# Implementasi Arsitektur MobilenetV3 (Studi Kasus Klasifikasi Jamur Beracun)

**GALIH ASHARI RAKHMAT[1]*, FIQRON RIZKIAWARMAN[1]**

[1]Studi Informatika, Institut Teknologi Nasional Bandung
Email: galihas@itenas.ac.id

*Received* 04 09 2023 **|** *Revised* 11 09 2023 **|** *Accepted* 11 09 2023

**ABSTRAK**

*Mobilenet adalah arsitektur model Convolutional Neural Network yang implementasinya digunakan pada perangkat mobile. Dasar dari pembuatan mobilenet adalah depthwise separable convolution yang membuat arsitektur ini memiliki beban komputasi yang rendah, namun hal tersebut dapat mengurangi tingkat akurasi dari arsitektur. Penelitian dilakukan dengan mengidentifikasi 6 kelas genus jamur dengan menggunakan arsitektur MobilenetV3 dengan tidak menghiraukan beban komputasi dari arsitektur MobilenetV3 yaitu dengan menggunakan data balancing, perubahan pada stride dan penambahan layer. Hasil terbaik yang diperoleh dari model didapatkan pada model arsitektur MobilenetV3-Large, stride 1, dengan hyperparameter learning rate 0.0001, batch size 32, epoch 30, optimizer ADAM. Berdasarkan hasil dari evaluasi performa model didapatkan akurasi sebesar 0.9981 pada pengujian 2076 data uji dalam 6 kelas yang terdiri dari jenis genus jamur yang dapat dikonsumsi dan beracun.*

***Kata kunci**: MobilenetV3, Jamur, Identifikasi, Akurasi*

**ABSTRACT**

*Mobilenet is a Convolutional Neural Network architecture whose applications are used on mobile devices. The foundation of Mobilenet's creation is depthwise separable convolution, which makes this architecture have a low computational load but can reduce the accuracy level of the architecture. The research was conducted by identifying 6 classes of mushroom genera using the MobilenetV3 architecture, disregarding the computational load of MobilenetV3 by employing data balancing, changes in stride, and adding layers. The best result obtained from the model was achieved with the MobilenetV3-Large architecture, stride 1, hyperparameter learning rate of 0.0001, batch size of 32, epoch 30, and ADAM optimizer. Based on the evaluation results of the model's performance, an accuracy of 0.9981 was achieved on testing 2076 test data across 6 classes consisting of edible and toxic mushroom genera.*

***Keywords**: MobilenetV3, Mushrooms, Identification, Accuracy*

## 1. PENDAHULUAN

Jamur beracun sangat sulit dibedakan dengan jamur konsumsi oleh orang awam dikarenakan banyaknya spesies jamur di alam liar, kurangnya pengetahuan tentang jamur, dan kurangnya ahli dalam bidang jamur sehingga pengidentifikasian jamur sulit dilakukan. **(Putra 2021)** melaporkan bahwa jamur liar merupakan salah satu komoditas pertanian musiman yang memiliki nilai ekonomi sehingga sering dicari oleh mayarakat lokal ketika merambah. Selain itu, beberapa jamur liar yang tumbuh di sekitar lokasi kegiatan antropogenik merupakan jenis yang dapat dimakan atau *edible* dan juga beracun. Hal ini menyebabkan kasus keracunan jamur merupakan salah satu ancaman kesehatan serius bagi masyarakat dan penggiat jamur.

Dibutuhkan teknologi yang dapat mengolah informasi dari suatu citra atau gambar untuk pengenalan objek secara otomatis. *Computer vision* meliputi metode untuk memperoleh, memproses, menganalisis, dan memahami gambar digital, dan ekstraksi data dimensi tinggi dari dunia nyata untuk menghasilkan informasi numerik atau simbolik, misalnya dalam bentuk keputusan.

*Deep Learning* (DL) merupakan sebuah teknik berbasis jaringan saraf tiruan telah banyak digunakan dalam beberapa tahun terakhir sebagai salah satu metode implementasi *Machine Learning* (ML) Pada beberapa artikel disebutkan bahwa DL tidak hanya spesifik untuk bidang tertentu, tetapi telah didefinisikan sebagai bentuk pembelajaran umum yang dapat menyelesaikan hampir berbagai macam masalah di berbagai bidang **(Haris et al. 2021)**.

Identifikasi akan dilakukan dengan menggunakan arsitektur *MobileNetV3* dengan tujuan untuk menciptakan model arsitektur yang ringan, sehingga dapat diimplementasikan pada sistem *Mobile*. *MobileNet* merupakan model yang memiliki ukuran kecil baik dari jumlah parameter maupun ukuran model yang dihasilkan. Perbedaan mendasar antara arsitektur *MobileNet* dan arsitektur CNN pada umumnya adalah penggunaan lapisan atau *layer* konvolusi dengan ketebalan filter yang sesuai dengan ketebalan *input image* **(Zakiya, Novamizanti, and Rizal 2021)**.

Berdasarkan penelitian yang sudah dilakukan oleh (Salafy, 2022) dengan judul Perbandingan Performa Deteksi pada Forward Collision Warning System menggunakan Metode MobilenetV3 dan YOLOv4 pada Perangkat Raspberry Pi 4, didapatkan hasil dari pengujian menggunakan arsitektur *MobilenetV3* yaitu akurasi sebesar 59.8%.

Salah satu permasalahan pada penelitian ini yaitu ketidakseimbangan data yang digunakan atau sering disebut dengan *imbalanced dataset* Cukup banyak penelitian yang melaporkan bahwa imbalanced dataset ini seringkali memberikan hasil yang keliru. Perlu ada penanganan khusus sebelum *imbalanced dataset* tersebut dapat digunakan pada *machine learning*. Cara paling populer dan efektif dalam mengatasi permasalahan *imbalanced dataset* adalah melakukan *resampling*, baik *oversampling*, *undersampling*, ataupun kombinasi keduanya **(Indrawati 2021)**.

Tujuan dari penelitian ini adalah melakukan impelementasi arsitektur *MobilenetV3* untuk melakukan klasifikasi jenis jamur beracun serta untuk mengetahui performa model arsitektur *MobilenetV3* dalam hal peningkatan akurasi jika menggunakan metode *data balancing* untuk meningkatkan jumlah *dataset* yang digunakan.

## 2. METODE PENELITIAN

### 2.1. *MobileNetV3*

*MobilenetV3* merupakan kombinasi dari kedua model sebelumnya (*MobilenetV1* dan *MobilenetV2)* dengan tujuan mendapatkan model yang paling effisien. *MobilenetV3* memiliki dua jenis model yaitu *MobilenetV3-Large dan MobilenetV3-Small,* perbedaannya terdapat pada jumlah *layer bottleneck* yang digunakan*.* Model ini digunakan pada kasus penggunaan sumber daya tinggi dan rendah **(Howard et al. 2019)**. Perbedaan arsitektur *MobilenetV3* dibandingkan dengan model *MobilenetV1* dan *MobilenetV2* yaitu berupa peningkatan fitur pada arsitektur yaitu dengan adanya *squeeze-And-Excite* (SE). Berikut ini adalah arsitektur dari *MobilenetV3-Large* dan *Mobilenetv3-Small.*

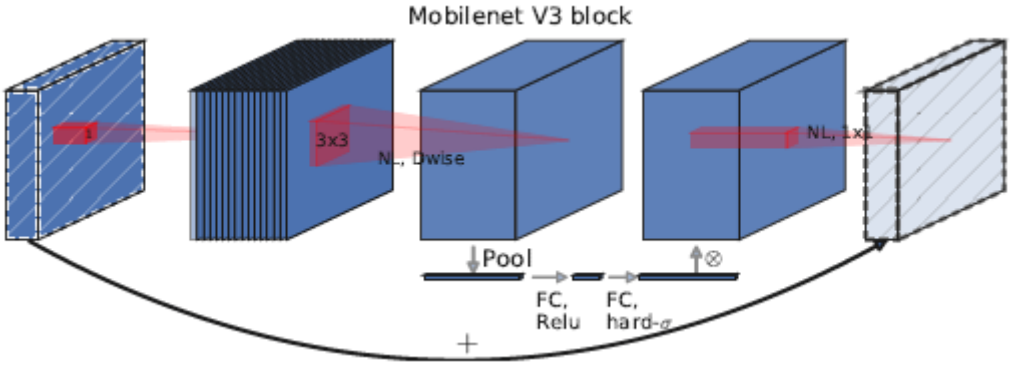
| Input | Operator | exp size | *#out* | SE | NL | *s* |
|---|---|---|---|---|---|---|
| $224^2 \times 3$ | conv2d | - | 16 | - | HS | 2 |
| $112^2 \times 16$ | bneck, 3x3 | 16 | 16 | - | RE | 1 |
| $112^2 \times 16$ | bneck, 3x3 | 64 | 24 | - | RE | 2 |
| $56^2 \times 24$ | bneck, 3x3 | 72 | 24 | - | RE | 1 |
| $56^2 \times 24$ | bneck, 5x5 | 72 | 40 | ✓ | RE | 2 |
| $28^2 \times 40$ | bneck, 5x5 | 120 | 40 | ✓ | RE | 1 |
| $28^2 \times 40$ | bneck, 5x5 | 120 | 40 | ✓ | RE | 1 |
| $28^2 \times 40$ | bneck, 3x3 | 240 | 80 | - | HS | 2 |
| $14^2 \times 80$ | bneck, 3x3 | 200 | 80 | - | HS | 1 |
| $14^2 \times 80$ | bneck, 3x3 | 184 | 80 | - | HS | 1 |
| $14^2 \times 80$ | bneck, 3x3 | 184 | 80 | - | HS | 1 |
| $14^2 \times 80$ | bneck, 3x3 | 480 | 112 | ✓ | HS | 1 |
| $14^2 \times 112$ | bneck, 3x3 | 672 | 112 | ✓ | HS | 1 |
| $14^2 \times 112$ | bneck, 5x5 | 672 | 160 | ✓ | HS | 2 |
| $7^2 \times 160$ | bneck, 5x5 | 960 | 160 | ✓ | HS | 1 |
| $7^2 \times 160$ | bneck, 5x5 | 960 | 160 | ✓ | HS | 1 |
| $7^2 \times 160$ | conv2d, 1x1 | - | 960 | - | HS | 1 |
| $7^2 \times 960$ | pool, 7x7 | - | - | - | - | 1 |
| $1^2 \times 960$ | conv2d 1x1, NBN | - | 1280 | - | HS | 1 |
| $1^2 \times 1280$ | conv2d 1x1, NBN | - | k | - | - | 1 |

**Gambar 1. Arsitektur *MobileNetV3-Large* (Howard et al. 2019)**

| Input | Operator | exp size | *#out* | SE | NL | *s* |
|---|---|---|---|---|---|---|
| $224^2 \times 3$ | conv2d, 3x3 | - | 16 | - | HS | 2 |
| $112^2 \times 16$ | bneck, 3x3 | 16 | 16 | ✓ | RE | 2 |
| $56^2 \times 16$ | bneck, 3x3 | 72 | 24 | - | RE | 2 |
| $28^2 \times 24$ | bneck, 3x3 | 88 | 24 | - | RE | 1 |
| $28^2 \times 24$ | bneck, 5x5 | 96 | 40 | ✓ | HS | 2 |
| $14^2 \times 40$ | bneck, 5x5 | 240 | 40 | ✓ | HS | 1 |
| $14^2 \times 40$ | bneck, 5x5 | 240 | 40 | ✓ | HS | 1 |
| $14^2 \times 40$ | bneck, 5x5 | 120 | 48 | ✓ | HS | 1 |
| $14^2 \times 48$ | bneck, 5x5 | 144 | 48 | ✓ | HS | 1 |
| $14^2 \times 48$ | bneck, 5x5 | 288 | 96 | ✓ | HS | 2 |
| $7^2 \times 96$ | bneck, 5x5 | 576 | 96 | ✓ | HS | 1 |
| $7^2 \times 96$ | bneck, 5x5 | 576 | 96 | ✓ | HS | 1 |
| $7^2 \times 96$ | conv2d, 1x1 | - | 576 | ✓ | HS | 1 |
| $7^2 \times 576$ | pool, 7x7 | - | - | - | - | 1 |
| $1^2 \times 576$ | conv2d 1x1, NBN | - | 1280 | - | HS | 1 |
| $1^2 \times 1280$ | conv2d 1x1, NBN | - | k | - | - | 1 |

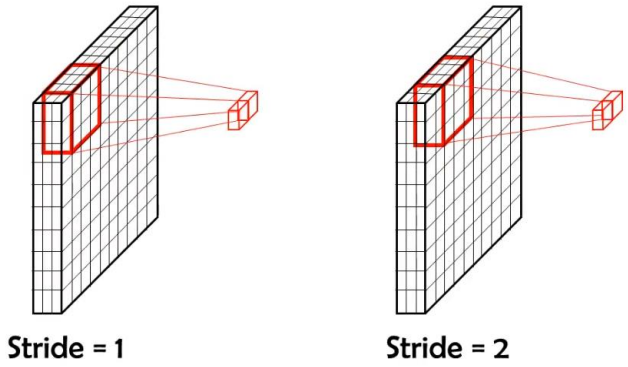
**Gambar 2. Arsitektur *MobileNetV3-Small* (Howard et al. 2019)**

*# out* adalah jumlah filter yang digunakan, *exp size* adalah singkatan dari *expansion size,* SE adalah *Squeeze-And-Excite*. HS adalah singkatan dari aktivasi *hard-swish* dan RE adalah singkatan dari aktivasi *Rectified Linear Unit (ReLU).* NBN pada *operator* merupakan singkatan dari *no batch normalization,* sedangkan s adalah singkatan dari *stride*.




**Gambar 3 *MobilenetV3 block* (Howard et al. 2019)**

Seperti yang dapat dilihat pada gambar 3, tahapan arsitektur *MobilenetV3* dimulai dari proses *input* citra dengan konvolusi2d, *kernel* 3x3, *stride* 2, dengan aktivasi *hard-swish* pada citra ukuran 224x224 dengan 3 *channel* warna yaitu *red, green,* dan *blue*. Tahap berikutnya adalah proses konvolusi dengan *bottleneck* 3x3 dimana konvolusi diawali dengan konvolusi 1x1 pada *input* yang berfungsi untuk mengurangi *depth* yang menyebabkan konvolusi 3x3 dapat lebih ringan untuk memproses fitur konvolusi karena *depth* akan menjadi lebih rendah dan konvolusi 1x1 untuk *output* agar *depth* dapat kembali seperti *input,* setelah itu *ReLU* digunakan untuk fungsi aktivasi. Proses *bottleneck* dilakukan bertujuan untuk meningkatkan effisiensi dari *model. Layer* selanjutnya adalah fungsi *bottleneck* yang sama dilakukan sebanyak 11 kali (*MobilenetV3-Small*) dan 15 (*MobilenetV3-Large*), pada setiap *layer* hanya terdapat perbedaan pada fungsi aktivasi, ukuran *kernel,* dan parameter *stride*. Lalu terdapat proses *pooling* 7x7 yang berfungsi untuk mengurangi dimensi *feature maps*, yang dapat mengurangi jumlah komputasi yang dilakukan oleh perangkat dan mengurangi jumlah parameter yang dipelajari. Pada tahap selanjutnya terdapat konvolusi2d *kernel* 1x1, *no batch normalization*, dan *hard-swish* sebagai aktivasinya.




**Gambar 4 Ilustrasi *stride***

Pada arsitektur *Mobilenet, stride* mengacu pada langkah yang diambil oleh kernel selama operasi konvolusi pada lapisan konvolusi (*convolutional layer*). Dalam konvolusi *filter* berukuran kecil bergeser pada matriks citra (*input*) untuk menghitung *output* yang sesuai. *Stride* mengontrol seberapa besar jarak (piksel) yang akan dilompati oleh *filter* setiap kali bergeser ke samping atau ke bawah. *Stride* ini akan mempengaruhi ukuran dari *output* yang dihasilkan. Jika *stride* lebih besar dari 1, maka *output* akan memiliki dimensi yang lebih kecil dari *input,* karena *filter* melompat lebih jauh.

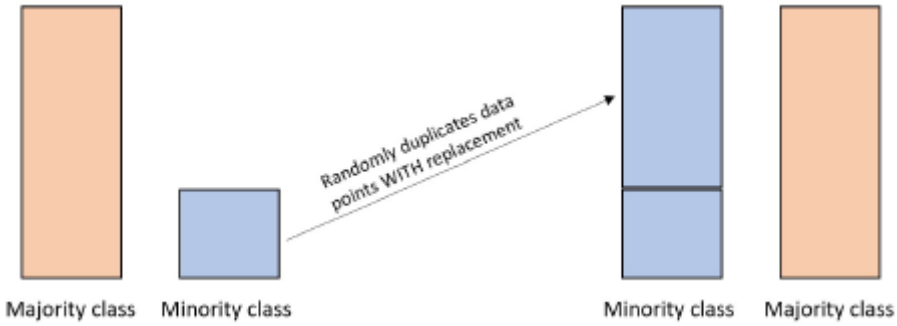
## 2.2. *Learning Rate*

*Learning rate* atau tingkat pembelajaran merupakan salah satu *hyperparameter* penting dalam algoritma optimisasi yang digunakan dalam proses pelatihan model. *Learning rate* menentukan seberapa besar langkah yang diambil oleh algoritma optimisasi untuk menyesuaikan bobot dan parameter *model* selama pelatihan. *Learning rate* mengontrol seberapa besar setiap gradien (turunan parsial dari fungsi *loss* terhadap parameter *model*) akan mempengaruhi perubahan bobot dan parameter. Dengan *learning rate* yang lebih tinggi, perubahan bobot akan lebih besar, dan sebaliknya, dengan *learning rate* yang lebih rendah, perubahan bobot akan lebih kecil.

Jika *learning rate* terlalu tinggi, algoritma optimisasi mungkin "terlompati" titik minimum dari fungsi *loss*, yang menyebabkan ketidakstabilan dan kesulitan untuk mencapai konvergensi yang baik. Di sisi lain, jika *learning rate* terlalu rendah, algoritma akan bergerak sangat lambat dan mungkin terjebak di suatu minimum lokal tanpa mencapai hasil yang optimal.

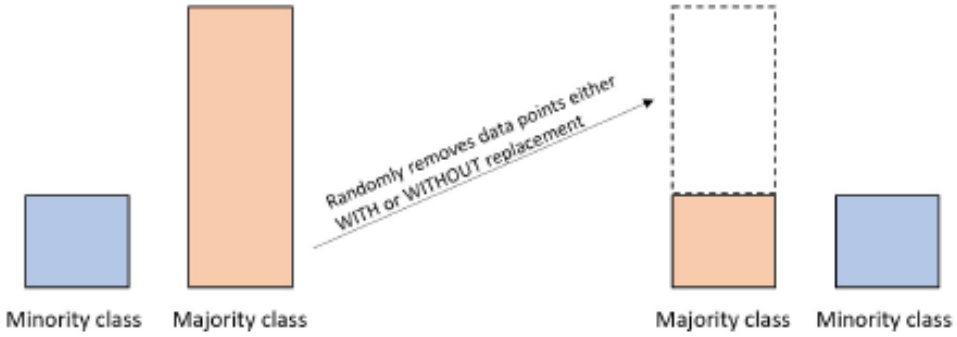
## 2.3. *Imbalanced Dataset*

Salah satu permasalahan pada *machine learning* yang cukup sering terjadi adalah ketidakseimbangan data yang digunakan atau sering disebut dengan *imbalanced dataset*. Cukup banyak penelitian yang melaporkan bahwa *imbalanced dataset* ini seringkali memberikan hasil yang keliru. Perlu ada penanganan khusus sebelum *imbalanced dataset* tersebut dapat digunakan pada *machine learning*. Cara paling populer dan efektif dalam mengatasi permasalahan *imbalanced dataset* adalah melakukan *resampling*, baik *oversampling*, *undersampling*, ataupun kombinasi keduanya (Indrawati, 2021). Pada penelitian ini metode *balancing data* yang digunakan adalah *Random oversampling* (ROS) dan *Random undersampling* (RUS).




**Gambar 5 Proses *random oversampling* (ROS) (Wongvorachan, He, and Bulut 2023)**

Seperti yang terlihat pada gambar 5, ROS beroperasi dengan cara menduplikasi data secara acak data kelas minoritas sampai proporsi kedua kelas seimbang. Dengan kata lain, data didalam kelas minoritas akan direplikasi secara acak sampai jumlahnya sesuai dengan data dari kelas mayoritas. Namun, data yang di duplikasi oleh ROS kemungkinan akan memiliki nilai yang mirip dan dapat meningkatkan kemungkinan *overfitting*. Selain itu ROS juga meningkatkan waktu *training* dikarenakan data pada *model* menjadi lebih besar dengan data tambahan pada kelas minoritas **(Wongvorachan, He, and Bulut 2023)**.




**Gambar 6 Proses *random undersampling* (RUS) (Wongvorachan, He, and Bulut 2023)**

Berbeda dengan ROS, *random undersampling* (RUS) beroperasi dengan secara acak menghapus data pada kelas mayoritas, baik dengan atau tanpa pengganti, sampai proporsi seluruh kelas seimbang. RUS mengurangi waktu *training* pada *model* dikarenakan oleh jumlah dataset yang berkurang. Namun, dengan mengurangi jumlah dari dataset, RUS memiliki kemungkinan untuk melewatkan informasi dan pola yang dapat membantu prediksi, oleh karena itu RUS dapat mengurangi akurasi prediksi **(Wongvorachan, He, and Bulut 2023)** proses *random undersampling* dapat dilihat pada gambar 6.

## 2.4. *Dataset*

*Input dataset* yang digunakan pada penelitian ini dikumpulkan didapatkan dari Kaggle yang diunggah oleh Derek Kuno-Williams serta dari *Science Data Bank* yang diunggah oleh Yao Zhixin dan Zhang Taihong. *Dataset* terdiri dari 6 kelas yaitu *Agaricus Edible, Agaricus Poisonous, Amanita Edible, Amanita Poisonous, Russula Edible,* dan *Russula Poisonous.*

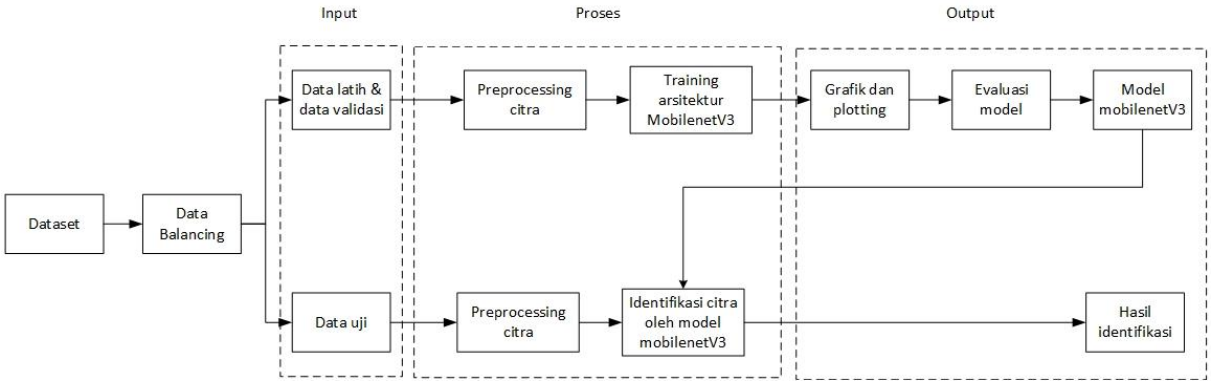
*Dataset* citra dibagi menjadi 3 yaitu data latih, data validasi, dan data uji dengan persentase pembagian data untuk data latih sebesar 80%, data validasi 10%, dan data uji 10% dari keseluruhan jumlah data sebanyak 5185. Pembagian *dataset* dapat dilihat pada tabel 1.

**Tabel 1 Pembagian *dataset***

| Kelas | Tipe data | | |
|---|---|---|---|
| | Latih | Validasi | Uji |
| *Agaricus Edible* | 756 | 252 | 252 |
| *Agaricus Poisonous* | 184 | 62 | 62 |
| *Amanita Edible* | 277 | 92 | 92 |
| *Amanita Poisonous* | 644 | 215 | 215 |
| *Russula Edible* | 1034 | 349 | 349 |
| *Russula Poisonous* | 210 | 70 | 70 |
| Total | 3105 | 1040 | 1040 |

## 2.5. *Block Diagram* Sistem

Pada perancangan model indentifikasi genus jamur konsumsi atau beracun ini terdapat beberapa tahap yang telah dilakukan, dapat dilihat pada gambar 7.




**Gambar 7 *Block diagram* sistem**

*Input* berupa citra yang didapatkan dari Kaggle dan *Science Data Bank.* Sebelum dilakukan *split data,* dilakukan *data balancing* terlebih dahulu menggunakan teknik *oversampling* dengan metode *random oversampling* (ROS) dan teknik *undersampling* dengan metode *random undersampling* (RUS) sehingga data menjadi seimbang. Setelah dilakukan *split data*, pada data latih dan data uji dilakukan *preprocessing* pada citra, dengan metode *resize* menjadi 224x224 dengan 3 *channel* warna RGB (*Red, Green, Blue*). Pada tahap *preprocessing* ini disesuaikan formatnya dengan format citra pada model *Mobilenet*. Kemudian pada proses selanjutnya menggunakan *Convolutional Neural Network* arsitektur *MobilenetV3. Output* yang didapatkan adalah hasil identifikasi genus jamur. Terakhir, dilakukan evaluasi model menggunakan *confusion matrix* lalu dilakukan *plotting* pada hasil yang didapatkan untuk mengetahui performa yang dihasilkan.

## 2.6. Metrik Evaluasi Performa

Evaluasi model dilakukan untuk mengetahui seberapa optimal performa model yang telah dirancang pada penelitian ini. Berdasarkan **(Agarwal, Gupta, and Biswas 2020)** metode metrik evaluasi dapat diperoleh dengan metode sebagai berikut.

*Accuracy* adalah kriteria utama dalam evaluasi seberapa efisiensi model. Model yang baik adalah model yang memiliki *accuracy* yang tinggi. Dapat dihitung menggunakan persamaan dibawah ini.

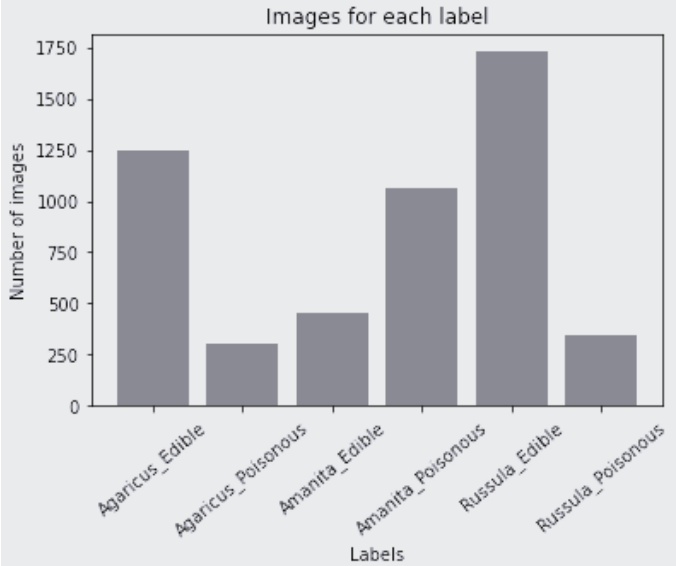
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- TP (True Positive): perkiraan banyaknya data positif dan terbukti kebenarannya
- FP (False Positive): perkiraan banyaknya data negatif dan terbukti kebenarannya
- TN (True Negative): perkiraan banyaknya data positif dan tidak terbukti kebenarannya
- FN (False Negative): perkiraan banyaknya data negatif dan tidak terbukti kebenarannya

## 3. HASIL DAN PEMBAHASAN

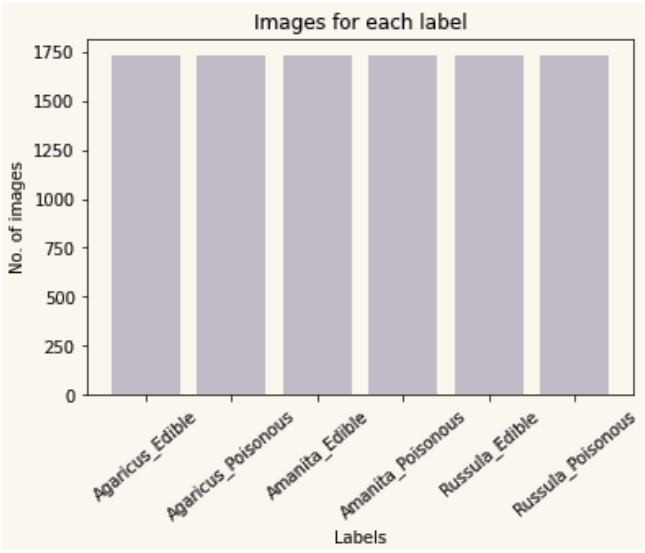
### 3.1. Pemrosesan *Dataset*

Citra pada *dataset* yang digunakan adalah citra dengan 3 buah *channel* warna yaitu RGB (*Red, Green, Blue*). Kemudian dilakukan *balancing data* untuk menyeimbangkan jumlah data pada kelas *dataset* menggunakan Teknik *oversampling* dan *undersampling.*



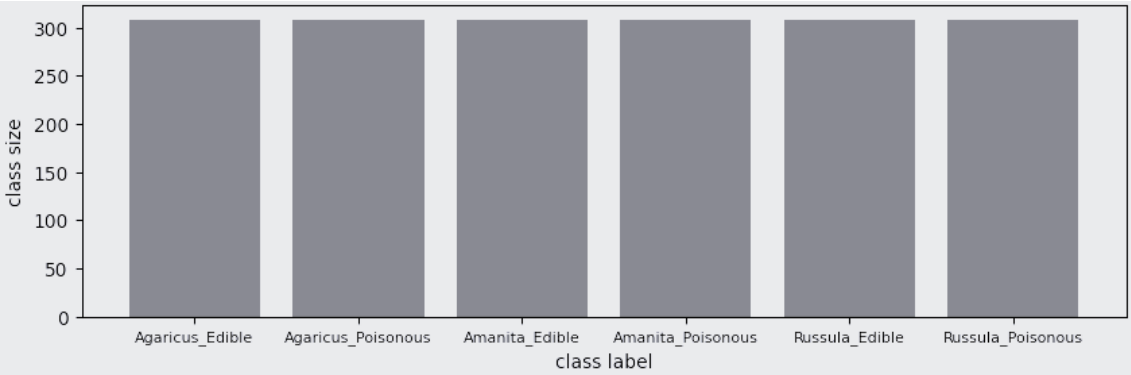

**Gambar 8 Grafik jumlah data pada tiap kelas**

Pada gambar 8 menunjukan bahwa jumlah persebaran data pada tiap kelas *dataset* terlihat tidak seimbang, dengan data pada kelas mayoritas yaitu *Russula Edible* berjumlah 1732 citra, sedangkan pada kelas minoritas yaitu *Agaricus Poisonous* berjumlah 308 citra. Oleh karena itu dilakukan *balancing data* untuk menyeimbangkan setiap kelas pada *dataset* dengan tujuan meningkatkan akurasi dari model.




**Gambar 9 Grafik jumlah data pada tiap kelas setelah *oversampling***

Gambar 9 menunjukan grafik jumlah data pada tiap kelas *dataset* setelah dilakukan *balancing data* menggunakan metode *random oversampling* (ROS). Dengan jumlah citra pada setiap kelas berjumlah 1732 citra berdasarkan kelas mayoritas yaitu *Russula Edible* dan total 10392 untuk seluruh data citra. Setelah itu dilakukan *split data* untuk proses pelatihan dengan rasio 80% untuk data latih, 10% untuk data validasi, dan 10% untuk data uji. Data dibagi menjadi 3 yaitu data latih dengan jumlah 1040 citra, data validasi dengan jumlah 346 citra, dan data uji dengan jumlah 346 citra.




**Gambar 10 Grafik jumlah data pada tiap kelas setelah *undersampling***

Gambar 10 menunjukan grafik jumlah data pada tiap kelas *dataset* setelah dilakukan *balancing data* menggunakan metode *random undersampling* (RUS). Dengan jumlah citra pada setiap kelas berjumlah 308 citra berdasarkan kelas minoritas yaitu *Agaricus Poisonous* dan total 1848 untuk seluruh data citra. Setelah itu dilakukan *split data* untuk proses pelatihan dengan rasio 80% untuk data latih, 10% untuk data validasi, dan 10% untuk data uji. Data dibagi menjadi 3 yaitu data latih dengan jumlah 246 citra, data validasi dengan jumlah 31 citra, dan data uji dengan jumlah 31 citra.

## 3.2. Pengujian Model

Pengujian dilakukan dengan *hyperparameter* berupa *batch size* 32, *learning rate* 0.0001 dan 0.00001, *optimizer Adam, epoch* 20 dan 30. Dengan menggunakan *stride* 1 dan 2 pada arsitektur serta uji coba dilakukan dengan *MobilenetV3-Large* dan *MobilenetV3-Small*. Berikut adalah hasil dari pengujian model.

Tabel 2 menunjukan hasil dari pengujian model menggunakan arsitektur *MobilenetV3-Large* dengan *batch size* 32, *learning rate* 0.0001 dan 0.00001, *dataset* yang digunakan yaitu normal (*Imbalanced data*), *oversampling,* dan *undersampling.* Sedangkan untuk *stride* nya yaitu 1 dan 2. Pada model arsitektur menggunakan model *MobilenetV3-Large* terlihat perbedaan signifikan dari waktu/*epoch* dan akurasi jika dilakukan *data balancing,* sedangkan *stride* juga berpengaruh sedikit dalam kinerja waktu/*epoch* dan akurasi.

**Tabel 2 Pengujian dengan *mobilenetv3-large***

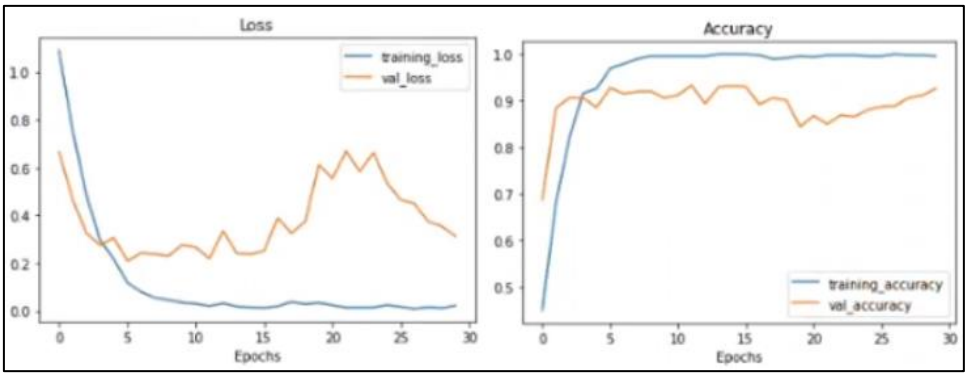
| Batch Size | LR | Epoch | S/L | Dataset | Stride | Akurasi | Waktu/Epoch |
|---|---|---|---|---|---|---|---|
| 32 | 0,0001 | 20 | L | Normal | 1 | 0,9962 | 133s |
| 32 | 0,0001 | 20 | L | Normal | 2 | 0,9870 | 131s |
| **32** | **0,0001** | **30** | **L** | **Normal** | **1** | **0,9981** | **137s** |
| 32 | 0,0001 | 30 | L | Normal | 2 | 0,9971 | 135s |
| 32 | 0,00001 | 20 | L | Normal | 1 | 0,8558 | 132s |
| 32 | 0,00001 | 20 | L | Normal | 2 | 0,8644 | 98s |
| 32 | 0,00001 | 30 | L | Normal | 1 | 0,9481 | 111s |
| 32 | 0,00001 | 30 | L | Normal | 2 | 0,9375 | 98s |
| 32 | 0,0001 | 20 | L | Over | 1 | 0,9880 | 192s |
| 32 | 0,0001 | 20 | L | Over | 2 | 0,9894 | 188s |
| 32 | 0,0001 | 30 | L | Over | 1 | 0,9952 | 180s |
| 32 | 0,0001 | 30 | L | Over | 2 | 0,9937 | 177s |
| 32 | 0,00001 | 20 | L | Over | 1 | 0,7052 | 182s |
| 32 | 0,00001 | 20 | L | Over | 2 | 0,6980 | 178s |
| 32 | 0,00001 | 30 | L | Over | 1 | 0,8039 | 190s |
| 32 | 0,00001 | 30 | L | Over | 2 | 0,7828 | 185s |
| 32 | 0,0001 | 20 | L | Under | 1 | 0,9516 | 20s |
| 32 | 0,0001 | 20 | L | Under | 2 | 0,9731 | 23s |
| 32 | 0,0001 | 30 | L | Under | 1 | 0,9892 | 19s |
| 32 | 0,0001 | 30 | L | Under | 2 | 0,9839 | 17s |
| 32 | 0,00001 | 20 | L | Under | 1 | 0,7528 | 22s |
| 32 | 0,00001 | 20 | L | Under | 2 | 0,5645 | 19s |
| 32 | 0,00001 | 30 | L | Under | 1 | 0,5860 | 20s |
| 32 | 0,00001 | 30 | L | Under | 2 | 0,5269 | 18s |

Tabel 3 menunjukan hasil dari pengujian model menggunakan arsitektur *MobilenetV3-Small* dengan *batch size* 32, *learning rate* 0.0001 dan 0.00001, *dataset* yang digunakan yaitu data normal (*Imbalanced data*), *oversampling,* dan *undersampling.* Untuk *stride* nya yaitu 1 dan 2.

**Table 3 Pengujian dengan *mobilenetv3-small***

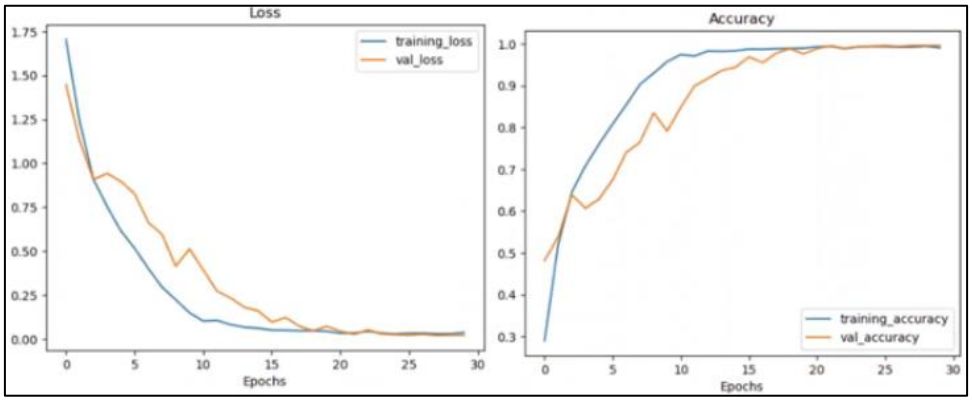
| Batch Size | LR | Epoch | S/L | Dataset | Stride | Akurasi | Waktu/Epoch |
|---|---|---|---|---|---|---|---|
| 32 | 0,0001 | 20 | S | Normal | 1 | 0,9942 | 58s |
| 32 | 0,0001 | 20 | S | Normal | 2 | 0,9913 | 50s |
| 32 | 0,0001 | 30 | S | Normal | 1 | 0,9962 | 61s |
| 32 | 0,0001 | 30 | S | Normal | 2 | 0,9981 | 59s |
| 32 | 0,00001 | 20 | S | Normal | 1 | 0,8038 | 56s |
| 32 | 0,00001 | 20 | S | Normal | 2 | 0,7865 | 42s |
| 32 | 0,00001 | 30 | S | Normal | 1 | 0,8596 | 48s |
| 32 | 0,00001 | 30 | S | Normal | 2 | 0,8760 | 45s |
| 32 | 0,0001 | 20 | S | Over | 1 | 0,9432 | 97s |
| 32 | 0,0001 | 20 | S | Over | 2 | 0,9566 | 104s |
| 32 | 0,0001 | 30 | S | Over | 1 | 0,9904 | 145s |
| 32 | 0,0001 | 30 | S | Over | 2 | 0,9937 | 112s |
| 32 | 0,00001 | 20 | S | Over | 1 | 0,5925 | 89s |
| 32 | 0,00001 | 20 | S | Over | 2 | 0,6305 | 83s |
| 32 | 0,00001 | 30 | S | Over | 1 | 0,7057 | 79s |
| 32 | 0,00001 | 30 | S | Over | 2 | 0,6922 | 78s |
| 32 | 0,0001 | 20 | S | Under | 1 | 0,9409 | 21s |
| 32 | 0,0001 | 20 | S | Under | 2 | 0,9677 | 16s |
| 32 | 0,0001 | 30 | S | Under | 1 | 0,9731 | 23s |
| 32 | 0,0001 | 30 | S | Under | 2 | 0,9946 | 22s |
| 32 | 0,00001 | 20 | S | Under | 1 | 0,5108 | 20s |
| **32** | **0,00001** | **20** | **S** | **Under** | **2** | **0,4247** | **18s** |
| 32 | 0,00001 | 30 | S | Under | 1 | 0,5914 | 16s |
| 32 | 0,00001 | 30 | S | Under | 2 | 0,4892 | 17s |

Akurasi terbesar yang diperoleh yaitu sebesar 99.81 dengan waktu/*epoch* sebesar 137s didapat pada arsitektur *MobilenetV3-Large, learning rate* 0.0001, *epoch* 30, *dataset* normal, dan *stride* 1. Namun, ketika dilakukan *plotting* grafik ternyata ada sedikit indikasi terjadinya *overfitting* pada pengujian tersebut ditampilkan pada gambar 11, hal ini bisa disebabkan karena perbedaan jumlah *dataset* dari data validasi dengan data latih terpaut jauh.



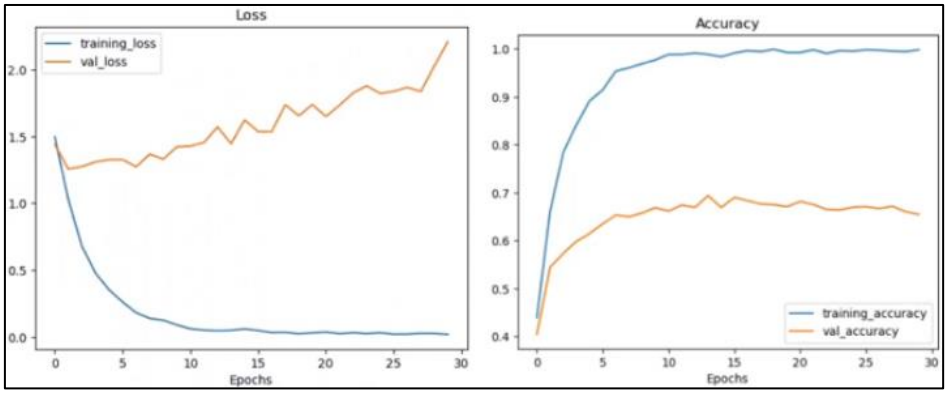

**Gambar 11 Grafik pelatihan dataset normal**

Lalu ketika melakukan pengujian menggunakan *dataset oversampling* dengan *hyperparameter* yang sama menghasilkan akurasi sebesar 99.52, akurasinya menurun dikarenakan data sintetis yang menambahkan jumlah *noise* sehingga membuat model mempelajari *noise* tersebut, dengan waktu/*epoch* sebesar 180s termasuk waktu yang lama untuk pelatihan arsitektur *Mobilenet* hal ini bisa disebabkan karena jumlah *dataset* yang banyak yaitu berjumlah 10392 citra, namun hasil *plotting* grafik mayoritas menunjukan hasil yang tidak *overfitting* karena jumlah data validasi yang mencukupi, bisa dilihat pada gambar 12.




**Gambar 12 Grafik pelatihan dataset oversampling**

Akurasi paling rendah didapat pada pengujian menggunakan arsitektur *MobilenetV3-Small, learning rate* 0.00001, *epoch* 20, *dataset undersampling*, dan *stride* 2 diperoleh akurasi sebesar 0.4247 dengan waktu/*epoch* sebesar 18s. Seperti yang dapat dilihat pada tabel 3.




**Gambar 13 Grafik pelatihan dataset undersampling**

## 4. KESIMPULAN

Berdasarkan penelitian yang telah dilakukan, dapat diambil kesimpulan bahwa arsitektur *MobilenetV3* merupakan metode *convolutional neural network* yang dapat digunakan untuk melakukan klasifikasi serta mempelajari struktur jamur beracun dan jamur yang dapat dikonsumsi. Arsitektur *MobilenetV3* memiliki kinerja yang baik dalam hal akurasi yaitu dengan akurasi tertinggi sebesar 99.81% ketika diimplementasikan untuk mengklasifikasi jamur beracun dan jamur yang dapat dikonsumsi pada perangkat *mobile* dengan jumlah *dataset* sebanyak 5199 citra. Ketika jumlah *dataset* ditingkatkan, kinerja sistem mengalami peningkatan dalam hal akurasi namun dengan beban komputasi yang meningkat juga. Dengan menggunakan teknik *data balancing* yaitu *oversampling,* model dapat meminimalisir terjadinya *overfitting* dan *underfitting.* Sedangkan jika menggunakan *undersampling,* kinerja sistem mengalami sedikit penurunan akurasi dan model rentan mengalami *overfitting* dan *underfitting.*

## DAFTAR PUSTAKA

Mohit Agarwal, Suneet Kr. Gupta, K.K. Biswas. (2020). Development of Efficient CNN model for Tomato crop disease. *Journal Pre-proof*.

Andrew Howard et al. (2019). Searching for MobileNetV3. *ICCV*, 1314-1318.

Darlis, A. R. (2015). Impementation Visible Light COmmunication. *International Optical Conference* (pp. 200 - 209). Bandung: Institut Teknologi Nasional Bandung.

Elok Iedfitra Haksoro, Abas Setiawan. (2021). PENGENALAN JAMUR YANG DAPAT DIKONSUMSI MENGGUNAKAN METODE TRANSFER LEARNING. *Jurnal ELTIKOM*, 81-91.

Frencis Matheos Sarimole, Randitia Ridad Diadi. (2022). KLASIFIKASI JENIS JAMUR MENGGUNAKAN EKSTRAKSI FITUR. *JINTEKS (Jurnal Informatika Teknologi dan Sains)*, 286 – 290.

Huili Li, dkk. (2020). Reviewing the world's edible mushroom species: A new. *COMPREH ENSIVE REVIEWS IN FOOD SCIENCE AND FOOD SAFETY*, 1987-1990.

Iin Annissa, Hanna Artuti Ekamawanti, Wahdina. (2017). KEANEKARAGAMAN JENIS JAMUR MAKROSKOPIS DI ARBORETUM SYLVA UNIVERSITAS TANJUNGPURA. *JURNAL HUTAN LESTARI (2017)*, Vol. 5 (4) : 969 - 977.

Indrawati, A. (2021). PENERAPAN TEKNIK KOMBINASI OVERSAMPLING DAN UNDERSAMPLING UNTUK MENGATASI PERMASALAHAN IMBALANCED DATASET. *JIKO (Jurnal Informatika dan Komputer)*.

Muhammad Iqbal Izzul Haq, Dini Adni Navastara, dan Shintami Chusnul Hidayati. (2023). Deteksi Ucapan untuk Sistem Pengawasan Asesmen (iProctor) Menggunakan Metode Deep Learning. *JURNAL TEKNIK ITS*, 70.

Putra, I. P. (2021). LAPORAN KASUS KERACUNAN Chlorophyllum cf. molybdites DI SURABAYA, INDONESIA. *JURNAL AGERCOLERE*, 1-6.

Putri Nada Zakiya, Ledya Novamizanti, Syamsul Rizal. (2021). KLASIFIKASI PATOLOGI MAKULA RETINA MELALUI CITRA OCT MENGGUNAKAN CONVOLUTIONAL NEURAL NETWORK DENGAN ARSITEKTUR MOBILENET. *e-Proceeding of Engineering*, 5072.

Salafy, M. F. (2022). PERBANDINGAN PERFORMA DETEKSI PADA FORWARD COLLISION WARNING SYSTEM MENGGUNAKAN METODE MOBILENETV3 DAN YOLOV4 PADA PERANGKAT RASPBERRY PI 4.

Tarid Wongvorachan, Surina He, Okan Bulut. (2023). A Comparison of Undersampling, Oversampling, and SMOTE Methods for Dealing with Imbalanced Classification in Educational Data Mining. *MDPI*.