OPTIMASI STRUKTUR UNTUK *DEEP NETWORK* PENGENALAN DIGIT TULISAN TANGAN DENGAN *MORPHNET*

TESIS

KONSENTRASI SISTEM KOMUNIKASI DAN INFORMASI

Ditujukan Untuk Memenuhi Persyaratan
Memperoleh Gelar Magister Teknik

RIDHO HERASMARA
NIM. 186060300111010

KEMENTERIAN RISET, TEKNOLOGI DAN PENDIDIKAN TINGGI
UNIVERSITAS BRAWIJAYA
FAKULTAS TEKNIK
MALANG
2019

TESIS

OPTIMASI STRUKTUR UNTUK DEEP NETWORK PENGENALAN DIGIT TULISAN TANGAN DENGAN MORPHNET

RIDHO HERASMARA
NIM. 186060300111010

Telah dipertahankan di depan penguji
Pada tanggal : **19 Desember 2019**
dinyatakan telah memenuhi syarat
untuk memperoleh gelar Magister Teknik

Komisi Pembimbing,

Pembimbing I	**Pembimbing II**
M. Azis Muslim, S.T., M.T., Ph.D.	**Dr. Eng. Panca Mudjirahardjo, S.T., M.T.**
NIP. 19741203 200012 1 001	NIP. 19700329 200012 1 001

Malang,

Universitas Brawijaya
Fakultas Teknik, Jurusan Teknik Elektro
Ketua Program Magister **Teknik Elektro**

Dr. Eng. Panca Mudjirahardjo, ST., MT.
NIP. 19700329 200012 1 001

PERNYATAAN ORISINALITAS TESIS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya dan berdasarka hasil penelusuran berbagai karya ilmiah, gagasan dan masalah ilmiah yang diteliti dan diulas di dalam Naskah Tesis ini adalah asli dari pemikiran saya. 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 dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata di dalam naskah Tesis ini dapat dibuktikan terdapat unsur-unsur jiplakan, saya bersedia Tesis dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, pasal 25 ayat 2 dan pasal 70).

Malang, 20 Desember 2019

Mahasiswa,

Nama : **RIDHO HERASMARA**

NIM : **186060300111010**

PM : **TEKNIK ELEKTRO**

PROGRAM MAGISTER TEKNIK ELEKTRO

JUDUL TESIS :

OPTIMASI STRUKTUR UNTUK DEEP NETWORK PENGENALAN DIGIT TULISAN TANGAN DENGAN MORPHNET

Nama Mahasiswa	: Ridho Herasmara
NIM	: 186060300111010
Program Studi	: Teknik Elektro
Kekhususan / Minat	: Sistem Komunikasi dan Informatika
KOMISI PEMBIMBING	:
Ketua	: M. Azis Muslim, S.T., M.T., Ph.D.
Anggota	: Dr. Eng. Panca Mudjirahardjo, S.T., M.T.
TIM DOSEN PENGUJI	:
Dosen Penguji I	: Dr. Ir. Sholeh Hadi Pramono, MS
Dosen Penguji II	: Rahmadwati, ST., MT., Ph.D.
Tanggal Ujian	: 19 Desember 2019
SK Penguji	:

RIWAYAT HIDUP

Ridho Herasmara, Dilahirkan di Kota Malang pada hari jum'at tanggal 22 Mei 1987. Anak pertama dari dua bersaudara pasangan dari Poedyasmoro, S.K.M., dan Netty Herawati, B.A. Penulis menyelesaikan studi di jurusan Teknik Elektronika di PENS – ITS, Surabaya, pada tahun 2008, dan jurusan Magister Administrasi Bisnis di ITB pada 2013. Karya tulis ini dibuat untuk meraih gelar Magister Teknik Elektro dari Universitas Brawijaya pada tahun 2019. Saat penulisan, penulis merupakan dosen di Universitas Islam Raden Rahmat.

UCAPAN TERIMA KASIH

Dalam penyusunan tesis ini tidak terlepas dukungan dari berbagai pihak. Peneliti secara khusus mengucapkan terima kasih yang sebesar-besarnya kepada semua pihak yang telah membantu. Atas bimbingan, dukungan, dan bantuannya, pada kesempatan ini penulis menyampaikan rasa terima kasih yang sebesar-besarnya kepada:

1. Allah SWT dengan segala rahmat serta karunia-Nya yang memberikan kekuattan bagi peneliti dalam menyelesaikan tesis ini.
2. Kepada dua orang tua tersayang, Poedyasmoro dan Netty Herawati yang selalu mendukung dalam berbagai kesempatang.
3. Istri saya Susilawati, dan kedua anak saya Adiva Rafanda Putri, dan Muhammad Rafif Alfarizki.
4. Kepada Bapak M. Azis Muslim, S.T., M.T., Ph.D., selaku dosen pembimbing yang selalu memberikan bimbingan, arahan, dorongan, dan bantuan sehingga tesis ini dapat terselesaikan tepat waktu.
5. Kepada Bapak Dr. Eng. Panca Mudjirahardjo, ST., MT. selaku Pembimbing sekaligus Ketua Jurusan Program Magister Teknik Elektro Universitas Brawijaya.
6. Segenap dosen dan seluruh staf akademik yang selalu membantu dalam memberikan fasilitas, ilmu, serta pendidikan pada peneliti hingga dapat menunjang dalam penyelesaian tesis ini.
7. Teman-teman seperjuangan SKI angkatan 2018, Mbak Nur, Mas Diki, Mas Hendri, Mas Ahmad dan Mas Wahyu yang telah memberikan dukungan, semangat, motivasi, serta doa hingga peneliti dapat menyelesaikan tesis ini dengan baik.
8. Serta masih banyak lagi pihak-pihak yang sangat berpengaruh dalam proses penyelesaian tesis yang yang tidak bisa penelti sebutkan satu persatu.

Semoga Allah SWT senantiasa membalas semua kebaikan yang telah diberikan yang telah diberikan. Semoga penelitian ini dapat bermanfaat bagi peneliti umumnya kepada para pembaca.

KATA PENGANTAR

Dengan mengucapkan Alhamdulillah segala puji dan syukur penulis panjatkan atas kehadirat Allah SWT, karena berkat rahmat dan hidayah-Nya penyusunan Tesis yang berjudul "Optimasi Struktur Untuk Deep Network Pengenalan Digit Tulisan Tangan Dengan Morphnet" ini dapat diselesaikan guna memenuhi salah satu persyaratan dalam menyelesaikan pendidikan untuk memperoleh gelar Magister pada Jurusan Elektro Fakultas Teknik Universitas Brawijaya. Diharapkan hasil penelitian ini nantinya dapat dijadikan sebagai bahan rujukan untuk penerapat optimasi deep network dengan pembelajaran menggunakan metode MorphNet. Mengingat metode yang masih relatif baru, diharapkan dapat dihasilkan lebih banyak penelitian pada bidang ini, untuk menambah kinerja dan efisiensi dari deep network pada umumnya.

RINGKASAN

Ridho Herasmara, Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya, Desember 2019, Optimasi Struktur Untuk Deep Network Pengenalan Digit Tulisan Tangan Dengan Morphnet. Dosen Pembimbing: Muhammad Azis Muslim, dan Panca Mudjirahardjo.

Proses perancangan struktur *neural network* umumnya tidak menghasilkan struktur yang optimal. Tidak optimalnya struktur dari *neural network* salah satunya ditunjukkan dengan tidak signifikannya peningkatan kinerja *neural network* ketika ditambahkan sumber daya, semisal dengan penambahan *neuron* maupun *layer* yang berujung pada peningkatan kebutuhan *FLOPS* (*floating operation*) ataupun memori. Hal ini disebabkan kecenderungan penelitian untuk melakukan maksimalisasi akurasi dengan tidak terlalu memperhitungkan sumberdaya yang dibutuhkan dari *neural network*. Telah ada beberapa penelitian untuk mengatasi masalah ini, namun bentuk yang ada baru menyasar kebutuhan memori, atau penyusutan secara *uniform* pada seluruh *layer* yang justru malah menurunkan kinerja *network* secara signifikan. Penelitian terbaru mengusulkan pendekatan MorphNet, yaitu penggunaan *regularizer* yang menyasar kebutuhan sumberdaya sebagai bagian dari *cost* untuk diminimalisir, dan telah diujicobakan pada *network* ImageNet, AudioSet, dan JFT. Namun, belum ada penelitian yang melakukan optimasi dengan pendekatan MorphNet ini pada jaringan yang mengklasifikasi digit tulisan tangan, dan khususnya yang dilatih dengan menggunakan dataset MNIST. Pada penelitian ini akan dilakukan optimasi pada dua *network*, yaitu LeNet5 yang berupa *convolutional neural network* yang telah sedikit dimodifikasi, dan Ciresan yang berupa *fully connected layer*. Diharapkan metode ini menghasilkan struktur baru yang lebih optimal.

Kata kunci : *MorphNet*, *convolutional neural network*, klasifikasi, digit tulisan tangan

SUMMARY

Ridho Herasmara, Electrical Engineering, Faculty of Engineering, University of Brawijaya, December 2019. Structure Optimization of Deep Network for Handwritten Digit Classification with Morphnet. Thesis Supervisor: Muhammad Azis Muslim and Panca Mudjirahardjo.

Generally, neural network structure design process produces inefficient structure. This inefficiency is indicated by insignificant increase in performance of neural network, when additional resource such as additional processing layer is added, resulting in increased FLOPS requirement or memory requirement. This is caused by effort by researcher to maximize accuracy with little regard to resource requirement of the neural network. Some studies has been performed to address the issue, but the current study aim to reduce memory footprint, or by uniformly reduce the size of all layer which caused network performance to decrease significantly. New study proposed MorphNet method, a use of *regularizer* that target resource utilization as a part of the cost to be minimized. This method has been applied to deep networks targeting ImageNet, AudioSet, and JFT. However, no study yet that uses MorphNet to optimize handwritten digit classification network, especially the one that uses MNIST dataset. This study optimize two networks. First, LeNet5 which is a modified convolutional neural network, and CiresanL6, a fully connected multilayer perceptron network. This method is expected to produce an improvised network structure which maintain accuracy while reducing FLOPS requirement.

keyword : *MorphNet*, *convolutional neural network*, classification, handwritten digit

DAFTAR ISI

Halaman

DAFTAR TABEL

No. Judul Halaman

DAFTAR GAMBAR

No. Judul Halaman

BAB I
PENDAHULUAN

1.1 Latar Belakang

Kecerdasan buatan adalah kecerdasan yang ditunjukkan oleh mesin yang menyerupai kecerdasan alamiah yang dimiliki manusia, dalam hal-hal semisal kemampuan menalarkan sesuatu, atau menyelesaikan masalah. Penggunaan kecerdasan buatan, pada saat ini umumnya diterapkan pada bidang-bidang yang membutuhkan repetisi tinggi, namun memiliki kompleksitas yang relatif rendah.

Berdasarkan (Russel & Norvig, 2010) kecerdasan buatan dibagi menjadi beberapa teknik pemecahan masalah. Teknik-teknik ini yaitu pencarian, perencanaan, penalaran, dan pembelajaran. Sejak 2015, pertumbuhan jumlah data yang tersedia seiring dengan otomatisasi pengumpulan data, telah memicu lonjakan jumlah riset dan pengembangan di bidang kecerdasan buatan yang menerapkan teknik pembelajaran.

Teknik pembelajaran mesin terbagi menjadi tiga yaitu *supervised learning*, *unsupervised learning*, dan *reinforcement learning*. Pada penelitian ini akan dibahas mengenai *supervised learning*, yaitu pembelajaran menggunakan solusi terhadap permasalahan yang telah ada untuk mempelajari polanya, dan kemudian digunakan untuk menyelesaikan suatu masalah baru. Pada pembahasan ini akan dibahas khususnya *supervised learning* dalam penerapannya dengan menggunakan jaringan syarat tiruan (*neural network*).

Perkembangan terbaru dari *supervised learning* dengan menggunakan jaringan syarat tiruan (*neural network*) memunculkan metode-metode semisal *convolutional neural network*, dan *deep network*, yang menghasilkan peningkatan kinerja dari *neural network* dalam ketepatan menyelesaikan suatu masalah klasifikasi, terutama yang berhubungan dengan pengenalan citra. Beberapa penelitian yang membahas mengenai penyelesaian masalah klasifikasi citra digit tulisan tangan, dengan dataset MNIST sebagai datanya adalah oleh (Ciresan, Meier, & Schmidhuber, 2012) yang memanfaatkan *deep learning model*, dan oleh (LeCun, Bottou, & Bengio, 1998) yang memanfaatkan pendekatan *convolutional neural network*.

Terlepas dari pengembangan model *neural network* terbaru ini, desain penelitian akademik umumnya berkutat pada maksimalisasi akurasi berapapun biaya yang terkait dengan komputasinya. Hal ini menimbulkan permasalahan

dimana arsitektur yang dihasilkan seringkali menjadi terlalu besar dari segi memori, membutuhkan daya komputasi yang intensif, terlalu kompleks, dan latensi tinggi sehingga tidak mudah untuk diimplementasikan kedalam aplikasi dunia nyata, terutama pada piranti yang memiliki keterbatasan kecepatan pengolahan data, dan memori.

Permasalahan ini juga timbul dalam bidang pengenalan digit tulisan tangan. Beberapa penelitian telah menghasilkan sistem untuk klasifikasi tulisan tangan, yaitu (LeCun, Bottou, & Bengio, 1998) yang menghasilkan *LeNet5*, sebuah *convolutional neural network*, dan oleh (Ciresan, Meier, & Schmidhuber, 2012) yang menghasilkan *CiresanL6*, sebuah *multilayer neural network*. Keduanya dilatih menggunakan database MNIST, yaitu sebuah database berisi 60.000 data digit tulisan tangan yang dikumpulkan oleh *National Institute of Science and Technology*. Kedua hasil tersebut tidak efisien dalam penggunaan sumber daya, salah satunya adalah dalam kebutuhan *FLOPS* (*floating operations*, sebuah ukuran kebutuhan komputasi). Untuk itu perlu dilakukan suatu optimasi terhadap struktur keduanya, khususnya untuk menekan kebutuhan *FLOPS*.

Untuk memecahkan permasalahan ini, telah ada beberapa pendekatan penyelesaian yang ditawarkan, antara lain dengan pencarian arsitektur (Zoph & Le, Neural Architecture Search with Reinforcement Learning, 2017), dan pembelajaran struktur (Zoph, Vasudevan, Shlens, & Le, 2018). Namun, kedua pendekatan ini membutuhkan biaya yang besar pada fase *training*. Selain itu, terdapat pula metode *uniform width multiplier* mampu melakukan optimasi kebutuhan sumber daya. Untuk mengatasi beberapa permasalahan tersebut, beberapa teknik optimasi *neural network* berkembang, khususnya teknik optimasi yang mampu melakukan optimasi dengan biaya kecil. Telah ada beberapa penelitian yang membahas mengenai metode optimasi biaya kecil ini. Salah teknik optimasi ini adalah dengan menggunakan *sparsity-inducing regularizers* (Collins & Kohli, 2014), yang berhasil mengurangi kebutuhan memori dari AlexNet menjadi seperempat saja dengan penurunan akurasi yang minim. Pengembangan selanjutnya membahas mengenai *regularizer* yang secara spesifik menargetkan jumlah *neuron* ketimbang *weight*, seperti pada penelitian oleh (Alvarez & Salzmann, 2016), yang berhasil mengurangi kebutuhan parameter dari sebuah *network*. Penelitian ini kemudian berlanjut pada otomatisasi dalam optimasi struktur *deep network* dengan menggunakan MorphNet seperti diusulkan oleh (Gordon, Eban, & Nachum, 2018),

yang menunjukkan keunggulan yaitu peningkatan kinerja *network* dalam batas penggunaan sumber daya yang sama atau lebih kecil. MorphNet melakukan optimasi terhadap struktur, yaitu jumlah *neuron*, atau *feature maps* dalam tiap lapisan *deep network*, dengan tidak mengubah topologinya. MorphNet telah diujicoba pada *neural network* yang dilatih pada dataset ImageNet (sebuah database citra), dan AudioSet (sebuah database audio).

Berdasarkan beberapa kelebihan tersebut, maka MorphNet dipilih untuk digunakan dalam penelitian **"Optimasi Struktur Untuk *Deep Network* Pengenalan Digit Tulisan Tangan dengan MorphNet"**.

1.2 Rumusan Masalah

Di satu sisi, penelitian mengenai metode optimasi *deep network*, telah merambah pada otomasi prosesnya, yaitu dengan menggunakan *MorphNet*. Di sisi lain, ada dua *network* pengenalan digit tulisan tangan *LeNet5* dan *CiresanL6*, yang tidak efisien. Untuk itu akan dilakukan optimasi dari sisi kebutuhan FLOPS. Dalam melakukan optimasi dengan MorphNet terhadap dua *network* tersebut, maka permasalahan yang akan dibahas dinyatakan sebagai berikut:

1. Bagaimana melakukan optimasi *FLOPS* sistem *LeNet5* dan *CiresanL6*?
2. Bagaimana mengendalikan skala regularisasi l1 dalam metode MorphNet, untuk mengatur imbal-tukar yang terjadi dalam optimasi *FLOPS*?
3. Bagaimana perbedaan kinerja *FLOPS* dan akurasi kedua *network* setelah dilakukan optimasi?

1.3 Pembatasan Masalah

Karena cukup banyak dataset mengenai tulisan tangan, dan banyaknya arsitektur model, metode optimasi, maupun parameter *neural network* yang telah dirancang untuk mengatasinya, maka akan dibatasi permasalahan dalam penelitian ini, menjadi sebagai berikut:

1. Arsitektur penelitian sebelumnya yang akan direplikasi ada dua, yaitu *Convolutional Neural Network* (CNN) dan *Multilayer Deep Network*, dalam hal ini LeNet5m dan CiresanL6, yang dilatih dengan menggunakan database MNIST.
2. Metode optimasi yang akan digunakan menggunakan pendekatan MorphNet.

3. Parameter optimalisasi yang akan dibandingkan adalah kebutuhan komputasional (FLOPS), dan akurasi dari masing-masing sistem yang telah dioptimasi.

1.4 Tujuan

Tujuan kegiatan ini adalah untuk optimasi efisiensi dan kinerja *network* sistem klasifikasi digit tulisan tangan, dengan menggunakan *MorphNet*. Parameter efisiensi dan kinerja dalam penelitian ini adalah kebutuhan *FLOPS* dan akurasi.

1.5 Manfaat

Manfaat yang diharapkan dapat diperoleh dari kegiatan ini antara lain sebagai berikut:

1. Bagi peneliti, dapat menjadi referensi penerapan metode optimasi *neural network* dengan MorphNet
2. Bagi para peneliti lain, dapat menjadi rujukan penelitian mengenai penerapan optimasi model *neural network* pada tulisan tangan.
3. Dapat menjadi dasar pengembangan riset lebih lanjut di bidang optimasi *neural network*, khususnya ke arah pengenalan karakter.

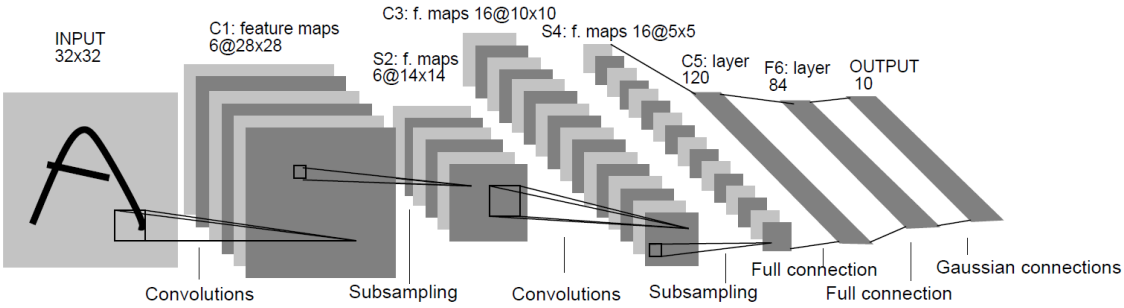
BAB II
TINJAUAN PUSTAKA

2.1 Hasil Penelitian Terkait

2.1.1 Deep Learning dalam Pengenalan Digit Tulisan Tangan

Dalam permasalahan klasifikasi citra digit tulisan tangan, metode pembelajaran mesin ini dilakukan dengan melakukan *training* atau pembelajaran terhadap sistem dengan memasukkan contoh citra yang telah diberi *label* (hasil klasifikasi yang benar), untuk kemudian melakukan perubahan secara gradasi terhadap unsur-unsur bobot di dalam jaringan syarat tiruan, hingga jaringan tersebut dapat melakukan suatu klasifikasi terdahap data citra baru yang belum pernah dilakukan *training* sebelumnya.

Salah satu penelitian dalam bidang klasifikasi ini adalah pengenalan digit tulisan tangan yang dilakukan oleh (LeCun, Bottou, & Bengio, 1998) dengan pendekatan *Convolutional Neural Network* (CNN). Pendekatan ini menggunakan proposisi bahwa nilai pada suatu citra merupakan nilai terbobot dari piksel pada suatu area. Akurasi yang dicapai dengan pendekatan ini adalah 98,3%.



Gambar 2. 1 Arsitektur Convolutional Neural Network LeNet5

Sumber: LeCun((1998, p.7)

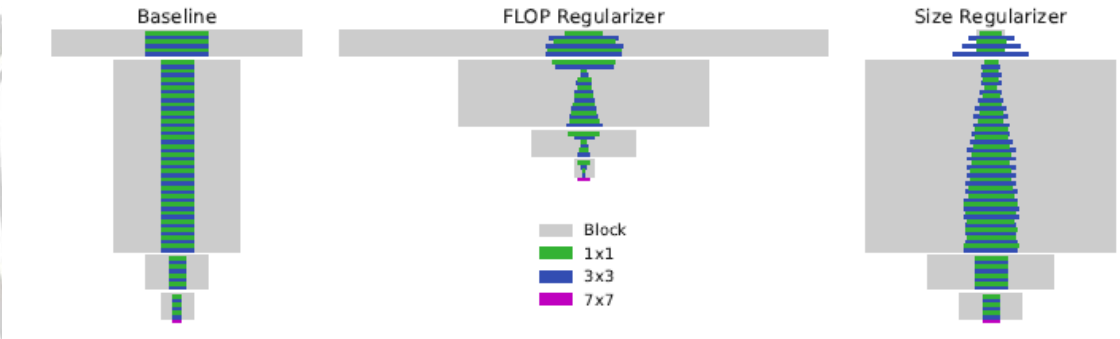
Penelitian lain dalam bidang ini dilakukan oleh (Ciresan, Meier, & Schmidhuber, 2012), yang menggunakan pendekatan *deep learning* untuk permasalahan klasifikasi ini. Akurasi yang dihasilkan oleh pendekatan ini adalah 99,77%.

2.1.2 Optimalisasi Neural network

Penelitian akademik dalam topik *neural network* umumnya menekankan pada peningkatan akurasi, tanpa memperhatikan biaya komputasi maupun ukuran model

jaringan syarat tiruan yang digunakan. Dalam penggunaan secara praktis di dunia nyata, hal ini hampir tidak mungkin dilakukan. Untuk itu beberapa peneliti mengusukan beberapa metode untuk optimalisasi arsitektur *neural network* dengan batasan sumberdaya, seperti yang diusulkan oleh (Collins & Kohli, 2014) dengan *memory-bound*, (Alvarez & Salzmann, 2016) dengan *neuron-targeted pruning*, dan (Gordon, Eban, & Nachum, 2018) dengan menggunkaan pendekatan algoritma yang disebut MorphNet.

MorphNet secara iteratif mengecilkan dan membesarkan jaringan, mengecilkan dengan *resource-weighted sparsifying regularizer* dan membesarkan dengan *uniform multiplicative factors* pada semua lapisan. Ketika diterapkan pada struktur jaringan standard, pendekatan ini mampu menemukan sebuah struktur yang baru pada tiap-tiap ranah, menghasilkan kinerja yang lebih baik dengan tetap mempertahankan batas-batas sumberdaya yang ditetapkan.



Gambar 2. 2 MorphNet untuk Regularisasi FLOP maupun Ukuran Model

Sumber: Gordon, Eban, & Nachum (2018, p. 1587)

MorphNet sendiri melakukan optimalisasi berdasarkan batasan sumberdaya, yaitu *FLOP* atau *Model Size*. Kedua batasan ini terkait cukup erat dengan batasan-batasan fisik pada aplikasi di dunia nyata, dimana biasanya keterbatasan terkait dengan kemampuan prosesor untuk melakukan operasi, yang umumnya dinyatakan dalam satuan FLOP.

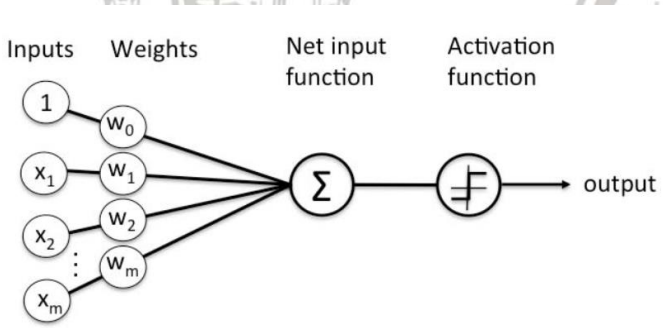
2.2 Dasar Teori

Pada kegiatan ini, teori dasar yang digunakan adalah mengenai *neural network*, dan *convolutional neural network* dengan pengkhususan dalam pengembangannya, yaitu mengenai optimalisasi struktur *neural network* untuk memperkecil ukuran dengan tidak mengorbankan akurasi, atau meningkatkan akurasi, dengan tidak memperbesar ukuran model.

2.2.1 *Neural Network*

Sebuah *neural network* adalah sebuah sistem komputasi yang mencoba mencontoh pola hubungan pada jaringan syaraf biologis yang menyusun sistem kecerdasan pada hewan (Suyanto, 2014). Sistem jenis ini belajar untuk melakukan suatu pekerjaan dengan mempelajari beberapa contoh, dan mencoba untuk melakukan pekerjaan tersebut tanpa diprogram secara khusus dengan aturan mengenai pekerjaan tersebut. Sebagai contoh, pada sistem pengenalan citra, sistem ini mencoba untuk mengidentifikasikan citra yang berisi sebuah kucing didalamnya, dengan menganalisa contoh citra yang telah ditandai secara manual (oleh manusia) sebagai 'kucing' atau 'bukan kucing', dan menggunakan hasilnya untuk mengidentifikasikan kucing pada citra lain. Sistem *neural network* mencoba melakukan hal ini tanpa memiliki pengetahuan pendahuluan mengenai citra, misalnya bahwa kucing memiliki mata, bulu, kumis, dan ciri-ciri khusus wajah kucing. Namun, hal ini secara otomatis dihasilkan dengan mengidentifikasikan karakteristik tersebut dari contoh yang telah dianalisa sebelumnya.

Neural Network didasarkan pada kumpulan unit simpul yang disebut sebagai *neuron*, yang dimodelkan berdasarkan *neuron* pada otak biologis. Setiap hubungan, mencontoh pada sinapsis pada otak biologis, dapat mengirimkan sinyal ke *neuron* lain. Sebuah *neuron* yang menerima sinyal, melakukan pengolahan terhadap sinyal tersebut dan dapat meneruskan sinyal tersebut kepada *neuron* lain yang terhubung dengannya.



Gambar 2. 3 Perceptron

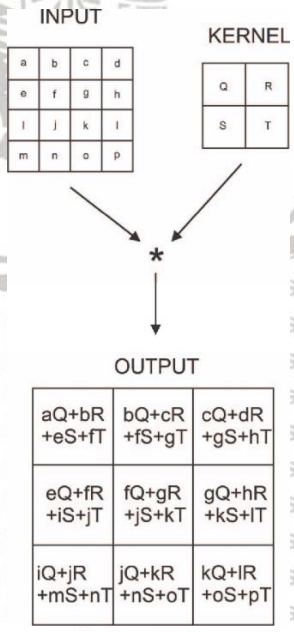
Sumber: https://www.simplilearn.com/what-is-perceptron-tutorial

Pada implementasi *neural network* paling sederhana, sinyal pada hubungan berbentuk nilai real, dan luaran dari tiap *neuron* dihitung menggunakan sebuah fungsi non linear yang menjumlahkan seluruh input, yaitu sinyal dari *neuron* sebelumnya dikalikan sebuah nilai bobot, untuk kemudian dilakukan sebuah fungsi

aktivasi terhadapnya. Fungsi aktivasi yang dilakukan terhadap jumlah semua input tersebut antara lain fungsi *sigmoid*, *rectified linear unit*, dan *tangen hiperbolik*. Model paling sederhana ini disebut dengan *perceptron*, yang digambarkan pada Gambar 2.3.

2.2.2 *Convolutional Neural Network*

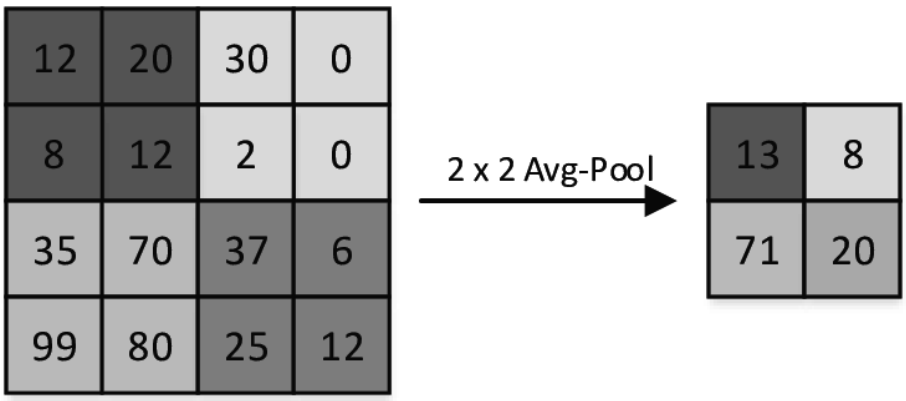
Convolutional Neural Network (CNN) merupakan sebuah kelas *neural network*, yang umum digunakan untuk melakukan analisa terhadap citra visual. *CNN* adalah versi yang teregularisasi dari *perceptron*. Apabila pada *perceptron* pola hubungan antara *neuron* adalah keterhubungan penuh, dimana tiap *neuron* pada suatu lapisan terhubung dengan seluruh *neuron* pada lapisan sebelum dan sesudahnya, pola ini menyebabkan *network* semacam ini rawan mengalami permasalahan *overfitting*, yaitu permasalahan dimana *network* mampu mengenali permasalah saat pelatihan dengan baik, namun memiliki kinerja buruk saat mengenali masalah yang belum pernah dilihat. Sementara, *CNN* menyelesaikan permasalahan ini dengan mengekstrak ciri yang terpola secara hirarkikal didalam data, dan menyusun kembali sebuah pola yang lebih kompleks, dengan menggunakan pola yang lebih kecil dan sederhana, sehingga pola keterhubungan dan kompleksitasnya menjadi sangat jauh berkurang. *CNN* memiliki rancangan yang berbeda dari *perceptron*, dimana rancangannya adalah seperti Gambar 2.4.



Gambar 2. 4 Rancangan Convolutional Neural Network
Sumber: https://www.semanticscholar.org/paper/

Pada *CNN*, luaran (*output*) dari *network* adalah hasil perkalian konvolusi antara masukan (*input*) dengan *kernel*. Ukuran kernel dapat bermacam-macam, namun pada umumnya memiliki bentuk bujur sangkar, dimana jumlah baris dan kolom dari kernel adalah sama. *Kernel* disebut pula sebagai sebuah *filter*, dimana apabila terdapat lebih dari satu buah *filter* konvolusi pada satu lapisan, maka hasil luaran akan terdiri dari beberapa kumpulan *feature maps*, dengan jumlah *feature maps* sama dengan jumlah *filter*.

Setelah *convolutional layer*, umumnya dibuat sebuah *pooling layer*, dimana *pooling layer* ini akan diterapkan sebuah fungsi *pooling* terhadap masukannya. Tujuan dari *layer* ini adalah untuk mengurangi sensitivitas dari *CNN* terhadap pergeseran citra. Fungsi ini menjumlahkan citra dalam cakupannya, melakukan operasi (dalam hal ini operasi rerata), dan mengeluarkan hasilnya sebagai data komposit beberapa masukannya.

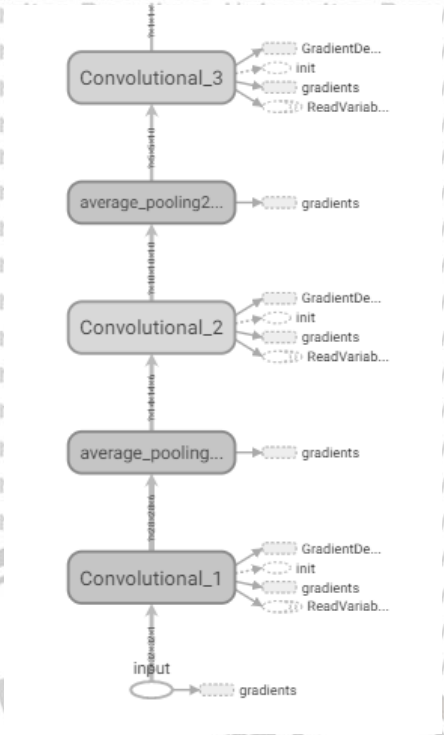


Gambar 2. 5 Perhitungan Pada Average Pooling Layer

Sumber: https://www.researchgate.net/figure/Average-pooling-example_fig21_329885401

2.2.3 Tensorflow

Tensorflow adalah suatu pustaka piranti lunak *(software library)* yang bersumber terbuka untuk komputasi numerik, yang mempermudah dan mempercepat suatu penerapan dari *machine learning*. Pustaka ini mempermudah dan mempercepat penerapan *machine learning* dalam proses akuisisi data, pelatihan model, penampilan prediksi, dan perbaikan kinerja luaran (Yegulalp, 2019). Pustaka ini dikembangkan oleh *Google Brain*, dan dapat diterapkan untuk suatu implementasi pembelajaran mesin dalam skala besar. Pustaka ini memiliki API (Application Programming Interface) dalam beberapa bahasa misalnya *python*, dan *JavaScript*, sementara eksekusi dilakukan pada C++ yang memiliki kinerja tinggi.



Gambar 2. 6 Contoh Visualisai Model Komputasi Tensorflow

Sumber: Data Pribadi

Pustaka tensorflow sedikit berbeda dengan pustaka ataupun *platform* semisal MATLAB, dimana cara kerja pustaka ini menggunakan sebuah pemodelan komputasi, yaitu *dataflow graphs*, dimana model ini mendeskripsikan bagaimana data bergerak melalui simpul-simpul pengolahan. Setiap simpul pada *graphs* melambangkan operasi matematis, sementara semua hubungan atau tepian (*edge)* adalah suatu data multidimensional, atau disebut *tensor*. Suatu contoh pemodelan komputasi ini digambarkan pada Gambar 2.5.

2.3 *Database* Digit Tulisan Tangan MNIST

MNIST (*Modified National Institute of Standards and Technology*) *database*, merupakan sebuah *database* berisi kumpulan digit (angka) tulisan tangan yang umum digunakan untuk keperluan pelatihan sistem pengolahan citra. *Database* ini juga digunakan secara luas untuk pelatihan dan pengujian di bidang pembelajaran mesin. *Database* ini disusun dengan cara mengolah ulang sampel dari dataset NIST asli. Pembuat *database* ini merasakan bahwa karena dataset pelatihan yang digunakan NIST diambil dari biro sensus amerika serikat, sementara dataset untuk validasi atau pengujian, diambil dari siswa sekolah menengah atas amerika, dataset

tersebut menjadi kurang cocok untuk eksperimen terkait pembelajaran mesin. Lebih lanjut lagi, data citra hitam putih dari dataset NIST, dinormalisasi menjadi berukuran piksel 28x28, dan terhadapnya dilakukan proses *anti-aliasing*, dan kemudian ditampilkan sebagai *grayscale* (1-255).

Gambar 2. 7 Sampel Citra Tulisan Tangan dalam MNIST Database

Sumber: https://upload.wikimedia.org/wikipedia/commons/2/27/MnistExamples.png

Database MNIST ini berisi 60.000 citra untuk pelatihan, dan 10.000 citra untuk pengujian. Separuh dari citra pelatihan dan pengujian diambil dari dataset pelatihan NIST, sementara separuh lainnya diambil dari dataset pengujian NIST. Pengembangan lebih lanjut dari MNIST ini disebut EMNIST yang diterbitkan pada 2017, yang berisi 240.000 citra pelatihan, dan 40.000 citra pengujian, yang berisi tulisan tangan digit, dan karakter.

2.4 *Stochastic Gradient Descent*

Stochastic Gradient Descent (umumnya disingkat sebagai SGD) adalah sebuah metode iterative untuk optimasi sebuah fungsi yang memiliki karakteristik kehalusan tertentu. Metode ini dapat dianggap sebagai sebuah estimasi stokastik terhadap optimasi menggunakan *gradient descent*, karena metode ini menggantikan gradien sesungguhnya (yang dikalkulasi dari seluruh dataset) dengan sebuah estimasi yang dihitung dari sebagian data yang dipilih secara acak. Penerapan

metode ini pada permasalahan yang memiliki dataset yang besar mengurangi beban komputasi, meningkatkan kecepatan iterasi, dengan sedikit mengurangi kecepatan konvergensi (Bottou & Bousquet, 2012). Kelebihan ini menjadikan *stochastic gradient descent* menjadi penting dalam penerapannya dalam bidang pembelajaran mesin.

Pada metode *gradient descent* biasa, *cost function* yang digunakan dapat dijabarkan dalam Persamaan 2.1. metode *gradient descent* biasa menggunakan seluruh data dalam perhitungan *cost function* dari *network*.

$$J(\Theta) = \frac{1}{2}\sum_{i=1}^{m}(y_{(i)} - h_\Theta(x_{(i)}))^2 \quad (2.1)$$

Dimana:

J : Fungsi *cost* dari *network*

m : jumlah data *training*

y : luaran sebenarnya

h_Θ : fungsi prediksi luaran

Pada metode *gradient descent*, permaslahan ini diselesaikan dengan mencari nilai Θ yang meminimalkan nilai J(Θ). Dengan menggunakan *gradient descent*, kita melakukan inisialisasi atas semua parameter, dan melakukan pembaruan parameter dengan menggunakan aturan pembaruan sebagai berikut:

$$\theta_j \coloneqq \theta_j - \alpha\frac{d}{d\theta_j}J(\theta) \quad (2.2)$$

Dimana:

θ_j' : Parameter network baru ke-j

θ_j : Parameter network lama ke-j

α : Rasio Pembelajaran (*Learning Rate)*

Ketika data berukuran besar dan banyak, penggunaan *gradient descent* menjadi mahal dari sisi biaya komputasi dan waktu yang dibutuhkan. Untuk itu digunakan *Stochastic Gradient Descent*, dimana perbaruan dilakukan setiap kali, dengan menggunakan data yang dipilih secara acak dari himpunan data pembelajaran.

2.5 Penelitian Sebelumnya

Optimasi struktur pada *neural network* dengan berbagai pendekatan telah dilakukan oleh beberapa penelitian sebelumnya, dan beberapa diantaranya dirangkum dalam tabel-tabel berikut. Didapati bahwa kondisi penelitian terkini adalah dengan menggunakan pendekatan pencarian, dan pendekatan pembelajaran dalam proses optimasi *deep network*. Metode pencarian memiliki keunggulan dari efektivitasnya, namun relatif tidak efisien. Metode pembelajaran memiliki keunggulan dari efisiensi yang tinggi, walau memiliki sedikit kekurangan dibandingkan metode pencarian.

Tabel 2. 1 Penelitian oleh Gordon Dkk

Author, Year	Gordon dkk, 2016
Problem	Optimasi kebutuhan sumber daya sistem *deep network* dengan menggunakan pendekatan pembelajaran, memanfaatkan *sparsity-inducing regularizer*.
Scope of problem	*Network* yang dioptimasi adalah beberapa *network benchmark* yaitu Inception V2, ResNet 101, dan AudioResNet.
Method	• Sparsity-inducing regularizer, yaitu *Group Lasso* dan *Gamma norm*, untuk memicu kekosongan pada *neuron*. • Uniform-width multiplier, untuk membesarkan ukuran *network*.
Result	Peningkatan akurasi pada *network*, yaitu Inception V2 meningkat sebesar 1.5%, ResNet101 meningkat sebesar 2.1%, dan AudioResNet meningkat sebesar 2.18%

Gordon dkk sebagaimana Tabel 2.1, mencoba mengatasi kekurangan dari metode pencarian dalam optimasi *deep network*, dengan menggunakan pendekatan pembelajaran. Pendekatan pembelajaran mencoba menyelesaikan permasalahan dengan memanfaatkan solusi yang sudah ada, yang belum optimal, atau memiliki cakupan bahasan yang mirip, untuk dioptimasi.

Tabel 2. 2 Penelitian oleh Zoph & Le

Author, Year	Zoph & Le, 2017
Problem	Pencarian arsitektur *neural network* yang optimal untuk menyelesaikan suatu permasalahan.
Scope of problem	*Struktur network* yang dicari adalah *network* untuk melakukan klasifikasi terhadap database berisi 60.000 citra, bernama *CIFAR-10*
Method	• *Pencarian* • *Recurrent Neural Network* • *Reinforcement Learning*
Result	Metode optimasi dengan pencarian, mampu menghasilkan struktur klasifikasi database *CIFAR-10* yang memiliki error hanya 3,65%, lebih tinggi dari hasil penelitian sebelummya.

Namun, penelitian oleh (Zoph & Le, 2017) sebagaimana Tabel 2.2 menekankan pada hasil akhir yang memiliki kinerja akurasi tinggi. Kinerja tinggi ini hanya dapat dicapai dengan sebuah metode pencarian yang membutuhkan biaya besar.

Tabel 2. 3 Penelitian oleh Zoph Dkk.

Author, Year	Zoph Dkk, 2018
Problem	Pencarian arsitektur *neural network* yang optimal untuk menyelesaikan suatu permasalahan.
Scope of problem	*Struktur network* yang dicari adalah *network* untuk melakukan klasifikasi terhadap database ImageNet.
Method	• *Pencarian* menggunakan NASNet, *neural architecture search network.* • *Convolutional Neural Network* • *Transfer learning.* • *ScheduledDropPath.*
Result	Metode pembelajaran dengan pendekatan pencarian, mampu menghasilkan struktur klasifikasi database *CIFAR-10* yang memiliki error hanya 3,65%, lebih tinggi dari hasil penelitian sebelummya.

Tabel 2. 4 Penelitian oleh Pham dkk

Author, Year	Pham dkk, 2018
Problem	Pencarian arsitektur optimal dengan mencari sub-graph yang optimal dari graph keseluruhan, untuk dilakukan parameter sharing kepada model turunan.
Scope of problem	*Struktur network* yang dicari adalah *network* untuk melakukan klasifikasi terhadap database berisi 60.000 citra, bernama *CIFAR-10*
Method	• Pencarian arsitektur. • *Parameter Sharing* • *Recurrent Neural Network*
Result	Metode pembelajaran dengan pendekatan pencarian, mampu menghasilkan struktur klasifikasi database *CIFAR-10* yang memiliki error hanya 2,89%, lebih tinggi dari hasil penelitian sebelummya.

2.6 Daftar Istilah

Penelitian ini akan menggunakan beberapa istilah yang digunakan dalam ranah penelitian terkait kecerdasan buatan. Penjelasan atas beberapa istilah tersebut dimuat dalam Tabel 2.5.

Tabel 2. 5 Daftar Istilah

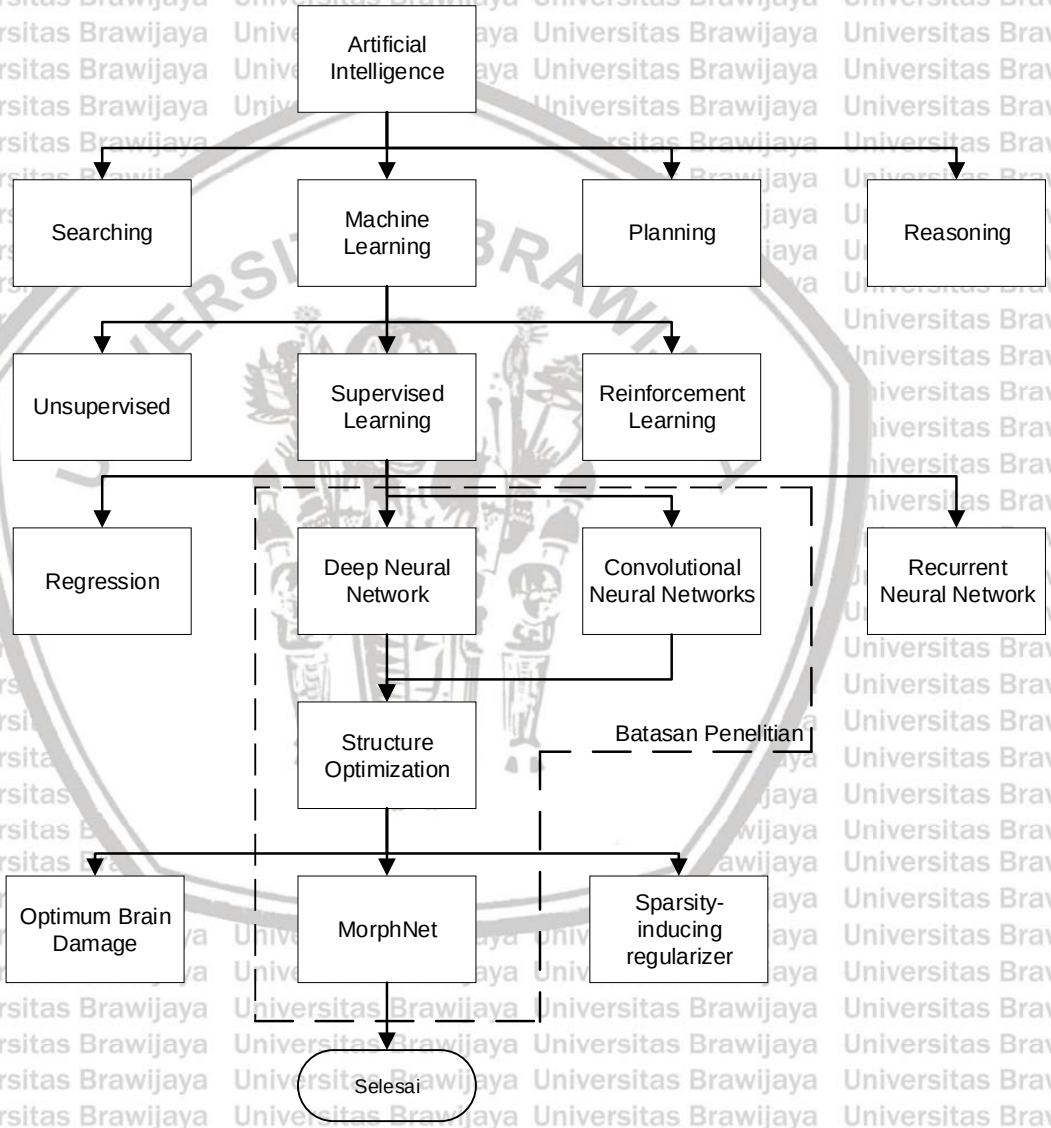
Istilah	Penjelasan
Convolutional Neural Network	*Neural network* yang bekerja dengan menggunakan metode konvolusi dari data n-dimensi, terhadap sebuah kernel m-dimensi.
Gradient Descent	Sebuah metode optimasi parameter dalam *neural network* untuk meminimalisir *cost function*. Metode ini menghitung gradien dari seluruh data untuk melakukan pembaruan parameter dengan rasio setara dengan *learning rate*.
Stochastic Gradient Descent	Sebuah metode optimasi parameter dalam *neural network* untuk meminimalisir *cost function*. Metode ini menghitung gradien dari sebagian data untuk

	melakukan pembaruan parameter dengan rasio setara dengan *learning rate*.
FLOPS	Floating operations, jumlah operasi yang diperlukan oleh komputer dalam menyelesaikan sebuah kalkulasi.
Grayscale	Skala keabuan yang melambangkan derajat kehitaman sebuah citra, dimana dalam penggunaan ini, 0 adalah putih, dan 1 adalah hitam.
MorphNet	Sebuah metode optimasi struktur *neural network*, dengan menahan pada suatu parameter.
Group LASSO	Sebuah metode regularisasi yang menerapkan penalti terhadap cost function untuk memicu kekosongan pada bobot grup-grup *neuron*.
Network benih	*Network* yang akan dioptimasi dengan menggunakan pendekatan metode MorphNet

BAB III
KERANGKA KONSEP PENELITIAN

3.1 Kerangka Berpikir

Kerangka pemikiran dalam penyusunan tulisan ini dituangkan kedalam Gambar 3.1. Gambar ini menunjukkan konsep yang dibahas dalam kedudukannya pada cabang ilmu kecerdasan buatan secara umum.



Gambar 3. 1 Kerangka Konsep Model Deep Network Yang Akan dioptimasi
Sumber: Data pribadi

Permasalahan utama dalam penelitian ini adalah bagaimana melakukan sebuah klasifikasi terhadap digit tulisan tangan, dengan menggunakan sumber daya yang terbatas. Penelitian yang telah ada sebelumnya telah membahas mengenai arsitektur

CNN dan *deep learning* untuk permasalahan klasifikasi, dengan menggunakan MNIST dataset sebagai data latihannya. CNN mampu mencapai akurasi tinggi, dan Deep Learning mampu mencapai akurasi yang lebih tinggi. Maka dalam kegiatan ini, akan coba digunakan pendekatan optimasi MorphNet untuk optimasi *neural network.*

3.2 Hipotesis

Berdasarkan penelitian sebelumnya, disusun hipotesis sementara sebagai berikut:

H1: MorphNet mampu menghasilkan *network* yang lebih optimal dari sisi penggunaan flops, yaitu memiliki akurasi yang sama dengan sebelum optimasi namun memiliki kebutuhan *FLOPS* lebih rendah, atau memiliki kebutuhan *FLOPS* yang relatif setara, namun memiliki akurasi yang lebih baik

H2: Skala regularisasi l1 dengan nilai (1/flops) akan mampu melakukan regularisasi terhadap *network* sasaran.

H3: Terjadi perbedaan efisiensi akibat terjadinya *feature selection* yang menghilangkan hubungan-hubungan yang tidak diperlukan.

3.3 Definisi Operasional

Variabel operasional penelitian akan didefinisikan sebagai berikut:

1. Jumlah *FLOPS*: Jumlah FLOP (*floating point operations*) dalam satu siklus inferensi.
2. Jumlah Neuron: Jumlah total neuron dalam model *neural network*.
3. Akurasi: Rasio ketepatan prediksi kelas terhadap kelas sesungguhnya.

Variabel operasional tersebut menunjukkan tingkat efisiensi *neural network* dalam menyelesaikan masalah terhadap kebutuhan sumberdayanya. Semakin sedikit jumlah neuron dan/atau FLOPS yang dibutuhkan dalam sebuah *neural network* untuk mencapai tingkat akurasi yang sama atau lebih baik, menggambarkan efisiensi yang lebih tinggi dalam arsitektur *neural network* tersebut. Hal ini diharapkan dapat dicapai dengan pendekatan MorphNet.

BAB IV
METODE PENELITIAN

Bab ini akan menjelaskan mengenai metode dan pendekatan yang digunakan dalam penelitian ini. Akan dijabarkan mengenai alur waktu penelitian, alat dan bahan yang digunakan, kerangka solusi, dan teknik yang digunakan. Bahasan utama dalam kegiatan ini adalah penggunaan teknik MorphNet untuk melakukan optimasi terhadap model.

4.1 Jenis dan Cara Perolehan Data

Jenis data kuantitatif yang terkait dalam penelitian ini terbagi menjadi dua buah, yaitu data tulisan tangan MNIST, dan data terkait struktur dan parameter *neural network.* Data MNIST berisi data digit tulisan tangan berbentuk angka *grayscale*. Data terkait struktur dan parameter *neural network* yang akan diperoleh antara lain, jumlah *neuron* tiap layer, jumlah *feature map* pada *convolutional layer*, *weight*, kebutuhan *FLOPS*, dan akurasi dari *network* tersebut.

Sementara berdasarkan sumbernya, data primer yang digunakan adalah berupa data struktur *neural network* hasil replikasi penelitian sebelumnya, yang dinamakan LeNet5m, dan CiresanL6, serta data struktur hasil optimasi menggunakan morphNet. Sementara untuk data sekunder berupa dataset digit tulisan tangan MNIST didapatkan dari *public repository* yang dapat diakses secara luas.

4.2 Variabel dan Cara Analisis Data

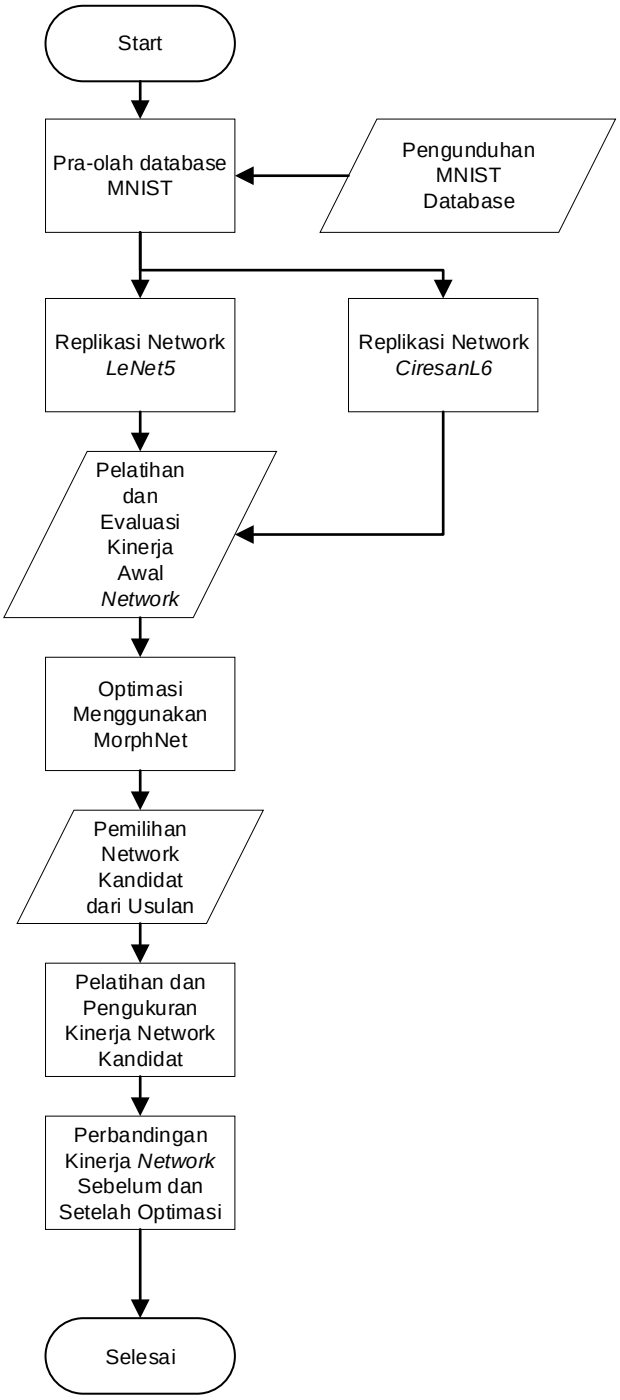
Dalam penelitian ini ada tiga variable yang akan diteliti:

1. Jumlah Neuron: Jumlah total *neuron*, atau *feature maps* dalam tiap lapisan model *neural network*.
2. Jumlah FLOPS: Kebutuhan komputasi saat melakukan inferensi.
3. Akurasi: Rasio ketepatan klasifikasi terhadap total klasifikasi yang dilakukan oleh model.

Analisis dilakukan dengan cara membandingkan jumlah neuron, jumlah FLOPS, dan akurasi dari model sebelum dan sesudah optimasi. Untuk memastikan data, terutama akurasi adalah valid, dilakukan validasi dengan menggunakan dataset *testing* MNIST.

4.3 Tahapan Penelitian

Prosedur untuk menjawab permasalahan penelitian adalah sebagaimana ditampilkan pada Gambar 4.1. Prosedur penelitian dimulai dengan pengunduhan dan pra-olah database MNIST, untuk kemudian diakhiri dengan analisis terhadap kinerja optimasi.



Gambar 4. 1 Bagan Alir Prosedur Penelitian

Sumber: Data pribadi

Tahapan-tahapan penelitian dijabarkan sebagai berikut:

1. Pengunduhan MNIST Database

   Pada kegiatan ini, database MNIST diunduh dari modul bawaan pustaka *tensorflow*. Data ini akan digunakan pada proses pembelajaran *network*.

2. Pra-olah database MNIST

   Database MNIST yang telah diunduh, memiliki skala *grayscale* yang berada pada rentang 0 – 255. Untuk mempercepat konvergensi saat proses pelatihan *neural network*, maka data akan diubah kedalam rentang 0 – 1. Untuk penggunaan pada *network LeNet5*, perlu dilakukan *padding*, yaitu penambahan nilai 0 pada sisi-sisi luar citra untuk memastikan bahwa titik tengah kernel membaca seluruh citra masukan.

3. Replikasi Network LeNet5

   Neural network dengan arsitektur yang mengikuti LeNet5, disusun ulang, dengan menggunakan pustaka piranti lunak *tensorflow*.

4. Replikasi Network CiresanL6

   Neural network dengan arsitektur yang mengikuti CiresanL6, disusun ulang, dengan menggunakan pustaka piranti lunak *tensorflow*.

5. Pelatihan dan Evaluasi Kinerja Awal *Network*

   Dari kedua network hasil replikasi, dilakukan pelatihan dengan menggunakan database MNIST, untuk kemudian diukur kinerjanya dari sisi Akurasi dan Penggunaan FLOPS

6. Optimasi Menggunakan MorphNet

   Pada tahapan ini, dilakukan optimasi terhadap kedua *network* dengan menggunakan pendekatan MorphNet. *Network* akan dikecilkan dengan menggunakan *L1-regularizer* Group LASSO. Proses ini akan menghasilkan beberapa *network* usulan.

7. Pemilihan *Network* Kandidat dari Usulan

   Dari usulan yang dihasilkan pada proses sebelumnya, dipilih beberapa kandidat yang mewakili beberapa tingkat pengerdilan, yaitu dengan rentang pengurangan FLOPS berkisar antara 10 – 20% pada tiap tahapan.

8. Pelatihan Dan Pengukuran Kinerja *Network* Kandidat

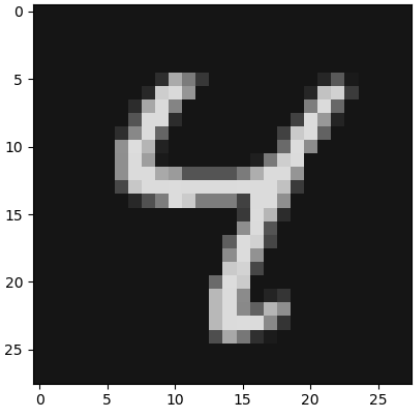
   Terhadap *network* kandidat, dilakukan pelatihan dengan menggunakan database MNIST, untuk kemudian diukur kinerjanya dari sisi Akurasi dan Penggunaan FLOPS.

9. Perbandingan Kinerja *Network* Sebelum dan Sesudah Optimasi

Hasil pengukuran kinerja dari *network* sebelum dan sesudah dilakukan optimasi, dibandingkan untuk menganalisa apakah benar terjadi optimasi dari sisi penggunaan FLOPS, atau tidak.

4.3.1 Pengunduhan MNIST Database

MNIST database adalah database berisi digit tulisan tangan, sebanyak 70.000 data yang berupa 60.000 data yang digunakna untuk pembelajaran, dan 10.000 data pengujian. Database ini disediakan untuk digunakan secara luas dalam pelatihan dan validasi dalam bidang *machine learning*. Bentuk database ini adalah data citra digit tulisan tangan berukuran 28 x 28, dimana setiap titik data memiliki nilai *grayscale* dalam rentang 1 – 255. Salah satu contoh citra yang ada dalam database MNIST ini ditunjukkan pada Gambar 4.2.



Gambar 4. 2 Citra Dalam Database MNIST

Sumber: http://yann.lecun.com/exdb/mnist/

Database ini diunduh dari pustaka *tensorflow* karena merupakan salah satu database bawaan dalam distribusinya. Dalam proses pengunduhan database ini, data akan diubah formatnya menjadi bentuk *array* multidimensi. Sehingga, database ini dinyatakan dalam bentuk matriks sebagaimana Persamaan 4.1.

$$X^m = \begin{bmatrix} x^m_{1,1} & \cdots & x^m_{1,28} \\ \vdots & \ddots & \vdots \\ x^m_{28,1} & \cdots & x^m_{28,28} \end{bmatrix} \quad (4.1)$$

Dimana:

X : matriks citra

4.3.2 Pra-olah *Database* MNIST

Database MNIST yang telah diunduh dari pustaka *tensorflow* masih memiliki nilai dengan rentang 0-255. Pada praktik *machine learning*, optimasi kinerja pembelajaran dapat dicapai dengan melakukan normalisasi terhadap data masukan. Namun pada kegiatan ini, data hanya dikonversi dari rentang 0-255 menjadi nilai pada rentang 0-1, untuk mempercepat konvergensi pada saat pelatihan. Hal ini dilakukan dengan mengikuti Persamaan 4.2.

$$x' = \frac{x}{255} \tag{4. 2}$$

Dimana:

x’ = Data setelah konversi

Maka, terhadap data citra berbentuk matriks yang telah diunduh sebelumnya, dilakukan operasi skalar terhadap semua elemennya. Operasi ini ditunjukkan pada Persamaan 4.3.

$$\mathrm{X'^m} = \frac{1}{255}\begin{bmatrix} x^m_{1,1} & \cdots & x^m_{1,28} \\ \vdots & \ddots & \vdots \\ x^m_{28,1} & \cdots & x^m_{28,28} \end{bmatrix} \tag{4. 3}$$

Dimana:

X’ : matriks citra setelah pra olah

X : matriks citra sebelum pra olah

Setelah dilakukan pengubahan rentang nilai dari nilai masukan perlu dilakukan suatu penyesuaian untuk *network LeNet5*. *Network* LeNet5 membutuhkan masukan dalam bentuk 32 x 32, sehingga perlu dilakukan penyesuaian terhadap data yang ada. Penyesuaian dilakukan dengan menambahkan *padding* pada tepian citra untuk menjadikan ukuran citra berukuran 32 x 32. Nilai yang ditambahkan pada saat proses padding ini adalah nilai 0 dengan tujuan bahwa nilai ini tidak akan

memberikan pengaruh negatif terhadap kinerja akurasi pengenalan. Bentuk citra setelah operasi padding dinyatakan dalam Persamaan 4.4.

$$X'^{m} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x_{1,1}^{m} & \cdots & x_{1,28}^{m} & 0 & 0 \\ 0 & 0 & \vdots & \ddots & \vdots & 0 & 0 \\ 0 & 0 & x_{28,1}^{m} & \cdots & x_{28,28}^{m} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.4)$$

Dimana:

X' : matriks citra hasil padding

Untuk *network* CiresanL6, tidak dilakukan operasi padding sehingga hanya data asli dari MNIST yang digunakan untuk pembelajaran dari *network* CiresanL6 adalah data yang tidak di-*padding*.

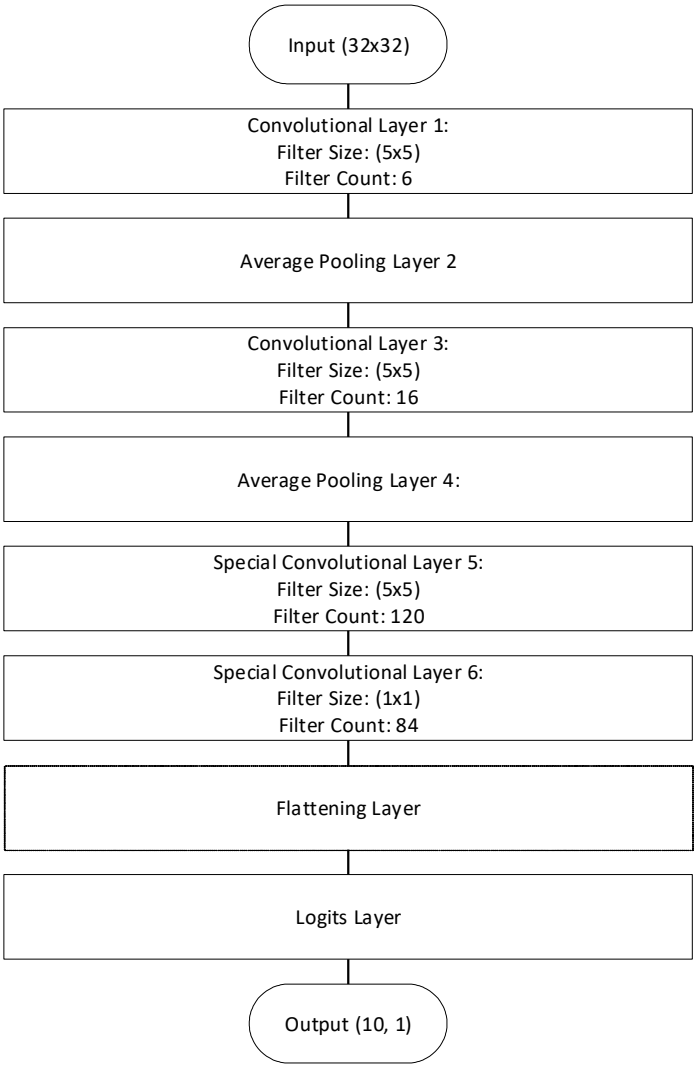
4.3.3 Replikasi Network LeNet5

Tahapan selanjutnya adalah dengan melakukan replikasi terhadap *network* yang dihasilkan oleh penelitian sebelumnya (LeCun, Bottou, & Bengio, 1998), yaitu network LeNet5. Pada tahapan ini, struktur LeNet5 dikonstruksi ulang dengan menggunakan pustaka piranti lunak *tensorflow* dengan menggunakan bahasa pemrograman *python*. Terdapat sedikit perbedaan dengan LeNet5, dimana pada penelitian ini, seluruh konektivitas terhubung secara penuh. Struktur ini akan disebut sebagai LeNet5m, yang strukturnya ditunjukkan pada Gambar 4.3.

Struktur ini tersusun secara sekuensial, dimana luaran suatu lapisan digunakan sebagai masukan lapisan setelahnya. Lapisan konvolusi berada pada lapisan 1, 3, 5 dan 6, sementara lapisan *average pooling* berada pada lapisan 2 dan 4. Pada penelitian ini, lapisan yang terhubung penuh diimplementasikan menggunakan lapisan konvolusi khusus (*Special Convolutional Layer*), yaitu konvolusi yang mana *feature maps* luarannya memiliki dimensi (1x1).

Implementasi komputasi pada *Convolutional Layer 1* menggunakan Persamaan 4.5. Komputasi pada lapisan pertama ini menggunakan *filter* dengan ukuran 5 x 5, dengan 6 buah *filter* yang berbeda. Masukan dari lapisan pertama ini

berupa citra dengan ukuran 32 x 32. Dari masukan berukuran 32 x 32 ini, setelah dilakukan konvolusi melalui lapisan ini, akan dihasilkan citra dengan ukuran 28 x 28 x 6.



Gambar 4. 3 Struktur LeNet5m

$$y_{i,j,n}^{m} = x_{i,j}^{m}.K_{i,j,n} + x_{i+1,j}^{m}.K_{i+1,j,n} + x_{i+2,j}^{m}.K_{i+2,j,n} + x_{i+3,j}^{m}.K_{i+3,j,n} + x_{i+4,j}^{m}.K_{i+4,j,n} + x_{i,j+1}^{m}.K_{i,j+1,n} + \cdots + x_{i+4,j+4}^{m}.K_{i+4,j+4,n} + b_{i,j} \quad (4.5)$$

Dimana:

y : unsur luaran konvolusi

x : unsur masukan konvolusi

K : kernel konvolusi

b : *bias*

m : indeks citra

n : indeks *feature maps* pada luaran konvolusi

i : indeks baris matriks citra

Implementasi komputasi pada *Average Pooling Layer 2* menggunakan Persamaan 4.6. Lapisan ini mengambil masukan berukuran (2 x 2) dari citra masukan, melakukan operasi rerata terhadap masukan tersebut, dan menghasilkan nilai rataannya. Citra masukan yang digunakan berukuran 28 x 28 x 6, dan akan menghasilkan citra luaran hasil *average pooling* dengan ukuran 14 x 14 x 6.

$$y_{i,j,n}^{m} = \frac{x_{2i-1,2j-1,n}^{m} + x_{2i,2j-1,n}^{m} + x_{2i-1,2j,n}^{m} + x_{2i,2j,n}^{m}}{4} \quad (4.6)$$

Dimana:

y : unsur luaran *pooling*

x : unsur masukan *pooling*

b : *bias*

m : indeks citra

n : indeks *feature maps* pada luaran konvolusi

Implementasi komputasi pada *Convolutional Layer 3* menggunakan Persamaan 4.7. Komputasi pada lapisan pertama ini menggunakan *filter* dengan

ukuran 5 x 5, dengan 16 buah *filter* yang berbeda. Masukan dari lapisan pertama ini berupa citra dengan ukuran 14 x 14 x 6. Dari masukan berukuran 14 x 14 x 6 ini, setelah dilakukan konvolusi melalui lapisan ini, akan dihasilkan citra dengan ukuran 10 x 10 x 16. Karena terdapat lebih dari satu *feature map*, maka terdapat sedikit perbedaan dibandingkan konvolusi pada layer pertama.

$$Y_{i,j,n}^{m} = x_{i,j,1}^{m}.K_{i,j,1} + x_{i,j,2}^{m}.K_{i,j,2} + x_{i,j,1}^{m}.K_{i,j,1} + \cdots + x_{i+4,j+4,6}^{m}.K_{i+4,j+4,6} + b_{i,j,n} \quad (4.7)$$

Dimana:

y : unsur luaran konvolusi

x : unsur masukan konvolusi

b : *bias*

m : indeks citra

n : indeks *feature maps* pada luaran konvolusi

i : indeks baris matriks citra

Implementasi komputasi pada *Average Pooling Layer 4* menggunakan Persamaan 4.8. Lapisan ini mengambil masukan berukuran (2 x 2) dari citra masukan, melakukan operasi rerata terhadap masukan tersebut, dan menghasilkan nilai rataannya. Citra masukan yang digunakan berukuran 10 x 10 x 16, dan akan menghasilkan citra luaran hasil *average pooling* dengan ukuran 5 x 5 x 16.

$$y_{i,j,n}^{m} = \frac{x_{2i-1,2j-1,n}^{m} + x_{2i,2j-1,n}^{m} + x_{2i-1,2j,n}^{m} + x_{2i,2j,n}^{m}}{4} \quad (4.8)$$

Dimana:

y : unsur luaran konvolusi

x : unsur masukan konvolusi

b : *bias*

m : indeks citra

n : indeks *feature maps* pada luaran konvolusi

i : indeks baris matriks citra

Implementasi komputasi pada *Special Convolutional Layer 5* menggunakan Persamaan 4.9. Komputasi pada lapisan pertama ini menggunakan *filter* dengan ukuran 5 x 5, dengan 120 buah *filter* yang berbeda. Masukan dari lapisan pertama ini berupa citra dengan ukuran 5 x 5 x 16. Dari masukan berukuran 5 x 5 x 16 ini, setelah dilakukan konvolusi melalui lapisan ini, akan dihasilkan citra dengan ukuran 1 x 1 x 120. Pola hubungan pada lapisan ini menyerupai struktur hubung penuh pada *perceptron*.

$$y_{1,1,n}^{m} = x_{1,1,1}^{m}.K_{1,1,1} + x_{1,1,2}^{m}.K_{1,1,2} + x_{1,1,3}^{m}.K_{1,1,3} + \cdots + x_{5,5,16}^{m}.K_{5,5,16} + b_n \quad (4.9)$$

Dimana:

y : unsur luaran konvolusi

x : unsur masukan konvolusi

K : kernel konvolusi

b : bias

m : indeks citra

Implementasi komputasi pada *Special Convolutional Layer 6* menggunakan Persamaan 4.10. Komputasi pada lapisan pertama ini menggunakan *filter* yang berukuran 1 x 1, dengan 84 buah *filter* yang berbeda. Masukan dari lapisan pertama ini berupa citra dengan ukuran 1 x 1 x 120. Dari masukan berukuran 1 x 1 x 120 ini, setelah dilakukan konvolusi melalui lapisan ini, akan dihasilkan citra dengan ukuran 1 x 1 x 84.

$$y_{1,1,n}^{m} = x_{1,1,1}^{m}.K_{n,1} + x_{1,1,2}^{m}.K_{n,2} + \cdots + x_{1,1,120}^{m}.K_{n,120} + b_n \quad (4.10)$$

Dimana:

y : unsur luaran konvolusi

x : unsur masukan konvolusi

K : kernel konvolusi

b : bias

m : indeks citra

Selanjutnya, dilakukan implementasi komputasi *flattening layer* yang dinyatakan dalam Persamaan 4.11. Lapisan ini hanya mengubah bentuk matriks citra hasil konvolusi masukan yang berukuran 1 x 1 x 84 menjadi data berukuran 84. Apabila masukannya sejumlah *m*, maka masukannya dapat dinyatakan sebagai m x 1 x 1 x 84, sementara luarannya berbentuk m x 84.

$$y_n^m = x_{1,1,n}^m \quad (4.11)$$

Dimana:

y : unsur luaran *flattening*

x : unsur masukan *flattening*

m : indeks citra

Lapisan terakhir, sering disebut sebagai *Logits Layer*, digunakan untuk menghasilkan nilai probabilitas kategori. Nilai ditunjukkan dalam sepuluh buah *neuron*, yang mana tiap *neuron* akan menghasilkan skor probabilitas sebuah citra termasuk pada suatu kelas tertentu. Implementasi komputasional untuk ini dinyatakan pada Persamaan 4.12.

$$y_o^m = x_1^m . w_{1,o} + x_2^m . w_{2,o} + \cdots + x_{84}^m w_{84,o} + b_o \quad (4.12)$$

Dimana:

y : unsur luaran

x : unsur masukan

w : bobot

b : *bias*

m : indeks citra

Dari urutan lapisan pada *network* LeNet5 ini, masukan citra sejumlah m berukuran 32 x 32 akan menghasilkan data probabilitas dalam bentuk *array* 1-dimensi sejumlah m dengan ukuran 10 tiap data. Untuk memprediksi suatu kategori citra, digunakan fungsi *softargmax* terhadap data ini, sebagaimana dijabarkan dalam Persamaan 4.13.

$$y^m = softargmax(x_1^m, x_2^m, \dots, x_{10}^m) \quad (4.13)$$

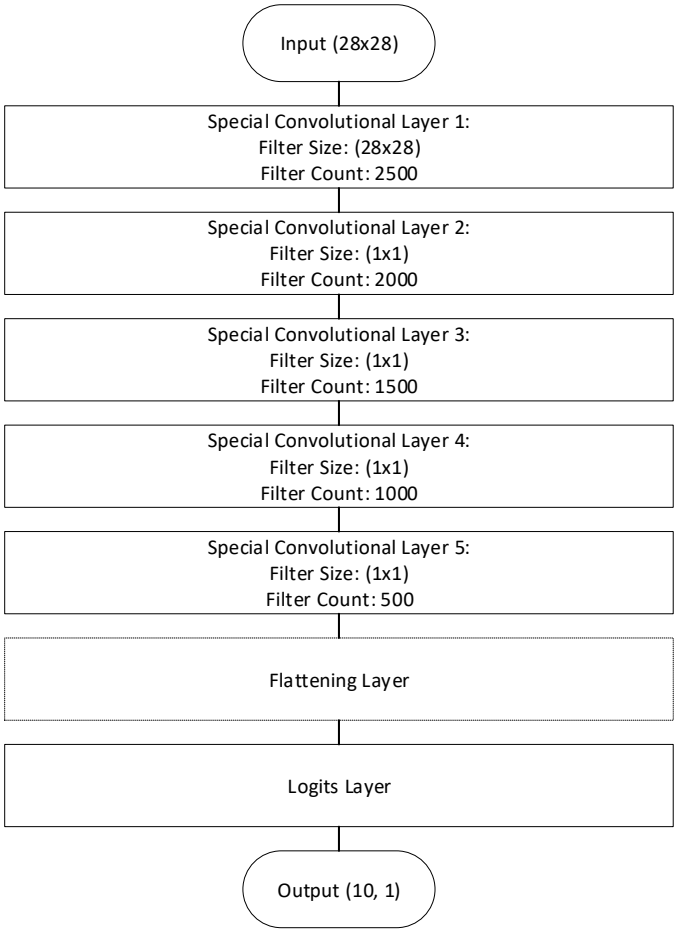
Dimana:

y : Kategori citra

x : Nilai probabilitas

4.3.4 Replikasi Network CiresanL6

Kegiatan yang sama dilakukan untuk membuat sebuah replikasi dari *network* yang dihasilkan oleh penelitian sebelumnya (Ciresan, Meier, & Schmidhuber, 2012), yaitu network *CiresanL6*. Struktur CiresanL6 dikonstruksi ulang dengan menggunakan *tensorflow*. Dalam replikasi ini, struktur *neuron* yang terhubung penuh, akan diimplementasikan dengan menggunakan lapisan *CNN*, yang ukuran luaran tiap filternya adalah 1x1. Struktur ditampilkan pada Gambar 4.4.



Gambar 4. 4 Struktur CiresanL6

Sebagaimana struktur pada *LeNet5*, struktur ini disusun secara sekuensial, dimana luaran suatu lapisan digunakan langsung sebagai masukan lapisan setelahnya.

Implementasi komputasi pada *Special Convolutional Layer 1* menggunakan Persamaan 4.14. Komputasi pada lapisan pertama ini menggunakan *filter* dengan ukuran 28 x 28, dengan 2500 buah *filter* yang berbeda. Masukan dari lapisan pertama ini berupa citra dengan ukuran 28 x 28. Dari masukan berukuran 28 x 28 ini, setelah dilakukan konvolusi melalui lapisan ini, akan dihasilkan citra dengan ukuran 1 x 1 x 2500.

$$y_{i,j,n}^{m} = x_{i,j}^{m}.K_{i,j,n} + x_{i+1,j}^{m}.K_{i+1,j,n} + x_{i+2,j}^{m}.K_{i+2,j,n} + x_{i+3,j}^{m}.K_{i+3,j,n} + x_{i+4,j}^{m}.K_{i+4,j,n} + \cdots + x_{i+27,j+27}^{m}.K_{i+27,j+27,n} + b_n \quad (4.14)$$

Dimana:

y : unsur luaran konvolusi

x : unsur masukan konvolusi

K : kernel konvolusi

b : bias

m : indeks citra

n : indeks *feature maps* luaran konvolusi

i : indeks baris

Implementasi komputasi pada *Special Convolutional Layer 2* menggunakan Persamaan 4.15. Komputasi pada lapisan ini menggunakan *filter* berukuran 1 x 1, dengan 2000 buah *filter* yang berbeda. Masukan dari lapisan pertama ini berupa array berukuran 1 x 1 x 2500. Dari masukan berukuran 1 x 1 x 2500 ini, setelah dilakukan konvolusi melalui lapisan ini, akan dihasilkan citra dengan ukuran 1 x 1 x 2000. Pada lapisan konvolusi ini juga dignakan *bias* seperti pada penerapan umumnya. Persamaan akan sedikit berbeda karena unsur masukan berupa *array* 3 dimensi

$$y_{1,1,n}^{m} = x_{1,1,1}^{m}.K_{1,n} + x_{1,1,2}^{m}.K_{2,n} + \cdots + x_{1,1,2500}^{m}.K_{2500,n} + b_n \quad (4.15)$$

Dimana:

y : unsur luaran konvolusi

x : unsur masukan konvolusi

K : kernel konvolusi

b : bias

m : indeks citra

n : indeks *feature maps* luaran konvolusi

i : indeks baris

Implementasi komputasi pada *Special Convolutional Layer 3* menggunakan Persamaan 4.16. Komputasi pada lapisan ini menggunakan *filter* berukuran 1 x 1, dengan 1500 buah *filter* yang berbeda. Masukan dari lapisan pertama ini berupa array berukuran 1 x 1 x 2000. Dari masukan berukuran 1 x 1 x 2000 ini, setelah dilakukan konvolusi melalui lapisan ini, akan dihasilkan citra dengan ukuran 1 x 1 x 1500.

$$y_{1,1,n}^{m} = x_{1,1,1}^{m}.K_{1,n} + x_{1,1,2}^{m}.K_{2,n} + \cdots + x_{1,1,2000}^{m}.K_{2000,n} + b_n \quad (4.16)$$

Dimana:

y : unsur luaran konvolusi

x : unsur masukan konvolusi

K : kernel konvolusi

b : bias

m : indeks citra

n : indeks *feature maps* luaran konvolusi

i : indeks baris

Implementasi komputasi pada *Special Convolutional Layer 4* menggunakan Persamaan 4.17. Komputasi pada lapisan ini menggunakan *filter* berukuran 1 x 1, dengan 1000 buah *filter* yang berbeda. Masukan dari lapisan pertama ini berupa array berukuran 1 x 1 x 1500. Dari masukan berukuran 1 x 1 x 1500 ini, setelah dilakukan konvolusi melalui lapisan ini, akan dihasilkan citra dengan ukuran 1 x 1 x 1000.

$$y_{1,1,n}^{m} = x_{1,1,1}^{m}.K_{1,n} + x_{1,1,2}^{m}.K_{2,n} + \cdots + x_{1,1,1500}^{m}.K_{1500,n} + b_n \quad (4.17)$$

Dimana:

y : unsur luaran konvolusi

x : unsur masukan konvolusi

K : kernel konvolusi

b : bias

m : indeks citra

n : indeks *feature maps* luaran konvolusi

i : indeks baris

Implementasi komputasi pada *Special Convolutional Layer 5* menggunakan Persamaan 4.18. Komputasi pada lapisan ini menggunakan *filter* berukuran 1 x 1, dengan 500 buah *filter* yang berbeda. Masukan dari lapisan pertama ini berupa array berukuran 1 x 1 x 1000. Dari masukan berukuran 1 x 1 x 1000 ini, setelah dilakukan konvolusi melalui lapisan ini, akan dihasilkan citra dengan ukuran 1 x 1 x 500. Pada lapisan setelah SCL5 ini akan ditempatkan sebuah lapisan *flattening*, yang akan mengubah data luaran dari lapisan ini, menjadi data vektor (data 1-dimensi). Data luaran dari *flattening layer* akan dijadikan sebagai masukan kepada *logit layer*, yaitu lapisan yang akan menghasilkan nilai probabilitas bahwa sebuah citra tergolong kedalam suatu kategori.

$$y_{1,1,n}^{m} = x_{1,1,1}^{m}.K_{1,n} + x_{1,1,2}^{m}.K_{2,n} + \cdots + x_{1,1,1000}^{m}.K_{1000,n} + b_n \quad (4.\ 18)$$

Dimana:

y : unsur luaran konvolusi

x : unsur masukan konvolusi

K : kernel konvolusi

b : bias

m : indeks citra

n : indeks *feature maps* luaran konvolusi

i : indeks baris

Selanjutnya, dilakukan implementasi komputasi *flattening layer* yang dinyatakan dalam Persamaan 4.19. Lapisan ini hanya mengubah bentuk matriks citra hasil konvolusi masukan yang berukuran 1 x 1 x 500 menjadi data berukuran 500. Apabila masukannya sejumlah *m*, dan masukannya dapat dinyatakan sebagai m x 1 x 1 x 500, maka luarannya akan berbentuk m x 500.

$$y_{n}^{m} = x_{1,1,n}^{m} \quad (4.\ 19)$$

Dimana:

y : unsur luaran *flattening*

x : unsur masukan *flattening*

m : indeks citra

Lapisan terakhir, sering disebut juga sebagai *Logits Layer*, digunakan untuk menghasilkan nilai probabilitas kategori. Nilai ditunjukkan dalam sepuluh buah *neuron*, yang mana tiap *neuron* akan menghasilkan skor probabilitas sebuah citra termasuk pada suatu kelas tertentu. Tiap *neuron* ini akan mengambil masukan dari seluruh data luaran lapisan sebelumnya, yaitu data sebanyak m berukuran 500. Implementasi komputasional untuk ini dinyatakan pada Persamaan 4.20.

$$y_o^m = x_1^m . w_{1,o} + x_2^m . w_{2,o} + \cdots + x_{500}^m w_{500,o} + b_o \quad (4.\ 20)$$

Dimana:

y : unsur luaran

x : unsur masukan

w : bobot

b : *bias*

m : indeks citra

Dari urutan lapisan pada *network* CiresanL6 ini, masukan citra sejumlah m berukuran 28 x 28 akan menghasilkan data probabilitas dalam bentuk *array* 1-dimensi sejumlah m dengan ukuran 10 tiap data. Untuk memprediksi suatu kategori citra, digunakan fungsi *softargmax* terhadap data ini, sebagaimana dijabarkan dalam Persamaan 4.21.

$$y^m = softargmax(x_1^m, x_2^m, \ldots, x_{10}^m) \quad (4.\ 21)$$

Dimana:

y : Kategori citra

x : Nilai probabilitas

4.3.5 Pelatihan dan Evaluasi Kinerja Awal *Network*

Setelah dilakukan replikasi terhadap dua *network* LeNet5m dan CiresanL6, kemudian dilakukan proses pelatihan terhadap keduanya, dengan menggunakan data pelatihan sebanyak 60.000 data yang berasal dari database MNIST. Pelatihan dilakukan dengan menggunakan algoritma pelatihan *Stochastic Gradient Descent* (SGD), yaitu algoritma pelatihan yang mencoba mencapai fungsi sasaran Persamaan 4.22, dengan mencari gradien dari fungsi sasaran.

$$\min_{\theta} L(\theta), \quad (4.\ 22)$$

Dimana:

Θ : Parameter *network*, yaitu nilai pada *weight*, *bias*, dan *kernel*

L : Fungsi *loss* model

Pelatihan menggunakan algoritma pelatihan SGD, pada kedua *network* ini dilakukan dengan menggunakan perhitungan seperti pada Persamaan 4. 23.

$$\theta_j' := \theta_j - \alpha \frac{d}{d\theta_j} J(\theta) \quad (4.\ 23)$$

Dimana:

θ_j' : Parameter network baru ke-j

θ_j : Parameter network lama ke-j

α : Rasio Pembelajaran (*Learning Rate)*

Network yang telah dilatih, kemudian diuji dengan menggunakan dataset MNIST sebanyak 10.000 data yang tidak digunakan pada pelatihan. Pada proses pengujian ini, dicatat akurasi dan kebutuhan FLOPS dari masing-masing *network*. Data ini akan dijadikan sebagai *baseline* (acuan) untuk membandingkan kinerja dari *network* hasil optimasi nantinya. Implementasi komputasional untuk perhitungan fungsi *cost* dari network digunakan Persamaan 4.24. Dari persamaan tersebut akan didapatkan nilai *loss* dari *network* pada saat kumpulan parameter tersebut.

$$J = \frac{y_1^1.\log(ypred_1^1) + y_2^1.\log(ypred_2^1) + \cdots + y_{10}^m.\log(ypred_{10}^m)}{m} \quad (4.\ 24)$$

Dimana:

y : kategori sebenarnya

ypred : kategori prediksi

m : jumlah

4.3.6 Optimasi Menggunakan MorphNet

MorphNet merupakan teknik yang diterapkan secara iterative dalam dua tahap, yaitu pengerdilan dan pembesaran. Pengerdilan dilakukan dengan *sparsity-inducing regularizer*, dan pembesaran dilakukan dengan menggunakan *fixed-width multiplier*. MorphNet melakukan optimasi pada **lebar *output*** dari tiap *layer*. Apabila luaran *network* benih (*network* awal yang akan dipelajari strukturnya)

dinyatakan sebagai $O^0_{1:M}$ dan kinerja FLOPS yang diinginkan dinyatakan dalam $\mathcal{F}(O_{1:M}) \leq \varsigma$, maka optimasi dilakukan mengikuti Persaman 4.25.

$$O^*_{1:M} = \underset{\mathcal{F}(O_{1:M}) \leq \varsigma}{\operatorname{argmin}} \min_{\theta} L(\theta) \quad (4.25)$$

Dimana:

$O^*_{1:M}$: Lebar Luaran Baru

$\mathcal{F}(O_{1:M})$: Fungsi Pengukuran FLOPS dari Lebar Luaran Lama

ς : Nilai FLOPS Awal

L : Fungsi *loss* model

Tahap pertama yang akan dilakukan yaitu proses pengerdilan *network* benih. Dengan menerapkan *sparsifying regularizer*, dilakukan penyusutan secara grup untuk memicu kekosongan *aktivasi* pada *neuron-neuron* di tiap *layer*. Teknik yang digunakan pada tahap ini adalah L1 regularizer, yaitu Group LASSO (*Least Absolute Shrinkage and Selection Operator*) seperti pada Persamaan 4.26.

$$(1-\beta).\sum\sqrt{\sum(\theta_g)^2} + \beta.\sum\|\theta_g\| \quad (4.26)$$

Dimana:

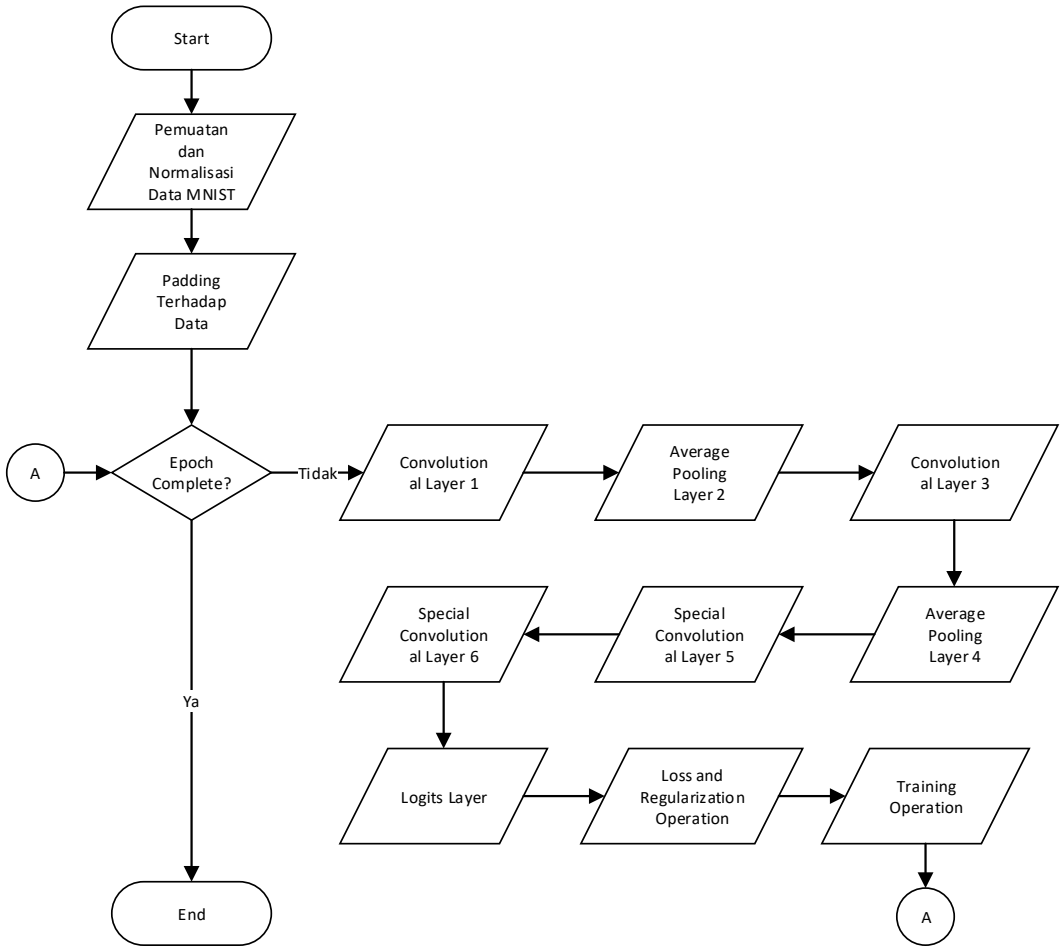
β = Koefisien Bobot Antara Regularisasi L1 dan L2

θ_g = Parameter Grup g

Tahap ini akan menghasilkan jaringan yang memiliki ukuran lebih kecil dari *network* benih, dan mungkin memiliki kinerja yang lebih rendah. Namun, proses pengerdilan ini memiliki kecenderungan menghilangkan *neuron-neuron* yang memiliki imbas rendah terhadap akurasi *network*. Implementasi metode MorphNet, pada *network LeNet5m* dijabarkan pada Gambar 4.5. Pada implementasinya, ditambahkan operasi regularisasi pada saat pelatihan *network*. Pada tiap beberapa iterasi pelatihan, dilakukan pencatatan terhadap struktur *network*, yaitu dicatat jumlah *filter* aktif pada tiap *convolutional layer*. Filter yang aktif adalah filter yang semua unsurnya memiliki nilai kurang dari 0,001.

Tahap kedua, adalah pembesaran, yaitu dengan menggunakan *width multiplier* ω yang diterapkan secara *uniform* terhadap semua layer dalam *network*, hingga mencapai ukuran yang tetap memenuhi batasan FLOPS maksimum yang

ditetapkan. Pembesaran secara uniform ini bertujuan meningkatkan kemampuan inferensi jaringan yang telah dioptimasi.



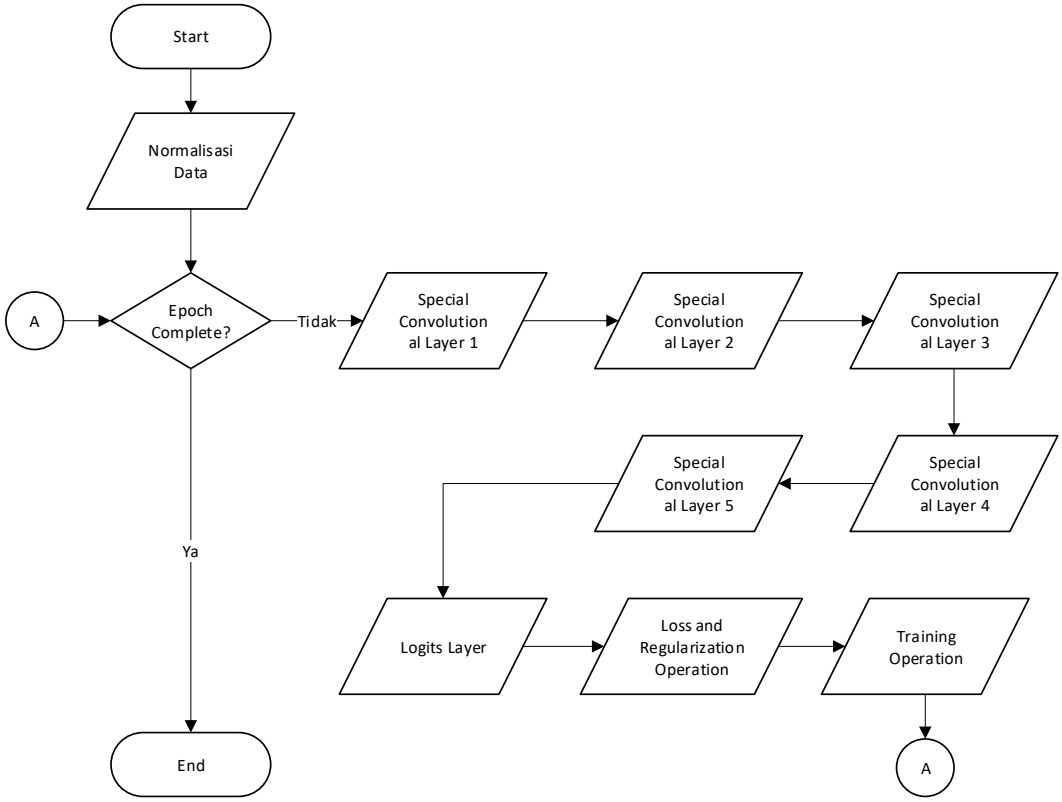
Gambar 4. 5 Implementasi Komputasional MorphNet untuk Optimasi LeNet5m

Siklus dua tahapan ini akan dilakukan satu kali atau diulang dalam beberapa iterasi untuk menghasilkan usulan struktur yang lebih optimal. Kemudian, dilakukan *training* terhadap struktur usulan yang telah dipilih, untuk mendapatkan hasil akhir *network* yang telah dioptimasi. Pada kasus tertentu, teknik ini dapat menimbulkan perubahan dalam topologi jaringan. Perubahan itu berupa penghilangan layer, apabila semuanya neuron dapat dihapus dan terdapat *skip connection*, atau penghilangan cabang yang salah satu layer dalam cabang tersebut/

Pada akhir tahapan penelitian, *network* hasil optimasi dengan pendekatan MorphNet ini dibandingkan kinerjanya dari sisi akurasi dan kebutuhan FLOPS terhadap data *baseline*. Diharapkan akurasi *network* hasil optimasi, lebih tinggi dibandingkan model *baseline* pada tingkat FLOPS yang sama, akurasi yang setara

dengan model *baseline* pada tingkat FLOPS yang lebih rendah, atau kombinasi dari keduanya.

Siklus dua tahap dari *MorphNet* yang diterapkan terhadap network *LeNet5m* ini akan diterapkan pula terhadap network *CiresanL6*. Implementasinya ditunjukkan pada Gambar 4.6.



Gambar 4. 6 Implementasi Komputasi untuk Optimasi CiresanL6

Pada dua implementasi MorphNet diatas, penggunaan regularisasi L1 dilakukan pada proses *Loss and Regularization Operation*. Pada operasi ini, perhitungan *cost* dari *network* memasukkan unsur *loss* dari akurasi dan kebutuhan sumberdaya *network*. Untuk kebutuhan sumberdaya *network*, digunakan *term* regularisasi yang dihitung dengan menggunakan Persamaan 4.27.

$$R = \lambda\left(\sqrt{\left(g_{1,1}\right)^2 + \left(g_{1,2}\right)^2 + \cdots + \left(g_{1,o}\right)^2} + \cdots + \sqrt{\left(g_{226,1}\right)^2 + \left(g_{226,2}\right)^2 + \cdots + \left(g_{226,o}\right)^2}\right) \quad (4.\ 27)$$

Dimana:

R : *term* regularisasi

λ : skala regularisasi

g : parameter unsur grup

Sehingga, dengan menggunakan MorphNet, *cost* dari suatu network adalah jumlah dari Persamaan 4.24 dan Persamaan 4.27, yang dituangkan kedalam Persamaan 4.28.

$$L = J + R \quad (4.\ 28)$$

Dimana:

L : *Cost* total dari *network*

J : Loss dari akurasi

Dengan rumusan *cost* dari *network* tersebut, maka aturan untuk melakukan perbaruan terhadap parameter *network* pada saat pembelajaran, akan mengikuti Persamaan 4.29.

$$\theta_i' = \theta_i - \frac{\alpha}{m} L\left(\frac{e_i^1 + \cdots + e_i^m}{m}\right) \quad (4.\ 29)$$

Dimana:

Θ : parameter sebelum pembaruan

Θ' : parameter setelah pembaruan

i : indeks parameter

α : *Learning Rate*

m : jumlah citra pembelajaran

4.3.7 Pemilihan Network Kandidat dari Usulan

Proses optimasi menggunakan MorphNet, akan menghasilkan beberapa *network* usulan. Usulan-usulan tersebut berupa usulan struktur dengan berbagai kombinasi jumlah luaran tiap lapisan. Pada kasus LeNet5 dan CiresanL6, kombinasi luaran adalah kombinasi jumlah *feature maps*, yang berarti berupa kombinasi jumlah *filter* konvolusi pada tiap lapisannya.

Usulan-usulan ini memiliki jumlah luaran yang berbeda-beda, yang berarti kebutuhan FLOPSnya juga berbeda. Untuk mendapatkan gambaran kinerja pada beberapa tingkat pengerdilan, maka pada proses ini akan diambil network dari beberapa tingkat kebutuhan FLOPS yang berbeda, dengan jarak sekitar 10 – 20% dari kebutuhan FLOPS awal untuk jarak antar tingkatnya. Akan diambil lima buah *network* kandidat dari *LeNet5m*, dan lima kandidat dari *CiresanL6*.

4.3.8 Pelatihan Dan Pengukuran Kinerja Network Kandidat

Setelah kandidat hasil optimasi didapatkan, kemudian dilakukan proses pelatihan kembali terhadap semua kandidat tersebut, dengan menggunakan data pelatihan sebanyak 60.000 data yang berasal dari database MNIST. Pelatihan dilakukan dengan menggunakan algoritma pelatihan *Stochastic Gradient Descent* (SGD). *Network* yang telah dilatih ini, kemudian diuji dengan menggunakan dataset MNIST sebanyak 10.000 data yang tidak digunakan pada pelatihan. Pada proses pengujian ini, dicatat akurasi dan kebutuhan FLOPS dari masing-masing *network*. Data ini akan dibandingkan dengan hasil pengukuran *baseline*.

4.3.9 Perbandingan Kinerja Network Sebelum dan Sesudah Optimasi

Pada tahapan ini, hasil pengukuran kinerja *network* sebelum dan sesudah optimasi, dibandingkan dari sisi akurasi dan sisi kebutuhan FLOPS. Diharapkan bahwa semua *network* hasil optimasi memiliki kebutuhan FLOPS yang lebih rendah namun tetap mempertahankan kinerja akurasi.

4.4 Pengujian *Network* Hasil Optimasi Pada Komputer Berbeda

Untuk memastikan bahwa hasil optimasi struktur dapat diimplementasikan dengan baik, maka struktur optimasi akan dicoba pada dua unit komputer yang berbeda, yang memiliki kemampuan berbeda, untuk dapat memastikan optimasi

yang dihasilkan dari optimasi struktur dengan menggunakan pendekatan *MorphNet*.

Komputer pertama yang mana struktur baru akan dicoba, memiliki dua *central processing unit core*. Penggunaan dua core ini akan termasuk dengan *overhead* yang dibutuhkan oleh *library* dan *platform* komputer. Komputer kedua yang mana struktur baru akan dicoba, memiliki empat *central processing unit core*. Penggunaan empat core ini akan termasuk dengan overhead yang dibutuhkan oleh *library* dan *platform* komputer.

Pengujian pada dua komputer ini dilakukan dengan mensimulasikan sepuluh kali pelatihan di tiap komputer, serta mencatat waktu yang dibutuhkan untuk eksekusi di kedua komputer tersebut. Dua catatan waktu ini kemudian dianalisa untuk mencari perbedaan kinerja eksekusi pada dua komputer yang berbeda.

BAB V
HASIL DAN PEMBAHASAN

Pada penelitian ini, digunakan dua buah *network* yang dilatih menggunakan database MNIST, untuk dioptimasi dengan pendekatan metode MorphNet. Skenario optimasi adalah sebagai berikut:

1. Optimasi *network* LeNet5m
2. Optimasi *network* CiresanL6

Hasil optimasi akan dianalisis dari segi penggunaan sumber daya *FLOPS*, dan kinerja akurasinya. Setelah dilakukan optimasi terhadap kedua network, struktur sebelum dan sesudah optimasi dari kedua *network* tersebut, akan diuji pada dua komputer berbeda untuk memastikan bahwa terdapat peningkatan kinerja yang dihasilkan dari optimasi.

5.1 Penggunaan *Tensorflow library*

Pada penelitian ini, bahasa pemrograman yang digunakan untuk implementasi model komputasinya adalah *python*, dengan menggunakan *library tensorflow* dengan versi 1.14 dan memiliki kemampuan untuk eksekusi pada GPU. *Tensorflow* sendiri, menggunakan paradigma pemrograman yang berbasis sesi, dimana pada saat pembuatan hanya dilakukan penyusunan terhadap model komputasi, sementara kalkulasi sesungguhnya dijalankan dengan memasukkan data pada saat sesi dijalankan.

5.2 Proses Replikasi Network Benih

Sesuai dengan langha 1 hingga 4 dari tahapan penelitian, maka dilakukan replikasi terhadap *network* yang telah dihasilkan sebelumnya, yaitu *network* yang diberi nama *LeNet5m* dan *CiresanL6*. Replikasi dilakukan dengan membuah sebuah model komputasi yang menggambarkan struktur dari *network* tersebut dalam bahasa *Python* dan *library tensorflow*. Network ini kemudian akan digunakan sebagai *network* benih, yaitu *network* yang akan dioptimasi strukturnya melalui proses pembelajaran dengan MorphNet.

5.2.1 Replikasi LeNet5

LeNet5 merupakan sebuah *convolutional neural network* untuk pengenalan digit tulisan tangan. Dalam pendekatan morphnet, *network* akan diasumsikan sebagai *convolutional neural network*, dimana sebuah *network* yang terhubung secara penuh dianggap sebagai sebuah kasus khusus dari konvolusi, yaitu filter yang digunakan memiliki ukuran 1x1. Kemudian, dari gambaran dan deskripsi pada penelitian sebelumnya (LeCun, Bottou, & Bengio, 1998), maka parameter dari jaringan di tiap lapisan dapat dijabarkan seperti ditunjukkan oleh Tabel 5.1.

Tabel 5. 1 Arsitektur LeNet5m

Lapisan ke-	**Jenis *Network***	**Parameter**
1	Input	Ukuran: (m,32x32x1)
2	Convolutional	Filters: 6, kernel_size: (5x5), activation = ‘tanh’
3	Pooling	type: average_pooling
4	Convolutional	Filters: 16, kernel_size: (5x5), activation = ‘tanh’
5	Pooling	type: average_pooling
6	Convolutional	Filters: 120, kernel_size: (5x5), activation = ‘tanh’
7	Convolutional	Filters: 84, kernel_size: (1x1), activation = ‘tanh’

dimana m adalah jumlah data yang terdapat dalam dataset. Untuk penelitian ini digunakan dataset MNIST, sehingga nilai m akan menjadi 60.000. Pada penelitian asli dari jaringan ini, ada beberapa jaringan yang tidak terhubung dengan jaringan setelahnya, namun pada penelitian ini, seluruh jaringan memiliki hubungan dengan lapisan setelahnya, sehingga penyebutannya menjadi *LeNet5m* untuk menunjukkan perbedaannya.

Dari rancangan tersebut, maka kode yang digunakan dalam *python* adalah sebagai berikut:

```
# Implementation of LeNet5m
# Convolutional layer 1
```

```
        conv1 = tf.keras.layers.Conv2D(filters=6, kernel_size=(5,5), padding='VALID',
            input_shape=(32,32,1), activation='tanh', name="Convolutional_1")(x)
        # Subsampling / Average Pooling 2
        pool2 = tf.keras.layers.AveragePooling2D()(conv1)
        # Convolutional layer 3
        conv3 = tf.keras.layers.Conv2D(filters=16, kernel_size=(5,5), padding='VALID',
            activation='tanh', name="Convolutional_2")(pool2)
        # Subsampling / Average Pooling 4
        pool4 = tf.keras.layers.AveragePooling2D()(conv3)
        # Flattening (Not in original)
        conv5 = tf.keras.layers.Conv2D(120, (5,5), padding='valid', activation='tanh', name='
Dense_3')(pool4)
        # Fully connected layer, input 120, output 84,
        conv6 = tf.keras.layers.Conv2D(84, (1,1), padding='valid', activation='tanh', name='D
ense_4')(conv5)
        # Flatten layer
        flat7 = tf.keras.layers.Flatten()(conv6)
        # Output logits
        logits = tf.keras.layers.Dense(10)(flat7)
```

Dan sesuai dengan tahapan ke 5 dari penelitian, setelah dilakukan replikasi, akan dilakukan pelatihan ulang terhadap jaringan dengan menggunakan database MNIST. Dari proses pelatihan ini didapatkan data *baseline* mengenai kinerja dari network LeNet5 sebagaimana Tabel 5.2.

Tabel 5. 2 Kinerja Awal Network LeNet5m

Deskripsi	Nilai
Akurasi Pelatihan	99,66%
Akurasi Validasi	98.41%
Kebutuhan FLOPS	833 kFLOPS

5.2.2 Replikasi CiresanL6

CiresanL6 adalah sebuah *multilayer neural network*, yang terdiri dari banyak *neuron* dalam beberapa lapisan. Arsitektur *network* ini berasal dari penelitian

terdahulu (Ciresan, Meier, & Schmidhuber, 2012), yang mencoba memecahkan permasalahan klasifikasi database MNIST dengan penambahan jumlah *neuron* dan *layer*. Dalam pendekatan morphnet, *network* akan diasumsikan sebagai *convolutional neural network*, dimana sebuah *network* yang terhubung secara penuh dianggap sebagai sebuah kasus khusus dari konvolusi, yaitu filter yang digunakan menghasilkan *feature maps* dengan ukuran 1x1. Parameter *network* ini adalah sebagai berikut:

Tabel 5. 3 Arsitektur CiresanL6

Lapisan ke-	**Jenis *Network***	**Parameter**
1	Input	Ukuran: (m,28x28x1)
2	Convolutional	Filters: 2500, kernel_size: (28x28), activation = ‘tanh’
3	Convolutional	Filters: 2000, kernel_size: (1x1), activation = ‘tanh’
4	Convolutional	Filters: 1500, kernel_size: (1x1), activation = ‘tanh’
5	Convolutional	Filters: 1000, kernel_size: (1x1), activation = ‘tanh’
6	Convolutional	Filters: 500, kernel_size: (1x1), activation = ‘tanh’
7	Flatten	input: (b,500,1), output: (b, 500)

dimana m adalah jumlah data yang terdapat dalam dataset. Untuk penelitian ini digunakan dataset MNIST, sehingga nilai m akan menjadi 60.000. Pada penelitian ini digunakan sebuah layer *flatten*, yang adalah sebuah layer untuk melakukan perubahan struktur data menjadi berukuran (m,x). Dari rancangan tersebut, maka kode yang digunakan dalam *python* adalah sebagai berikut:

```
# Create the model for layers
# Dense network, a special case of convolutional network
dense2 = tf.keras.layers.Conv2D(d2, kernel_size=(28,28), padding='valid',
    input_shape=input_shape, activation='tanh', name='Dense2')(x)
# Dense network, a special case of convolutional network
```

```
dense3 = tf.keras.layers.Conv2D(d3, kernel_size=(1,1), padding='valid',
    input_shape=input_shape, activation='tanh', name='Dense3')(dense2)
# Dense network, a special case of convolutional network
dense4 = tf.keras.layers.Conv2D(d4, kernel_size=(1,1), padding='valid',
    input_shape=input_shape, activation='tanh', name='Dense4')(dense3)
# Dense network, a special case of convolutional network
dense5 = tf.keras.layers.Conv2D(d5, kernel_size=(1,1), padding='valid',
    input_shape=input_shape, activation='tanh', name='Dense5')(dense4)
# Dense network, a special case of convolutional network
dense6 = tf.keras.layers.Conv2D(d6, kernel_size=(1,1), padding='valid',
    input_shape=input_shape, activation='tanh', name='Dense6')(dense5)
# Flattening, which changes shape from (?,1,1,500) to (?,500)
flat1  = tf.keras.layers.Flatten()(dense6)
# Output logits
logits = tf.keras.layers.Dense(10)(flat1)
```

Dan sesuai dengan tahapan ke 5 dari penelitian, setelah dilakukan replikasi terhadap *CiresanL6*, akan dilakukan pelatihan ulang terhadap jaringan dengan menggunakan database MNIST. Dari proses pelatihan ulang ini didapatkan data *baseline* mengenai kinerja dari network LeNet5 sebagaimana Tabel 5.4.

Tabel 5. 4 Kinerja Awal Network CiresanL6

Deskripsi	Nilai
Akurasi Pelatihan	97,42%
Akurasi Validasi	96.56%
Kebutuhan FLOPS	23.9 MFLOPS

5.3 Proses Optimasi Dengan MorphNet

Proses selanjutnya mengikuti tahapan keenam dari tahapan yang telah dirancang, yaitu melakukan optimasi dengan *MorphNet*. Proses optimasi dengan metode MorphNet menggunakan metode pengecilan dengan menggunakan *l1 regularizer* yang menyasar grup dari parameter dalam melakukan regularisasi. Dalam penelitian ini digunakan *l1 regularizer* group LASSO, sehingga, pada kedua

network ditambahkan fungsi regularisasi saat perhitungan *loss* dalam proses pembelajaran menggunakan *Stochastic Gradient Descent (SGD)*. Sasaran optimasi pada penelitian ini adalah FLOPS, sehingga perhitungan regularisasi akan didasarkan kebutuhan FLOPS tiap parameter dalam operasi. Kode *python* yang digunakan adalah sebagai berikut:

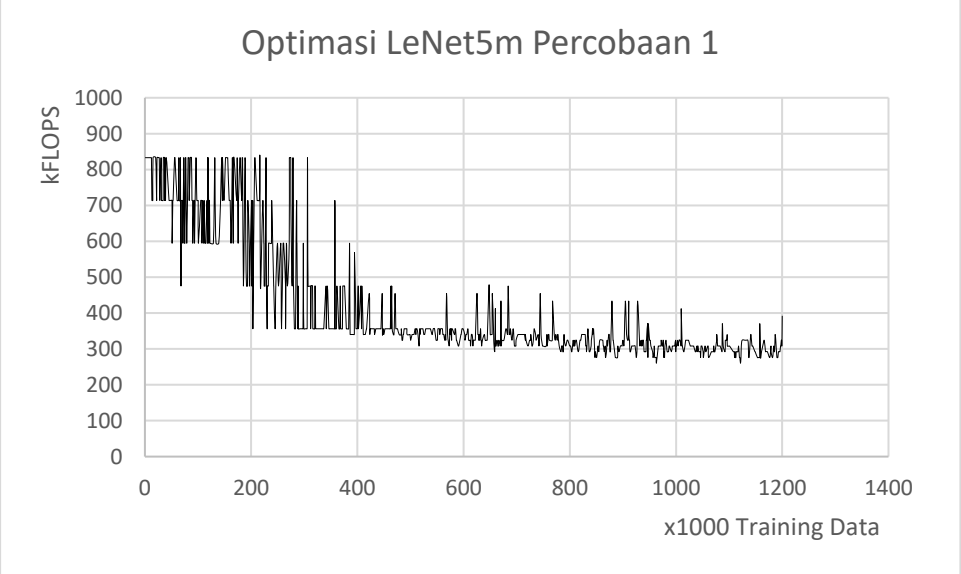
```
# Prepare morphnet regularizer
network_regularizer = flop_regularizer.GroupLassoFlopsRegularizer(
    output_boundary=[logits.op],
    input_boundary=[x.op, y_.op],
    threshold=1e-3  # Determine which neuron group can be eliminated
)
regularization_strength = reg_strength
regularizer_loss = (network_regularizer.get_regularization_term() *
    regularization_strength)

# Prepare structure exporter, to save proposed structure at specified intervals
exporter = structure_exporter.StructureExporter(
   network_regularizer.op_regularizer_manager)
#
model_loss = tf.nn.softmax_cross_entropy_with_logits(labels = y_, logits = logits,
    name='Model_Loss')
loss_operation = tf.reduce_mean(model_loss) # Mean of loss, max is 1
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
training_operation = optimizer.minimize(loss_operation + regularizer_loss)
# Define training and evaluation ops
correct_prediction = tf.equal(tf.argmax(logits,1), tf.argmax(y_,1),
    name='Correct_prediction')
accuracy_operation = tf.reduce_mean(tf.cast(correct_prediction, tf.float32),
    name='Accuracy')
```

5.3.1 Pengerdilan *network*

Pengerdilan *network* dilakukan dengan melakukan pelatihan menggunakan algoritma *Stochastic Gradient Descent*, yang melakukan perhitungan untuk minimalisir nilai *loss* dari model. Dengan penambahan metode *MorphNet*, proses pembelajaran tidak lagi hanya memperkecil *loss* yang diakibatkan kesalahan prediksi, namun juga meminimalisir *loss* yang ditimbulkan oleh ukuran *network*, yang secara langsung akan terkait dengan penggunaan sumber daya FLOPS yang akan diminimalisir. Pelatihan dilakukan menggunakan database MNIST *training set*. Dalam proses pelatihan ini tidak dilakukan validasi terhadap hasil pelatihan, karena tujuan utama dari proses pengerdilan ini adalah mengidentifikasi struktur yang lebih baik secara penggunaan sumberdaya.

Pengaturan pertukaran antara *loss* akurasi, dengan *loss* yang diakibatkan oleh kebutuhan FLOPS dapat diatur dengan mengatur λ atau *regularization_strength* yang digunakan. Dalam pendekatan MorphNet, nilai awal dari λ dihitung dari 1/FLOPS. Maka untuk *network LeNet5m*, λ awal yang digunakan adalah $1{,}2.10^{-6}$, sedangkan *network CiresanL6* akan menggunakan λ awal $4{,}18.10^{-8}$.



Gambar 5. 1 Percobaan Optimasi LeNet5m

Pengerdilan network LeNet5 menggunakan nilai $\lambda = 1{,}2\text{x}10^{-6}$ berhasil menekan jumlah FLOPS, seperti ditampilkan pada Gambar 5.1. Sumbu x menyatakan jumlah *training data* yang telah dilakukan, sementara sumbu y menyatakan kebutuhan sumberdaya dalam kilo FLOPS. Dari Gambar 5.1 terlihat bahwa seiring dengan kemajuan pembelajaran terjadi penurunan pada nilai kebutuhan FLOPS. Hal ini

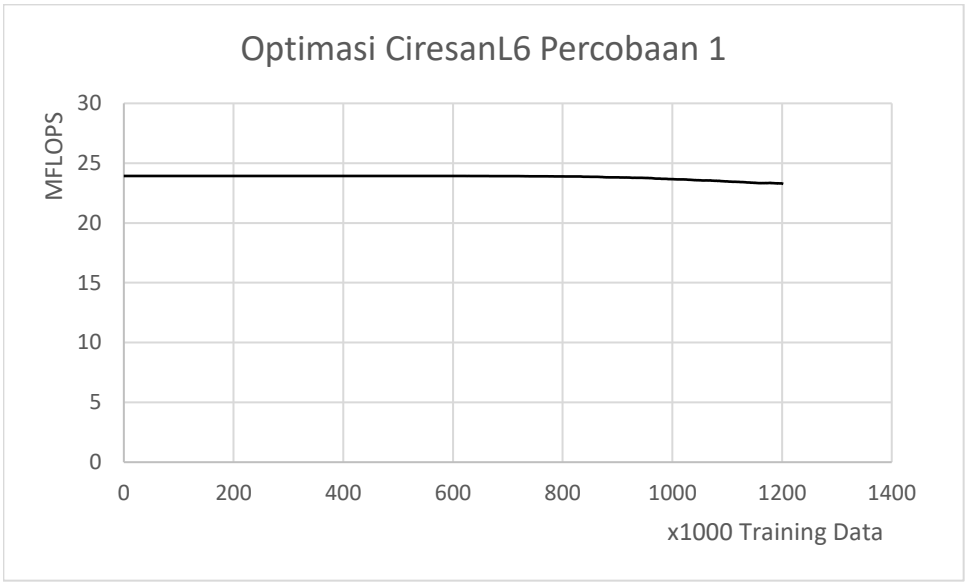
diakibatkan oleh terdapatnya *filter-filter* yang nilai-nilai unsurnya mengecil hingga lebih kecil dari ambang batas 0.001, akibat proses regularisasi L1. Dengan kecilnya nilai unsurnya, filter-filter tersebut menjadi dianggap *nonaktif* karena relatif tidak memiliki kontribusi terhadap akurasi dari *network* dalam melakukan prediksi. Kemudian, dalam proses pengusulan, akan diusulkan struktur *network* baru yang tidak menyertakan *filter* tersebut.

Dalam proses pengerdilan ini, dihasilkan beberapa usulan *network* baru. Untuk melihat kinerja *network* pada berbagai tingkat pengerdilan, maka dipilih beberapa network pada tingkat penggunaan FLOPS yang berbeda. Struktur yang dipilih ditampilkan pada Tabel 5.5, dan dijabarkan pula jumlah filter konvolusional yang digunakan. Pada kegiatan ini, sebuah layer yang terhubung penuh, diimplementasikan sebagai sebuah layer konvolusional.

Tabel 5. 5 Kandidat Struktur Hasil Optimasi LeNet5m

Nama	FLOPS	Filter Konvolusional			
		C1	C3	C5	C6
LeNet5m11	714 kFLOPS	5	16	120	84
LeNet5m12	595 kFLOPS	4	16	120	84
LeNet5m13	475 kFLOPS	3	16	120	84
LeNet5m14	308 kFLOPS	2	13	120	84

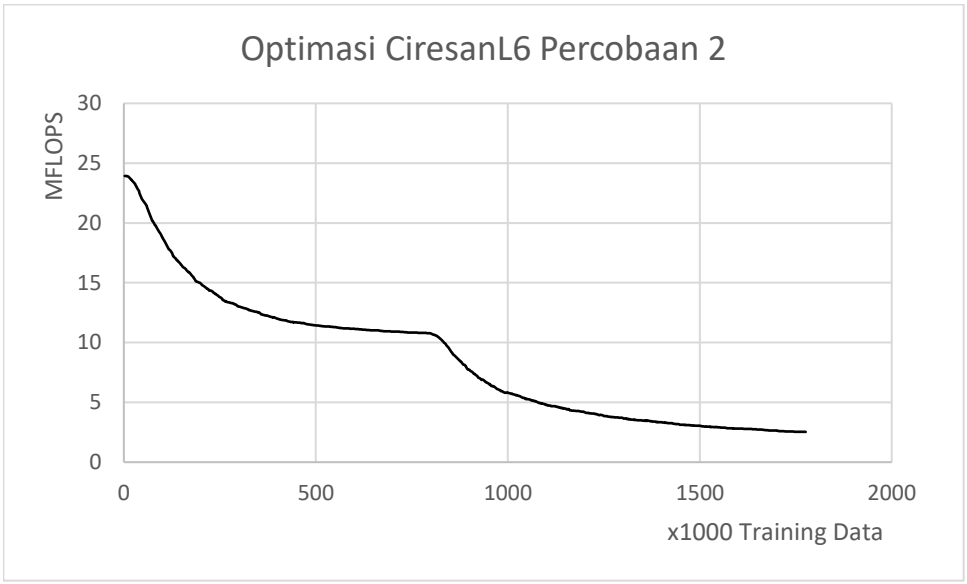
Kemudian, dilakukan pengerdilan terhadap *network CiresanL6* dengan menggunakan λ awal bernilai $4{,}18 \times 10^{-8}$. Nilai ini ternyata gagal menekan nilai FLOPS. Hal ini disebabkan karena pada perhitungan fungsi *loss* saat pelatihan, dengan banyaknya jumlah parameter, nilai regularisasi menjadi terlalu kecil, sehingga tidak dapat menekan nilai parameter untuk mencapai ambang batas dimana grup parameter dianggap nonaktif. Hal ini ditunjukkan pada Gambar 5.2



Gambar 5. 2 Percobaan Optimasi ke-1 CiresanL6

Pada Gambar 5.2, ditunjukkan mengenai kemajuan pengerdilan jaringan, seiring dengan bertambahnya data pembelajaran yang telah digunakan. Sumbu x adalah jumlah data pembelajaran yang telah digunakan (dalam ribuan), dan sumbu y merupakan kebutuhan sumber daya FLOPS *network*. Dari Gambar 5.2, ditunjukkan bahwa tidak terdapat penurunan yang berarti dari kebutuhan FLOPS seiring dengan pembelajaran. Ini berarti bahwa nilai λ atau *regularization_strength* yang digunakan terlalu kecil, sehingga pada saat proses pembelajaran, gradien penurunan nilai parameter (*weight, bias)*, menjadi terlalu lambat mengecil hingga kurang dari nilai ambang batas 0.001.

Solusi untuk permasalahan ini adalah dengan memperbesar nilai ambang batas nonaktif, atau memperbesar nilai λ. Untuk menjaga nilai ambang batas nonaktif yang sama dengan *LeNet5m*, maka dilakukan pembesaran terhadap nilai λ menjadi $4{,}18 \times 10^{-6}$. Dengan nilai baru ini, proses pengerdilan berhasil dilakukan, dimana kemajuan pengerdilan ditunjukkan pada Gambar 5.3. Pembesaran nilai λ berhasil mempercepat laju pengerdilan ukuran network.



Gambar 5. 3 Percobaan Optimasi ke-2 CiresanL6

Pada Gambar 5.3, ditunjukkan mengenai kemajuan pengerdilan jaringan, seiring dengan bertambahnya data pembelajaran yang telah digunakan. Sumbu x adalah jumlah data pembelajaran yang telah digunakan (dalam ribuan), dan sumbu y merupakan kebutuhan sumber daya FLOPS *network*. Dari Gambar 5.3, ditunjukkan terjadi penurunan FLOPS seiring dengan berjalannya proses pembelajaran, dan hampir mencapai nilai nol pada akhir pembelajaran. Menurunnya jumlah kebutuhan FLOPS ini dikarenakan *filter-filter* yang nonaktif karena nilainya menjadi lebih kecil dari ambang batas.

Dari proses pengerdilan ini, dihasilkan beberapa struktur usulan. Dari beberapa struktur usulan ini, dipilih beberapa kandidat *network* yang mewakili beberapa tingkatan penyusutan. Tabel 5.6 menunjukkan lima buah kandidat struktur baru yang dipilih untuk dilakukan pelatihan ulang dan dianalisa kinerjanya.

Tabel 5. 6 Kandidat Struktur Hasil Optimasi CiresanL6

Nama	FLOPS	Filter Konvolusional				
		C1	C2	C3	C4	C5
CiresanL6m1	20.4 MFLOPS	1867	2000	1500	1000	500
CiresanL6m2	12.4 MFLOPS	436	2000	1500	1000	500
CiresanL6m3	10.8 MFLOPS	147	2000	1500	1000	500
CiresanL6m4	6.1 MFLOPS	106	2000	754	1000	500
CiresanL6m5	2.8 MFLOPS	70	2000	237	989	500

5.3.2 Pelatihan Ulang *Network* Usulan

Setelah didapatkan beberapa *network* usulan, tahapan selanjutnya yang dilakukan adalah dengan melakukan pelatihan ulang terhadap kandidat yang telah diperoleh. Pelatihan dilakukan dengan menggunakan database MNIST seperti sebelumnya. Pelatihan dilakukan dengan menggunakan data *training set* dari database MNIST, dan validasi dilakukan dengan menggunakan data *test set* dari database MNIST. Pelatihan dilakukan dalam 2 *epochs* pelatihan. Hasil pelatihan dan pengujian terhadap usulan struktur baru *network LeNet5m* tertuang dalam Tabel 5.7.

Tabel 5. 7 Hasil Pelatihan Terhadap Network Usulan LeNet5m

Nama	Filter Konvolusional				Kinerja		
	C1	C2	C3	C4	Akurasi	Akurasi Validasi	FLOPS
LeNet5m11	5	16	120	84	99.64%	98.54%	713 k
LeNet5m12	4	16	120	84	99.72%	98.68%	594 k
LeNet5m13	3	16	120	84	99.62%	98.56%	475 k
LeNet5m14	2	13	120	84	99.73%	98.40%	308 k
LeNet5m15	2	10	120	84	99.73%	98.48%	260 k

Hasil pelatihan terhadap *network* usulan *LeNet5m* menghasilkan kinerja akurasi pada rentang yang sama walau kebutuhan FLOPS relatif ditekan cukup jauh. Setelah itu, dilakukan pula hal yang sama terhadap *network* usulan *CiresanL6*. Hasil pelatihan dan pengujian terhadap *network* tersebut ditunjukkan kedalam Tabel 5.8.

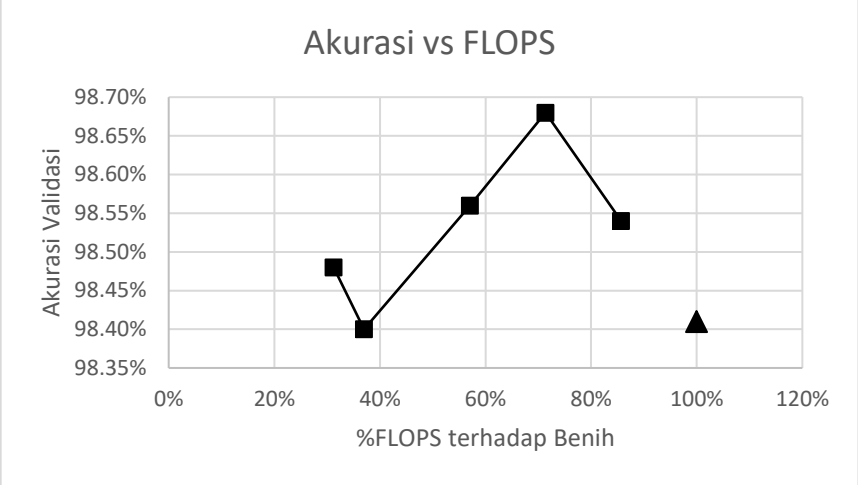
Tabel 5. 8 Hasil Pelatihan Terhadap Network Usulan CiresanL6

Nama	Filter Konvolusional					Kinerja		
	C1	C2	C3	C4	C5	Acc	Val_Acc	FLOPS
CiresanL6m1	1867	2000	1500	1000	500	97.84%	96.52%	20.4 M
CiresanL6m2	436	2000	1500	1000	500	98.28%	97.27%	12.4 M
CiresanL6m3	147	2000	1500	1000	500	98.23%	96.97%	10.8 M
CiresanL6m4	106	2000	754	1000	500	98.06%	95.81%	6.1 M
CiresanL6m5	70	2000	237	989	500	97.08%	96.19%	2.8 M

Hasil pelatihan terhadap network usulan *CiresanL6* menghasilkan kinerja akurasi yang relatif mirip, namun memiliki kecenderungan menurun. Dari sisi penggunaan FLOPS, terjadi penurunan yang signifikan dibanding *network* asli.

5.3.3 Analisis Hasil Optimasi

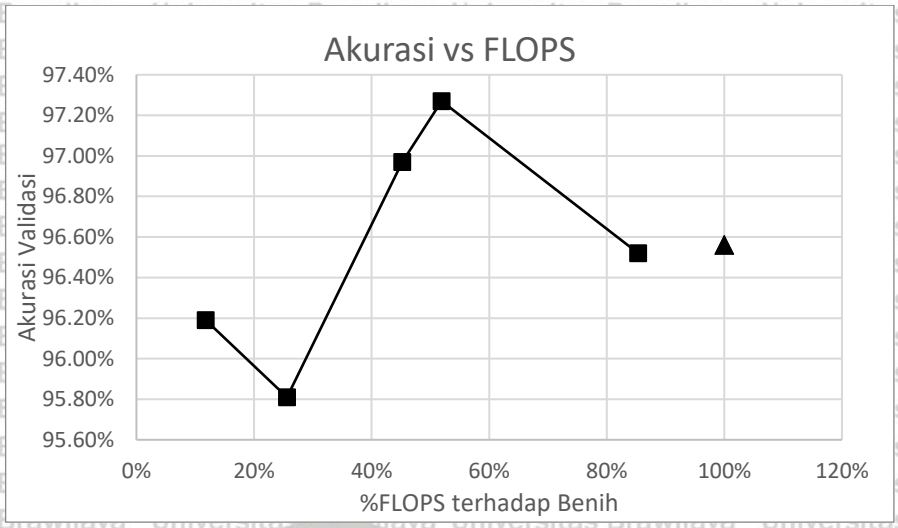
Hasil pengukuran kinerja *network* hasil optimasi, kemudian dibandingkan terhadap *network* awal. Kinerja *network* awal dan hasil optimasi untuk *network* *LeNet5m* ditampilkan dalam Gambar 5.4.



Gambar 5. 4 Grafik Kinerja Akurasi dan FLOPS network LeNet5m Terhadap Usulan

Pada Gambar 5.4, sumbu x menyatakan prosentase kebutuhan FLOPS dari *network* terhadap *network* benih, sementara sumbu y menyatakan akurasi validasi. Titik segitiga adalah titik nilai kinerja dari *network* LeNet5 awal, sementara titik-titik kotak adalah kinerja dari *network* usulan hasil optimasi. *Network* dapat disebut sebagai lebih optimal dibandingkan patokan, apabila berada pada posisi lebih kiri dan lebih atas dibandingkan *network* asli.

Dari Gambar 5.4, dapat dilihat bahwa optimasi terhadap *LeNet5m* menghasilkan beberapa *network* usulan yang berada lebih kiri dan lebih atas dibandingkan *network* benih. Ini menunjukkan bahwa *network* usulan tersebut selain lebih efisien dalam penggunaan FLOPS, juga memiliki kinerja akurasi prediksi yang lebih baik. Dari 5 *network* usulan terpilih, didapati 4 *network* yang memiliki kinerja akurasi prediksi lebih baik, namun memiliki kebutuhan *FLOPS* yang lebih rendah dibandingkan *network* awal.



Gambar 5. 5 Grafik Kinerja Akurasi dan FLOPS network CiresanL6 Terhadap Usulan

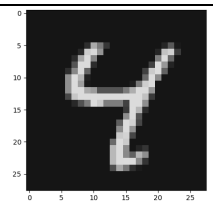
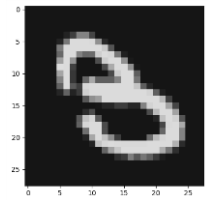
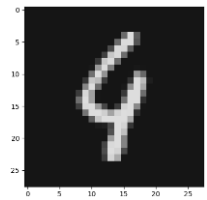
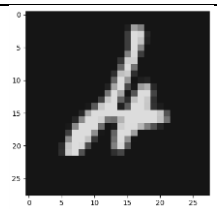
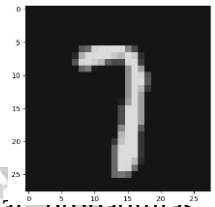
Pada Gambar 5.5, sumbu x menyatakan prosentase kebutuhan FLOPS terhadap *network* benih, sementara sumbu y menyatakan akurasi validasi. Titik segitiga adalah titik nilai kinerja dari *network* CiresanL6 yang asli, sementara titik-titik kotak adalah kinerja dari *network* usulan hasil optimasi. *Network* dapat disebut sebagai lebih optimal dibandingkan patokan, apabila berada pada posisi lebih kiri dan lebih atas dibandingkan *network* asli.

Dari Gambar 5.5 didapati beberapa *network* usulan berada lebih kiri dan lebih atas dibandingkan *network* benih. Hal ini menunjukkan bahwa *network* usulan tersebut selain lebih efisien dalam penggunaan FLOPS, juga memiliki kinerja yang lebih baik. Dari 5 *network* usulan yang telah dipilih, didapati 2 *network* yang memiliki kinerja akurasi yang lebih baik dan penggunaan kebutuhan *FLOPS* yang lebih rendah dibandingkan *network* awal.

5.3.3 Analisa Hasil Prediksi

Dari hasil pelatihan ulang, perlu dilihat hasil klasifikasi pada beberapa data yang mengalami salah prediksi. Dari beberapa data yang mengalami salah prediksi oleh *network* baru, beserta nilai probabilitas luarannya, ditampilkan pada Gambar 5.9.

Tabel 5. 9 Prediksi Salah

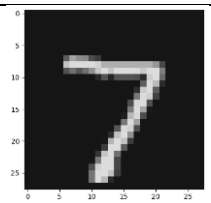
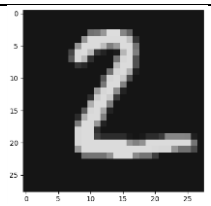
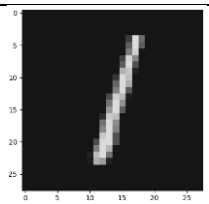
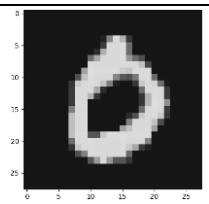
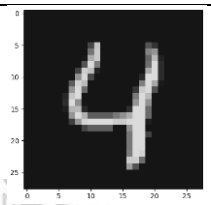
Gambar	Prediksi	Sebenarnya	Analisa
	8	4	Nilai probabilitas yang dihasilkan network: 4: 0.49 8: 0.51
	8	3	Nilai probabilitas yang dihasilkan network: 3: 0.00 8: 0.99
	6	4	Nilai probabilitas yang dihasilkan network: 4: 0.26 6: 0.70
	2	4	Nilai probabilitas yang dihasilkan network: 2: 0.79 4: 0 .03
	2	7	Nilai probabilitas yang dihasilkan network: 2: 0.57

Nilai probabilitas pada Tabel 5.9 menunjukkan nilai probabilitas yang dihasilkan oleh luaran *neural network*. Nilai probabilitas ini menunjukkan bahwa pada beberapa kasus, perbedaan nilai probabilitas kelas hasil prediksi, dan kelas sebenarnya tidak terlalu signifikan. Namun pada beberapa kasus, nilai perbandingan probabilitasnya cukup besar karena beberapa ciri dari salah klasifikasinya dimiliki pula oleh kelas sebenarnya.

Selanjutnya, perlu dilihat beberapa hasil prediksi yang berhasil dianalisa dengan benar, untuk memastikan sistem ini melakukan prediksinya dengan tepat.

Hasil prediksi oleh sistem yang benar tersebut ditampilkan pada Gambar 5.10.

Tabel 5. 10 Prediksi Benar

Gambar	Prediksi	Sebenarnya	Analisa
	7	7	Nilai probabilitas yang dihasilkan network: 7: 1.00
	2	2	Nilai probabilitas yang dihasilkan network: 2: 0.99
	1	1	Nilai probabilitas yang dihasilkan network: 1: 0.99
	0	0	Nilai probabilitas yang dihasilkan network: 0: 0.99
	4	4	Nilai probabilitas yang dihasilkan network: 4: 0.99

Dari Tabel 5.10 dapat dilihat bahwa sistem dapat secara meyakinkan menentukan probabilitas dari angka-angka tersebut secara meyakinkan, yaitu pada probabilitas 0.99 hingga 1. Apabila direferensi ulang dengan hasil salah klasifikasi, beberapa kasus terjadi ketika tulisan tangan yang dideteksi menghasilkan keraguan, bahkan ketika diamati oleh manusia.

5.4 Pengujian *Network* Pada Komputer Berbeda

Sesuai tahapan penelitian, untuk memastikan bahwa pengurangan FLOPS benar-benar memiliki dampak terhadap waktu inferensi yang diperlukan, maka perlu dilakukan sebuah pengujian untuk mengukur waktu eksekusi yang diperlukan

masing-masing *network* hasil optimasi. Pengujian pada komputer dengan GPU, dilakukan pada sebuah komputer dengan 4 *core* CPU dan 1 GPU, sementara pengujian pada komputer tanpa GPU dilakukan pada sebuah komputer virtual pada *cloud*, dengan 12 *core* CPU. Hasil pengukuran ini kemudian dibandingkan dengan waktu eksekusi dari *network* benih. Hasil pengukuran waktu inferensi untuk *network* hasil optimasi *LeNet5* adalah sebagai berikut:

Tabel 5. 11 Waktu Inferensi Per Sample LeNet5m

Nama	FLOPS ($x10^3$)	Waktu Inferensi (µs)	
		CPU dengan GPU	CPU tanpa GPU
LeNet5m	833	165	204
LeNet5m11	713	154	187
LeNet5m12	594	135	181
LeNet5m13	475	130	174
LeNet5m14	308	132	171
LeNet5m15	260	127	171

Sementara itu, waktu inferensi untuk *network* hasil optimasi dari *CiresanL6* adalah sebagai berikut:

Tabel 5. 12 Waktu Inferensi Per Sample CiresanL6

Nama	FLOPS ($x10^6$)	Waktu Inferensi (µs)	
		CPU dengan GPU	CPU tanpa GPU
CiresanL6	23.9	294	655
CiresanL6m1	20.4	289	646
CiresanL6m2	12.4	214	454
CiresanL6m3	10.8	206	419
CiresanL6m4	6.1	152	258
CiresanL6m5	2.8	121	152

Dari kedua hasil tersebut, didapatkan bahwa penurunan FLOPS mengurangi kebutuhan waktu eksekusi, namun tidak menguranginya secara proporsional. Hal ini mungkin disebabkan oleh terjadinya *bottleneck* pada proses diluar proses kalkulasi. Peningkatan kinerja terbesar terjadi pada *network* CiresanL6, pada komputer yang hanya menggunakan CPU untuk kalkulasi.

BAB VI
KESIMPULAN

Penelitian telah selesai dilakukan, dan optimasi dengan menggunakan *MorphNet* telah menghasilkan *network* usulan yang telah dibandingkan kinerjanya. Dari proses optimasi yang telah dilakukan, disimpulkan:

1. Optimasi terhadap *network CiresanL6* dan *LeNet5* berhasil dilakukan dengan metode MorphNet, dimana *neuron* yang nonaktif berhasil diseleksi dan dibuang dari struktur usulan.
2. Pengendalian kecepatan dan imbal hasil pengerdilan *network* diatur dengan mengendalikan nilai skala regularisasi λ. Proses pengerdilan dimulai dengan menggunakan skala regularisasi 1/Nilai FLOPS Dilakukan pembesaran terhadap nilai λ apabila proses optimasi gagal mengerdilkan *network*.
3. Hasil analisis kinerja *network* usulan hasil optimasi adalah pada *LeNet5m* terjadi pengurangan kebutuhan FLOPS hingga 69%, dan peningkatan akurasi sebesar 0,27%. Optimasi pada *CiresanL6* berhasil mengurangi kebutuhan FLOPS hingga 88%, dan peningkatan akurasi 0.71%. Ketika diuji pada komputer yang berbeda, pada struktur usulan baru LeNet5 dihasilkan peningkatan kecepatan eksekusi dalam rentang 8 – 23%, sementara optimasi CiresanL6 menghasilkan peningkatan kecepatan eksekusi dalam rentang 1 – 77%.

DAFTAR PUSTAKA

Alvarez, J. M., & Salzmann, M. (2016). Learning the Number of Neurons in Deep Networks. *Conference on Neural Information Processing Systems.* Barcelona.

Bottou, L., & Bousquet, O. (2012). The Tradeoffs of Large Scale Learning. Cambridge: MIT Press.

Ciresan, D., Meier, U., & Schmidhuber, J. (2012). *Multi-column Deep Neural Networks for Image Classification.* Manno: IDSIA.

Collins, M. D., & Kohli, P. (2014). Memory Bounded Deep Convolutional Networks. *arXiv preprint arXiv:1412.1442*.

Gordon, A., Eban, E., & Nachum, O. (2018). MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, (pp. 1586-1595). Salt Lake City, UT.

LeCun, Y., Bottou, L., & Bengio, Y. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceeding of the IEEE*, (pp. 1-46).

Russel, S., & Norvig, P. (2010). *Artificial Intelligence a Modern Approach Third Ed.* New Jersey: Prentice Hall.

Suyanto. (2014). *Artificial Intelligence: Searching, Reasoning, Planning and Learning.* Bandung: Informatika.

Yegulalp, S. (2019, Jun 18). *What is TensorFlow? The machine learning library explained.* Retrieved from Info World: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

Zoph, B., & Le, Q. (2017). Neural Architecture Search with Reinforcement Learning. *ArXiv*, 1-15.

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. (2018). Learning Transferable Architectures for Scalable Image Recognition. *ArXiv*, 1-14.