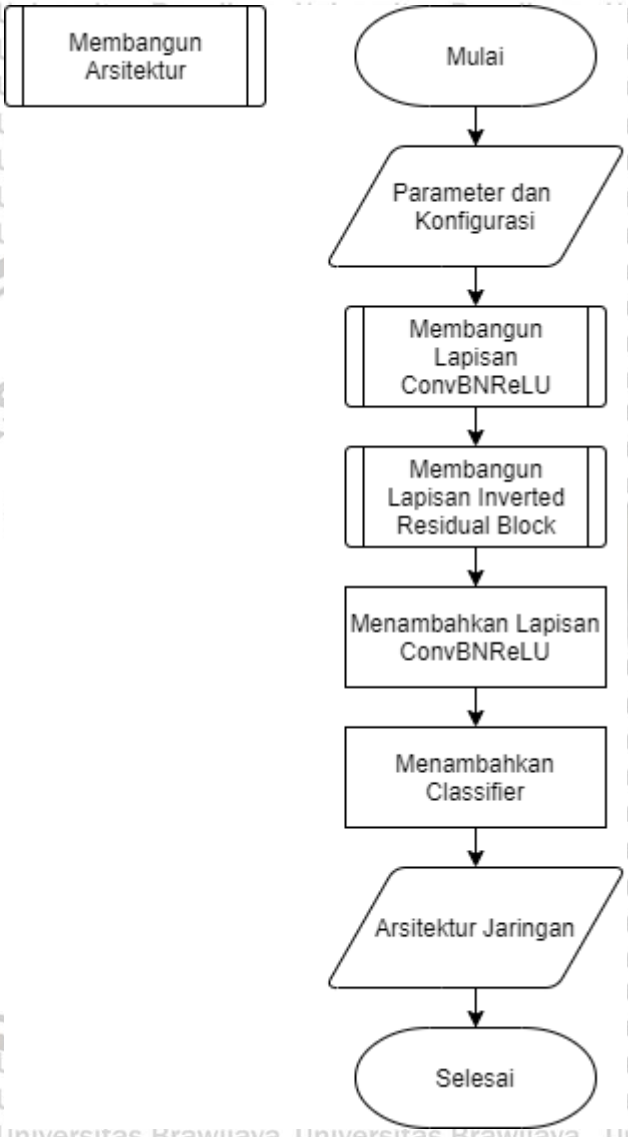
Augmentasi citra yang digunakan dalam penelitian ini adalah merotasi citra (*random rotation)*, merubah ukuran citra *(resize)*, memutar citra secara horizontal (*horizontal flip*), merubah citra dalam bentuk tensor, dan normalisasi citra. Diagram alir dari augmentasi citra ditunjukkan pada Gambar 4.2



Gambar 4.2 Diagram Alir Augmentasi Citra

4.1.2 Diagram Alir Membangun Arsitektur

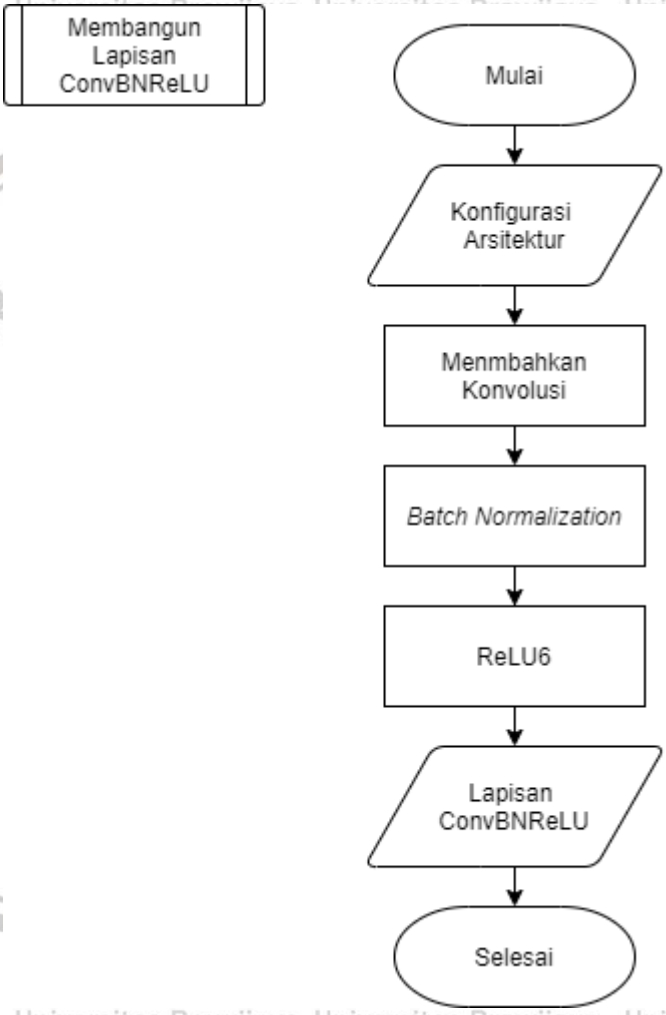
Proses pembangunan arsitektur berdasarkan konfigurasi dan susunan layer pada arsitektur MobileNetV2. Pada MobileNetV2 tersusun atas 18 layer *inverted residual* yang didalamnya terdapat *convolutional block*. Diagram alir dari pembangunan arsitektur MobileNetV2 ditunjukkan pada Gambar 4.3



Gambar 4.3 Diagram Alir Pembangunan Arsitektur

4.1.2.1 Diagram Alir Pembangunan Lapisan *Convolutional Block*

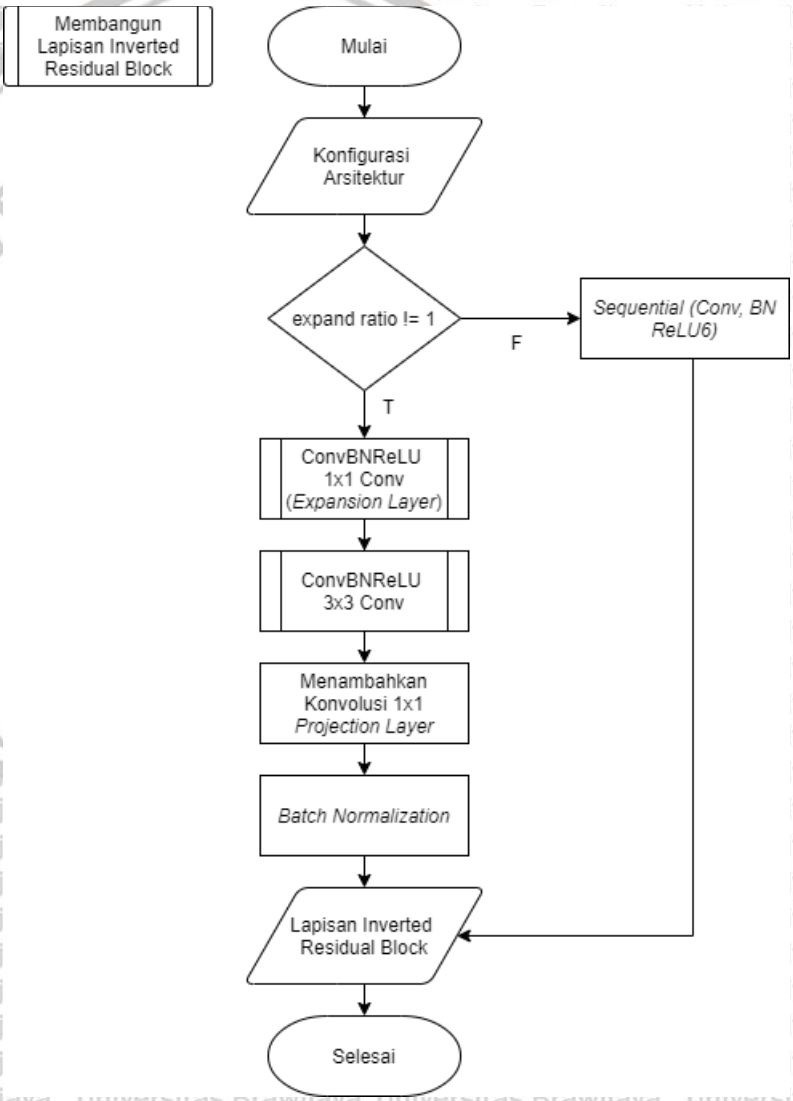
Pada arsitektur MobileNetV2, terdapat lapisan *convolutional block* karena pada *block* ini terdiri dari lapisan konvolusi, *batch normalization* dan aktivasi ReLU6. Ketiga bagian ini akan diulang secara terus menerus dan digunakan di berbagai macam lapisan serta juga akan digunakan dalam lapisan *inverted residual*. Sehingga untuk memberikan efektifitas, dibuatlah *block* tersendiri agar bisa dipakai berulang kali dan mengurangi redundansi. Pembangunan lapisan *convolutional block* ditunjukkan pada Gambar 4.4



Gambar 4.4 Diagram Alir Pembangunan Lapisan *Convolutional Block*

4.1.2.2 Diagram Alir Pembangunan Lapisan *Inverted Residual*

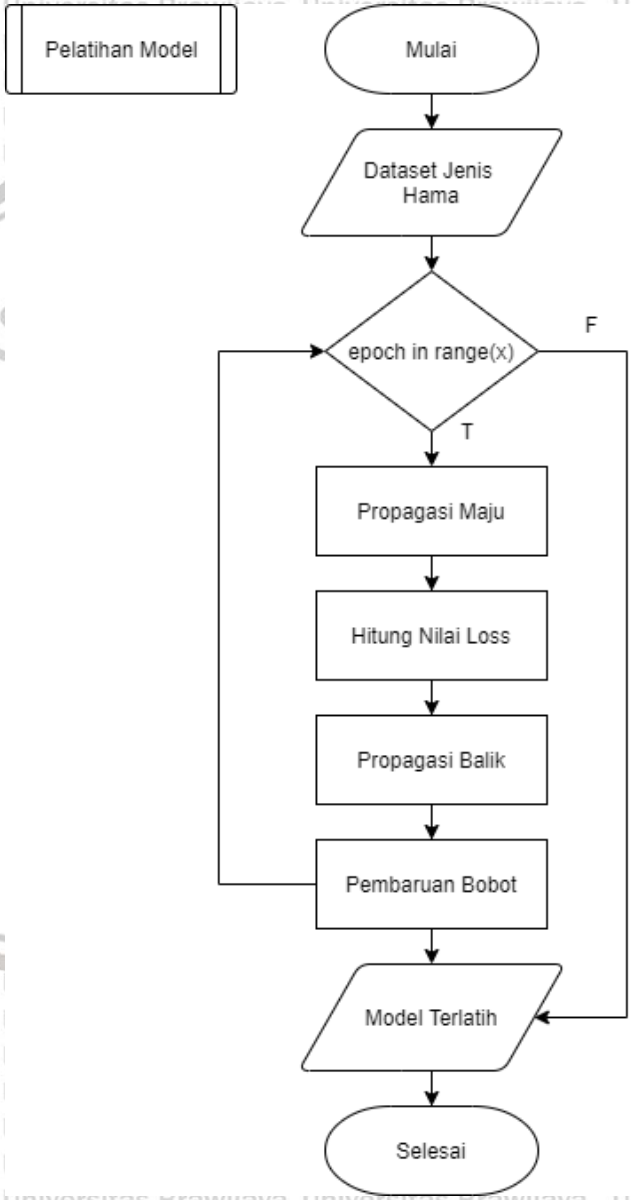
Lapisan *inverted residual* pada MobileNetV2 hampir sama dengan *residual network* (ResNet) yang menerapkan *skip connection*. Perbedaannya adalah pada isi *block* tersebut. Pada *inverted residual* MobileNetV2 didalamnya terdapat kumpulan *convolutional block* yang sudah dibangun pada tahap sebelumnya. Pada *inverted residual* terdiri atas tiga *convolutional block*. *Convolutional block* yang pertama adalah 1x1 *convolutional block* yang biasa disebut dengan *expansion layer*, kemudian 3x3 *convolutional block* yang disebut juga *depthwise separable convolution*, dan yang terakhir adalah 1x1 *projection layer* yang juga menggunakan konvolusi tanpa menggunakan fungsi aktivasi dan menggunakan *batch normalization*. Lapisan *inverted residual* ini akan diulang sebanyak 18 kali sehingga akan menjadi 18 lapisan secara keseluruhan. Diagram alir pembangunan lapisan inverted residual ditunjukkan pada Gambar 4.5



Gambar 4.5 Diagram Alir Pembangunan Lapisan *Inverted Residual*

4.1.3 Diagram Alir Pelatihan Model

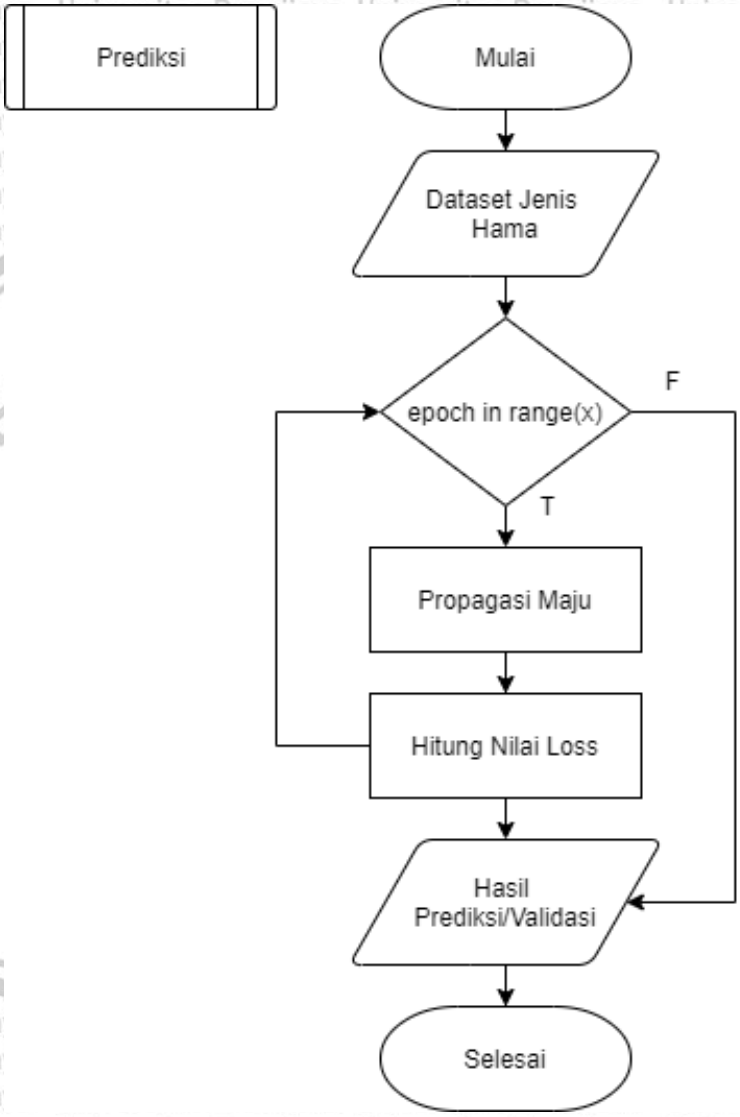
Pada pelatihan model, dilakukan iterasi sebanyak *epoch* yang telah ditentukan. Dalam proses iterasi tersebut terdapat beberapa proses, yang meliputi *feed forward* (propagasi maju), menghitung nilai loss, *backpropagation* (propagasi balik), kemudian *update* bobot. Proses inilah yang diulang secara terus menerus sampai jumlah *epoch* memenuhi. Diagram alir dari pelatihan model ditunjukkan pada Gambar 4.6



Gambar 4.6 Diagram Alir Pelatihan Model

4.1.4 Diagram Alir Prediksi atau Validasi

Pada proses prediksi atau validasi, dilakukan setelah proses pelatihan pada data latih. Secara garis besar prosesnya sama, hanya saja pada prediksi atau validasi tidak dilakukan propagasi balik dan *update* bobot. Proses hanya berhenti sampai menghitung nilai *loss*. Diagram alir dari prediksi atau validasi ditunjukkan pada Gambar 4.7



Gambar 4.7 Diagram Alir Prediksi atau Validasi

4.2 Perhitungan Manual

Pada tahap selanjutnya dilakukan proses perhitungan manual terhadap contoh data kecil dan arsitekturnya yang sederhana. Perhitungan manual ini bertujuan untuk melakukan pengujian konsep dan implementasi. Data masukan yang digunakan ditunjukkan pada Gambar 4.8, Gambar 4.9, dan Gambar 4.10 dengan satu gambar yang terdiri dari tiga channel RGB. Target dari proses klasifikasi adalah kelas satu dan arsitektur yang digunakan untuk proses perhitungan manual ini ditunjukkan pada Gambar 4.11.

1	0	1	0	1
0	0	1	1	1
1	1	0	0	1
0	1	0	1	1
0	0	1	1	1

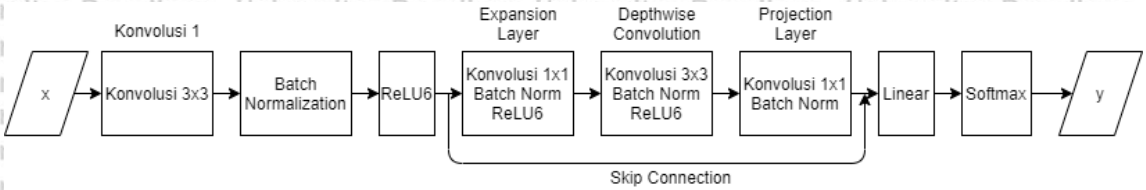
Gambar 4.8 Data Masukkan *Channel* 1

1	0	1	0	1
0	0	-1	1	1
1	1	0	0	-1
-1	1	0	1	0
0	-1	2	-1	0

Gambar 4.9 Data Masukkan *Channel* 2

0	-1	1	1	1
0	0	-1	1	1
1	0	0	-1	-1
-1	1	0	1	0
0	-1	-1	-1	0

Gambar 4.10 Data Masukkan *Channel* 3



Gambar 4.11 Arsitektur Jaringan Perhitungan Manual

4.2.1 Propagasi Maju

Propagasi maju atau yang biasa disebut *forward propagation* merupakan proses pelatihan dengan memasukkan data ke dalam model kemudian akan dilakukan prediksi. Setelah proses propagasi maju selesai, dilakukan perhitungan nilai *error* terhadap prediksi dengan target yang sebenarnya.

4.2.1.1 Propagasi Maju Pada Lapisan Konvolusi 1

Lapisan Konvolusi 1 merupakan lapisan pertama setelah data masukan. Operasi yang ada pada lapisan ini adalah konvolusi 3x3, kemudian dilanjutkan dengan perhitungan *batch normalization*, dan diakhir dengan aktivasi ReLU6. Pada lapisan ini terdapat satu buah filter dengan tiga *channel* berukuran 3x3. Nilai dari filter tersebut atau yang sering disebut bobot ditunjukkan pada Gambar 4.12 sampai Gambar 4.14.

2	1	1
0	-2	1
1	-1	0

Gambar 4.12 Bobot Filter Konvolusi 1 (*Channel* 1)

-1	1	0
0	-2	1
1	-1	1

Gambar 4.13 Bobot Filter Konvolusi 1 (*Channel* 2)

-1	-1	1
0	-2	-1
1	-1	0

Gambar 4.14 Bobot Filter Konvolusi 1 (*Channel* 3)

Proses konvolusi dilakukan dengan melakukan perhitungan pada input channel dan dikalikan dengan bobot dari masing-masing filter. Perhitungan tersebut dilakukan pada setiap channel. Contoh perhitungan dapat dilihat pada Gambar 4.15 dan Gambar 4.16 dan ditunjukkan oleh warna biru.

1	0	1	0	1
0	0	1	1	1
1	1	0	0	1
0	1	0	1	1
0	0	1	1	1

Gambar 4.15 Perhitungan Konvolusi Input Channel 1

2	1	1
0	-2	1
1	-1	0

Gambar 4.16 Perhitungan Konvolusi Bobot Filter Channel 1

Pada perhitungan tersebut dapat dilakukan perhitungan dengan persamaan 2.1 sehingga. Proses perhitungan dilakukan menggunakan perkalian *dot product.* Hasil dari perhitungan konvolusi dapat dilihat pada Gambar 4.17 yang diberi warna hijau. Kemudian konvolusi akan berlanjut dengan menggeser proses perhitungan pada *input channel* ke sebelah kanan dengan pergeseran satu langkah, atau biasa disebut dengan *stride* satu. Dan seterusnya sampai mendapatkan kesemua nilai pada setiap *pixel*.

4	1	2
-2	3	4
1	2	0

Gambar 4.17 Hasil Konvolusi Channel 1

Hasil dari semua proses konvolusi pada masing-masing channel akan digabungkan sehingga menghasilkan feature map yang dapat dilihat pada Gambar 4.18.

6	8	-4
-9	8	7
0	-4	-1

Gambar 4.18 *Feature Map* Hasil Konvolusi 1

4.2.1.2 Propagasi Maju Pada Lapisan *Batch Normalization*

Lapisan *batch normalization* adalah lapisan yang digunakan untuk melakukan normalisasi dari hasil *feature map* konvolusi 1. Proses perhitungan yang dilakukan adalah dengan menghitung telebih dahulu *mini batch mean* dari hasil *feature map* pada konvolusi satu menggunakan persamaan 2.2. Selanjutnya adalah

menghitung *mini batch variance* pada hasil *feature map* konvolusi satu dengan menggunakan persamaan 2.3. Hasil dari perhitungan *mini batch mean* dan *mini batch variance* ditunjukkan pada Gambar 4.19. Setelah nilai *mini batch mean* dan *mini batch variance* ditemukan, bisa dilanjutkan dengan perhitungan normalisasi yang didapatkan dengan perhitungan nilai pada setiap data *feature map* konvolusi satu dengan nilai *mini batch mean* dan *mini batch variance*. Proses perhitungan menggunakan persamaan 2.4 dan hasil perhitungan dapat dilihat pada Gambar 4.20.

Mini Batch Mean	***Mini Batch Variance***
1,2222	34,849

Gambar 4.19 Hasil Perhitungan *Mini Batch Mean* dan *Variance*

0,80945	1,148289	-0,88475
-1,73185	1,148289	0,978869
-0,20707	-0,88475	-0,37649

Gambar 4.20 Hasil *Normalize Feature Map* Konvolusi Satu

Setelah dilakukan mendapatkan hasil *normalize* pada *feature map*, bisa dilakukan *scale and shift (batch normalization)* dengan menggunakan persamaan 2.5. Hasil dari *batch normalization* tersebut ditunjukkan pada Gambar 4.21.

0,80945	1,148289	-0,88475
-1,73185	1,148289	0,978869
-0,20707	-0,88475	-0,37649

Gambar 4.21 Hasil Propagasi Maju Pada Lapisan *Batch Normalization*

4.2.1.3 Propagasi Maju Pada Lapisan ReLU6

Setelah proses propagasi maju pada lapisan *batch normalization*, proses selanjutnya adalah melakukan aktivasi pada hasil *batch normalization* tersebut. Proses aktivasi dilakukan menggunakan fungsi aktivasi ReLU6. Hasil dari aktivasi tersebut dapat dilihat pada Gambar 4.22.

0,80945	1,148289	0
0	1,148289	0,978869
0	0	0

Gambar 4.22 Hasil Propagasi Maju Pada Lapisan ReLU6

4.2.1.4 Propagasi Maju Pada Lapisan *Expansion Inverted Residual Block*

Lapisan ini merupakan lapisan pertama pada blok inverted residual. Pada lapisan ini terjadi konvolusi menggunakan tiga filter berukuran 1x1. Hasil konvolusi tersebut akan dilakukan batch normalization dan selanjutnya dilakukan aktivasi

menggunakan fungsi aktivasi ReLU6. Data masukan pada lapisan ini adalah hasil keluaran dari propagasi maju pada lapisan ReLU6 konvolusi satu yang terdapat pada Gambar 4.22. Data masukan tersebut dilakukan konvolusi dengan tiga filter berukuran 1x1 dengan bobot yang ada pada Gambar 4.23 dan hasil konvolusi menghasilkan tiga channel feature map seperti pada Gambar 4.24, Gambar 4.25, dan Gambar 4.26.

1,2		1		1,1

Gambar 4.23 Bobot Filter Lapisan *Expansion* (3 filter 1 *Channel*)

0,97134	1,377947	0
0	1,377947	1,174643
0	0	0

Gambar 4.24 *Feature Map* Hasil Konvolusi Lapisan *Expansion* (*Channel* 1)

0,80945	1,148289	0
0	1,148289	0,978869
0	0	0

Gambar 4.25 *Feature Map* Hasil Konvolusi Lapisan *Expansion* (*Channel* 2)

0,80945	1,263118	0
0	1,263118	1,076756
0	0	0

Gambar 4.26 *Feature Map* Hasil Konvolusi Lapisan *Expansion* (*Channel* 3)

4.2.1.5 Propagasi Maju Pada *Batch Normalization* Lapisan *Expansion*

Hasil konvolusi dari tahap sebelumnya dilakukan normalisasi pada tahap ini. Hasil normalisasi tersebut dapat dilihat pada Gambar 4.27, Gambar 4.28, dan Gambar 4.29.

0,689058	1,34569	-0,87956
-0,87956	1,34569	1,01737
-0,87956	-0,87956	-0,87956

Gambar 4.27 Hasil *Batch Normalization* Lapisan *Expansion* (*Channel* 1)

0,689054	1,345682	-0,87956
-0,87956	1,345682	1,01737
-0,87956	-0,87956	-0,87956

Gambar 4.28 Hasil *Batch Normalization* Lapisan *Expansion* (*Channel* 2)

0,689054	1,345682	-0,87956
-0,87956	1,345682	1,01737
-0,87956	-0,87956	-0,87956

Gambar 4.29 Hasil *Batch Normalization* Lapisan *Expansion* (*Channel* 3)

4.2.1.6 Propagasi Maju Pada Fungsi Aktivasi ReLU6 Lapisan *Expansion*

Pada tahap ini dilakukan proses aktivasi pada setiap nilai pada hasil batch normalization yang sudah dihitung di tahap sebelumnya. Hasil proses aktivasi dapat dilihat pada Gambar 4.30, Gambar 4.31, dan Gambar 4.32.

0,689054	1,345682	0
0	1,345682	1,01737
0	0	0

Gambar 4.30 Hasil Aktivasi Lapisan *Expansion* (*Channel* 1)

0,689054	1,345682	0
0	1,345682	1,01737
0	0	0

Gambar 4.31 Hasil Aktivasi Lapisan *Expansion* (*Channel* 2)

0,689054	1,345682	0
0	1,345682	1,01737
0	0	0

Gambar 4.32 Hasil Aktivasi Lapisan *Expansion* (*Channel* 3)

4.2.1.7 Propagasi Maju Pada Lapisan *Depthwise Convolution*

Hasil dari aktivasi ditahap sebelumnya dimasukkan menjadi input pada lapisan *depthwise convolution*. Pada lapisan ini akan dilakukan konvolusi menggunakan satu filter tiga channel berukuran 3x3 seperti pada Gambar 4.33, Gambar 4.34, dan Gambar 4.35, dan sebelum dilakukan konvolusi data masukan dilakukan *padding* terlebih dahulu dengan *zero padding* agar ukuran citra yang awalnya berukuran 3x3 menjadi 5x5. Hasil konvolusi berupa citra tiga channel berukuran 3x3 dan dapat dilihat pada Gambar 4.36, Gambar 4.37, dan Gambar 4.38.

2	1	1
0	-2	1
1	-1	0

Gambar 4.33 Bobot Pada Filter Lapisan *Depthwise* (*Channel* 1)

-1	1	0

0	-2	1
1	-1	1

Gambar 4.34 Bobot Pada Filter Lapisan *Depthwise* (*Channel* 2)

-1	-1	1
0	-2	-1
1	-1	0

Gambar 4.35 Bobot Pada Filter Lapisan *Depthwise* (*Channel* 3)

-0,03242	-4,03707	0,32831
3,38043	1,04980	0,65663
1,34569	2,36306	3,70875

Gambar 4.36 *Feature Map* Hasil *Depthwise* Konvolusi (*Channel* 1)

1,31325	-3,01967	0,32831
2,03473	-1,01736	-3,38041
0	1,34568	-0,32831

Gambar 4.37 *Feature Map* Hasil *Depthwise* Konvolusi (*Channel* 2)

-2,72379	-4,03706	0,32831
-0,68905	-5,74348	-3,38042
1,34568	-0,32831	-2,36305

Gambar 4.38 *Feature Map* Hasil *Depthwise* Konvolusi (*Channel* 3)

4.2.1.8 Propagasi Maju Pada *Batch Normalization* Lapisan *Depthwise* Konvolusi

Hasil dari konvolusi pada tahap sebelumnya akan dilakukan normalisasi. Proses normalisasi sama dengan tahap-tahap sebelumnya yaitu menggunakan batch normalization. Hasil dari batch normalization dapat dilihat pada

-0,46582	-2,31995	-0,29880
1,11431	0,03523	-0,14679
0,17223	0,64327	1,26632

Gambar 4.39 Hasil *Batch Normalization* Lapisan *Depthwise* (*Channel* 1)

0,90571	-1,52290	0,35365
1,31010	-0,40060	-1,72509
0,16963	0,92388	-0,01438

Gambar 4.40 Hasil *Batch Normalization* Lapisan *Depthwise* (*Channel* 2)

-0,35678	-0,96591	1,05887
0,58698	-1,75740	-0,66135
1,53075	0,75430	-0,18946

Gambar 4.41 Hasil *Batch Normalization* Lapisan *Depthwise* (*Channel* 3)

4.2.1.9 Propagasi Maju Pada ReLU6 Lapisan *Depthwise*

Sama seperti tahap-tahap sebelumnya, setelah melakukan batch normalization akan dilakukan aktivasi menggunakan ReLU6. Hasil aktivasi dapat dilihat pada Gambar 4.42, Gambar 4.43, dan Gambar 4.44.

0	0	0
1,11431	0,03523	0
0,17223	0,64327	1,26632

Gambar 4.42 Hasil Aktivasi ReLU6 (*Channel* 1)

0,90571	0	0,35365
1,31010	0	0
0,16963	0,92388	0

Gambar 4.43 Hasil Aktivasi ReLU6 (*Channel* 2)

0	0	1,05887
0,58698	0	0
1,53075	0,75430	0

Gambar 4.44 Hasil Aktivasi ReLU6 (*Channel* 3)

4.2.1.10 Propagasi Maju Pada Lapisan *Projection*

Tahap ini merupakan lanjutan dari tahap sebelumnya. Hasil dari aktivasi ReLU6 di tahap sebelumnya dimasukkan pada lapisan *projection* untuk dilakukan proses konvolusi. Proses konvolusi dilakukan menggunakan satu filter berukuran 1x1 sebanyak tiga *channel.* Bobot dari filter tersebut dapat dilihat pada Gambar 4.45 dan hasil konvolusi dapat dilihat pada Gambar 4.46.

1		1,4		1,1

Gambar 4.45 Bobot Filter Lapisan *Projection* (1 Filter 3 *Channel*)

1,26799	0	1,65987
3,59413	0,03523	0
2,09355	2,76645	1,26632

Gambar 4.46 *Feature Map* Hasil Konvolusi Lapisan *Projection*

4.2.1.11 Propagasi Maju Pada *Batch Normalization* Lapisan *Projection*

Tahap terkahir pada proses konvolusi ini adalah batch normalization. Hasil konvolosi pada tahap sebelumnya dilakukan normalisasi dan didapatkan hasil seperti pada Gambar 4.47.

-0,11734	-1,17047	0,20812
1,81461	-1,14120	-1,17047
0,56831	1,12718	-0,11873

Gambar 4.47 Hasil *Batch Normalization* Lapisan *Projection*

4.2.1.12 Propagasi Maju Pada *Skip Connection*

Pada tahap ini merupakan proses penambahan hasil dari konvolusi yang ada pada Gambar 4.22 Dan akan ditambahkan dengan hasil konvolusi pada Gambar 4.47, proses ini biasanya disebut dengan penambahan skip connection. Hasil dari penambahan tersebut dapat dilihat pada Gambar 4.48.

0,69210	-0,02218	0,20812
1,81461	0,00708	-0,19160
0,56831	1,12718	-0,11873

Gambar 4.48 Hasil Penambahan *Skip Connection*

4.2.1.13 Propagasi Maju Pada Lapisan Linear

Setelah proses penambahan pada skip connection, proses selanjutnya adalah memasukkan hasil dari skip connection ke layer linear. Namun fitur yang ada dilakukan flatten terlebih dahulu sehingga menghasilkan sembilan masukan neuron. Setelah proses flatten, masing-masing dari nilai pada neuron tersebut akan dikalikan dengan bobot dan akan menghasilkan tiga keluaran nilai. Bobot yang digunakan ditunjukkan pada Gambar 4.49 dan hasil operasi perkalian ditunjukan pada Gambar 4.50.

-1	1	-2		1	1	2		0	1	2
1	2	2		0	-1	1		0	-1	1
0	1	-1		1	0	-1		-2	1	1

Gambar 4.49 Bobot Lapisan Linear

2,15937	5,09031	3,34476

Gambar 4.50 Hasil Operasi Lapisan Linear

4.2.1.14 Propagasi Maju Pada Aktivasi *Softmax*

Tahap ini merupakan tahap lanjutan dari hasil operasi pada lapisan linear. Setiap keluaran dari lapisan linear dilakukan aktivasi menggunakan fungsi aktivasi softmax dengan nilai kelas (0, 1, 2). Hasil aktivasi dapat dilihat pada Gambar 4.51.

0,04344		0,81440		0,14215

Gambar 4.51 Hasil Aktivasi *Softmax* Lapisan Linear

4.2.2 Perhitungan Akurasi

Setelah proses propagasi maju selesai, dilakukan perhitungan akurasi. Dari hasil aktivasi pada Gambar 4.51, didapatkan nilai prediksi yaitu 0,81440 yaitu memiliki kelas 1 dan sesuai dengan kelas sebenarnya. Proses perhitungan akurasi dapat menggunakan persamaan 2.8.

$$Akurasi = \frac{1+0}{1+0+0+0} = 1$$

4.2.3 Perhitungan Nilai *Loss*

Setelah proses propagasi maju selesai, dilakukan perhitungan nilai *loss*. Nilai *loss* dihitung agar bisa diketahui seberapa besar nilai kesalahan dari nilai prediksi dengan nilai sebenarnya. Perhitungan nilai *loss* dapat dilakukan menggunakan persamaan 2.7 dan didapatkan nilai loss sebesar 0,20530.

$$L = \sum_j 0^{(j)} log\sigma(0{,}43446)^{(j)}, 1^{(j)} log\sigma(0{,}814402)^{(j)}, 0^{(j)} log\sigma(0{,}142153)^{(j)}$$

4.2.4 Propagasi Balik

Propagasi balik dilakukan setelah perhitungan nilai *loss* dan pada prosesnya dilakukan perhitungan nilai gradien pada setiap lapisan dalam jaringan. Nilai gradien tersebut nantinya akan digunakan pada proses selanjutnya yaitu pembaruan nilai bobot dalam jaringan.

4.2.4.1 Propagasi Balik Nilai *Loss* terhadap Lapisan Linear (dLoss/dWeight)

Pada tahap ini dilakukan perhitungan gradien loss terhadap bobot linear. Untuk mencari gradien *loss* terhadap bobot linear, digunakan sebuah *chain rule* dari turunan *loss* terhadap *output softmax* yang ditunjukkan pada Gambar 4.52 kemudian turunan *output softmax* terhadap *input softmax* yang ditunjukkan pada Gambar 4.53 dan turunan *input softmax* terhadap bobot linear yang ditunjukkan pada Gambar 4.54. Ketiga nilai gradien tersebut akan dikalikan sehingga menghasilkan gradien *loss* terhadap bobot linear, seperti yang ditunjukkan pada Gambar 4.55, Gambar 4.56, Gambar 4.57.

1,0454		-1,2278		1,1657

Gambar 4.52 Gradien Loss Terhadap Output Softmax (dLoss/dOutSoftmax)

0,0415		0,1511		0,1219

Gambar 4.53 Gradien Input Softmax Terhadap Output Softmax (dOutSoftmax/dInSoftmax)

0,692104076	-0,022185904	0,208121638
1,814619201	0,007081901	-0,191605618
0,56831258	1,127188749	-0,118739092

Gambar 4.54 Gradien Input Softmax Terhadap Bobot Linear (dOutSoftmax/dWeightLinear)

0,030069081	-0,000963886	0,009042031
0,078837756	0,00030768	-0,008324478
0,024690849	0,048971724	-0,005158726

Gambar 4.55 Gradien Loss Terhadap Bobot Linear (dLoss/dWeight) 1

-0,128453439	0,004117669	-0,038627052
-0,336790499	-0,00131439	0,035561704
-0,10547793	-0,209204477	0,022037791

Gambar 4.56 Gradien Loss Terhadap Bobot Linear (dLoss/dWeight) 2

0,098384358	-0,003153783	0,029585021
0,257952744	0,00100671	-0,027237227
0,080787081	0,160232753	-0,016879064

Gambar 4.57 Gradien Loss Terhadap Bobot Linear (dLoss/dWeight) 3

4.2.4.2 Pembaruan Bobot Linear

Setelah proses perhitungan gradien *loss* terhadap bobot linear, tahap selanjutnya adalah pembaruan bobot. Proses pembaruan bobot dilakukan menggunakan persamaan 4.1 dan hasil pembaruan bobot dapat dilihat pada Gambar 4.58, Gambar 4.59, dan Gambar 4.60.

$$Bobot\ baru = bobot\ lama - (\alpha * Gradien) \quad (4.1)$$

Keterangan:
α = *Learning rate*

-1,000030069	1,000000964	-9,04203
0,999921162	1,999999692	1,000008324
-2,000024691	1,999951028	-0,999994841

Gambar 4.58 Hasil Pembaruan Bobot 1

1,000128453	-4,11767	1,000038627
1,00033679	-0,999998686	-3,55617

2,000105478	1,000209204	-1,000022038

Gambar 4.59 Hasil Pembaruan Bobot 2

-9,83844	3,15378	-2,000029585
0,999742047	-1,000001007	1,000027237
1,999919213	0,999839767	1,000016879

Gambar 4.60 Hasil Pembaruan Bobot 3

4.3 Perancangan Pengujian

Pada bagian perancangan pengujian, dijelaskan mengenai teknik perancangan pada pengujian yang dilakukan. Pengujian ini difokuskan pada konfigurasi *hyperparameter* dan penambahan metode augmentasi data. Kedua hasil pengujian tersebut dibandingkan dengan *hyperparameter* sebelum dilakukan konfigurasi dan sebelum penambahan metode augmentasi data.

4.3.1 Perancangan Pengujian Pengaruh *Dynamic Learning Rate* Terhadap Akurasi

Learning rate merupakan salah satu hyperparameter yang sangat berpengaruh dalam proses pelatihan jaringan saraf tiruan. Penentuan *learning rate* yang terlalu besar atau terlalu kecil akan berdampak pada proses pelatihan. Pada pengujian ini dilakukan penggunaan *dynamic learning rate*, yaitu proses perubahan *learning rate* secara otomatis pada *epoch* tertentu. Pemilihan pengujian ini dilakukan karena pada beberapa kasus penggunaan *dynamic learning rate* bisa membuat bobot menuruni gradient dengan perlahan, sehingga bisa mencapai *global* minima dengan efektif dan perlahan. Hasil akurasi akan dibandingkan dengan penggunaan *static learning rate* dan *dynamic learning rate*. Skenario pengujian *dynamic learning rate* dapat dilihat pada Tabel 4.1.

Tabel 4.1 Pengaruh *Dynamic Learning Rate* Terhadap Akurasi

Arsitektur	Akurasi	
	Data Validasi	Data Uji
MobileNetV2		
MobileNetV2 + *Dynamic Learning Rate*		

4.3.2 Perancangan Pengujian Pengaruh Metode Augmentasi Data Terhadap Akurasi

Augmentasi data merupakan salah satu cara untuk meningkatkan kinerja dari model. Pada kebanyakan kasus, augmentasi data dapat meningkatkan akurasi dari sebuah model secara signifikan. Pada penelitian ini akan dilakukan pengujian

pengaruh salah satu metode augmentasi data MSDA *(Mixed Sample Data Augmentation)* yaitu *CutMix* terhadap akurasi. Penggunaan *CutMix* pada beberapa penelitian terdahulu, menunjukkan peningkatan yang signifikan. Pada prosesnya *CutMix* juga bisa memperbanyak data dan meningkatan generalisasi pada model serta memberikan efek regularisasi. Beberapa alasan itulah yang mendasari dipilihnya *CutMix* dalam penelitian ini. Hasil akurasi penggunaan MSDA *CutMix* akan dibandingkan dengan tanpa penggunaan MSDA. Skenario pengujian penambahan *CutMix* dapat dilihat pada Tabel 4.2.

Tabel 4.2 Pengaruh Penambahan *CutMix* Terhadap Akurasi

Arsitektur	Akurasi	
	Data Validasi	Data Uji
MobileNetV2		
MobileNetV2 + *CutMix* + *Dynamic Learning Rate*		

4.3.3 Perancangan Pengujian Pengaruh *Sparse Regularization* Terhadap Akurasi

Sparse regularization merupakan salah satu metode regularisasi L1. *Sparse regularization* dilakukan pada input fungsi aktivasi ReLU dan diharapkan dapat mencegah peningkatan nilai yang terlalu berlebih pada ouput aktivasi ReLU dan dapat meningkatkan kemampuan generalisasi. Penggunaan *sparse regularization* ini didasarkan pada proses pelatihan dalam *deep learning* yang cenderung *overfitting* apabila nilai keluaran dari fungsi aktivasi terlalu jauh dari nol, oleh karena itu perlu adanya metode regularisasi tambahan yang bisa menekan terjadinya *overfitting*. Hal tersebut yang mendasari penggunaan *sparse regularization* dalam penelitian ini. Skenario pengujian *sparse regularization* dapat dilihat pada Tabel 4.3.

Tabel 4.3 Pengaruh *Sparse Regularization* Terhadap Akurasi

Arsitektur	Akurasi	
	Data Validasi	Data Uji
MobileNetV2 + *Sparse*		
MobileNetV2 + *Sparse* + *Dynamic Learning Rate*		
MobileNetV2 + *Sparse* + *Weight Decay* (0,001) + *Dynamic Learning Rate*		
MobileNetV2 + *Sparse* + *CutMix* + *Dynamic Learning Rate*		

BAB 5 IMPLEMENTASI

5.1 Implementasi Perhitungan Manual

Pada subbab ini berisi pembahasan dari implementasi jaringan dengan arsitektur yang sederhana.

Implementasi Perhitungan Manual

```
1   import torch
2   from torch import nn, optim
3
4   X = torch.tensor([[[[1, 0, 1, 0, 1],
5                       [0, 0, 1, 1, 1],
6                       [1, 1, 0, 0, 1],
7                       [0, 1, 0, 1, 1],
8                       [0, 0, 1, 1, 1]],
9                      [[1, 0, 1, 0, 1],
10                      [0, 0, -1, 1, 1],
11                      [1, 1, 0, 0, -1],
12                      [-1, 1, 0, 1, 0],
13                      [0, -1, 2, -1, 0]],
14                     [[0, -1, 1, 1, 1],
15                      [0, 0, -1, 1, 1],
16                      [1, 0, 0, -1, -1],
17                      [-1, 1, 0, 1, 0],
18                      [0, -1, -1, -1, 0]]]], dtype=torch.float32)
19
20  print(X.size())
21
22  class MyMobileNetV2(nn.Module):
23      def __init__(self):
24          super().__init__()
25          self.conv1 = nn.Conv2d(in_channels=3,
26                                 out_channels=1,
27                                 kernel_size=3,
28                                 bias=False)
29          self.conv1.weight = nn.Parameter(torch.tensor(
30                              [[[[2, 1, 1],
31                                 [0, -2, 1],
32                                 [1, -1, 0]],
33                                [[-1, 1, 0],
34                                 [0, -2, 1],
35                                 [1, -1, 1]],
36                                [[-1, -1, 1],
37                                 [0, -2, -1],
38                                 [1, -1, 0]]]],
39                               dtype=torch.float32,
40  requires_grad=True))
41          self.batchnorm = nn.BatchNorm2d(1, affine=True)
42          self.activation = nn.ReLU6()
43
44          self.expansion_layer = nn.Conv2d(in_channels=1,
45                                           out_channels=3,
46                                           kernel_size=1,
47                                           bias=False)
48          self.expansion_layer.weight = nn.Parameter(torch.tensor(
49                                [[[[1.2]]],
50                                 [[[1]]],
51                                 [[[1.1]]]],
52                                dtype=torch.float32,
53   requires_grad=True))
54          print("Exapnsion Layer Size:\n",
55
```

```
56
57  self.expansion_layer.weight.size())
58          self.batchnorm2 = nn.BatchNorm2d(3, affine=True)
59
60          self.depthwise = nn.Conv2d(in_channels=3,
61                                     out_channels=3,
62                                     kernel_size=3,
63                                     padding=1,
64                                     padding_mode='zeros',
65                                     groups=3,
66                                     bias=False)
67          self.depthwise.weight = nn.Parameter(torch.tensor(
68                                               [[[[2, 1, 1],
69                                                  [0, -2, 1],
70                                                  [1, -1, 0]]],
71                                                [[[-1, 1, 0],
72                                                  [0, -2, 1],
73                                                  [1, -1, 1]]],
74                                                [[[-1, -1, 1],
75                                                  [0, -2, -1],
76                                                  [1, -1, 0]]]],
77                                     dtype=torch.float32,
78  requires_grad=True))
79
80          self.projection_layer = nn.Conv2d(in_channels=3,
81                                            out_channels=1,
82                                            kernel_size=1,
83                                            bias=False)
84          self.projection_layer.weight = nn.Parameter(torch.tensor(
85                                                [[[[1]],
86                                                  [[1.4]],
87                                                  [[1.1]]]],
88                                     dtype=torch.float32,
89  requires_grad=True))
90          print("Projection Layer Size:\n",
91
92  self.projection_layer.weight.size())
93
94          self.fc = nn.Linear(in_features=9,
95                              out_features=3,
96                              bias=False)
97          self.fc.weight = nn.Parameter(torch.tensor(
98                               [[-1, 1, 0, 1, 2, 1, -2, 2, -1],
99                                [1, 0, 1, 1, -1, 0, 2, 1, -1],
100                               [0, 0, -2, 1, -1, 1, 2, 1, 1]],
101                              dtype=torch.float32,
102 requires_grad=True))
103         self.softmax = nn.Softmax()
104
105     def forward(self, X):
106         x = self.conv1(X)
107         print("conv:\n",x)
108         x = self.batchnorm(x)
109         print("Batch Norm:\n", x)
110         x = self.activation(x)
111         print("Relu6:\n", x)
112
114         skip_conn = x
115
116         x = self.expansion_layer(x)
117         print("expansion layer:\n", x)
118         x = self.batchnorm2(x)
120         print("expansion layer batch norm:\n", x)
121         x = self.activation(x)
122         print("expansion layer activation:\n", x)
```

```
123             x = self.depthwise(x)
124             print("Depthwise:\n",x)
125             x = self.batchnorm2(x)
126             print("Depthwise Batchnorm:\n", x)
127             x = self.activation(x)
128             print("Depthwise activation:\n", x)
129             x = self.projection_layer(x)
130             print("Projection Layer:\n",x)
131             x = self.batchnorm(x)
132             print("Projection Batchnorm:\n",x)
133
134             x += skip_conn
135             print("Result Skip Connection:\n", x)
136
137             hasil = torch.flatten(x)
138             print("Flatten:\n",hasil)
139             x = self.fc(torch.flatten(x, start_dim=1))
140             print("Result Neural Network:\n",x)
141             x = self.softmax(x)
142             print("softmax:\n", x)
143             return x
144
145     model = MyMobileNetV2()
146     criterion = nn.NLLLoss()
147     optimizer = optim.SGD(model.parameters(), lr=0.001)
148     output = model(X)
149     loss = criterion(torch.log(output), torch.tensor([2]))
150     print("Loss:\n", loss)
151
152     loss.backward()
153     optimizer.step()
154     print("Fully Connected grad:\n", model.fc.weight.grad)
155     print("New weight FC:\n", model.fc.weight)
```

Kode Sumber 5.1 Implementasi Perhitungan Manual

Pembahasan mengenai kode program perhitungan manual dapat dilihat pada Kode Sumber 5.1.

1. Baris 1-2 merupakan proses pemanggilan *library*/pustaka dari pytorch.
2. Baris 4-18 merupakan proses deklarasi nilai x yang akan digunakan sebagai input. Nilai x disini terdiri dari citra 5x5 sebanyak tiga *channel.*
3. Baris 22-93 merupakan proses pembuatan arsitektur MobileNetV2 secara sederhana yang terdiri dari *expansion layer*, *depthwise separable convolution,* dan *projection layer*.
4. Baris 94-103 merupakan pembuatan arsitektur jaringan linear.
5. Baris 105-143 merupakan pembuatan fungsi *forward propagation*.
6. Baris 145 merupakan inisialisasi arsitektur MyMobileNetV2 pada sebuah variable model.
7. Baris 146 merupakan inisilisasi nilai loss atau dengan menggunakan NLLLoss.
8. Baris 147 merupakan inisialisasi optimizer yang digunakan yaitu *Stohastic Gradient Descent* (SGD).
9. Baris 152 merupakan proses *backpropagation* atau propagasi balik.
10. Bari 153 merupakan proses pembaruan nilai pada optimizer.
11. Baris 154-155 merupakan proses menampilkan *gradient* dan bobot baru.

5.2 Implementasi Pemuat Dataset

Pada subbab ini dijelaskan mengenai implementasi dari pemuat dataset yang digunakan.

5.2.1 Implementasi Struktur Dataset

Implementasi Struktur Dataset

```
1  class ImageFolder(DatasetFolder):
2      def __init__(
3              self,
4              root: str,
5              transform: Optional[Callable] = None,
6              target_transform: Optional[Callable] = None,
7              loader: Callable[[str], Any] = default_loader,
8              is_valid_file: Optional[Callable[[str], bool]] = None,):
9          super(ImageFolder, self).__init__(
10                 root,
11                 loader,
12                 IMG_EXTENSIONS if is_valid_file is None else None,
13                 transform=transform,
14                 target_transform=target_transform,
15                 is_valid_file=is_valid_file)
16         self.imgs = self.samples
```

Kode Sumber 5.2 Implementasi Struktur Dataset

1. Baris 1 merupakan pembuatan kelas `ImageFolder` yang merupakan turunan dari kelas `DatasetFolder`
2. Baris 2-8 merupakan inisialisasi variable yang digunakan pada kelas.
3. Baris 9-15 merupakan pengambilan beberapa atribut dari kelas `DatasetFolder`
4. Baris 16 merupakan pembuatan variable `imgs`

5.2.2 Implementasi Pemuat Dataset (*Dataloader*)

Implementasi Pemuat Dataset

```
1  train_set = ImageFolder(
2                  '/content/ip102_v1.1/images/train',
3                  transform=train_transform)
4  trainloader = DataLoader(
5                    train_set,
6                    batch_size=bs,
7                    shuffle=True)
8
9  val_set = ImageFolder(
10                 '/content/ip102_v1.1/images/val',
11                 transform=val_transform)
12 validationloader = DataLoader(
13                       val_set,
14                       batch_size=bs,
15                       shuffle=True)
16
17 test_set = datasets.ImageFolder(
18                 '/content/ip102_v1.1/images/test',
19                 transform=test_transform)
20 testloader = DataLoader(test_set, batch_size=bs, shuffle=True)
```

Kode Sumber 5.3 Implementasi Pemuat Dataset

1. Baris 1-3 merupakan proses pengambilan lokasi direktori data training dan dilakukan transformasi data.
2. Baris 4-7 merupakan proses pemuatan data training yang sudah didefinisikan direktori penyimpananya.
3. Baris 9-11 merupakan proses pengambilan lokasi direktori data validasi dan dilakukan transformasi data.
4. Baris 12-15 merupakan proses pemuatan data validasi yang sudah didefinisikan direktori penyimpananya.
5. Baris 17-19 merupakan proses pengambilan lokasi direktori data uji dan dilakukan transformasi data.
6. Baris 20 merupakan proses pemuatan data uji yang sudah didefinisikan direktori penyimpananya.

5.3 Implementasi Fungsi Loss

Pada subbab ini dijelaskan mengenai implementasi fungsi *loss crossentropy*.

Implementasi Fungsi Loss Crossentropy

```
1  from .. import functional as F
2  from torch import Tensor
3  from typing import Optional
4
5  class CrossEntropyLoss(_WeightedLoss):
6      __constants__ = ['ignore_index', 'reduction']
7      ignore_index: int
8
9      def __init__(self,
10                  weight: Optional[Tensor] = None,
11                  size_average=None,
12                  ignore_index: int = -100,
13                  reduce=None, reduction: str = 'mean') -> None:
14         super(CrossEntropyLoss, self).__init__(weight,
15                                                size_average,
16                                                reduce,
17                                                reduction)
18         self.ignore_index = ignore_index
19
20     def forward(self, input: Tensor, target: Tensor) -> Tensor:
21         assert self.weight is None or isinstance(self.weight, Tensor)
22         return F.cross_entropy(input,
23                                target,
24                                weight=self.weight,
25                                ignore_index=self.ignore_index,
26                                reduction=self.reduction)
```

Kode Sumber 5.4 Implementasi Fungsi Loss CrossEntropy

1. Baris 1-3 merupakan pemuatan *library* yang akan digunakan
2. Baris 5 merupakan inisialisasi kelas `CrossEntropyLoss` dan menurunkan dari kelas `WeightedLoss`
3. Baris 6-7 merupakan inisialisasi variable `_constants_` dan `ignore_index` yang digunakan pada tahap selanjutnya.
4. Baris 9-13 merupakan tahap inisialisasi atribut yang digunakan pada kelas.
5. Baris 15-17 merupakan pemanggilan atribut dari kelas induk.
6. Baris 18 merupakan proses definisi nilai variable `ignore_index`.
7. Baris 20-26 merupakan pembuatan fungsi `forward` pada crossentropy.

5.4 Implementasi Optimizer

Pada subbab ini dibahas menganai imlementasi optimizer yang digunakan yaitu optimizer AdamW.

Implementasi Optimizer AdamW

```
import torch
from . import _functional as F
from .optimizer import Optimizer

class AdamW(Optimizer):
    def __init__(self,
                 params,
                 lr=1e-3,
                 betas=(0.9, 0.999),
                 eps=1e-8,
                 weight_decay=1e-2,
                 amsgrad=False):
        if not 0.0 <= lr:
            raise ValueError("Invalid learning rate:
{}".format(lr))
        if not 0.0 <= eps:
            raise ValueError("Invalid epsilon value:
{}".format(eps))
        if not 0.0 <= betas[0] < 1.0:
            raise ValueError("Invalid beta parameter at index 0:
{}".format(betas[0]))
        if not 0.0 <= betas[1] < 1.0:
            raise ValueError("Invalid beta parameter at index 1:
{}".format(betas[1]))
        if not 0.0 <= weight_decay:
            raise ValueError("Invalid weight_decay value:
{}".format(weight_decay))
        defaults = dict(lr=lr, betas=betas, eps=eps,
                        weight_decay=weight_decay,
amsgrad=amsgrad)
        super(AdamW, self).__init__(params, defaults)

    def __setstate__(self, state):
        super(AdamW, self).__setstate__(state)
        for group in self.param_groups:
            group.setdefault('amsgrad', False)

    @torch.no_grad()
    def step(self, closure=None):
        loss = None
        if closure is not None:
            with torch.enable_grad():
                loss = closure()

        for group in self.param_groups:
            params_with_grad = []
            grads = []
            exp_avgs = []
            exp_avg_sqs = []
            state_sums = []
            max_exp_avg_sqs = []
            state_steps = []
            amsgrad = group['amsgrad']

            for p in group['params']:
                if p.grad is None:
                    continue
                params_with_grad.append(p)
                if p.grad.is_sparse:
```

```
60                      raise RuntimeError('AdamW does not support
61  sparse gradients')
62                  grads.append(p.grad)
63
64                  state = self.state[p]
65
66                  # State initialization
67                  if len(state) == 0:
68                      state['step'] = 0
69                      # Exponential moving average of gradient
70  values
71                      state['exp_avg'] = torch.zeros_like(p,
72  memory_format=torch.preserve_format)
73                      # Exponential moving average of squared
74  gradient values
75                      state['exp_avg_sq'] = torch.zeros_like(p,
76  memory_format=torch.preserve_format)
77                      if amsgrad:
78                          # Maintains max of all exp. moving avg.
79  of sq. grad. values
80                          state['max_exp_avg_sq'] =
81  torch.zeros_like(p, memory_format=torch.preserve_format)
82
83                  exp_avgs.append(state['exp_avg'])
84                  exp_avg_sqs.append(state['exp_avg_sq'])
85
86                  if amsgrad:
87
88  max_exp_avg_sqs.append(state['max_exp_avg_sq'])
89
90                  beta1, beta2 = group['betas']
91                  # update the steps for each param group update
92                  state['step'] += 1
93                  # record the step after step update
94                  state_steps.append(state['step'])
95
96              F.adamw(params_with_grad,
97                      grads,
98                      exp_avgs,
99                      exp_avg_sqs,
100                     max_exp_avg_sqs,
101                     state_steps,
102                     amsgrad,
103                     beta1,
104                     beta2,
105                     group['lr'],
106                     group['weight_decay'],
107                     group['eps'])
108
109         return loss
```

Kode Sumber 5.5 Implementasi Optimizer AdamW

1. Baris 1-3 merupakan pemuatan *library* yang akan digunakan
2. Baris 5 merupakan inisialisasi kelas `AdamW` dan menurunkan dari kelas `Optimizer`.
3. Baris 6-12 merupakan inisialisasi artibut yang digunakan pada kelas.
4. Baris 13-30 merupakan pengecekan parameter yang digunakan.
5. Baris 33-36 merupakan pembuatan fungsi `setstate` yang digunakan untuk memberikan nilai *default* pada parameter.
6. Baris 39 merupakan pembuatan fungsi `step`.

7. Baris 45-53 merupakan pembuatan *variable* dengan tipe data list, yang digunakan ditahap selanjutnya.
8. Baris 55 merupakan proses dimulainya perulangan pada nilai `group['params']`.
9. Baris 58 merupakan proses penambahan nilai dari perulangan ke sebuah list params_with_grad.
10. Baris 62 merupakan proses menambahan nilai dari `p.grad` ke sebuah list `grads`.
11. Baris 67-81 merupakan proses pengecekan apabila panjang `state` bernilai nol.
12. Baris 88 merupakan penambahan nilai dari `state` ke sebuah list `max_exp_avg_sqs`.
13. Baris 92 merupakan proses update `steps` dari setiap `param group`.
14. Baris 94 merupakan proses penyimpanan step yang sudah diupdate ke sebuah list `state_steps`.
15. Baris 96-107 merupakan proses untuk memasukkan semua nilai yang telah diproses ke dalam Functional `adamw`.
16. Baris 109 merupakan pengembalian nilai *loss*.

5.5 Implementasi *Learning Rate Scheduler*

Pada subbab ini dibahas mengenai implementasi *learning rate scheduler*.

Implementasi Learning Rate Scheduler

```
1   import warnings
2   from .optimizer import Optimizer
3
4   class StepLR(_LRScheduler):
5           def __init__(self,
6                        optimizer,
7                        step_size,
8                        gamma=0.1,
9                        last_epoch=-1,
10                       verbose=False):
11              self.step_size = step_size
12              self.gamma = gamma
13              super(StepLR, self).__init__(optimizer,
14                                           last_epoch,
15                                           verbose)
16
17          def get_lr(self):
18              if not self._get_lr_called_within_step:
19                  warnings.warn("To get the last learning rate computed
20  by the scheduler please use `get_last_lr()`.", UserWarning)
21              if (self.last_epoch == 0) or
22                                  (self.last_epoch % self.step_size != 0):
23                  return [group['lr']
24                                  for group in self.optimizer.param_groups]
25              return [group['lr'] * self.gamma
26                                  for group in self.optimizer.param_groups]
27
28          def _get_closed_form_lr(self):
29              return  [base_lr  *  self.gamma  **  (self.last_epoch  //
30  self.step_size)
31                      for base_lr in self.base_lrs]
```

Kode Sumber 5.6 Implementasi *Learning Rate Scheduler*

1. Baris 1-2 merupakan proses pemuatan *library* yang dibutuhkan.
2. Baris 4 merupakan inisialisasi kelas `StepLR` yang merupakan turunan dari kelas `LRScheduler`.
3. Baris 5-10 merupakan inisialisasi atribut dari kelas.
4. Baris 11-12 merupakan deklarasi variable `step_size` dan `gamma`.
5. Baris 12-15 merupakan inisialisasi atribut turunan dari kelas induk.
6. Baris 17-26 merupakan pembuatan fungsi `get_lr` yang digunakan untuk mengambil nilai *learning rate* saat ini.
7. Baris 28-31 merupakan pembuatan fungsi `get_closed_form_lr` yang digunakan untuk mengembalikan perhitungan `base_lr`, `gamma`, jumlah epoch terkahir dan `step_size`.

5.6 Implementasi *Sparse Regularization*

Pada subbab ini dibahas mengenai proses implementasi dari *sparse regularization*.

Implementasi Sparse Regularization

```
1  models_children = list(model.children())
2
3  def sparse_loss(model, image):
4      loss_value = 0
5      value = image
6      for i in range(len(models_children)):
7          value = F.relu6((models_children[i](value)))
8          loss_value += torch.mean(torch.abs(value))
9      return loss_value
10
11 cross_entropy_loss = criterion(output, target)
12 sparse_reg = sparse_loss(model, feature)
13 loss = cross_entropy_loss + 0.001 * sparse_reg
```

1. Baris 1 merupakan proses pemanggilan semua *layer* pada model.
2. Baris 3 merupakan deklarasi fungsi `sparse_loss` dengan *argument* model dan fitur dari citra.
3. Baris 6-8 merupakan merupakan proses perulangan dalam setiap *layer* pada model yang digunakan untuk perhitungan *sparse regularization*.
4. Baris 11 merupakan deklarasi fungsi *loss* menggunakan *argument* keluaran dari model dan target yang sebenarnya.
5. Baris 12 merupakan penggunaan fungsi `sparse_loss` yang sudah dibuat sebelumnya.
6. Baris 13 merupakan proses modifikasi fungsi *loss*, pada tahap ini hasil fungsi *loss* crossentropy akan ditambahkan dengan nilai *lambda* dan dikalikan hasil perhitungan pada fungsi `sparse_reg`.

BAB 6 PENGUJIAN DAN ANALISIS HASIL

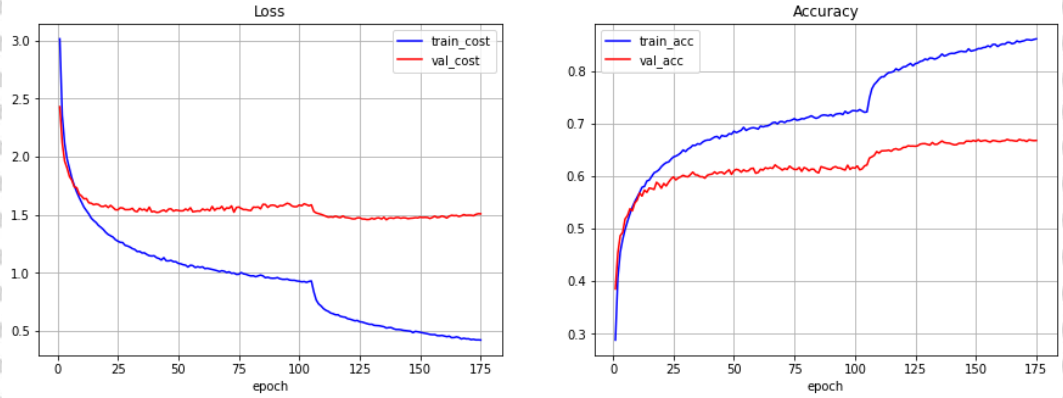
Teknik pelatihan yang digunakan pada semua pengujian adalah pelatihan menggunakan *batch size* sebesar 128 dan *optimizer* AdamW dengan nilai *learning rate* awal adalah 0,0001 serta penambahan dropout sebesar 0,2. Penggunaan AdamW dilakukan karena *optimizer* tersebut sudah memiliki *momentum* dan sudah terdapat sedikit L2 *Regularization* didalamnya yaitu *weight decay*. Selain itu augmentasi data seperti *random rotation*, *random resize crop*, dan *random horizontal flip* juga dilakukan pada semua tahap pengujian agar bisa menambah data dan membuat mesin belajar lebih banyak dari gambar atau citra yang berbeda-beda. Augmentasi data sangat layak dilakukan pada data citra atau *image*, karena bisa meningkatkan kualitas dari data. Proses normalisasi juga dilakukan karena pada tahap pengujian akan menggunakan *transfer learning* dengan *pre-trained* model dari ImageNet agar proses pelatihan bisa menjadi lebih efisien.

6.1 Pengujian Pengaruh *Dynamic Learning Rate* Terhadap Akurasi

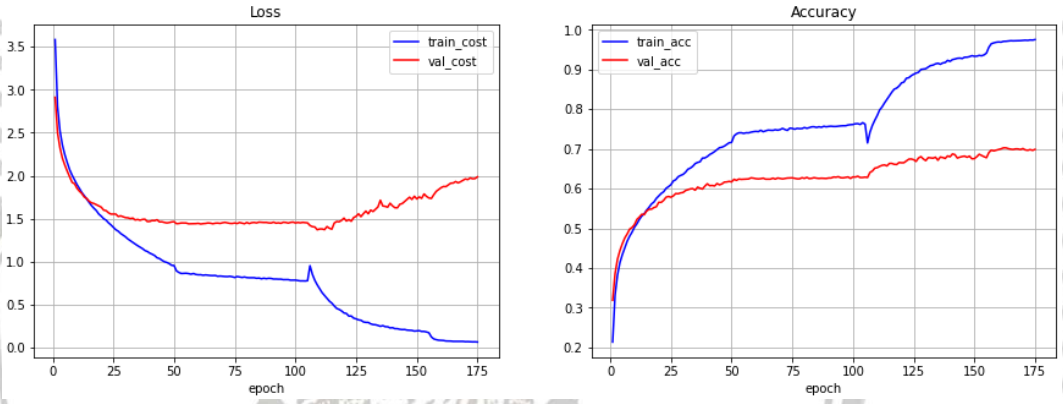
Pengujian *dynamic learning rate* dilakukan pada arsitektur yang sama dan *optimizer* yang sama yaitu AdamW, namun pada pelatihan tanpa *dynamic learning rate*, nilai awal *learning rate* diinisialisasi sebesar 0,001 dan menambahkan *dropout* sebesar 0,2. Sedangkan pengujian menggunakan *dynamic learning rate* menggunakan inisialisasi awal nilai *learning rate* sebesar 0,0001. Penggunaan *dynamic learning rate* menggunakan *step size* 50 dan nilai *gamma* sebesar 0,1. Hasil pengujian dapat dilihat pada Tabel 6.1. Kemudian grafik akurasi dan *loss* dari pelatihan mobilenetv2 ditunjukkan pada Gambar 6.1 sedangkan grafik akurasi dan loss dari pelatihan mobilenetv2 serta penambahan *dynamic learning rate* ditunjukkan pada Gambar 6.2.

Tabel 6.1 Hasil Pengujian Pengaruh *Dynamic Learning Rate* Terhadap Akurasi

Arsitektur	Akurasi	
	Data Validasi	Data Uji
MobileNetV2	0,6698	0,6704
MobileNetV2 (learning rate 0,0001) + *Dynamic Learning Rate*	**0,7028**	**0,7006**



Gambar 6.1 Grafik Hasil Pengujian MobileNetV2



Gambar 6.2 Grafik Hasil Pengujian MobileNetV2 dan *Dynamic Learning Rate*

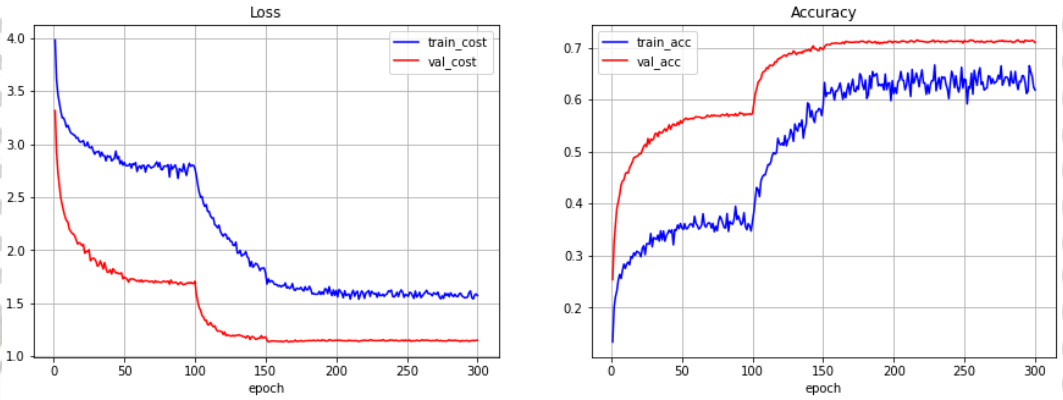
Pada hasil pelatihan serta pengujian pada data validasi dan data uji, penggunaan *dynamic learning rate* menunjukkan hasil yang lebih baik daripada tanpa penggunaan *dynamic learning rate*. Hal ini disebabkan karena *dynamic learning rate* akan membuat nilai *learning rate* menurun setiap 50 *epoch* (sesuai dengan nilai *step size*) dan nilai penurunan *learning rate* didapatkan dari pekalian nilai *gamma* dengan nilai *learning rate* sebelumnya. Penurunan *learning rate* tersebut membuat metode *optimizer* mencari *gradient* dengan perlahan sesuai dengan *learning rate* yang semakin kecil, sehingga tidak sampai terjebak di lokal minima.

6.2 Pengujian Pengaruh Metode *CutMix* Terhadap Akurasi

Pengujian dengan metode augmentasi data yaitu *CutMix* dilakukan setelah pengujian menggunakan *dynamic learning rate*. Pada pengujian ini konfigurasi yang digunakan sama dengan sebelumnya yaitu menggunakan *optimizer* AdamW dengan *learning rate* 0,0001 dan penggunaan *dropout* sebesar 0,2. Hasil pengujian dapat dilihat pada Tabel 6.2 dan grafik hasil pengujian dapat dilihat pada Gambar 6.3.

Tabel 6.2 Hasil Pengujian Pengaruh Metode *CutMix* Terhadap Akurasi

Arsitektur	Akurasi	
	Data Validasi	Data Uji
MobileNetV2 (learning rate 0,001)	0,6698	0,6704
MobileNetV2 + *CutMix* + *Dynamic Learning Rate*	**0,7144**	**0,7122**



Gambar 6.3 Grafik Hasil Pengujian Penggunaan *CutMix*

Dari hasil pengujian menunjukkan bahwa penggunaan *CutMix* membuat hasil akurasi pada data validasi dan data uji meningkat dari sebelumnya. Namun pada grafik juga terlihat bahwa model tersebut mengalami *underfitting*. Hal tersebut terlihat pada grafik akurasi dari data validasi yang memiliki nilai lebih baik daripada data latih. Meskipun model mengalami *underfitting*, akurasi ketika dilakukan pengecekan menggunakan data uji juga memiliki akurasi yang setara dengan data validasi. *Underfitting* pada penggunaan *CutMix* terjadi karena *dataset* yang digunakan terlihat cukup rumit untuk dilakukan *Cut* dan *Mix*, sehingga model mengalami kebingungan ketika melakukan pelatihan. Contohnya adalah pada *dataset* terdapat beberapa citra yang ukuran serangganya lebih kecil dari lingkungan sekitar (terlihat pada Gambar 6.4), sehingga hanya nampak daun-daun pada tanaman saja, hal inilah yang mengakibatkan *CutMix* akan kesulitan untuk mengganti salah satu bagian pada citra. Meskipun terjadi *underfitting*, hasil dari data uji menunjukkan peningkatan, oleh karena itu pada pengujian ini bisa dikatakan bahwa model sudah mengalami peningkatan daripada sebelumnya.

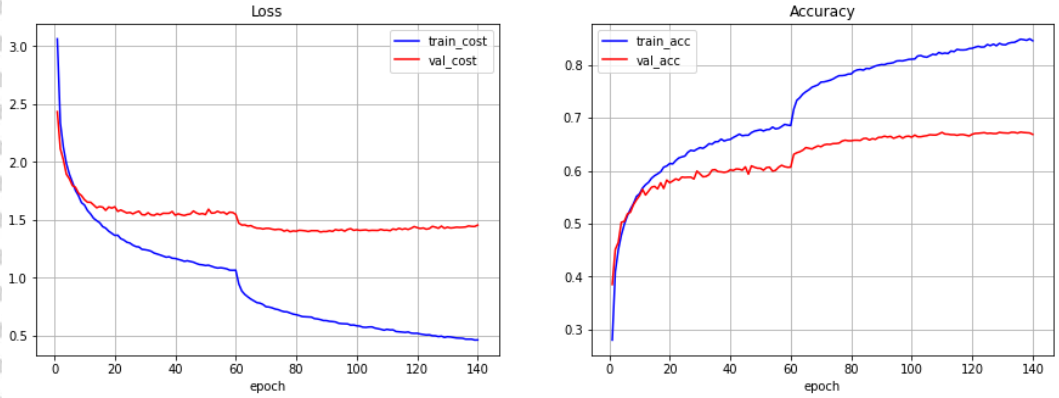
Gambar 6.4 Tanaman Yang Mendominasi Pada Citra

6.3 Pengujian Pengaruh *Sparse Regularization* Terhadap Akurasi

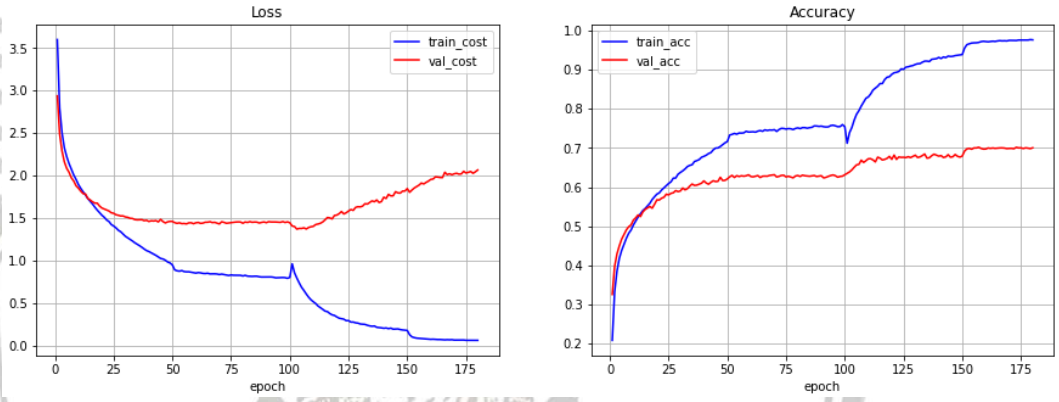
Pengujian penggunaan *sparse regularization* dilakukan dengan konfigurasi yang mirip dengan pengujian sebelumnya. Menggunakan *batch size* 128, *optimizer* AdamW dengan nilai awal *learning rate* 0,0001, dan penambahan *dropout* sebesar 0,2. Pada penggunaan *dynamic learning rate*, masih tetap mempertahankan konfigurasi sebelumnya yaitu nilai *gamma* sebesar 0,1 dan penurunan *learning rate* di setiap 50 *epoch*. Semua pengujian menggunakan nilai *default* dari *weight decay* yaitu 0,01 kecuali pada satu pengujian mencoba untuk mengganti nilai *weight decay* menjadi 0,001. Hasil pengujian dapat dilihat pada Tabel 6.3 dan grafik pengujian dapat dilihat pada Gambar 6.5, Gambar 6.6, Gambar 6.7, dan Gambar 6.8.

Tabel 6.3 Hasil Pengujian Pengaruh *Sparse Regularization* Terhadap Akurasi

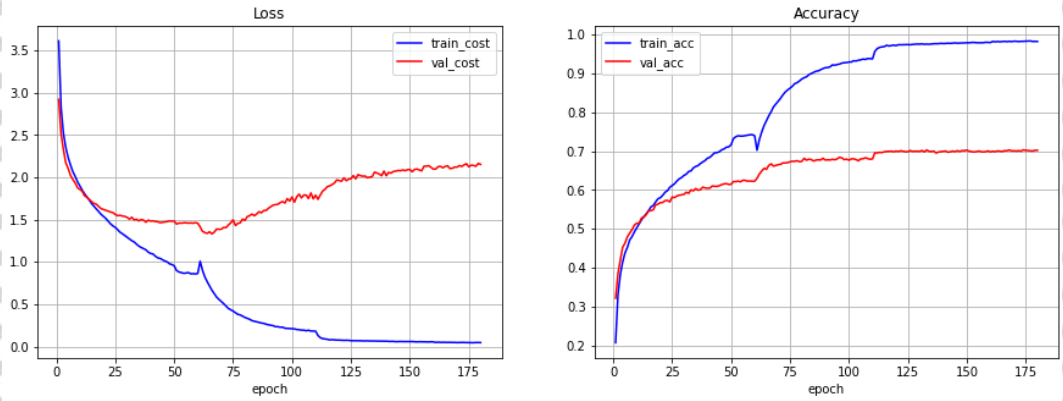
Arsitektur	Akurasi	
	Data Validasi	Data Uji
MobileNetV2	0,6698	0,6704
MobileNetV2 + *Sparse*	0,6730	0,6728
MobileNetV2 + *Sparse* + *Dynamic Learning Rate*	0,7017	0,6989
MobileNetV2 + *Sparse* + *Weight Decay* (0,001) + *Dynamic Learning Rate*	0,7032	0,6995
MobileNetV2 + *Sparse* + *CutMix* + *Dynamic Learning Rate*	**0,7136**	**0,7132**



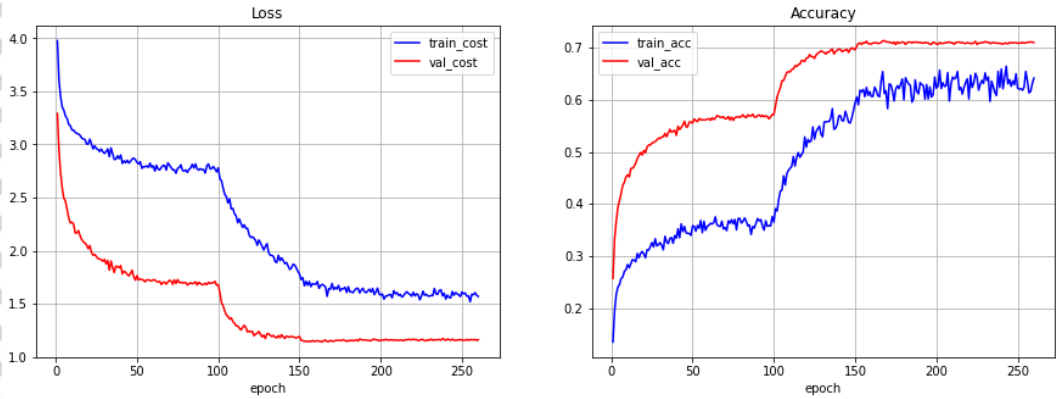
Gambar 6.5 Grafik Hasil Pengujian MobileNetV2 + *Sparse*



Gambar 6.6 Grafik Hasil Pengujian MobileNetV2 + *Sparse* + *Dynamic LR*



Gambar 6.7 Grafik Hasil Pengujian MobileNetV2 + *Sparse* + *Weight Decay* + *Dynamic Learning Rate*



Gambar 6.8 Grafik Hasil Pengujian MobileNetV2 + *Sparse* + *Cutmix* + *Dynamic Learning Rate*

Dari hasil pengujian menunjukkan bahwa penggunaan *sparse regularization* sangat berpengaruh terhadap peningkatan akurasi. Pada pengujian mobilenetv2 tanpa *sparse regularization* dan menggunakan *sparse regularization* mengalami peningkatan satu persen pada data validasi dan peningkatan sedikit pada data uji. Namun untuk penggunaan *dynamic learning rate* tidak ada peningkatan yang signifikan Ketika menggunakan *sparse regularization*. Peningkatan akurasi Kembali terjadi pada penggunaan *cutmix*. Kombinasi *cutmix* kemudian ditambah dengan *sparse regularization* akan membuat model lebih general dalam melakukan pelatihan. Meskipun terjadi sedikit *underfitting*, tetapi hal tersebut adalah hal yang normal karena pada *dataset* yang digunakan terlihat cukup rumit untuk dilakukan *Cut* dan *Mix*, sehingga model mengalami kebingungan ketika melakukan pelatihan. Namun hasil dari data validasi dan data uji menunjukkan hasil yang baik, ditambah penggunaan *sparse regularization* membuat hasil akurasi sedikit mengalami peningkatan dari pengujian sebelumnya tanpa penggunaan *sparse regularization*. Penggunaan *sparse regularization* pada pengujian ini akan membuat hasil output fungsi aktivasi menjadi tidak terlalu jauh dari nol dan memberikan efek *batch normalization* serta meningkatkan kemampuan generalisasi pada model.

6.4 Perbandingan Hasil Pengujian Dengan Penelitian Terdahulu

Beberapa penelitian terdahulu sudah dilakukan dengan topik dan penggunaan *dataset* yang sama, yaitu IP102 *dataset*. Wu et al., (2019) yang merupakan pengumpul *dataset* tersebut juga melakukan pengujian dan mendapatkan akurasi sebesar 49,5% menggunakan arsitektur ResNet. Ditahun yang sama Ren et al., (2019) juga melakukan penelitian dengan dataset tersebut dengan menggunakan sebuah arsitektur modifikasi dari ResNet. Arsitektur ini merupakan arsitektur yang tergolong baru karena merupakan hasil modifikasi pada layer arsitektur ResNet dan diberi nama FR-ResNet. Hasil akurasi yang didapatkan pada penelitian ini adalah 55,24%. Nanni et al., (2019) juga melakukan penelitian dengan *dataset* yang sama namun mengunakan arsitektur yang berbeda. Hasil akurasi tertinggi didapatkan ketika menggunakan arsitektur DenseNet dengan penambahan *FusionSum*. Hasil akurasi yang didapatkan adalah 61,93%. Selanjutnya, Ayan et al.,

(2020) juga melakukan penelitian serupa, namun pada penelitian ini menggunakan GAEnsemble. GAEnsemble merupakan hasil penggabungan dari ketiga arsitektur terdahulu yaitu Inception-V3, Xception, dan MobileNet. Hasil dari penelitian ini mendapatkan akurasi sebesar 67,13. Dan yang terakhir adalah pada penelitian ini, dengan penggunaan MobileNetV2 dan dilakukan penambahan *sparse regularization*, metode augmentasi data *CutMix*, dan *dynamic learning rate*, membuat hasil akurasi meningkat dari penelitian terdahulu yaitu mencapai 71,36%. Hasil perbandingan akurasi dapat dilihat pada Tabel 6.4.

Tabel 6.4 Hasil Akurasi Perbandingan Dengan Penelitian Terdahulu

Arsitektur	Akurasi Data Uji
ResNet (Wu *et al.*, 2019)	49,5
FR-ResNet (Ren, Liu dan Wu, 2019)	55,24
DenseNet + Fusion Sum (Nanni, Maguolo dan Pancino, 2019)	61,93
GAEnsemble (Ayan, Erbay dan Varçin, 2020)	67,13
MobileNetV2 + *Sparse* + *CutMix* + *Dynamic Learning Rate*	**71,36**

BAB 7 KESIMPULAN

7.1 Kesimpulan

Dari implementasi serta hasil pengujian yang sudah dilakukan berdasarkan perancangan, dapat dilakukan penarikan kesimpulan dari penelitian ini yaitu:

1. Implementasi dari arsitektur MobileNetV2 pada *dataset* IP102 adalah dengan menggunakan batch size sebesar 128 dan *optimizer* AdamW dengan nilai *learning rate* awal adalah 0,0001. Penggunaan nilai dropout sebesar 0,2 juga ditambahkan agar model bisa belajar lebih baik. Pada beberapa pengujian juga dilakukan penggunaan *dynamic learning rate*, metode augmentasi data *CutMix* dan *sparse regularization* yang terbukti bisa membuat hasil akurasi menjadi lebih meningkat.
2. Teknik pelatihan yang menghasilkan akurasi tertinggi dari *dataset* IP102 adalah pelatihan dengan penggunaan *dynamic learning rate*, penambahan *cutmix*, dan *sparse regularization*. Faktor lain yang mempengaruhi adalah penggunaan *optimizer* AdamW yang membuat proses penurunan *gradien* menjadi lebih efektif dengan adanya *momentum* pada *optimizer* tersebut serta adanya *L2 regularization* yaitu *weight decay*. Beberapa metode dan konfigurasi tersebut yang membuat hasil akurasi meningkat dari pengujian sebelumnya.
3. Hasil akurasi tertinggi yang didapatkan dari dataset IP102 adalah sebesar 0,7132. Hasil tersebut didapatkan setelah melakukan beberapa pengujian berdasarkan perancangan. Akurasi tersebut didapatkan dengan pelatihan menggunakan *dynamic learning rate*, penambahan *cutmix*, dan *sparse regularization*. Perbandingan dengan penelitian terdahulu juga sudah dilakukan untuk melakukan perbandingan lebih lanjut.

7.2 Saran

Pada penelitian ini tidak lepas dari kekurangan yang masih belum dapat diimplementasikan. Oleh karena itu penulis memberikan beberapa saran yaitu:

1. Mencoba penggunaan augmentasi data FMix pada proses pelatihan agar bisa terlihat perbedaan antara penggunaan *CutMix* dan FMix.
2. Melakukan pelatihan dengan menambahkan *Weighted Random Sampler* untuk jumlah data disetiap kelas yang beragam jumlahnya.

DAFTAR REFERENSI

Alom, M. Z. *et al.* (2019) "A State of the Art Survey on Deep Learning Theory and Architectures," *Electronics*, 8, hal. 292. doi: 10.3390/electronics8030292.

Ayan, E., Erbay, H. dan Varçın, F. (2020) "Crop pest classification with a genetic algorithm-based weighted ensemble of deep convolutional neural networks," *Computers and Electronics in Agriculture*, 179(September), hal. 105809. doi: 10.1016/j.compag.2020.105809.

Babu, B. . dan Shailesh, M. (2014) "Adaptive Networks for Fault Diagnosis," (May).

Bansari, S. (2019) *Introduction to how CNNs Work*, *Data Driven Investor Medium*. Tersedia pada: https://medium.datadriveninvestor.com/introduction-to-how-cnns-work-77e0e4cde99b (Diakses: 20 Juli 2021).

Bjorck, J. *et al.* (2018) "Understanding Batch Normalization," *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, (NeurIPS), hal. 12.

Djojosumarto, P. (2008) *Pestisida dan Aplikasinya*. Jakarta: AgroMedia Pustaka.

Howard, A. G. *et al.* (2017) "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv*.

Ide, H. dan Kurita, T. (2017) "Improvement of learning for CNN with ReLU activation by sparse regularization," *Proceedings of the International Joint Conference on Neural Networks*, 2017-May(May 2017), hal. 2684–2691. doi: 10.1109/IJCNN.2017.7966185.

Ioffe, S. dan Szegedy, C. (2015) "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *32nd International Conference on Machine Learning, ICML 2015*, 1, hal. 448–456.

Krizhevsky, A., Sutskever, I. dan Hinton, G. E. (2017) "ImageNet Classification with Deep Convolutional Neural Networks," *Communications of the ACM*, 60(6). doi: 10.1145/3065386.

Meilin dan Nasamsir, A. (2016) "SERANGGA dan PERANANNYA DALAM BIDANG PERTANIAN dan KEHIDUPAN," *Jurnal Media Pertanian*, 1(1), hal. 18. doi: 10.33087/jagro.v1i1.12.

Nanni, L., Maguolo, G. dan Pancino, F. (2019) "Insect Pest Image Detection and Recognition Based on Bio-Inspired Methods," *arXiv*.

O'Shea, K. dan Nash, R. (2015) "An Introduction to Convolutional Neural Networks," (November). Tersedia pada: http://arxiv.org/abs/1511.08458.

PyTorch (2019) *ReLU6*, *PyTorch Documentation*. Tersedia pada: https://pytorch.org/docs/master/generated/torch.nn.ReLU6.html (Diakses: 8 Maret 2021).

Ren, F., Liu, W. dan Wu, G. (2019) "Feature reuse residual networks for insect pest recognition," *IEEE Access*, 7, hal. 122758–122768. doi:

10.1109/ACCESS.2019.2938194.

Rothschild, L. G. E. K. P. A. van der L. G. H. L. (1981) *Pests of Crop in Indonesia*. Jakarta: P.T Ichtiar Baru Van Hoeve.

Sandler, M. *et al.* (2018) "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, hal. 4510–4520. doi: 10.1109/CVPR.2018.00474.

Schmidhuber, J. (2015) "Deep Learning in neural networks: An overview," *Neural Networks*, 61, hal. 85–117. doi: 10.1016/j.neunet.2014.09.003.

Wang, Y. *et al.* (2017) "Specific and intrinsic sequence patterns extracted by deep learning from intra-protein binding and non-binding peptide fragments," *Scientific Reports*, 7(1), hal. 1–9. doi: 10.1038/s41598-017-14877-w.

Westerkamp, C. dan Gottsberger, G. (2000) "Diversity Pays in Crop Pollination," *Crop Science*, 40(5), hal. 1209–1222. doi: https://doi.org/10.2135/cropsci2000.4051209x.

Wu, X. *et al.* (2019) "IP102: A large-scale benchmark dataset for insect pest recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June, hal. 8779–8788. doi: 10.1109/CVPR.2019.00899.

Yamashita, R. *et al.* (2018) "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, 9(4), hal. 611–629. doi: 10.1007/s13244-018-0639-9.

Yudistira, N., Widodo, A. W. dan Rahayudi, B. (2020) "Deteksi Covid-19 pada Citra Sinar-X Dada Menggunakan Deep Learning yang Efisien," *Jurnal Teknologi Informasi dan Ilmu Komputer*, 7(6), hal. 1289. doi: 10.25126/jtiik.2020763651.

Yun, S. *et al.* (2019) "CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features." Tersedia pada: https://arxiv.org/abs/1905.04899 (Diakses: 10 Juni 2021).