

Model CNN Hasil Running Result :

Hasil running model AlexNet pada dataset CIFAR-10

```
Extracting /content/cifar-10-python.tar.gz to /content
Files already downloaded and verified
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: True
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU cores
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
/usr/local/lib/python3.10/dist-packages/pytorch_lightning/trainer/configuration_validator.py:72: PossibleUserWarning: You defined a `validation_data_loader` but no `validation` key in the `datamodule`. This may lead to unexpected behavior.
rank_zero_warn(
WARNING:pytorch_lightning.loggers.tensorboard:Missing logger folder: /content/lightning_logs
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.callbacks.model_summary:
  | Name          | Type           | Params
-----|-----|-----
0 | features       | Sequential     | 2.3 M
1 | avgpool        | AdaptiveAvgPool2d | 0
2 | classifier     | Sequential     | 54.6 M
-----|-----|-----
56.8 M  Trainable params
0       Non-trainable params
56.8 M  Total params
227.307 Total estimated model params size (MB)
Epoch 19: 100% 782/782 [00:30<00:00, 25.29it/s, v_num=0, train_loss_step=1.130, train_acc_step=0.625, train_loss_epoch=0.817, train_acc_epoch=0.6662999987602234]
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=20` reached.
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
Testing DataLoader 0: 100% 157/157 [00:03<00:00, 46.7it/s]
```

Test metric	DataLoader 0
test_acc	0.6662999987602234

Hasil running model VGGNet

```
Epoch 19: 100% [Progress bar]
```


```
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=20` reached.
INFO:pytorch_lightning.utilities.rank_zero:Restoring states from the checkpoint
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.utilities.rank_zero:Loaded model weights from the checkpoint

Testing DataLoader 0: 100% [Progress bar]
```

Test metric	DataLoader 0
test_acc	0.786899983882904

Prediction: 6

Hasil Running pada Kaggle Notebook (GoogleNet):

Epoch 19: 100% 
470/470 [00:38<00:00, 12.25it/s, loss=1.96, v_num=4, train_loss_step=2.050, train_acc_step=0.237, val_loss_step=1.720,
val_acc_step=0.312, val_loss_epoch=1.930, val_acc_epoch=0.286, train_loss_epoch=1.990, train_acc_epoch=0.280]

Hasil Running pada Kaggle Notebook (ResNet) :

Testing DataLoader 0: 100% 


Test metric	DataLoader 0
test_acc	0.42879998683929443
test_loss	1.8781439065933228

```
13]: [{'test_loss': 1.8781439065933228, 'test_acc': 0.42879998683929443}]
```


Hasil Running pada Google Colab (DenseNet)

Name	Type	Params
0 model	DenseNet	7.0 M

7.0 M	Trainable params
0	Non-trainable params
7.0 M	Total params
27.856	Total estimated model params size (MB)

Epoch 19: 100% 

INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=20` reached
INFO:pytorch_lightning.utilities.rank_zero:Restoring states from the checkpoint path at /
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.utilities.rank_zero:Loaded model weights from the checkpoint at /c


Testing DataLoader 0: 100% 

Test metric	DataLoader 0
test_acc_epoch	0.7732999920845032
test_loss_epoch	0.6535437703132629

```
[{'test_loss_epoch': 0.6535437703132629, 'test_acc_epoch': 0.7732999920845032}]
```

Result dari Running Model RNN :


Hasil Running LSTM (Google Colab) :

```
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.utilities.rank_zero:Loaded model weights from the checkpoint
Testing DataLoader 0: 100% 
```

Test metric	DataLoader 0
test_acc	0.9872000217437744
test_loss	0.056949835270643234

```
[{'test_loss': 0.056949835270643234, 'test_acc': 0.9872000217437744}]
```


Hasil Running GRU (Google Colab) :


```
INFO:pytorch_lightning.utilities.rank_zero:Restoring states from the checkpoint path at /content
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.utilities.rank_zero:Loaded model weights from the checkpoint
Testing DataLoader 0: 100% 
```

Test metric	DataLoader 0
test_acc	0.9879999756813049
test_loss	0.04305749386548996

```
[{'test_loss': 0.04305749386548996, 'test_acc': 0.9879999756813049}]
```

Hasil Running DRNN (Google Colab) :

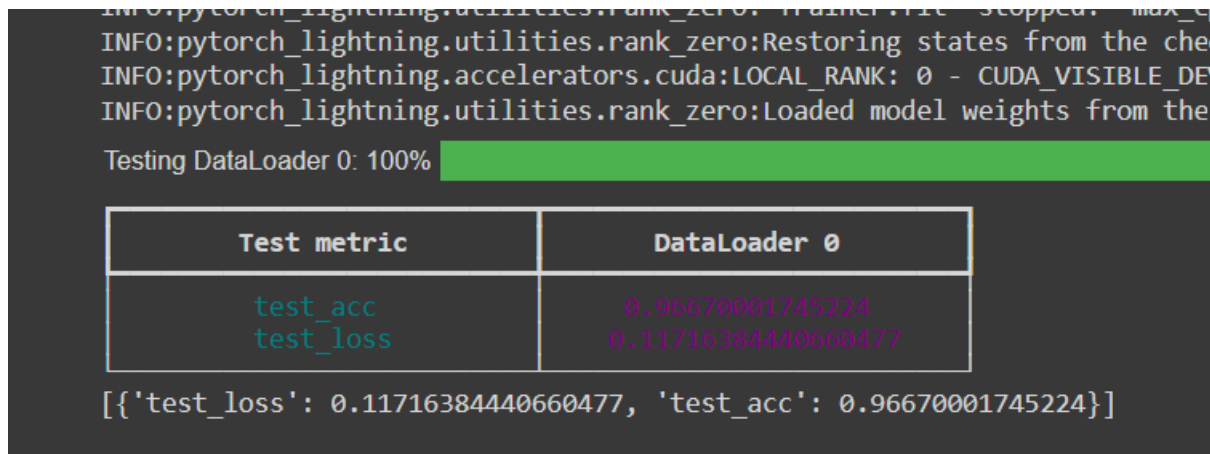
```
87.6 K Total params
0.350 Total estimated model params size (MB)
Epoch 19: 100% 
```

```
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=20` reached.
INFO:pytorch_lightning.utilities.rank_zero:Restoring states from the checkpoint path at /content
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.utilities.rank_zero:Loaded model weights from the checkpoint at /content
Testing DataLoader 0: 100% 
```

Test metric	DataLoader 0
test_acc	0.9153000116348267
test_loss	0.2805738151073456

```
[{'test_loss': 0.2805738151073456, 'test_acc': 0.9153000116348267}]
```

Hasil Running BIRNN (Google Colab) :



Penjelasan dan Analisis Code adalah sebagai berikut :

EPOCH distandarkan pada kesemua model menjadi 20 epoch

Hasil Akurasi memiliki beberapa variasi, namun perlu dilakukan percobaan lebih lanjut mengenai optimasi parameter terkait .

Pada model CNN digunakan dataset CIFAR-10 sedangkan pada RNN menggunakan MINST dataset

Secara umum akurasi yang dihasilkan CNN sangat rendah, bahkan Resnet hanya mencapai 42%, rata -rata dikisaran 70%. Namun, yang dihasilkan RNN sangat tinggi karena semuanya sudah diatas 97%.

Berikut Report untuk masing masing model :

AlexNet

Berikut adalah penjelasan yang lebih rinci mengenai setiap layer pada AlexNet:

1.Convolutional Layer 1: Layer pertama dalam AlexNet adalah layer konvulsi yang terdiri dari 96 filter dengan ukuran 11x11 dan stride 4. Input dari layer ini adalah gambar berukuran 227x227x3, dengan tiga saluran warna (RGB). Output dari layer ini memiliki dimensi 55x55x96.

2.Max Pooling Layer 1: Setelah layer konvulsi pertama, terdapat max pooling layer dengan ukuran filter 3x3 dan stride 2. Output dari layer ini memiliki dimensi 27x27x96.

3.Convolutional Layer 2: Layer kedua adalah layer konvulsi dengan 256 filter, masing-masing dengan ukuran 5x5 dan stride 1. Output dari layer ini memiliki dimensi 27x27x256.

4.Max Pooling Layer 2: Setelah layer konvulsi kedua, terdapat max pooling layer dengan ukuran filter 3x3 dan stride 2. Output dari layer ini memiliki dimensi 13x13x256.

5.Convolutional Layer 3: Layer ketiga adalah layer konvulsi dengan 384 filter, masing-masing dengan ukuran 3x3 dan stride 1. Output dari layer ini memiliki dimensi 13x13x384.

6.Convolutional Layer 4: Layer keempat adalah layer konvulsi dengan 384 filter, masing-masing dengan ukuran 3x3 dan stride 1. Output dari layer ini memiliki dimensi 13x13x384.

7.Convolutional Layer 5: Layer kelima adalah layer konvulsi dengan 256 filter, masing-masing dengan ukuran 3x3 dan stride 1. Output dari layer ini memiliki dimensi 13x13x256.

8.Max Pooling Layer 3: Setelah layer konvulsi kelima, terdapat max pooling layer dengan ukuran filter 3x3 dan stride 2. Output dari layer ini memiliki dimensi 6x6x256.

9.Fully Connected Layer 1: Setelah semua layer konvulsi dan max pooling, input yang dihasilkan diberikan pada layer fully connected pertama dengan 4096 neuron.

10.Fully Connected Layer 2: Layer kedua dari fully connected layer memiliki 4096 neuron.

11.Fully Connected Layer 3: Layer terakhir dari fully connected layer adalah output layer dengan 1000 neuron, yang merepresentasikan 1000 kelas dari dataset ImageNet.

12.Selain layer-layer di atas, AlexNet juga menggunakan beberapa teknik seperti dropout dan data augmentation untuk mencegah overfitting dan meningkatkan generalisasi dari model.

Penjelasan atas **vggnet**

VGGNet (Visual Geometry Group Network) adalah sebuah arsitektur jaringan saraf konvolusional (Convolutional Neural Network/CNN) yang dikembangkan oleh tim peneliti di University of Oxford. VGGNet menjadi salah satu arsitektur CNN yang sangat populer dan sering digunakan sebagai dasar untuk banyak aplikasi visi komputer. VGGNet terdiri dari 16 atau 19 lapisan konvolusi yang mendalam (deep), yang membuatnya memiliki kemampuan untuk mempelajari fitur-fitur kompleks dari gambar secara efektif. Berikut adalah penjelasan mengenai setiap layer pada arsitektur VGGNet:

1. Input Layer: layer pertama dari VGGNet adalah layer input, yang digunakan untuk memasukkan gambar ke dalam arsitektur CNN. Ukuran gambar yang dimasukkan ke dalam arsitektur ini biasanya adalah 224x224 piksel dengan 3 channel warna RGB.

2. Convolutional Layers: VGGNet menggunakan layer konvolusi dengan kernel 3x3 untuk mengekstrak fitur dari gambar. Arsitektur ini memiliki 13 atau 16 lapisan konvolusi, dimana masing-masing layer terdiri dari beberapa konvolusi yang diikuti oleh fungsi aktivasi ReLU. Konvolusi berguna untuk mengekstrak fitur-fitur dasar seperti garis, tepi, dan sudut dari gambar.

3. Max Pooling Layers: Setiap beberapa layer konvolusi, VGGNet menggunakan layer max pooling dengan ukuran filter 2x2 dan stride 2 untuk mengurangi resolusi gambar dan menjaga informasi spasial yang penting. Max pooling mengambil nilai maksimum dari setiap jendela 2x2, sehingga menghasilkan setengah ukuran gambar dari layer sebelumnya.

4. Fully Connected Layers: Setelah melalui beberapa lapisan konvolusi dan max pooling, VGGNet menggunakan 3 atau 4 layer fully connected sebagai classifier. Layer fully connected ini berguna untuk mengkombinasikan fitur-fitur kompleks dari gambar menjadi representasi fitur yang lebih tinggi dan menjadikan prediksi kelas dari gambar. Fully connected layers biasanya diikuti oleh fungsi aktivasi ReLU.

5. Output Layer: Layer terakhir dari VGGNet adalah layer output, yang menghasilkan probabilitas kelas untuk gambar masukan. Layer output ini biasanya menggunakan softmax activation untuk menghasilkan probabilitas kelas yang digunakan untuk prediksi.

Secara keseluruhan, VGGNet memiliki 16 atau 19 lapisan konvolusi yang dalam, dimana masing-masing lapisan terdiri dari beberapa konvolusi dan fungsi aktivasi ReLU, max pooling layer untuk mengurangi resolusi gambar, dan fully connected layers sebagai classifier. Arsitektur VGGNet telah terbukti sangat efektif dalam

mempelajari fitur-fitur kompleks dari gambar dan menjadi salah satu arsitektur CNN yang sangat populer.

Penjelasan mengenai GoogleNet :

GoogleNet, juga dikenal sebagai Inception-v1, adalah sebuah arsitektur model Deep Learning yang dikembangkan oleh tim peneliti Google pada tahun 2014. Arsitektur ini memenangkan kompetisi ImageNet Large Scale Visual Recognition Challenge pada tahun yang sama. GoogleNet terdiri dari 22 lapisan (layers) dan mengandung lebih dari 5 juta parameter. GoogleNet menggunakan modul Inception yang terdiri dari beberapa jalur paralel, masing-masing dilatih pada tingkat resolusi yang berbeda, dan kemudian hasilnya digabungkan menjadi satu keluaran. Konsep utama di balik Inception adalah untuk mengurangi jumlah parameter model sambil meningkatkan kekayaan ekspresif model dengan cara mengekstraksi fitur pada berbagai tingkat resolusi dan kemudian menggabungkannya.

Arsitektur GoogleNet terdiri dari tiga bagian utama:

1. Lapisan Konvolusi Awal (Initial Convolution Layer) yang melakukan ekstraksi fitur awal dari gambar input dengan kernel berukuran 7x7 dan stride 2.
2. Inception Blocks, yang terdiri dari beberapa modul Inception yang saling terhubung, yang menghasilkan output yang kompleks dari tingkat fitur yang berbeda. Modul Inception dirancang untuk memperluas kedalaman dan lebar model dengan cara yang efisien. Pada Inception Block, setiap jalur memiliki ukuran kernel yang berbeda dan kemudian menggabungkan hasilnya.
3. Lapisan klasifikasi akhir, yang terdiri dari lapisan pooling global dan lapisan dense (fully connected) dengan softmax sebagai fungsi aktivasi.

GoogleNet juga mengandung beberapa teknik regulisasi, seperti dropout, L2 regularisation, dan "label smoothing", yang digunakan untuk menghindari overfitting dan meningkatkan generalisasi model.

Dalam hal kinerja, GoogleNet berhasil mencapai tingkat akurasi yang lebih tinggi daripada arsitektur Deep Learning lainnya pada saat itu, dengan tingkat error 6,7% pada ImageNet Large Scale Visual Recognition Challenge.

DenseNet dan ResNet

Densenet dan Resnet adalah dua jenis arsitektur jaringan saraf tiruan yang sangat populer untuk tugas pengenalan gambar. Kedua arsitektur ini dirancang untuk mengatasi masalah degradasi yang terjadi pada jaringan saraf tiruan yang sangat dalam. Resnet, atau disebut juga Residual Neural Network, dikembangkan pada tahun 2015 oleh Microsoft Research Asia dan memenangkan kompetisi ImageNet dan COCO 2015. Arsitektur Resnet didasarkan pada blok residu yang memungkinkan jaringan untuk melakukan pelatihan yang lebih dalam tanpa mengalami masalah degradasi performa. Blok residu adalah unit dasar dari arsitektur Resnet dan terdiri dari dua lapisan konvolusi. Blok residu menerima masukan x dan menghasilkan keluaran y melalui jalur pendekatan yang dikenal sebagai shortcut connection atau skip connection. Shortcut connection memungkinkan model untuk mengalirkan informasi dari lapisan yang lebih awal ke lapisan yang lebih dalam dengan cara yang lebih mudah. Densenet, atau disebut juga Dense Convolutional Network, dikembangkan pada tahun 2016 oleh Gao Huang dan rekan-rekannya di Cornell University. Arsitektur Densenet memanfaatkan konsep koneksi bertingkat dan bertumpuk dengan cara yang inovatif. Pada Densenet, setiap blok terhubung langsung dengan setiap blok di lapisan berikutnya. Hal ini berarti setiap blok menerima sinyal dari semua blok sebelumnya dan menghasilkan sinyal yang dikirim ke semua blok berikutnya. Ini memungkinkan model untuk mengekstraksi fitur yang lebih kaya dan lebih kuat, sehingga memungkinkan klasifikasi gambar yang lebih akurat. Selain itu, Densenet juga memiliki jumlah parameter yang lebih sedikit dibandingkan dengan arsitektur lain yang memperoleh akurasi yang setara. Kedua arsitektur ini telah digunakan dalam berbagai aplikasi pengenalan gambar dan telah menunjukkan keunggulan dalam beberapa tugas pengenalan gambar dibandingkan dengan arsitektur lainnya.

Penjelasan mengenai LSTM dan GRU :

LSTM (Long Short-Term Memory) dan GRU (Gated Recurrent Unit) adalah dua jenis arsitektur jaringan saraf rekuren (RNN) yang dirancang untuk mengatasi masalah vanishing gradient descent pada RNN tradisional. Pada RNN tradisional, ketika informasi berulang kali melalui unit, gradien yang sangat kecil terkumpul dalam waktu lama, mengakibatkan informasi penting terhapus dari memori unit. LSTM dan GRU memperbaiki masalah ini dengan mengintegrasikan "gate" ke dalam unit RNN.

LSTM menggunakan tiga jenis gate: forget gate, input gate, dan output gate. Forget gate menentukan seberapa banyak informasi lama yang akan diabaikan, input gate menentukan seberapa banyak informasi baru yang akan ditambahkan

ke memori, dan output gate menentukan seberapa banyak informasi yang akan dipindahkan ke output. Dalam hal ini, LSTM secara efektif dapat mengontrol aliran informasi yang akan dilewatkan dan yang akan disimpan. Sementara itu, GRU menggunakan dua jenis gate: reset gate dan update gate. Reset gate menentukan seberapa banyak informasi lama yang akan diabaikan, sedangkan update gate menentukan seberapa banyak informasi baru yang akan ditambahkan ke memori. GRU lebih sederhana daripada LSTM, karena hanya menggunakan dua gate, sehingga lebih mudah dan lebih cepat untuk dilatih.

Kedua arsitektur ini terbukti sangat efektif dalam memproses data sequential seperti teks dan suara, dan banyak digunakan dalam berbagai aplikasi pemrosesan bahasa alami, deteksi wajah, dan prediksi vektor waktu. Namun, karena kompleksitas arsitektur, LSTM dan GRU cenderung memerlukan lebih banyak waktu dan daya komputasi untuk dilatih dan dievaluasi dibandingkan dengan jaringan saraf lainnya.

Penjelasan mengenai B-RNN,D-RNN,Seq2Seq

Deep Recurrent Neural Network (DRNN) adalah jenis arsitektur jaringan saraf tiruan yang dirancang untuk memproses data berurutan, seperti teks atau sinyal waktu. DRNN memungkinkan kita untuk memodelkan dan mempelajari dependensi temporal jangka panjang dalam data. Secara umum, DRNN terdiri dari beberapa lapisan berulang yang saling terhubung. Setiap lapisan menerima input dan output dari lapisan sebelumnya, dan memperbarui representasi internalnya berdasarkan urutan input yang diberikan. DRNN dapat berupa model sekuensial atau paralel. Bi-Directional Neural Network (BiRNN) adalah jenis arsitektur jaringan saraf tiruan yang mampu memodelkan dependensi temporal dari kedua arah dalam suatu urutan. Dalam BiRNN, input disajikan ke jaringan dari dua arah - maju dan mundur - dan output keduanya digabungkan untuk menghasilkan representasi terbaik dari urutan tersebut.

Seq2Seq (Sequence-to-Sequence) adalah jenis model jaringan saraf yang memungkinkan kita untuk memproses data berurutan pada dua domain yang berbeda, seperti terjemahan mesin atau tugas pemrosesan bahasa alami lainnya. Seq2Seq terdiri dari dua bagian: encoder dan decoder. Encoder mengambil input dalam satu domain, seperti bahasa sumber, dan menghasilkan representasi kontekstual dari input tersebut. Kemudian, decoder menggunakan representasi ini untuk menghasilkan output dalam domain lain, seperti bahasa target.

Untuk membangun model Seq2Seq, kita biasanya menggunakan jenis lapisan RNN seperti LSTM atau GRU sebagai encoder dan decoder. Ini memungkinkan model untuk memperhitungkan dependensi temporal jangka panjang dalam input dan output.