Tugas #1:

Nama : Varuliantor Dear
NIM : 33218010

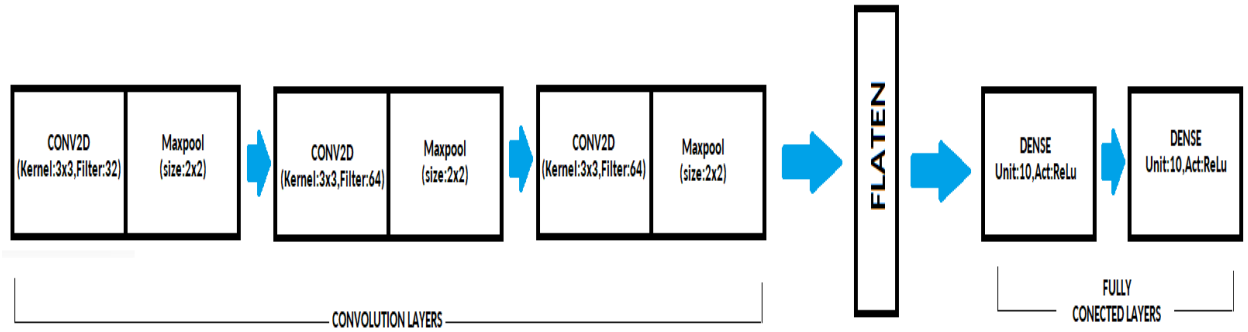
Eksplorasi Hyperparameter Convolutional Neural Network dan Neural Network

A. Classification Problems

Untuk medapatkan pemahaman tentang Deep Learning beserta parameter yang dapat dioptimalisasikan, dalam tugas ini dilakukan Eksplorasi Model Convolutional Neural Network (CNN) dengan datasets CIFAR10 dalam proses klasifikasi identifikasi nama gambar.

A.1. Arsitektur CNN

Dari proses eksplorasi model CNN yang sederhana diperoleh arsitektur CNN yang optimal dengan ilustrasi disajikan pada Gambar A.1.



Gambar A.1. Arsitektur CNN sederhana yang optimal dari proses eksplorasi

Arsitektur yang dirancang tersebut terdiri dari:

- 3 Layer Convolutional (3x3, Filter 32,64,64) with Maxpool
- 2 Dense Layer 10 Unit with ReLU activation
- Dimensi image data 32x32
- Optimizer Adam default Learning rate
- Sparse Categorical Entropy Probabilistic Loss

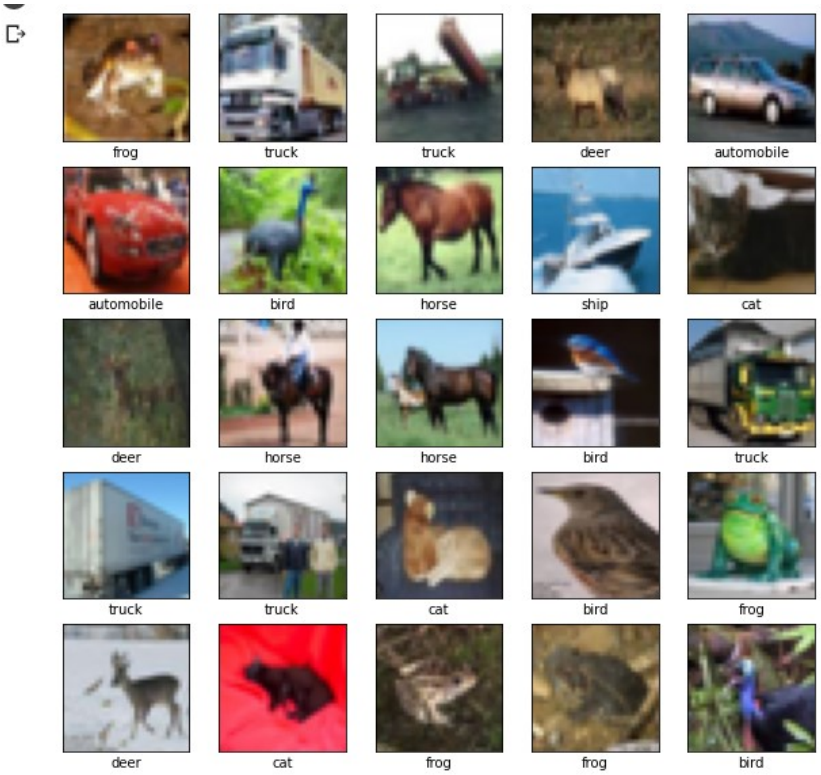
Spesifikasi dari model yang dibangun disajikan pada Tabel A.1.

Tabel A.1. Summary Model CNN yang digunakan

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_3 (MaxPooling 2D)	(None, 15, 15, 32)	0
conv2d_4 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_4 (MaxPooling 2D)	(None, 6, 6, 64)	0
conv2d_5 (Conv2D)	(None, 4, 4, 64)	36928
flatten_1 (Flatten)	(None, 1024)	0
dense_2 (Dense)	(None, 10)	10250
dense_3 (Dense)	(None, 10)	110

Total params: 66,680
Trainable params: 66,680
Non-trainable params: 0

Proses pengujian menggunakan data set CIFAR10 yang merupakan klasifikasi 10 jenis objek dalam proses training maupun validasi dengan contoh gambar disajikan pada Gambar A.2

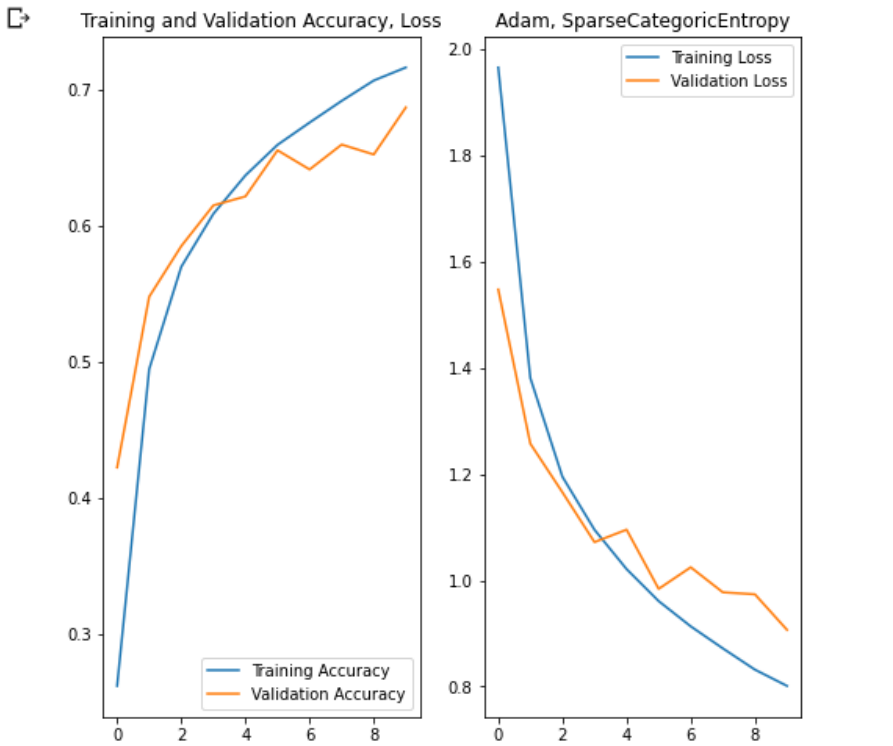


Gambar A.2. Contoh gambar data set CIFAR10 yang digunakan dallam pembuatan model CNN

Performa dari model yang dibangun menghasilkan tingkat akurasi yang mencapai 69% dengan proses iterasi training dan validasi berjumlah 10 epoch. Setiap epoch memerlukan durasi mencapai 4 detik menggunakan google colab. Gambar dari salah satu hasil proses training disajikan pada Gambar A.3 dengan grafik performa disajikan pada Gambar A.4.

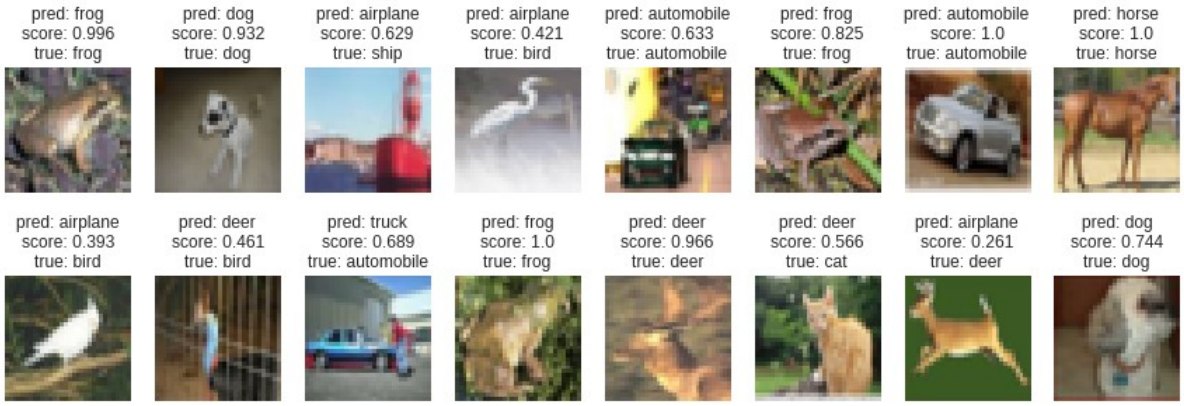
```
Epoch 1/10
1563/1563 [==============================] - 74s 47ms/step - loss: 1.9658 - accuracy: 0.2619 - val_loss: 1.5479 - val_accuracy: 0.4224
Epoch 2/10
1563/1563 [==============================] - 69s 44ms/step - loss: 1.3811 - accuracy: 0.4946 - val_loss: 1.2575 - val_accuracy: 0.5478
Epoch 3/10
1563/1563 [==============================] - 68s 44ms/step - loss: 1.1952 - accuracy: 0.5698 - val_loss: 1.1659 - val_accuracy: 0.5850
Epoch 4/10
1563/1563 [==============================] - 68s 44ms/step - loss: 1.0954 - accuracy: 0.6087 - val_loss: 1.0721 - val_accuracy: 0.6149
Epoch 5/10
1563/1563 [==============================] - 68s 44ms/step - loss: 1.0213 - accuracy: 0.6370 - val_loss: 1.0956 - val_accuracy: 0.6215
Epoch 6/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.9610 - accuracy: 0.6594 - val_loss: 0.9841 - val_accuracy: 0.6554
Epoch 7/10
1563/1563 [==============================] - 68s 43ms/step - loss: 0.9135 - accuracy: 0.6758 - val_loss: 1.0247 - val_accuracy: 0.6413
Epoch 8/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.8720 - accuracy: 0.6917 - val_loss: 0.9776 - val_accuracy: 0.6596
Epoch 9/10
1563/1563 [==============================] - 67s 43ms/step - loss: 0.8320 - accuracy: 0.7067 - val_loss: 0.9738 - val_accuracy: 0.6523
Epoch 10/10
1563/1563 [==============================] - 67s 43ms/step - loss: 0.8014 - accuracy: 0.7162 - val_loss: 0.9069 - val_accuracy: 0.6869
```

Gambar A.3 Contoh hasil training model CNN dengan 10 Epoch



Gambar A.4. Grafik perform model CNN serta tingkat akurasi yang dicapai

Pengujian dari model yang dibangun menghasilkan proses prediksi yang salah dan benar dengan contoh seperti yang disajikan pada Gambar A.5.



Gambar A.5. Contoh hasil prediksi yang dilakukan

A.2 Penjelasan kode program

Inisiasi system dengan melakukan import library yang dibutuhkan

```
import tensorflow as tf
from tensorflow import keras
#from tensorflow.keras import datasets, layers, models, optimizers
from keras import datasets
from keras import layers
from keras import models
from keras import losses
from keras import optimizers
from keras import metrics
import matplotlib.pyplot as plt
from numpy import mean
from numpy import std
```

Menyiapkan Dataset dari CIFAR 10 sebagai data training dan data test

```
[ ] (train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

    # Normalize pixel values to be between 0 and 1
    train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 11s 0us/step
170508288/170498071 [==============================] - 11s 0us/step
```

Menyiapkan Dataset dari CIFAR 10 sebagai data training dan data test

```
[ ] (train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

    # Normalize pixel values to be between 0 and 1
    train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 11s 0us/step
170508288/170498071 [==============================] - 11s 0us/step
```

Klasifikasi dan Verifikasi Dataset yang digunakan

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,  # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

–

Membuat based Model CNN dengan dua convotunial hidden layer

```
[ ] model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Memeriksa model yang dibuat

```
[ ] model.summary()
```

Menambahkan Dense Layer

```
[ ] model.add(layers.Flatten())
    model.add(layers.Dense(10, activation='relu'))
    model.add(layers.Dense(10))
```

Periksa kembali based model yang telah dibangun dengan menambahkan Dense layer

```
model.summary()
```

COMPILE dan LATIH MODEL yang dibangun untuk analisis performa (**PILIH SALAH SATU OPSI** untuk pengujian yang dibutuhkan)

OPSI PERTAMA

Probabilistic CategoricalCrossEntropy

```
[ ] # Opsi satu CategoricalCrossEntropy
    model.compile(optimizer='Adam',
                  loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
                  metrics=['accuracy'])
```

Agar dapat memeriksa dengan tipe loss probability CategoricalCrossentropy maka digunakan kode dibawah ini dengan megubah calsifikasi tipe integers menjadi one hot representation

```
[ ] from tensorflow.keras.utils import to_categorical #impor fungsi to_categorical agar fiting process dapat dilakuakn
    epochs=10
    history= model.fit(train_images,to_categorical(train_labels),epochs=epochs)#, validation_data=(test_images, to_categorical(test_labels)), verbose=2)
```

```
history_dict = history.history   # Memeriksa fitur yang ada di history agar bisa dibuat plotnya
print(history_dict.keys())

acc = history.history['accuracy']
loss = history.history['loss']
epochs_range = range(epochs)

plt.figure(figsize=(4, 4))
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, loss, label='Training Loss')
#plt.ylim([0.1, 1])
plt.legend(loc='lower right')
plt.title('Training Accuracy and Loss, Adam, CategoricalCrossentropy')
```

```
[ ] test_loss, test_acc = model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
    print(test_acc)
```

OPSI KEDUA

Probilistic SparseCategoricalCrossEntropy

```
[ ] model.compile(optimizer='Adam',
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics=['accuracy'])
```

Probabilistic Loss Poisson

```
[ ]  model.compile(optimizer='Adam',loss=tf.keras.losses.Poisson(),metrics=['accuracy'])
```

Probabilistic Loss BinaryCrossEntropy

```
[ ]  model.compile(optimizer='Adam',loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),metrics=['accuracy'])
```

Probabilistic Loss KL Divergence

```
[ ]  model.compile(optimizer='Adam',loss=tf.keras.losses.KLDivergence(),metrics=['accuracy'])
```

SETING TIPE OPTIMIZER dan learning rate (Option untuk variabel learning rate bukan Default)

UNTUK MENGUJI LEARNING RATE

```
[ ]  opt = keras.optimizers.Adam(learning_rate=5)
     epochs=10
     model.compile(optimizer=opt,
                   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                   metrics=['accuracy'])
```

MODEL FIT untuk test performa

```
epochs=10
history = model.fit(train_images, train_labels, epochs=epochs, validation_data=(test_images, test_labels))
```

Evaluasi Model dan Plot hasil

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
#plt.ylim([0.1, 1])
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy, Loss')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
#plt.ylim([0.1, 1])
plt.legend(loc='upper right')
plt.title('Adam, SparseCategoricEntropy')
plt.show()


test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print(test_acc)
```

*TEST MODEL *yang dibangun dengan Image sendiri

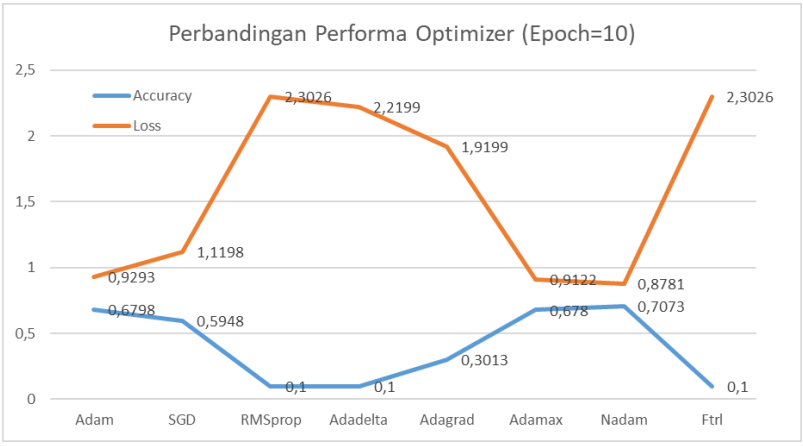
```
import numpy as np
from google.colab import files
from keras.preprocessing import image
import matplotlib.image as mpimg

uploaded = files.upload()  #Pilih file yang diinginkan

for fn in uploaded.keys():

  # predicting images
  path = '/content/' + fn
  img = image.load_img(path, target_size=(32, 32))   # Memaksa ukuran image sesuai dengan dimensi yang diperlukan
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)
  plt.imshow(img)

  images = np.vstack([x])

  prediction_scores = model.predict(np.expand_dims(img, axis=0))
  #classes = model.predict(images, batch_size=10)
  #print(classes[0])
```

```
print(prediction_scores)
#print(class_names)# = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
score = tf.nn.softmax(prediction_scores[0])
predicted_index = np.argmax(prediction_scores)
print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)],  np.max(score))
)
```

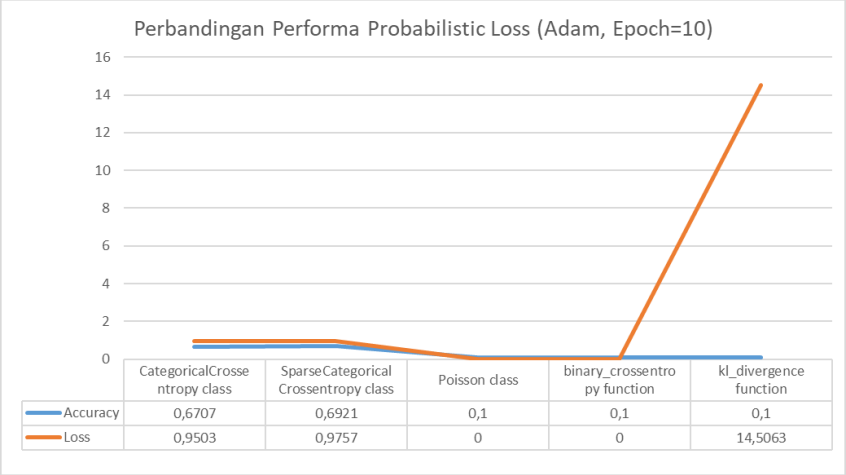
```
[[ 3574.1099  5073.635   3365.7065  1304.514   1346.8253  2594.3596
  -5399.4146  4955.9224  1398.4666  8449.8   ]]
This image most likely belongs to truck with a 1.00 percent confidence.
```

A.3. Eksplorasi Hyperparameter

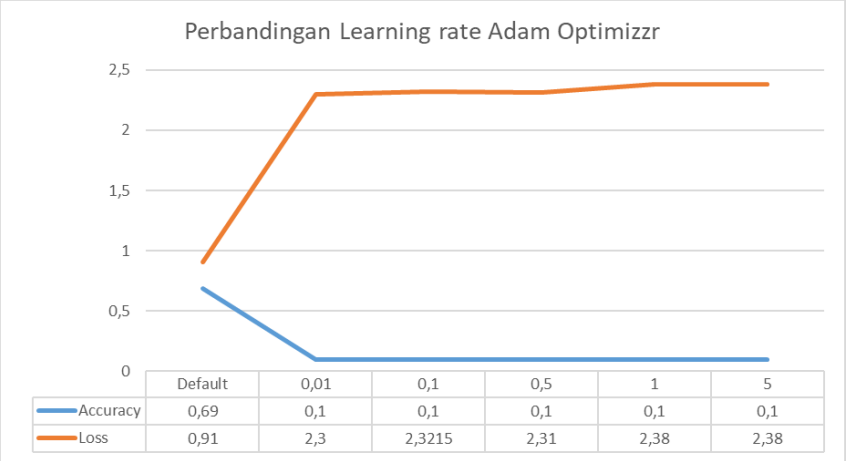
Proses eksplorasi hyperparameter yang dilakukan meliputi perubahan jumlah convotional layer, Nilai Learning rate, Tipe Optimizer, dan Tipe Probabilistic Loss. Pada kode program tersaji penggunaan variabel-variabel tersebut agar dapat direplika kembali oleh pembaca. Dibawah ini adalah grafik hasil analisis dari eksplorasi hyperparameter.



A.6. Hasil dari proses perbandingan tipe Optimizer



A.7. Hasil dari proses perbandingan tipe Probibilistic Loss



A.8. Perbandingan nilai Learning rate dengan tipe Otimizer Adam dan SparseCategoricalEntropy probibilistic Loss

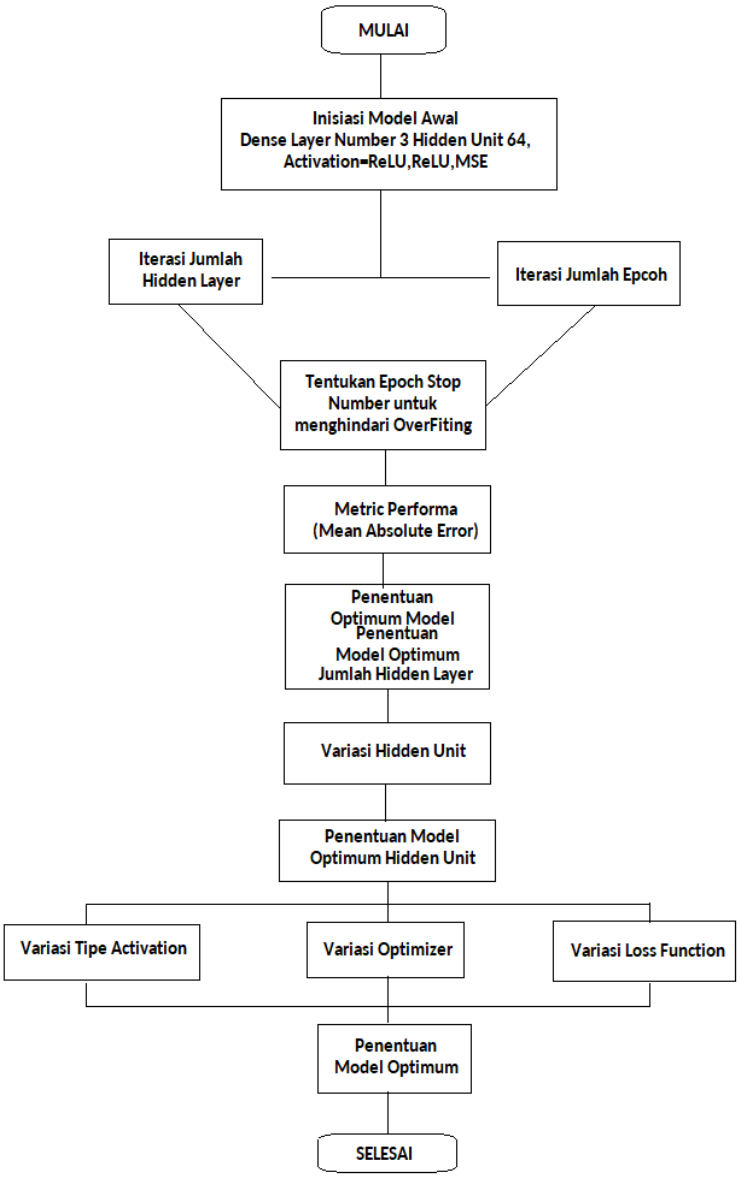
Beberapa contoh hasil pengujian performa disajikan pada lampiran

A.4. Kesimpulan

Model CNN sederhana yang dibangun dengan menggunakan dataset dari CIFAR10 untuk proses training memiliki performa optimal dengan jumlah 3 Convolution layer dan 2 Dense Layer serta 10 epoch dengan menggunakan Adam optimizer dan learning rate default serta SparseCategorical Entropy Probabilistic Loss. Hasil eksplorasi parameter dengan mengubah nilai vairabel learning rate , tipe Optimizer dan tipe Probilistic Loss menunjukkan bahwa konfigurasi CNN perlu dilakukan dengan cermat dan tidak linear dengan kompleksitas arsitektur yang digunakan. Secara teknis penerapan perbedaan pemilihan tipe optimizer dan probabilistic loss harus juga disesuaikan dengan proses fitting model sehingga dapat dilakukan pengujian performa arsitektur yang dibuat. Potensi dari peningkatan performa masih dapat dilakukan dengan variabel yang difokuskan meliputi resolusi data set, tipe optimizer, dan tipe probabilistic loss namun dengan mempertimbangkan kompromi waktu proses yang diperlukan dan resolusi tingkat akurasi yang dicapai.

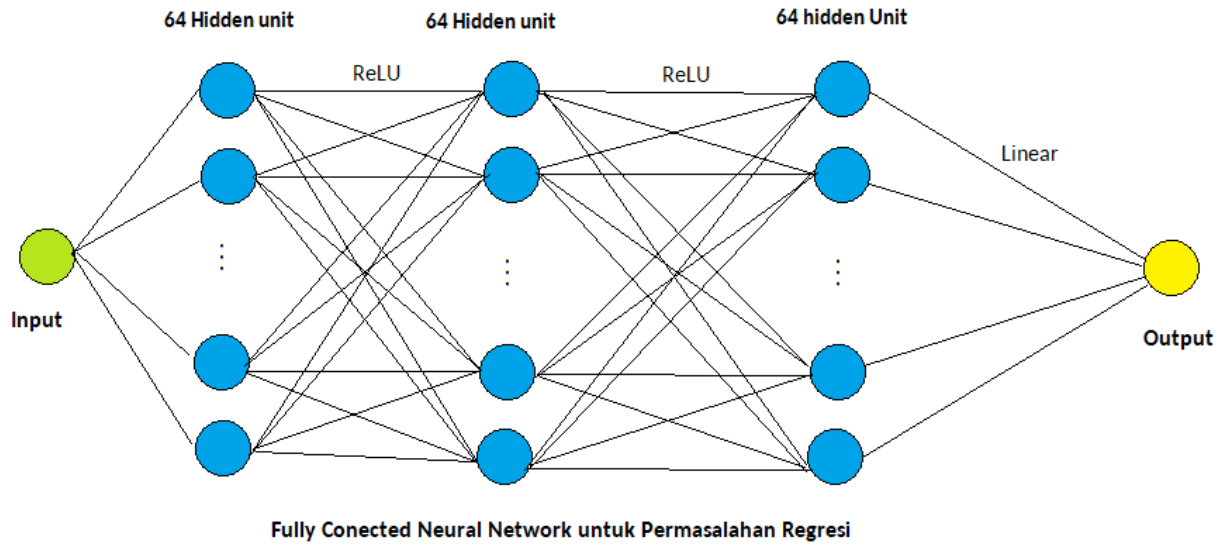
B. Regression Problems

Untuk tugas ini digunakan model Fully Connected Neural Network Layer dan data set Boston Housing Price dengan tujuan melakukan prediksi nilai kontinue dari data diskrit. Salah satu contoh dari aplikasi yang dapat menggunakan model ini adalah prediksi harga rumah. Parameter performa model dilihat dari nilai Mean Absolute Error (MAE) dengan menggunakan prosedur K-Fold Cross Validation sebanyak 4 untuk proses evaluasi dari model yang dibangun. Diagram alur eksplorasi yang dilakukan disajikan pada Gambar B.1.



Gambar B.1. Diagram alur eksplorasi model Fully Connected Neural Network untuk permasalahan Regresi

B.1. Arsitektur Model

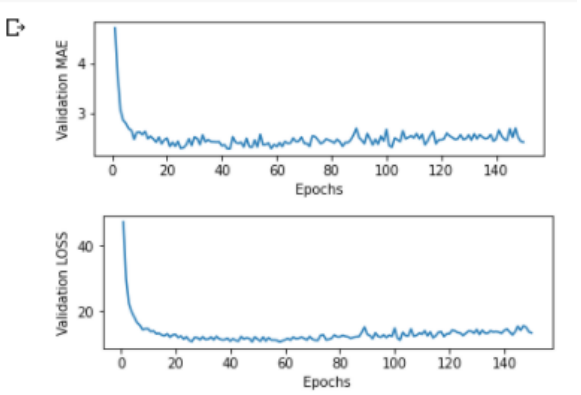


Fully Conected Neural Network untuk Permasalahan Regresi

B.2 Variasi Hidden Layer

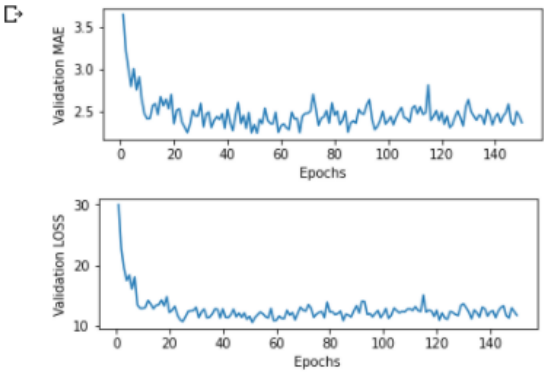
Epoch=150

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 64)                896

 dense_16 (Dense)            (None, 64)                4160

 dense_17 (Dense)            (None, 1)                 65

=================================================================
Total params: 5,121
Trainable params: 5,121
Non-trainable params: 0
_________________________________________________________________
```



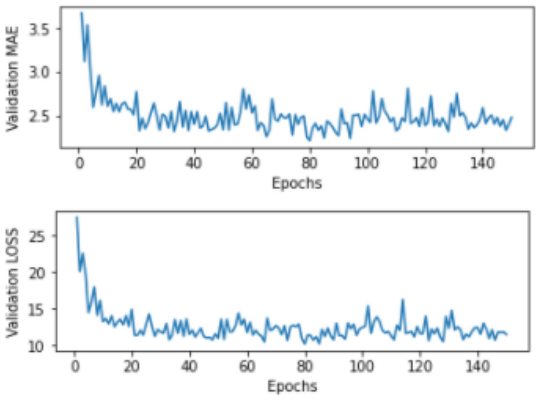
Epoch stop untuk menghindari overfiting=80, MAE=2,79

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_21 (Dense)            (None, 64)                896

 dense_22 (Dense)            (None, 64)                4160

 dense_23 (Dense)            (None, 64)                4160

 dense_24 (Dense)            (None, 1)                 65

=================================================================
Total params: 9,281
Trainable params: 9,281
Non-trainable params: 0
_________________________________________________________________
```



Epoch stop untuk menghindari overfiting=80, MAE=2,357

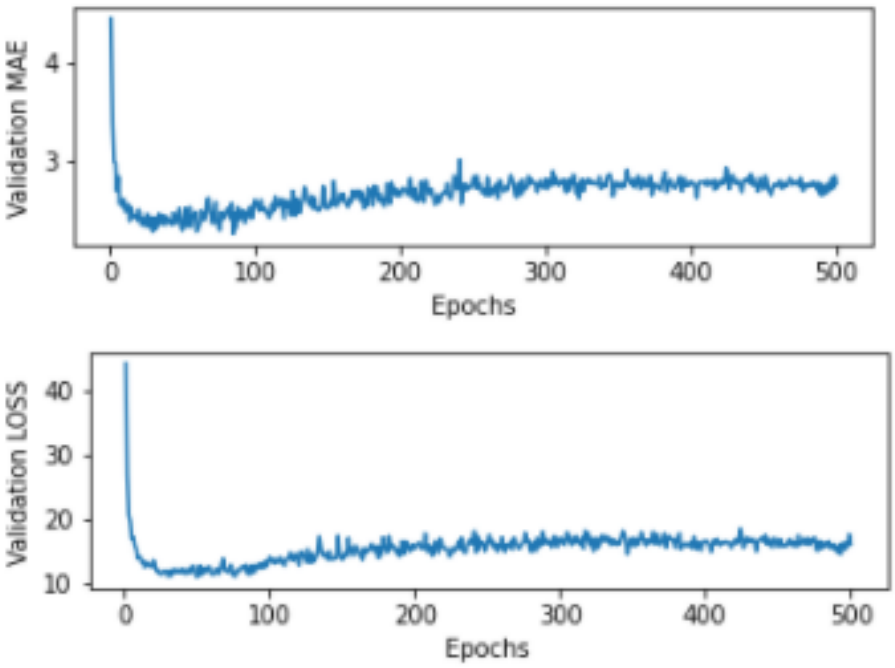
```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_28 (Dense)            (None, 64)                896

 dense_29 (Dense)            (None, 64)                4160

 dense_30 (Dense)            (None, 64)                4160

 dense_31 (Dense)            (None, 64)                4160

 dense_32 (Dense)            (None, 1)                 65

=================================================================
Total params: 13,441
Trainable params: 13,441
Non-trainable params: 0
_________________________________________________________________
```



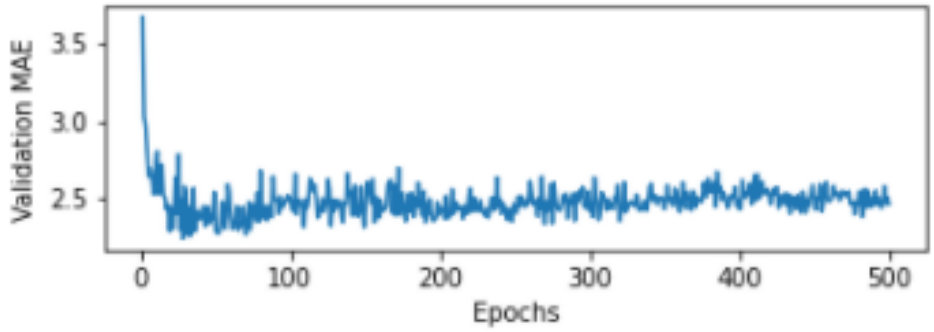
Epoch stop untuk menghindari overfiting=80, MAE=2,451

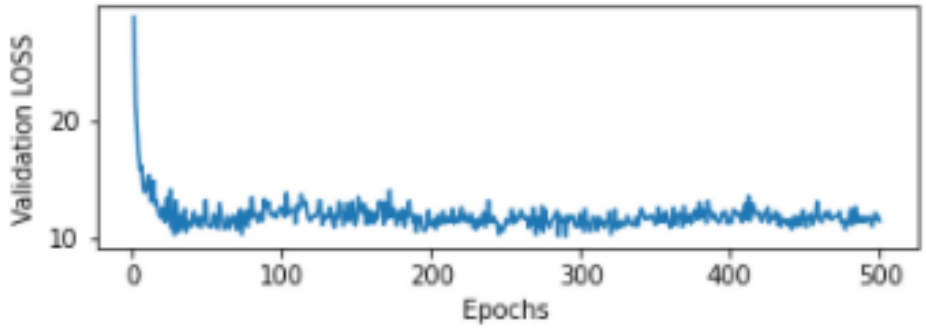
Epoch=500

- 3 Hidden Layer: Terjadi overfiting pada saat epoch mulai dari 80

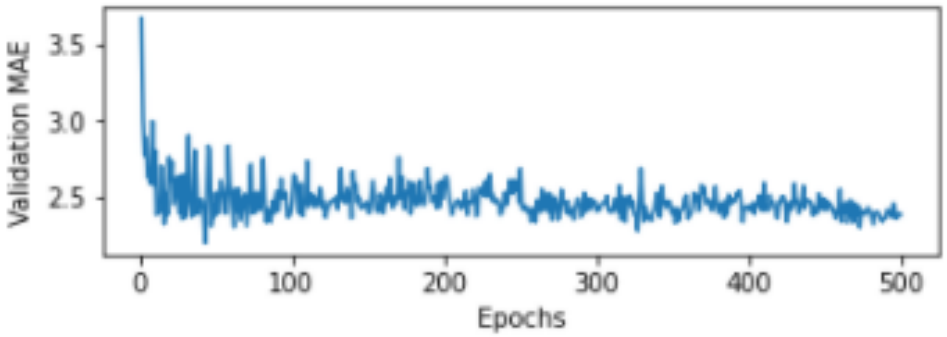


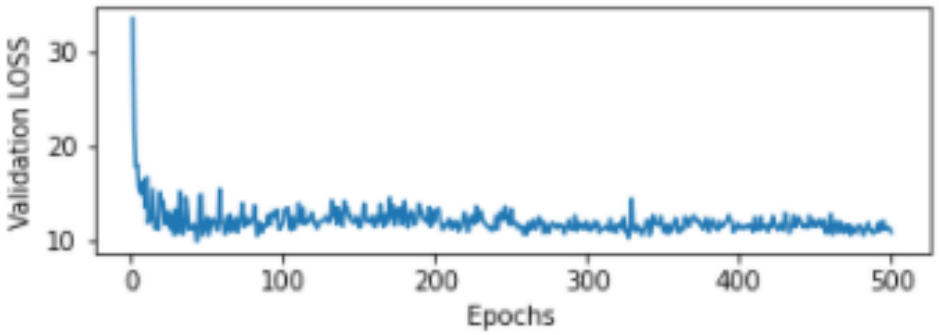
- 4 Hidden Layer: Fluktuasi MAE saat epoch mencapai 80 namun cenderung stabil





- 5 Hidden Layer: Fluktuasi MAE terjadi sebelum Epoch 350

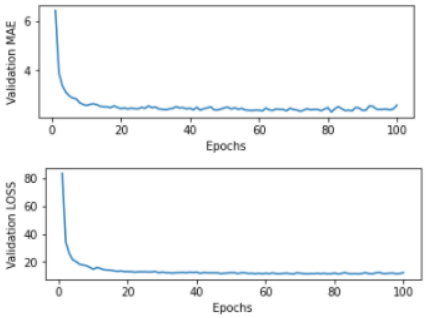




Berdasarkan hasil ini dipilih model dengan 3 Hidden Layer karena jumlah Epoch yang tidak perlu besar untuk mendapatkan nilai MAE yang rendah

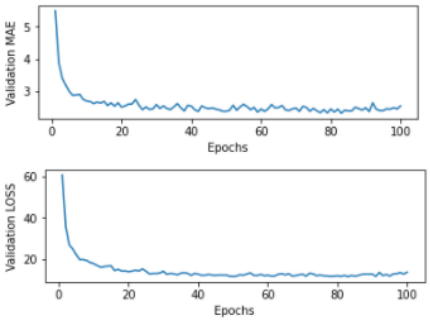
B.3 Variasi Hidden Unit

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 32)                448

 dense_16 (Dense)            (None, 32)                1056

 dense_17 (Dense)            (None, 1)                 33

=================================================================
Total params: 1,537
Trainable params: 1,537
Non-trainable params: 0
_________________________________________________________________
```



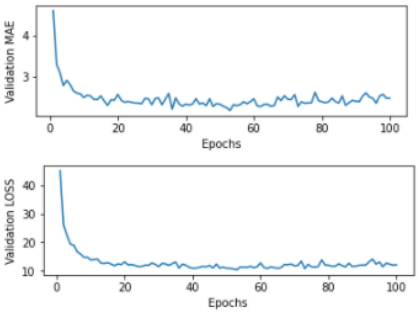
score @epoch 80 = `0s 4ms/step - loss: 19.0235 - mae: 2.8989`

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 32)                448

 dense_13 (Dense)            (None, 64)                2112

 dense_14 (Dense)            (None, 1)                 65

=================================================================
Total params: 2,625
Trainable params: 2,625
Non-trainable params: 0
_________________________________________________________________
```



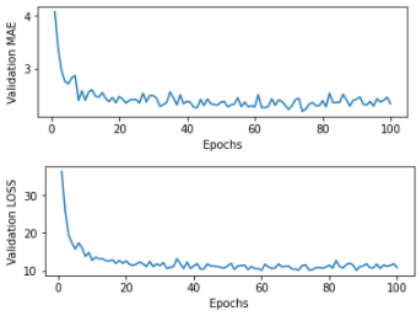
score @epoch 80 = `0s 4ms/step - loss: 19.6326 - mae: 2.6672`

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 64)                896

 dense_16 (Dense)            (None, 64)                4160

 dense_17 (Dense)            (None, 1)                 65

=================================================================
Total params: 5,121
Trainable params: 5,121
Non-trainable params: 0
_________________________________________________________________
```



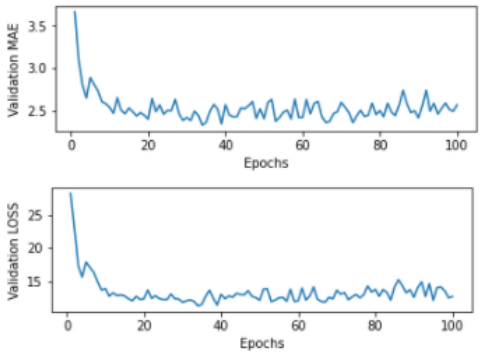
score @epoch 80= `0s 4ms/step - loss: 17.2081 - mae: 2.6820`

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 64)                896

 dense_16 (Dense)            (None, 128)               8320

 dense_17 (Dense)            (None, 1)                 129

=================================================================
Total params: 9,345
Trainable params: 9,345
Non-trainable params: 0
_________________________________________________________________
```



score @epoch 80= `0s 3ms/step - loss: 17.4700 - mae: 2.6033`

```
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_15 (Dense)             (None, 128)               1792

 dense_16 (Dense)             (None, 128)               16512

 dense_17 (Dense)             (None, 1)                 129

=================================================================
Total params: 18,433
Trainable params: 18,433
Non-trainable params: 0
_________________________________________________________________
```

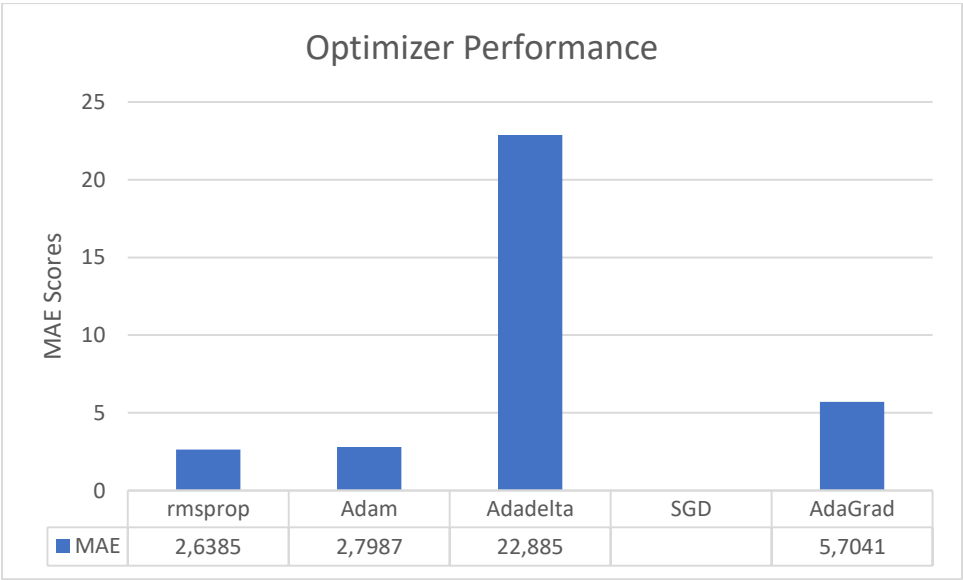


score @epoch 80= `- 0s 3ms/step - loss: 19.2725 - mae: 2.8726`

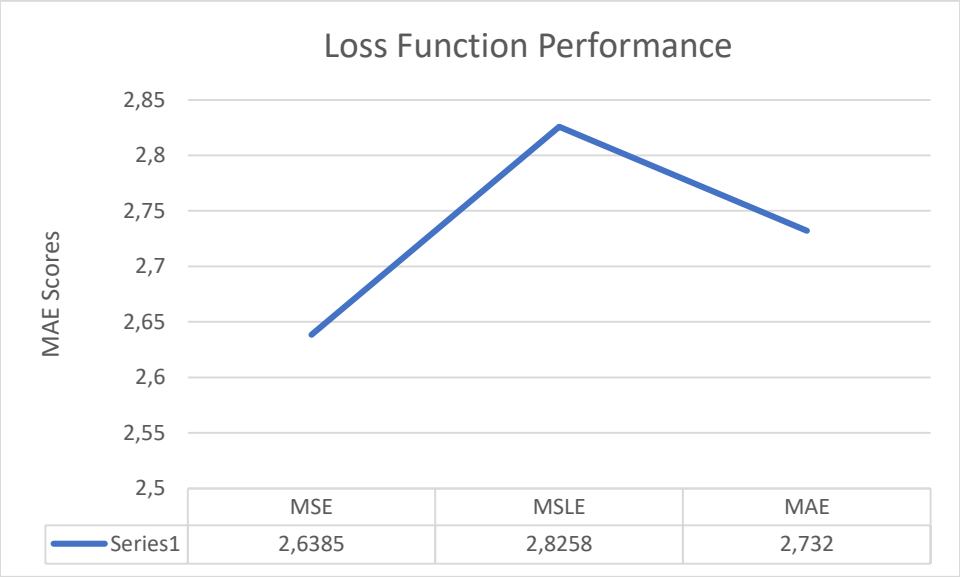
B.4 Variasi fungsi aktivasi

Activation Combination		
Dense 1	**Dense 2**	**MAE**
ReLU	TanH	2,4512
ReLU	Sigmoid	2,7332
tanH	Sigmoid	2,751
ReLU	Linear	2,486
tanH	ReLU	2,507
ReLU	ReLU	2,6385

B.5 Variasi Optimizer



B.6 Variasi Loss Function



B.6. Kesimpulan

Permasalahan regresi dilakukan dengan mempelajari data set yang digunakan untuk menghasilkan keluaran dengan nilai error seminim mungkin. Pemilihan jumlah hidden layer dan unit hidden layer menentukan performa arsitektur neural network yang dibangun dengan metric berupa nilai mean absolute error (MAE). Eksplorasi hyperparameter yang dilakukan meliputi jumlah dense layer, junlah unit hidden layer, tipe optimasi, dan tipe prbabilistic loss memiliki hasil performa yang berbeda walaupun untuk beberapa tipe tidak terlalu signinfikan. Selain variasi dari hyperparamter tersebut, jumlah epoch yang digunakan dalam proses training model fit juga dapat menunjukkan performa secara komperhensive. Jumlah epoch yang tinggi juga tidak menjamin performa model yang optimal karena adanya proses overfiting. Untuk mengatasi hal tesebut maka digunakan teknik early stoping. Pada arsitektur yang dirancang hal ini dilakukan dengan mebatasi epoch dari fiting model dengan besar 80 iterasi yang mengacu pada grafik performa test dan validasi dengan epoch yang mencapai 500. Metric model yang optimal juga ditentukan oleh convergen time sehingga menjadi indikator proses penentuan hyperparamter model yang dibangun.

B.7 Penjelasan Kode Program

Inisiasi sistem

```
[3]  import tensorflow as tf
     from tensorflow import keras
     #from tensorflow.keras import datasets, layers, models, optimizers
     from keras import datasets
     from keras import layers
     from keras import models
     from keras import losses
     from keras import optimizers
     from keras import metrics
     import matplotlib.pyplot as plt
     from numpy import mean
     from numpy import std
     import numpy as np
```

Menyiapkan data dari Boston Pricing House

```
     from keras.datasets import boston_housing
     (train_data, train_targets), (test_data, test_targets) =  boston_housing.load_data()
```

Training Data dan membentuk data untuk diolah

```
     train_data[1],train_data.shape
     train_targets
```

Mengolah data set : Mean dan standar deviasi

```
[6]  mean = train_data.mean(axis=0)
     train_data -= mean
     std = train_data.std(axis=0)
     train_data /= std

     test_data -= mean
     test_data /= std
```

Definisikan model yang akan dibangun

```
[7]  def build_model():
         # Because we will need to instantiate
         # the same model multiple times,
         # we use a function to construct it.
         model = models.Sequential()
         model.add(layers.Dense(64, activation='relu',
                                input_shape=(train_data.shape[1],)))
         model.add(layers.Dense(64, activation='linear'))
         model.add(layers.Dense(1))
         model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
         return model
```

Membuat model yang telah didefinisikan

```
[8]  model = build_model()
```

Summary Model yang dibangun

```
[9]  model.summary() #Periksa model yang telah dibangun
```

Model Validation dengan prosedur K-Fold, yaitu pengulangan test tanpa adanya overlaping antara tiap "folded"

```
from keras import backend as K
K.clear_session() # Clean memory K-Fold yang telah digunakan
k=4
num_val_samples = len(train_data) // k
num_epochs = 100
all_mae_histories = []
all_los_histories = []
for i in range(k):
    print('processing fold #', i)
    # Prepare the validation data: data from partition # k
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]

    # Prepare the training data: data from all other partitions
    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0)

    # Bangun kembali Keras model (already compiled)
    model = build_model()
    # Train the model (in silent mode, verbose=0)
    history = model.fit(partial_train_data, partial_train_targets,
                        validation_data=(val_data, val_targets),
                        epochs=num_epochs, batch_size=1, verbose=0)
    mae_history = history.history['val_mae']
    los_history=history.history['val_loss']
    all_mae_histories.append(mae_history)
    all_los_histories.append(los_history)
```

Periksa property dari history sehingga dapat dibuat grafik analisisnya

```
[11] history_dict = history.history   # Memeriksa fitur yang ada di history agar bisa dibuat plotnya
     print(history_dict.keys())
```

```
dict_keys(['loss', 'mae', 'val_loss', 'val_mae'])
```

Evaluasi hasil dengan menghitung nilai rata-rata nilai MAE (mean Absolute Error)

```
[12] average_mae_history = [
         np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
     average_los_history = [
         np.mean([x[i] for x in all_los_histories]) for i in range(num_epochs)]
```

Buat grafik hasil sebagai performa

```
[13] import matplotlib.pyplot as plt

     plt.subplot(2, 1, 1)
     plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
     plt.xlabel('Epochs')
     plt.ylabel('Validation MAE')
     plt.show()

     plt.subplot(2, 1, 2)
     plt.plot(range(1, len(average_los_history) + 1), average_los_history)
     plt.xlabel('Epochs')
     plt.ylabel('Validation LOSS')
     plt.show()
```

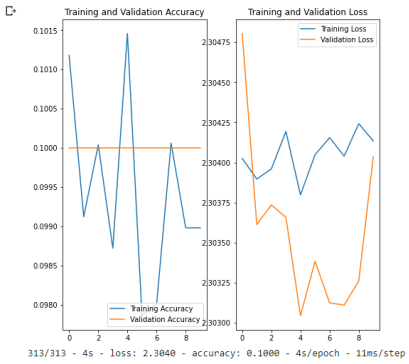
Test perfoma dari data test dengan menentukan jumlah epoch yang merupakan titik awal overfiting (eknik Early Stoping T) berdasarkan grafik nilai MAE

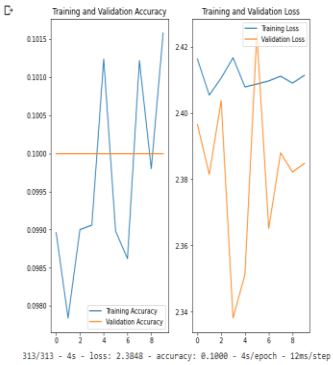
```
# Get a fresh, compiled model.
model = build_model()
# Train it on the entirety of the data.
model.fit(train_data, train_targets,
          epochs=60, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
```

```
test_mae_score
```

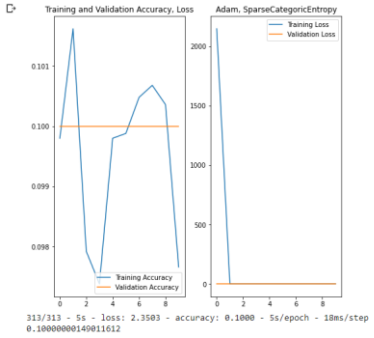
Lampiran hasil eksplorasi dan pembelajaran

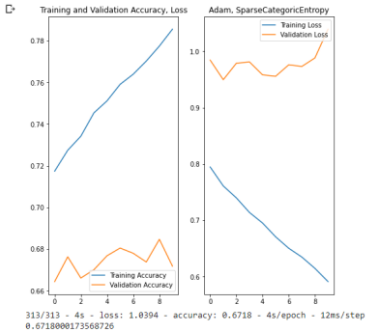
- **Hasil learning rate Keras Adam Optimizer = 0.01, 0.1, dan 1**

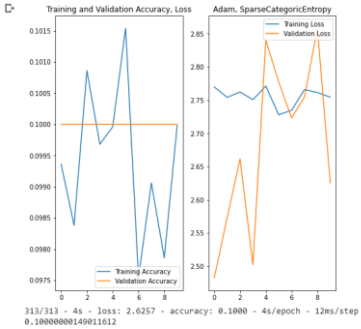




Learning rate 0,05; 0,5, dan 5







Learning rate default values


```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

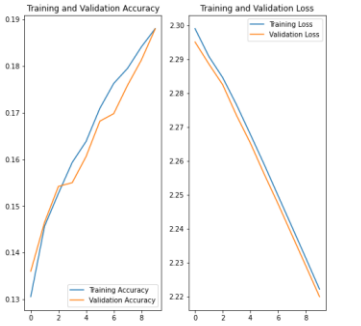
- **Hasil Perbandingan Tipe Optimizer**


```
313/313 - 4s - loss: 1.1198 - accuracy: 0.5948 - 4s/epoch - 11ms/step
```

```
[82] model.compile(optimizer='SGD',
                   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                   metrics=['accuracy'])
```


```
313/313 - 3s - loss: 2.3026 - accuracy: 0.1000 - 3s/epoch - 11ms/step
```

```
[92] model.compile(optimizer='RMSprop',
                   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                   metrics=['accuracy'])
```




```
313/313 - 4s - loss: 2.2199 - accuracy: 0.1880 - 4s/epoch - 11ms/step
```

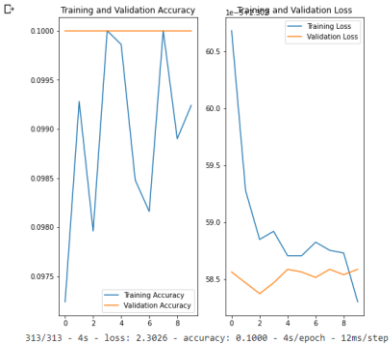
```
[102] model.compile(optimizer='Adadelta',
                    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                    metrics=['accuracy'])
```


```
313/313 - 4s - loss: 1.9199 - accuracy: 0.3013 - 4s/epoch - 11ms/step
```

```
[112] model.compile(optimizer='Adagrad',
                    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                    metrics=['accuracy'])
```


```
model.compile(optimizer='Adamax',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```


```
[141] model.compile(optimizer='Nadam',
                    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                    metrics=['accuracy'])
```




```
[151] model.compile(optimizer='Ftrl',
                    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                    metrics=['accuracy'])
```

- **Variasi Probabilistic Loss**

Variasi Probabilistic losses https://keras.io/api/losses/probabilistic_losses/

- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- Poisson class
- KLDivergence class

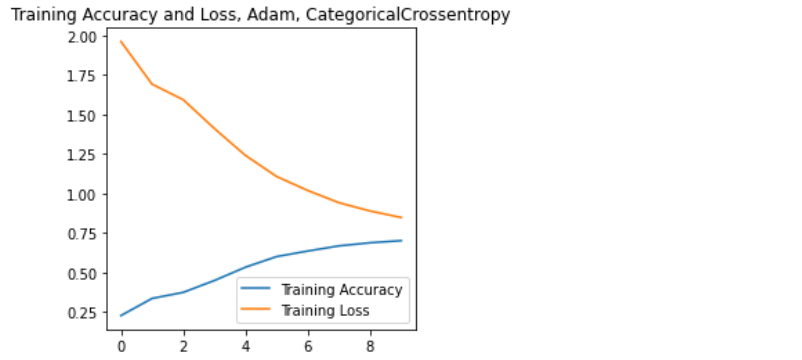
CategoricalCrossentropy

Use this crossentropy loss function when there are two or more label classes. We expect labels to be provided in a one_hot representation. If you want to provide labels as integers, please use SparseCategoricalCrossentropy loss. There should be # classes floating point values per feature.
Use this crossentropy loss function when there are two or more label classes. We expect labels to be **provided as integers**. If you want to provide labels using one-hot representation, please

use CategoricalCrossentropy loss. There should be # classes floating point values per feature for y_pred and a single floating point value per feature for y_true.
Beda tipe antara Sparse dan Categorical crossentropy sehingga model fit nya juga jadi berubah
Pendekatan: ubah model fit atau tidak pakai yang presetansinya one hot
Agar bisa memeriksa Losses tipe categorical makadi buat perubahan menjadi

```
from tensorflow.keras.utils import to_categorical
epochs=10
#history = model.fit(train_images, train_labels, epochs=epochs)
history= model.fit(train_images,to_categorical(train_labels) ,epochs=10)
```

```
dict_keys(['loss', 'accuracy'])
Text(0.5, 1.0, 'Training Accuracy and Loss, Adam, CategoricalCrossentropy')
```

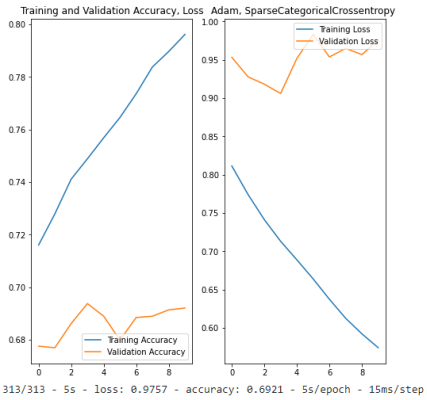



```
test_loss, test_acc = model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
print(test_acc)
```

```
313/313 - 4s - loss: 0.9503 - accuracy: 0.6708 - 4s/epoch - 12ms/step
0.670799970626831
```

Poisson

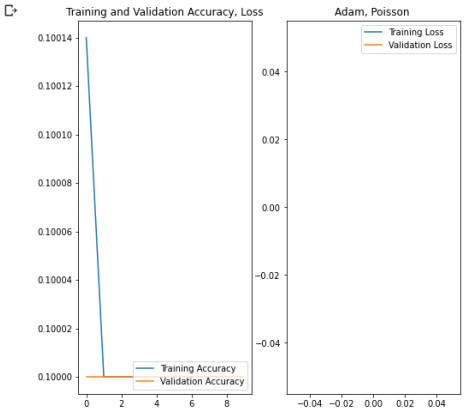
```
[ ]  model.compile(optimizer='Adam',loss=tf.keras.losses.Poisson(),metrics=['accuracy'])
```




```
313/313 - 5s - loss: 0.9757 - accuracy: 0.6921 - 5s/epoch - 15ms/step
```

BinaryCrossEntropy

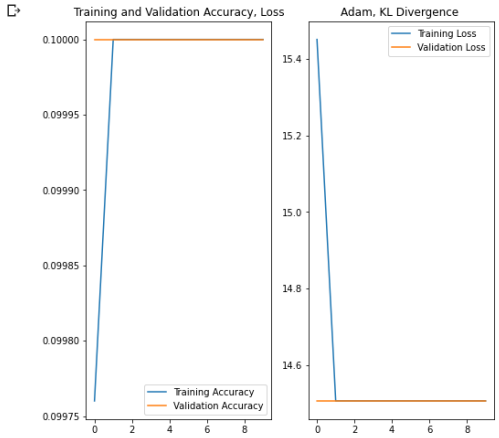
```
[14] model.compile(optimizer='Adam',loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),metrics=['accuracy'])
```




```
313/313 - 4s - loss: nan - accuracy: 0.1000 - 4s/epoch - 14ms/step
0.10000000149011612
```

KL Divergence

```
[8]  model.compile(optimizer='Adam',loss=tf.keras.losses.KLDivergence(),metrics=['accuracy'])
```




```
313/313 - 7s - loss: 14.5063 - accuracy: 0.1000 - 7s/epoch - 22ms/step
0.10000000149011612
```