Project Machine Learning Kelas B

Image Classification with CIFAR-10 Images Dataset

Anggota kelompok dan Pembagian tugas :

1. Andy - C14190103 (Dikerjakan bersama)
2. Michael Kevin T - C14190167 (Dikerjakan bersama)

- Teori dan Model yang diujikan :

  Kelompok kami melakukan proyek percobaan *Image Classification* terhadap *dataset* CIFAR 10. Metode atau teori dasar yang kelompok kami gunakan adalah *CNN (Convolutional Neural Network / ConvNet)*. Tujuan dari percobaan proyek ini adalah jika terdapat gambar, program dapat membangun model *CNN* yang dapat memprediksi dan mengklasifikasikan kelas yang benar dari setiap gambar tersebut. Sedangkan di dalam *dataset* yang kami gunakan yaitu CIFAR 10, terdapat 10 kelas gambar yaitu :

  1. *Airplane* / pesawat
  2. *Automobile* / mobil
  3. *Bird* / burung
  4. *Cat* / kucing
  5. *Deer* / rusa
  6. *Dog* / anjing
  7. *Frog* / katak
  8. *Horse* / kuda
  9. *Ship* / kapal
  10. *Truck* / truk

  Cara kerja dari program kami sendiri adalah kami memiliki 2 *main files*, yang pertama digunakan untuk melakukan *training* dari model yang kami dapatkan dari sumber referensi, *file* yang kedua digunakan untuk melakukan *testing* dari model yang sudah di *train* sebelumnya. Cara kerja dari *file training* yang pertama program mendapatkan dataset yang digunakan dan pada proyek kali ini kelompok kami menggunakan *dataset* CIFAR 10. Step pertama program akan membuat semacam array / kontainer untuk menyimpan data yang sudah di load :

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

Untuk kontainer akan terdapat beberapa yang merupakan *training images* dan juga *test images,* karena pada saat di *load images* yang ada di *dataset* tidak bisa dipisah-pisahkan. Selanjutnya adalah dengan melakukan sedikit data pre-processing :

```
# Scale the data
X_train = X_train / 255.0
X_test = X_test / 255.0

# Transform target variable into one-hotencoding
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

Alasan untuk melakukan *preprocessing* adalah untuk normalisasi nilai dari *dataset* yang digunakan, yang dibagi dengan 255.0 supaya range tetap berada di antara 0 dan 1. Setelah itu adalah model building sebagai berikut :

```
model = Sequential()

# Convolutional Layer
model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
# Pooling layer
model.add(MaxPool2D(pool_size=(2, 2)))
# Dropout layers
model.add(Dropout(0.25))

model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=128, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
#model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(10, activation='softmax'))
```

Dalam mendesain suatu model memerlukan banyak variabel yang harus dipertimbangkan, namun contoh kali ini, kami menggunakan model tipe *sequential*, tetapi sebenarnya *Keras* dan *Tensorflow* menyediakan banyak tipe model, namun tipe *sequential* adalah yang paling sering digunakan. Cara pembuatan model adalah dengan membuat variabel model lalu di *assign* dengan *constructor sequential*. Untuk model buildingnya sendiri terdiri dari beberapa layer dan pada gambar diatas terdapat 3 layer, per layer yang pertama diperlukan adalah *Convolutional layer* yang digunakan untuk menerima input dan melakukan filter convolutional pada input tersebut, layar ini memerlukan beberapa parameter yang pertama adalah channel atau filter, kernel size yang akan digunakan, input shape, tipe aktivasi yang digunakan dan tipe padding. Untuk filter, kami menggunakan *default* 32 dan untuk kernel 3 x 3, *input shape* 32 x 32 x 3, *activation* relu, yang merupakan salah satu metode paling umum digunakan, *padding* tipe *same* yang dimana input dan output tidak berubah dimensi. Setelah itu terdapat *batch normalization sub-layer* yang berfungsi untuk menormalisasikan input yang diterima sebelum dikirimkan ke layer setelahnya.

Setelah itu terdapat *Max pooling layer* yang berguna untuk membuang detail yang dianggap (oleh program) tidak penting yang nantinya menghasilkan *output* yang *size*nya lebih kecil dari inputnya, karena *max pooling* ini membuang detail sesuai dengan *decision* dari program, bisa diharapkan bahwa semakin lama model ini di *train* (epoch yang digunakan) model bisa belajar untuk memilih mana yang tidak penting dan mana yang penting. Ini merupakan salah satu cara untuk mencegah *over fitting*. Selanjutnya akan ditambah dengan *sublayer dropout* yang berguna untuk mencegah *over fitting*, namun bedanya dengan *max pooling* adalah dengan membuang koneksi diantara layer, dan dalam projek kali ini kami menggunakan 0.25 atau membuang 25% dari koneksi antar layer yang ada saat itu. Setelah mendesain layer yang diinginkan, selanjutnya menambah layer untuk melakukan *flatten data* dengan menggunakan fungsi seperti gambar diatas. Setelah itu kami menambahkan layer Dense yang digunakan untuk membuat *densely connected layer*, layer ini akan meminta jumlah neuron yang akan dibuat serta tipe aktivasi yang digunakan, dan dalam kasus ini kami menggunakan relu. Jumlah neuron yang diperlukan adalah sesuai dengan jumlah kategori dari input, dalam kasus ini karena kami menggunakan *dataset* CIFAR 10, maka diperlukan 10 neuron.

Setiap neuron akan mewakili 10 kelas yang diberikan oleh CIFAR 10 dan akan menampung probabilitas / prediksi dari gambar yang telah diolah. Untuk aktivasi kami menggunakan *softmax* untuk mengambil probabilitas terbesar, nilai tersebut merupakan prediksi model akan gambar yang diinputkan.

```
METRICS = [
    'accuracy',
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
]
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=METRICS)
```

Dalam *compiling* sebuah model, diperlukan *optimizer* yang digunakan untuk tuning berat / *weights* dari model tersebut guna mencapai *lowest point of weight loss*. Dalam kasus ini kami menggunakan *optimizer* tipe Adam, tapi sebenarnya ada banyak optimizer seperti Nadam, RMSProp dan lain-lain. Setelah di compile, bisa dilihat summary / evaluasi dari model yang kelompok kami buat.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 batch_normalization_1 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 dropout (Dropout)           (None, 16, 16, 32)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 batch_normalization_2 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 batch_normalization_3 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 8, 8, 64)          0

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 batch_normalization_4 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 batch_normalization_5 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 dropout_2 (Dropout)         (None, 4, 4, 128)         0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 128)               262272

 dropout_3 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 552,362
Trainable params: 551,466
Non-trainable params: 896
```

Setelah *compile* dan *summary*, selanjutnya adalah *training model*, kami menggunakan fungsi fit() yang memiliki beberapa parameter antara lain :

```
early_stop = EarlyStopping(monitor='val_loss', patience=2)

batch_size = 32
data_generator = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
train_generator = data_generator.flow(X_train, y_train, batch_size)
steps_per_epoch = X_train.shape[0] // batch_size

r = model.fit(train_generator,
              epochs=40,
              steps_per_epoch=steps_per_epoch,
              validation_data=(X_test, y_test),
#               callbacks=[early_stop],
#               batch_size=batch_size,
              )
```

Selain dari itu fungsi fit () juga memberikan opsi untuk menyimpan *history* dari *training model*, yang nanti berguna untuk penilaian model yang kelompok kami buat.

- Test Model 1, dengan tingkat akurasi 86 %

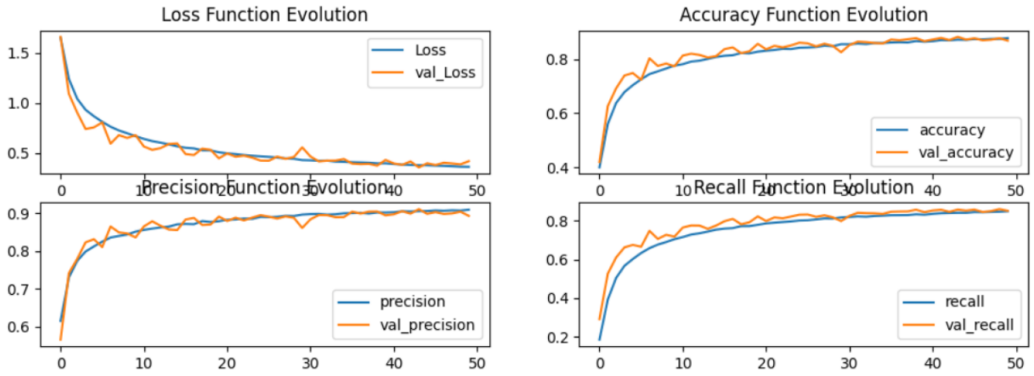
  50 *Epochs*, 3 *Layer.*

```
#Test Model 1
# Convolutional Layer
model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
# Pooling layer
model.add(MaxPool2D(pool_size=(2, 2)))
# Dropout layers
model.add(Dropout(0.25))

model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=128, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())Z
#model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(10, activation='softmax'))
```



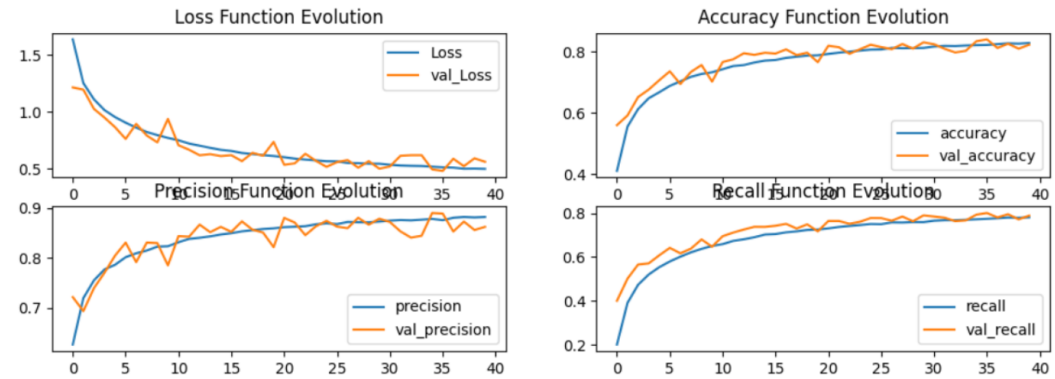
❖ Test Model 2, dengan tingkat akurasi 82 %

40 *Epochs*, 2 *Layer.*

```
#Test Model 2
# Convolutional Layer
model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
# Pooling layer
model.add(MaxPool2D(pool_size=(2, 2)))
# Dropout layers
model.add(Dropout(0.25))

model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
#model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(10, activation='softmax'))
```



❖ Test Model 3, dengan tingkat akurasi 70 %

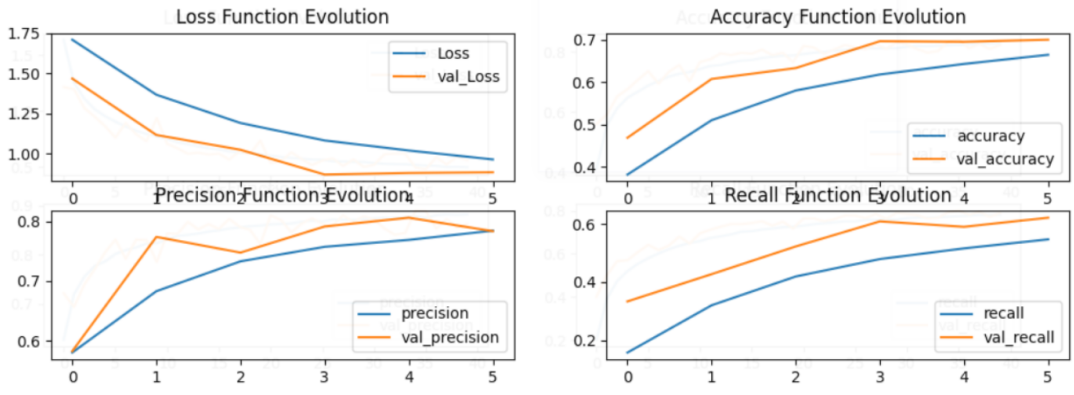
40 *Epochs*, 3 *Layer*, 1 tambahan *dropout layer*, *filter*(*sublayer conv2*d) diganti ke 16-32-64.

```
#Test Model 1
# Convolutional Layer
model.add(Conv2D(filters=16, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=16, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
# Pooling layer
model.add(MaxPool2D(pool_size=(2, 2)))
# Dropout layers
model.add(Dropout(0.25))

model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
#model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(10, activation='softmax'))
```



❖ Test Model 4, dengan tingkat akurasi 79%

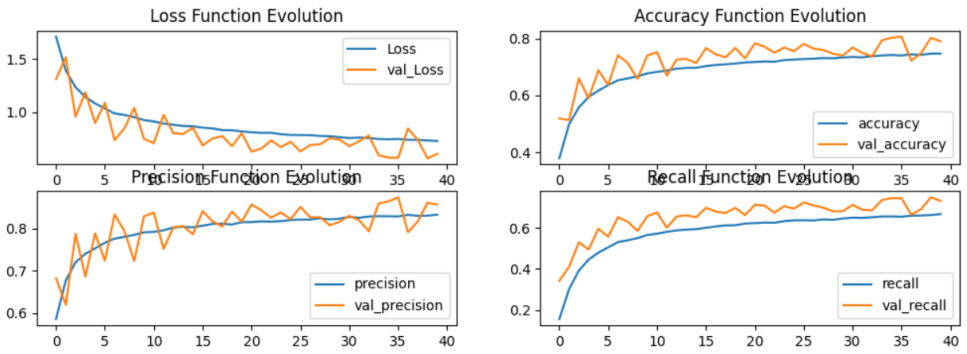
40 *Epochs*, 3 *Layer*, *Conv2D filter* mulai dari 32-64-128, density 64 neurons.

```
#Test Model 4
model.add(Conv2D(filters=32, kernel_size=(3,3), input_shape=(32,32,3), activation='relu', padding='same'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(BatchNormalization())

model.add(Conv2D(filters=64, kernel_size=(3,3), input_shape=(32,32,3), activation='relu', padding='same'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(BatchNormalization())

model.add(Conv2D(filters=128, kernel_size=(3,3), input_shape=(32,32,3), activation='relu', padding='same'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(10, activation='softmax'))
```



- Penjelasan Dataset yang digunakan :

  Dataset yang digunakan adalah dataset CIFAR-10 yang terdiri dari 60.000 gambar berwarna 32x32 dalam 10 kelas, dengan 6000 gambar per kelas. Ada 50000 *training images* dan 10.000 *test images*.

  Di dalam dataset CIFAR 10, terdapat 10 kelas gambar yaitu :

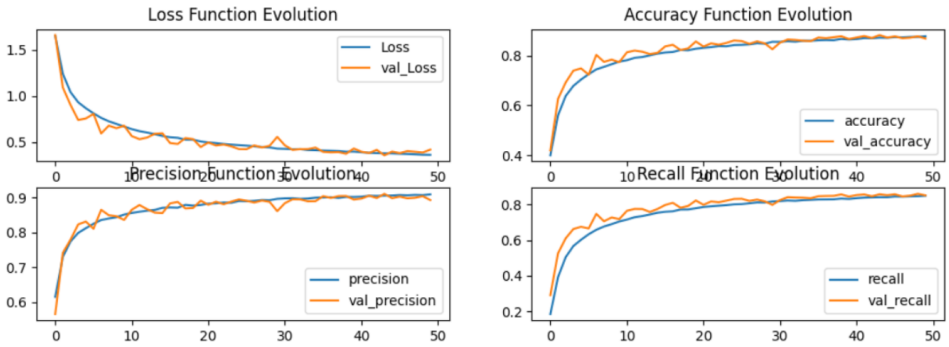
  1. *Airplane* / pesawat
  2. *Automobile* / mobil

3. *Bird* / burung
4. *Cat* / kucing
5. *Deer* / rusa
6. *Dog* / anjing
7. *Frog* / katak
8. *Horse* / kuda
9. *Ship* / kapal
10. *Truck* / truk

Dataset dibagi menjadi lima *batch training* dan satu *batch testing*, masing-masing dengan 10.000 gambar. *Batch testing* berisi tepat 1000 gambar yang dipilih secara acak dari setiap kelas. *Batch training* berisi gambar yang tersisa dalam urutan acak, tetapi beberapa *batch training* mungkin berisi lebih banyak gambar dari satu kelas daripada yang lain.

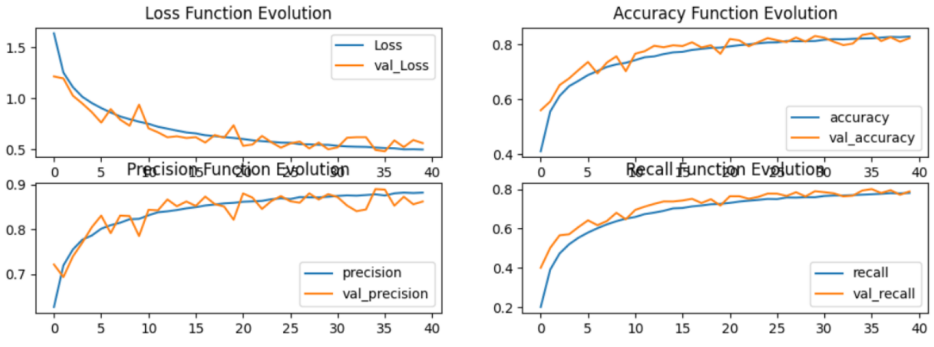
- Hasil Pengujian dan Analisa :
  1. Percobaan dengan mengganti jumlah *epoch* (mengurangi) & variasi jumlah *layer*
     - ❖ Test Model 1, dengan tingkat akurasi 86 %

       50 *Epochs*, 3 *Layer.*



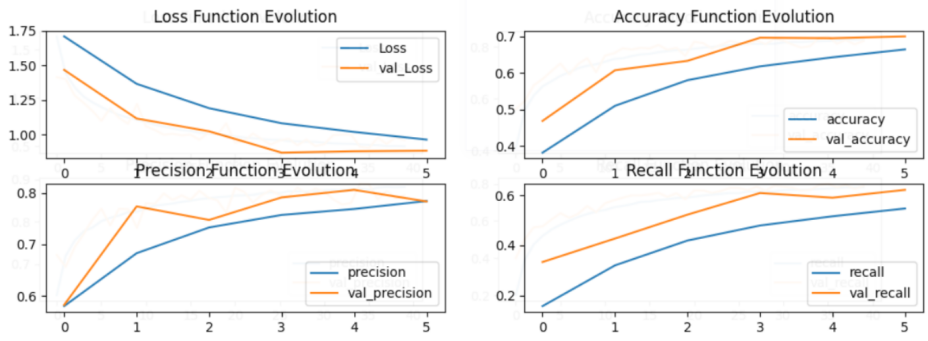
     - ❖ Test Model 2, dengan tingkat akurasi 82 %

       40 *Epochs*, 2 *Layer.*



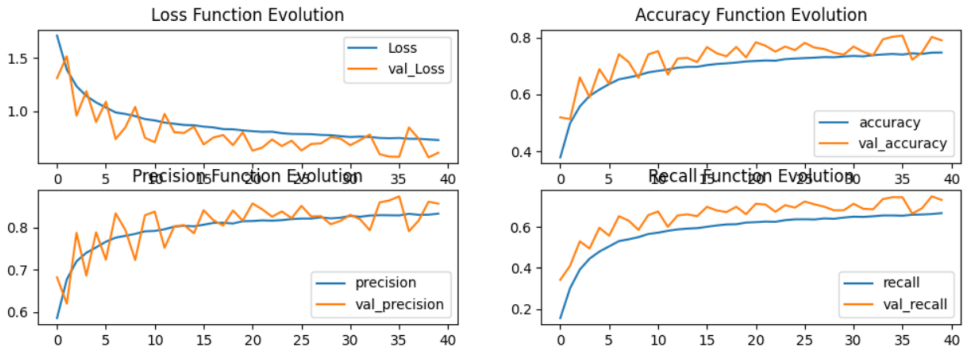
❖ Test Model 3, dengan tingkat akurasi 70 %

40 *Epochs*, 3 *Layer*, 1 tambahan *dropout layer*, *filter*(*sublayer conv2*d) diganti ke 16-32-64.



❖ Test Model 4, dengan tingkat akurasi 79%

40 *Epochs*, 3 *Layer*, *Conv2D filter* mulai dari 32-64-128, density 64 neurons.



2. Variasi jumlah data per kelas (6000, 3000)

   Tidak bisa dilakukan, dikarenakan *dataset* CIFAR 10 setiap batch image disimpan dalam *file extension* berbentuk berbentuk .bin yang dimana sulit untuk dimanipulasi isinya.

- Kesimpulan :

Proyek kami berbasis *Image Classification* akan memerlukan *epoch* yang lebih banyak, jika *epoch* sedikit maka tingkat akurasinya akan kecil, semakin tinggi *epoch* maka tingkat akurasi akan semakin tinggi. Bagaimana cara menyusun *layer* juga mempengaruhi tingkat akurasi, secara *general* bagaimana menyusun *layer* hanya datang dari pengalaman atau hasil mengutak atik parameter yang sudah dijelaskan sebelumnya diatas.