Klasifikasi Bunga dengan Convolutional Neural Network

Chrystian
18/430257/PA/18770

Karunia Eka Putri
18/430265/PA/18778

Widad Abida Rahmah
18/430275/PA/18788

June 9, 2021

1 Arsitektur

Untuk melakukan klasifikasi bunga dengan Convolutional Neural Network(CNN) kali ini arsitektur model yang kami gunakan berupa Input Layer, 1 Convolution Layer dan 1 Pooling Layer untuk Feature Extraction, dan 1 Hidden Layer serta Output Layer untuk proses Classification.

```
model = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal"),
    tf.keras.layers.experimental.preprocessing.RandomRotation(1),
    tf.keras.layers.Conv2D(n_filters, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(n_hidden_neurons),
    tf.keras.layers.Dense(3, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics='acc')
```

Listing 1: Model

Untuk convolution dan hidden layer perlu akan dilakukan percobaan untuk menentukan model yang paling baik.

1.1 Layer

1.1.1 Convolution Layer

Convolution layer atau layer konvolusi adalah inti dari CNN. Apabila dibayangkan convolution layer adalah seperti saringan berbetuk kotak kecil (filter) yang digeser berulang kali seluas input yang ingin disaring dan berlapis-lapis (ketebalan).

```
tf.keras.layers.Conv2D(8, (3, 3), activation='relu'),
```

Pada baris kode di atas merupakan implementasi convolution layer dalam kode python dengan menggunakan library tensorflow keras. Conv2D tersebut menerima parameter berupa banyaknya atau seberapa tebal convolution layer (n_filters) dan ukuran saringan atau filternya yang berukuran dua dimensi (tinggi dan lebar). Dalam contoh baris kode di atas n_filters yang digunakan adalah 8 dan ukuran filter yang digunakan adalah 3 x 3 dengan fungsi aktivasi yang digunakan adlah relu.

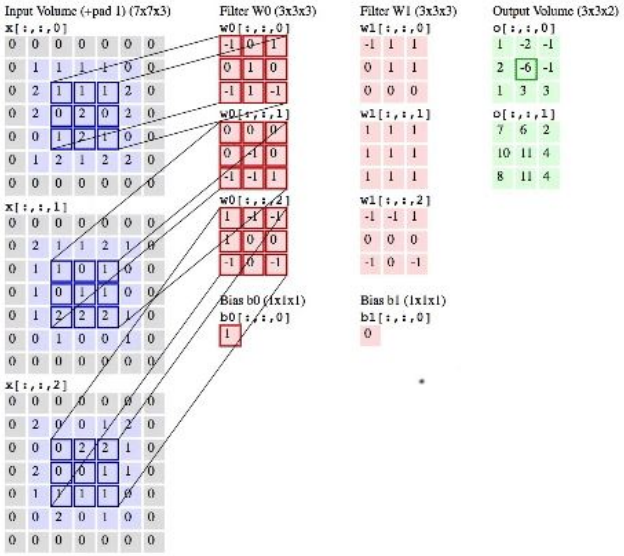

Figure 1: **Ilustrasi Convolution Layer.** Sumber : https://miro.medium.com/proxy/1*bx3kWA2cKm14OrNP1M-6gw.gif

1.1.2 Max Pooling Layer

Setelah melalui convolution layer maka data akan masuk ke pooling layer. Tujuan dari pooling layer adalah untuk mempercepat komputasi dengan mengurangi dimensi dari feature map (output dari convolution layer) sehingga parameter yang harus di update semakin sedikit dan mengatasi overfitting. Pooling layer yang digunakan dalam arsitektur model ini adalah Max Pooling Layer. Max Pooling Layer menghitung nilai maksimal setiap patch dari setiap feature map. Max Pooling mengambil warna pixel yang lebih terang sehingga lebih cocok apabila digunakan untuk input gambar yang memiliki warna background gelap.

```
        tf.keras.layers.MaxPooling2D((2, 2)),
```

Pada baris kode diatas MaxPooling2D mengurangi dimensi feature map 3 x 3 menjadi dimensi 2 x 2.

1.1.3 Dense Layer dan Output Layer

Setelah melalui convolution layer dan pooling layer (proses feature extraction) maka data gambar akan masuk ke klasifikasi. Prinsip dense layer yang digunakan sama seperti dense layer yang digunakan pada MLP kemarin.

```
        tf.keras.layers.Dense(16),
        tf.keras.layers.Dense(3, activation='softmax')
```

Dense layer pada arsitektur ini hanya satu lapis dengan banyak neuron pada hidden layer yaitu 16 dan banyak neuron pada output layer adalah 3 dengan fungsi aktivasi softmax.

1.1.4 Random Flip Layer

```
        tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal"),
```

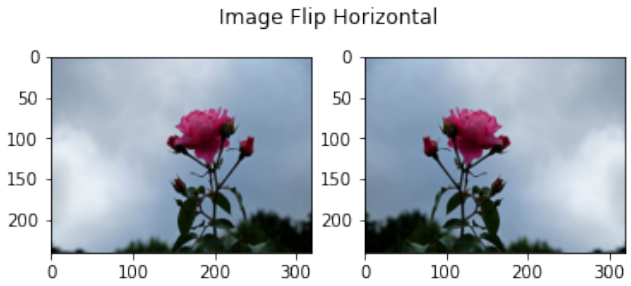

Figure 2: **Visualisasi hasil Random Flip Layer Horizontal.**

Random Flip Layer digunakan untuk melakukan Data Augmentation sebelum gambar dimasukan pada Convolution Layer. Layer ini secara acak membalikkan gambar awal secara horizontal. Layer ini hanya berjalan ketika melakukan training dan akan menghasilkan output yang sama dengan input ketika inference (testing).

1.1.5 Random Rotation Layer

```
tf.keras.layers.experimental.preprocessing.RandomRotation(k),
```

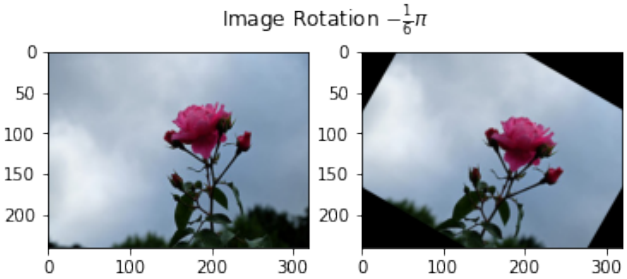

Figure 3: **Visualisasi contoh hasil Random Rotation.**

Random Rotation Layer digunakan untuk melakukan Data Augmentation sebelum gambar dimasukan pada Convolution Layer. Layer ini secara acak akana merotasikan gambar dengan parameter suatu faktor $k \in [0, 1]$. Faktor ini dalam bentuk float merepresentasikan fraction dari 2π. Membuat gambar dapat dirotasi dari $[-k2\pi, k2\pi]$ (nilai postive menandakan rotasi counter clock-wise, dan nilai negative menandakan rotasi clock-wise).

1.2 Aktivasi

1.2.1 ReLU

Salah satu fungsi aktivasi yang bisa digunakan adalah aktivasi ReLU (Rectified Linear Unit. Secara matematis fungsi ReLU dapat didefinisikan sebagai:

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

or

$$f(x) = max(0, x)$$

Fungsi ini digunakan pada hidden layer.

1.2.2 Softmax

Fungsi aktivasi softmax digunakan pada output layer untuk menentukan probabilitas. Probabilitas dihitung dengan cara membagikan hasil eksponen dari output neuron ke-k hidden layer (I_k) dengan penjumlahan hasil eksponen dari output semua neuron hidden layer. Formula aktivasi softmax:

$$y_k = \frac{e^{I_k}}{\sum_\alpha e^{I_\alpha}}, 1 \leq k \leq \alpha$$

1.3 Loss Function

Pada output layer digunakan fungsi Softmax dan Categorical Cross Entropy Loss

$$E = -\sum y_i \log \hat{y}_i$$

$$\hat{y}_i = \frac{e^{x_i}}{\sum_k e^{x_k}}$$

1.4 Adam Optimizer

Adam (Adaptive Momentum Optimizer) adalah optimizer yang menggunakan adaptive learning rates untuk setiap parameter. Tidak hanya menyimpan exponentially decaying average dari squared gradients sebelumnya seperti RMSprop, Adam juga menyimpan decaying average dari gradients, seperti momentum. Analogi momentum seperti bola yang turun dari suatu kemiringan, Adam dapat dibayangkan seperti bola berat yang memiliki friksi, membuat Adam lebih memilih minima yang rata. Untuk menghitung decaying average dari gradient dan squared gradient dapat menggunakan:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Tetapi m dan v memiliki bias menuju ke arah 0, untuk mencegah ini maka dapat menghitung bias-corrected moment estimates:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Pada akhirnya untuk mengupdate parameters, dapat menggunakan:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Dimana penulis menyarankan 0.9 untuk β_1, 0.999 untuk β_2, dan 10^{-8} untuk ϵ.

2 Listing Kode

Berikut ini kode lengkap implementasi CNN dalam Python dengan menggunakan tensorflow:

```
import os, fnmatch
from PIL import Image
import numpy as np
import random

def convertToGrayscale(imageRGB):
    """return grayscale conversion of RGB image"""
    # https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html
    # Basically  Grayscale = 0.299  R   + 0.587  G   + 0.114  B
    # We could easily vectorized and just do the dot product of imageRGB with
[0.299,0.587,0.114]

    imageGrayscale = np.dot(imageRGB, [0.299,0.587,0.114])

    return imageGrayscale



def getImages(label, n=100):
    """return list of (default 100) images for a given label"""

    # Check Dataset
    checkDataset()

    # Try and get path
    path = "data/flowers/" + label.lower()
    if(not os.path.exists(path)):
        print("No label found")
        return -1

    # Get all images address
    allFiles = fnmatch.filter(os.listdir(path), '*.jpg')
    random.seed(42)
    usedFiles = random.sample(allFiles, n) # get 100 samples from allFiles

    # Read the data into numpy array
    imagesRGB = []
```

```
        for file in usedFiles:
            img = Image.open(path+'/'+file)
            img = img.resize((320,240), Image.ANTIALIAS)
            imagesRGB.append(np.asarray(img))

        imagesRGB = np.asarray(imagesRGB) #(n, 240, 320, 3)

        return imagesRGB

    def preprocessImage(images):
        """return np.array to be loaded into a model,
        flatten dimension (100, 240, 320) -> (100, 76800), and
        Normalize data value from int [0, 255] -> float [0,1]"""

        x = images.reshape(*images.shape[:1], -1) #(100, 76800)
        x = x/255.0
        return x
```

Listing 2: Definisi Fungsi-Fungsi

```
    labels = ['Sunflower', 'Dandelion', 'Rose']
    n_data = 100

    X = None
    y = None
    for i in range(len(labels)):
        if(X is None):
            X = getImages(labels[i], n=n_data)
        else:
            X = np.append(X, getImages(labels[i], n=n_data), axis=0)

        if(y is None):
            y = np.full(shape=n_data,fill_value=i,dtype=np.int)
        else:
            y = np.append(y, np.full(shape=n_data,fill_value=i,dtype=np.int), axis=0)
```

Listing 3: Load Image

```
    from sklearn.model_selection import train_test_split
    n_values = np.max(y) + 1
    y = np.eye(n_values)[y]
    print(y.shape)
    X = X/255.0

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42,
     stratify=y)
```

Listing 4: Split Data

```
    model = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal"),
    tf.keras.layers.experimental.preprocessing.RandomRotation(1),
    tf.keras.layers.Conv2D(8, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(16),
    tf.keras.layers.Dense(3, activation='softmax')
])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics='acc')
```

Listing 5: Model

```
    history = model.fit(X_train, y_train, batch_size=5, epochs=30, validation_data=(X_test,
    y_test))
```

Listing 6: Training dan Testing

```
    import tensorflow_docs as tfdocs
    import tensorflow_docs.modeling
    import tensorflow_docs.plots

    models_histories = {'n_filters_08':history,'n_filters_16':history2,'n_filters_32':history3
    , 'n_filters_64':history4, 'n_filters_128':history5}

    from matplotlib.pyplot import figure
```

```
import matplotlib.pyplot as plt

# acc
plotter = tfdocs.plots.HistoryPlotter(metric = 'acc', smoothing_std=1)
figure(figsize=(15, 10), dpi=80)
plotter.plot(models_histories)

# loss
plotter = tfdocs.plots.HistoryPlotter(metric = 'loss', smoothing_std=1)
figure(figsize=(15, 10), dpi=80)
plt.ylim([0, 2])
plotter.plot(models_histories)
```

Listing 7: Visualisasi Akurasi dan Loss

3 Hasil Percobaan

3.1 N Filters

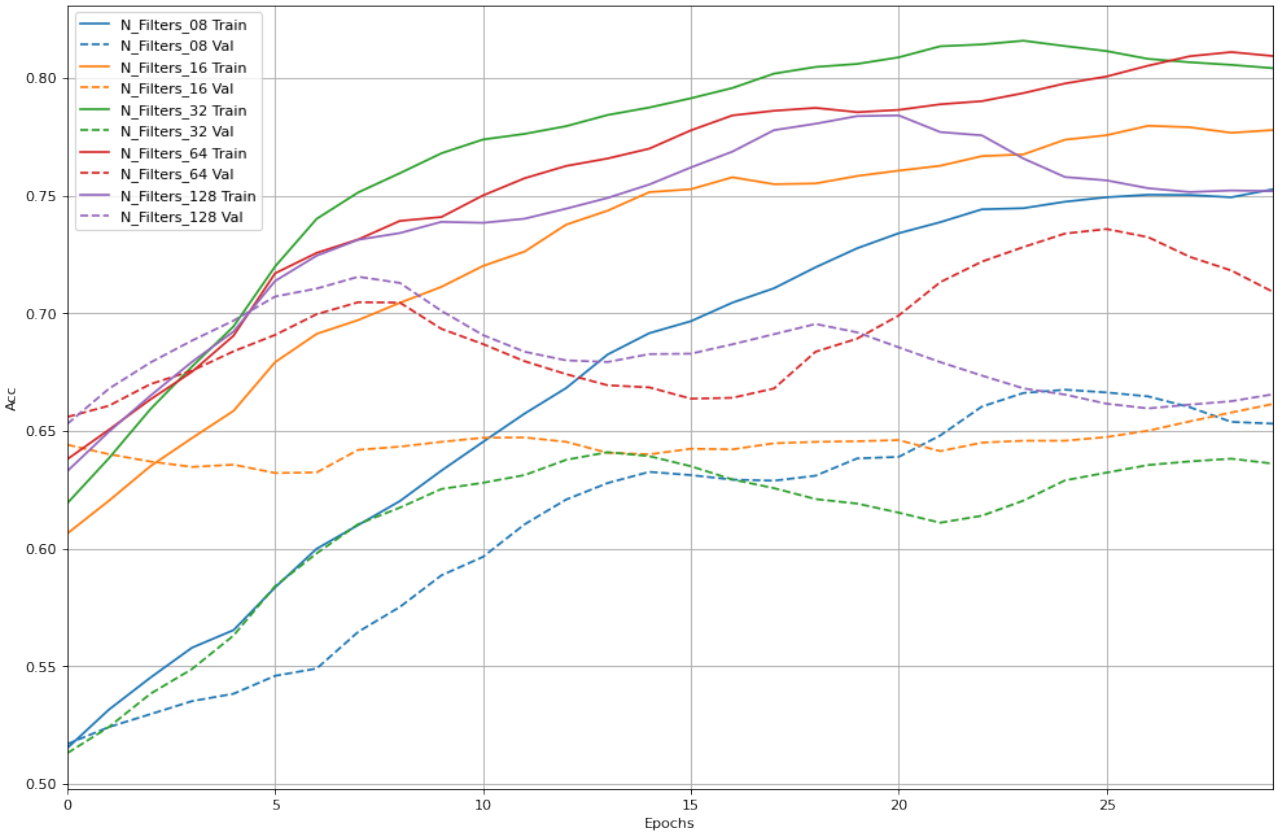

Figure 4: **Visualisasi akurasi model CNN ketika menggunakan N filter yang berbeda.** Filter yang digunakan merupakan bilangan pangkat dua dari 8 sampai 128.

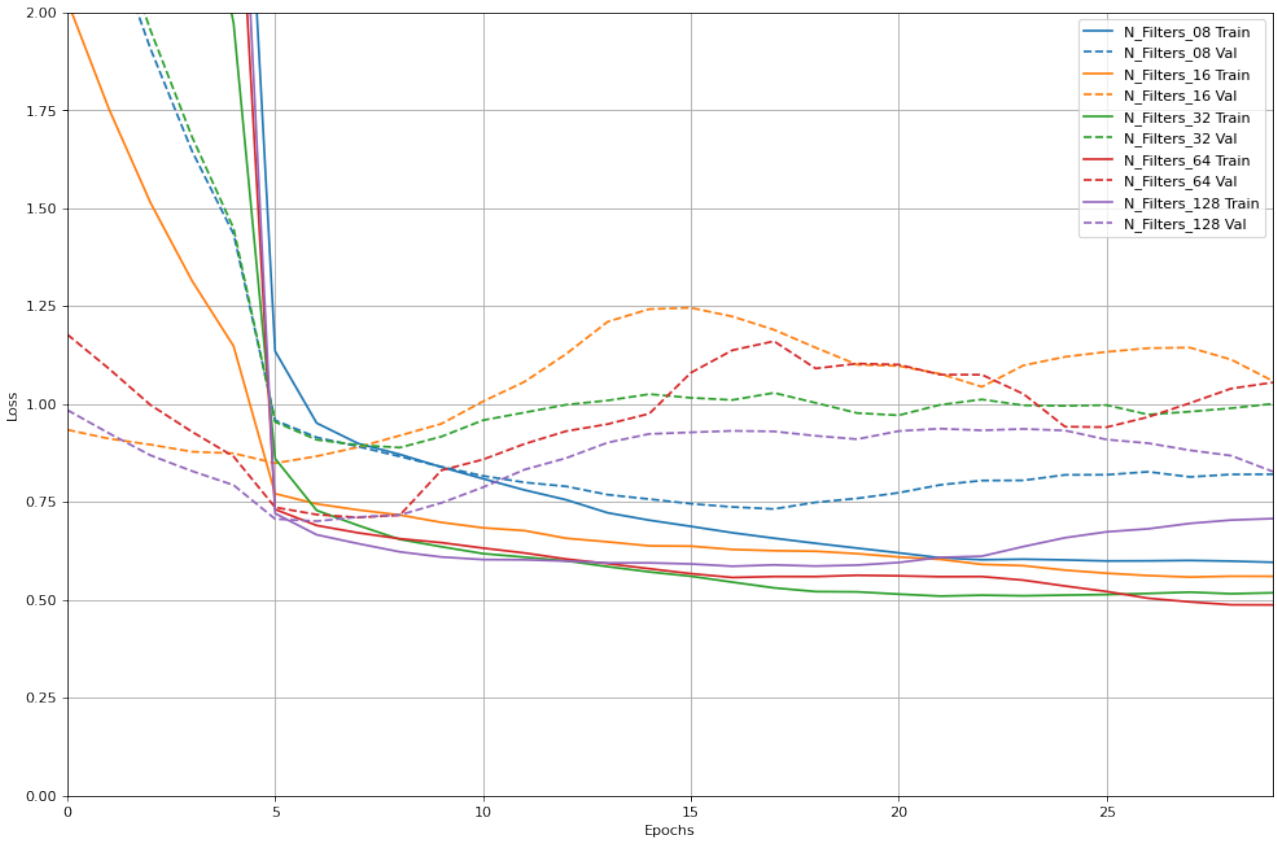

Figure 5: **Visualisasi loss model CNN ketika menggunakan N filter yang berbeda.** Filter yang digunakan merupakan bilangan pangkat dua dari 8 sampai 128.

3.1.1 Tabel Hasil

Model (n_filters)	Train Accuracy	Train Loss	Test Accuracy	Test Loss
8 filter	0.7708	0.5432	0.6333	**0.8369**
16 filter	0.7875	0.5032	0.6333	0.9834
32 filter	0.7875	0.5293	0.6167	1.0333
64 filter	0.8000	**0.5012**	**0.7000**	1.0721
128 filter	**0.8083**	0.6171	0.6667	0.8509

3.2 N Hidden Neuron

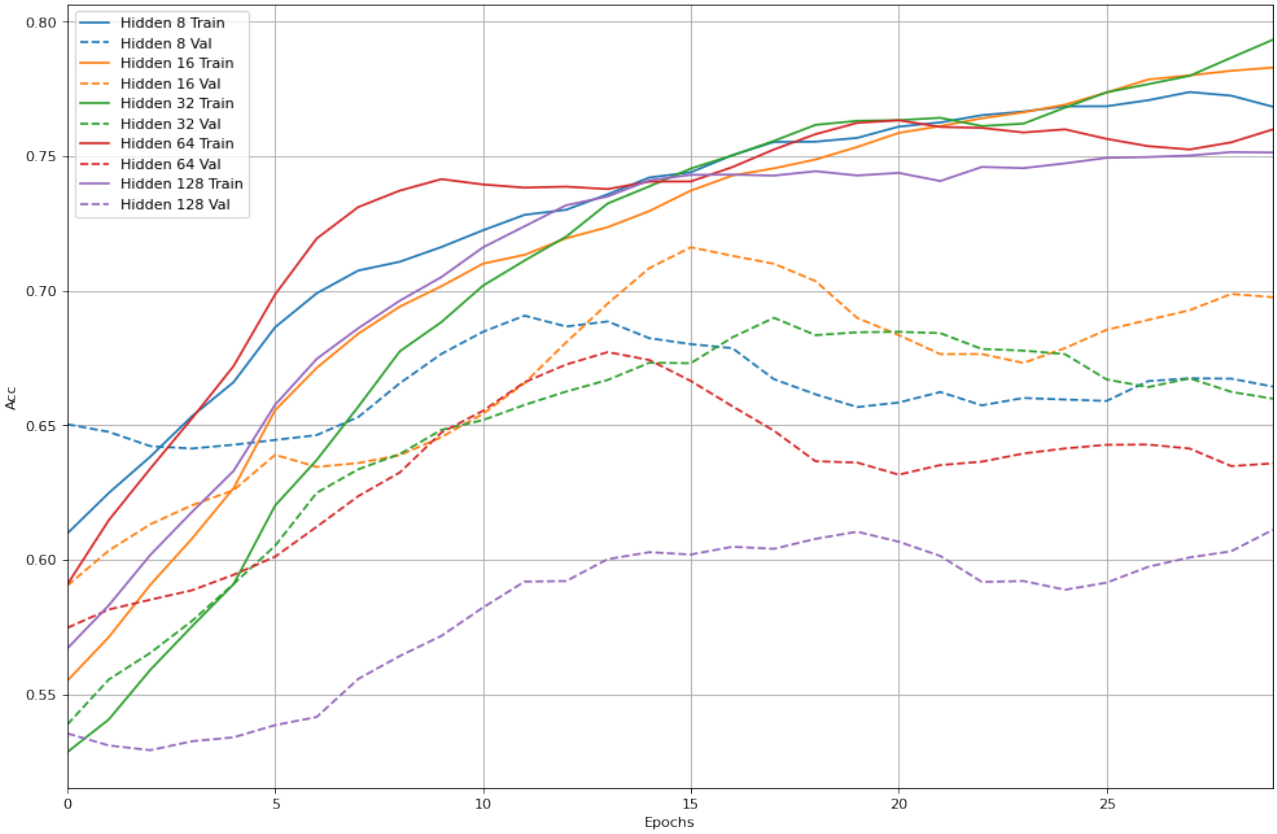

Figure 6: **Visualisasi akurasi model CNN ketika menggunakan N hidden neuron yang berbeda.** Hidden layer yang digunakan merupakan bilangan pangkat dua dari 8 sampai 128, dengan filter dan data augmentation yang sama.

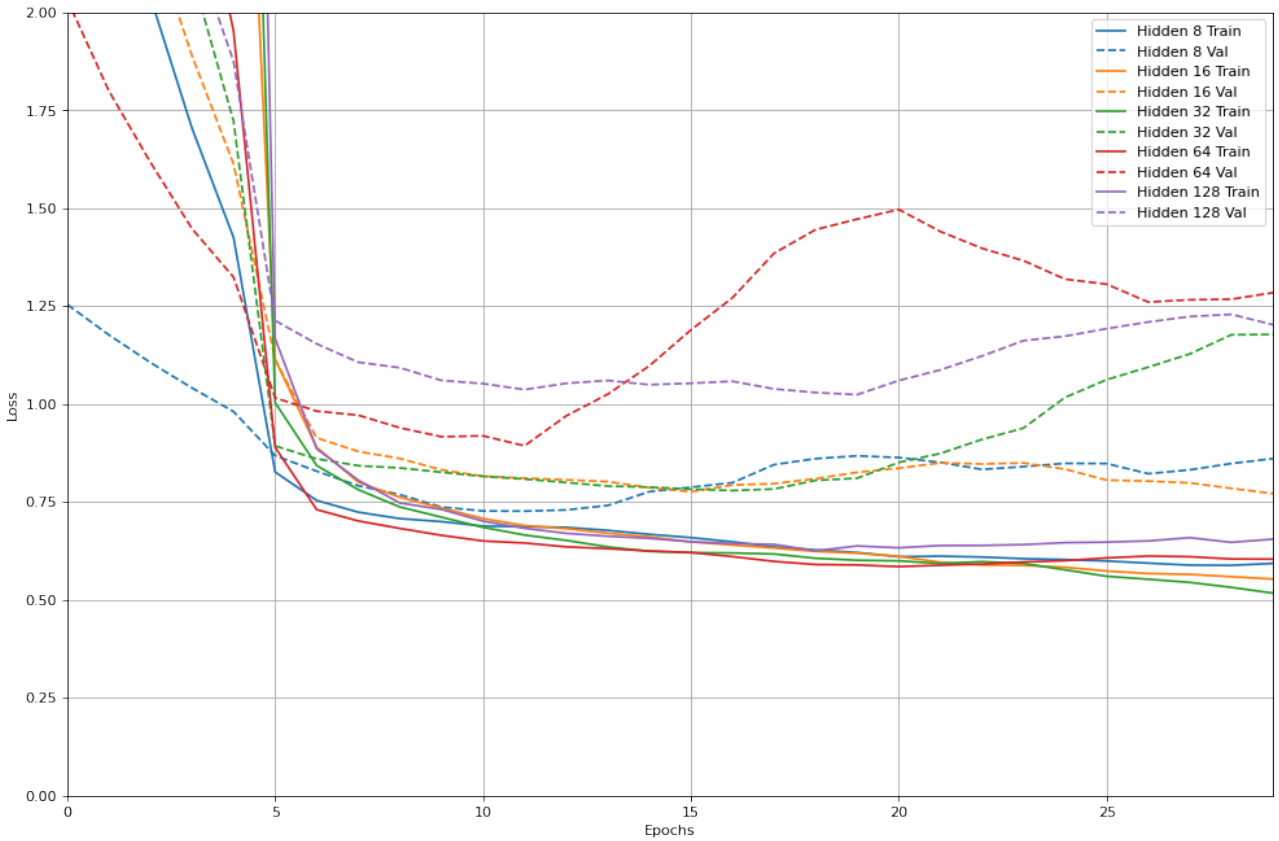

Figure 7: **Visualisasi loss model CNN ketika menggunakan N hidden Neuron yang berbeda.** Hidden layer yang digunakan merupakan bilangan pangkat dua dari 8 sampai 128, dengan filter dan data augmentation yang sama.

3.2.1 Tabel Hasil

Model (n_hidden_neuron)	Train Accuracy	Train Loss	Test Accuracy	Test Loss
Hidden 8	0.7667	0.5741	0.6833	**0.7107**
Hidden 16	0.8042	0.4919	**0.7167**	0.8732
Hidden 32	**0.8250**	**0.4395**	0.6167	1.0881
Hidden 64	0.7458	0.6049	0.6500	1.5772
Hidden 128	0.7583	0.6505	0.7000	1.2991

3.3 Data Augmentation

3.3.1 Plot Akurasi dan Loss

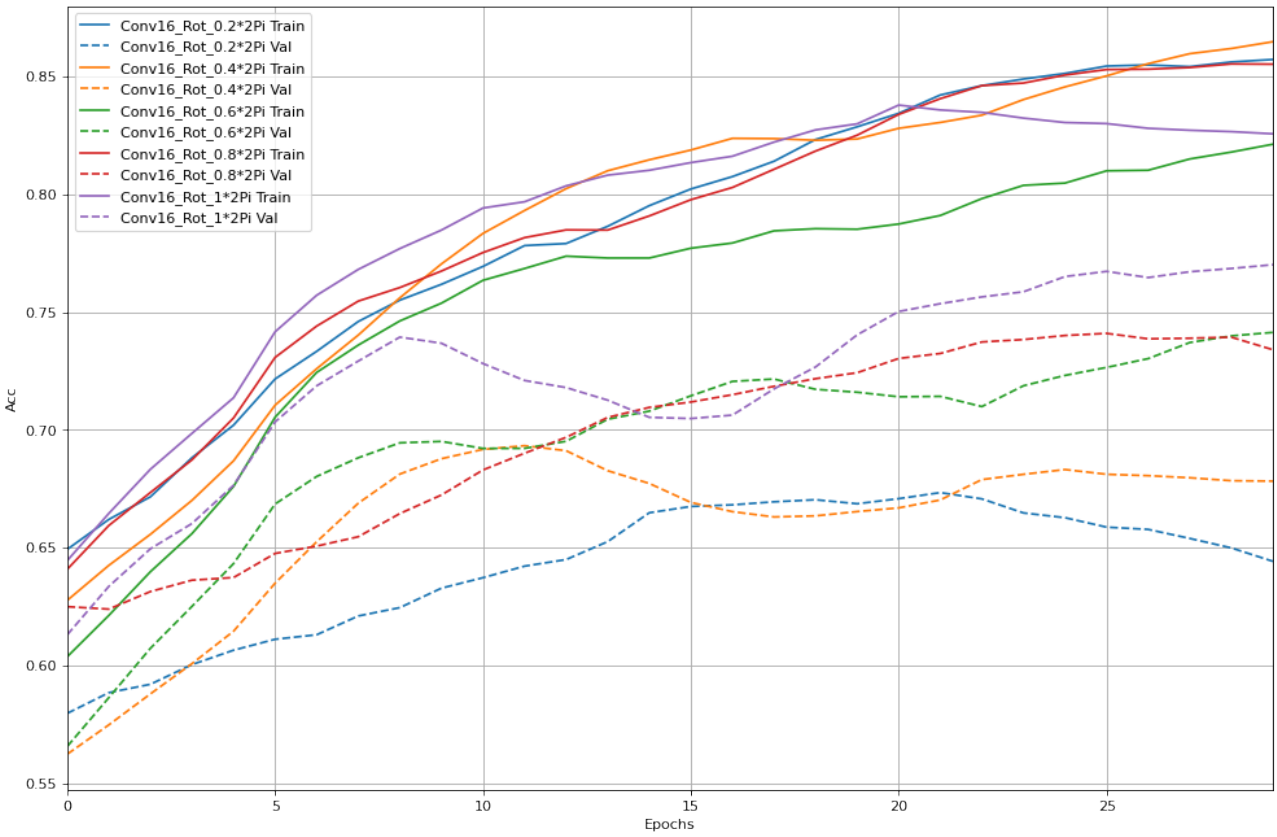

Figure 8: **Visualisasi akurasi model CNN ketika menggunakan Data Augmentation.** Visualisasi plot dengan menggunakan smoothing_std=1. Model augmentasi dengan rotasi range mulai dari $0.2 \times 2\pi$ sampai 2π

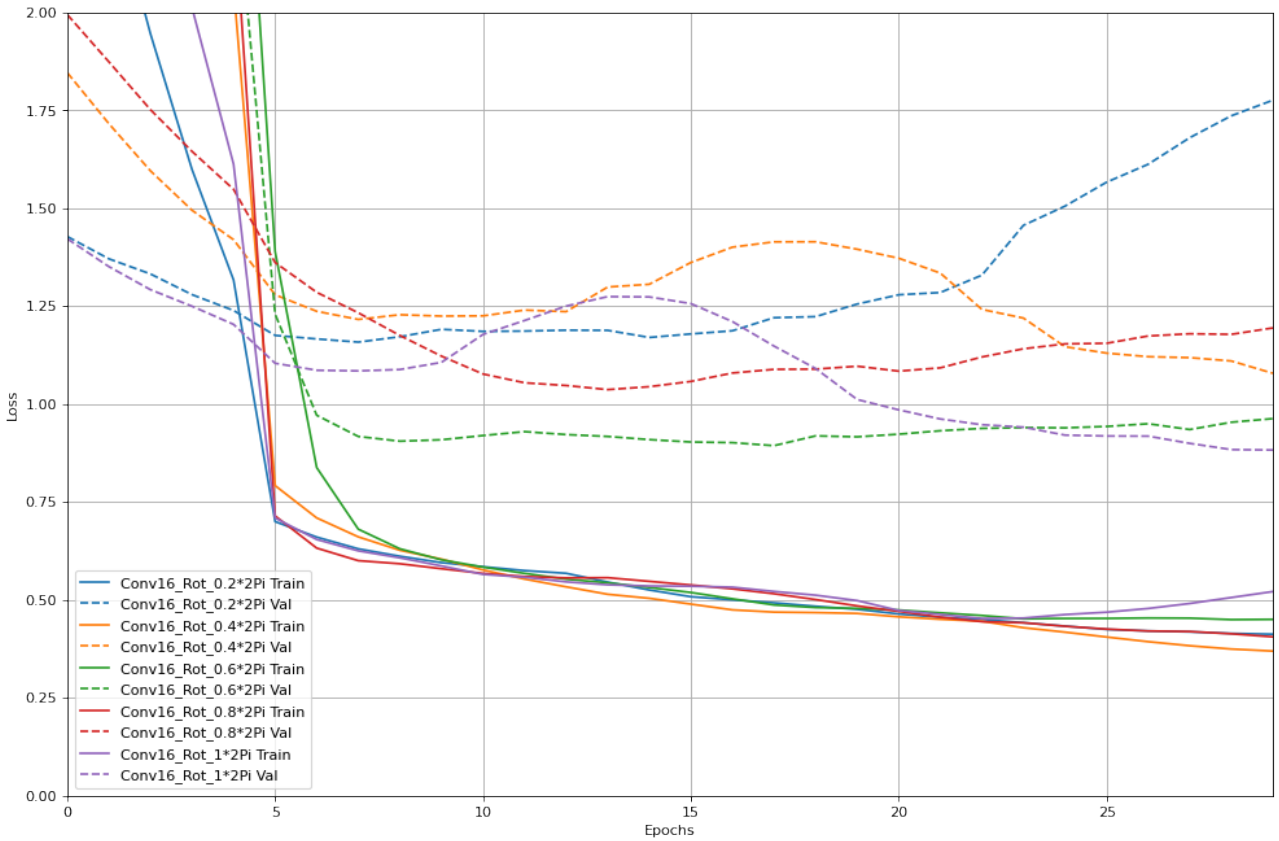

Figure 9: **Visualisasi akurasi dan error model CNN ketika menggunakan Data Augmentation.** Visualisasi plot dengan menggunakan smoothing_std=1. Model augmentasi dengan rotasi range mulai dari $0.2 \times 2\pi$ sampai 2π

3.3.2 Tabel Hasil

Model (Data Augmentation)	Train Accuracy	Train Loss	Test Accuracy	Test Loss
$filters_{16}h_{16}$ with Rot $[0.2 \times 2\pi]$	**0.8875**	0.3611	0.6500	1.5139
$filters_{16}h_{16}$ with Rot $[0.4 \times 2\pi]$	0.8792	**0.3121**	0.6500	1.3683
$filters_{16}h_{16}$ with Rot $[0.6 \times 2\pi]$	0.8292	0.4945	0.7667	1.0620
$filters_{16}h_{16}$ with Rot $[0.8 \times 2\pi]$	0.8375	0.4042	0.7667	1.0522
$filters_{16}h_{16}$ **with Rot** $[\mathbf{2\pi}]$ (epoch 29)	0.8458	0.6158	**0.8167**	**0.8253**

4 Kesimpulan

Pada flower dataset, ketika menggunakan augmentasi data, model memiliki performa terbaik ketika faktor rotasi bernilai 1, atau gambar memiliki kebebasan untuk diputar kesegala arah. Dengan ini pada epoch ke-29 dari 30, model dapat memprediksi dataset train 84.58% dan test 81.67%. Terdapat kenaikan sebab dengan augmentasi data, data dapat ditambah berdasarkan data yang ada. Data dapat dibalik, dirotasi membuat banyak variasi dari satu sumber gambar. Dengan ini CNN dapat mencegah Overfitting.

Berdasarkan tabel hasil pada poin sebelumnya setelah dilakukan percobaan model dengan banyak filter berbeda yaitu 8, 16, 32, 64, 128 dengan augmentasi data ketika faktor rotasi bernilai satu, didapatkan bahwa untuk test akurasi terbaik yang didapat adalah 70% dengan filter pada convolution layer nya sebanyak 64 buah . Namun apabila dilihat lagi loss dari 64 filter yaitu 1.0721, dibandingkan dengan 128 filter yang loss hanya 0.85 dengan akurasi mencapai 67%, maka agaknya bisa dikatakan bahwa CNN dengan filter sebanyak 128 filter memberikan hasil yang lebih baik.

Sedangkan ketika dilakukan eksperimen dengan menggunakan N hidden neuron yang berbeda yaitu 8, 16, 32, 64, 128 dan dengan augmentasi data ketika faktor rotasi bernilai 1, ditemukan bahwa model dengan performa terbaik adalah dengan jumlah hidden neuron sebanyak 16 yaitu menghasilkan test akurasi 71% dan train akurasi 80%.

Dari hasil percobaan pada poin sebelumnya didapatkan bahwa ketika menggunakan arsitektur CNN akurasi

mencapai sekitar 70%-80%. Sementara pada percobaan pekan sebelumnya, arsitektur Multi Layer Perceptron (MLP) menghasilkan akurasi hanya sampai 35% ketika menggunakan softmax padahal menggunakan dataset yang sama dengan yang digunakan oleh CNN.

Jika dilihat dari perbandingan akurasi dari CNN dan MLP, dapat disimpulkan bahwa arsitektur Convolutional Neural Networ (CNN)k lebih baik daripada arsitektur MLP apabila digunakan untuk melakukan pembelajaran mesin dengan data berupa data image.

5 Tugas

Github repository terdapat pada :
https://github.com/Chrysophyt/MultilayerPerceptronFS/tree/main/CNN
Pembagian Tugas :

1. Chrystian: Mengurus GitHub, Eksperimen Data Augmentation, Penulisan Laporan.
2. Karunia Eka Putri: Eksperimen Hidden, Penulisan Laporan.
3. Widad Abida R: Eksperimen Filter, Penulisan Laporan