ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR – KI141502

IMPLEMENTASI *CONVOLUTIONAL NEURAL NETWORK* UNTUK KLASIFIKASI OBYEK PADA CITRA

SINDUNURAGA RIKARNO PUTRA
NRP 5111100076

Dosen Pembimbing I
Dr. Chastine Fatichah, S.Kom., M.Kom.

Dosen Pembimbing II
Dr. Nanik Suciati, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2015

UNDERGRADUATE THESES – KI141502

IMPLEMENTATION OF CONVOLUTIONAL NEURAL NETWORK FOR THE CLASSIFICATION OF OBJECT IN IMAGES

SINDUNURAGA RIKARNO PUTRA
NRP 5111100076

Supervisor I
Dr. Chastine Fatichah, S.Kom., M.Kom.

Supervisor II
Dr. Nanik Suciati, S.Kom., M.Kom.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2015

LEMBAR PENGESAHAN

IMPLEMENTASI *CONVOLUTIONAL NEURAL NETWORK* UNTUK KLASIFIKASI OBYEK PADA CITRA

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Cerdas Visual
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh
SINDUNURAGA RIKARNO PUTRA
NRP : 5111 100 076

Disetujui oleh Dosen Pembimbing Tugas Akhir:

1. Dr. Chastine Fatichah, S.Kom., M.Kom. .....
   NIP: 19751220 200112 2 002 (Pembimbing 1)

2. Dr. Nanik Suciati, S.Kom., M.Kom. .....
   NIP: 19710428 199412 2 001 (Pembimbing 2)

SURABAYA
JUNI, 2015

IMPLEMENTASI *CONVOLUTIONAL NEURAL NETWORK* UNTUK KLASIFIKASI OBYEK PADA CITRA

Nama Mahasiswa	**: SINDUNURAGA RIKARNO PUTRA**
NRP	**: 5111100076**
Jurusan	**: Teknik Informatika FTIF-ITS**
Dosen Pembimbing 1	**: Dr. Chastine Fatichah, S.Kom., M.Kom.**
Dosen Pembimbing 2	**: Dr. Nanik Suciati, S.Kom., M.Kom.**

ABSTRAK

Deep Learning adalah sebuah bidang keilmuan baru dalam bidang Machine Learning yang akhir-akhir ini berkembang karena perkembangan teknologi GPU acceleration. Deep learning memiliki kemampuan yang sangat baik dalam visi komputer. Salah satunya adalah pada kasus klasifikasi obyek pada citra yang telah lama menjadi problem yang sulit diselesaikan.

Tugas akhir ini mengimplementasikan salah satu jenis model Deep Learning yang memiliki kemampuan yang baik dalam klasifikasi data dengan struktur dua dimensi seperti citra, yaitu Convolutional Neural Network (CNN). Digunakan dataset CIFAR-10 yang telah lama menjadi benchmark dalam kasus klasifikasi citra. Model CNN akan dikembangkan menggunakan library Theano yang memiliki kemampuan baik dalam memanfaatkan GPU acceleration.

Dalam penyusunan model, dilakukan optimasi hyperparameter jaringan dan analisa penggunaan memori untuk mendapatkan intuisi yang lebih baik terhadap perilaku model CNN. Tugas akhir ini membandingkan tiga arsitektur CNN, yaitu DeepCNet, NagadomiNet, dan NetworkInNetwork, dengan kedalaman convolution layer maksimal lima layer pada setiap arsitektur.

Dari hasil uji coba, didapatkan nilai error klasifikasi terkecil yaitu 17.69% dengan menggunakan arsitektur NagadomiNet yang terdiri dari convolution layer dengan ukuran kernel 3x3, penggunaan Global Average Pooling sebelum Softmax Layer, serta implementasi inverted dropout dengan nilai drop rate incremental sebesar 0.1, 0.2, 0.3, 0.4, dan 0.5 pada masing-masing convolution layer.

Kata kunci: Convolutional Neural Network, Deep Learning, GPGPU, Klasifikasi Citra, CIFAR

IMPLEMENTATION OF CONVOLUTIONAL NEURAL NETWORK FOR THE CLASSIFICATION OF OBJECT IN IMAGES

Student's Name	**: SINDUNURAGA RIKARNO PUTRA**
Student's ID	**: 5111100076**
Department	**: Teknik Informatika FTIF-ITS**
First Advisor	**: Dr. Chastine Fatichah, S.Kom., M.Kom.**
Second Advisor	**: Dr. Nanik Suciati, S.Kom., M.Kom.**

ABSTRACT

Deep Learning is a new branch of knowledge in the field of Machine Learning that in the past few years has developed due to the improvement in GPU Acceleration technologies. Deep Learning has great capabilities in the field of computer vision. One of its capabilities is in the case of object classification in images, a problem that has been unsolved for a long time.

This final project will implement a form of Deep Learning that is designed for the processing of two dimensional structured data such as images, which is called The Convolutional Neural Network (CNN). The CIFAR-10 dataset is used because it has been a classic benchmark for the case of image classification. The CNN model is implemented using the pyhton Theano Library because it is designed for use with GPU Acceleration.

In the creation of the model, hyperparameter tuning and memory usage analysis will also be done to get a better intuition on the characteristics of CNN. In this final project, three CNN model architectures is compared, which respectively is DeepCNet, NagadomiNet, and Network in Network, each with a maximum convoutional layer depth of 5.

From the experiment, the lowest classification error gotten is 17.69% from a model that uses the NagadomiNet architecture which consists of convolutional layers with a kernel of size 3x3, the usage of Global Average Pooling before the Softmax Layer, and also an implementation of inverted dropout using an incremental drop rate with a value of 0.1, 0.2, 0.3, 0.4, and 0.5 on each convolutional layers.

Keywords: Convolutional Neural Network, Deep Learning, GPGPU, Image Classification, CIFAR

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin. Segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **"IMPLEMENTASI *CONVOLUTIONAL NEURAL NETWORK* UNTUK KLASIFIKASI OBYEK PADA CITRA".**

Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis. Dengan pengerjaan Tugas Akhir ini, penulis dapat belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Jurusan Teknik Informatika ITS. Dengan Tugas Akhir ini, penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari.

Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT dan Nabi Muhammad SAW.
2. Papa, Mama, Mas Dika dan Dek Lita yang selalu memberi semangat dan motivasi kepada penulis dalam menjalankan perkuliahan meskipun terpisah oleh jarak.
3. Ibu Dr. Chastine Fatichah, S.Kom, M.Kom. selaku pembimbing I yang telah membantu, membimbing, dan memotivasi penulis dalam mendalami bidang *Machine Learning* selama menjalankan perkuliahan di ITS, sehingga penulis memiliki pemahaman yang cukup untuk menyelesaikan tugas akhir ini.
4. Ibu Dr. Nanik Suciati, S.Kom., M.Kom. selaku Kepala Jurusan Teknik Informatika ITS, serta pembimbing II yang telah membimbing penulis dalam menjalankan perkuliahan di Teknik Informatika ITS, serta membina penulis dalam menyelesaukan tugas akhir ini.

5. Bapak Dwi Sunaryono, S.Kom., M.Kom. selaku dosen wali yang telah banyak membimbing dan membantu penulis dalam menjalankan kegiatan perkuliahan di Teknik Informatika ITS dari semenjak mahasiswa baru hingga mahasiswa tingkat akhir.
6. Bapak Radityo Anggoro, S.Kom., M.Sc. selaku koordinator Tugas Akhir, dan segenap dosen Teknik Informatika yang telah memberikan ilmunya.
7. Teman-teman mahasiswa Teknik Informatika angkatan 2011 yang yang telah membantu, berbagi ilmu, menjaga kebersamaan, dan memberi motivasi kepada penulis.
8. Teman-teman keluarga besar Jamaah Masjid Manarul Ilmi ITS yang telah banyak membina penulis dalam keseharian sehingga menjadi insan yang lebih baik.
9. Teman-teman Keluarga Muslim Informatika ITS yang telah menjaga penulis dalam lingkungan yang kondusif dalam kebermanfaatan dan ukhuwah.
10. Serta semua pihak yang yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Juni 2015

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR KODE

BAB I
PENDAHULUAN

1.1 Latar Belakang

Salah satu *problem* dalam visi komputer yang telah lama dicari solusinya adalah klasifikasi obyek pada citra secara umum. Bagaimana menduplikasi kemampuan manusia dalam memahami informasi citra, agar komputar dapat mengenali obyek pada citra selayaknya manusia. Proses *feature engineering* yang digunakan pada umumnya sangat terbatas dimana hanya dapat berlaku pada dataset tertentu saja tanpa kemampuan generalisasi pada jenis citra apapun. Hal tersebut dikarenakan berbagai perbedaan antar citra antara lain perbedaan sudut pandang, perbedaan skala, perbedaan kondisi pencahayaan, deformasi obyek, dan sebagainya.

Kalangan akademisi telah lama bergelut pada problem ini. Salah satu pendekatan yang berhasil digunakan adalah menggunakan Jaringan Saraf Tiruan (JST) yang terinspirasi dari jaringan saraf pada manusia. Konsep tersebut kemudian dikembangkan lebih lanjut dalam *Deep Learning*.

Pada tahun 1989, Yann LeCun dan teman teman berhasil melakukan klasifikasi citra kode zip menggunakan kasus khusus dari *Feed Forward Neural Network* dengan nama *Convolutional Neural Network* (CNN) [1]. Karena keterbatasan perangkat keras, *Deep Learning* tidak dikembangkan lebih lanjut hingga pada tahun 2009 dimana Jurgen mengembangkan sebuah *Recurrent Neural Network* (RNN) yang mendapatkan hasil signifikan pada pengenalan tulisan tangan [2]. Semenjak itu, dengan berkembangnya komputasi pada perangkat keras *Graphical Processing Unit* (GPU), pengembangan DNN berjalan dengan pesat. Pada 2012, sebuah CNN dapat melakukan pengenalan citra dengan akurasi yang menyaingi manusia pada dataset tertentu [3]. Dewasa ini, *Deep Learning* telah menjadi salah satu topik hangat dalam dunia *Machine*

Learning karena kapabilitasnya yang signifikan dalam memodelkan berbagai data kompleks seperti citra dan suara.

Metode *Deep Learning* yang hingga kini memiliki hasil paling signifikan dalam pengenalan citra adalah *Convolutional Neural Network* (CNN) [4]. Hal tersebut dikarenakan CNN berusaha meniru sistem pengenalan citra pada visual cortex manusia [5] sehingga memiliki kemampuan mengolah informasi citra. Namun CNN, seperti metode *Deep Learning* lainnya, memiliki kelemahan yaitu proses pelatihan model yang lama. Dengan perkembangan perangkat keras, hal tersebut dapat diatasi menggunakan teknologi *General Purpose Graphical Processing Unit* (GPGPU).

Dalam tugas akhir ini, akan diterapkan model CNN dalam pengenalan obyek pada citra CIFAR-10. Untuk mengatasi permasalahan proses pelatihan model CNN yang memakan waktu lama, akan diterapkan *GPU Acceleration* untuk mempercepat proses pelatihan model. Diharapkan model yang dibangun dapat melakukan pengenalan obyek citra dengan akurasi tinggi, serta dengan proses pelatihan model yang cepat.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana cara membangun model *Convolutional Neural Network* untuk klasifikasi obyek pada citra?
2. Bagaimana cara mempercepat proses pembuatan model *Convolutional Neural Network*?
3. Bagaimana meminimalkan tingkat *error* yang didapatkan?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Data yang digunakan adalah dataset CIFAR-10 yang tersedia secara terbuka di internet.
2. Model klasifikasi yang digunakan adalah *Convolutional Neural Network* (CNN).
3. Kedalaman CNN maksimal adalah 5 *convolution layer*.

1.4 Tujuan

Tujuan dari tugas akhir ini adalah untuk membangun model *Convolutional Neural Network* yang dapat melakukan klaifikasi obyek pada citra.

1.5 Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir.
   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Proposal Tugas Akhir yang diajukan memiliki gagasan yang sama dengan Tugas Akhir ini, yaitu implementasi *Convolutional Neural Network* untuk melakukan klasifikasi obyek pada citra.

2. Studi literatur
   Pada tahap ini dilakukan pencarian, pembelajaran dan pemahaman informasi dan literatur yang diperlukan untuk implementasi *Convolutional Neural Network*. Informasi dan literatur didapatkan dari literatur jurnal ilmiah dan sumber-sumber informasi lain yang berhubungan.

3. Analisis dan desain perangkat lunak
   Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Tahap ini mendefinisikan alur

dari implementasi. Langkah-langkah yang dikerjakan juga didefinisikan pada tahap ini.

4. Implementasi perangkat lunak
   Implementasi merupakan tahap membangun rancangan program yang telah dibuat. Pada tahapan ini direalisasikan apa yang terdapat pada tahapan sebelumnya, sehingga menjadi sebuah program yang sesuai dengan apa yang telah direncanakan.

5. Pengujian dan evaluasi
   Pada tahapan ini dilakukan uji coba pada data yang telah dikumpulkan. Pengujian dan evaluasi akan dilakukan dengan menggunakan bahasa Python. Tahapan ini dimaksudkan untuk mengevaluasi kesesuaian data dan program serta mencari masalah yang mungkin timbul dan mengadakan perbaikan jika terdapat kesalahan.

6. Penyusunan buku Tugas Akhir.
   Pada tahapan ini disusun buku yang memuat dokumentasi mengenai pembuatan serta hasil dari implementasi perangkat lunak yang telah dibuat.

1.6 Sistematika Penulisan Laporan Tugas Akhir

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini:

Bab I Pendahuluan

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

Bab II Dasar Teori

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

Bab III Perancangan Perangkat Lunak

Bab ini berisi tentang desain sistem yang disajikan dalam bentuk diagram ataupun *pseudocode*.

Bab IV Implementasi

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa *code* yang digunakan untuk proses implementasi.

Bab V Uji Coba Dan Evaluasi

Bab ini menjelaskan kemampuan perangkat lunak dengan melakukan pengujian kebenaran dan pengujian kinerja dari sistem yang telah dibuat.

Bab VI Kesimpulan Dan Saran

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

BAB II
TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori dan materi yang berkaitan dengan algoritma yang diajukan. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

2.1 Feature Learning

Berbeda dengan *feature engineering* yang digunakan dalam *machine learning* pada umumnya, *feature learning* adalah metode dimana proses *feature extraction* dilakukan secara otomatis dan adaptif oleh model [6]. *Feature learning* muncul karena *feature engineering* sangatlah terbatas dalam artian tiap kasus data yang berbeda memerlukan *feature extraction* yang berbeda. Hal tersebut menjadikan metode *feature engineering* tidak memiliki kemampuan generalisasi pada keragaman jenis data seperti yang dibutuhkan dalam kasus klasifikasi obyek pada citra.

Pada pengolahan data kompleks seperti citra dan suara, umumnya dilakukan ekstraksi fitur untuk mengubah data menjadi bentuk yang dapat dimengerti oleh metode *learning*. Namun proses tersebut sangat memakan waktu dan cenderung kurang dapat menggambarkan keseluruhan nilai informasi dari data. *Feature learning* mengatasi hal tersebut dengan membuat proses ekstraksi fitur adaptif yang dapat melakukan penyesuaian otomatis terhadap data yang digunakan.

Dalam perkembangannya, terdapat beragam metode *feature learning* yang dikembangkan dan secara garis besar dibagi menjadi *unsupervised feature leaning* dan *supervised feature learning*. Metode *unsupervised* adalah metode *feature* learning yang berkembang terlebih dahulu dan memiliki tujuan untuk mentransformasi data menjadi representasi lain yang

lebih mudah dipahami oleh komputer. Diantaranya terdapat metode *Autoencoder, Deep Belief Network, Gaussian Mixture Models,* dan bahkan juga terdapat metode *K-Means* [7].

Pada awal perkembangannya, klasifikasi obyek pada citra menggunakan metode tersebut, namun kemudian berkembang metode feature learning yang bersifat *supervised* sehingga metode *unsupervised* mulai ditinggalkan dikarenakan memiliki performa yang lebih buruk. Metode *supervised feature learning* yang banyak berkembang adalah *deep learning* dimana digunakan sebuah *deep network* yang melakukan proses ekstraksi fitur dan klasfiikasi menggunakan *learning.*

Model *deep learning* yang umum digunakan pada pengolahan citra adalah *Convolutional Neural Network* (CNN). Untuk kasus citra, *feature learning* dilakukan oleh CNN pada serangkaian *convolution layer* di dalamnya. Karena hal tersebut, sebuah CNN tidak memerlukan proses ektraksi fitur khusus dan pada umumnya hanya memerlukan praproses dasar untuk normalisasi data.

2.2 Convolutional Neural Network

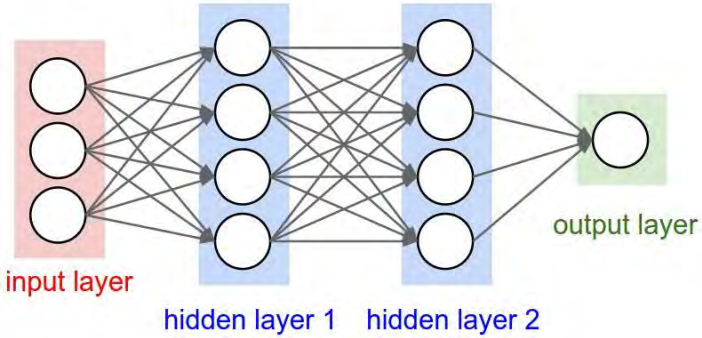
Convolutional Neural Network (CNN) adalah pengembangan dari Multilayer Perceptron (MLP) yang didesain untuk mengolah data dua dimensi. CNN termasuk dalam jenis *Deep Neural Network* karena kedalaman jaringan yang tinggi dan banyak diaplikasikan pada data citra. Pada kasus klasifikasi citra, MLP kurang cocok untuk digunakan karena tidak menyimpan informasi spasial dari data citra dan menganggap tiap piksel adalah fitur yang independen sehingga menghasilkan hasil yang kurang baik.

CNN pertama kali dikembangkan dengan nama NeoCognitron oleh Kunihiko Fukushima, seorang peneliti dari *NHK Broadcasting Science Research Laboratories*, Kinuta, Setagaya, Tokyo, Jepang [8]. Konsep tersebut kemudian dimatangkan oleh Yann LeCun, seorang peneliti dari *AT&T*

Bell Laboratories di Holmdel, New Jersey, USA. Model CNN dengan nama LeNet berhasil diterapkan oleh LeCun pada kasus pengenalan angka tulisan tangan [1]. Pada tahun 2012, Alex Krizhevsky dengan penerapan CNN miliknya berhasil menjuarai kompetisi *ImageNet Large Scale Visual Recognition Challenge* 2012. Prestasi tersebut menjadi momen pembuktian metode *Deep Learning*, khususnya CNN. Metode CNN terbukti berhasil mengungguli metode *Machine Learning* lainnya seperti SVM pada kasus klasifikasi obyek pada citra.

2.2.1 Konsep

Cara kerja CNN mirip dengan MLP, namun dalam CNN, tiap neuron direpresentasikan dalam bentuk dua dimensi, tidak seperti MLP yang tiap neuron hanya berukuran satu dimensi.



***Gambar 2.1 Arsitektur MLP Sederhana* [9]**

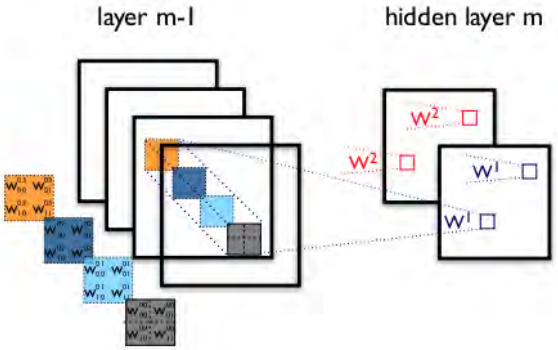
Sebuah MLP memiliki *i layer* dengan masing-masing *layer* berisi j_i neuron. MLP menerima input data satu dimensi dan mempropagasikan data tersebut pada jaringan hingga menghasilkan output. Tiap hubungan antar neuron pada dua layer yang bersebelahan memiliki parameter bobot satu dimensi yang menentukan kualitas model. Pada tiap data input pada *layer* dilakukan operasi linear dengan nilai bobot yang ada, lalu

hasil komputasi akan ditransformasi menggunakan operasi non linear yang disebut sebagai fungsi aktivasi.

Pada CNN, data yang dipropagasikan pada jaringan adalah data dua dimensi, sehingga operasi linear dan parameter bobot pada CNN berbeda. Pada CNN operasi linear menggunakan operasi konvolusi, sedangkan bobot tidak lagi satu dimensi saja, namun berbentuk empat dimensi yang merupakan kumpulan kernel konvolusi seperti pada **Gambar 2.2**. Dimensi bobot pada CNN adalah:

$$neuron\ input \times neuron\ output \times tinggi \times lebar$$

Karena sifat proses konvolusi, maka CNN hanya dapat digunakan pada data yang memiliki struktur dua dimensi seperti citra dan suara.



***Gambar 2.2 Proses Konvolusi Pada CNN* [10]**

2.2.2 Bentuk Jaringan

JST terdiri dari berbagai layer dan beberapa neuron pada masing-masing layer. Kedua hal tersebut tidak dapat ditentukan menggunakan aturan yang pasti dan berlaku berbeda-beda pada data yang berbeda [11].

Pada kasus MLP, sebuah jaringan tanpa *hidden layer* dapat memetakan persamaan linear apapun, sedangkan jaringan

dengan satu atau dua *hidden layer* dapat memetakan sebagian besar persamaan pada data sederhana. Namun pada data yang lebih kompleks, MLP memiliki keterbatasan. Pada kasus jumlah hidden layer dibawah tiga layer, terdapat pendekatan untuk menentukan jumlah neuron pada masing-masing layer untuk mendekati hasil optimal. Penggunaan layer diatas dua pada umumnya tidak direkomendasikan dikarenakan rawan terhadap *overfitting* serta kekuatan *backpropagation* berkurang secara signifikan.

Dengan berkembangnya *deep learning*, ditemukan bahwa untuk mengatasi kekurangan MLP dalam menangani data kompleks, diperlukan fungsi untuk mentransformasi data input menjadi bentuk yang lebih mudah dimengerti oleh MLP. Hal tersebut memicu berkembangnya deep learning dimana dalam satu model diberi beberapa layer untuk melakukan transformasi data sebelum data diolah menggunakan metode klasifikasi. Hal tersebut memicu berkembangnya model neural network dengan jumlah layer diatas tiga. Namun dikarenakan fungsi layer awal sebagai metode ekstraksi fitur, maka jumlah layer dalam sebuah DNN tidak memiliki aturan universal dan berlaku berbeda-beda tergantung dataset yang digunakan.

Karena hal tersebut, jumlah layer pada jaringan serta jumlah neuron pada masing-masing layer dianggap sebagai hyperparameter dan dioptimasi menggunakan pendekatan *searching*.

Sebuah CNN terdiri dari beberapa *layer*. Berdasarkan arsitektur LeNet5, terdapat empat macam *layer* utama pada sebuah CNN antara lain [12]:

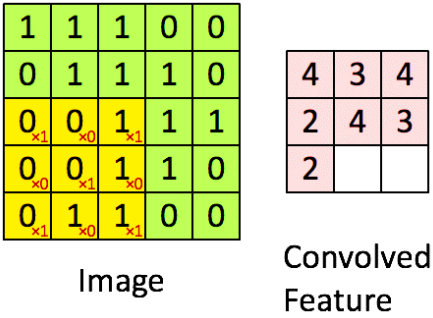
a. *Convolution Layer*

Convolution layer melakukan operasi konvolusi pada output dari *layer* sebelumnya. *Layer* tersebut adalah proses utama yang mendasari sebuah CNN.

Konvolusi adalah sebuah istilah matematis yang berarti mengaplikasikan sebuah fungsi pada output fungsi

lain secara berulang. Dalam pengolahan citra, konvolusi berarti mengaplikasikan sebuah *kernel* pada citra di semua offset yang memungkinkan.

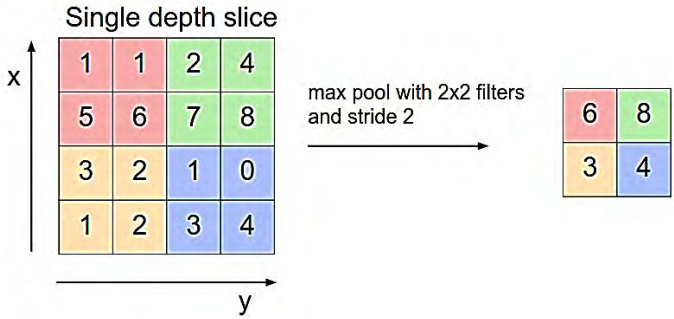
Tujuan dilakukannya konvolusi pada data citra adalah untuk mengekstraksi fitur dari citra input. Konvolusi akan menghasilkan transformasi linear dari data input sesuai informasi spasial pada data. Bobot pada *layer* tersebut menspesifikasikan kernel konvolusi yang digunakan, sehingga kernel konvolusi dapat dilatih berdasarkan input pada CNN.



***Gambar 2.3 Operasi Konvolusi* [13]**

b. Subsampling Layer

Subsampling adalah proses mereduksi ukuran sebuah data citra. Dalam pengolahan citra, *subsampling* juga bertujuan untuk meningkatkan invariansi posisi dari fitur. Dalam sebagian besar CNN, metode *subsampling* yang digunakan adalah *max pooling*. *Max pooling* membagi output dari *convolution layer* menjadi beberapa *grid* kecil lalu mengambil nilai maksimal dari tiap *grid* untuk menyusun matriks citra yang telah direduksi. Proses tersebut memastikan fitur yang didapatkan akan sama meskipun obyek citra mengalami translasi (pergeseran).



***Gambar 2.4 Operasi Max Pooling* [9]**

Menurut Springenberg et al. [14], penggunaan *pooling layer* pada CNN hanya bertujuan untuk mereduksi ukuran citra sehingga dapat dengan mudah digantikan dengan sebuah *convolution layer* dengan *stride* yang sama dengan *pooling layer* yang bersangkutan.

c. *Fully Connected Layer*

Layer tersebut adalah *layer* yang biasanya digunakan dalam penerapan MLP dan bertujuan untuk melakukan transformasi pada dimensi data agar data dapat diklasifikasi secara linear.

Tiap *neuron* pada *convolution layer* perlu ditransformasi menjadi data satu dimensi terlebih dahulu sebelum dapat dimasukkan ke dalam sebuah *fully connected layer*. Karena hal tersebut menyebabkan data kehilangan informasi spasialnya dan tidak reversibel, *fully connected layer* hanya dapat diimplementasikan di akhir jaringan.

Dalam sebuah paper oleh Lin et al., dijelaskan bahwa *convolution layer* dengan ukuran kernel 1x1 melakukan fungsi yang sama dengan sebuah *fully connected layer* namun dengan tetap mempertahankan karakter spasial dari data. Hal tersebut membuat penggunaan *fully connected layer* pada CNN sekarang tidak banyak dipakai.

d. Loss Layer

Seluruh JST selalu diakhiri dengan sebuah *loss layer*, yaitu *layer* yang bertanggung jawab menghitung *error* dari jaringan untuk proses pelatihan sekaligus melakukan fungsi khusus jaringan. Untuk kasus klasifikasi, yang digunakan pada umumnya adalah *softmax classifier*.

Softmax classifier menerima nilai input sejumlah kelas pada kasus klasifikasi, kemudian mengubahnya menjadi nilai normalized exponential menggunakan operasi softmax:

$$q = \frac{e^{f_{yi}}}{\sum_j e^{f_{ji}}} \tag{2.1}$$

Dimana f_{yi} adalah nilai label untuk data ke i dan f_{ji} adalah nilai input ke j pada data ke i. Akan dihasilkan sebuah vector q dengan nilai untuk masing-masing kelas yang menggambarkan tingkat keyakinan. Label dari data input akan ditentukan berdasarkan kelas dengan nilai keyakinan tertinggi. Untuk perhitungan besarnya *error* pada tahap pelatihan, digunakan *cross entropy loss*. Fungsi tersebut akan menghitung *error* antara nilai prediksi q dengan nilai sebenarnya p menggunakan :

$$H(p,q) = -\sum_x p(x) \log q(x) \tag{2.2}$$

Fungsi softmax menerima input sejumlah kelas, sehingga diperlukan satu layer tambahan sebelum loss layer yang mentransformasi data menjadi sejumlah kelas:

- *Fully Connected Layer*

  Sebelum *loss layer*, dipasang sebuah *fully connected layer* yang menerima seluruh input data yang

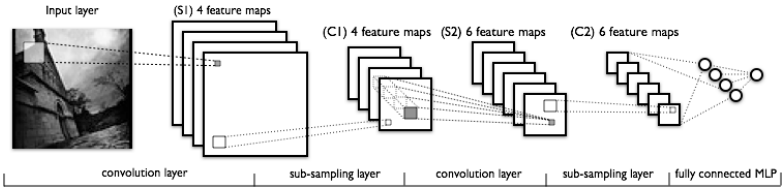
telah diubah menjadi satu dimensi dan memiliki jumlah *neuron* sesuai jumlah kelas pada kasus klasifikasi. Metode tersebut memiliki kekurangan yang sama dengan *fully connected layer* pada umumnya yaitu hilangnya informasi spasial dari data.

- *Global Average Pooling Layer*

  Sebelum loss layer, dipasang sebuah *convolutional neural network* dengan dimensi filter 1x1 dan jumlah *feature maps* sesuai jumlah kelas pada kasus klasifikasi, lalu dilanjutkan dengan sebuah *global average pooling layer* yang mengambil nilai rata-rata dari masing-masing *feature maps* sehingga akan menghasilkan output berupa satu nilai untuk tiap kelas klasifikasi. Pendekatan tersebut memiliki kelebihan yaitu tetap mempertahankan informasi spasial dari data.

Arsitektur umum CNN terdiri dari *convolution layer* dan *subsampling layer* yang disusun secara bergantian dan *fully connected layer* pada beberapa *layer* terakhir yang kemudian dihubungkan dengan sebuah *loss layer*.

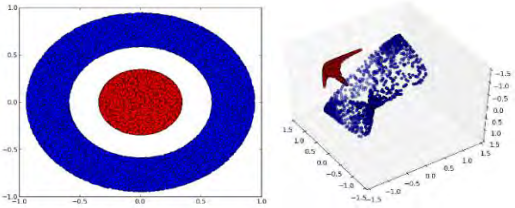
Model LeNet5 merupakan arsitektur CNN pertama yang berhasil melakukan klasifikasi citra dengan hasil memuaskan. Pengembangan CNN kekinian menggunakan model LeNet sebagai dasaran dan kemudian diubah ukuran maupun urutan *layer*-nya.



***Gambar 2.5 Arsitektur LeNet 5* [15]**

2.2.3 Fungsi Aktivasi

Fungsi aktivasi adalah fungsi non linear yang memungkinkan sebuah JST untuk dapat mentransformasi data input menjadi dimensi yang lebih tinggi sehingga dapat dilakukan pemotongan *hyperplane* sederhana yang memungkinkan dilakukannya klasifikasi. Dalam CNN terdapat tiga fungsi aktivasi yang banyak digunakan yaitu sigmoid, tanh, dan ReLU.



***Gambar 2.6 Transformasi Non Linear Pada JST* [16]**

a. Fungsi Sigmoid

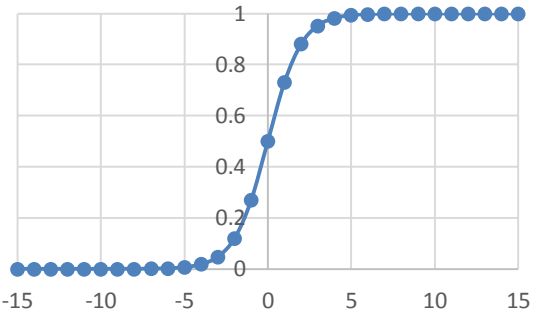
Fungsi Sigmoid mentransformasi *range* nilai dari input x menjadi antara 0 dan 1 dengan distribusi fungsi seperti pada **Gambar 2.2.8**. Fungsi sigmoid memiliki bentuk sebagai berikut:

$$\sigma(x) = \frac{1}{(1 + e^{-x})} \tag{2.3}$$

Fungsi sigmoid sekarang sudah tidak banyak digunakan dalam praktek [9] karena memiliki kelemahan utama yaitu *range* nilai output dari fungsi sigmoid tidak terpusat pada angka nol.

Hal tersebut menyebabkan terjadinya proses *backpropagation* yang tidak ideal dikarenakan bobot pada JST tidak terdistribusi rata antara nilai positif dan negatif serta nilai bobot akan banyak mendekati ekstrim 0 atau 1. Dikarenakan komputasi nilai propagasi menggunakan perkalian, maka nilai ekstrim tersebut akan menyebabkan

efek *saturating gradients* dimana jika nilai bobot cukup kecil, maka lama kelamaan nilai bobot akan mendekati salah satu ekstrim sehingga memiliki gradien yang mendekati nol. Jika hal tersebut terjadi, maka neuron tersebut tidak akan dapat mengalami *update* yang signifikan dan akan nonaktif.



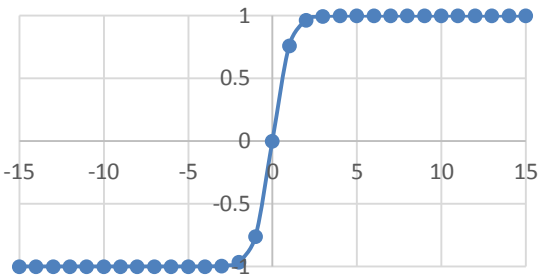
Gambar 2.7 Distribusi Fungsi Sigmoid

b. Fungsi Tangen Hiperbolik (tanh)

Fungsi tanh memiliki distribusi yang sangat mirip dengan fungsi sigmoid namun dalam *range* antara -1 sampai 1 dan distribusi yang lebih sempit. Fungsi tanh memiliki bentuk sebagai berikut:

$$\sigma(x) = \tanh(x) \tag{2.4}$$

Karakteristik terpusat pada nol tersebut mengatasi kelemahan utama pada fungsi sigmoid, sehingga dalam praktik, fungsi tanh selalu lebih bagus daripada fungsi sigmoid.



Gambar 2.8 Distribusi Fungsi tanh

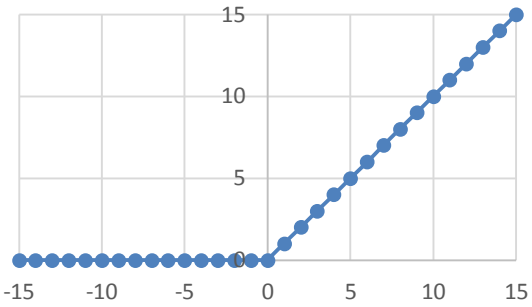
c. *Fungsi Rectified Linear Unit (ReLU)*

Fungsi ReLU termasuk salah satu fungsi aktivasi baru yang menjadi populer bersamaan dengan kemunculan CNN modern. Fungsi ReLU melakukan operasi *thresholding* nilai linear pada nol. Secara matematis fungsi ReLU memiliki bentuk:

$$\sigma(x) = \max(0, x) \tag{2.5}$$

Meskipun bentuknya yang sederhana sekali, namun fungsi ReLU telah menjadi standar terbaru dalam implementasi CNN [9]. Menurut Krizhevsky et. al. [17], konvergensi proses pelatihan jika menggunakan fungsi ReLU lebih cepat hingga 6 kali dibandingkan fungsi tanh.

Namun fungsi ReLU memiliki kelemahan yang mirip dengan fungsi sigmoid yaitu *saturating gradient*, dikarenakan range nilai bawah nol. Namun hal tersebut dapat dimitigasi dengan menggunakan nilai *learning rate* yang sesuai pada proses pelatihan.



Gambar 2.9 Distribusi Fungsi ReLU

d. Fungsi Leaky ReLU

Fungsi Leaky ReLU muncul atas permasalahan *saturating gradient* pada Fungsi ReLU biasa. Hal tersebut dilakukan dengan tidak melakukan *thresholding* utuh, namun hanya melakukan reduksi nilai dibawah nol, biasanya dengan faktor nilai 0.01. Fungsi Leaky ReLU memiliki bentuk sebagai berikut:

$$\sigma(x) = \max(x/100, x) \tag{2.6}$$

Benjamin et.al, telah melaporkan kesuksesan [18] dalam menggunakan fungsi Leaky ReLU, namun hasilnya tidak selalu konsisten untuk seluruh kasus, sehingga mayoritas orang masih lebih memilih untuk menggunakan fungsi ReLU standar dengan mitigasi pada kasus *saturating gradient*.

2.2.4 Metode Pelatihan

Proses pelatihan pada JST adalah sebuah *problem* optimasi dimana parameter *layer* pada jaringan (bobot dan bias) menjadi obyek optimasi sedangkan nilai *error* menjadi *objective function* yang diminimalisir.

Metode optimasi pada JST umumnya menggunakan g*radient descent* dimana proses *update* parameter dilakukan perlahan sesuai dengan arah gradien (turunan pertama) agar dipastikan menuju sebuah optima. Keunggulan gradient descent adalah kemampuannya untuk dilakukan secara online, sehingga memungkinkan untuk dilakukan pada data pelatihan yang sangat besar. Pada kasus data besar, digunakan modifikasi pada *gradient descent*, yaitu *mini batch stochastic gradient descent*:

```
while True:
   batch = sample_train_data(data, batch_size)
   params_grad = evaluate_gradient(loss, batch, params)
   params += learn_rate * params_grad
```

Gambar 2.10 Pseudocode Mini Batch SGD

Pada kasus *feed-forward neural network* seperti MLP dan CNN, untuk melakukan *update parameter* pada jaringan, digunakan algoritma *back propagation*. Algoritma *back propagation* adalah metode untuk mendistribusikan *error* dari output pada masing-masing parameter secara proporsional sesuai peran masing-masing parameter dalam pelatihan. Hal tersebut dilakukan dengan melakukan propagasi ulang dari output ke input (*back propagation)* berdasarkan nilai *partial derivative* dari error awal pada masing-masing parameter.

```
network = construct_network_layers()
network_weights = init_weights(network, size)
for (i=1 to max_iter):
   pattern[i] = select_pattern(input_patterns)
   output[i] = forward_propagate(pattern[i], network)
   backward_propagate (pattern[i], output[i], network)
   update(pattern[i], output[i], network, learn_rate)
return network
```

Gambar 2.11 Pseudocode Back Propagation

2.2.5 Parameter

Seluruh JST adalah sebuah *parametric model* yang berarti sebuah model yang melakukan *fitting* terhadap data dengan memodifikasi nilai dari parameter jaringannya (bobot dan bias. Berhubungan dengan ini, JST adalah metode *machine learning* yang memiliki kelemahan utama yaitu banyaknya parameter pada model, baik parameter internal jaringan maupun parameter eksternal. Hal tersebut membuat pelatihan model serta optimasi model pada JST menjadi sebuah proses yang sangat memakan waktu.

Karena dalam *machine learning* terdapat banyak sekali parameter dalam pembuatan model, maka dikenal sebuah istilah yaitu hyperparameter. Hyperparameter didefinisikan sebagai parameter dari sebuah distribusi di luar distribusi pada model [19]. Dalam konteks model klasifikasi, distribusi model adalah distribusi dari permasalahan klasifikasi yang sedang dipelajari oleh model.

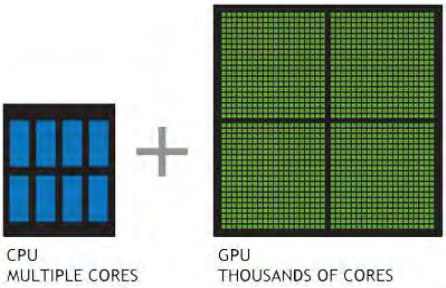
Pada JST, yang disebut parameter adalah nilai bobot dan bias dari *layer*. Seluruh parameter lain disebut dengan istilah hyperparameter karena berada di luar domain distribusi dari model.

2.3 GPU *Acceleration*

Salah satu kelemahan utama dari metode *Deep Learning* pada umumnya adalah proses komputasi yang lama. Hal tersebut dikarenakan parameter model JST yang sangat banyak serta data input yang biasanya sangat besar. Hal tersebut juga menyebabkan penelitian JST sempat berhenti beberapa tahun. Hal yang memungkinkan penelitian JST untuk berlanjut adalah perkembangan pesat pada sektor perangkat keras, khususnya pada *Graphical Processing Unit* (GPU).

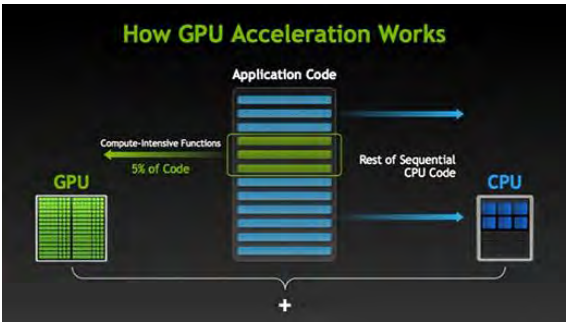
Perkembangan GPU yang pesat inilah yang mulai mengalihkan penggunaan GPU di luar sektor *computer graphics* yang lebih dikenal dengan istilah *General-Purpose*

GPU (GPGPU). Hal tersebut memungkinkan perkembangan *Machine Learning* yang sangat pesat.



***Gambar 2.12 Perbandingan CPU dan GPU* [20]**

GPU dapat melakukan komputasi dengan performa yang sangat baik dikarenakan arsitekturnya yang memungkinkan komputasi paralel. Hal tersebut membuat performa GPU dalam komputasi jauh mengungguli *Central Processing Unit* (CPU) komputer [20]. Hal tersebut dikarenakan sebuah GPU terdiri dari hingga ribuan *processing core*, sedangkan CPU pada umumnya hanya memiliki empat hingga delapan *processing core*.



***Gambar 2.13 Cara Kerja GPU Acceleration* [20]**

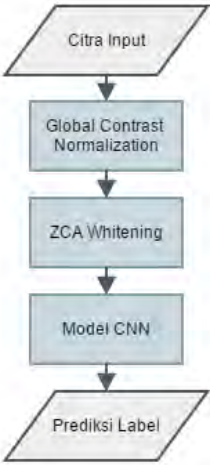
Namun dalam prakteknya proses komputasi tidak dapat sepenuhnya dilakukan oleh GPU karena kapabilitas komputasi GPU yang jauh terbatas dibandingkan CPU. Sebagian besar proses komputasi tetap dilaksanakan pada CPU, sedangkan GPU digunakan dalam proses komputasi yang berat dan cenderung repetitif, terutama dalam proses pelatihan model pada *Machine Learning*. Dalam bidang *deep learning*, teknologi komputasi GPU yang paling sering digunakan adalah *Compute Unified Device Architecture* (CUDA). CUDA dikembangkan oleh Nvidia dan telah menjadi teknologi standar dalam melakukan *GPU Acceleration.*

BAB III
DESAIN PERANGKAT LUNAK

Pada bab ini akan dijelaskan gambaran umum program utama dalam sebuah *flowchart* dan dilanjutkan dengan penjabaran terkait lingkungan pengembangan perangkat lunak. Selanjutnya, untuk penjelasan lebih detil akan disajikan dalam penjelasan mendalam dan disertai *pseudocode* atau diagram.

3.1 Desain Metodologi

Dalam tugas akhir ini, disusun sebuah metodologi untuk melakukan klasifikasi obyek pada citra. Metodologi diawali dengan praproses. Metode yang paling umum digunakan meliputi *Global Contrast Normalization* dan *ZCA Whitening* [9] [13] [21] [22] [23]. Lalu dilanjutkan pembuatan model klasifikasi menggunakan *Convolutional Neural Network*.



Gambar 3.1 Metodologi Program

3.2 Desain Program Secara Umum

Dalam tugas akhir ini, akan dibuat dua program terpisah, yaitu program untuk praproses data, dan program untuk klasifikasi. Program praproses akan menerima input data CIFAR-10 asli, lalu melakukan pengolahan data hingga siap untuk digunakan oleh program klasifikasi. Program klasifikasi akan mengolah data jadi untuk membangun sebuah model klasifikasi. Pemisahan tersebut dilakukan untuk efisiensi *run time* dikarenakan program klasifikasi akan dijalankan berkali-kali dengan data hasil praproses yang sama.

Desain perangkat lunak pada tugas akhir ini akan dibagi menjadi tiga bagian utama, yaitu:

1. Praproses : Mengolah data asli sehingga menghasilkan data olahan yang lebih mudah digunakan oleh program klasifikasi.
2. Pembangunan Model : Membuat model CNN untuk kasus klasifikasi citra.
3. *Fitting* Model : Melatih model CNN yang telah dibuat terhadap dataset CIFAR-10.

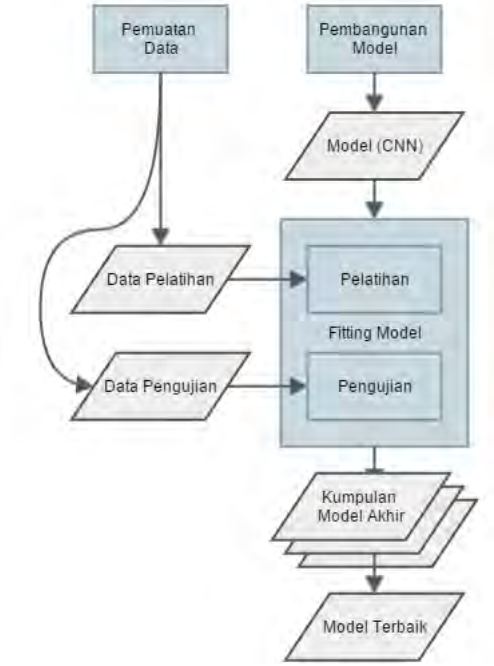
Program praproses akan menerima input data CIFAR-10 mentah, lalu membaginya menjadi data pengujian dan pelatihan. Kemudian akan dilakukan transformasi data citra agar citra lebih cocok untuk pemakaian dalam *learning*. Hal tersebut menggunakan *Global Contrast Normalization* dan *ZCA Whitening*. Lalu pada data pelatihan yang telah ditransformasi akan dilakukan augmentasi data untuk memungkinkan model mempelajari lebih banyak data dengan variansi yang sedikit berbeda dari data awal.

Gambar 3.2 Flowchart Program Praproses

Program klasifikasi akan menyusun sebuah model untuk melakukan klasifikasi citra. Ada dua proses besar dalam program tersebut, yaitu pembangunan model CNN, dan fitting model CNN.

Pada proses pembangunan model CNN, akan diterima sejumlah hyperparameter yang mendefinisikan jaringan CNN

yang digunakan, lalu akan dibangun sebuah model CNN awal sesuai input. Model awal tersebut kemudian akan diolah dalam proses *fitting* data dimana parameter pada jaringan (bobot dan bias) akan dikonfigurasi sesuai dengan dataset melalui proses pelatihan. Proses *fitting* data akan menghasilkan beberapa model CNN akhir, lalu akan dipilih model dengan nilai *error* terkecil sebagai model terbaik.



Gambar 3.3 Flowchart Program Klasifikasi

3.3 Lingkungan Pengembangan Perangkat Lunak

Program akan dibangun dengan menggunakan *library* bernama Theano [24]. Hal tersebut dikarenakan Theano

didesain untuk pemodelan matematis dengan memanfaatkan GPU *Acceleration*. Dalam beberapa tahun terakhir, Theano juga dikembangkan dengan fokusan pada *deep learning* sehingga memiliki banyak fungsi untuk menunjang pembuatan model CNN.

Terdapat beberapa *library* alternatif selain theano untuk implementasi CNN, diantaranya:

a. Caffe

Caffe adalah library deep learning yang didesain secara *black box,* yaitu pengguna cukup mengatur konfigurasi model saja tanpa perlu memahami kinerja dalam dari model. Karena hal tersebut, caffe tidak menawarkan banyak kebebasan dalam mengatur model, sehingga cocok digunakan pada lingkungan aplikatif, namun kurang cocok untuk lingkungan riset.

b. Torch

Torch adalah kerangka kerja deep learning lengkap dengan kode syntax independen yang dikembangkan oleh Deepmind Facebook. Torch dibangun untuk lingkungan riset, khususnya di dalam facebook sendiri, namun tidak mendapat keuntungan dari penggunaan bahasa umum, sehingga lingkup penggunaannya lebih terbatas.

c. Pylearn2

Pylearn2 adalah library wrapper dari Theano untuk mempermudah penggunaan Theano dengan pendekatan *black box*. Pada saat pengerjaan tugas akhir ini, Pylearn2 masih dalam tahap pengembangan dan belum stabil, sehingga dipilih untuk menggunakan Theano.

3.3.1 *Theano*

Theano adalah library pada bahasa pemrograman Python yang menggunakan *symbolic programming* untuk mendefinisikan, mengoptimasi, dan mengevaluasi fungsi

matematis secara cepat dan efisien, terutama dalam penanganan *multidimensional array (matrix)*.

***Gambar 3.4 Logo Library Theano* [15]**

Theano dikembangkan oleh LISA lab di University of Montreal, lab yang fokus pada pengembangan teknologi *Machine Learning*. LISA dipimpin langsung oleh Yoshua Bengio, salah satu tokoh pengembang *Deep Learning* dunia.

Beberapa kelebihan Theano antara lain [15]:

- Integrasi yang sangat dekat dengan library pengolahan matriks pada Python (NumPy). Hal tersebut mempermudah pengolahan data.
- Menggunakan GPU *Acceleration* secara otomatis.
- Memungkinkan diferensiasi otomatis.
- Teroptimasi menggunakan *dynamic C code generation*

3.3.2 *GPU Acceleration*

Penggunaan GPU pada theano sangat mudah karena dilakukan secara otomatis oleh *library*. Namun karena keterbatasan GPU secara umum, terdapat beberapa batasan yang harus dipatuhi dalam menyusun kode theano agar penggunaan GPU dapat optimal.

a. Penggunaan *Shared Variable*

Variabel theano dan python secara *default* disimpan di RAM. Jika operasi yang melibatkan penggunaan intensif GPU mengakses variabel pada RAM, maka *overhead* dari proses tersebut akan lebih mahal dari percepatan yang dihasilkan oleh penggunaan GPU. Oleh karena itu, untuk mengatasi hal tersebut, variabel

sebaiknya disimpan di dalam GPU Memory agar tidak memerlukan *overhead* yang berat.

Hal tersebut dapat dilakukan menggunakan *Shared Variable* pada theano. Dalam tugas akhir ini, variabel yang banyak diakses oleh GPU akan dimasukkan dalam *Shared Variable,* antara lain dataset, parameter *layer*, dan variabel akumulasi proses pelatihan. Memori GPU jauh lebih kecil dari RAM, sehingga pada pembuatan CNN perlu diperhatikan penggunaan memori.

```
def build_shared_zeros(shape, name):
    return theano.shared(value=np.zeros(shape,
                         dtype=theano.config.floatX),
                         name=name, borrow=True)
```

Gambar 3.5 Penggunaan Shared Variable pada Theano

b. *float32 Casting*

Penggunaan GPU oleh Theano memiliki batasan yaitu hanya dapat mengolah variabel float32. Oleh karena hal tersebut, seluruh variabel yang diolah pada GPU harus dipastikan memiliki tipe float32. Jika hal tersebut tidak dilakukan maka Theano akan mengalihkan proses yang bersangkutan ke CPU sehingga akan lambat.

Karena hal tersebut, perlu dilakukan *float32 casting* setelah operasi yang melibatkan *Shared Variable*. Dalam theano disarankan untuk menggunakan variabel `theano.config.floatX` yang berisi definisi floatX pada file konfigurasi .theanorc sehingga memungkinkan perubahan cepat tipe float jika diperlukan.

```
new_lr=np.cast[theano.config.floatX](self._lr.get_value()*0.1)
self._lr.set_value(new_lr)
```

Gambar 3.6 Penggunaan float32 Casting pada Theano

3.4 Praproses

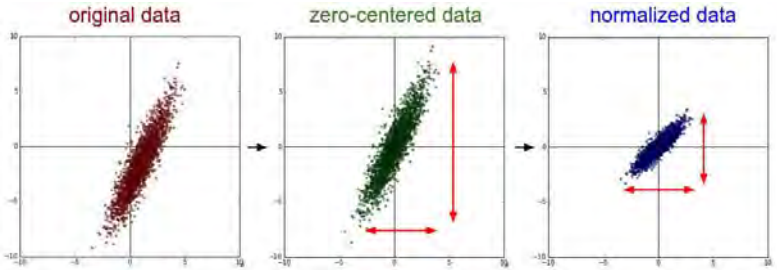
Pengolahan data di awal karena tidak semua data dapat langsung dipahami oleh program *learning*. Dalam kasus tugas akhir ini, dilakukan *Global Contrast Normalization*, *ZCA Whitening*, dan *Data Augmentation*.

Data citra CIFAR-10 memiliki 60000 data dengan dimensi tiap citra sebesar 3 x 32 x 32, sehingga jumlah fitur tiap data adalah sebanyak 3072. Meskipun jumlah fitur pada data sangat besar, namun ekstraksi fitur maupun reduksi fitur tidak dilakukan, karena akan dilakukan sendiri oleh CNN.

3.4.1 *Global Contrast Normalization* (GCN)

Dalam hampir seluruh kasus pengolahan data, normalisasi adalah tahap praproses yang wajib dilakukan dengan tujuan agar bobot tiap fitur kurang lebih sama sehingga tidak ada fitur yang lebih berpengaruh dari fitur lainnya.

GCN melibatkan pergeseran nilai agar memiliki nilai rata-rata nol, serta pengubahan skala nilai agar memiliki standar deviasi bernilai 1. Hal tersebut sama seperti proses normalisasi, namun dilakukan pada seluruh data untuk masing-masing piksel citra. Pada kasus data citra natural, seluruh piksel hampir pasti memiliki skala yang sama yaitu antara 0 sampai 255 [13]. Oleh karena hal tersebut, GCN cukup dilakukan dengan mengurangi nilai mean tanpa perlu perubahan skala.



***Gambar 3.7 Pengaruh GCN terhadap Data* [9]**

Pada pengolahan statistik data seperti pada GCN, proses harus dilakukan menggunakan data pelatihan saja, lalu informasi hasil pengolahan data pelatihan diaplikasikan ke data pengujian. Hal tersebut bertujuan agar data pelatihan dipastikan independen dari data pelatihan sehingga akan menghindari *overfitting* [9].

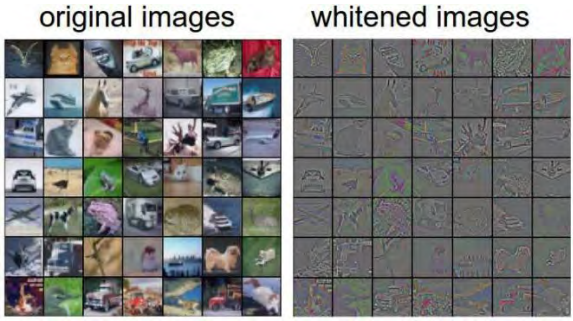
Input	train_data, test_data
Output	train_data, test_data

```
1.  mean = hitung_mean_per_piksel(train_data)
2.
3.  for data in train_data:
4.      data -= mean
5.
6.  for data in test_data:
7.      data -= mean
8.
9.  return train_data, test_data
```

Gambar 3.8 Pseudocode Global Contrast Normalization

3.4.2 *ZCA Whitening*

Pada data citra natural, nilai tiap piksel akan cenderung redundan karena antar piksel yang bersebelahan sangatlah berkolerasi. ZCA Whitening bertujuan untuk mengurangi korelasi antar piksel sehingga informasi spasial dari tiap piksel akan lebih kuat [13].



***Gambar 3.9 ZCA Whitening* [23]**

Hal tersebut dilakukan dengan menghitung matriks kovarian dari data, lalu menghitung eigenvector dan eigenvalue menggunakan *Singular Value Decomposition*. *Whitening matrix* didapatkan dengan membagi eigenvector dengan eigenvalue (akar dari diagonal nilai singular), lalu mengalikannya lagi dengan transpos dari eigenvector. Semua menggunakan perkalian matriks. ZCA whitening dilakukan dengan mengalikan data dengan *whitening matrix*.

Input	train_data, test_data
Output	train_data, test_data

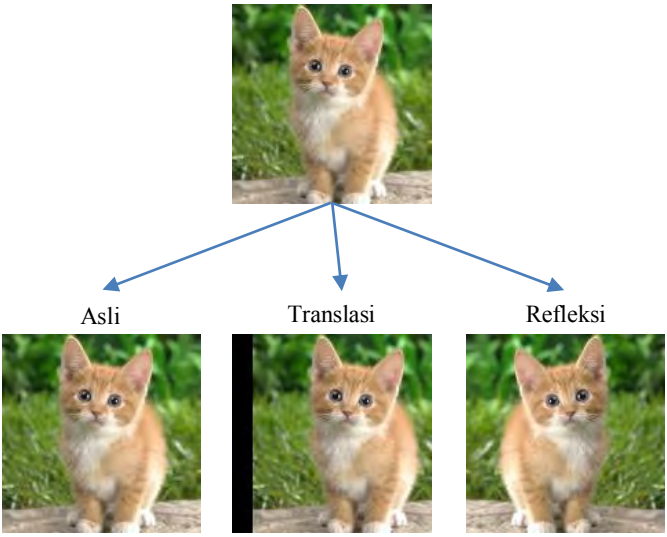
```
1.  x = train_data
2.
3.  covariance = dot(transpose(x), x) / jumlah_data(x)
4.  u, s, v = svd(covariance)
5.  eigenvalue = sqrt(diag(s))
6.  whitening = dot(dot(u,1/eigenvalue),transpose(u))
7.
8.  whitened_train_data = dot(train_data, whitening)
9.  whitened_test_data = dot(test_data, whitening)
10.
11. return whitened_train_data, whitened_test_data
```

Gambar 3.10 Pseudocode ZCA Whitening

Sama dengan GCN, penghitungan whitening matrix hanya dilakukan menggunakan data pelatihan, namun digunakan untuk kedua data pelatihan dan data pengujian.

3.4.3 Augmentasi Data

Augmentasi data melakukan penambahan data pelatihan dengan melakukan berbagai transformasi dasar pada citra. Hal tersebut bertujuan agar model dapat mengenali variansi dasar pada citra obyek. Tranformasi yang umum digunakan dalam augmentasi data adalah translasi dan refleksi citra [10].



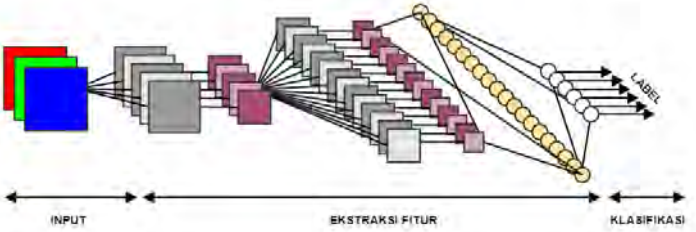
Gambar 3.11 Data Augmentation

3.5 Pembangunan Model

JST memiliki karakteristik unik yaitu bahwa struktur model JST berpengaruh pada kualitas model. Pada CNN, jenis *layer* lebih bervariasi dibandingkan MLP sehingga arsitektur jaringan memiliki banyak variasi. Ukuran jaringan (kedalaman dan lebar) adalah hyperparameter jaringan yang perlu di *tuning* terhadap dataset yang digunakan [25]. Dikarenakan proses *tuning* ukuran jaringan yang sangat lama, pada tugas akhir ini kedalaman akan dibatasi maksimal lima *convolution layer* dan lebar jaringan akan mengambil nilai awal dari *All Convolutional Nets* yaitu 96 neuron, lalu akan dilakukan eksplorasi pada nilai kelipatan 96.

Pada CNN, terdapat dua bagian utama dari arsitektur model, bagian ekstraksi fitur, dan bagian klasifikasi. Bagian ekstraksi fitur berfungsi untuk mentransformasi data input sehingga dapat diklasifikasi dengan baik. Hal tersebut

dilaksanakan dengan mempropagasi data melalui berbagai transformasi linear maupun non linear. Bagian klasifikasi berfungsi untuk mereduksi data sehingga dapat dimasukkan ke dalam *softmax classifier* untuk dilakukan klasifikasi.



Gambar 3.12 Pembagian Arsitektur CNN

Penjelasan model akan dibagi menjadi dua bagian, yaitu *Network Layer* dan *Network Architecture*.

3.5.1 *Network Layer*

Pada tugas akhir ini akan digunakan 5 macam *layer*. Tiap *layer* menerima input data dari seluruh neuron sebelumnya, lalu menghasilkan output data sesuai hasil pengolahan pada *layer*. Akan dijelaskan desain internal dari tiap *layer*.

3.5.1.1 *Perancangan Layer*

Pada bagian ini akan dijelaskan secara mendalam terkait jenis *layer* pada CNN serta proses yang dilakukan pada masing-masing *layer*.

a. Convolutional Layer

Convolutional layer melakukan operasi konvolusi untuk mentransformasi data secara linear berdasarkan informasi spasial data, lalu melakukan transformasi non linear menggunakan fungsi aktivasi. Karena mengolah data dua dimensi, input dari *layer* tersebut berukuran 4

dimensi, yaitu *[jumlah_data, jumlah_kernel_output, tinggi_data, lebar_data]*.

Layer tersebut memiliki parameter bobot dan bias. Bobot terdiri dari banyak kernel dan memiliki ukuran empat dimensi yaitu *[jumlah_kernel_output, jumlah_kernel_input, tinggi_kernel, lebar_kernel]*, sedangkan bias memiliki ukuran satu dimensi saja yaitu *[jumlah_kernel_output]*. Bobot digunakan dalam melakukan konvolusi, sedangkan bias ditambahkan setelah operasi konvolusi.

Inisialisasi nilai bobot tidak boleh nol karena akan menyebabkan seluruh neuron memiliki nilai yang sama meskipun telah melalui pelatihan. Menurut sebuah paper yang menganalisa hal tersebut oleh He et. al. [26], bobot pada sebuah neuron harus acak dengan standar deviasi $\sqrt{2.0/n}$ dimana *n* adalah *jumlah_kernel_input*. Untuk bias, inilisiasi nilai dapat menggunakan nol.

Hyperparameter yang digunakan dalam konvolusi adalah ukuran *padding*. Jika padding bernilai satu, maka data citra akan diberi tambahan nilai nol berdimensi satu di sekeliling citra ketika melalui proses konvolusi. Hal tersebut berpengaruh pada ukuran citra output.

Input	kern_size
Output	-

```
1.  weight = init_matrix(kern_size, random(sqrt(2/n_in)))
2.  bias = init_matrix(kern_size[1], 0)
```

Gambar 3.13 Pseudocode Inisialisasi Convolutional Layer

Input	input_data, padsize, activation
Output	output_data

```
1.  padded = pad_data(input_data, padsize)
2.  lin_out = convolve(padded,weight)
3.  lin_out += bias
4.  output_data = activation(lin_out)
```

Gambar 3.14 ***Pseudocode Operasi Convolutional Layer***

b. Fully Connected Layer

Fully connected layer melakukan operasi perkalian matrix untuk mentransformasi data secara linear. Namun berbeda dengan *convolution layer*, *layer* tersebut tidak menyimpan informasi spasial dan input harus berukuran dua dimensi yaitu *[jumlah_data, data]*. Oleh karena hal tersebut, bobot dari *fully connected layer* hanya berukuran dua dimensi saja, yaitu *[jumlah_kernel_output, jumlah_kernel_input]*.

Layer tersebut sama persis dengan *convolutional* layer, namun berbeda dalam penanganan data.

Input	kern_size
Output	-
1. weight = init_matrix(kern_size, random(sqrt(2/n_in))) 2. bias = init_matrix(kern_size[1], 0)	

Gambar 3.15 Pseudocode Inisialisasi Fully Connected Layer

Input	input_data, activation
Output	output_data
1. lin_out = dot(input_data,weight) 2. lin_out += bias 3. output_data = activation(lin_out)	

Gambar 3.16 Pseudocode Operasi Fully Connected Layer

c. Max Pooling Layer

Layer tersebut adalah sebuah *subsampling layer* yang bertujuan mereduksi ukuran data. Citra akan dibagi menjadi beberapa *pool*, lalu akan dikalkulasi satu nilai dari tiap *pool* untuk menghasilkan data output. Dalam kasus ini, diambil nilai terbesar pada setiap *pool*.

Seluruh *subsampling layer* tidak memiliki parameter bobot sehingga tidak memerlukan inisialisasi. Yang menjadi hyperparameter dalam *max pooling layer* adalah ukuran *pool (pool_size)*, serta jarak antar *pool (pool_stride)*. Pooling dilakukan pada tiap *feature maps* pada data.

Input	input_data, psize, pstride
Output	output_data
1. pooled = extract_pool(input_data, psize, pstride) 2. output_data = max(pooled, axis=2)	

Gambar 3.17 Pseudocode Operasi Max Pooling Layer

d. Global Average Pooling Layer

Global average pooling layer juga merupakan sebuah *subsampling layer,* namun dalam hal ini dilakukan *global subsampling* dimana pada tiap *feature maps*, hanya dibuat satu *pool*. Karena hal tersebut, tidak terdapat hyperparameter *pool_size* dan *pool_stride*.

Seluruh *global subsampling layer* pasti akan menghasilkan data output tanpa dimensi spasial, oleh karena itu, *layer* tersebut hanya digunakan pada bagian klasifikasi dari CNN. Dalam kasus ini digunakan *average pooling* dimana diambil nilai rata-rata pada tiap *pool*.

Input	input_data
Output	output_data
1. output_data = mean(input_data, axis=2)	

Gambar 3.18 Pseudocode Operasi Max Pooling Layer

e. Dropout Layer

Dropout Layer adalah *layer* yang berfungsi melakukan regularisasi pada JST dengan mematikan neuron secara acak dengan kemungkinan *drop_rate* pada tiap iterasi pelatihan [27]. Ketika tahap pengujian, dropout dinonaktifkan untuk mendapatkan hasil menyeluruh, namun output dari masing-masing neuron yang menggunakan dropout dikalikan dengan *drop_rate* agar outputnya proporsional dengan ketika tahap pelatihan.

Dalam tugas akhir ini akan diterapkan *inverted dropout* [9] dimana penyesuaian skala output dilakukan pada tahap pelatihan dan bukan pada tahap pengujian. Hal

tersebut dilakukan agar model dapat bekerja lebih cepat pada tahap pengujian.

Input	input_data, drop_rate
Output	output_data

```
1.  mask = binomial(size(input_data), drop_rate)
2.  if training:
3.      output_data = (input_data * mask) / drop_rate
4.  else:
5.      output_data = input_data
```

Gambar 3.19 Pseudocode Operasi Dropout Layer

f. Softmax Layer

Softmax Layer berbeda dengan layer lainnya karena harus berada pada ujung jaringan. Layer tersebut menerima input sejumlah kelas pada klasifikasi, lalu menghitung *cross entropy* untuk tiap kelas. Prediksi kelas dilakukan dengan mengambil kelas dengan nilai *cross entropy* tertinggi.

Dalam *fitting* model, digunakan dua fungsi yang berhubungan dengan nilai *cross entropy*. Pada tahap pelatihan, digunakan perhitungan *cross entropy loss* untuk menghitung *loss* hasil propagasi data. Pada tahap pengujian, digunakan perhitungan *error* yang membandingkan label prediksi terhadap label yang benar.

Input	input_data, label_data
Output	loss / error

```
1.  p_y_given_x = softmax(input_data)
2.
3.  if training:
4.      #return loss
5.      return –mean(label_data * log(p_y_given_x))
6.  else:
7.      #return error
8.      prediction = argmax(p_y_given_x)
9.      return mean(wrong_pred(prediction, label_data))
```

Gambar 3.20 Pseudocode Operasi Softmax Layer

3.5.1.2 Perancangan Model CNN

Pada tugas akhir ini, akan dirancang model CNN yang dan dapat menyesuaikan arsitektur jaringan dengan format input yang sederhana. *Layer* pada program akan disederhanakan menjadi tiga jenis saja. Satu jenis *layer* untuk tahap ekstraksi fitur, dan dua jenis *layer* untuk tahap klasifikasi.

a. *Layer* Ekstraksi Fitur

Pada ekstraksi fitur, dibuat *layer* yang melakukan fungsi konvolusi, *max pooling*, sekaligus *dropout* yang akan disebut sebagai *ConvPoolDropLayer*. Hal tersebut bertujuan untuk menyederhanakan penyusunan arsitektur jaringan dikarenakan pada tahap tersebut, penyusunan *layer* berorientasi pada *convolutional layer* saja.

Input	input_data, padsize, activation, psize, pstride, drop_rate
Output	output_data

```
1.  # Convolution
2.  padded = pad_data(input, padsize)
3.  lin_out = convolve(padded, weight)
4.  lin_out += bias
5.  non_linear = activation(lin_out)
6.
7.  # Pooling
8.  if psize:
9.      pooled = extract_pool(non_linear, psize, pstride)
10.     pooled = max(pooled, axis=2)
11. else:
12.     pooled = non_linear
13.
14. # Dropout
15. if drop_rate:
16.     mask = binomial(size(pooled), drop_rate)
17.     if training:
18.         output_data = (input_data * mask) / drop_rate
19.     else: output_data = input_data
20. else: output_data = pooled
```

Gambar 3.21 Pseudocode ConvPoolDropLayer

b. Layer Klasifikasi

Untuk tahap klasifikasi, diperlukan kombinasi satu layer transformasi linear untuk mereduksi jumlah *feature maps* yang kemudian disusul dengan *softmax layer*. Pada umumnya terdapat dua pendekatan dalam mereduksi jumlah *feature maps* seperti yang telah dijelaskan pada bab sebelumnya, sehingga disini dibuat dua jenis layer klasifikasi, yaitu *FullyConnectedSoftmax* dan *GlobalAverageSoftmax*.

FullyConnectedSoftmax menggabungkan sebuah *fully connected layer* tanpa fungsi non linear dengan *softmax layer* untuk klasifikasi. Pendekatan tersebut menghilangkan informasi spasial namun masih banyak digunakan.

Input	input_data, label_data
Output	loss / error

```
1.  # Fully Connected
2.  lin_out = dot(input_data, weight)
3.  lin_out += bias
4.
5.  # Softmax
6.  p_y_given_x = softmax(lin_out)
7.
8.  if training:
9.      loss = -mean(label_data * log(p_y_given_x))
10. else:
11.     prediction = argmax(p_y_given_x)
12.     error = mean(wrong_pred(prediction, label_data))
```

Gambar 3.22 Pseudocode FullyConnectedSoftmax

GlobalAverageSoftmax bertujuan untuk mempertahankan informasi spasial dalam melakukan reduksi data. *Layer* tersebut menggabungkan *convolutional layer* dengan ukuran kernel 1x1, *global average pooling layer*, dan *softmax layer*.

Input	input_data, label_data
Output	loss / error

```
1.  # Convolution
2.  lin_out = convolve(input_data, weight)
3.  lin_out += bias
4.
5.  # Global Average Pooling
6.  pooled = mean(lin_out, axis=2)
7.
8.  # Softmax
9.  p_y_given_x = softmax(pooled)
10.
11. if training:
12.     #return loss
13.     loss = –mean(label_data * log(p_y_given_x))
14. else:
15.     #return error
16.     prediction = argmax(p_y_given_x)
17.     error = mean(wrong_pred(prediction, label_data))
```

Gambar 3.23 Pseudocode GlobalAverageSoftmax

c. Struktur Umum Model CNN

Bentuk umum model CNN adalah beberapa *ConvPoolDropLayer* pada bagian ekstraksi fitur yang diikuti oleh satu *layer* pada bagian klasifikasi, antara *FullyConnectedSoftmax* dan *GlobalAverageSoftmax.*

Input	jumlah_layer, param_layer, jenis_softmax
Output	-

```
1.  # Ekstraksi Fitur
2.  for i in jumlah_layer:
3.      layer_append(ConvPoolDropLayer(param_layer[i]))
4.
5.  # Klasifikasi
6.  if jenis_softmax = “GA”:
7.      layer_append(GlobalAverageSoftmax())
8.  else:
9.      layer_append(FullyConnectedSoftmax())
```

Gambar 3.24 Pseudocode Model CNN

3.5.2 *Network Architecture*

Performa dari sebuah CNN sangat bergantung pada arsitektur jaringan. Pada tugas akhir ini dibandingkan arsitektur yang dibangun berdasarkan beberapa referensi. Lebar *layer* pada jaringan dibuat *incremental* mengikuti kelipatan K, dimana K adalah variabel ketebalan layer pada jaringan (*k size*).

Dalam mendeskripsikan arsitektur jaringan, akan digunakan notasi arsitektur CNN yang mendeskripsikan jenis, ukuran, dan urutan layer yang digunakan pada arsitektur yang bersangkutan. Notasi disusun dengan urutan dari kiri ke kanan dan menggunakan karakter '-' sebagai pemisah antar layer. Notasi yang digunakan adalah sebagai berikut:

- *x*C*y* : *Convolutional layer* dengan lebar *x* dan ukuran kernel *y*x*y*
- MP*x* : *Max pooling layer* dengan pool size *x* dan pool stride *x*

a. DeepCNet [18]

Arsitektur tersebut diusulkan oleh peserta kompetisi Kaggle terkait klasifikasi citra CIFAR-10 yang mendapat peringkat 1 [28]. Diusulkan untuk mengaplikasikan *convolutional layer* dan *max pooling layer* secara perlahan-lahan menggunakan ukuran terkecil, yaitu kernel berukuran 2x2 dan *pool size* 2x2 dengan *pool stride* 2.

Dalam tugas akhir ini, arsitektur serupa diaplikasikan dengan batasan kedalaman 5 layer. Layer terakhir menggunakan *convolutional layer* dengan kernel size 1x1 dikarenakan ukuran data citra sudah berukuran 1x1 pada layer 4. Hal tersebut berbeda dengan implementasi aslinya dikarenakan pada tugas akhir ini tidak diterapkan transformasi citra menjadi *sparse*.

```
KODE : KC2-MP2-2KC2-MP2-3KC2-MP2-4KC2-MP2-4KC1
```

b. NagadomiNet [23]

Nagadomi adalah peserta kompetisi Kaggle terkait klasifikasi citra CIFAR-10 yang mendapat peringkat 5. Diusulkan struktur konvolusi yang terpusatkan pada pola C3 – C3 – MP, yaitu dua buah *convolutional layer* dengan ukuran kernel 3 berturut-turut yang dilanjutkan dengan sebuah *max pooling layer*.

Dalam tugas akhir ini diberi batasan kedalaman 5 layer sehingga pola tersebut hanya dapat dilakukan dua kali kemudian ditutup dengan sebuah *convolutional layer* dengan ukuran kernel 3.

```
KODE : KC3-KC3-MP2-2KC3-2KC3-MP2-3KC3
```

c. NiN [22]

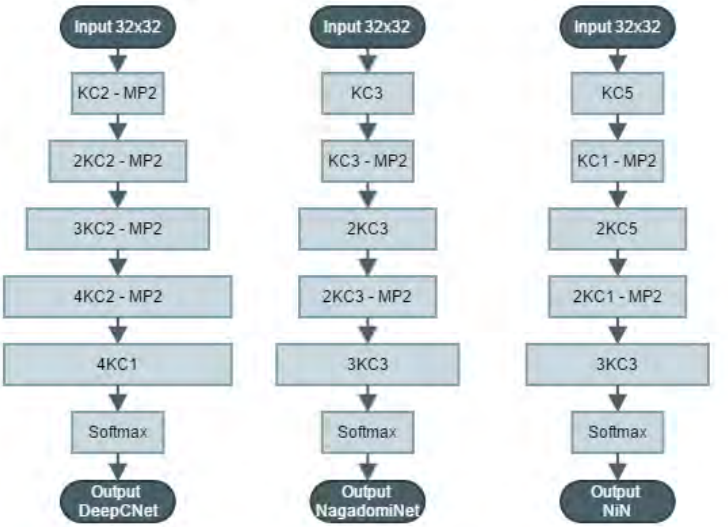
NiN atau Network in Network adalah konsep terkait penambahan sebuah MLP setelah proses konvolusi untuk memperkuat kekuatan transformasi dari layer. Pada tataran praktek, hal tersebut dicapai dengan menggunakan beberapa *convolutional layer* dengan ukuran kernel 1x1 setelah *convolutional layer* utama.

Pada referensi dari pembuat paper, digunakan pola C5 – C1 – C1 – MP [29]. Namun karena batasan layer, pola tersebut kami reduksi menjadi C5 – C1 – MP. Hal tersebut dibuktikan memberi hasil yang tidak terlalu jauh dari pola aslinya [23].

```
KODE : KC5-KC1-MP2-2KC5-2KC1-MP2-3KC3
```

3.6 *Fitting* Model

Pada subbab ini dijelaskan terkait proses *fitting* dari model CNN terhadap dataset CIFAR-10 sehingga didapatkan model terbaik. Penjelasan akan dibagi menjadi tiga bagian, yaitu pelatihan, pengujian, dan regularisasi.



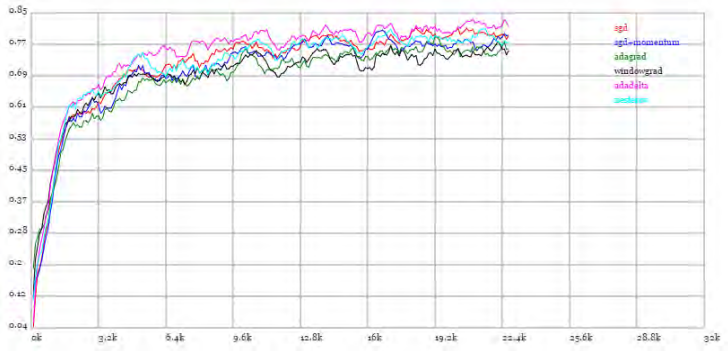
Gambar 3.25 Arsitektur CNN Yang Digunakan

3.6.1 Pelatihan

Pelatihan adalah inti dari seluruh metode *Machine Learning*. Proses tersebut melakukan adaptasi model terhadap data yang ada sehingga model dapat 'mempelajari' data dan memiliki kemampuan prediktif terhadap data serupa.

Pada kasus JST, basis dari proses pelatihan selalu berdasarkan Stochastic Gradient Descent (SGD), namun SGD polos terbukti sangat tidak efektif untuk aplikasi pada umumnya. Telah muncul berbagai metode pengembangan SGD untuk pelatihan pada JST diantaranya adalah metode berbasis momentum, metode berbasis *second order derivative*, dan metode adaptif.

Pada tugas akhir ini digunakan metode adaptif bernama Adadelta karena telah terbukti sangat cepat untuk konvergen pada dataset MNIST [30]. Juga disertakan penjelasan metode SGD + Momentum sebagai perbandingan.



***Gambar 3.26 Perbandingan Metode Pelatihan* [30]**

a. SGD + Momentum

Metode SGD + Momentum adalah pengembangan pertama dari metode SGD polos yaitu dengan menambahkan momentum pada proses pelatihan. Momentum memberikan sifat akselerasi pada proses *update* parameter. Ketika gradien sedang tajam, maka *update* akan semakin cepat, dan sebaliknya. Hal tersebut memungkinkan untuk menghindari *local optima*.

Namun metode tersebut memiliki kelemahan utama yaitu memiliki hyperparameter yang sangat sensitif terhadap performa proses pelatihan, sehingga dalam penggunaan praktis memerlukan *tuning* yang lama.

Input	loss, param, momentum, learn_rate
Output	-

```
1.  grad = gradient(loss, param)
2.  delta = velocity * momentum + learn_rate * grad
3.  param = param + delta
```

Gambar 3.27 Pseudocode SGD + Momentum

b. Adadelta [31]

Adadelta adalah metode SGD adaptif yang memiliki keunggulan utama tidak sensitif terhadap hyperparameter dengan memperkenalkan dua variabel akumulatif baru

yaitu *agrad* dan *adelta*. Namun karena hal tersebut juga, hasil dari Adadelta kemungkinan besar lebih jelek dibandingkan metode momentum yang telah di-*tuning* dengan baik [30]. Untuk alasan waktu, dalam tugas akhir ini digunakan metode Adadelta.

Input	loss, param, rho, eps, adelta, agrad
Output	-

```
1.  grad = gradient(loss, param)
2.  agrad = rho * agrad + (1-rho) * grad * grad
3.  delta = -sqrt((adelta + eps) / (agrad + eps))
4.  adelta = rho * adelta + (1-rho) * delta * delta
5.  param = param + delta
```

Gambar 3.28 Pseudocode Adadelta

3.6.2 Pengujian

Proses pengujian penting untuk diperhatikan karena mempengaruhi ketepatan pengukuran performa dari model. Bagian pengujian akan dibagi menjadi dua bagian yaitu performa dan pembagian data.

a. Performa

Pada tugas akhir ini, performa yang diperhatikan adalah *error* dari model dikarenakan tujuan utama model adalah mendapatkan tingkat kesalahan yang sekecil mungkin. *Error* didefinisikan dengan:

$$error = \frac{prediksi\_salah}{total\_data} \times 100\% \qquad (3.1)$$

Selain *error*, *run time* proses *fitting* model juga menjadi salah satu ukuran performa dalam beberapa skenario uji coba. Hal tersebut dilakukan dengan merekam waktu sebelum dan setelah *fitting* model.

Input	model
Output	run_time

```
1.  start_time = time_now()
2.  model.train()
3.  end_time = time_now()
4.  run_time = (end_time – start_time) / 60
```

Gambar 3.29 Pseudocode Penghitungan Run Time

b. Pembagian Data

Dalam *machine learning*, pembagian data yang benar dapat membantu menghindari kesalahan pengukuran performa model. Data pengujian sebaiknya hanya digunakan untuk pengukuran performa akhir dan tidak digunakan dalam proses *tuning* hyperparameter. Untuk *tuning* hyperparameter digunakan bagian dari data pelatihan sebagai data validasi.

Dataset akan dibagi menjadi 6 bagian. 5/6 data digunakan sebagai data pelatihan dan sisa 1/6 akan digunakan sebagai data pengujian. Pada kasus *tuning* hyperparameter, data pengujian tidak digunakan, namun 1/6 terakhir data pelatihan akan digunakan sebagai data validasi, sedangkan pengujian menggunakan sisa 4/6 dari data. Data validasi pada tugas akhir ini diperlakukan sama persis dengan data pengujian, hanya saja digunakan pada saat *tuning* hyperparameter saja.

Tuning	**Train (4/6)**	**Valid (1/6)**	**X**
Pengujian	**Train (5/6)**		**Test (1/6)**

Gambar 3.30 Pembagian Data Train, Test dan Valid

Data juga diolah sehingga distribusi kelas pada masing-masing bagian dari data adalah proporsional. Hal tersebut untuk menghindari ketimpangan kelas dalam *fitting* model.

Input	input_data
Output	train_data, test_data

```
1.  data_by_class = split_class
2.
3.  if tuning:
4.      for i in nclass:
5.          train_data += data_by_class[i][0:4]
6.          test_data += data_by_class[i][4:5]
7.  else:
8.      for i in nclass:
9.          train_data += data_by_class[i][0:5]
10.         test_data += data_by_class[i][5:6]
```

Gambar 3.31 Pseudocode Pembagian Data

3.6.3 Regularisasi

Regularisasi adalah metode yang diterapkan pada sebuah model untuk menghindari *overfitting*. Overfitting adalah kasus ketika model terlalu cocok dengan data pelatihan, sehingga generalisasi pola berkurang dan kemampuan prediktif model menjadi kurang optimal.

Pada tugas akhir ini diterapkan dua metode regularisasi, yaitu *L2 regularization* dan penambahan *dropout layer*.

a. L2 Regularization

Metode tersebut adalah metode regularisasi yang paling umum digunakan [9]. Metode L2 memberikan penalti yang proporsional dengan kuadrat besar nilai bobot pada *loss function* dikalikan dengan lambda L2. Hal tersebut bertujuan agar tidak ada neuron dominan pada jaringan sehingga proses *fitting* lebih merata. Pada kasus citra, L2 memaksa model untuk melakukan *learning* pada seluruh bagian dari citra.

Input	params
Output	loss

```
1.  # Hitung L2
2.  L2 = 0
3.  for param in params:
4.      L2 += sum(param ** 2)
5.
6.  # Hitung Loss
7.  loss = softmax.loss + lambda * L2
```

Gambar 3.32 Pseudocode L2 Regularization

b. Dropout

Dropout berfungsi dengan mencegah adanya ketergantunagn antar neuron pada jaringan. Nilai dropout pada umumnya adalah 0.5 pada hidden layer, dan 0.2 pada input layer. Namun terdapat beberapa variasi lain dari nilai dropout. Penerapan dropout dapat dilihat pada ***subbab 3.4.1.***

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Sebelum penjelasan implementasi akan ditunjukkan terlebih dahulu lingkungan untuk melakukan implementasi. Penjabaran implementasi akan dibagi menjadi tiga bagian sesuai dengan bab sebelumnya.

4.1 Lingkungan Implementasi

Lingkungan implementasi yang akan digunakan untuk melakukan implementasi adalah menggunakan *Python 2.7* pada sistem operasi *Linux* dengan menggunakan IDE Spyder. Sistem operasi telah diinstal *library* python Theano dan Numpy. Theano berisi tools untuk membangun CNN dan memungkinkan GPU *Acceleration*, dan Numpy berisi tools untuk pengolahan data. Digunakan *library* GPU berupa CUDA sebagai kerangka kerja GPU *acceleration,* dan CuDNN untuk mengoptimasi operasi dasar pada DNN, terutama konvolusi dan subsampling.

4.1.1 Konfigurasi Theano

Theano melakukan GPU *Acceleration* secara otomatis, namun untuk mendapatkan hasil yang optimal perlu diatur beberapa konfigurasi pada Theano. Konfigurasi pada theano terletak pada file *.theanorc* di dalam folder lokal user (~/).

- `floatX = float32`

  Konfigurasi tersebut adalah variabel yang akan diakses pada program untuk penentuan tipe float pada program. Karena theano masih mensupport float32 daam penggunaan GPU, maka digunakan konfigurasi tersebut.

- `device = gpu`
  Konfigurasi tersebut menyatakan device yang menjadi prioritas penggunaan theano. Dalam hal ini digunakan GPU sebagai prioritas utama. Untuk menggunakan CPU, konfigurasi tersebut cukup diubah menjadi `device = cpu`

- `optimizer_including = cudnn`
  Konfigurasi tersebut membuat theano memunculkan error ketika tidak dapat menggunakan CuDNN untuk melakukan *convolution* dan *pooling*.

- `root = /usr/local/cuda`
  Konfigurasi tersebut memberi tahu theano letak dari library CUDA. Hal tersebut diperlukan untuk menggunakan GPU *Acceleration* pada GPU Nvidia.

```
[global]
floatX = float32
device = gpu
optimizer_including = cudnn

[cuda]
root = /usr/local/cuda
```

Kode 4.1 Kode Konfigurasi .theanorc

4.1.2 Pemrograman Simbolik

Untuk melakukan optimasi, theano menggunakan pemrograman simbolik dimana fungsi yang akan diimplementasikan terlebih dahulu disusun menggunakan variabel simbolik, lalu di-*compile* menjadi kode *dynamic c* yang efisien di akhir menggunakan fungsi `theano.function`.

Dengan pemrograman simbolik, tidak perlu dibuat fungsi kompleks dengan input dan output eksplisit pada masing-masing bagian program, melainkan cukup disusun ekspresi simbolik sesuai dengan proses yang terjadi. Fungsi `theano.function` akan menyambungkan alur propagasi variabel simbolik dalam sebuah theano graph lalu melakukan

optimasi graf sebelum mengkonversinya menjadi fungsi sehingga menjadi efisien [15].

```
import theano
a = theano.tensor.vector("a")      # declare symbolic variable
b = a + a ** 10                    # build symbolic expression
f = theano.function([a], b)        # compile function
```

Gambar 4.1 Contoh Pemrograman Simbolik Sederhana

Paradigma pemrograman simbolik cukup berbeda dengan *functional programming* sehingga dalam membuat implementasinya perlu penyesuaian. Ketika run-time, operasi matematis tidak langsung dilakukan melainkan hanya disusun ekspresi simboliknya.

Hal tersebut membuat pengecekan terhadap kondisi data pada tengah operasi sulit untuk dilakukan. Ketika terjadi error, pencarian kode yang menyebabkan error sulit dilakukan karena tidak mungkin untuk mengecek langsung alur propagasi data. Karena hal tersebut, dalam melakukan pemrograman simbolik, alur matematis data perlu disusun dengan rapi dan dengan pengecekan yang ketat untuk menghindari *error*.

4.2 Implementasi Praproses

Pada subbab ini akan dijabarkan implementasi fungsi-fungsi pada tahap praproses, yaitu dari pembacaan data, transformasi data, hingga penyimpanan data akhir. Juga disertakan kode sumber dari masing-masing fungsi. Seluruh fungsi pada praproses tidak menggunakan Theano dan GPU kecuali fungsi **load_shared_data()**.

4.2.1 Implementasi Fungsi load_data()

Fungsi **load_data()** membaca data dari file asli data CIFAR-10 yang berupa file pickle yang terpisah menjadi 6 bagian. Fungsi tersebut membaca masing-masing file terpisah dan mengubahnya menjadi format utama data pada tugas akhir

ini, yaitu dua variabel terpisah *data* yang memiliki empat dimensi dengan format standar bc01 (batch x channel x dim0 x dim1) yang merupakan format input standar pada Theano [32] dan *labels* dengan satu dimensi. Data citra juga akan dikonversi ke dalam range 0 – 1 untuk mempermudah komputasi.

```
def load_data(base_path =
"/home/sindunuragarp/Documents/TA/Resource/cifar-10-batches-py/"):

    batch_name = "data_batch_"
    batch_nums = 6
    data = np.array([]).reshape(0,channels,img_dim,img_dim)
    labels = np.array([])

    for batch_num in xrange(batch_nums):
        # open_data
        batch = unpickle(base_path + batch_name + str(batch_num+1))

        # combine data
        data = np.concatenate((data,np.array(batch["data"]).
               reshape(num_cases_per_batch,channels,img_dim,img_dim)))
        labels = np.concatenate((labels,np.array(batch["labels"])))

    # rgb value is converted to base 0..1
    data = data/255
    return (data,labels)
```

Kode 4.2 Kode Sumber Fungsi load_data()

4.2.2 Implementasi Fungsi extract_data()

Fungsi **extraxt_data()** mengambil data asli yang telah diubah formatnya oleh **load_data()**, lalu melakukan *data splitting* yang dilanjutkan dengan memanggil fungsi **preprocess()**. Karena format data adalah empat dimensi dan terpisah antara data dengan label, maka perlu dilakukan penyesuaian khusus untuk memungkinkan distribusi ulang data. Fungsi mengembalikan data yang telah melalui praproses citra dan dipisah antara data pelatihan dan data pengujian sesuai tahap program yang sedang dijalankan. Fungsi tersebut menerima parameter berupa:

- *amount* : persentase data yang akan diekstrak
- *split* : jumlah potongan data, ditetapkan 6

- *tuning* : program digunakan untuk tuning atau tidak
- *nlabels* : jumlah kelas label pada data

```
def extract_data(amount=1.0, nlabels=10, split=6, tuning=False):
    data, labels = load_data()
    total = data.shape[0]
    size = total/split

    # flatten & combine data
    data_ = data.reshape(data.shape[0],-1)
    labels_ = labels.reshape(labels.shape[0],-1)
    dataset = np.c_[data_,labels_]

    # sort data by labels
    sorted = dataset[dataset[:,-1].argsort()]

    # reduce data
    reduced = []
    size = dataset.shape[0] / nlabels
    taken = size * amount / split

    # uniform random sample. every split has equal class spread
    temp = []
    for j in xrange(nlabels):
        shuffled = sorted[size*j:size*(j+1),:]
        temp.append(shuffled)

    for i in xrange(split):
        for j in xrange(nlabels):
            reduced.append(temp[j][taken*i:taken*(i+1)])

    # assign data
    reduced_ = np.array(reduced)
    if tuning:
        # validation
        train = reduced_[:-nlabels*2].reshape(-1,dataset.shape[-1])
        test = reduced_[-nlabels*2:-nlabels].reshape(-1,dataset.shape[-1])
    else:
        # testing
        train = reduced_[:-nlabels].reshape(-1,dataset.shape[-1])
        test = reduced_[-nlabels:].reshape(-1,dataset.shape[-1])

    # shuffle data
    np.random.shuffle(train)
    np.random.shuffle(test)

    # reshape & split data
    train_set_x = train[:,:data_.shape[1]].reshape(-1,data.shape[1],
                        data.shape[2],data.shape[3])
    train_set_y = train[:,data_.shape[1]:].reshape(-1,)
    test_set_x = test[:,:data_.shape[1]].reshape(-1,data.shape[1],
                        data.shape[2],data.shape[3])
```

```
    # preprocess data
    train_set_x, test_set_x = preprocess((train_set_x, test_set_x))

    # return data
    rval = [(train_set_x,train_set_y),
            (test_set_x,test_set_y)]

    return rval
```

Kode 4.3 Kode Sumber Fungsi extract_data()

4.2.3 Implementasi Fungsi preprocess()

Fungsi **preprocess()** memanggil dua fungsi lain yaitu **global_contrast_normalization()** dan **zca_whitening()** untuk mengolah data sesuai desain praproses data citra.

```
def preprocess(split_data):
    norm_data = global_contrast_normalization(split_data)
    white_data = zca_whitening(norm_data)
    return white_data
```

Kode 4.4 Kode Sumber Fungsi preprocess()

Fungsi **global_contrast_normalization()** menerima data pelatihan dan pengujian lalu melakukan *global contrast normalization.*

```
def global_contrast_normalization((train, test)):
    # find training mean
    mean = train.mean(axis=0)

    # normalize according to training data
    n_train = train - mean
    n_test = test - mean

    return n_train, n_test
```

Kode 4.5 Kode Sumber global_contrast_normalization()

Fungsi **zca_whitening()** melakukan whitening pada data input. Whitening melibatkan operasi matriks dua dimensi sehingga data yang berbentuk empat dimensi perlu di*reshape* terlebih dahulu menjadi dua dimensi. Fungsi tersebut memiliki

satu hyperparameter yang perlu di *tuning* yaitu epsilon yang menentukan derajat penghapusan korelasi antar data.

```
def zca_whitening((train, test),epsilon=0.01):
    # flatten data
    f_train = train.reshape(train.shape[0],-1)
    f_test = test.reshape(test.shape[0],-1)

    # find training ZCA
    sigma = np.dot(f_train.T,f_train) / f_train.shape[0]
    u, s, v = np.linalg.svd(sigma)
    ZCAwhite = np.dot(np.dot(u,np.diag(1./np.sqrt(s + epsilon))),u.T)

    # whiten according to training data
    w_train = np.dot(f_train,ZCAwhite)
    w_test = np.dot(f_test,ZCAwhite)

    # reshape data
    w_train = w_train.reshape(train.shape)
    w_test = w_test.reshape(test.shape)

    return w_train, w_test
```

Kode 4.6 Kode Sumber Fungsi zca_whitening()

4.2.4 Implementasi Fungsi pre_load_data()

Fungsi **pre_load_data()** mengambil data dari fungsi **extract_data()** lalu memanggil fungsi **augment_data()** dan terakhir menyimpan data ke dalam file pickle untuk penggunaan pada tahap *fitting data*. Terdapat parameter baru augment pada fungsi tersebut yang menyatakan apakah program menggunakan *data augmentation* atau tidak.

```
def pre_load_data(amount, split=6, augment=False, tuning=False):
    import cPickle

    train_set, test_set = extract_data(amount=amount,split=split,
                                       tuning=tuning)
    if augment:
        train_set = augment_data(train_set)
    obj = (train_set, test_set)

    f = file('/home/sindunuragarp/Documents/data.save', 'wb')
    cPickle.dump(obj, f, protocol=cPickle.HIGHEST_PROTOCOL)
    f.close()
```

Kode 4.7 Kode Sumber Fungsi pre_load_data()

4.2.5 Implementasi Fungsi augment_data()

Fungsi **augment_data()** menerima input data pelatihan dan melakukan augmentasi data dengan operasi refleksi vertikal, rotasi acak, dan 4 operasi translasi acak.

```
def augment_data((data,labels)):
    # generate new data
    reflected = reflect_data(data)
    rotated = rotate_data(data)
    trans1 = translate_data(data)
    trans2 = translate_data(data)
    trans3 = translate_data(data)

    # concatenate data
    new_data = np.concatenate((data, reflected, rotated, trans1,
                               trans2, trans3))
    new_labels = np.concatenate((labels,labels, labels, labels,
                                 labels, labels))
    return (new_data,new_labels)
```

Kode 4.8 Kode Sumber Fungsi augment_data()

Fungsi **reflect_data()** melakukan refleksi vertikal terhadap data citra dengan membalikkan dimensi ke 4.

```
def reflect_data(data):
    return data[:,:,:,::-1]
```

Kode 4.9 Kode Sumber Fungsi reflect_data()

Fungsi **rotate_data()** melakukan rotasi acak seragam pada data citra dengan standar deviasi sudut 15 derajat.

```
def rotate_data(data, angle=None):
    if angle == None:
        rng = np.random.RandomState()
        scale = 15
        angle = rng.uniform(low=-scale,high=scale)
    return ndimage.rotate(data, angle, axes=(2,3), reshape=False)
```

Kode 4.10 Kode Sumber Fungsi rotate_data()

Fungsi **translate_data()** melakukan translasi acak yang berbeda pada tiap data citra dengan standar deviasi translasi pada sumbu x maupun y sebesar 5 piksel.

```
def translate_data(data):
    img = data.copy()
    rng = np.random.RandomState()
    new_data = []
    for i in xrange(img.shape[0]):
        x = rng.uniform(low=-5, high=5)
        y = rng.uniform(low=-5, high=5)
        new_data.append(ndimage.shift(img[i], (0,y,x)))
    return new data
```

Kode 4.11 Kode Sumber Fungsi translate_dat()

4.2.6 Implementasi Fungsi load_shared_data()

Fungsi **load_shared_data()** digunakan pada tahap *fitting* model untuk memuat data menggunakan pickle dan memasukkannya ke dalam memory GPU menggunakan *shared variable*. Hal tersebut bertujuan agar dataset dapat diakses dengan cepat oleh GPU. Data terlebih dahulu dikonversi menjadi tipe float32 dengan fungsi `numpy.asarray()`, lalu dimasukkan ke *shared variable* menggunakan fungsi `theano.shared()`. Untuk data label, dicasting ulang menjadi data int32 agar sesuai dengan format theano.

```
def load_shared_data():
    train_set, test_set = load_pickled_data()

    def shared_dataset(data_xy, borrow=True):
        data_x, data_y = data_xy
        shared_x = theano.shared(np.asarray(data_x,
                                            dtype=theano.config.floatX),
                                 borrow=borrow)
        shared_y = theano.shared(np.asarray(data_y,
                                            dtype=theano.config.floatX),
                                 borrow=borrow)
        return shared_x, T.cast(shared_y, 'int32')

    test_set_x, test_set_y = shared_dataset(test_set)
    train_set_x, train_set_y = shared_dataset(train_set)
    return [(train_set_x, train_set_y), (test_set_x, test_set_y)]
```

Kode 4.12 Kode Sumber load_shared_data()

4.3 Implementasi Pembangunan Model

Pada subbab ini akan dijabarkan implementasi fungsi-fungsi pada tahap pembangunan model, yaitu dari penyusunan *layer* hingga pembangunan model CNN.

Seluruh layer akan dibuat sebagai class yang berisi inisialisasi parameter dan ekspresi simbolik proses pada layer. Output dari layer akan disimpan dalam variabel simbolik `self.output` sehingga dapat diakses oleh fungsi diluar class. Tidak terdapat fungsi untuk melakukan proses perhitungan karena seluruh perhitungan akan dilakukan oleh mekanisme `theano.function` di program utama.

4.3.1 Implementasi Class ConvPoolDropLayer

ConvPoolDropLayer diimplementasikan dalam bentuk class dengan fungsi inisialisasi class **_init_()** sebagai bagian utama dari class dan dilengkapi dua fungsi untuk menunjang implementasi dropout, yaitu **_dropout()**, dan **drop_on()**.

Class tersebut menerima input berupa data citra dalam format bc01, *random number generator*, ukuran kernel, ukuran citra, serta parameter spesifik proses pada *layer*. Karena pengecekan terhadap dimensi variabel simbolik sulit untuk dilakukan, maka dilakukan pengecekan manual terhadap dimensi data propagasi dengan dimensi kernel agar dipastikan cocok. Kemudian dilakukan inisialisasi parameter *layer* yaitu *weight* dan *bias* sesuai dengan bab sebelumnya.

```
class ConvPoolDropLayer(object):

    def __init__(self, rng, input, filter_shape, image_shape,
                 poolsize, poolstride,
                 pad=0, droprate=0.0, activation):

        # number of feature map between image and kernel must be equal
        assert image_shape[1] == filter_shape[1]
        self._rng = rng
        self._drop = theano.shared(np.cast[theano.config.floatX](1.0),
                                   borrow=True)
```

```
        # inputs to each hidden unit
        fan_in = np.prod(filter_shape[1:])

        # initialize weights = uniform dist
        W_bound = np.sqrt(2. / fan_in)
        self.W = theano.shared(
            np.asarray(
                rng.uniform(low=-W_bound, high=W_bound,
                            size=filter_shape),
                dtype=theano.config.floatX
            ),
            name='W',
            borrow=True
        )

        # initialize biases = 0
        b_values = np.zeros((filter_shape[0],), dtype=theano.config.floatX)
        self.b = theano.shared(value=b_values, name='b', borrow=True)
```

Kode 4.13 Kode Sumber Inisialisasi ConvPoolDropLayer

Pada fungsi **_init_()** juga, dilakukan inisialisasi ekspresi simbolik untuk proses pada *layer*, yaitu proses *convolution*, *max pooling*, dan *dropout*. Pada proses *max pooling* dan *dropout* diberi pengecekan parameter fungsi karena tidak pasti dilakukan tergantung pada parameter spesifik *poolsize* dan *droprate*. Seluruh fungsi tersebut didukung oleh Theano.

```
        # convolve input feature maps with filters
        conv_out = dnn.dnn_conv(
            img=input,
            kerns=self.W,
            border_mode=(pad,pad)
        )

        if poolsize == (1,1):
            pooled_out = conv_out
        else:
            # max pooling
            pooled_out = downsample.max_pool_2d(
                input=conv_out,
                ds=poolsize,
                st=poolstride
            )

        # nonlinearity + bias
        out = activation(pooled_out + self.b.dimshuffle('x', 0, 'x', 'x'))
```

```
        # nonlinearity + bias
        out = activation(pooled_out + self.b.dimshuffle('x', 0, 'x', 'x'))

        # dropout
        if droprate > 0.0:
            # determine training or testing
            dropped = self._dropout(out,p=droprate)
            self.output = T.cast((self._drop * dropped +
                                  (1.-self._drop) * out),
                                  dtype=theano.config.floatX)
        else:
            self.output = T.cast(out, dtype=theano.config.floatX)

        # model parameters
        self.params = [self.W, self.b]
```

Kode 4.14 Kode Sumber Transformasi ConvPoolDropLayer

Fungsi **_dropout()** berfungsi untuk perhitungan operasi *dropout* menggunakan ekspresi simbolik juga, sedangkan fungsi **drop_on()** berfungsi sebagai saklar yang menyalakan atau mematikan proses dropout berdasarkan tahap proses yang sedang ditempuh.

```
    # inverse dropout function
    def _dropout(self, x, p=0.5):
        if p > 0. and p < 1.:
            seed = self._rng.randint(2 ** 30)
            srng = T.shared_randomstreams.RandomStreams(seed)
            mask = (srng.binomial(n=1, p=1.-p, size=x.shape,
                                  dtype=theano.config.floatX))/(1.-p)
            return x * mask
        return x

    # dropout switch
    def drop_on(self, flag):
        if flag:
            self._drop.set_value(1.0)
        else:
            self._drop.set_value(0.0)
```

Kode 4.15 Kode Sumber Dropout ConvPoolDropLayer

4.3.2 Implementasi Class GlobalAverageSM

Class **GlobalAverageSM** diimplementasikan dalam bentuk class dengan fungsi inisialisasi class **_init_()**, dan dua fungsi softmax **negative_log_likelihood()** untuk menghitung *loss*, dan **errors()** untuk menghitung *error*. Pada awal dilakukan pengecekan dan inisialisasi seperti pada **ConvPoolDropLayer**.

```
class GlobalAverageSM(object):

    def __init__(self, input, rng, filter_shape, image_shape):
        assert image_shape[1] == filter_shape[1]

        # initialize weights = uniform dist
        fan_in = np.prod(filter_shape[1:])
        W_bound = np.sqrt(2. / fan_in)
        self.W = theano.shared(
            np.asarray(
                rng.uniform(low=-W_bound, high=W_bound, size=filter_shape),
                dtype=theano.config.floatX
            ),
            name='W',
            borrow=True
        )

        # initialize biases = 0
        b_values = np.zeros((filter_shape[0],), dtype=theano.config.floatX)
        self.b = theano.shared(value=b_values, name='b', borrow=True)
```

Kode 4.16 Kode Sumber Inisialisasi GlobalAverageSM

Pada class tersebut digunakan fungsi linear dari *convolutional layer* dan *global average pooling* untuk melakukan reduksi data. Global average dilaksanakan dengan mudah menggunakan fungsi `theano.tensor.mean`.

```
        # convolve input feature maps with filters
        conv_out = dnn.dnn_conv(
            img=input,
            kerns=self.W
        )

        # layer output : global average
        out = conv_out + self.b.dimshuffle('x', 0, 'x', 'x')
        flat = out.flatten(3)
        self.output = T.mean(flat, axis=2)
```

Kode 4.17 Kode Sumber Fungsi Reduksi GlobalAverageSM

Layer tersebut merupakan *layer* klasifikasi sehingga ditutup dengan operasi softmax lengkap dengan fungsi perhitungan loss dan error. Implementasi softmax sepenuhnya disupport oleh tools dari Theano sehingga mudah diimplementasikan dalam bentuk ekspresi simbolik.

```
        # softmax function
        self.p_y_given_x = T.nnet.softmax(self.output)

        # prediction : class with max confidence
        self.y_pred = T.argmax(self.p_y_given_x, axis=1)

        # model parameters
        self.params = [self.W, self.b]

    def negative_log_likelihood(self, y):
        # mean of inverse log likelihood. Good for batch SGD
        return -T.mean(T.log(self.p_y_given_x)[T.arange(y.shape[0]), y])

    def errors(self, y):
        # returns percentage of mistake in prediction
        return T.mean(T.neq(self.y_pred, y))
```

Kode 4.18 Kode Sumber Softmax GlobalAverageSM

4.3.3 Implementasi Class FullyConnectedSM

Class **FullyConnectedSM** diimplementasikan mirip dengan class **GlobalAverageSM** dengan perbedaan pada inisialisasi dan fungsi reduksi. Karena class tersebut menggunakan *fully connected layer*, maka tidak perlu pengecekan terhadap citra masukan, namun input perlu dipastikan hanya berdimensi dua terlebih dahulu.

```
class FullyConnectedSM(object):

    def __init__(self, input, rng, n_in, n_out):

        # initialize weights = uniform dist
        W_bound = np.sqrt(2. / n_in)
        self.W = theano.shared(
            value=np.asarray(
                rng.uniform(low=-W_bound, high=W_bound, size=(n_in,n_out)),
                dtype=theano.config.floatX
            ),
            name='W',
            borrow=True
        )

        # initialize biases = 0
        self.b = theano.shared(
            value=np.zeros((n_out,), dtype=theano.config.floatX),
            name='b',
            borrow=True
        )
```

Kode 4.19 Kode Sumber Inisialisasi FullyConnectedSM

Fungsi reduksi pada **FullyConnectedSM** menggunakan operasi linear dari *fully connected layer* yang dapat diimplementasikan dengan perkalian matriks biasa.

```
        # linear fully connected layer
        self.output = T.dot(input, self.W) + self.b
```

Kode 4.20 Kode Sumber Fungsi Reduksi FullyConnectedSM

Fungsi-fungsi softmax pada **FullyConnectedSM** sama persis dengan pada **GlobalConnectedSM**.

4.3.4 Implementasi Fungsi Aktivasi

Pada tugas akhir ini diimplementasikan empat macam activation function yaitu relu, leaky, very_leaky, dan tanh. Keempat fungsi tersebut sangat mudah untuk diimplementasikan dengan operasi matriks dasar pada Theano, yaitu operasi `theano.tensor.maximum` untuk jenis relu, serta operasi `tensor.maximum.tanh` untuk jenis tanh.

```
def relu(x):
    return T.maximum(0, x)

def leaky(x):
    return T.maximum(x/100, x)

def very_leaky(x):
    return T.maximum(x/3, x)

def tanh(x):
    return T.tanh(x)
```

Kode 4.21 Kode Sumber Fungsi Aktivasi

4.3.5 Hyperparameter Model CNN

CNN memiliki karakteristik unik yaitu memiliki jumlah parameter dan hyperparameter yang sangat banyak. Dalam pembuatan model CNN, terdapat beberapa hyperparameter yang menjadi masukan fungsi. Hyperparameter tersebut akan dibagi menjadi tiga kelompok besar:

- Hyperparameter jaringan
    - max / min_epochs : batasan epoch proses *fitting*
    - batch_size : ukuran mini batch pada pelatihan
    - k_size : ukuran lebar layer pada jaringan
    - lambda : faktor penalti terhadap *weight*
    - activation : Fungsi aktivasi
- Hyperparameter pelatihan :
    - training_method : Adadelta / SGD Momentum
    - lr : untuk SGD Momentum (Learning Rate)
    - momentum : untuk SGD Momentum
    - rho : untuk Adadelta

    - epsilom : untuk Adadelta
- Hyperparameter arsitektur:
    - nlayers : kedalaman jaringan (jumlah layer)
    - loss : jenis loss layer yang dipakai
        - GA : GlobalAverageSM
        - FC : FullyConnectedSM
    - pad : padding konvolusi tiap layer
    - kerns : ukuran kernel konvolusi tiap layer
    - nkerns : jumlah kernel konvolusi tiap layer
    - pools : poolsize tiap layer
    - stride : poolstride tiap layer
    - rdrops : drop rate tiap layer

4.3.6 Implementasi Pembuatan Model CNN

Pembuatan model dibuat dengan mempertimbangkan kemudahan dalam melakukan konfigurasi arsitektur. Dengan pertimbangan tersebut, program utama disusun menjadi sebuah class **ConvNet** dengan sejumlah hyperparameter input.

Model kemudian akan menginisiasi variabel-variabel jaringan yaitu kumpulan *layer*, kumpulan hyperparameter *layer*, serta dua variabel akumulatif untuk proses pelatihan. Inisialisasi diakhiri dengan memanggil fungsi **prepare_data()** untuk memasukkan dataset ke dalam *shared variable.*

```
class ConvNet(object):
    def __init__(self,
                 # network parameters
                 max_epochs, min_epochs, lambda, batch_size, k_size
                 activation, #tanh, relu, leaky, very_leaky
                 # training parameters
                 training_method = "adadelta", #sgd_momentum, adadelta
                 lr, momentum,
                 rho = 0.95,
                 eps = 1.E-6,
                 # architecture parameters
                 nlayers, pad, kerns, pools, stride, nkerns, rdrops
                 loss, #FC, GA
                 ):
```

```
        # save parameter
        self.lambda = lambda
        self._training_method = training_method
        self._max_epochs = max_epochs
        self._min_epochs = min_epochs
        self._rho = rho
        self._eps = eps
        self._rng = np.random.RandomState()
        self._lr = theano.shared(np.cast[theano.config.floatX](lr))
        self._momentum =
                  theano.shared(np.cast[theano.config.floatX](momentum))

        # prepare network
        self.layers = []
        self.params = []
        self._agrads = []
        self._adeltas = []

        # load data
        self.prepare_data(batch_size)
```

Kode 4.22 Kode Sumber Inisialisasi ConvNet

Setelah inisialisasi, dilakukan pembangunan model CNN menggunakan ekspresi simbolik sesuai parameter arsitektur.

```
        print '... building network model'

        # allocate symbolic variables for the data
        self._x = T.tensor4('x')   # data : 4D tensor with format bc01
        self._y = T.ivector('y')   # labels : 1D vector of [int] labels

        # Construct convolutional pooling layers:
        for lnum in xrange(nlayers):

            # prepare input vars
            kern_dim = kerns[lnum]
            pool_dim = pools[lnum]
            nowkern = nkerns[lnum]
            pool_stride = stride[lnum]

            if lnum == 0:
                input = self._x.reshape((batch_size, channels,
                                         img_dim, img_dim))
                prevkern = channels
                in_dim = img_dim
            else:
                input = self.layers[lnum-1].output
                prevkern = nkerns[lnum-1]
```

```
            # create layer
            layer = ConvPoolDropLayer(
                self._rng,
                input = input,
                droprate = rdrops[lnum],
                image_shape = (batch_size, prevkern, in_dim, in_dim),
                filter_shape = (nowkern, prevkern, kern_dim, kern_dim),
                poolsize = (pool_dim, pool_dim),
                poolstride = (pool_stride, pool_stride),
                pad = pad[lnum],
                activation = activation
            )
            # update output dimension
            out_conv = in_dim - kern_dim + 1 + pad[lnum]*2
            out_pool = out_conv/pool_dim
            in_dim = out_pool

            # update network
            self.layers.append(layer)
            self.params += layer.params
            self._agrads.extend([build_shared_zeros(t.shape.eval(),
                                 'agrad') for t in layer.params])
            self._adeltas.extend([build_shared_zeros(t.shape.eval(),
                                  'adelta') for t in layer.params])
```

Kode 4.23 Kode Sumber Bagian Ekstraksi Fitur

Setelah pembuatan *layer* untuk bagian ekstraksi fitur, dilakukan pembuatan *layer* untuk bagian klasifikasi sesuai dengan tipe *layer* yang didefinisikan pada hyper parameter jaringan. Pada akhir pembuatan *layer* juga dilakukan *update* terhadap keempat variabel jaringan.

```
        # construct softmax layer for classification
        prevkern = nkerns[-1]
        nowkern = nlabels
        if loss == "GA":
            input = self.layers[-1].output
            layer = GlobalAverageSM(input=input,
                                    rng = self._rng,
                                    image_shape=(batch_size, prevkern,
                                                 in_dim, in_dim),
                                    filter_shape=(nowkern, prevkern, 1,1))
        else:
            input = self.layers[-1].output.flatten(2)
            layer = FullyConnectedSM(input=input,
                                     rng = self._rng,
                                     n_in=prevkern * in_dim * in_dim,
                                     n_out=nowkern)
```

```
        # update network
        self.layers.append(layer)
        self.params += layer.params
        self._agrads.extend([build_shared_zeros(t.shape.eval(),
                             'agrad') for t in layer.params])
        self._adeltas.extend([build_shared_zeros(t.shape.eval(),
                              'adelta') for t in layer.params])
```

Kode 4.24 Kode Sumber Bagian Klasifikasi

4.4 Implementasi *Fitting* Model

Pada subbab ini akan dijabarkan implementasi fungsi-fungsi pada tahap *fitting* model, yang meliputi penyiapan data, penyusunan fungsi pelatihan dan pengujian, serta proses iterasi utama *fitting* model.

4.4.1 Implementasi Fungsi prepare_data()

Fungsi **prepare_data()** menyiapkan dataset dalam *shared variable* serta menyiapkan pembagian batch untuk proses pelatihan sesuai parameter *batch_size*.

```
    def prepare_data(self, batch_size):
        print '... preparing data'

        # load data
        datasets = load_shared_data()
        self.train_set_x, self.train_set_y = datasets[0]
        self.test_set_x, self.test_set_y = datasets[1]

        # compute number of minibatches
        self._n_train_batches = self.train_set_x.get_value().shape[0]
        self._n_test_batches = self.test_set_x.get_value().shape[0]
        self._n_train_batches /= batch_size
        self._n_test_batches /= batch_size

        # update batch size
        self._batch_size = batch_size
```

Kode 4.25 Kode Sumber Fungsi prepare_data()

4.4.2 Implementasi Fungsi build_functions()

Fungsi **build_functions()** membangun fungsi jaringan dengan mengkompilasi ekspresi simbolik yang telah dibangun

menggunakan `theano.function`. Pertama dibuat variabel simbolik untuk index dari *mini batch* yang sedang di proses, serta didefinisikan fungsi *loss* dan *error* dari model.

Untuk penyusunan metode SGD, pertama dihitung turunan dari parameter *layer* terhadap *loss*, lalu disusun ekspresi simbolik sesuai metode SGD yang digunakan.

```
    def build_functions(self):
        print '... building network functions'
        index = T.lscalar()  # index to a [mini]batch

        # calculate L2 regularization
        L2 = theano.shared(np.cast[theano.config.floatX](0.))
        for param in self.params:
            L2 += T.sum(param ** 2)

        # the cost function (training) and error function (testing)
        loss  = self.layers[-1].negative_log_likelihood(self._y) \
               + self.weight_decay*L2
        error = self.layers[-1].errors(self._y)

        # create a list of all params and its gradients (training)
        gparams = T.grad(loss, self.params)

        # get update function
        if self._training_method == "sgd_momentum":
            updates = self._get_sgd_momentum_trainer(self.params, gparams)
        else:
            updates = self._get_adadelta_trainer(self.params, gparams)

        # functions for each phase
        train_model = theano.function(
            [index],
            loss,
            updates = updates,
            givens = {
                self._x: self.train_set_x[index * self._batch_size:
                                          (index + 1) * self._batch_size],
                self._y: self.train_set_y[index * self._batch_size:
                                          (index + 1) * self._batch_size]
            },
            allow_input_downcast=True
        )
```

```
        test_model = theano.function(
            [index],
            error,
            givens = {
                self._x: self.test_set_x[index * self._batch_size:
                                         (index + 1) * self._batch_size],
                self._y: self.test_set_y[index * self._batch_size:
                                         (index + 1) * self._batch_size]
            }
        )

        return train_model, test_model
```

Kode 4.26 Kode Sumber Fungsi build_functions()

4.4.3 Implementasi Fungsi Trainer

Disediakan dua fungsi metode SGD untuk digunakan, _get_sgd_momentum_trainer() dan _get_adadelta_trainer(). Kedua fungsi mengembalikan ekspresi simbolik yang mengatur metode *update* parameter pada proses SGD. Namun dalam uji coba hanya digunakan adadelta trainer untuk menghindari *fine tuning* berlebihan.

```
    def _get_adadelta_trainer(self, params, gparams):
        print 'using Adadelta trainer...'
        updates = OrderedDict()

        for agrad, adelta, param, gparam in zip(self._agrads, self._adeltas,
                                                params, gparams):
            newagrad = T.cast((self._rho * agrad + (1 - self._rho) \
                               * gparam * gparam), theano.config.floatX)
            delta = T.cast((- T.sqrt((adelta + self._eps) /
                                     (newagrad + self._eps)) * gparam),
                           theano.config.floatX)
            newadelta = T.cast((self._rho * adelta + (1 - self._rho)
                                * delta * delta), theano.config.floatX)
            updates[agrad] = T.cast(newagrad, theano.config.floatX)
            updates[adelta] = T.cast(newadelta, theano.config.floatX)
            updates[param] = T.cast(param + delta, theano.config.floatX)

        return updates
```

Kode 4.27 Kode Sumber Adadelta Trainer

```
    def _get_sgd_momentum_trainer(self, params, gparams):
        print 'using SGD Momentum trainer...'
        updates = OrderedDict()

        for adelta, param, gparam in zip(self._adeltas, params, gparams):
            updates[adelta] = T.cast((self._momentum * adelta \
                                      + self._lr * gparam),
                                     theano.config.floatX)
            updates[param] = T.cast((param - adelta), theano.config.floatX)

        return updates
```

Kode 4.28 Kode Sumber SGD Momentum Trainer

4.4.4 Implementasi Fungsi train()

Fungsi **train()** adalah fungsi utama tahap *fitting* model dimana dilakukan iterasi proses pelatihan selama sejumlah epoch. Epoch adalah istilah yang digunakan untuk menyebut satu iterasi penuh data pelatihan. Pada iterasi digunakan metode *early stopping* dimana terdapat *max_epoch* dan *min_epoch*. Proses pelatihan akan berjalan minimal sejumlah *min_epoch* kecuali mendapatkan perbaikan error yang signifikan dimana *patience* iterasi akan ditambah.

```
    def train(self):
        # get functions
        train_model, test_model = self.build_functions()

        # early-stopping parameters
        patience = (self.train_set_x.get_value().shape[0]) \
                   /self._batch_size * self._min_epochs
        patience_increase = 2
        improvement_threshold = 0.995
        validation_frequency = min(self._n_train_batches, patience / 2)

        # preset values
        best_loss = np.inf
        best_epoch = 0
        start_time = time.clock()
        epoch = 0
        done_looping = False

        # experiment var
        self.history = []
```

Kode 4.29 Kode Sumber Inisialisasi Fungsi train()

Pada proses iterasi, dilakukan pelatihan untuk setiap iterasi mini batch, sedangkan dilakukan pengujian pada akhir dari setiap epoch. Hal tersebut dilakukan untuk pengecekan *error*. Jika mendapatkan error terkecil, maka hasil dari model pada iterasi tersebut akan disimpan sebagai model terbaik.

Namun pada tugas akhir ini, karena yang dipantau hanya error saja, maka hanya nilai error terbaik beserta iterasi epochnya saja yang disimpan, sedangkan parameter *layer* dari model terbaik tidak disimpan. Riwayat *error* pengujian juga disimpan untuk penggambaran graf capaian pelatihan.

Terdapat beberapa fungsi tambahan pendukung proses iterasi pelatihan yaitu fungsi **_drop_all()** yang berfungsi mematikan seluruh layer dropout untuk membedakan dropout pada proses pelatihan dan pengujian, serta fungsi **print_results()** dan **plot_results()** untuk menampilkan hasil akhir pelatihan.

```
        print '... training'
        while (epoch < self._max_epochs) and (not done_looping):
            epoch = epoch + 1

            for minibatch_index in xrange(self._n_train_batches):
                iter = (epoch-1) * self._n_train_batches + minibatch_index+1
                # training function
                cost_ij = train_model(minibatch_index)

                if (iter) % validation_frequency == 0:
                    # turn off dropout
                    self._drop_all(False)

                    # compute zero-one loss on test set
                    losses = [test_model(i) for i in x
                              range(self._n_test_batches)]
                    current_loss = np.mean(losses) * 100.
                    self.history.append(current_loss)
                    print('epoch %i : patience %i/%i : error %f %%' %
                          (epoch, iter, patience, current_loss))
```

```
                    # if we got the best score until now
                    if current_loss < best_loss:

                        #improve patience if loss improvement is good enough
                        if current_loss < best_loss * improvement_threshold:
                            patience = max(patience,
                                           iter * patience_increase)

                        # save best result
                        print('!!! new best score')
                        best_loss = current_loss
                        best_epoch = epoch

                    # turn on dropout
                    self._drop_all(True)

                if patience <= iter:
                    done_looping = True
                    break

        # Count Time
        end_time = time.clock()

        # Save result
        self._run_time = (end_time - start_time) / 60.
        self._best_loss = best_loss
        self._best_epoch = best_epoch

        # print results
        self.print_results()
        self.plot_results()
```

Kode 4.30 Kode Sumber Iterasi Fungsi train()

```
    def _drop_all(self, flag):
        # turn off dropout in every layer except for last layer
        for i in xrange(len(self.layers)-1):
            self.layers[i].drop_on(flag)
```

Kode 4.31 Kode Sumber Fungsi _drop_all()

```
    def print_results(self):
        print('====================================================')
        print('Optimization complete.')
        print('The code ran for %.2fm' % (self._run_time))
        print('Lowest error of %f %% obtained at epoch %i, ' %
              (self._best_loss, self._best_epoch))
```

Kode 4.32 Kode Sumber Fungsi print_results()

```
    def plot_results(self):
        import matplotlib.pyplot as plt
        plt.plot(self.history, label=self._training_method)

        plt.xlabel('epoch')
        plt.ylabel('error')

        plt.ylim((0,100))
        plt.xlim((0,self._max_epochs))

        plt.title("Training Accuracy")
        plt.legend()
        plt.show()
```

Kode 4.33 Kode Sumber Fungsi plot_results()

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan uji coba yang dilakukan pada aplikasi yang telah dikerjakan serta analisa dari uji coba yang telah dilakukan. Pembahasan pengujian meliputi lingkungan uji coba, skenario uji coba yang meliputi beberapa tahap, serta analisa setiap pengujian.

5.1 Lingkungan Uji Coba

Lingkungan uji coba meliputi perangkat lunak dan perangkat keras. Pada tugas akhir ini, digunakan lingkungan uji coba pada sebuah komputer *desktop* dengan spesifikasi sebagai berikut:

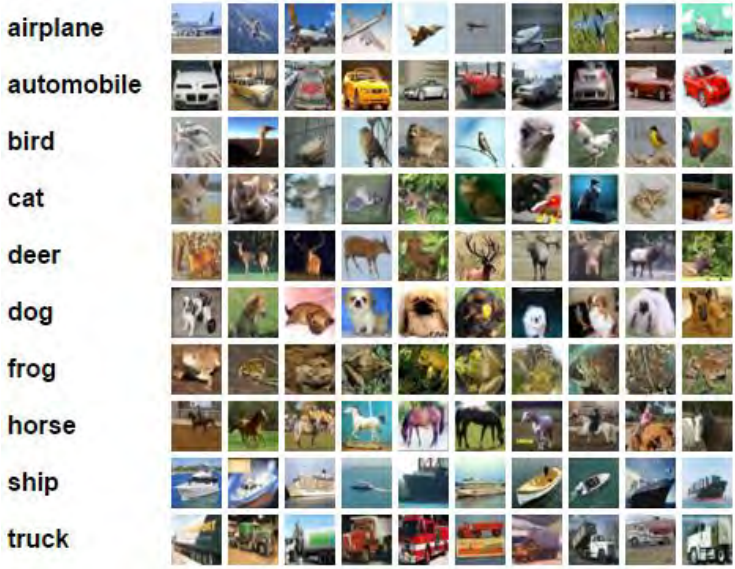
Perangkat Lunak	
OS	Ubuntu 14.10 - 64 Bit
IDE	Spyder
Perangkat Keras	
CPU	Intel Core i7 – 4770K – 4.7 GHz
RAM	16 GB
GPU	Nvidia GTX 750 Ti
GPU Memory	2 GB

Tabel 5.1 Spesifikasi Lingkungan Uji Coba

Spesifikasi tersebut untuk *running* model CNN cukup memadai untuk mendapatkan kecepatan komputasi yang baik, namun GPU Memory masih cenderung kurang. Memory GPU minimal untuk pemakaian CNN standar adalah 3 GB [33], sehingga uji coba arsitektur jaringan pada tugas akhir ini cenderung sangat terbatas.

5.2 Dataset

Pada pengujian tugas akhir ini, digunakan dataset CIFAR-10. CIFAR-10 adalah sebuah dataset citra yang dibuat oleh *Canadian Institute for Advanced Research* (CIFAR). Data pada CIFAR-10 dikumpulkan oleh peneliti di CIFAR yaitu Alex Krizhevsky, Vinod Nair, dan G. Hinton [34].



***Gambar 5.1 Kelas dan Contoh Citra Dataset CIFAR-10* [34]**

Dataset CIFAR-10 terdiri atas 60000 citra berwarna dengan ukuran 32 x 32 piksel. Data pada CIFAR-10 terdiri atas 10 kelas obyek dengan 6000 citra untuk masing-masing kelas. Tiap kelas bersifat *mutually exclusive* dimana tidak ada tumpang tindih antar kelas dan masing-masing citra dipastikan hanya dapat dikategorikan menjadi salah satu kelas. Dataset CIFAR-10 telah lama digunakan sebagai *benchmark* untuk kasus klasifikasi obyek pada citra.

Data citra pada dataset CIFAR-10 untuk masing-masing kelas sangat beragam. Posisi dan sudut pandang obyek citra berbeda-beda. Misal pada kelas *"bird"* atau burung, posisi burung ada yang di tengah, menyamping dan bahkan ada yang hanya terlihat kepalanya saja. Jenis dan ukuran burung yang ada juga beragam jenisnya mulai dari ayam hingga burung unta. Hal tersebut membuat permasalahan pengenalan obyek pada citra menjadi sulit jika dilakukan dengan metode *Feature Engineering* seperti kasus *Computer Vision* pada umumnya.

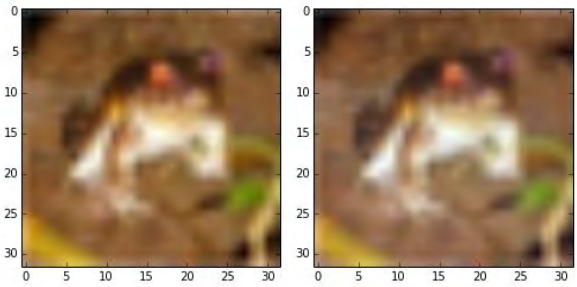
Dataset akan dibagi menjadi data pelatihan, data validasi, dan data pengujian sesuai dengan rancangan pada **subbab 3.5.2.** Pembagian data tidak perlu menggunakan metode *crossvalidation*, dikarenakan ukuran data cukup besar [9].

5.3 Uji Coba Program

Sebelum pengujian utama, akan dilakukan uji coba terhadap proses yang terjadi pada program. Uji coba tersebut meliputi uji coba terhadap tahap praproses, dan uji coba terhadap proses klasifikasi model.

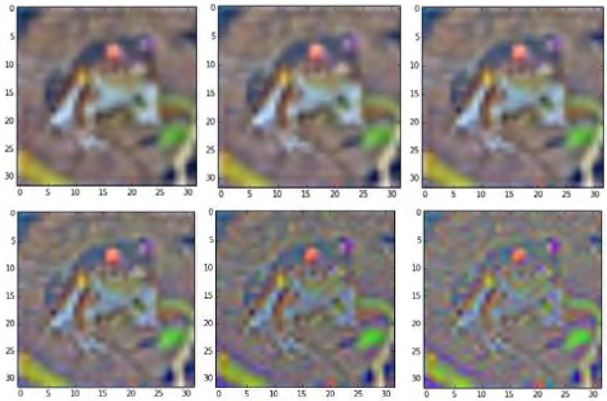
5.3.1 Uji Coba Praproses

Pada uji coba ini dilakukan pengujian terhadap data citra input dan dibandingkan dengan setelah dilakukan praproses.



Gambar 5.2 Citra Asli (kiri) dan Citra Ternormalisasi (kanan)

Proses *Global Contrast Normalization* tidak banyak merubah citra, namun citra menjadi tampak lebih pudar.

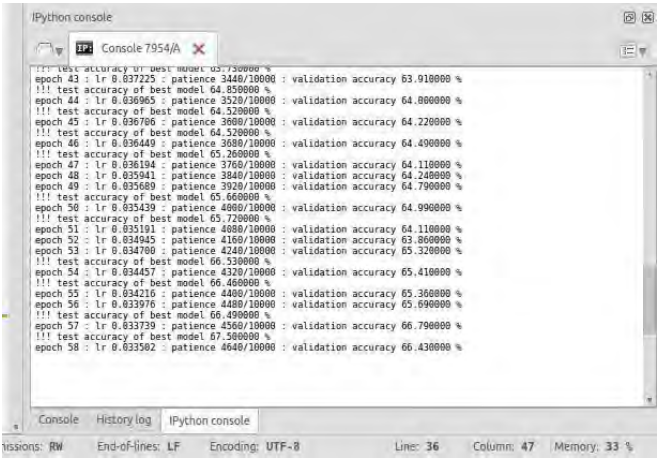


Gambar 5.3 Citra Hasil ZCA 0.2, 0.1, 0.05, 0.01, 0.001, 0.0001

Proses ZCA Whitening diuji dengan berbagai nilai epsilon. Pada ***Gambar 5.3*** ditampilkan hasil terhadap 6 nilai epsilon. Tampak bahwa citra menjadi semakin lebih pudar dengan fitur-fitur yang menonjol seperti bentuk obyek semakin terlihat jelas. Pada nilai epsilon yang rendah, mulai bermunculan banyak *noise* pada gambar. Reduksi terhadap informasi citra yang redundan secara berlebihan akan membuat *noise* lebih tampak pada citra.

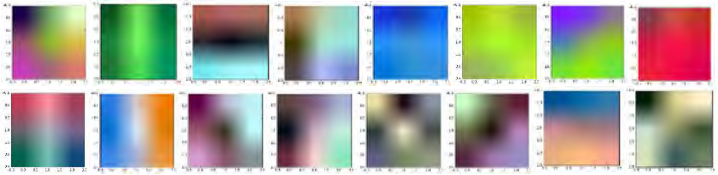
5.3.2 Uji Coba Klasifikasi

Pada uji coba ini akan dilakukan pengujian terhadap pembuatan model klasifikasi. Program klasifikasi akan dieksekusi dan model akhir akan diamati. Pada pengujian ini, akan digunakan model CNN yang cukup kecil dengan lebar jaringan 16 neurons saja dan dijalankan untuk epoch yang terbatas, serta konfigurasi hyperparameter dasar, karena kualitas model belum menjadi fokusan.



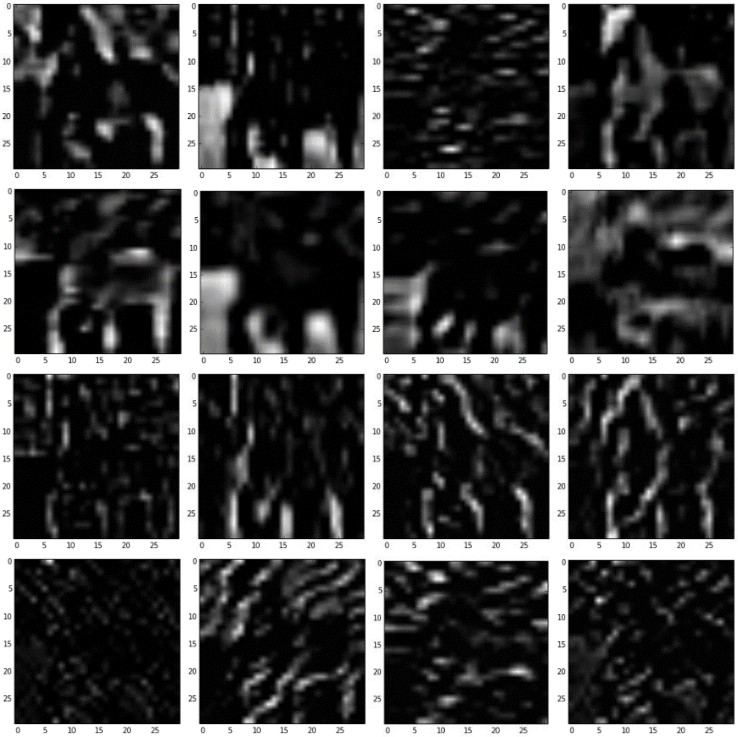
Gambar 5.4 Eksekusi Program Klasifikasi

Pada ***Gambar 5.4*** tampak bahwa program telah berhasil dijalankan dan dilakukan iterasi proses *fitting* model pada tiap epoch dengan nilai *error* yang beragam.



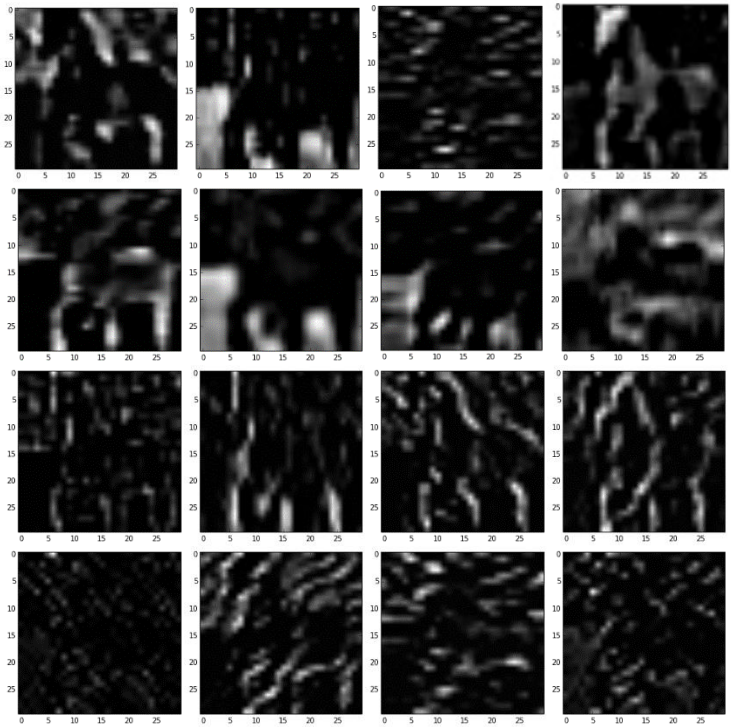
Gambar 5.5 Visualisasi Feature Maps Pada Layer Pertama

Pada ***Gambar 5.5*** divisualisasikan 16 *feature maps* (kernel) pada *layer* pertama menggunakan pencetakan RGB menggunakan ketiga *channel* pada masing-masing *feature maps*. Bobot pada layer pertama berdimensi 16 x 3 x 3 x 3, yaitu terdapat 16 *feature maps*, masing-masing memiliki 3 channel dan berukuran 3x3. Tampak bahwa tiap kernel sangat berbeda. Ada yang berbentuk garis diagonal, ada yang bentuk garis horizontal, dan banyak yang memiliki bentuk abstrak.



Gambar 5.6 Output Layer Pertama

Pada ***Gambar 5.6*** ditampilkan data output dari *layer* pertama untuk data citra kuda yang berjumlah 16 *channel*. Masing-masing *feature maps* pada *layer* pertama tampak mengekstraksi fitur yang berbeda-beda dari citra kuda. Terdapat channel yang mengekstraksi fitur bentuk kuda, ada yang mengekstraksi background, dsb. Ini adalah proses ekstraksi fitur citra otomatis atau *feature learning* pada CNN. Pada output layer kedua pada ***Gambar 5.7*** yang terdiri dari proses konvolusi dan *max pooling,* tampak terekstraksi fitur serupa, namun dengan fitur yang lebih tegas.



Gambar 5.7 Output Layer Kedua

5.4 Skenario dan Evaluasi Pengujian

Pengujian pada tugas akhir ini bertujuan untuk mendapatkan model dengan nilai error terkecil. Hal tersebut dilakukan dengan mengoptimasi hyperparameter pada model. Dikarenakan banyaknya hyperparameter pada CNN, proses pengujian akan dilakukan melalui beberapa tahap pengujian.

1. Uji Performa GPU : Untuk menganalisa pertambahan performa yang didapatkan setelah menggunakan GPU *Acceleration*.

2. Fine Tuning Hyperparameter : Untuk mengoptimasi hyperparameter jaringan agar mendapat model yang berfungsi dengan lebih optimal.
3. Analisa Penggunaan Memori : Untuk mendapatkan penggunaan memori GPU yang optimal sesuai batas memori GPU.
4. Perbandingan Arsitektur : Untuk mencari arsitektur model CNN dengan nilai *error* terkecil.

Pada tahap pengujian 1 sampai 3 digunakan data validasi karena masih bersifat *tuning*, sedang pada tahap pengujian 4 digunakan dataset penuh dengan data pengujian karena output nilai *error* dari tahap ini sudah dianggap nilai akhir dan akan menjadi acuan komparasi model terbaik.

5.4.1 Variabel Tetap Pengujian

Dalam melakukan pengujian, beberapa hyperparameter jaringan dibuat tetap untuk kemudahan pengujian, antara lain:

Tabel 5.2 ***Hyperparameter Tetap Uji Coba***

Parameter	Nilai
Pelatihan	
Metode	Adadelta
Rho	0.95
Epsilon	1.e-6
Arsitektur Jaringan	
Padding	0
Jumlah Layer	5

Hyperparameter pelatihan diambil dari referensi komparasi metode pelatihan pada dataset MNIST [30]. Dikarenakan sifat Adadelta yang tidak sensitif terhadap nilai hyper-parameter, maka dapat langsung digunakan konfigurasi dari referensi. Arsitektur jaringan diberi batasan kedalaman 5

layer serta tidak menggunakan *padding* dikarenakan memori GPU yang terbatas.

Arsitektur jaringan juga akan dibatasi tiga saja sesuai desain arsitektur pada **subbab 3.4.2**, yaitu DeepCNet, NagadomiNet, dan NiN. Untuk selanjutnya, masing-masing arsitektur akan disebut menggunakan namanya. Setelan parameter class ConvNet untuk masing-masing arsitektur sebagai berikut.:

```
# DeepCNet
kerns = [3, 2, 2, 2, 1],
pools = [2, 2, 2, 2, 1],
stride = [2, 2, 2, 2, 1],
nkerns = [k, 2*k, 3*k, 4*k, 4*k]
```

```
# NagadomiNet
kerns = [3, 3, 3, 3, 3],
pools = [1, 2, 1, 2, 1],
stride = [1, 2, 1, 2, 1],
nkerns = [k, k, 2*k, 2*k, 3*k]
```

```
# NiN
kerns = [5, 1, 5, 1, 3],
pools = [1, 2, 1, 2, 1],
stride = [1, 2, 1, 2, 1],
nkerns = [k, k, 2*k, 2*k, 3*k]
```

Gambar 5.8 Parameter Arsitektur Yang Digunakan

5.4.2 Uji Performa GPU

Salah satu tujuan dari tugas akhir ini adalah mencari cara untuk mempercepat proses *fitting* model dari sebuah CNN. Cara mempercepat hal tersebut adalah dengan menggunakan *GPU Acceleration* untuk mempercepat komputasi.

Akan dibandingkan tiga pemanfaatan GPU, yaitu tidak menggunakan GPU, menggunakan GPU dengan CUDA, dan yang terakhir dengan menggunakan GPU disertai library CUDA khusus Deep Learning yaitu CuDNN.

Parameter yang digunakan untuk pengujian tersebut cukup dasar dengan ukuran jaringan yang kecil dikarenakan komparasi utama pada *run time*, yaitu:

Tabel 5.3 Hyperparameter Tetap Pengujian 1

Parameter	Nilai
Hyperparameter Jaringan	
Max / Min Epoch	100 / 100
Batch Size	1000
K Size	16
L2 Lambda	0
ZCA Epsilon	-
Fungsi Aktivasi	ReLU
Arsitektur Jaringan	
Arsitektur	DeepCNet
Loss Layer	FullyConnectedSoftmax

Tabel 5.4 Hasil Pengujian 1

Pemanfaatan Perangkat Keras	Run Time (Menit)	Error
Tanpa GPU (CPU)	118.31	31.16%
GPU CUDA	15.31	30.36%
GPU CUDA + CuDNN	**4.84**	**31.26%**

Dari pengujian dapat dilihat bahwa penggunaan GPU memberi percepatan yang sangat besar meskipun pada model CNN yang kecil. Penggunaan GPU dengan CUDA menunjukkan percepatan sebesar 7.7 kali lipat dari penggunaan CPU saja, sedangkan penggunaan library khusus CuDNN memberikan percepatan sebesar 24.4 kali lipat. Hal tersebut dikarenakan library CuDNN menyediakan implementasi operasi konvolusi dan *max pooling* yang di optimasi untuk digunakan pada CUDA, dua operasi paling memakan waktu pada CNN.

Penggunaan GPU juga tidak menimbulkan pengurangan signifikan pada nilai error dan hanya terjadi fluktuasi kecil.

Untuk pengujian selanjutnya akan digunakan GPU CUDA beserta library CuDNN.

5.4.3 *Fine Tuning Hyperparameter*

Akan dilakukan proses *fine tuning* hyperparameter yang bertujuan untuk mencari konfigurasi yang berfungsi cukup bagus pada dataset CIFAR-10. Pada tugas akhir ini akan dilakukan *fine tuning* pada hyperparameter jaringan *ZCA Whitening*, *L2 Regularization*, fungsi aktivasi jaringan, *batch size*, dan *k size*.

Proses *fine tuning* akan menggunakan konfigurasi jaringan sama dengan pada pengujian 1, namun dengan penyesuaian konfigurasi terbaik pada tiap tahap.

Tabel 5.5 Hyperparameter Tetap Pengujian 2

Parameter	Nilai
Hyperparameter Jaringan	
Max / Min Epoch	50 / 50
Batch Size	500
K Size	16
L2 Lambda	0 <*variable*>
ZCA Epsilon	- <*variable*>
Fungsi Aktivasi	ReLU <*variable*>
Arsitektur Jaringan	
Arsitektur	DeepCNet
Loss Layer	FullyConnectedSoftmax

Agar proses *fine tuning* dapat berjalan dengan cepat, ukuran jaringan dibuat kecil dengan jumlah epoch yang sangat kecil. Untuk memitigasi fluktuasi pada kualitas model, akan diambil nilai rata-rata dari 3 pengujian pada masing-masing konfigurasi hyperparameter.

a. *ZCA Epsilon*

Operasi praproses ZCA Whitening memiliki hyper parameter epsilon yang menentukan derajat pengurangan korelasi citra. Semakin besar nilai epsilon, semakin kecil pengurangan korelasi. Hyper parameter tersebut memiliki efisiensi berbeda untuk tiap dataset sehingga perlu dilakukan *tuning*. Hasil dari pengujian adalah:

Tabel 5.6 Hasil Pengujian 2 – ZCA Whitening

ZCA Epsilon	Rata-rata Run Time (Menit)	Rata-rata Error
Tanpa ZCA	2.77	35.21%
0.2	2.77	34.48%
0.1	2.75	34.27%
0.05	2.75	33.75%
0.01	**2.75**	**33.56%**
0.001	2.75	34.44%

Didapatkan bahwa semakin kecil nilai epsilon, semakin kecil juga nilai *error* dari model. Namun setelah nilai ZCA Epsilon sebesar 0.01, nilai *error* mulai meningkat secara signifikan. Hal tersebut disebabkan nilai epsilon yang terlalu kecil akan membuat noise pada citra terlalu nampak [9] sehingga justru menyebabkan bertambahnya nilai *error*.

Untuk pengujian selanjutnya akan menggunakan nilai ZCA Epsilon terbaik yaitu sebesar 0.01 yang memberikan pengurangan nilai *error* sebesar 1.65% pada pengujian.

b. *L2 Lambda*

Metode regularisasi L2 menghukum adanya nilai bobot yang terlalu besar. Hukuman tersebut proporsional dengan nilai L2 lambda. Nilai tersebut menentukan

besarnya sensitifitas model terhadap nilai bobot. Hasil dari pengujian adalah :

Tabel 5.7 Hasil Pengujian 2 – L2 Lambda

L2 Lambda	Rata-rata Run Time (Menit)	Rata-rata Error
Tanpa L2	2.75	33.56%
0.0001	2.77	33.27%
0.0005	2.76	31.35%
0.00075	2.80	31.29%
0.001	**2.78**	**30.93%**
0.0015	2.83	31.32%
0.01	2.91	67.80%
0.1	2.66	90.00%

Penggunaan regularisasi L2 secara konsisten memberikan nilai error yang lebih kecil. Namun setelah nilai lambda 0.001 terjadi penambahan nilai *error* secara signifikan hingga pada nilai 0.1 kemampuan model untuk melakukan *learning* terhadap data telah hilang dan seluruh data diprediksi sebagai satu kelas.

Nilai L2 Lambda terbaik didapatkan pada nilai 0.001 yang memberikan pengurangan nilai *error* sebesar 2.63% pada pengujian.

c. Fungsi Aktivasi

Pada tugas akhir ini, empat fungsi aktivasi akan dikomparasi, yaitu tanh, ReLU, Leaky ReLU, dan Very Leaky ReLU. Dikarenakan fungsi aktivasi adalah bagian dari jaringan, maka pada proses pengujian fungsi aktivasi, ukuran network akan diperbesar dan jumlah epoch diperlama.

Tabel 5.8 Hyperparameter Pengujian 2 – Fungsi Aktivasi

Parameter	Nilai
Hyperparameter Jaringan	
Max / Min Epoch	100 / 350
Batch Size	1000
K Size	32
L2 Lambda	0.001
ZCA Epsilon	0.01
Fungsi Aktivasi	<variable>
Arsitektur Jaringan	
Arsitektur	DeepCNet
Loss Layer	FullyConnectedSoftmax

Tabel 5.9 Hasil Pengujian 2 – Fungsi Aktivasi

Fungsi Aktivasi	Rata-rata Run Time (Menit)	Rata-rata Error
Tanh	21.09	25.81%
ReLU	36.32	24.60%
Leaky ReLU	**49.68**	**24.00%**
Very Leaky ReLU	45.95	24.29%

Nilai error dari seluruh fungsi ReLU relatif sama, yaitu sekitar 24%, sedangkan fungsi tanh cukup jauh yaitu sebesar 25.81%. Hal tersebut menunjukkan bahwa secara garis besar, metode ReLU menghasilkan hasil yang lebih memuaskan dibandingkan metode tanh. Dalam hal ini fungsi aktivasi Leaky ReLU menghasilkan nilai error terkecil yaitu sebesar 24.00%.

Run time dari masing-masing metode sangat bervariasi dikarenakan digunakan *early stopping* dengan range antara 100-350 epoch. Namun dapat disimpulkan bahwa fungsi tanh dan ReLU lebih mudah terjebak pada local optima karena perbaikan nilai *error* berhenti pada

epoch awal, sedangkan Leaky ReLU memiliki kemampuan untuk menghindari local optima yang lebih baik.

d. *Batch Size*

Batch size adalah ukuran banyak data yang dimasukan ke proses pelatihan dalam satu iterasi. Ukuran batch selain mempengaruhi jumlah data yang disimpan oleh GPU dalam satu waktu, juga mempengaruhi ukuran data yang perlu disimpan GPU pada tiap layer sehingga sangat mempengaruhi penggunaan GPU [9]. Pada pengujian *batch size*, akan dimulai dari ukuran 125 data dan dicoba kelipatannya hingga memory habis.

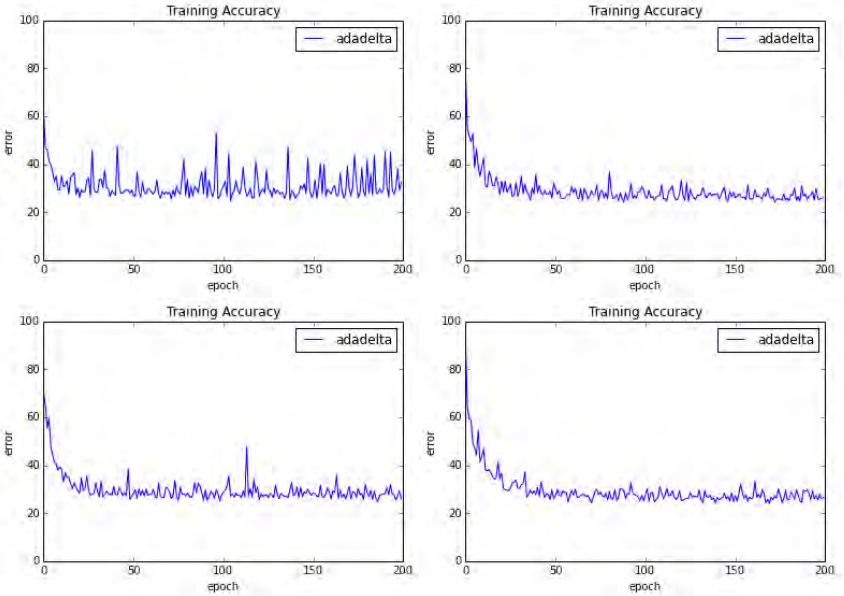
Karena ukuran model dibesarkan sehingga *run time* menjadi lama, serta perbedaan nilai *error* maupun *run time* antar konfigurasi cukup signifikan, maka tiap kasus pengujian cukup dilakukan satu kali saja.

Tabel 5.10 Hasil Pengujian 2 – Batch Size

Batch Size	Run Time (Menit)	Error
125	40.14	24.78%
250	36.51	24.45%
500	**34.28**	**23.99%**
750	33.12	24.91%
1000	33.32	24.15%
1500	32.23	24.92%
2000	32.41	25.53%
3000	Out Of Memory	

Percobaan batch size dengan ukuran K 32 melebihi batasan maksimal GPU Memory pada ukuran 3000. Secara garis besar, run time mengalami percepetan yang cukup signifikan hingga ukuran 750 dimana setelahnya run time cenderung sama. Ukuran batch size yang terlalu kecil atau terlalu besar juga menimbulkan bertambahnya nilai *error*.

Berdasarkan grafik nilai *error* pada ***Gambar 5.9***, dapat dilihat bahwa ukuran *batch size* yang kecil menyebabkan tidak stabilnya proses pelatihan. Pada *batch size* yang semakin besar, proses pelatihan cenderung lebih konsisten.proses pelatihan yang terlalu konsisten akan lebih mudah jatuh pada *local optima*, sehingga lebih baik mengambil nilai batch size yang tidak terlalu kecil dan tidak terlalu besar.



Gambar 5.9 Grafik Batch Size 125, 500, 750, dan 1000

Berdasarkan pengujian tersebut, mengambil ukuran *batch size* antara 250 hingga 750 merupakan pilihan yang terbaik. Untuk pengujian selanjutnya akan dipilih *batch size* dengan nilai *error* terkecil yaitu 500 dan dilakukan pengurangan *batch size* sesuai kebutuhan.

e. *K Size*

K size adalah ukuran variabel *K* yang menentukan lebar jaringan (jumlah feature maps pada tiap *layer*). Ukuran jaringan yang semakin lebar akan menambah kemampuan generalisasi dari model sehingga mencapai *error* yang lebih rendah. Pada tugas akhir ini akan dilakukan analisa terhadap pengaruh lebar jaringan.

Tabel 5.11 Hasil Pengujian 2 – K Size

K Size	Run Time (Menit)	Error
16	19.72	28.39%
32	34.28	23.99%
64	80.74	21.55%
96	151.70	21.01%
128	**227.17**	**20.24%**
160	360.64	20.52%
192	Out Of Memory	

K size melebihi batas memory pada ukuran 192. *Run time* dari model bertambah secara signifikan seiring dengan bertambahnya *k size*. *Error* juga berkurang seiring bertambahnya *k size* namun mulai memburuk diatas ukuran 128. Untuk jaringan dengan batasan kedalaman 5 *layer*, lebar jaringan efektif maksimal adalah 128. Untuk jaringan yang lebih dalam, batasan lebar jaringan efektif akan lebih besar.

Dengan mempertimbangkan *tradeoff* waktu, untuk komparasi jaringan akan digunakan *k size* sebesar 96 karena di atas nilai 96, perbaikan akurasi cenderung kurang signifikan.

5.4.4 *Perbandingan Arsitektur*

Pada pengujian ini akan dilakukan pengujian penuh terhadap model CNN dengan tujuan melakukan komparasi

arsitektur jaringan untuk mendapatkan model terbaik. Pada pengujian ini, akan digunakan data pengujian.

Komparasi jaringan akan dilakukan melalui tiga tahap, yaitu pengujian arsitektur jaringan dan pengujian dropout. Arsitektur jaringan terbaik kemudian akan dijalankan ulang pada dataset teraugmentasi.

Tabel 5.12 Hyperparameter Tetap Pengujian 3

Parameter	Nilai
Hyperparameter Jaringan	
Max / Min Epoch	200 / 350
Batch Size	500
K Size	96
L2 Lambda	0.001
ZCA Epsilon	0.01
Fungsi Aktivasi	Leaky ReLU
Arsitektur Jaringan	
Arsitektur	<variable>
Loss Layer	<variable>
Dropout	<variable>

a. *Arsitektur Jaringan*

Akan dilakukan pengujian penuh menggunakan data lengkap pada ketiga arsitktur CNN yang telah ditentukan di awal. Hyper parameter jaringan akan menggunakan hasil *tuning* sebelumnya sehingga diharapkan dapat menghasilkan *error* seminimal mungkin. Pada tiap Arsitektur akan dicoba kedua kemungkinan *loss layer*, yaitu FullyConnectedSoftmax (FC) dan GlobalAverageSoftmax (GA).

Tabel 5.13 Hasil Pengujian 3 – Arsitektur Jaringan

Arsitektur	Run Time (Menit)	Error
DeepCNet – FC	230.89	21.07%
DeepCNet – GA	168.07	21.04%
NagadomiNet – FC	608.86	23.43%
NagadomiNet – GA	**696.40**	**18.59%**
NiN – FC	288.61	23.68%
NiN – GA	463.51	20.79%

Dari hasil pengujian, NagadomiNet dengan disertai *loss layer* jenis *Global Average Pooling* mencapai *error* terkecil sebesar 18.59%. Hasil tersebut cukup signifikan dibandingkan model lainnya, namun arsitektur tersebut juga memakan waktu *run time* paling lama. Dapat disimpulkan bahwa jenis konvolusi yang menghasilkan *error* terkecil adalah konvolusi dengan ukuran kernel 3x3. Hal tersebut dikarenakan kernel 3x3 merupakan kernel terkecil yang masih menyimpan informasi arah, namun dengan tradeoff waktu pelatihan yang lebih lama.

Global Average Pooling Softmax juga terbukti jauh mengungguli *Fully Connected Softmax* dikarenakan menyimpan informasi spasial dalam melakukan reduksi data. Namun karena menggunakan operasi konvolusi, jenis *softmax* tersebut juga memakan waktu yang lebih lama.

Pada pengujian selanjutnya akan di lakukan tuning lanjutan pada arsitektur NagadomiNet – GA.

b. Dropout

Dropout akan diterapkan kepada model NagadomiNet untuk lebih memperbaiki performa model. Dalam tugas akhir ini akan diterapkan tiga macam dropout yang sering diterapkan dalam JST, yaitu dropout sebesar 0.5 pada

seluruh layer, dropout incremental pada seluruh layer, dan dropout 0.5 setelah operasi *max pooling* saja.

Tabel 5.14 Hasil Pengujian 3 – Dropout

Dropout	Run Time (Menit)	Error
All - 0.5	896.75	19.75%
All - Incremental	**1441.76**	**17.69%**
After Pooling - 0.5	517.18	19.22%

Dari hasil pengujian, didapatkan bahwa kedua metode dropout menggunakan nilai *drop rate* 0.5 justru menghasilkan nilai *error* yang lebih tinggi. Namun pada penerapan dropout incremental dimana nilai *drop rate* bernilai 0.1 pada *layer* pertama dan ditambah hingga bernilai 0.5 pada *layer* kelima, didapatkan pengurangan *error* sebesar 0.9% meskipun dengan *run time* yang bertambah secara sangat signifikan.

Hal tersebut dikarenakan pada CNN, parameter jaringan pada *layer* awal masih sangat sedikit dibandingkan *layer* akhir sehingga penggunaan *drop rate* tinggi pada layer awal justru memberi dampak yang kurang baik pada model. [9]

c. *Pengujian Data Teraugmentasi*

Dilakukan pengujian model akhir menggunakan data teraugmentasi sesuai desain awal yaitu dengan menambahkan data yang direfleksi pada sumbu horizontal, data yang dirotasi secara acak, dan data yang ditranslasi secara acak empat kali. Ketika dijalankan program praproses untuk menyimpan data, terjadi error dikarenakan ukuran data yang terlalu besar.

Kemudian dilakukan augmentasi ulang namun hanya menggunakan data yang direfleksi pada sumbu horizontal.

Karena ukuran data yang semakin besar sehingga terdapat semakin kecil memori di GPU untuk model, maka hyperparameter *k size* terpaksa direduksi menjadi 64. Konfigurasi hyperparameter lainnya sama dengan konfigurasi model terbaik pada pengujian sebelumnya.

Setelah dilakukan pengujian, didapatkan nilai error dari model akhir 19.67%. Bertambahnya nilai *error* dikarenakan pengurangan lebar jaringan yang menyebabkan kemampuan generalisasi oleh model berkurang.

5.5 Analisis Hasil Uji Coba

Dari serangkaian hasil pengujian, didapatkan model akhir dengan nilai *error* pengujian 17.69%. Pelatihan dari model akhir tersebut berjalan selama 1441.76 menit atau sekitar 24 jam. *Run time* dari model akhir ternyata tetap berjalan lama meskipun telah menggunakan akselerasi GPU yang telah mempercepat *run time* pada model kecil sebesar 24 kali lipat. Namun hal tersebut cukup wajar dikarenakan ukuran data yang sangat besar yaitu 60000 data citra berdimensi 3 x 32 x 32.

Penggunaan GPU terbukti sangat membantu sehingga memungkinkan pengujian yang sangat beragam pada jaringan sehingga dapat dilakukan perbaikan error dari awal pengujian sebesar sekitar 30% hingga akhirnya mencapai nilai 17.69%.

Tabel 5.15 Hyperparameter Model Akhir

Parameter	Nilai
Hyperparameter Jaringan	
Max / Min Epoch	200 / 350
Batch Size	500, 300
K Size	96
L2 Lambda	0.001
ZCA Epsilon	0.01

Fungsi Aktivasi	Leaky ReLU
Arsitektur Jaringan	
Arsitektur	NagadomiNet
Loss Layer	GlobalAverageSoftmax
Dropout	Incremental (0.1, 0.2, 0.3, 0.4, 0.5)

Proses *fine tuning hyper parameter* jaringan menggunakan jaringan berukuran kecil merupakan pilihan yang tepat mengingat waktu *run time* model akhir yang sangat lama. Dari proses *fine tuning* didapatkan nilai *hyper parameter* jaringan yang cocok digunakan pada kasus dataset CIFAR-10. Fungsi aktivasi jenis *Rectified Linear Unit* juga terbukti mengungguli fungsi aktivasi tanh. Penggunaan regularisasi dropout tidak selalu memperbaiki kualitas model. Diperlukan nilai dropout yang tepat pada kasus penggunaan JST untuk mendapatkan perbaikan.

5.5.1 Analisis Arsitektur

Untuk menentukan arsitektur pada CNN, tidak ada rumus universal dikarenakan hasil sangat bergantung kepada data yang digunakan. Namun dengan mengamati karakteristik dasar dari masing-masing hyperparameter jaringan serta data, dapat dilakukan estimasi yang cukup baik.

Jika dianalisa, penentuan arsitektur pada CNN adalah menentukan bentuk dari parameter yang digunakan. Parameter pada seluruh macam JST adalah untuk melakukan transformasi non linier pada data kompleks menjadi bentuk yang lebih sederhana sehingga dapat dengan mudah diklasifikasi. Aturan dasar dari JST adalah semakin banyak parameter, maka kemampuan *learning* dan transformasi non linear akan semakin meningkat, namun resiko *overfitting* akan juga meningkat. Pada sebagian besar kasus, resiko *overfitting* ini jauh melebihi peningkatan performa dari model sehingga sebelum tren *deep learning* mulai muncul, kebanyakan akademisi menghakimi

bahwa MLP hanya efektif dengan jumlah hidden layer maksimal 2, serta jumlah neuron yang tidak terlalu besar.

Pada CNN, hal tersebut menjadi lebih kompleks karena yang menentukan bentuk parameter dari model bukan hanya jumlah hidden layer dan jumlah neuron pada tiap *weight* layer, namun juga bentuk kernel pada tiap *convolution* layer serta *subsampling layer* yang digunakan.

Dataset CIFAR-10 adalah dataset citra natural sehingga tentu bentuk datanya kompleks dan tidak linier. Karena hal tersebut, jumlah layer yang sedikit tidak mencukupi. Penggunaan banyak layer sekarang memungkinkan dikarenakan telah berkembangnya teknik regularisasi model untuk menekan resiko *overfitting*, serta inisialisasi *weight* yang baik sehingga neuron pada CNN akan lebih mudah melakukan *learning* yang efektif. Semakin banyak jumlah layer pada model, maka semakin bagus kemampuan transformasi data yang dimiliki model. Pada tugas akhir ini digunakan pilihan hidden layer 5 dengan jumlah neuron 96 dan kelipatannya tergantung arsitektur yang digunakan. Namun hasil akhir penguijan yang didapatkan hanya sebesar 17.69% yang kurang signifikan. Regularisasi yang digunakan kurang memadai dan terjadi *overfitting* karena jumlah parameter yang besar.

Digunakan *convolution layer* agar dapat lebih mudah mengekstraksi informasi spasial dari data citra. *Best practice* dalam menentukan ukuran kernel adalah dengan memilih dimensi ganjil agar kernel memiliki titik tengah dan mengolah informasi spasial sembilan arah. Dengan demikian, informasi lokal yang diolah oleh proses konvolusi lebih mewakili struktur citra. Pengecualian dari hal tersebut adalah pada kasus DeepCNet yang mengikuti paper "Spatially Sparse CNN" dan "Fractional Max Pooling" yang menggunakan kernel ukuran 2x2 dengan sukses. Hal tersebut dikarenakan pada paper digunakan transformasi data untuk mengubah citra cifar menjadi representasi yang *sparse* sehingga kernel 2x2 tetap dapat mengambil informasi yang sebanding. Pada tugas akhir

ini digunakan arsitektur serupa tanpa transformasi data yang sama karena kendala GPU sehingga didapatkan hasil yang paling buruk menggunakan arsitektur DeepCNet.

Dengan ukuran citra yang relatif kecil yaitu 32x32 saja, pilihan ukuran kernel yang ideal cukup terbatas. Ukuran kernel yang terlalu besar tidak akan mampu mendeteksi fitur lokal citra, sedangkan ukuran citra yang terlalu kecil tidak akan mengolah informasi yang cukup. Jumlah *convolutional layer* juga perlu dipertimbangkan karena dengan jumlah layer yang besar, pengenalan fitur menjadi bertahap sehingga digunakan ukuran kernel yang lebih kecil pada tiap *layer*. Ukuran kernel 7 keatas kemungkinan sudah terlalu besar pada data 32x32 dan jumlah layer 5 karena menimbulkan reduksi citra yang terlalu cepat sehingga generalisasi dari model akan kurang. Oleh karena itu pada tugas akhir ini digunakan ukuran kernel masing-masing 5x5 – 1x1 (Network in Network), dan 3x3 – 3x3 (NagadomiNet). Dari pengujian didapatkan NagadomiNet dengan pilihan kernel 3x3 memiliki hasil yang lebih baik. Hal tersebut sejalan dengan hasil dari paper lain pada **Tabel 5.16** dimana ukuran kernel yang digunakan antara 2x2, 3x3, dan 5x5, dengan arsitektur Nagadomi asli mendapatkan hasil yang lebih baik dari Network in Network asli.

Tabel 5.16 Perbandingan Arsitektur CNN

Sumber	Akurasi	Depth : Arsitektur
Fractional Max Pooling [35]	96.53%	`14 : 12(160kC2-FMP)-C2-C1`
Nagadomi [23]	94.15%	`10 : 2(96kC3-96kC3-MP2)-4(256C3)-MP2-FC1024-FC1024`
Spatially Sparse Convolutional Neural Network [18]	91.63%	`5 : 300C2-MP2-600C2-MP2-900C2-MP2-1200C2-MP2-1500C2-MP2`

Network In Network [22]	91.20%	8 : 192C5-160C1-96C1-MP3/2-192C5-192C1-192C1-MP3/2-192C5-192C1
All Convolutional Network [14]	90.92%	8 : 96C3-96C3-96C3-192C3-192C3-192C3-192C3-192C1
DropConnect [36]	90.68%	4 : 64C5-MP3/2-64C5-MP3/2-64LC3-32LC3
Maxout Networks [21]	90.65%	4 : 96C5-MP3/2-192C5-MP3/2-192C5-MP3/2-FC500
Multi-Column Deep Neural Networks [3]	88.79%	6 : 300C3-MP2-300C2-MP2-300C3-MP2-300C2-MP2-FC300-FC100
Deep Learning using Linear Support Vector Machines [37]	88.10%	3 : 32C5-MP2-64C5-MP2-FC3072
Stochastic Pooling [38]	84.87%	3 : 64C5-SP3/2-64C5-SP3/2-64C5-SP3/2
Dropout [27]	84.40%	4 : 64C5-MP3/2-64C5-MP3/2-64C5-MP3/2-16LC3
Hasil Tugas Akhir	**82.31%**	**5 : 96C3-96C3-MP2-192C3-192C3-MP2-288C3**
High-Performance Neural Networks [39]	80.49%	5 : 300C3-MP3-300C3-MP2-300C3-MP2-FC300-FC100

5.5.2 Evaluasi Model Akhir

Meskipun telah terjadi perbaikan nilai *error* yang signifikan pada serangkaian pengujian, hasil akhir sebesar error 17.69%, atau akurasi 82.31% masih cukup rendah

dibandingkan hasil klasifikasi CNN lain pada dataset serupa seperti di **Tabel 5.17**. Bahkan terdapat CNN dengan kedalaman yang lebih kecil namun hasil yang lebih baik. Hal tersebut menunjukkan bahwa model yang digunakan pada tugas akhir ini masih belum optimal dan penggunaan regularisasi yang lebih baik dapat meningkatkan akurasi. Namun jika diamati pada **Tabel 5.16**, penggunaan jumlah layer yang banyak dapat menekan akurasi menjadi sangat tinggi, sehingga jumlah layer memang memiliki dampak positif terhadap akurasi model, namun dengan catatan memerlukan penanganan yang baik pada regularisasi model serta pada pelatihan menggunakan *back-propagation*.

Hasil uji coba yang didapatkan juga tidak berbeda jauh dengan metode non CNN. Pendekatan Unsupervised Feature Learning (UFL) dengan metode berbasis Sparse Coding maupun metode berbasis Convolution juga dapat mencapai angka 82%. Metode UFL memiliki kelebihan dapat dilakukan pada kasus dimana data terlabel yang tersedia tidak banyak, serta cenderung memiliki proses pelatihan model yang lebih cepat. Sparse Coding sendiri memiliki kekurangan yaitu proses pengujian cenderung lebih lama dan model yang kurang *scalabele* jika dibandingkan dengan model convolution, baik pada CNN maupun UFL, sehingga alternatif metode convolution lebih tepat untuk digunakan.

Metode UFL berbasis convolution bekerja sama dengan CNN, namun proses *feature learning* dan klasifikasi dipisah. Proses *feature learning* dilakukan menggunakan pendekatan unsupervised seperti K-Means, autoencoder, dan Deep Belief Network. Sedangkan proses klasifikasi menggunakan arsitektur yang sama dengan CNN yaitu dengan *convolution layer* yang berlapis-lapis, namun digunakan kernel yang telah dihasilkan oleh proses UFL. Keunggulan utama adalah tidak memerlukan data berlabel, serta proses pelatihan cenderung lebih cepat. Namun tentu metode unsupervised tidak memiliki akurasi sebesar metode supervised.

Karena hal tersebut, *drawback* dari CNN hanyalah pada lamanya proses pelatihan dan juga diperlukannya data berlabel, sedangkan waktu pengujian yang merupakan pengguaan aplikatif utama dari model, sama cepatnya dengan metode UFL berbasis convolution. Pada kasus dimana data berlabel tidak sebanyak CIFAR-10, maka penggunaan metode UFL lebih memadai. Juga pada kasus dimana kekuatan komputasi kurang, maupun waktu yang tersedia untuk pelatihan terbatas, maka metode UFL lebih cocok digunakan (Metode K Means memiliki waktu pelatihan yang sangat cepat serta model yang sangat *scalabele*). Namun selain kasus diatas, model CNN tetap lebih unggul, namun dengan catatan diperlukan regularisasi model yang memadai agar hasil akhir lebih baik.

Tabel 5.17 Perbandingan Metode Klasifikasi CIFAR-10

Sumber	Akurasi	Jenis
Fractional Max Pooling [35]	96.53%	CNN - Depth 14
Nagadomi [23]	94.15%	CNN - Depth 10
Spatially Sparse Convolutional Neural Network [18]	91.63%	CNN - Depth 5
Network In Network [22]	91.20%	CNN - Depth 8
All Convolutional Net [14]	90.92%	CNN - Depth 8
DropConnect [36]	90.68%	CNN - Depth 4
Maxout Networks [21]	90.65%	CNN - Depth 4
Multi-Column Deep Neural Networks [3]	88.79%	CNN - Depth 6
Deep Learning using Linear Support Vector Machines [37]	88.10%	CNN - Depth 3
Stochastic Pooling [38]	84.87%	CNN - Depth 3
Dropout [27]	84.40%	CNN - Depth 4
Representation Learning with Nonnegativity Constraints [40]	82.90%	Sparse Coding

Learning Invariant Representations with Local Transformations [41]	82.20%	Sparse Coding
Hasil Tugas Akhir	**82.31%**	**CNN**
Convolutional Kernel Networks [42]	82.18%	CNN - Depth 2
Learning Feature Representations with K-means [43]	82.00%	Convolution
Sparse Coding and Vector Quantization [44]	81.50%	Sparse Coding
High-Performance Neural Networks [39]	80.49%	CNN - Depth 5
Hierarchical Kernel Descriptors [45]	80.00%	Sparse Coding
An Analysis of Single-Layer Networks in Unsupervised Feature Learning [7]	79.60%	Convolution
Convolutional Deep Belief Networks [46]	78.90%	Convolution
Spike-and-Slab Sparse Coding [47]	78.80%	Sparse Coding
PCANet [48]	78.67%	Convolution
Learning Separable Filters [49]	76.00%	Sparse Coding
Kernel Descriptors [50]	76.00%	Sparse Coding
Semiparametric Latent Variable Models [51]	75.82%	Convolution
Improved Local Coordinate Coding using Local Tangents [52]	74.50%	Sparse Coding
An Analysis of the Connections Between Layers of Deep Neural Networks [53]	73.20%	Convolution
Tiled convolutional neural networks [54]	73.10%	CNN - Depth 2
Factorized Third-Order Boltzmann Machines [55]	71.00%	Convolution

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap pembuatan model, dapat diambil kesimpulan sebagai berikut:

1. Telah berhasil dibangun model *Convolutional Neural Network* untuk kasus klasifikasi obyek pada citra menggunakan *library* Theano pada bahasa Python. Library Theano terbukti sebagai *tools* pembuatan model jaringan saraf tiruan yang memberi keleluasaan dalam mengatur desain internal model. Theano juga memiliki dokumentasi yang sangat baik sehingga mempermudah eksplorasi *library*.
2. Penggunaan *Graphical Processing Unit* dalam komputasi dengan CUDA telah menghasilkan percepatan proses *fitting* model hingga 24.4 kali lipat ketika diberi library tambahan CuDNN.
3. Nilai *error* terkecil 17.69% didapatkan dengan menggunakan arsitektur NagadomiNet yang menggunakan repetisi dari struktur C3–C3–MP, yaitu dua *convolutional layer* berturut-turut dengan ukuran kernel 3x3 dan lebar layer sama yang kemudian ditutup dengan sebuah *max pooling layer*. Juga diterapkan *dropout* dengan nilai *drop rate* incremental dari nilai 0.1 hingga 0.5 pada masing-masing *layer*. Namun model tersebut masih belum optimal dan perlu perbaikan pada regularisasi model serta proses pelatihan model.

6.2 Saran

Saran yang diberikan untuk pengembangan aplikasi ini adalah:

1. Pengembangan proses klasifikasi pada model CNN untuk *multiple object recognition*.
2. Menggunakan GPU dengan memori yang lebih tinggi sehingga dapat membuat CNN dengan kedalaman *convolution layer* sebesar 7 ke atas [14] [22]. Pada sebuah referensi disebutkan bahwa memori standar untuk pembuatan model CNN adalah 4GB [33].
3. Melakukan eksplorasi terhadap metode proses *subsampling*. Dapat dilakukan pengujian terhadap parameter *max pooling* seperti poolsize dan poolstride. Juga telah muncul beberapa referensi ilmiah terkait pendekatan *subsampling* baru, diantaranya Fractional Max Pooling [35] dan All Convolutional Nets [14].
4. Melakukan *fine tuning* metode pelatihan SGD jenis momentum pada proses pelatihan model CNN. Disebutkan bahwa penggunaan *momentum* dan *learning rate* yang tepat selalu menghasilkan hasil yang lebih baik dibandingkan metode adadelta [30].

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap pembuatan model, dapat diambil kesimpulan sebagai berikut:

1. Telah berhasil dibangun model *Convolutional Neural Network* untuk kasus klasifikasi obyek pada citra menggunakan *library* Theano pada bahasa Python. Library Theano terbukti sebagai *tools* pembuatan model jaringan saraf tiruan yang memberi keleluasaan dalam mengatur desain internal model. Theano juga memiliki dokumentasi yang sangat baik sehingga mempermudah eksplorasi *library*.
2. Penggunaan *Graphical Processing Unit* dalam komputasi dengan CUDA telah menghasilkan percepatan proses *fitting* model hingga 24.4 kali lipat ketika diberi library tambahan CuDNN.
3. Nilai *error* terkecil 17.69% didapatkan dengan menggunakan arsitektur NagadomiNet yang menggunakan repetisi dari struktur C3–C3–MP, yaitu dua *convolutional layer* berturut-turut dengan ukuran kernel 3x3 dan lebar layer sama yang kemudian ditutup dengan sebuah *max pooling layer*. Juga diterapkan *dropout* dengan nilai *drop rate* incremental dari nilai 0.1 hingga 0.5 pada masing-masing *layer*. Namun model tersebut masih belum optimal dan perlu perbaikan pada regularisasi model serta proses pelatihan model.

6.2 Saran

Saran yang diberikan untuk pengembangan aplikasi ini adalah:

1. Pengembangan proses klasifikasi pada model CNN untuk *multiple object recognition*.
2. Menggunakan GPU dengan memori yang lebih tinggi sehingga dapat membuat CNN dengan kedalaman *convolution layer* sebesar 7 ke atas [14] [22]. Pada sebuah referensi disebutkan bahwa memori standar untuk pembuatan model CNN adalah 4GB [33].
3. Melakukan eksplorasi terhadap metode proses *subsampling*. Dapat dilakukan pengujian terhadap parameter *max pooling* seperti poolsize dan poolstride. Juga telah muncul beberapa referensi ilmiah terkait pendekatan *subsampling* baru, diantaranya Fractional Max Pooling [35] dan All Convolutional Nets [14].
4. Melakukan *fine tuning* metode pelatihan SGD jenis momentum pada proses pelatihan model CNN. Disebutkan bahwa penggunaan *momentum* dan *learning rate* yang tepat selalu menghasilkan hasil yang lebih baik dibandingkan metode adadelta [30].

DAFTAR PUSTAKA

[1] Y. LeCun, "Handwritten Digit Recognition with a Back-Propagation Network," 1990.

[2] J. Schmidhuber and A. Graves, "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks," 2009.

[3] D. C. Ciresan, U. Meier and J. Schmidhuber, "Multi-column Deep Neural Networks for Image Classification," *IEEE Conf. on Computer Vision and Pattern Recognition CVPR 2012.*

[4] R. Benenson, "What is the class of this image?," [Online]. Available: http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html. [Accessed 22 Desember 2014].

[5] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient Based Learning Applied to Document Recognition," 1998.

[6] Wikipedia, "Feature Learning," [Online]. Available: http://en.wikipedia.org/wiki/Feature_learning. [Accessed 24 Desember 2014].

[7] A. Coates, H. Lee and A. Y. Ng, "An Analysis of Single-Layer Networks in Unsupervised Feature Learning," 2011.

[8] K. Fukushima, "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position," *Biological Cybernetics,* 1980.

[9] A. Karpathy, "CS231n Convolutional Neural Networks for Visual Recognition," Stanford University, [Online]. Available: http://cs231n.github.io/. [Accessed 17 Mei 2015].

[10] LISA Lab, "Deep Learning Tutorial," [Online]. Available: http://deeplearning.net/tutorial/contents.html. [Accessed 29 Mei 2015].

[11] D. Stathakis, "How Many Hidden Layers And Nodes?," *International Journal of Remote Sensing,* 2008.

[12] Stanford University, "An Introduction to Convolutional Neural Network," Vision Imaging Science and Technology Lab, Stanford University, [Online]. Available: http://white.stanford.edu/teach/index.php/An_Introduction_to_Convolutional_Neural_Networks. [Accessed 24 Desember 2014].

[13] A. Ng, J. Ngiam, C. Yu Foo, Y. Mai and C. Suen, "Unsupervised Feature Learning and Deep Learning Tutorial," Stanford University, [Online]. Available: http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial. [Accessed 07 Mei 2015].

[14] J. T. Springenberg, A. Dosovitskiy, T. Brox and M. Riedmiller, "Striving For Simplicity: The All Convolutional Net," *ICLR 2015,* 2015.

[15] LISA Lab, "Theano 0.7 Documentation," University of Montreal, [Online]. Available: http://deeplearning.net/software/theano. [Accessed 05 Mei 2015].

[16] C. Olah, "Neural Networks, Manifolds, and Topology," [Online]. Available: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/. [Accessed 13 Mei 2015].

[17] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," 2012.

[18] B. Graham, "Spatially-Sparse Convolutional Neural Networks," 2015.

[19] F. Brunet, "Hyperparameters," [Online]. Available: http://www.brnt.eu/phd/node14.html. [Accessed 16 Juni 2015].

[20] Nvidia, "What Is GPU Accelerated Computing?," Nvidia, [Online]. Available: http://www.nvidia.com/object/what-is-gpu-computing.html. [Accessed 24 Desember 2014].

[21] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville and Y. Bengio, "Maxout Networks," *arXiv,* 2013.

[22] M. Lin, Q. Chen and S. Yan, "Network In Network," 2014.

[23] Nagadomi, "Code for Kaggle-CIFAR10 Competition. 5th place.," [Online]. Available: https://github.com/nagadomi/kaggle-cifar10-torch7. [Accessed 21 Mei 2015].

[24] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley and Y. Bengio, "Theano : New Features and Speed Improvements," *NIPS,* 2012.

[25] N. Tasfi, "Image Scaling using Deep Convolutional Neural Networks," Flipboard, [Online]. Available: http://engineering.flipboard.com/2015/05/scaling-convnets/. [Accessed 15 Juni 2015].

[26] K. He, X. Zhang, S. Ren and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *arXiv,* 2015.

[27] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors," *arXiv,* 2012.

[28] H. Jacob van Veen, "CIFAR-10 Competition Winners: Interviews with Dr. Ben Graham, Phil Culliton, &

Zygmunt Zając," Kaggle, [Online]. Available: http://blog.kaggle.com/2015/01/02/cifar-10-competition-winners-interviews-with-dr-ben-graham-phil-culliton-zygmunt-zajac/. [Accessed 15 Mei 2015].

[29] M. Lin, "Network in Network Cifar-10," [Online]. Available: https://gist.github.com/mavenlin/e56253735ef32c3c296d. [Accessed 23 Mei 2015].

[30] A. Karpathy, "ConvNetJS Trainer demo On MNIST," Stanford University, [Online]. Available: http://cs.stanford.edu/people/karpathy/convnetjs/demo/trainers.html. [Accessed 27 Mei 2015].

[31] M. D. Zeiler, "Adadelta: An Adaptive Learning Rate Method".

[32] S. Dieleman, "3x Faster Convolutions in Theano," [Online]. Available: http://benanne.github.io/2014/04/03/faster-convolutions-in-theano.html. [Accessed 23 Mei 2015].

[33] T. Dettmers, "Which GPU(s) to Get for Deep Learning: My Experience and Advice for Using GPUs in Deep Learning," [Online]. Available: https://timdettmers.wordpress.com/2014/08/14/which-gpu-for-deep-learning/. [Accessed 5 Juni 2015].

[34] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2009.

[35] B. Graham, "Fractional Max Pooling," *arXiv,* 2015.

[36] L. Wan, M. Zeiler, S. Zhang, Y. LeCun and R. Fergus, "Regularization of Neural Networks using DropConnect," 2013.

[37] Y. Tang, "Deep Learning Using Linear Support Vector Machines," 2013.

[38] M. Zeiler and R. Fergus, "Stochastic Pooling For Regularization of Deep Convolutional Neural Networks," 2013.

[39] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella and J. Schmidhuber, "High-Performance Neural Networks for Visual Object Classification," 2011.

[40] T.-H. Lin and H. T. Kung, "Stable and Efficient Representation Learning with Nonnegativity Constraints," 2014.

[41] K. Sohn and H. Lee, "Learning Invariant Representations with Lo cal Transformations," 2012.

[42] J. Mairal, P. Koniusz, Z. Harchaoui and C. Schmid, "Convolutional Kernel Networks," 2014.

[43] A. Y. Ng and A. Coates, "Learning Feature Representations with K-means," 2012.

[44] A. Coates and A. Y. Ng, "The Importance of Encoding Versus Training with Sparse Coding," 2011.

[45] L. Bo, K. Lai, X. Ren and D. Fox, "Object Recognition with Hierarchical Kernel Descriptors," 2011.

[46] A. Krizhevsky, "Convolutional Deep Belief Networks on CIFAR-10," 2010.

[47] I. J. Goodfellow, A. Courville and Y. Bengio, "Spike-and-Slab Sparse Coding for Unsupervised Feature Discovery," 2012.

[48] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng and Y. Ma, "PCANet: A Simple Deep Learning Baseline for Image Classification?," 2014.

[49] R. Rigamonti, V. Lepetit and P. Fua, "Learning Separable Filters," 2012.

[50] L. Bo, X. Ren and D. Fox, "Kernel Descriptors for Visual Recognition," 2010.

[51] J. Snoek, R. P. Adams and H. Larochelle, "Semiparametric Latent Variable Models for Guided Representation," 2011.

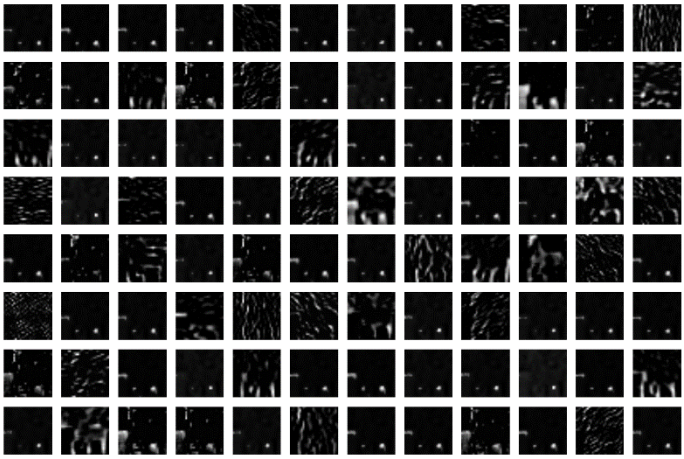
[52] K. Yu and T. Zhang, "Improved Local Coordinate Coding using Local Tangents," 2010.

[53] E. Culurciello, A. Dundar, J. Jin and J. Bates, "An Analysis of the Connections Between Layers of Deep Neural Networks," 2013.

[54] Q. V. Le, J. Ngiam, Z. Chen, D. Chia, P. W. Koh and A. Y. Ng, "Tiled Convolutional Neural Networks," 2010.

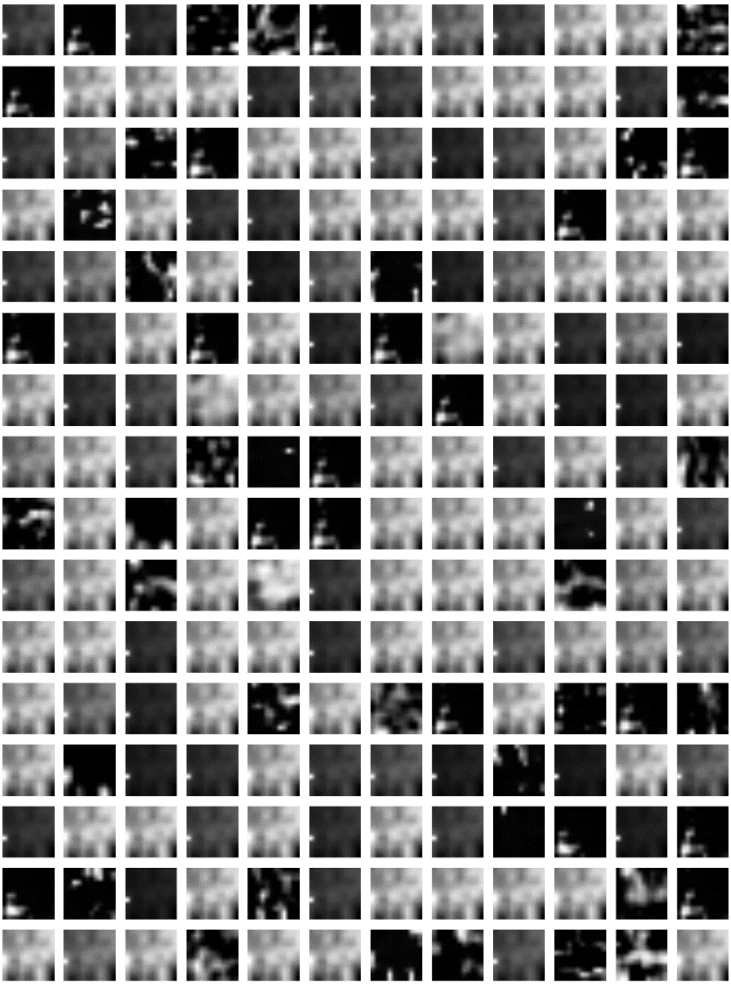
[55] M. A. Ranzato and G. E. Hinton, "Modeling Pixel Means and Covariances Using Factorized Third-Order Boltzmann Machines," 2010.

LAMPIRAN

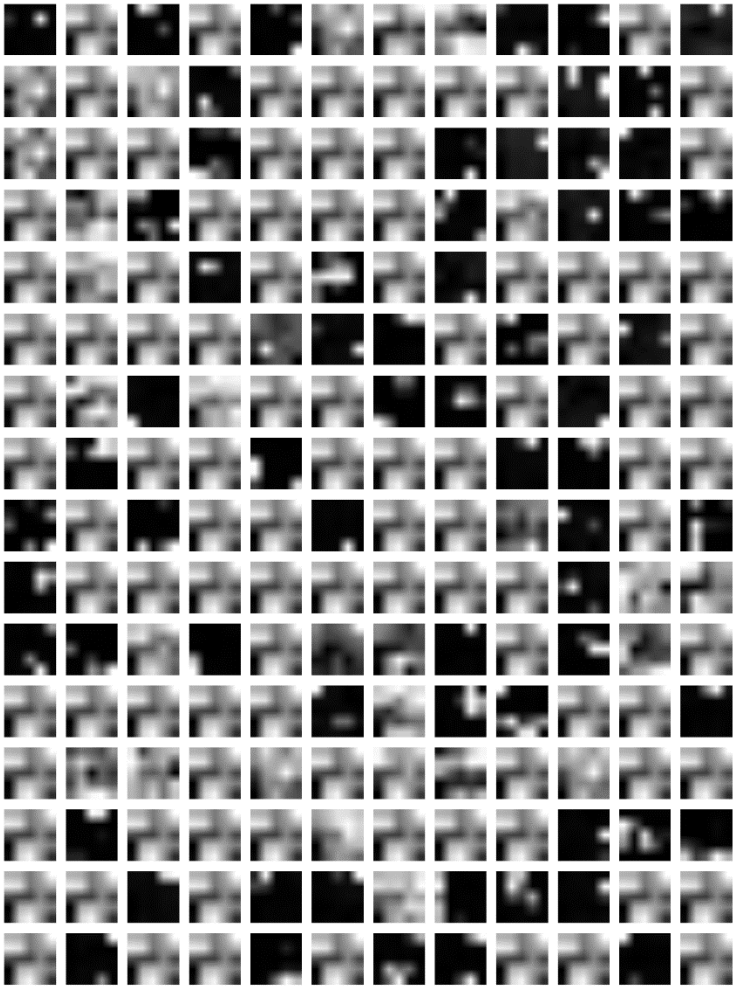


Gambar A.1 Contoh Fitur Layer 1 (NagadomiNet)

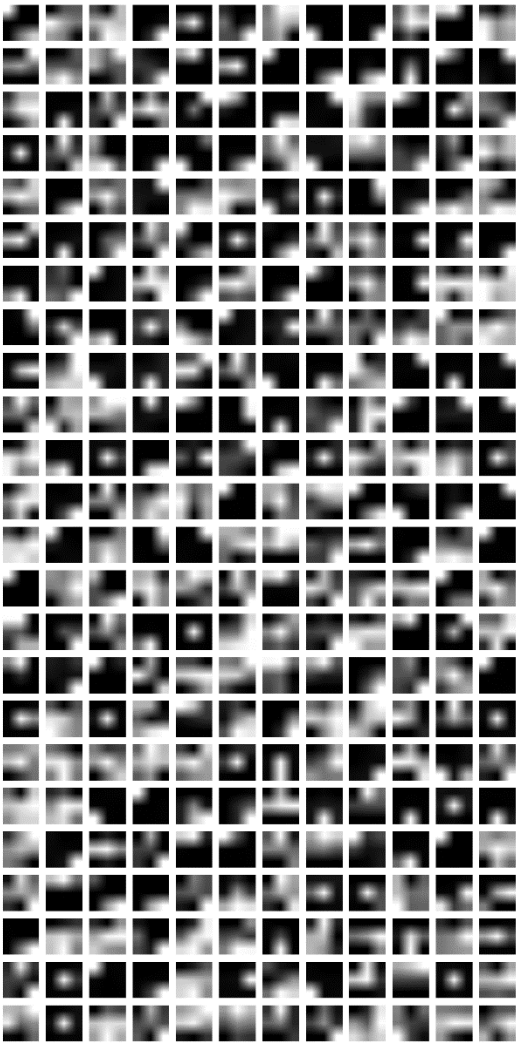
Gambar A.2 Contoh Fitur Layer 2 (NagadomiNet)



Gambar A.3 Contoh Fitur Layer 3 (NagadomiNet)



Gambar A.4 Contoh Fitur Layer 4 (NagadomiNet)



Gambar A.5 Contoh Fitur Layer 5 (NagadomiNet)

BIODATA PENULIS

Sindunuraga Rikarno Putra, lahir di Surakarta, pada tanggal 27 Agustus 1993. Penulis menempuh pendidikan mulai dari SMP Negeri 5 Semarang (2007-2008), SMA Negeri 3 Semarang (2008-2011) dan S1 Teknik Informatika ITS (2011-2015).

Selama masa kuliah, penulis aktif dalam organisasi Keluarga Muslim Informasika ITS (KMI) dan Jamaah Masjid Manarul Ilmi ITS (JMMI). Diantaranya adalah menjadi ketua divisi media KMI ITS 2013-2014 dan ketua departemen media JMMI ITS 2014-2015. Penulis juga aktif dalam laboratorium Komputasi Cerdas dan Visi (KCV) Teknik Informatika ITS sebagai Administrator. kegiatan kepanitiaan Schematics. Melalui laboratorium KCV, penulis aktif dalam kegiatan studi terkait Komputasi Cerdas dan telah berhasil meraih medali emas Pagelaran Mahasiswa Nasional Bidang TIK (Gemastik) 2014 pada cabang kompetisi *Data Mining*.

Selama kuliah di teknik informatika ITS, penulis mengambil bidang minat Komputasi Cerdas Visual (KCV) dengan fokus studi pada bidang *Machine Learning*. Penulis pernah menjadi asisten dosen mata kuliah pemrograman terstruktur dan mata kuliah struktur data. Komunikasi dengan penulis dapat dilakukan melalui email: **sindunuragarp@gmail.com**.