

Tugas Pembelajaran Mesin Lanjut



Eksplorasi Hyperparameter CNN dan Neural Network

Nama : Achmad Indra Aulia

NIM : 33221023

A. Persoalan Klasifikasi

Arsitektur *Convolutional Neural Network* (CNN) yang dibangun digunakan untuk menyelesaikan persoalan klasifikasi dari dataset Fashion MNIST. Dataset ini telah tersedia di *library* Keras sehingga siap untuk digunakan. Di sini, akan dijelaskan eksplorasi yang telah dilakukan terhadap arsitektur dari CNN.

Arsitektur dan parameter dari CNN secara detail akan dibahas pada tiap bagian eksplorasi. Secara umum, arsitektur terdiri atas *layer* input, *layer* konvolusi beserta *pooling*, proses *flattening*, *layer* yang *fully connected*, dan *layer* output. Struktur dari data di dataset ini adalah input berupa gambar *grayscale* berukuran 28x28 dengan label outputnya yang memiliki 10 kelas. Untuk itu, ukuran output adalah 10. Banyak *layer* output adalah 10 karena akan dilakukan *one hot encoding* untuk 10 kelas output. Berikut adalah eksplorasi terhadap *hyperparameter* yang telah dilakukan.

1. Banyak *Convolutional Layer*

Eksplorasi yang pertama dilakukan adalah mengenai pengubahan banyak *convolutional layer*. Untuk eksplorasi ini, berikut adalah *setting* parameter yang lain:

- Banyak filter untuk setiap *convolutional layer* adalah 32.
- Ukuran filter (kernel) untuk setiap *convolutional layer* adalah 3x3.
- Setiap *convolutional layer* dilanjutkan oleh *max pooling* berukuran 2x2.
- Banyak *hidden unit* pada *fully connected network* adalah 128.
- Fungsi aktivasi tiap *layer* adalah ReLU sedangkan fungsi aktivasi *layer* sebelum output adalah *softmax*. Fungsi untuk output ini dipilih karena cocok digunakan pada persoalan klasifikasi dan untuk menghindari *vanishing gradient*.
- *Optimizer* yang digunakan adalah Adam.
- *Loss function* yang digunakan adalah *categorical crossentropy*, salah satu fungsi yang cocok untuk persoalan regresi.
- Pengukuran kinerja dilakukan dengan *mean absolute error* (MAE).
- Ukuran *batch* adalah 32.
- Banyak *epoch* adalah 10 yang diperoleh dari beberapa kali percobaan. *Epoch* yang terlalu tinggi dapat menyebabkan *overfit*.

Berikut adalah hasil dari eksplorasi banyak *hidden layer* yang telah dilakukan.

No	Banyak <i>Convolutional Layer</i>	Akurasi training	Akurasi testing
1	1	0.97	0.913
2	2	0.95	0.911
3	3	0.91	0.89

Dari hasil eksplorasi, diketahui bahwa untuk kasus ini, semakin banyak *convolutional layer* yang lebih dari 2, performa dari model menurun. Hal ini dapat diakibatkan semakin kompleksnya jaringan sehingga kemampuan generalisasi berkurang. Sebenarnya, telah dilakukan jumlah *epoch* yang lain untuk jumlah *convolutional layer* yang lebih tinggi, akan tetapi hasil juga menunjukkan performa yang mirip dengan yang ada di tabel dan *epoch* yang lebih tinggi meningkatkan *overfit*.

Dari eksplorasi ini, banyak *convolutional layer* yang dipilih adalah 1. Jumlah ini akan digunakan pada eksplorasi-eksplorasi selanjutnya.

2. Ukuran Filter untuk Setiap *Convolutional Layer*

Eksplorasi yang selanjutnya dilakukan adalah terhadap ukuran filter untuk setiap *convolutional layer*. Ada beberapa kombinasi yang dilakukan pada saat eksplorasi yang hasilnya ditunjukkan pada tabel di bawah.

No	Ukuran filter (kernel)	Akurasi training	Akurasi testing
1	2x2	0.96	0.913
2	3x3	0.97	0.913
3	4x4	0.97	0.915
4	5x5	0.97	0.914
5	6x6	0.96	0.902
6	9x9	0.95	0.906

Dari hasil eksplorasi, diketahui bahwa ukuran filter yang menghasilkan performa paling tinggi adalah 4x4. Perbedaan yang diamati tidak terlalu signifikan. Ukuran filter berkaitan dengan proses ekstraksi fitur sehingga diketahui bahwa pada filter berukuran 4x4, lebih banyak informasi penting yang dapat diekstrak dari input.

Dari eksplorasi ini, banyak ukuran yang dipilih adalah 4x4. Ukuran ini akan digunakan pada eksplorasi-eksplorasi selanjutnya.

3. Banyak Filter untuk Setiap *Convolutional Layer*

Hal yang selanjutnya dilakukan adalah melakukan eksplorasi terhadap banyak filter untuk setiap *convolutional layer*. Berikut adalah hasil eksplorasi yang telah dilakukan.

No	Banyak filter	Akurasi training	Akurasi testing
1	4	0.94	0.9
2	8	0.95	0.908
3	16	0.96	0.911
4	32	0.97	0.915
5	64	0.97	0.913
6	128	0.97	0.915
7	256	0.98	0.913

Banyak filter di setiap *convolutional layer* menunjukkan banyak jenis fitur yang diekstrak. Semakin banyak nilai ini, semakin banyak fitur yang diekstrak. Oleh karena itu, semakin banyak filter, performa arsitektur akan meningkat walaupun peningkatan mungkin tidak signifikan jika filter sudah banyak. Selain itu, jumlah filter yang banyak akan meningkatkan kompleksitas arsitektur sehingga lebih rentan terhadap *overfit*.

Hal yang dijelaskan sebelumnya terjadi di eksplorasi. Peningkatan banyak filter meningkatkan performa arsitektur sampai suatu titik yaitu saat banyak filter adalah 32. Melebihi nilai ini, peningkatan performa sistem tidak signifikan. Dari eksplorasi ini, banyak filter yang dipilih adalah 32 dilihat dari performa dan untuk mengurangi kemungkinan *overfit*. Ukuran ini akan digunakan pada eksplorasi-eksplorasi selanjutnya.

4. Banyak *Hidden Unit* pada *Fully Connected Network*

Eksplorasi selanjutnya mengenai banyak *hidden unit* pada *fully connected network*. Berikut adalah hasil eksplorasi yang telah dilakukan.

No	Banyak <i>hidden unit</i>	Akurasi training	Akurasi testing
1	32	0.94	0.908
2	64	0.96	0.904
3	128	0.97	0.915
4	256	0.98	0.911
5	512	0.98	0.910

Dari hasil yang diperoleh, secara umum, peningkatan jumlah *hidden unit* akan meningkatkan akurasi training dan testing. Akan tetapi, pada suatu titik, akurasi dari testing akan menurun. Hal ini dikarenakan meningkatnya jumlah *hidden unit* akan meningkatkan kompleksitas arsitektur sehingga rawan terhadap *overfit*. Di eksplorasi, diketahui bahwa *hidden unit* sebanyak 128 unit masih memiliki sifat generalisasi dilihat dari akurasi testing yang paling tinggi. Oleh karena itu, nilai ini yang digunakan pada eksplorasi selanjutnya.

5. *Optimizer*

Selanjutnya, dilakukan eksplorasi terhadap *optimizer* yang digunakan saat training. Tabel di bawah menunjukkan hasil yang diperoleh dari *optimizer* yang ada pada *library* Keras.

No	Algoritma <i>optimizer</i>	Akurasi training	Akurasi testing
1	Adam	0.97	0.915
2	RMSprop	0.95	0.903
3	Gradient Descent	0.9	0.89
4	Adamax	0.94	0.911
5	Nadam	0.97	0.912
6	Adadelata	0.76	0.75
7	Adagrad	0.84	0.833

Hasil menunjukkan bahwa pada *epoch* sebesar 10, kebanyakan algoritma mampu mendekati nilai optimum sehingga akurasi yang dihasilkan tidak berbeda secara signifikan. Kasus yang menjadi pengecualian adalah algoritma dasar *gradient descent*, Adagrad, dan Adadelata. Dari website Keras, diinformasikan bahwa *default setting* dari Adadelata memiliki *learning rate* yang rendah dibandingkan dengan yang biasa digunakan untuk Adadelata sehingga performanya kurang maksimal. Dari hasil ini, algoritma *optimizer* yang digunakan tetap Adam walaupun beberapa algoritma lain memiliki performa yang tidak berbeda secara signifikan.

6. *Pengubahan Parameter Optimizer dan Learning Rate Schedule*

Eksplorasi selanjutnya adalah dengan mengubah parameter dari *optimizer* yang digunakan, yaitu Adam. Parameter utama yang dieksplorasi di sini adalah *learning rate*. Konfigurasi dan hasil yang diperoleh ditunjukkan pada tabel di bawah.

No	Konfigurasi Adam <i>optimizer</i>	Akurasi training	Akurasi testing
1	Default (<i>learning rate</i> =0.001)	0.97	0.915
2	<i>Learning rate</i> =0.01	0.92	0.88
3	<i>Learning rate</i> =0.1	0.1	0.1

Dari hasil di atas, diketahui bahwa konfigurasi *default* masih memiliki performa paling tinggi yang menunjukkan Adam *optimizer* membutuhkan *learning rate* yang relatif rendah. *Learning rate* yang tinggi justru akan menyebabkan langkah perubahan *weight* yang terlalu besar sehingga menyebabkan tidak ditemukannya nilai minimum global dari *loss function*.

Selanjutnya, dieksplorasi *learning rate* 0.001 ini dengan menggunakan *learning rate schedule* yang dapat mengubah nilai dari *learning rate* pada saat training berlangsung. Berikut adalah hasil yang diperoleh.

No	Konfigurasi Adam optimizer			Akurasi training	Akurasi testing
	Decay step	Decay rate	Jenis		
1	10000	0.9	Exponential decay	0.97	0.915
2	1000	0.9	Exponential decay	0.97	0.92
3	100	0.9	Exponential decay	0.9	0.88
4	1000	0.8	Exponential decay	0.95	0.88

Dari hasil eksplorasi ini, diketahui bahwa penerapan *learning rate schedule* yang tepat dapat meningkatkan akurasi sistem. Hal ini dikarenakan *learning rate* akan semakin kecil saat mendekati nilai optimum sehingga ketelitian langkah training akan lebih tinggi sehingga nilai optimum yang diperoleh juga lebih teliti yang akhirnya menghasilkan akurasi sistem yang lebih tinggi. Akan tetapi, perlu diperhatikan bahwa *decay* yang terlalu cepat atau terlalu besar akan menyebabkan *learning rate* cepat mengecil sehingga tidak mampu mencapai nilai optimum untuk nilai *epoch* yang sama.

Dari hasil eksplorasi ini, selanjutnya digunakan Adam optimizer dengan *learning rate schedule* berjenis *exponential decay* dengan *decay step* 1000 dan *decay rate* 0.9.

7. Loss Function

Hal yang terakhir dieksplorasi pada persoalan klasifikasi dengan CNN ini adalah *loss function*. Berikut adalah hasil eksplorasi pada *loss function* yang tersedia di Keras.

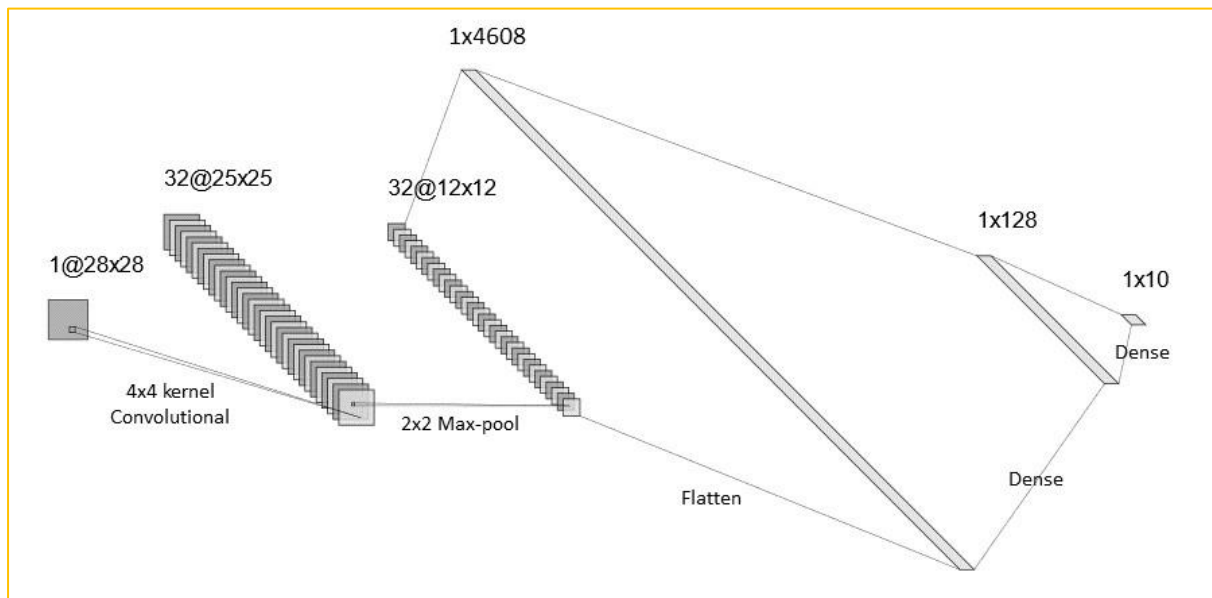
No	Loss function	Akurasi training	Akurasi testing
1	Categorical Crossentropy	0.97	0.92
2	Poisson	0.97	0.92
3	KLDivergence	0.97	0.919

Hasil eksplorasi menunjukkan bahwa perubahan *loss function* untuk kasus ini tidak berpengaruh signifikan karena *optimizer* mampu melakukan optimisasi secara baik. Untuk itu, fungsi aktivasi yang digunakan sebagai arsitektur terakhir adalah tetap *categorical crossentropy*.

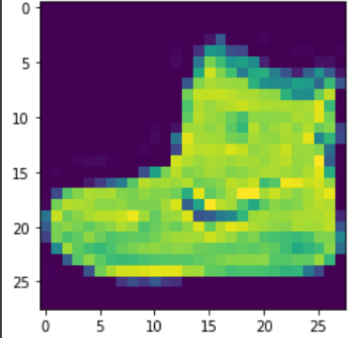
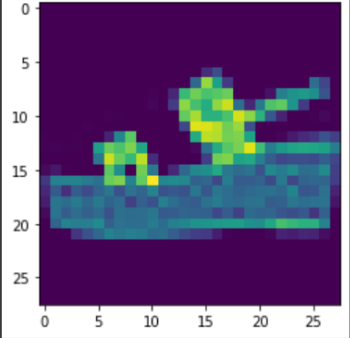
Dari yang telah dilakukan, berikut adalah beberapa *hyperparameter* yang menghasilkan performa yang optimal menurut eksplorasi yang dijalankan pada ukuran *batch* 32 dan 200 *epoch*:

- Banyak *convolutional layer* adalah 1.
- Ukuran filter di setiap *convolutional layer* adalah 4x4.
- Banyak filter di setiap *convolutional layer* adalah 32.
- Banyak *hidden unit* di *fully connected network* adalah 128.
- *Optimizer* yang digunakan adalah Adam.
- Parameter *optimizer* adalah *learning rate* awal 0.001. *Learning rate schedule* berjenis *exponential decay* dengan *decay step* 1000 dan *decay rate* 0.9.
- *Loss Function* yang digunakan adalah *categorical crossentropy*.

Analisis dari setiap pilihan ini telah dilakukan di setiap bagian eksplorasinya. Apabila dikompilasi, gambar di bawah mengilustrasikan arsitektur dari CNN yang telah dibangun. Arsitektur ini dapat melakukan klasifikasi pada data testing dengan akurasi 92%



Berikut adalah satu contoh hasil klasifikasi yang berhasil dan gagal.

Klasifikasi yang Berhasil	Klasifikasi yang Gagal
 <p>Label di dataset: Ankle boot Label hasil prediksi: Ankle boot</p>	 <p>Label di dataset: Sneaker Label hasil prediksi: Bag</p>

Kesimpulan akhir akan digabung dengan kesimpulan eksplorasi FCNN di bagian C.

B. Persoalan Regresi

Arsitektur *Fully Connected Neural Network* (FCNN) yang dibangun digunakan untuk menyelesaikan persoalan regresi dari dataset *Boston Housing Price*. Dataset ini telah tersedia di *library* Keras sehingga siap untuk digunakan. Di sini, akan dijelaskan eksplorasi yang telah dilakukan terhadap arsitektur dari FCNN.

Arsitektur dari FCNN secara detail akan dibahas pada tiap bagian eksplorasi. Secara umum, arsitektur terdiri atas *layer* input, *layer* untuk normalisasi, dan *layer* output. Layer normalisasi digunakan karena fitur memiliki *range* yang berbeda-beda. Struktur dari data di dataset ini adalah 13 fitur dengan 1 output sehingga ukuran *layer* input adalah 13 sedangkan *layer* output adalah 1.

1. Banyak *Hidden Layer*

Eksplorasi yang pertama dilakukan adalah mengenai pengubahan banyak *hidden layer*. Untuk eksplorasi ini, berikut adalah *setting* parameter yang lain:

- Banyak *hidden unit* tiap layer adalah 32.
- Fungsi aktivasi tiap *hidden layer* adalah ReLU sedangkan fungsi aktivasi *layer* sebelum output adalah linier. Fungsi ini dipilih karena cocok digunakan pada persoalan regresi.
- *Optimizer* yang digunakan adalah Adam.
- *Loss function* yang digunakan adalah *mean squared error*, salah satu fungsi yang cocok untuk persoalan regresi.
- Pengukuran kinerja dilakukan dengan *mean absolute error* (MAE).
- Ukuran *batch* adalah 32.
- Banyak *epoch* adalah 200. *Epoch* yang terlalu tinggi dapat menyebabkan *overfit*.

Berikut adalah hasil dari eksplorasi banyak *hidden layer* yang telah dilakukan.

No	Banyak <i>Hidden Layer</i>	MAE training	MAE testing	Waktu training (s)
1	1	2.35	3.14	8.9
2	2	1.69	2.70	10.6
3	3	1.50	2.71	9.46
4	4	1.20	2.50	10.64
5	5	1.13	2.61	10.69
6	6	0.83	2.66	10.16

Dari hasil di tabel, dapat diketahui bahwa secara umum, peningkatan banyak *hidden layer* akan meningkatkan performa arsitektur terhadap data training. Tetapi, peningkatan banyak *hidden layer* yang melebihi suatu nilai dapat mengurangi performa terhadap data testing seperti yang ditunjukkan saat banyak *hidden layer* lebih dari 4. Hal ini menunjukkan adanya *overfit* atau berkurangnya generalisasi yang dikarenakan jaringan semakin kompleks.

Untuk itu, pada eksplorasi selanjutnya, akan digunakan 4 buah *hidden layer* untuk menghindari *overfit* ini.

2. Banyak *Hidden Unit* dalam *Layer*

Yang akan dilakukan selanjutnya adalah eksplorasi terhadap banyak *hidden unit* dalam *layer*. *Setting* parameter yang digunakan sama dengan eksplorasi sebelumnya, tetapi dengan

pengubahan banyak *hidden unit* dan menetapkan *hidden layer* sebanyak 4. Berikut adalah hasil dari eksplorasi.

No	Banyak <i>Hidden Unit</i>	MAE training	MAE testing
1	4	2.37	3.46
2	8	1.98	2.95
3	16	1.56	2.68
4	32	1.20	2.50
5	64	0.99	2.36
6	128	0.7	2.3
7	256	0.54	2.35

Dari hasil di tabel di atas, juga dapat diketahui bahwa semakin kompleks model (semakin banyak *hidden unit* di tiap *layer*), performa arsitektur terhadap data training akan semakin baik. Akan tetapi, performa terhadap data testing akan menurun setelah suatu titik (128 ke 256) yang dikarenakan adanya *overfit* pada jaringan yang lebih kompleks seperti pada kasus sebelumnya. Dari hasil ini, pada eksplorasi selanjutnya, akan digunakan 4 buah *hidden layer* dengan 128 *hidden unit* pada tiap *layer*.

3. Fungsi aktivasi

Hal berikutnya yang dilakukan adalah eksplorasi terhadap fungsi aktivasi yang dapat digunakan. Karena persoalan yang diselesaikan adalah regresi, beberapa fungsi aktivasi yang dapat digunakan untuk *layer output* adalah ReLU (Rectified Linier Unit) dan linier. Di *layer* selain *layer output*, fungsi lainnya dapat digunakan. Pada eksplorasi ini, dicoba beberapa kombinasi fungsi aktivasi yang hasilnya ditunjukkan di tabel berikut.

No	Fungsi aktivasi					Train MAE	Test MAE
	Hidden 1	Hidden 2	Hidden 3	Hidden 4	Output		
1	ReLU	ReLU	ReLU	ReLU	Linier	0.7	2.3
2	sigmoid	sigmoid	sigmoid	sigmoid	Linier	1.83	2.58
3	tanh	tanh	tanh	tanh	Linier	0.73	2.53
4	SELU	SELU	SELU	SELU	Linier	0.98	2.53
5	sigmoid	sigmoid	ReLU	ReLU	Linier	2.05	2.94
6	tanh	tanh	ReLU	ReLU	Linier	0.6	2.47
7	ReLU	ReLU	ReLU	ReLU	ReLU	0.56	2.22

Dari hasil eksplorasi, diketahui beberapa hal:

- Proses training dengan fungsi aktivasi ReLU dan linier lebih cepat untuk dikomputasi.
- Dilihat dari MAE training dan testing, arsitektur dengan fungsi aktivasi ReLU, baik dengan ReLU ataupun linier sebagai *layer output*, memiliki performa yang lebih baik dibanding fungsi aktivasi lain. Selain itu, fungsi ReLU juga dapat menghindari adanya *vanishing gradient*.

Untuk eksplorasi selanjutnya, arsitektur di tabel no. 1 akan digunakan, yaitu semua *hidden layer* menggunakan fungsi aktivasi ReLU sedangkan output dengan fungsi aktivasi linier.

4. Optimizer

Eksplorasi yang selanjutnya dilakukan adalah pengubahan *optimizer* yang digunakan untuk training. Tabel berikut adalah hasil eksplorasi dengan beberapa *optimizer* yang disediakan di Keras untuk nilai *epoch* yang sama.

No	Optimizer	Train MAE	Test MAE	Waktu training (s)
1	Adam	0.7	2.3	21.2
2	RMSprop	2.4	3.5	21.3
3	Gradient descent (rate : 0.01, momentum 0.8)	1.01	2.5	14
4	Adadelata, learning rate = 1	0.85	2.34	21.1
5	Adamax	0.93	2.21	21.1

Dari hasil yang diperoleh, dibandingkan dengan algoritma yang lain (dengan setting *default* kecuali Adadelata dan *gradient descent*), algoritma optimisasi Adam membutuhkan langkah paling sedikit untuk mendapatkan titik loss yang minimum. Di sini, dalam 200 *epoch*, algoritma Adam sudah dapat mencapai MAE training dan testing yang paling kecil. Untuk itu, eksplorasi selanjutnya akan menggunakan Adama sebagai *optimizer*.

5. Loss Function

Yang selanjutnya dilakukan adalah eksplorasi beberapa *loss function* yang ada di *library* Keras. Perlu diperhatikan bahwa persoalan yang dipecahkan adalah regresi sehingga *loss function* yang digunakan juga harus sesuai. Tabel berikut adalah hasil eksplorasi dari beberapa *loss function* untuk persoalan regresi yang disediakan Keras.

No	Loss Function	Train MAE	Test MAE
1	Mean squared error	0.7	2.3
2	Mean absolute error	0.66	2.41
3	Mean absolute percentage error	0.83	2.42
4	Mean Squared Logarithmic Error	0.67	2.25
5	Huber	0.5	2.3
6	LogCosh	0.5	2.33

Dari hasil yang diperoleh, untuk kasus ini, performa arsitektur tidak terlalu dipengaruhi oleh pemilihan *loss function*. Hal ini dilihat dari nilai MAE testing yang mirip. Oleh karena itu, arsitektur final tetap akan menggunakan *mean squared error*, yaitu rata-rata dari error kuadrat yang relatif mudah dipahami secara intuitif.

6. Penambahan Regularisasi

Selain dari eksplorasi di atas, karena dilihat bahwa MAE training masih lebih baik dibandingkan dengan MAE training dengan *gap* yang cukup signifikan, dicoba juga penambahan salah satu metode regresi yaitu *dropout*. Berikut adalah hasil eksplorasi yang diperoleh.

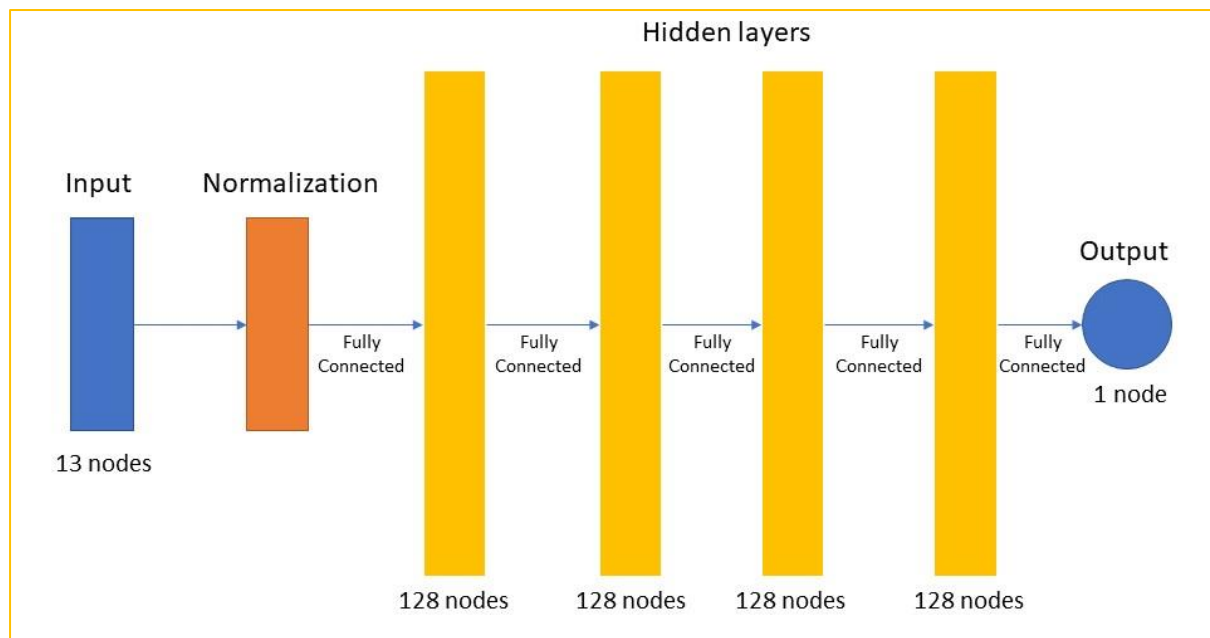
No	Konfigurasi Dropout	Train MAE	Test MAE
1	Tanpa <i>dropout</i>	0.7	2.3
2	10%	1.5	2.9
3	20% <i>dropout</i> tiap <i>layer</i>	3	3.9
4	30%	3.45	3.9
5	40%	4.03	3.9

Dari hasil *dropout*, diketahui bahwa *dropout* akan membuat model memiliki generalisasi yang lebih baik, yaitu performa saat testing mirip dengan saat training. Akan tetapi, secara performa, model tanpa *dropout* masih yang terbaik dalam kasus ini. Untuk itu, model akhir yang digunakan tidak menggunakan *dropout*.

Dari yang telah dilakukan, berikut adalah beberapa *hyperparameter* yang menghasilkan performa yang optimal menurut eksplorasi yang dijalankan pada ukuran *batch* 32 dan 200 *epoch*:

- Banyak *hidden layer* adalah 4.
- Banyak *hidden unit* tiap layer adalah 128.
- Fungsi aktivasi tiap *hidden layer* adalah ReLU sedangkan fungsi aktivasi *layer* sebelum output adalah linier. Fungsi ini dipilih karena cocok digunakan pada persoalan regresi.
- *Optimizer* yang digunakan adalah Adam.
- *Loss function* yang digunakan adalah *mean squared error*, salah satu fungsi yang cocok untuk persoalan regresi.

Analisis dari setiap pilihan ini telah dilakukan di setiap bagian eksplorasinya. Apabila dikompilasi, gambar di bawah mengilustrasikan arsitektur dari FCNN yang telah dibangun. Arsitektur ini dapat melakukan regresi pada data testing dengan nilai MAE 2.3.



Berikut adalah satu contoh hasil regresi yang mendekati target di testing dan relatif jauh dari target.

No.	Input	Output di dataset	Ouput model	Delta
1	1.27346 0. 19.58 1. 0.605 6.25 92.6 1.7984 5. 403. 14.7 338.92 5.5	27.0	32.3	5.3
2	7.1510e-02 0.0000e+00 4.4900e+00 0.0000e+00 4.4900e-01 6.1210e+00 5.6800e+01 3.7476e+00 3.0000e+00 2.4700e+02 1.8500e+01 3.9515e+02 8.4400e+00	22.2	22.24	0.04

Kesimpulan akhir eksplorasi akan digabung dengan eksplorasi CNN di bagian C.

C. Kesimpulan

Dari eksplorasi persoalan klasifikasi dengan CNN dan regresi dengan FCNN yang telah dilakukan, dapat disimpulkan beberapa hal:

- Eksplorasi *hyperparameter* pada saat pengembangan arsitektur *machine learning* penting dilakukan untuk mengetahui potensi maksimal dari suatu arsitektur.
- Untuk CNN, ukuran filter *convolutional* mempengaruhi ukuran dari fitur yang diekstraksi. Banyak filter menentukan banyak jenis fitur yang diekstrak.
- Secara umum, semakin kompleks sistem, sistem akan lebih mudah mengalami *overfit*. Akan tetapi, sistem yang kurang kompleks dapat menyebabkan *underfit*. Sistem akan semakin kompleks jika banyak *layer* semakin banyak dan banyak *unit* tiap *layer* meningkat.
- *Optimizer* dapat dipilih untuk mengatur kecepatan training dan ketelitian hasil minimisasi *loss function*. Semakin cepat training, *epoch* yang digunakan bisa lebih sedikit. Minimisasi *loss function* yang lebih teliti dapat meningkatkan performa sistem.
- Pemilihan fungsi aktivasi juga penting. Terutama untuk layer output, terdapat fungsi aktivasi yang hanya dapat digunakan pada persoalan regresi atau klasifikasi. Sebagai contoh, fungsi sigmoid tidak dapat digunakan untuk output regresi karena output terbatas pada nilai 1. Beberapa fungsi aktivasi memiliki karakteristik tertentu yang dapat dimanfaatkan, misalkan ReLU untuk menghindari *vanishing gradient*.
- Regularisasi dapat meningkatkan generalisasi. Kekurangannya adalah dapat menurunkan performa untuk *epoch* dan arsitektur yang sama tanpa regularisasi.