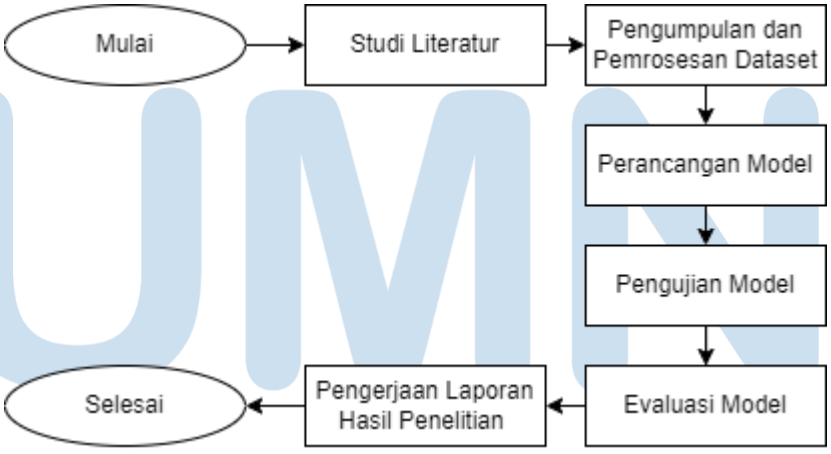
BAB III
METODOLOGI PENELITIAN

3.1 Metode Penelitian

Pada penelitian ini terdapat beberapa tahap metode penelitian yang dilakukan antara lain, penelitian dimulai dengan melakukan studi literatur terhadap beberapa penelitian terdahulu yang berkaitan dengan klasifikasi nematoda dan teknik *transfer learning*. Kemudian penelitian dilanjutkan dengan pemrosesan dataset nematoda yang telah terkumpul di tahap *pre-processing.* Setelah itu, dilakukan perancangan model klasifikasi, di mana di tahap ini penulis memilih *pre-trained* model yang cocok untuk digunakan dalam pengklasifikasian nematoda, kemudian penulis melakukan uji coba pada masing-masing *pre-trained* model tersebut. Setelah mendapatkan hasil dari uji coba yang dilakukan, penulis mengevaluasi performa setiap model yang sudah selesai dibuat dan membandingkannya satu sama lain. Hasil evaluasi setiap model kemudian ditulis ke penyusunan laporan Tugas Akhir.



Gambar 3. 1 Alur langkah penelitian

3.2 Studi Literatur

Dalam melakukan studi literatur, selain tiga sumber rujukan utama dan beberapa jurnal penelitian lainnya, penulis juga menggunakan *platform* Medium dan Towards Data Science (TDS) untuk membantu penulis melakukan *research* tambahan seputar topik penelitian. Medium dan TDS adalah dua *platform* yang berhubungan dengan publikasi dan berbagi konten terkait teknologi, ilmu data, dan topik relevan lainnya. TDS adalah salah satu publikasi di Medium yang berfokus pada topik terkait ilmu data, kecerdasan buatan, pembelajaran mesin, analisis data, dan rumpun keilmuan relevan lainnya. Selain itu penulis juga melakukan diskusi seputar topik penelitian dengan Dosen Pembimbing Teknik Komputer penulis.

3.3 Pengumpulan dan Pemrosesan Data

3.3.1 Dataset

Dataset diperoleh dari Bagian Nematologi Fakultas Pertanian Universitas Gajah Mada (UGM). Dataset ini merupakan dataset citra nematoda non parasitik yang umum ditemukan di Indonesia. Genus yang didapatkan antaralain *Genus Acrobeles*, *Genus Acrobeloides*, *Genus Rhabditis*, *Genus Aphelenchus*, *Genus Tylenchus*, *Genus Dorylaimida*, dan *Genus Mononchida*. Dataset tersebut dikelompokkan menjadi tiga kelas yaitu *bacterivore*, *fungivore*, *predator-omnivore*.

Pada penelitian ini, karena dataset yang digunakan tidak seimbang (*imbalanced*), maka penulis menggunakan dua dataset yaitu dataset awal dan dataset yang sudah di *oversampling* untuk melihat apakah berpengaruh pada performa model. *Oversampling* adalah salah satu teknik dalam pengolahan dataset yang tidak seimbang yang bertujuan untuk meningkatkan jumlah data pada kelas minoritas agar sebanding dengan jumlah data pada kelas mayoritas. Penulis memilih untuk melakukan *oversampling* karena mempertimbangkan pada saat pelatihan model, semakin banyak dataset yang

digunakan, model akan dapat mempelajari pola-pola yang relevan dan dapat melakukan prediksi dengan akurasi yang lebih baik. Dataset yang digunakan dibagi dengan perbandingan 80:10;10, alasan pembagian tersebut didasari dari teknik umum yang memang sudah sering dipakai pada pembagian dataset dan penulis memberikan 80% dataset untuk data pelatihan agar model dapat mempelajari pola sampel lebih banyak. Penyebaran data masing-masing citra ditunjukkan pada Tabel 3.1, 3.2, dan 3.3.

Tabel 3. 1 Dataset awal (*non oversampling*) Nematoda Non Parasitik

Kelas Nematoda	Jumlah Data
Bacterivore	287
Fungivore	80
Predator-Omnivore	554
Total: 928	

Tabel 3. 2 Pembagian dataset *non oversampling*

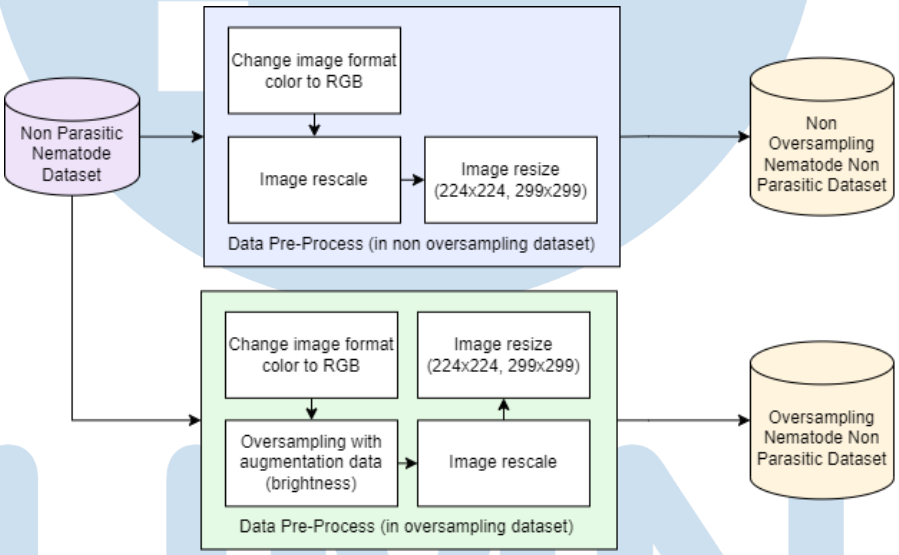
Kelas Nematoda	Pembagian Dataset		
	Train	Val	Test
Bacterivore	229	28	30
Fungivore	64	8	8
Predator-Omnivore	443	55	56
Total	736	91	94

Tabel 3. 3 Pembagian dataset *oversampling*

Kelas Nematoda	Pembagian Dataset		
	Train	Val	Test
Bacterivore	443	28	30
Fungivore	443	8	8
Predator-Omnivore	443	55	56
Total	1329	91	94

3.3.2 Pre-Processing

Citra mikroskopik yang diperoleh memiliki ukuran asli sebesar 2560x2048 *pixel* dan 4100x3075 *pixel*. Karena format warna dataset asli berbeda- beda, penulis juga mengubah seluruh citra menjadi RGB. Kemudian citra nematoda dilakukan pengskalaan piksel untuk mengurangi perhitungan besar dan memungkinkan untuk konvergensi yang lebih cepat. Untuk menyesuaikan *input shape* dari setiap model, dilakukan *resize* ke ukuran 224x224 dan 299x299 (sesuai *input size* model). Alur *pre-processing* ditunjukkan pada Gambar 3.2.

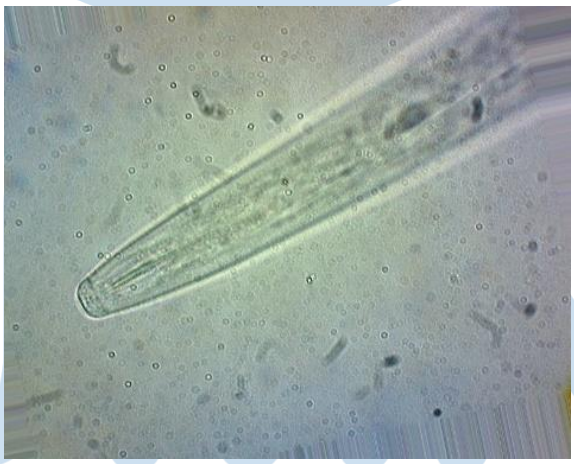


Gambar 3. 2 Diagram alur *pre-processing* data

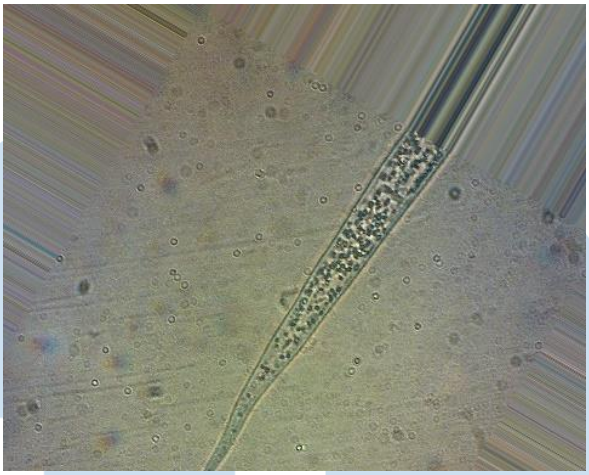
3.3.3 Oversampling

Mengingat jumlah citra dataset yang diperoleh penulis tidak seimbang di setiap kelasnya, maka penulis melakukan *oversampling* untuk menyeimbangkan data ke kelas dataset yang memiliki data terbanyak. *Oversampling* ini dilakukan dengan memperbanyak citra sebanyak dua kali melalui proses augmentasi *brightness*. Augmentasi *brightness* dipilih karena tingkat pencahayaan citra pada dataset yang digunakan berbeda-beda, sehingga augmentasi *brightness* dipilih untuk menyeimbangkan tingkat

pencahayaan antara satu citra dengan yang lainnya. Rentang perubahan pencahayaan sebesar 0.2 hingga 0.4 sampai mencapai jumlah yang sesuai dengan kelas terbanyak yaitu 443 citra. Rentang tersebut merupakan seberapa besar citra akan ditingkatkan kecerahannya secara *random*. Rentang tersebut dipilih untuk menjaga agar citra nematoda non parasit yang sudah memiliki pencahayaan tinggi tidak teraugmentasi menjadi terlalu terang. Augmentasi *brightness* juga dipilih karena *brightness* menjadi augmentasi yang tidak merubah morfologi, karena dengan augmentasi seperti *flip* atau *rotate* dikhawatirkan akan merubah orientasi citra dan morfologi nematoda non parasitnya yang nantinya akan menghasilkan citra nematoda menjadi tertarik "*stretched*". Contoh citra hasil augmentasi *rotate* dan *flip* ditunjukkan pada Gambar 3.3 dan 3.4. Selain itu augmentasi *brightness* dipilih karena pada penelitian [5] menghasilkan performa yang baik untuk meningkatkan akurasi model.



Gambar 3. 3 Hasil augmentasi *rotate* 20º



Gambar 3. 4 Hasil augmentasi *rotate* 180º

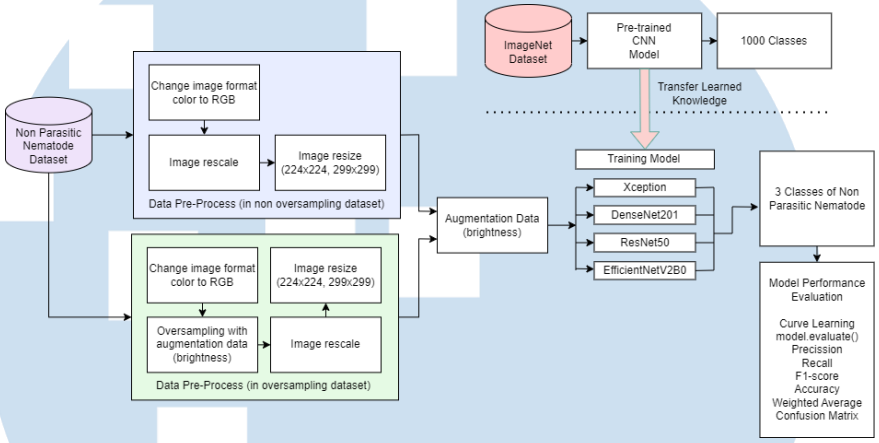
3.3.4 Augmentasi dan Pengskalaan Piksel

Pada proses pelatihan model dilakukan augmentasi sebesar 0.4 hingga 0.8 dengan menggunakan ImageDataGenerator, rentang tersebut merupakan seberapa besar kecerahan pada citra akan ditingkatkan secara random dan melakukan pengskalaan piksel untuk mengurangi perhitungan besar dan memungkinkan untuk konvergensi yang lebih cepat. Pengskalaan gambar yaitu 1/255. Dengan melakukan penskalaan 1/255, setiap nilai piksel dalam citra akan dibagi oleh 255. Dengan demikian, rentang nilai piksel awal dari 0 hingga 255 akan diubah menjadi rentang baru dari 0 hingga 1. Dalam hal ini, mengubah rentang skala piksel sehingga dapat memperoleh representasi piksel yang sesuai dengan rentang nilai yang diharapkan oleh model atau algoritma, yang harapannya dapat memudahkan proses pengolahan atau pelatihan lebih lanjut.

3.4 Pembuatan dan Pelatihan Model

Pengklasifikasian CNN terdiri dari dua bagian: (i) bagian ekstraksi fitur (*feature extractor*) yang mencakup beberapa *convolutional layer* diikuti oleh *pooling layer* dan *activation function* yaitu ReLU; dan (ii) bagian klasifikasi (*classifier*) yang merupakan beberapa *fully connected layer*

dilengkapi dengan *loss function* yaitu *softmax*. Pembuatan model pelatihan menggunakan *pre-trained* model yang sebelumnya sudah dilatih dan tersedia dari Keras.



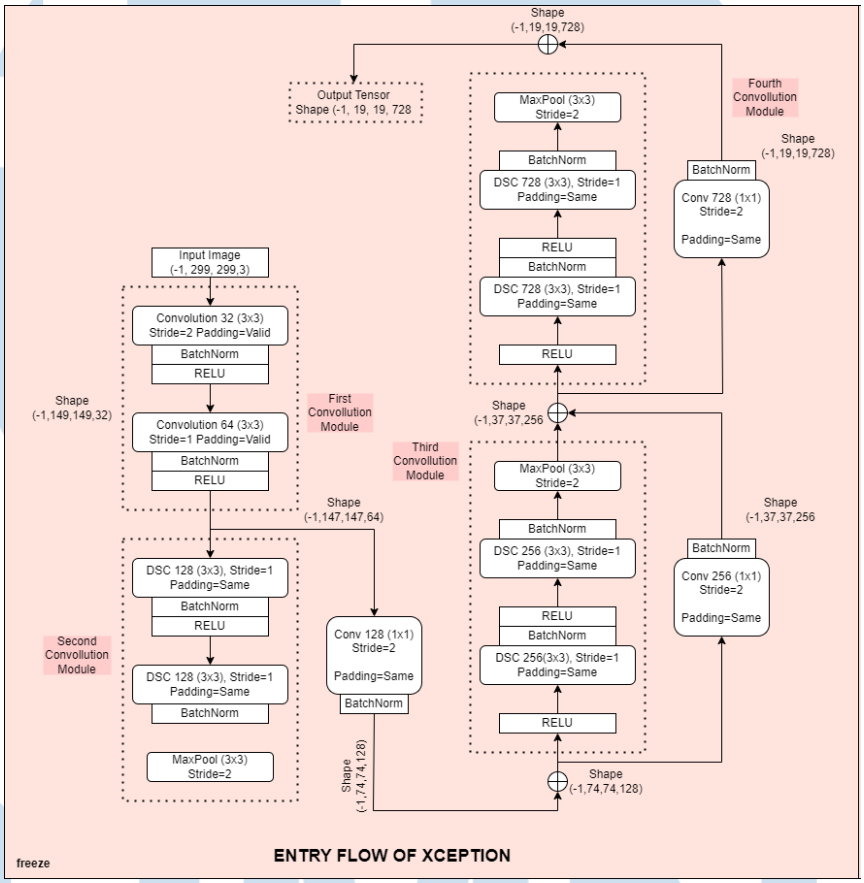
Gambar 3. 5 Diagram alur proses pembuatan model

Seperti yang ditunjukkan oleh diagram alur pada Gambar 3.6, proses *transfer knowledge* dari dataset ImageNet ke model dataset nematoda non parasitik adalah dengan menggunakan *transfer learning* yaitu dengan melakukan *freeze layer* pada *feature extractor*, sehingga pada saat proses *training* bagian *feature extractor* tidak perlu diubah dan di-*training* ulang. Kemudian mengganti jumlah kelas menjadi tiga kelas sesuai dengan kelas pada dataset. Lalu pada *classifier* ditambahkan beberapa *layer* baru yang nantinya akan di-*training*. Pada penelitian ini penulis menambahkan *layer Flatten, Dense,* dan *Dropout*. *Flatten* berperan dalam mengubah hasil *feature map* pada *convolutional layer* menjadi *array* satu dimensi, karena sebelum masuk ke *fully connected layer*, hasil *feature map* masih dalam bentuk *array* multidimensi. *Dense* berperan dalam menghubungkan setiap *neuron* dalam satu *layer* dengan yang ada di *layer* berikutnya. *Dropout* berperan dalam untuk mengurangi *overfitting*, karena dengan menggunakan *Dropout* terkadang model akan melihat fitur-fitur yang belum dilihat dikarenakan *neuron-neuron* yang menjadi fitur utamanya dihilangkan dengan *Dropout* layer. Pemilihan *layer* ini didasarkan pada teori *layer* CNN pada umumnya.

3.4.1 Arsitektur Model

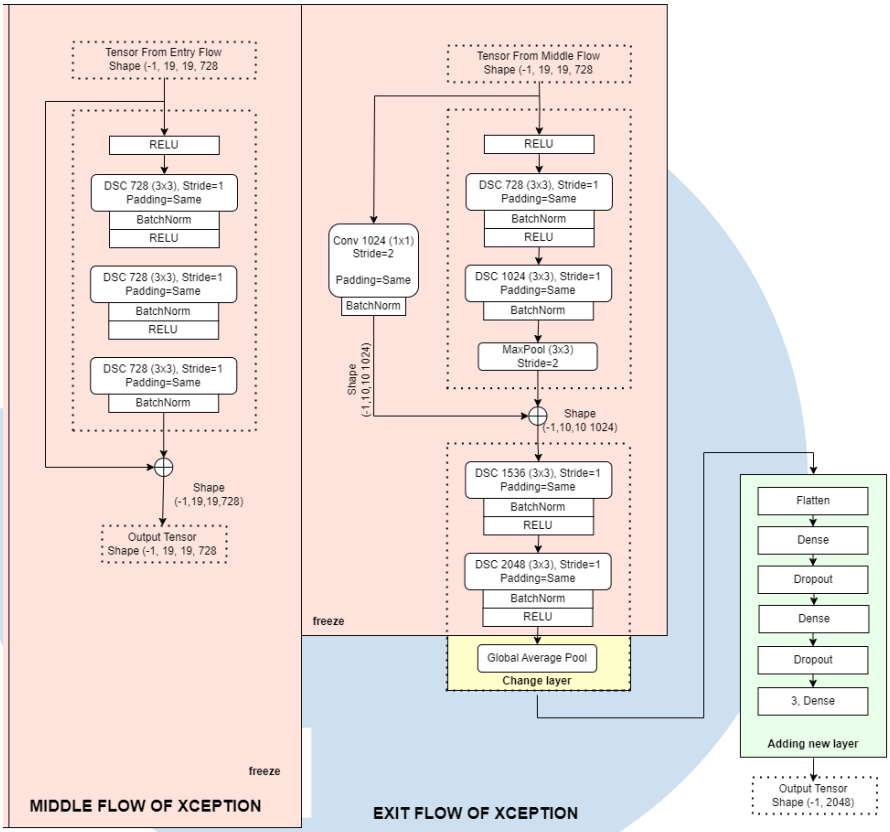
Model *pre-trained* yang digunakan adalah Xception, DenseNet201, ResNet50, dan EfficientNetV2B0.

3.4.1.1 Arsitektur Xception



Gambar 3. 6 *Entry Flow of* Xception

Modul pertama ini berisi lapisan konvolusi konvensional dan tidak memiliki lapisan DSC (*Depthwise Separable Convolution*). Mereka menerima tensor input dengan ukuran (-1, 299, 299, 3). Konvolusi dengan langkah (*stride*) 2 mengurangi ukurannya hampir setengahnya. Bentuk *output* ditunjukkan oleh sisi yang dihitung menggunakan rumus konvolusi. Kecuali modul pertama, semua modul lainnya dalam aliran masuk (*entry flow*) memiliki *residual skip connection*. *Parallel skip connection* memiliki lapisan konvolusi titik demi titik (*pointwise convolution layer*) yang ditambahkan ke output dari jalur utama (*main path*).

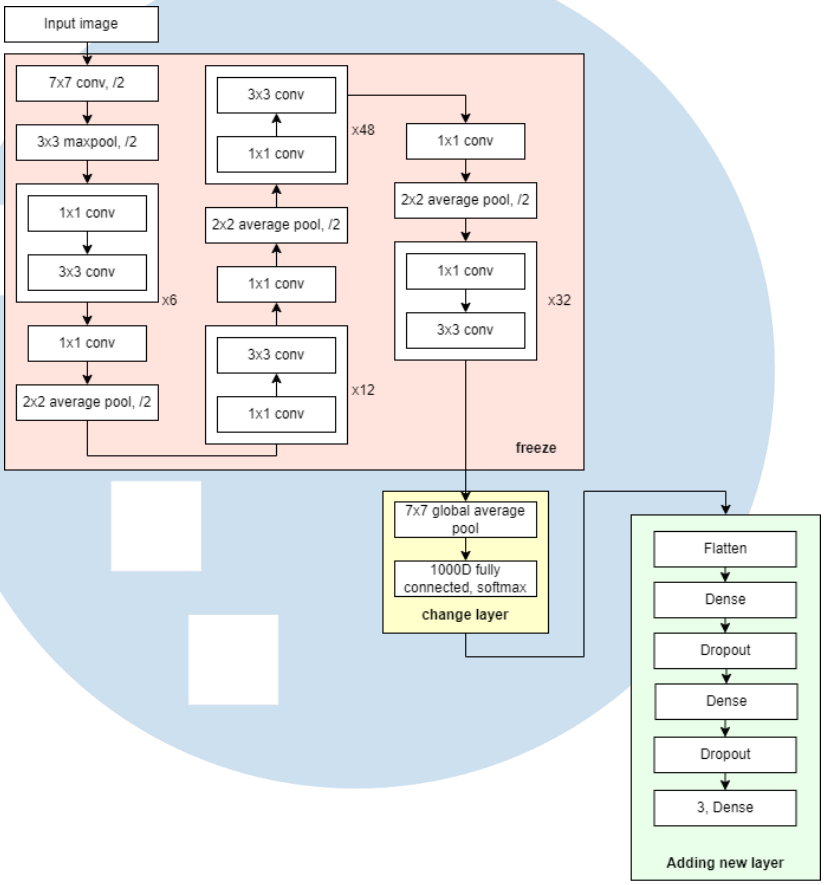


Gambar 3. 7 *Midde and Exit Flow* Xception

Pada aliran tengah (*middle flow*), terdapat delapan modul perulangan. Modul pada *middle flow* diulang delapan kali untuk membentuk aliran tengah. Seluruh delapan modul dalam aliran tengah menggunakan *stride* 1 dan tidak memiliki lapisan penggabungan (*pooling layers*). Oleh karena itu, ukuran spasial dari tensor yang dilewatkan dari aliran masuk tetap sama. Kedalaman saluran (*channel depth*) juga tetap sama karena semua modul dalam aliran tengah memiliki 728 filter. Dan itu sama dengan kedalaman *input*.

Aliran keluar (*exit flow*) hanya memiliki dua modul konvolusi dan modul kedua tidak memiliki *skip connection*. Modul kedua menggunakan *Global Average Pooling*, berbeda dengan modul sebelumnya yang menggunakan *Maxpooling*. Vektor *output* dari lapisan *average pooling* dapat langsung diberikan ke lapisan regresi logistik. Namun, secara opsional juga dapat menggunakan lapisan *Fully Connected* (FC) perantara [14].

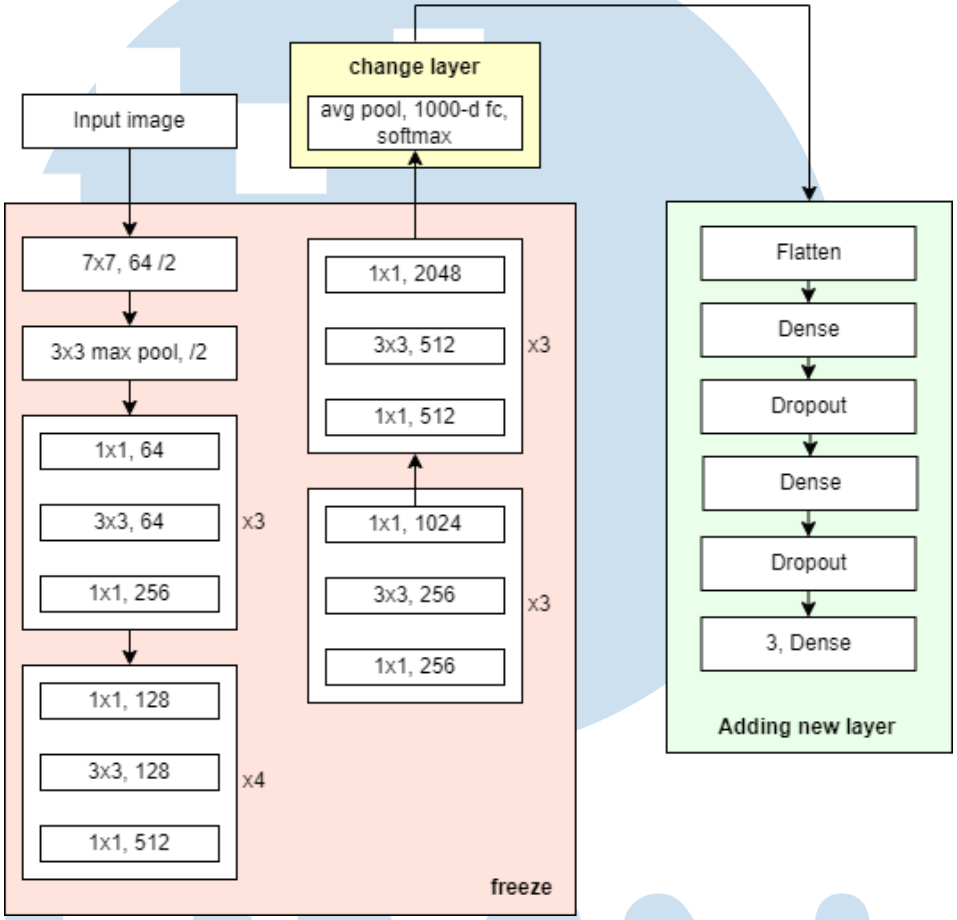
3.4.1.2 Arsitektur DenseNet201



Gambar 3. 8 Arsitektur DenseNet201 beserta *freeze* dan tambahan layer

Untuk lapisan konvolusi dengan ukuran kernel 3x3, setiap sisi *input* akan diberi *padding nol* sebanyak satu piksel agar ukuran peta fitur tetap. DenseNet201 menggunakan konvolusi 1x1 diikuti oleh *average pooling* 2x2 sebagai lapisan transisi antara dua blok *dense* yang saling berdekatan. Pada akhir blok *dense* terakhir, dilakukan *global average pooling* dan kemudian ditambahkan pengklasifikasi *softmax*. Ukuran peta fitur pada tiga blok *dense* adalah 32 [15].

3.4.1.3 Arsitektur ResNet50

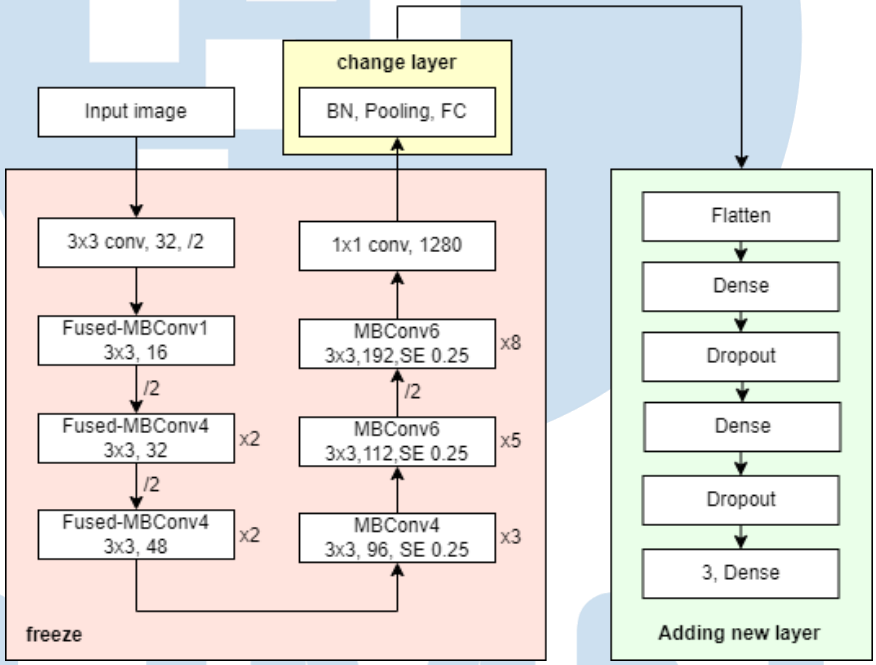


Gambar 3. 9 Arsitektur ResNet50 beserta *freeze* dan tambahan layer

Untuk setiap fungsi residual (F), ResNet menggunakan tumpukan 3 lapisan bukan 2. Ketiga lapisan tersebut adalah konvolusi 1x1, konvolusi 3x3, dan konvolusi 1x1, di mana lapisan 1x1 bertanggung jawab untuk mengurangi dan kemudian meningkatkan (memulihkan) dimensi, sehingga lapisan 3x3 menjadi *bottleneck* dengan dimensi *input*/*output* yang lebih kecil. *Shortcut* identitas tanpa parameter sangat penting untuk arsitektur *bottleneck*. Jika *shortcut* identitas digantikan dengan proyeksi, dapat ditunjukkan bahwa kompleksitas waktu dan ukuran model akan dua kali lipat, karena *shortcut* terhubung ke dua ujung berdimensi tinggi. Oleh

karena itu, *shortcut* identitas menghasilkan model yang lebih efisien untuk desain *bottleneck*. ResNet 50-layer: Kami menggantikan setiap blok 2 lapisan dalam jaringan 34-layer dengan blok *bottleneck* 3 lapisan ini, menghasilkan ResNet 50-layer [18].

3.4.1.4 Arsitektur EfficientNetV2B0



Gambar 3. 10 Arsitektur EfficientNetV2B0 beserta *freeze* dan tambahan layer

Peningkatan model baru yang disebut EfficientNetV2, yang dengan metode pelatihan khusus, dapat mencapai kecepatan konvergensi 5× - 11× lebih cepat dibandingkan dengan model *state-of-the-art* lainnya dengan ukuran hingga 6× lebih kecil. EfficientNetV2 yang diimplementasikan dalam penelitian ini adalah versi B0 dan M. EfficientNetV2B0 memiliki *trade-off* yang lebih baik antara akurasi dan FLOPs, sementara EfficientNetV2M mengurangi parameter dan FLOPs tetapi berjalan lebih cepat dalam pelatihan dan inferensi dibandingkan dengan versi V1-B7 [19].

3.4.2 Hyperparameter

Hyperparameter yang digunakan pada dataset asli dan dataset *oversampling* hampir sama, perbedaannya hanya pada *epoch* dan *batch size*-nya. *Hyperparameter* yang digunakan secara lengkap ditunjukkan pada Tabel 3.4.

Tabel 3. 4 *Hyperparameter* yang digunakan

Hyperparameter	Data Asli	Data Oversampling
Learning rate	0.001	0.0001
Batch size	32	64
Epoch	30	50
Loss	Categorical Crossentropy	Categorical Crossentropy
Optimizer	SGD	SGD

Deep learning belajar dari memperbarui parameter *weight* dan *bias* dengan tujuan untuk mengurangi *loss* pada *training* data. Membutuhkan gradien *loss* terhadap *weight* yang di-*update* dengan *learning rate*. *Learning rate* yang digunakan pada penelitian ini berbeda karena pada awalnya penulis melakukan *training* terlebih dahulu pada dataset asli, setelah melihat hasil *training* model, terdapat ketidakstabilan berdasarkan grafik, sehingga penulis mencoba untuk memperkecil *learning rate* dengan harapan model memiliki peluang lebih besar untuk mencapai konvergensi yang stabil dan memberikan model waktu untuk mengenali dan mengekstraksi fitur-fitur ini secara efektif.

Batch size merupakan berapa banyak data yang diambil dalam satu iterasi. Dalam satu iterasi akan mengambil *batch size* secara *random* untuk memperbarui *weight*, lalu proses tersebut terus dilakukan hingga data habis yang di mana proses tersebut sudah mencapai 1 *epoch*. Penggunaan *batch size* dan *epoch* yang berbeda untuk melihat apakah dapat membantu dan memberikan peningkatan pada proses *training*. *Optimizer* SGD dipilih karena pada *gradient descent* memerlukan semua data latih hanya untuk satu kali *update weight*, sehingga apabila semakin banyak dataset, akan membutuhkan komputasi yag lebih banyak hanya untuk satu kali *update*. Dengan

menggunakan SGD, hanya memerlukan banyak data sesuai *batch size* yang digunakan untuk satu kali *update*.

3.5 Metode Evaluasi

Evaluasi model membantu dalam memahami sejauh mana model mampu melakukan prediksi yang akurat dan relevan terhadap data baru yang belum pernah dilihat sebelumnya pada dataset *test*. Pada penelitian ini metrik evaluasi seperti *accuracy*, *precision*, *recall*, *F1-score* digunakan untuk memberikan gambaran tentang kualitas prediksi model.

- *Accuracy* : Menghitung rasio kelas yang diprediksi benar dengan jumlah total sampel yang dievaluasi.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \text{ [11]}$$

- *Precision* : Menghitung sampel positif yang diprediksi dengan benar oleh semua sampel yang diprediksi dalam kelas positif. Semakin tinggi nilai *precision*, semakin sedikit kesalahan dalam mengklasifikasikan sampel negatif sebagai positif.

$$Precision = \frac{TP}{TP+FP} \text{ [11]}$$

- *Recall* atau *Sensitivity* : Menghitung fraksi sampel positif yang diklasifikasikan dengan benar. *Recall* mengukur seberapa baik model dalam menemukan semua sampel positif. Nilai *recall* dihitung untuk setiap kelas secara terpisah. Semakin tinggi nilai *recall*, semakin banyak sampel positif yang berhasil ditemukan.

$$Recall = \frac{TP}{TP+FN} \text{ [11]}$$

- *F1-score* : Menghitung rata-rata harmonik antara *recall* dan *precision*. Semakin tinggi nilai *F1-score*, semakin baik performa model dalam mengklasifikasikan sampel positif dengan baik dan menemukan semua sampel positif.

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$ [11]

- *Support* : Menunjukkan jumlah contoh yang ada dalam setiap kelas.
- *Macro Average* : Rata-rata dari *precision*, *recall*, dan *F1-score* secara rata-rata untuk semua kelas. Metrik ini memberikan perlakuan yang sama terhadap setiap kelas tanpa mempertimbangkan ketidakseimbangan kelas dalam distribusi datanya.
- *Weighted Average* : Rata-rata ponderasi dari *precision*, *recall*, dan *F1-score* berdasarkan jumlah sampel dalam masing-masing kelas. Metrik ini memberikan bobot pada setiap kelas berdasarkan proporsi jumlah sampel dalam setiap kelas.
- *Confusion Matrix* : *Confusion matrix* mencakup nilai-nilai *true positive*, *true negative*, *false positive*, dan *false negative*. Pada dasarnya, ini membantu memahami tingkat kesalahan yang dilakukan oleh model dan sejauh mana model dapat membedakan antara kelas-kelas yang berbeda.

Tabel 3. 5 *Confusion matrix*

Confusion Matrix	
True Positive (TP)	Sampel yang sebenarnya positif dan diprediksi dengan benar sebagai positif.
False Positive (FP)	Sampel yang sebenarnya negatif tetapi salah diprediksi sebagai positif.
False Negative (FN)	Sampel yang sebenarnya positif tetapi salah diprediksi sebagai negatif.
True Negative (TN)	Sampel yang sebenarnya negatif dan diprediksi dengan benar sebagai negatif.