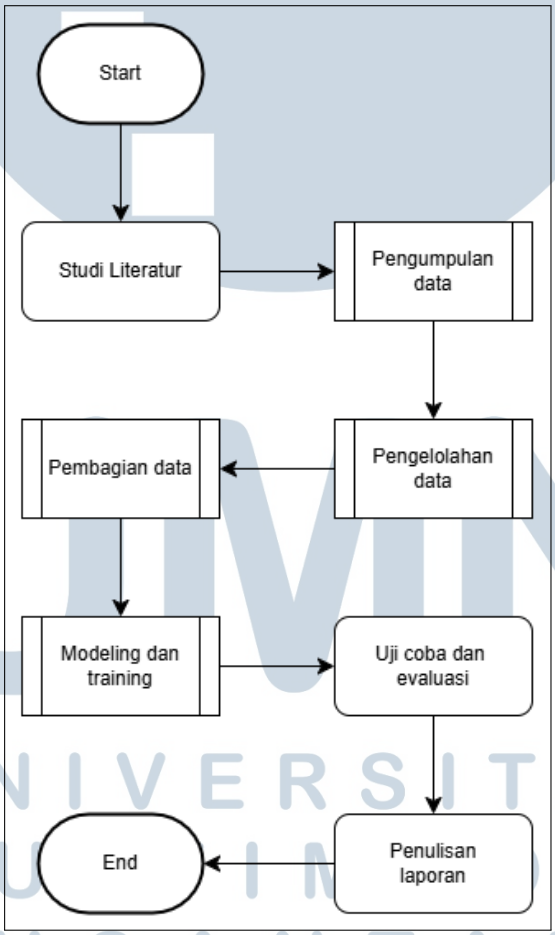
BAB 3
METODOLOGI PENELITIAN

3.1 Gambaran umum

Pertama, data dikumpulkan dan diproses sebelum digunakan dalam pengujian. Setelah itu, proses *modeling* dan pelatihan model dilakukan. Untuk mengevaluasi model yang sudah dilatih, *matrix confusion* akan digunakan untuk menghitung nilai *accuracy, precision, recall*, dan f1 *score*. Setelah itu, dilakukan analisis mencari model yang memiliki nilai *accuracy, precision, recall*, dan f1 *score* tertinggi. Susunan metodologi dapat dilihat pada Gambar 3.1.



Gambar 3.1. *Flowchart* metodologi

3.2 Spesifikasi System

Spesifikasi perangkat keras dan perangkat lunak yang digunakan untuk melakukan implementasi algoritma *Convolutional Neural Network* untuk melakukan identifikasi daging sapi segar dan beku yang dicairkan. Spesifikasi perangkat lunak dan perangkat keras yang digunakan dilampirkan sebagai berikut.

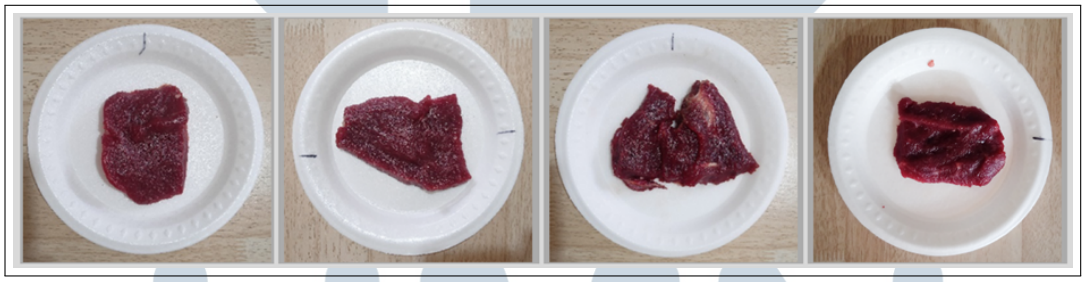
1. Perangkat Lunak
    - Sistem Operasi : Windows 10 64 bit
    - Text Editor : PyCharm
    - IDE : Anaconda3
    - python : 3.11.5
    - TensorFlow : 2.15.0
2. Perangkat Keras
    - Prosesor : Intel Core i7 8750H
    - VGA : NVIDIA GeForce GTX 1050 Ti (4GB GDDR5)
    - RAM : 16 GB Dual Channel
    - local Storage : SSD nvme m.2 512 GB

3.3 Studi Literatur

Tahap untuk mencari informasi mengenai topik penelitian yang dilakukan untuk mempelajari materi sebelum membuat perancangan sistem, studi literatur adalah tahap pertama yang harus dijalani. Studi literatur ini juga mencakup pencairan referensi untuk membantu membuat dasar analisis. Beberapa teori yang dipelajari dan dicari seperti ciri dan perbedaan daging sapi segar dan beku yang dicairkan, *Machine Learning, Convolutional Neural Network (CNN), Transfer Learning* dan *Confusion Matrix*. Studi literatur ini akan berguna dan berfungsi sebagai landasan teori dan sebagai dasar pemahaman mengenai objek dan metode yang akan digunakan. Informasi dicari dan dikumpulkan dari berbagai sumber seperti karya ilmiah, buku, jurnal ilmiah, dan lainnya.

3.4 Pengumpulan Data

Dataset yang digunakan dalam penelitian menggunakan dua *dataset* berbeda dan hanya mengambil gambar kelas tertentu untuk dijadikan satu *dataset*. *Dataset* pertama yang digunakan adalah *dataset LOCBEEF Meat Quality Image* dan *Images of fresh and non-fresh beef meat samples* yang diambil dari data Mendeley. Pada *dataset LOCBEEF Meat Quality Image* terdapat 3.268 gambar dari daging sapi lokal dari Aceh dan diambil pada pukul tujuh pagi sampai sepuluh malam. *Dataset LOCBEEF Meat Quality Image* terdapat dua jenis data yaitu *train* berjumlah 2228 gambar dan *test* 980 gambar, berisi data *fresh* dan *rotten* berjumlah 1114 gambar *train* dan 490 gambar *test* dengan resolusi gambar mulai dari 176 x 144 *pixel*, 320 x 240 *pixel*, 640 x 480 *pixel*, 720 x 480 *pixel*, 720 x 720 *pixel*, 1280 x 720 *pixel*, 1920 x 1080 *pixel*, 2560 x 1920 *pixel*, 3120 x 3120 *pixel*, 3264 x 2248 *pixel*, dan 4160 x 3120 *pixel*. Gambar *dataset LOCBEEF Meat Quality Image* dapat dilihat pada Gambar 3.2.



Gambar 3.2. LOCBEEF *meat quality image*
Sumber: [16]

Komposisi pada *Dataset LOCBEEF Meat Quality Image* menggunakan 70% *training* data dan 30% *test* data. Pada *Dataset LOCBEEF Meat Quality Image* mengambil gambar kelas *fresh* untuk digunakan sebagai *dataset* daging sapi *fresh* untuk penelitian yang berjumlah 1.604 gambar.

Dataset kedua yang digunakan adalah *Dataset Images of fresh and non-fresh beef meat samples*. Pada *Dataset Images of fresh and non-fresh beef meat samples* terdapat total 120 gambar daging sapi yang sudah dipotong. *Dataset* menggunakan sepuluh potongan untuk setiap potongan dengan 5cm x 5cm. Gambar diambil pada hari pertama dan kelima setelah pembelian. *Dataset* terdiri dari 60 gambar asli dan 60 gambar yang telah diproses kalibrasi warna dan segmentasi, setengah dari gambar adalah daging segar setelah pembelian, dan setengah adalah daging non segar setelah lima hari pembelian. Karena jumlah data gambar yang lebih sedikit

sehingga menggunakan seluruh gambar setelah kalibrasi warna dan segmentasi untuk menjadi *dataset thawed* pada penelitian.

3.5 Pengelolaan Data

Setelah tahap pengumpulan data, dilanjutkan tahap pengelolaan data. Pada *Dataset LOCBEEF Meat Quality Image* gambar daging hanya sekitar 20% sampai 30% sedangkan sisanya bias pada gambar seperti *background* dan piring pada gambar. Hal ini dapat mempengaruhi proses identifikasi gambar. Berikut hasil training pada *dataset* tanpa melalui pengelolaan data.

Oprimizer RMSprop

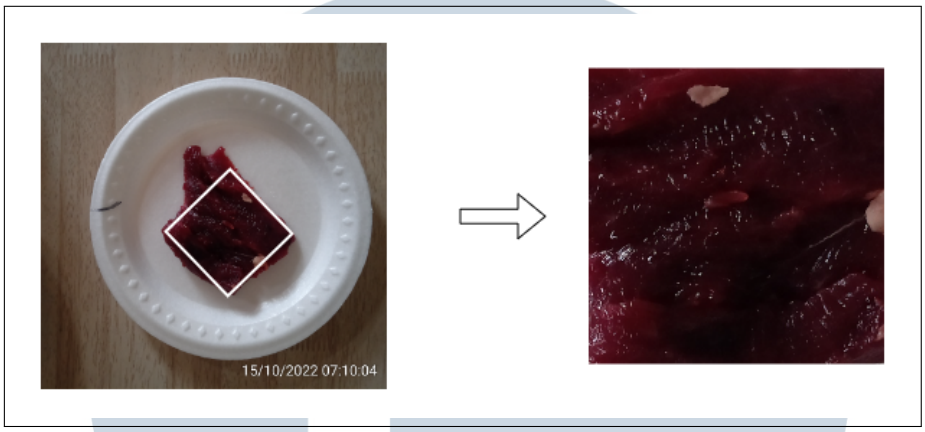
		Precision		Recall		F1-score		
Learning rate	Model	Fresh	Melted	Fresh	Melted	Fresh	Melted	Accuracy
0.1	CNN_16	0.00	0.50	0.00	1.00	0.00	1.00	0.50
	CNN_32	0.00	0.50	0.00	1.00	0.00	0.67	0.50
	CNN_64	0.50	0.00	1.00	0.00	0.67	0.00	0.50
0.01	CNN_16	0.00	0.53	0.00	1.00	0.00	0.69	0.53
	CNN_32	0.52	0.52	0.51	0.53	0.52	0.53	0.52
	CNN_64	0.48	0.48	0.48	0.48	0.48	0.48	0.48
0,001	CNN_16	0.45	0.52	0.33	0.64	0.38	0.57	0.49
	CNN_32	0.51	0.51	0.51	0.51	0.51	0.51	0.51
	CNN_64	0.50	0.50	0.50	0.50	0.50	0.50	0.50
0,0001	CNN_16	0.49	0.55	0.49	0.55	0.49	0.55	0.53
	CNN_32	0.47	0.47	0.47	0.47	0.47	0.47	0.47
	CNN_64	0.50	0.50	0.50	0.50	0.50	0.50	0.50
0,00001	CNN_16	0.47	0.53	0.47	0.53	0.47	0.53	0.51
	CNN_32	0.50	0.50	0.50	0.50	0.50	0.50	0.50
	CNN_64	0.49	0.49	0.49	0.49	0.49	0.49	0.49
0.000001	CNN_16	0.47	0.53	0.47	0.53	0.47	0.53	0.51
	CNN_32	0.52	0.52	0.52	0.52	0.52	0.52	0.52

Gambar 3.3. Hasil akurasi *train* tanpa pengelolaan data rmsprop model kedua

Pada Gambar 3.3 hasil *training* pada model sebelum pengelolaan data model tidak dapat mempelajari gambar dengan baik. Bisa dilihat dari Gambar 3.3 pada pengujian data, akurasi tertinggi terdapat pada *learning rate* 0.001 dengan F1 *score* 53%. Untuk 30 *epochs* dari akurasi dan *loss* pada *training* cukup baik, namun pada *loss* validasi dan akurasi validasi model tidak dapat mempelajari gambar dengan baik. Sehingga perlu dilakukan pengelolaan data sebelum proses *training* pada model.

Pada *Dataset LOCBEEF Meat Quality Image* pertama dilakukan proses *croping* pada gambar agar gambar pada *dataset fresh* penuh dengan gambar daging

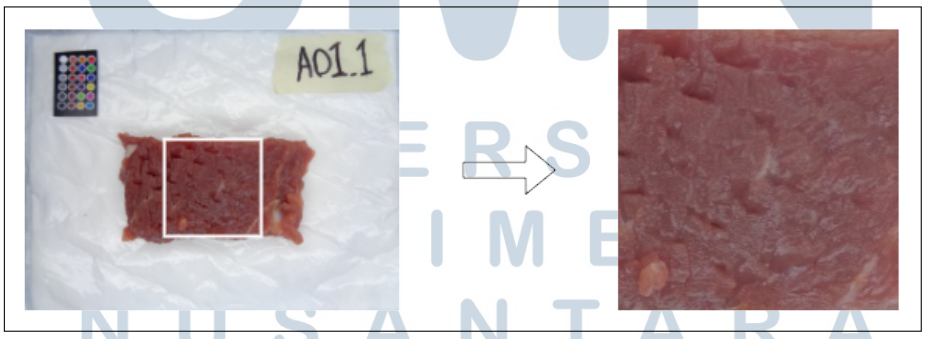
tanpa ada bias pada gambar dengan cara melakukan *croping* langsung pada gambar dagingnya saja.



Gambar 3.4. Dataset *fresh* sebelum dan sesudah *croping*

Pada Gambar 3.4 yang sebelumnya hanya sekitar 20 hingga 30% gambar daging saja dan lebih banyak ada pada objek piring, setelah dilakukan proses *croping* pada gambar menjadi gambar dengan seluruh bagian adalah daging. Proses *croping* ini dilakukan pada seluruh gambar *dataset fresh* namun ukuran pada setiap gambar tidak dapat sama, karena pada dokumentasi *dataset* ukurannya yang berbeda sehingga sulit untuk dilakukan *croping* dengan ukuran yang sama, namun untuk ketentuan *croping* masih sama yaitu seluruh bagian gambar adalah daging.

Pada *dataset Images of fresh and non-fresh beef meat samples* juga dilakukan proses *croping* pada seluruh gambar untuk menyamai gambar pada *dataset fresh* agar model dapat mendeteksi fitur pada daging dengan lebih baik. Berikut sebelum dan sesudah proses *croping* pada Gambar *dataset thawed*.



Gambar 3.5. Dataset *thawed* sebelum dan sesudah *croping*

Pada Gambar 3.5 dilakukan *croping* pada dagingnya saja seperti pada *dataset fresh* dan untuk ukuran *shape* pada gambar juga berbeda karena ukuran

daging yang difoto berbeda sehingga harus menyesuaikan dengan ukuran daging agar fitur pada gambar dapat masuk dalam gambar dengan baik.

Pada *dataset Images of fresh and non-fresh beef meat samples* karena jumlah data lebih sedikit dibandingkan dengan jumlah dari *dataset fresh*, salah satu cara untuk menutupi kekurangan tersebut adalah dengan *generate image data* dari data gambar yang sudah ada dengan penambahan variasi gambar hingga jumlahnya menyamai dengan *dataset fresh*. *ImageDataGenerator* menyediakan banyak parameter untuk menambah variasi data yang diinginkan.

```
import TensorFlow as tf
tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    zca_epsilon=1e-06,
    rotation_range=0,
    width_shift_range=0.0,
    height_shift_range=0.0,
    brightness_range=None,
    shear_range=0.0,
    zoom_range=0.0,
    channel_shift_range=0.0,
    fill_mode='nearest',
    cval=0.0,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=None,
    preprocessing_function=None,
    data_format=None,
    validation_split=0.0,
    interpolation_order=1,
    dtype=None
)
```

Kode 3.1: Contoh kode fungsi ImageDataGenerator pada TensorFlow Keras

Pada contoh Kode 3.1 fungsi *ImageDataGenerator* pada *TensorFlow* Keras banyak variasi yang dapat dilakukan mulai dari pengelolaan data sederhana seperti *zoom*, *vertikal* dan *horizontal flip* serta masih banyak lagi. Untuk melakukan *generate* data gambar pertama harus membuat dan melakukan *setting* variasi *augmentation* yang dipakai dalam *ImageDataGenerator* data. Berikut variasi yang digunakan pada *dataset thawed* setelah dilakukan *croping*.

```
import TensorFlow as tf
# Konfigurasi augmentasi
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
            rotation_range=40,
            width_shift_range=0.2,
            height_shift_range=0.2,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True,
            vertical_flip=True,
            fill_mode='nearest'
        )
```

Kode 3.2: Konfigurasi ImageDataGenerator dataset thawed

Pada potongan Kode 3.2 *dataset thawed* dilakukan *augmentation* data dengan rotasi sebesar 40 derajat, *widith shift range* dan *height shift range* sebesar 20%, *shear range* sebesar 20%, pembesaran gambar sebesar 20%, dengan penambahan dibalik horizontal dan vertikal, untuk bagian yang tidak terkena gambar ditambahkan *fill mode* untuk mengisi gambar yang kosong dengan warna terdekat atau *nearest*. Pada dokumentasi tutorial Keras juga disarankan untuk menggunakan *data augmentation* yang lebih agresif [41]. Sehingga pada konfigurasi *ImageDataGenerator* ditambahkan konfigurasi *vertical flip* pada penelitian yang akan dilakukan. Setelah menentukan *augmentation* gambar yang ingin digunakan selanjutnya membuat kode untuk melakukan *generate* pada seluruh gambar *dataset thawed* secara berulang dan menyimpannya ke dalam *file* sementara.

```
import os
# List all file to folder melted_data
file_list = os.listdir(melted_data)

# Loop file
for filename in file_list:
    if filename.lower().endswith(('png', 'jpg', 'jpeg')):
        try:
            img_path = os.path.join(melted_data, filename)
            img = load_img(img_path)
            img_array = img_to_array(img)
            img_array = img_array.reshape((1,) + img_array.shape)

            # Make augmented images and save
            i = 0
            for batch in datagen.flow(img_array, batch_size=1,
```

```
                                        save_to_dir=output_dir,
                                        save_prefix='Melted',
                                        save_format='jpg'):
                i += 1
                if i >= 25:  # Save 25 augmented images
                    break
        except Exception as e:
            logging.error(f"Error processing file {filename}: {e}"
)
```

Kode 3.3: Potongan kode perulangan augmentasi

Pada potongan Kode 3.3 pertama membuat *file* sementara untuk menyimpan gambar asli *thawed* dengan *library* os dengan nama *File_list* lalu masuk ke proses perulangan pertama di mana selama nama *file* dalam *File_list* dimulai dengan huruf kecil dan diakhiri dengan format *file png*, *jpg*, dan *jpeg*, selanjutnya mencoba untuk menyusun *path* lengkap dari gambar, memuat gambar menggunakan fungsi *load_img*, setelah itu mengubah gambar ke *array* menggunakan fungsi *img_to_array* dan melakukan *reshape array* pada tinggi, lebar dan *color channel* gambar sama seperti aslinya untuk dipersiapkan ke generator.

Dalam perulangan kedua membuat *augmentation* gambar dengan fungsi *datagen.flow* menghasilkan *augmentation* gambar dari fungsi *img_array* sebelumnya. Hasil *augmentation* gambar disimpan dalam *file* dengan fungsi *output_dir* nama gambar *melted* dan format *jpg*. Perulangan berhenti setelah menghasilkan 25 *augmentation* gambar untuk setiap gambar asli. sehingga menghasilkan 1.500 gambar karena untuk *dataset thawed* hanya menggunakan 60 gambar dari *dataset*. Berikut beberapa hasil dari *generate* gambar menggunakan *ImageDataGenerator* dari *tensorflow keras*.

(a) Melted_0_12 (b) Melted_0_740 (c) Melted_0_269

Gambar 3.6. Contoh gambar hasil *ImageDataGenerator*

Setelah melalui proses *croping* Gambar 3.4, 3.5 dan *generate* Gambar 3.6 selanjutnya masuk ke tahap pembagian data.

3.6 Pembagian Data

Setelah dilakukan proses pengelolaan data, selanjutnya dilakukan pembagian data di mana membagi jumlah data untuk *train*, *validation* dan *testing*. Untuk pembagian data *training* dan testing dilakukan secara manual dengan membagi *dataset* menjadi tiga bagian yaitu *Train, test* dan validasi. Pada setiap bagian data terdapat dua *file* kelas yaitu *fresh* untuk daging sapi segar dan *melted* daging sapi beku yang telah dicairkan. Data pelatihan membantu model memahami data untuk keperluan klasifikasi. Data validasi mengevaluasi kinerja model selama proses pelatihan, tetapi tidak digunakan untuk melatih model itu sendiri. Pengujian data adalah tahap terakhir untuk menguji kemampuan model pada data yang sebelumnya tidak pernah dilihat dengan penambahan data dari luar *dataset* agar menambah variasi data *test* untuk prediksi model. Pada *dataset fresh* setelah dilakukan *croping* pada seluruh data, dibagi menjadi 70% data *train* dengan jumlah 1033 gambar, 15% data validasi berjumlah 232 gambar, dan 15% data *test* berjumlah 232 gambar. Begitu juga untuk *dataset thawed* dibagi menjadi 70% data *train* dengan jumlah 1033 gambar, 15% data validasi berjumlah 232 gambar, dan 15% data *test* berjumlah 232 gambar.

3.7 Modeling CNN

Penelitian ini menggunakan model *CNN* yang telah disesuaikan khusus dengan tujuan dapat membuat model yang ringan, namun tetap memberikan akurasi yang tinggi tanpa mengalami *overfit*. Arsitektur model CNN menggunakan arsitektur *LeNet* dan *InceptionV3* dengan beberapa modifikasi untuk sesuai dengan *dataset* yang digunakan.

Karena *activation ReLU* memiliki keunggulan dibandingkan dengan *sigmoid* dan *tanh*, yaitu menghindari perhitungan eksponensial yang rumit, mengurangi jumlah perhitungan yang dilakukan, dan mencegah kerusakan *network* [13]. *ReLU* adalah fungsi *activation* komputasi yang efektif karena *neuron* tidak diaktifkan secara bersamaan seperti *tanh*. *ReLU* tidak hanya memberikan kesederhanaan empiris, tetapi juga memiliki kemungkinan yang lebih rendah untuk mengalami *vanishing gradient*. *ReLU* dipilih untuk digunakan dalam model ini

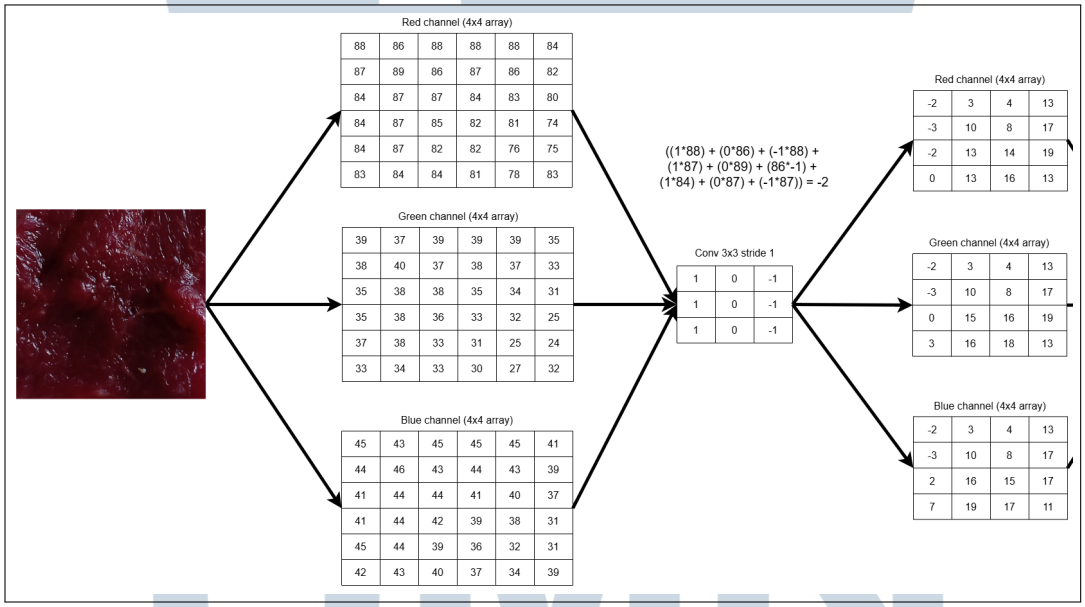
karena keuntungan tersebut. Bentuknya yang *linear* dan tidak jenuh memungkinkan *ReLU* untuk mempercepat konvergensi jaringan. Selain itu, mengaktifkan *ReLU* dapat membantu mengurangi kemungkinan *overfitting* [42].

Tabel 3.1. Arsitektur model CNN

Layer (type)	Output Shape	Param
conv2d (Conv2D)	(None, 222, 222, 16)	448
batch normalization (Batch Normalization)	(None, 222, 222, 16)	64
max pooling2d (MaxPooling2D)	(None, 111, 111, 16)	0
batch normalization (Batch Normalization)	(None, 111, 111, 16)	64
conv2d (Conv2D)	(None, 109, 109, 32)	4640
max pooling2d (MaxPooling2D)	(None, 54, 54, 32)	0
batch normalization (Batch Normalization)	(None, 54, 54, 32)	128
conv2d (Conv2D)	(None, 52, 52, 64)	18496
max pooling2d (MaxPooling2D)	(None, 26, 26, 64)	0
batch normalization (Batch Normalization)	(None, 26, 26, 64)	256
conv2d (Conv2D)	(None, 24, 24, 128)	73856
max pooling2d (MaxPooling2D)	(None, 12, 12, 128)	0
flatten (Flatten)	(None, 18432)	0
dense (Dense)	(None, 128)	2359424
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 2)	258

Pada Tabel 3.1 tersebut adalah model *LeNet* sederhana yang telah dimodifikasi khusus penelitian ini atau bisa dibilang menggunakan model *Simple* CNN. Filter pada model adalah 16 pada Conv2D pertama, 32 pada Conv2D kedua, 64 pada Conv2D ketiga, dan 168 pada Conv2D keempat, penambahan *batch normalization layer* sebelum *convolutional layer*, dilanjutkan dengan satu *flatten layer*, dua *hidden dense layer*, satu *dropout layer* dan satu *output layer*. Pada Conv2D bergerak dari lapisan *input* gambar ke lapisan berikutnya dalam model secara berurutan. Ukuran filter 3×3 digunakan dalam *convolution layer* untuk melakukan operasi *convolution* pada gambar. Fungsi *batch normalization* berguna untuk mengoptimalkan model, dan jika model sudah optimal maka akan mengembalikan *value* ke awal sebelum proses normalisasi. *Batch normalization*

biasanya digunakan setelah *dense layer* dan *activation*. Ukuran *pool* 2×2 digunakan dalam *pooling layer* untuk mengurangi dimensi *spasial* dari fitur yang diekstraksi oleh *convolution layer* sebelumnya. Dalam hal ini, setiap *pool size* 2×2 akan mengambil nilai maksimum dari setiap blok 2×2 dari fitur yang dilewati oleh *convolution layer* sebelumnya. Penggunaan *pool size* 2×2 membantu mengurangi dimensi fitur dan merangkum informasi yang penting dalam blok 2×2, sehingga mempercepat proses komputasi dan menghindari *overfitting* pada model. Dengan *input* gambar berukuran 224×224, maka gambar akan di-*flatten* menjadi vektor satu dimensi 3×3 setelah melewati semua *layer*. *Dropout layer* digunakan untuk mematikan *neuron* yang tidak dipakai agar mempercepat proses komputasi. Berikut metode yang dilakukan menggunakan perhitungan manual.



Gambar 3.7. Proses *convolutional*

Pada Gambar 3.7 gambar pertama diubah ke dalam bentuk *array* untuk setiap kode warna, lalu setiap *array* warna dikalikan dengan matriks konvolusi yang berisi nilai acak. Perhitungan pada Conv2D dengan matriks 3x3 dan *stride* satu dilakukan secara terus menerus pada ketiga matriks warna. Proses perkalian matriks pada lapisan konvolusi pada matriks *Red* sebagai berikut.

$(1\times88)+(0\times86)+(-1\times88)+(1\times87)+(0\times89)+(86\times-1)+(1\times84)+(0\times87)+(-1\times87)= -2$

$(-1\times86)+(0\times88)+(1\times88)+(-1\times89)+(0\times86)+(87\times-1)+(-1\times87)+(0\times87)+(87\times-1)= 3$

$(-1\times88)+(0\times88)+(1\times88)+(-1\times86)+(0\times87)+(86\times-1)+(-1\times87)+(0\times84)+(83\times-1)= 4$

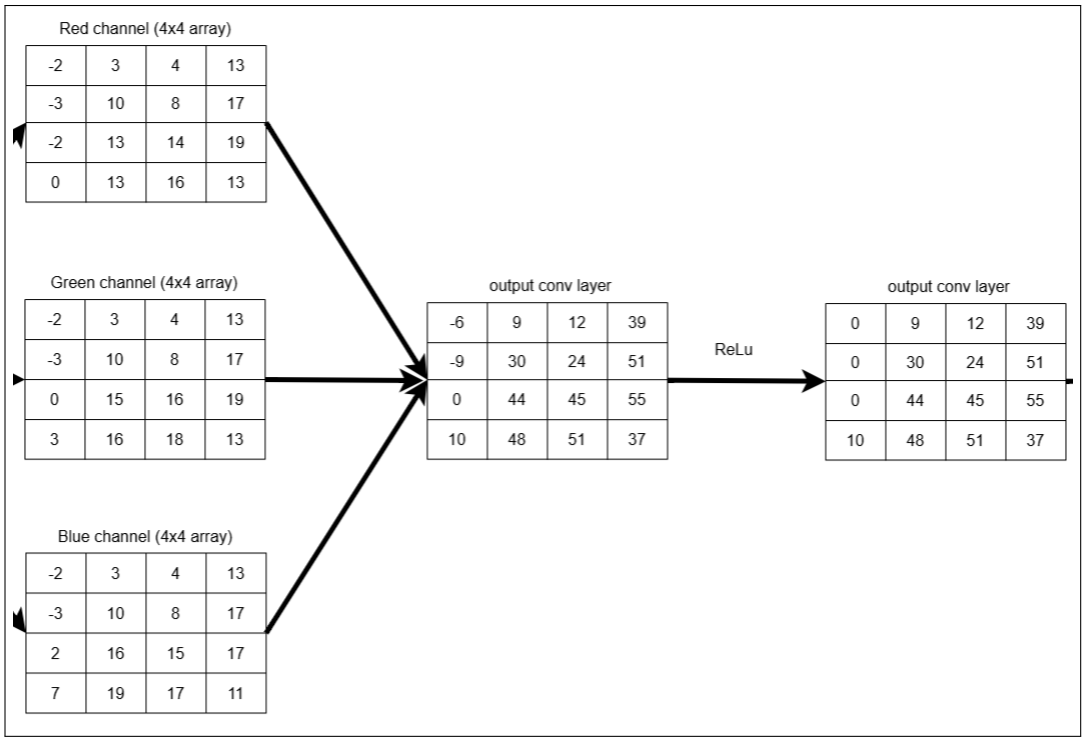
$(-1\times88)+(0\times88)+(1\times88)+(-1\times87)+(0\times86)+(87\times-1)+(-1\times86)+(0\times87)+(87\times-1) = 13$

(-1×87)+(0×89)+(86×-1)+(-1×84)+(0×87)+(87×-1)+(-1×84)+(0×87)+(85×-1)= -3
(-1×89)+(0×86)+(87×-1)+(-1×87)+(0×87)+(85×-1)+(-1×87)+(0×85)+(82×-1)= 10
(-1×86)+(0×87)+(87×-1)+(-1×87)+(0×85)+(82×-1)+(-1×87)+(0×82)+(82×-1)= 8
(-1×87)+(0×85)+(82×-1)+(-1×84)+(0×82)+(81×-1)+(-1×84)+(0×82)+(82×-1)= 17

(-1×84)+(0×87)+(87×-1)+(-1×87)+(0×8785×-1)+(-1×84)+(0×85)+(82×-1)= -2
(-1×87)+(0×87)+(85×-1)+(-1×84)+(0×8582×-1)+(-1×84)+(0×82)+(81×-1)= 13
(-1×87)+(0×85)+(82×-1)+(-1×84)+(0×8281×-1)+(-1×84)+(0×81)+(82×-1)= 14
(-1×84)+(0×82)+(81×-1)+(-1×84)+(0×8182×-1)+(-1×84)+(0×82)+(81×-1)= 19

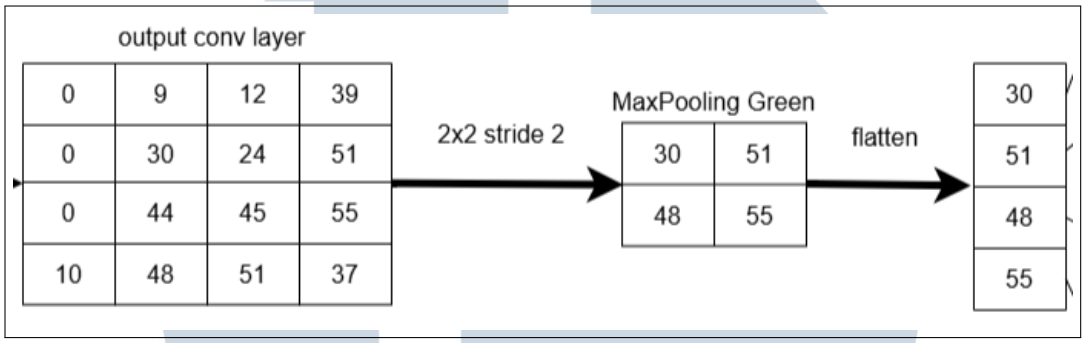
(-1×84)+(0×87)+(85×-1)+(-1×87)+(0×85)+(82×-1)+(-1×87)+(0×82)+(81×-1)=0
(-1×87)+(0×85)+(82×-1)+(-1×84)+(0×82)+(81×-1)+(-1×84)+(0×81)+(82×-1)= 13
(-1×87)+(0×82)+(81×-1)+(-1×84)+(0×81)+(82×-1)+(-1×84)+(0×82)+(81×-1)= 16
(-1×84)+(0×81)+(82×-1)+(-1×84)+(0×82)+(81×-1)+(-1×84)+(0×82)+(81×-1)= 13

Perkalian dilakukan sehingga menghasilkan matriks *Red* baru sama halnya dengan *Green* dan *Blue color* pada gambar. Hasil dari konvolusi matriks dijumlahkan menjadi satu matriks lalu masuk ke *activation*. Berikut setelah melalui tahap konvolusi selanjutnya masuk ke tahap *activation*.



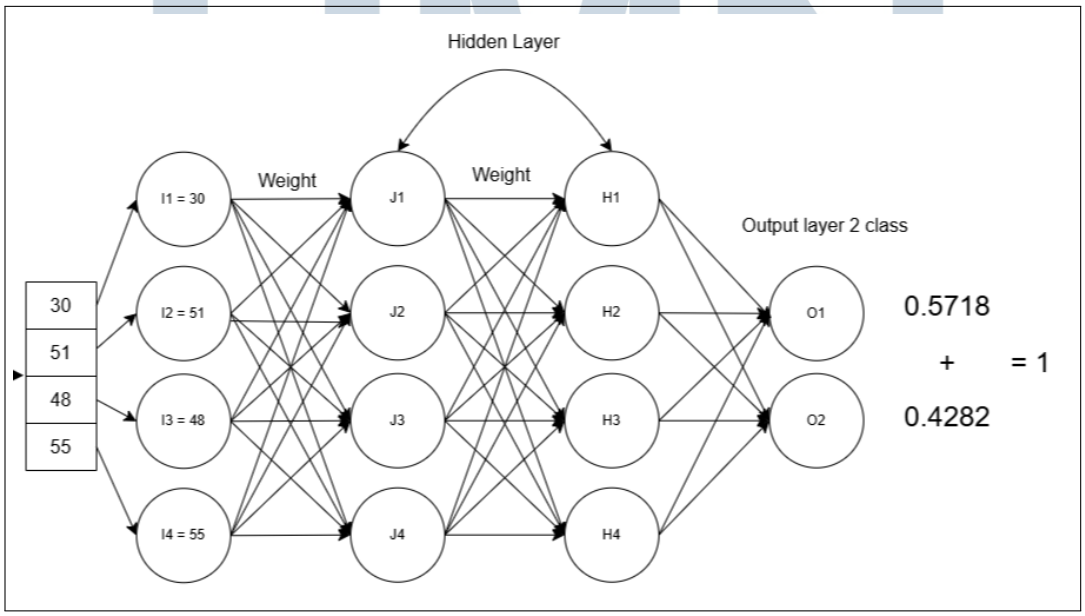
Gambar 3.8. Proses *activation relu*

Pada Gambar 3.8 setelah seluruh matriks gambar dijumlahkan dan dijadikan satu, lalu masuk ke fungsi *activation*. Pada *activation* yang dipakai pada penelitian adalah *ReLU* yaitu mengubah nilai negatif (-) menjadi 0 dan menjadi *output convolutional layer*. Selanjutnya masuk ke tahap *pooling*.



Gambar 3.9. Proses *pooling* dan *flatten*

Pada Gambar 3.9 hasil *output convolutional layer* digunakan untuk proses lapisan berikutnya yaitu *pooling*. Pada *pooling layer* ini mencari nilai *pooling* tergantung dari jenis *pooling* yang digunakan. Umumnya terdapat tiga jenis *pooling* yaitu *minimum pooling*, *average pooling* dan *maximum pooling* Seperti contoh pada Gambar 3.9 menggunakan *MaxPooling* dengan ukuran 2x2 dan *stride* dua untuk mengambil nilai maksimum dari hasil *ouput* matriks sebelumnya. Setelah itu masuk ke tahap *flatten layer* di mana mengubah matriks 2x2 menjadi matriks satu dimensi. Hasil dari *flatten* akan digunakan pada lapisan berikutnya yaitu *fully connected layer*.



Gambar 3.10. Proses *fully connected layer*

Pada Gambar 3.10 merupakan contoh gambaran proses pada *Fully Connected Layer* yang di mana pada contoh memiliki dua *dense layer* dengan empat *neuron*, lalu diakhiri dengan satu *output layer* dengan *activation softmax*. Berikut perhitungan matematis pada *fully connected layer*.

Pada *hidden dense Layer* Pertama.

J1 = (30 × 0.4) + (51 × 0.4) + (48 × 0.4) + (55 × 0.4) = 289.6
J2 = (30 × 0.1) + (51 × 0.1) + (48 × 0.1) + (55 × 0.1) = 18.4
J3 = (30 × 0.6) + (51 × 0.6) + (48 × 0.6) + (55 × 0.6) = 349.2
J4 = (30 × 0.3) + (51 × 0.3) + (48 × 0.3) + (55 × 0.3) = 55.2

Perhitungan matematis pada *hidden dense Layer* Kedua.

H1 = (289.6 × 0.5) + (18.4 × 0.5) + (349.2 × 0.5) + (55.2 × 0.5) = 356.2
H2 = (289.6 × 0.7) + (18.4 × 0.7) + (349.2 × 0.7) + (55.2 × 0.7) = 498.68
H3 = (289.6 × 0.2) + (18.4 × 0.2) + (349.2 × 0.2) + (55.2 × 0.2) = 142.48
H4 = (289.6 × 0.4) + (18.4 × 0.4) + (349.2 × 0.4) + (55.2 × 0.4) = 284.96

Perhitungan matematis pada *output layer* dengan dua kelas.

O1 = (356.2 × 0.8) + (498.68 × 0.8) + (142.48 × 0.8) + (284.96 × 0.8) = 1026.648
O2 = (356.2 × 0.6) + (498.68 × 0.6) + (142.48 × 0.6) + (284.96 × 0.6) = 769.392

Perhitungan matematis pada *activation softmax* setelah *output layer*

$$\sigma(z_A) = \frac{e^{z_A}}{e^{z_A} + e^{z_B}} = \frac{1026.648}{1796.04} = 0.5718$$

$$\sigma(z_B) = \frac{e^{z_B}}{e^{z_A} + e^{z_B}} = \frac{769.392}{1796.04} = 0.4282$$

Pada hasil dari perhitungan *softmax* untuk dua kelas dengan nilai eksponensial *logits* yang diberikan probabilitas kelas A adalah 0.5718 atau 57% dan untuk probabilitas kelas B adalah 0.4282 atau 42%. Pada perhitungan yang dilakukan model akan lebih cenderung memprediksi gambar tersebut sebagai daging sapi *fresh* dan pada label data di mana A merupakan label untuk daging sapi *fresh* dan B adalah label untuk *thawed* (daging sapi beku yang telah dicairkan).

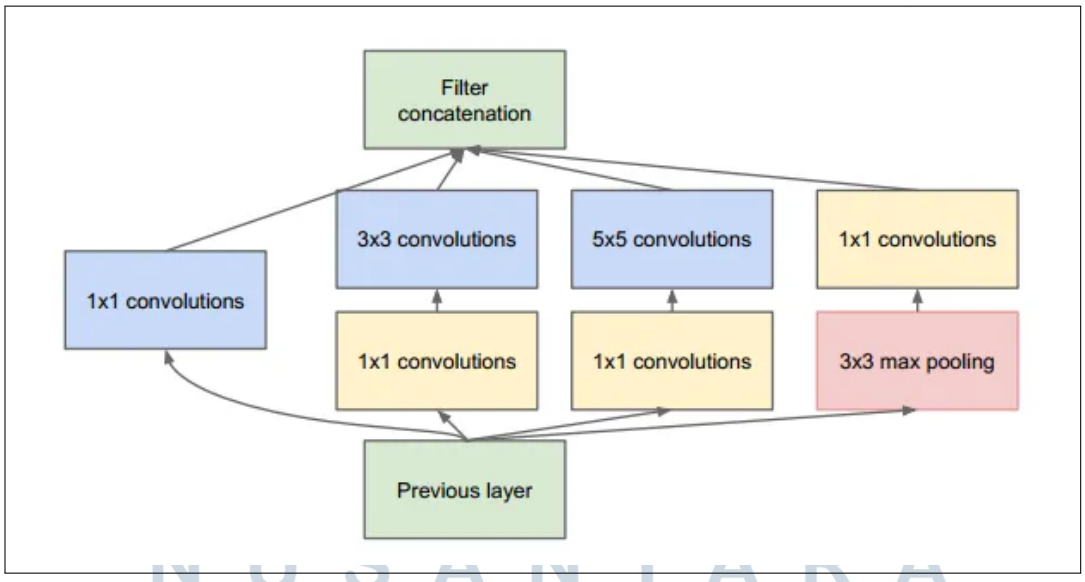
Pada penelitian juga menggunakan *pretrained* atau *transfer learning* model *InceptionV3* dalam proses *training* data karena jumlah *dataset thawed* lebih sedikit dan tidak sebanding dengan *dataset fresh* sehingga membutuhkan model yang sudah dilakukan

pelatihan pada *dataset* yang lebih banyak seperti menggunakan *dataset ImageNet* agar pengetahuan pada model sebelumnya dapat di transfer *dataset* yang baru. Kebutuhan arsitektur yang lebih efisien dan lebih baik dalam menangkap fitur gambar juga menjadi salah satu alasan agar model dapat mengidentifikasi daging dengan baik, sehingga menggunakan *pretrained* model adalah pilihan yang tepat untuk menutupi kekurangan tersebut. Salah satu model yang memiliki biaya komputasi yang rendah dan dapat melakukan deteksi fitur yang lebih baik salah satunya menggunakan *GoogleNet* atau *Inception* model [13], [43].

3.7.1 Inception Model

Model *Inception* adalah jenis arsitektur jaringan saraf *convolutional* (CNN) yang dikembangkan oleh *Google*. Model ini pertama kali diperkenalkan dalam makalah *Going Deeper with Convolutions* pada tahun 2014. Model *Inception* dikenal juga dengan nama *GoogLe Net* atau (InceptionV1) dan telah mengalami beberapa pembaruan, termasuk *InceptionV2*, *InceptionV3*, dan yang terbaru adalah *InceptionV4*. Pada api *tensorflow* yang digunakan model yang ada pada *tensorflow* baru *InceptionV3*, belum ada *update* mengenai model *InceptionV4*.

Arsitektur *Inception* memperkenalkan berbagai blok *inception*, yang berisi beberapa lapisan *convolutional* dan *pooling* yang ditumpuk bersama, untuk memberikan hasil yang lebih baik dan mengurangi biaya komputasi.



Gambar 3.11. *Inception* blok awal

Sumber: [43]

Pada Gambar 3.11 blok awal *inception* memiliki konvolusi 1x1 dilanjutkan

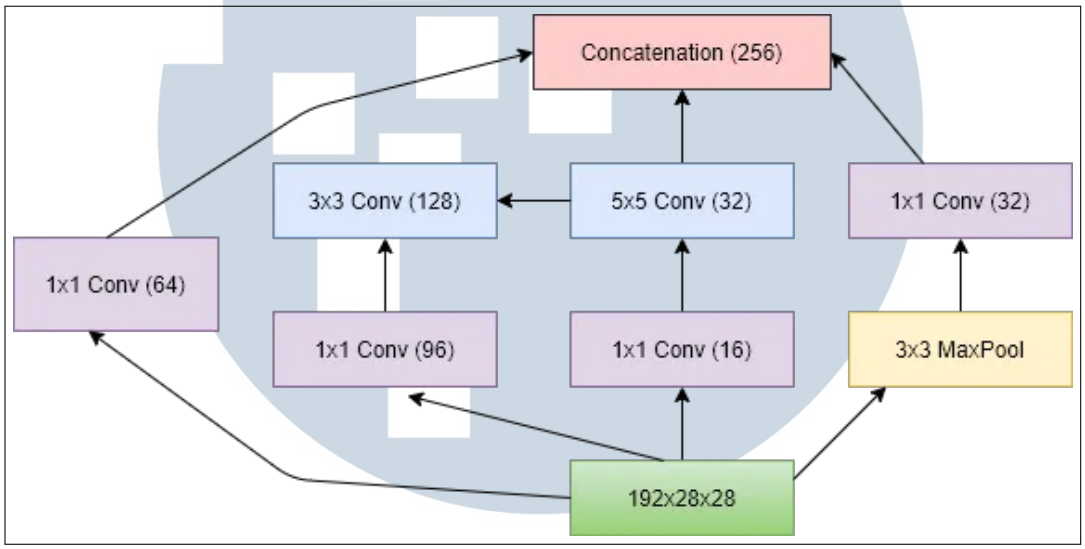
konvolusi 3x3, memiliki konvolusi 1x1 dilanjutkan konvolusi 5x5, memiliki lapisan kolom maksimal 3x3 dilanjutkan konvolusi 1x1 dan memiliki konvolusi 1x1 tunggal. Untuk menjaga dimensi gambar tetap, konvolusi 3x3 memiliki *padding* 1, dan lapisan 5x5 memiliki *padding* 2, sehingga kedua gambar *input* dan *output* memiliki ukuran yang sama.

Semua konvolusi, termasuk yang ada di dalam modul *Inception*, menggunakan *activation linier* yang diperbaiki [43]. Ukuran bidang reseptif dalam jaringan adalah 224 x 224 dalam ruang warna RGB dengan *mean* nol [43]. Jaringan ini dirancang dengan mempertimbangkan efisiensi dan kepraktisan komputasi, sehingga inferensi dapat dijalankan pada perangkat yang berbeda, termasuk perangkat dengan sumber daya komputasi yang terbatas, terutama yang memiliki jejak memori yang rendah [43]. Berikut arsitektur dari *inception model*.

Tabel 3.2. Arsitektur model *inception*

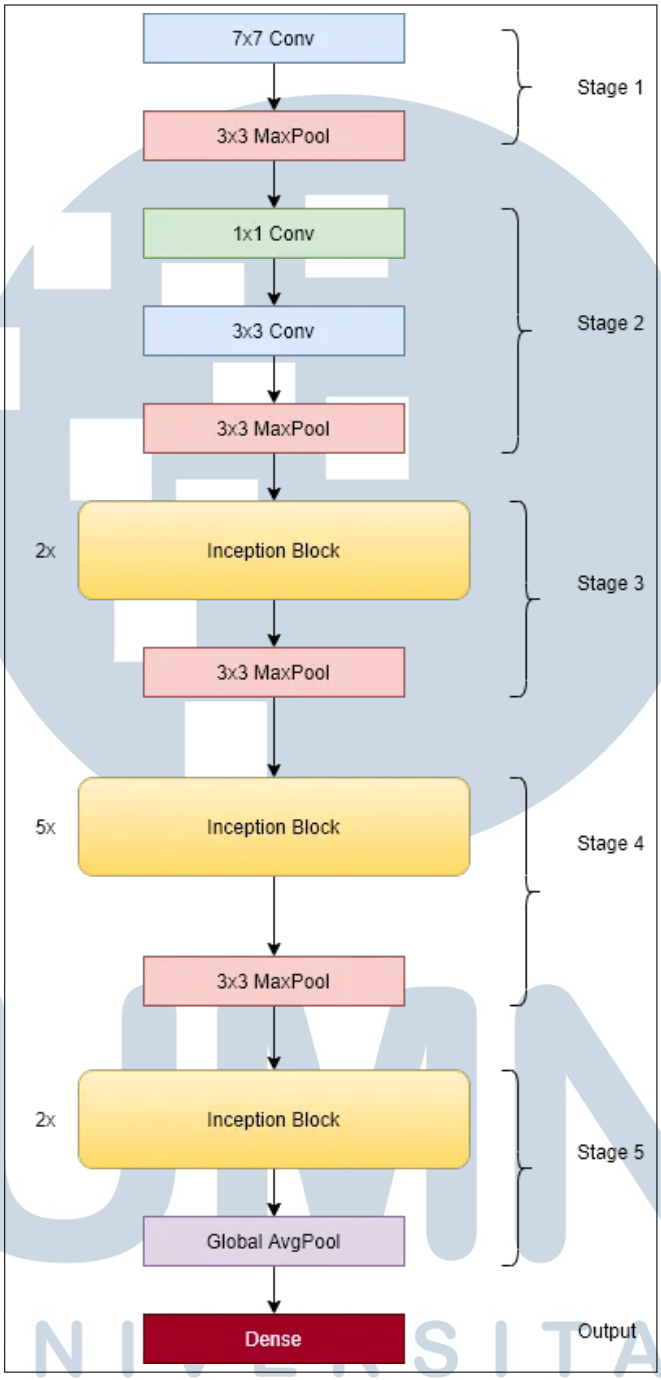
Layer Type	Filter Size	Units	Output Shape
Input	-	-	224 x 224 x 3
Conv2D	7x7 / stride 2	64	112 x 112 x 64
MaxPooling2D	3x3 / stride 2	-	56 x 56 x 64
Conv2D	1x1 / stride 1	64	56 x 56 x 64
Conv2D	3x3 / stride 1	192	56 x 56 x 192
MaxPooling2D	3x3 / stride 2	-	28 x 28 x 192
Inception Module 1a	1a (64, 96, 128, 16, 32, 32)	-	28 x 28 x 256
Inception Module 1b	1b (128, 128, 192, 32, 96, 64)	-	28 x 28 x 480
MaxPooling2D	3x3 / stride 2	-	14 x 14 x 480
Inception Module 2a	2a (192, 96, 208, 16, 48, 64)	-	14 x 14 x 512
Inception Module 2b	2b (160, 112, 224, 24, 64, 64)	-	14 x 14 x 512
Inception Module 2c	2c (128, 128, 256, 24, 64, 64)	-	14 x 14 x 512
Inception Module 2d	2d (112, 144, 288, 32, 64, 64)	-	14 x 14 x 528
Inception Module 2e	2e (256, 160, 320, 32, 128, 128)	-	14 x 14 x 832
MaxPooling2D	3x3 / stride 2	-	7 x 7 x 832
Inception Module 3a	3a (256, 160, 320, 32, 128, 128)	-	7 x 7 x 832
Inception Module 3b	3b (384, 192, 384, 48, 128, 128)	-	7 x 7 x 1024
AveragePooling2D	7x7	-	1 x 1 x 1024
Dropout	-	-	1 x 1 x 1024
Dense	-	1000	1 x 1 x 1000
Softmax	-	-	1 x 1 x 1000

Pada Model 3.2 *Input* total terdapat 21 lapisan, gambar maksimum 299x299x3, Conv2D perkalian *array* dengan berbagai filter untuk mengambil fitur gambar. *MaxPooling2D* adalah penggabungan maksimum untuk mengurangi dimensi. *AveragePooling*2D menggunakan *pooling* rata-rata untuk mengurangi dimensi sebelum lapisan akhir. *dropout* untuk menghindari *overfitting* dengan mematikan *neuron* yang tidak digunakan. Lapisan *dense* terdiri dari unit sebanyak jumlah kelas yang dapat diklasifikasikan. Fungsi *activation Softmax* untuk menghasilkan probabilitas kelas.



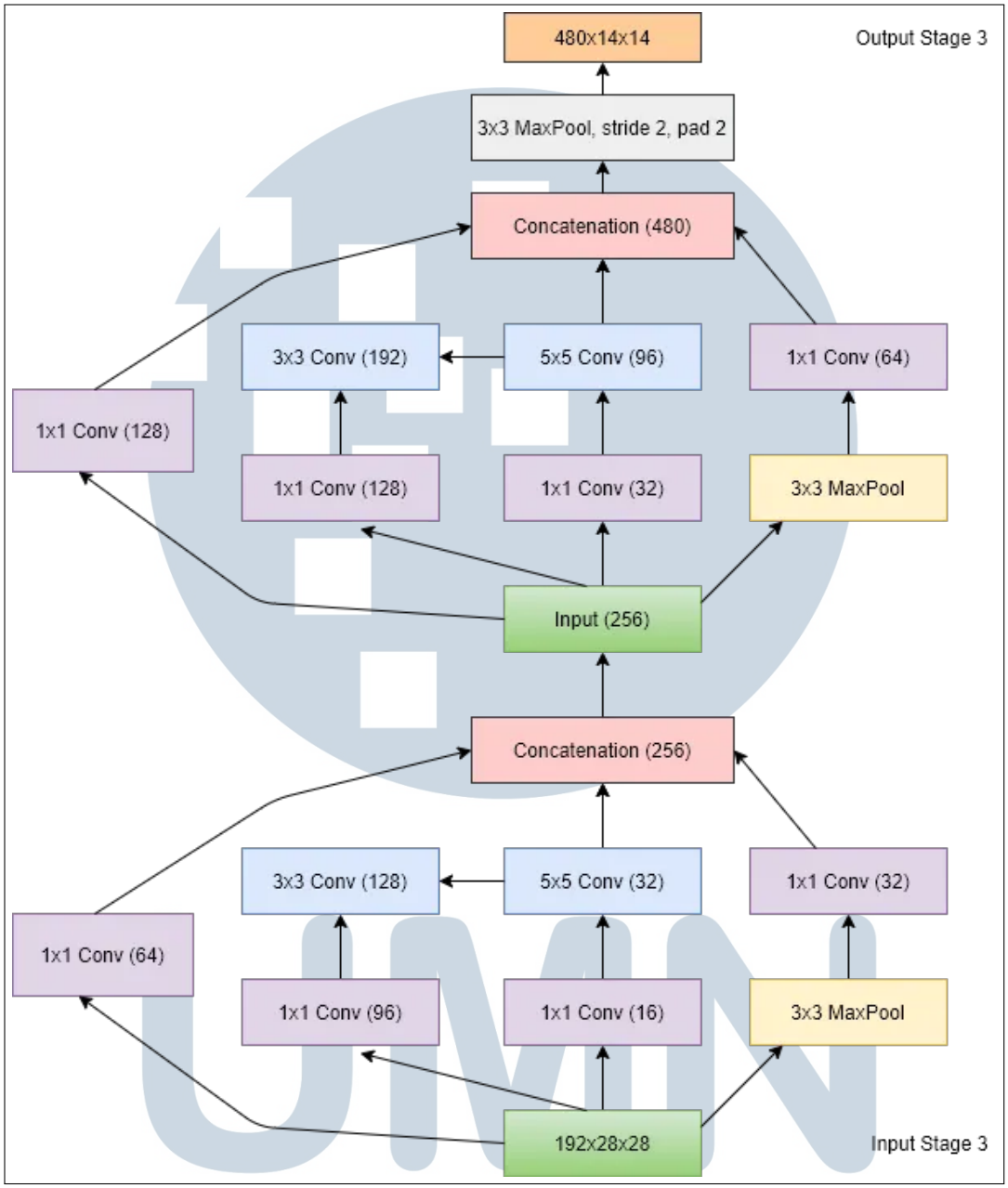
Gambar 3.12. *Inception modul* 1a

Pada Gambar 3.12 modul *Inception* adalah kombinasi dari beberapa filter (1x1, 3x3, 5x5) dan *pooling* yang bekerja secara paralel untuk mengumpulkan fitur dengan ukuran yang berbeda contohnya pada *Inception Module 1a* (64, 96, 128, 16, 32, 32) menunjukkan penggunaan 64 filter untuk konvolusi 1x1, 96 dan 128 filter untuk konvolusi 3x3, 16 dan 32 filter untuk konvolusi 5x5, dan 32 filter untuk *pooling*. Berikut arsitektur *Inception* yang disederhanakan.



Gambar 3.13. *Inception model* yang disederhanakan.

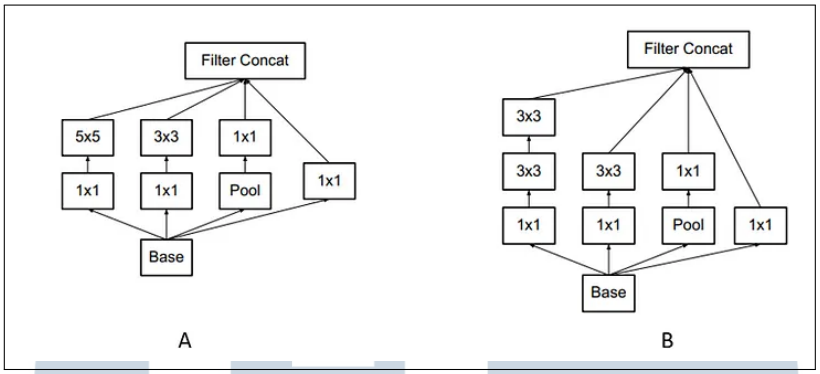
Pada Gambar 3.13 *Stage* satu jaringan dimulai dengan ukuran gambar 224x224x3 kemudian melewati 3x3 *MaxPooling*. *Stage* dua Kemudian melewati 1x1 Conv2D, 3x3 Conv2D , 3x3 *MaxPooling*, dan menghasilkan gambar ukuran 192x28x28. Selanjutnya masuk ke *Stage* dua di mana berisi *Inception Block*.



Gambar 3.14. *Stage* tiga *inception*

Pada Gambar 3.14 tahap tiga memiliki dua blok *Inception* dan pada akhirnya lapisan *Max Pool*. Tetapi blok awal tidak memiliki alokasi saluran yang sama, seperti yang terlihat pada gambar. Blok satu memiliki 256 saluran, sedangkan blok dua memiliki 480 saluran. Jadi gambar *input* 192x28x28 sekarang menjadi 480x14x14, setelah dua blok awal dan lapisan *MaxPooling*. *Stage* empat dan lima sangat mirip dengan *stage* tiga. *Stage* empat memiliki lima blok awal dilanjutkan *MaxPool*, dan tahap lima memiliki dua blok awal dilanjutkan *GlobalAveragePool*. Pada *InceptionV2* ditambahkan normalisasi *batch*,

InceptionV3 pada blok awal terdapat modifikasi seperti pada Gambar 3.15.



Gambar 3.15. A blok *inception* asli, B blok awal di *InceptionV3*

Pada Gambar 3.15 gambar A merupakan *InceptionV1* awal lalu terdapat perubahan dengan mengubah 5x5 dengan beberapa konvolusi 3x3. Selain itu juga terdapat pergantian 5x5 dengan konvolusi 1x7 dan 7x1 dan pergantian 3x3 dengan konvolusi 1x3 dan 3x1.

Arsitektur *Inception* memperkenalkan berbagai blok *inception*, yang berisi beberapa lapisan *convolutional* dan *pooling* yang ditumpuk bersama, untuk memberikan hasil yang lebih baik dan mengurangi biaya komputasi. Jaringan *Inception* telah meningkat perlahan dengan versi yang lebih baru dan lebih baru, dan telah mengungguli arsitektur lain seperti *VGG* dan *AlexNet* dalam hal akurasi [43].

3.8 Uji coba dan evaluasi

Pada tahap ini, data uji akan digunakan untuk menguji model yang telah dilatih. *confusion matrix* digunakan untuk mengevaluasi hasil identifikasi daging dari model yang telah dibuat sebelumnya. Proses evaluasi dilakukan dengan menghitung nilai kinerja dari *confusion matrix*, seperti *accuracy, precision, recall*, dan f1 *score*. Setelah itu menampilkan gambar dan label hasil uji coba.

3.9 Penulisan Laporan

Laporan ditulis dari awal hingga akhir penelitian, serta implementasi algoritma dan hasil penelitian.