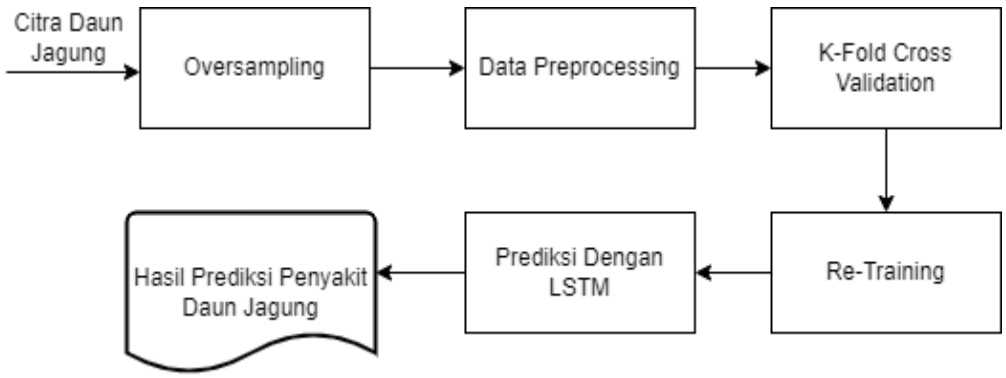
BAB IV
HASIL DAN PEMBAHASAN

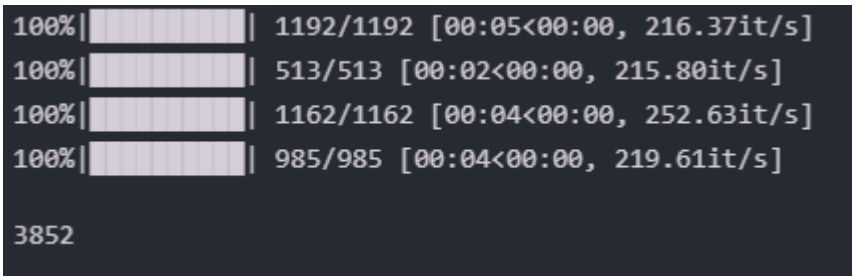
4.1 Hasil Metode *LSTM*

Untuk proses eksperimen deteksi penyakit pada daun jagung dengan metode *LSTM* pertama-tama dilakukan teknik *oversampling* terlebih dahulu, lalu dilakukan proses memecah *dataset* menjadi 3 bagian yaitu *training, testing,* dan *validation.* Langkah selanjutnya adalah tahap pra-pemrosesan data yang dilanjutkan dengan eksperimen untuk mengetahui ukuran citra yang tepat, lalu dilakukan *hyperparameter tuning* untuk mendapatkan struktur *model* yang tepat, setelah itu dilakukan *k-fold cross validation* untuk mengetahui performa *model*, lalu sebelum prediksi dilakukan *re-training* dengan dataset dari bagian *training* dan *validation* agar dapat dihasilkan model yang dapat digunakan untuk prediksi penyakit daun jagung dengan metode LSTM. Untuk gambaran proses eksperimen *LSTM* secara keseluruhan dapat dilihat pada Gambar 4.1.



Gambar 4.1*: Flowchart* Eksperimen *LSTM*

Sebelum masuk ke tahap prediksi dengan *LSTM* diperlukan tahap *oversampling* yaitu pertama-tama dilakukan pembagian ukuran partisi data antara data *training*, data *validation*, data *testing* dengan menggunakan bantuan dari *library* Python yaitu *splitfolders*. Selain itu dikarenakan dataset yang digunakan pada penelitian ini adalah dataset yang tidak seimbang seperti yang bisa dilihat pada Gambar 4.2, maka jumlah data pada setiap kelasnya perlu dilakukan *oversampling* atau *undersampling*, pada penelitian ini digunakan metode *oversampling.* Oleh karena penelitian [20] menyatakan bahwa teknik *oversampling* memiliki hasil yang lebih baik dibandingkan dengan *undersampling* maka pada penelitian ini digunakan teknik *oversampling*.



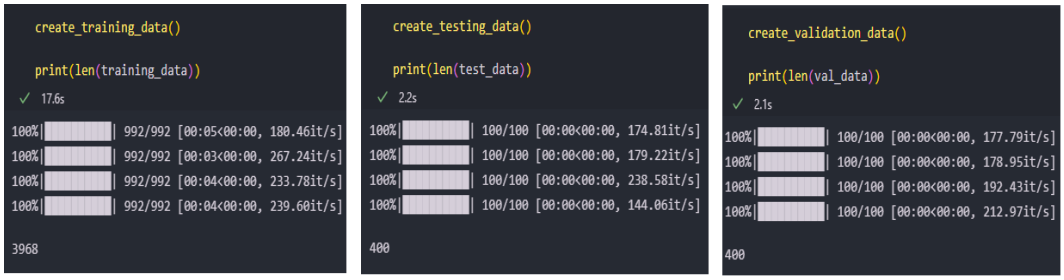
Gambar 4.2*: Imbalanced* Dataset

Dalam *library splitfolders* juga disediakan metode *oversampling* yaitu dengan membuat parameter "*oversampling*" dengan nilai *True*, pada penelitian ini diambil 100 data dari masing-masing kelas sebagai dataset untuk validasi dan 100 data lagi diambil untuk dataset *testing*, lalu sisanya adalah dataset untuk *training.* Inisiasi dari *splitfolders* yang digunakan pada penelitian ini dapat dilihat pada Gambar 4.3.

```
splitfolders.fixed(
    input_folder,output_folder,
    seed = 1337,
    fixed = (100,100),
    oversample=True,
    group_prefix = None
)
```

Gambar 4.3*:* Inisalisasi *splitfolders*

Setelah itu dilakukan proses *splitfolders* dengan *oversampling* dan didapatkan jumlah data untuk dataset *training* pada masing-masing kelas sebanyak 992 data dengan total data keseluruhan adalah 3968 dan total data untuk dataset validasi sebanyak 400 data, dan data *testing* sebanyak 400 data. Hasil dari *splitfolders* dan *oversampling* dapat dilihat pada Gambar 4.4 dimana dapat dilihat jumlah citra pada masing-masing kelas telah memiliki jumlah yang sama dan tidak ada lagi kelas yang memiliki citra yang jumlahnya lebih banyak dari kelas yang lain, pada Gambar 4.4 yang sebelah kiri adalah hasil *splitting* dan *oversampling* untuk data *training* dengan jumlah 3968 citra, yang tengah adalah hasil *splitting* dan *oversampling* untuk data *testing* dengan jumlah 400 citra, dan yang kanan adalah hasil *splitting* dan *oversampling* untuk data validasi dengan jumlah 400 citra.



Gambar 4.4*:* Hasil *splitfolders* dan *oversampling*

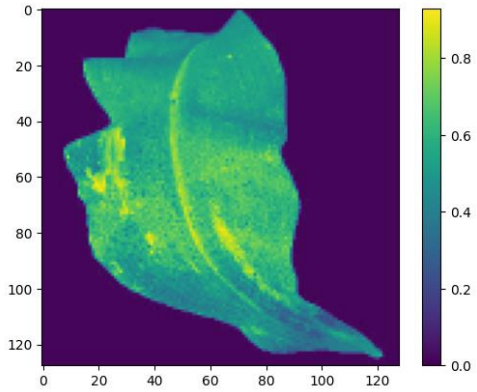
Setelah direktori masing-masing kelas sudah seimbang maka sudah bisa dimulai proses *pre-processing* citra daun jagung, yaitu mengkonversi citra daun jagung dari *RGB* ke *grayscale* dengan bantuan *library OpenCV* yaitu *function imread*(), kemudian untuk ukuran gambar diperkecil dengan bantuan *function resize*() pada *OpenCV* hal ini dilakukan agar proses dari *deep learning* bertambah cepat, namun hal ini diperlukan eksperimen kecil yaitu apakah ukuran gambar berpengaruh kepada akurasi model *deep learning* atau tidak bisa dilihat pada Tabel 4.1 adalah tabel hasil dari eksperimen menentukan ukuran citra daun jagung yang sesuai dengan menggunakan model *deep learning* sederhana 1 *layer LSTM* dengan jumlah *node* sesuai dengan ukuran dari gambar citra dan dilakukan *training* dan *validation* selama 20 *epochs* dengan 32 *batch size*. *Epochs* adalah banyaknya iterasi yang dilakukan model *deep learning*, dan *epochs* adalah salah satu faktor yang mempengaruhi waktu pemrosesan *deep learning* oleh karena itu pada eksperimen ini digunakan jumlah *epochs* yang sama agar tidak terjadi kalkulasi evaluasi yang tidak valid. Hasil yang didapat bahwa akurasi untuk *validation* tidak berbeda jauh tetapi waktu yang dibutuhkan untuk *training* dan *validation* sangat banyak berkurang. Penelitian [24] juga telah melakukan eksperimen untuk meneliti ukuran citra yang tepat untuk akurasi model *deep learning* dan dari hasil yang didapat akurasi untuk ukuran yang lebih tinggi belum tentu memiliki akurasi yang lebih tinggi dibandingkan dengan ukuran citra yang lebih kecil dan juga terdapat faktor waktu eksekusi program yang sudah pasti bertambah jika

ukuran citra semakin besar. Oleh karena itu dari eksperimen *resizing data* ini dapat diambil kesimpulan ukuran gambar 128 X 128 adalah yang paling tepat untuk penelitian ini.

Tabel 4.1: Hasil Eksperimen *Resizing Data*

No	Ukuran Gambar	Hasil				
		Training		*Validation*		Waktu Eksekusi
		Accuracy	*Loss*	*Accuracy*	*Loss*	
1	256 X 256	0.7664	0.5023	0.7175	0.5419	75 menit 22 detik
2	200 X 200	0.7500	0.5328	0.7225	0.5151	43 menit 37 detik
3	128 X 128	0.7354	0.5834	0.7100	0.5460	6 menit 59 detik

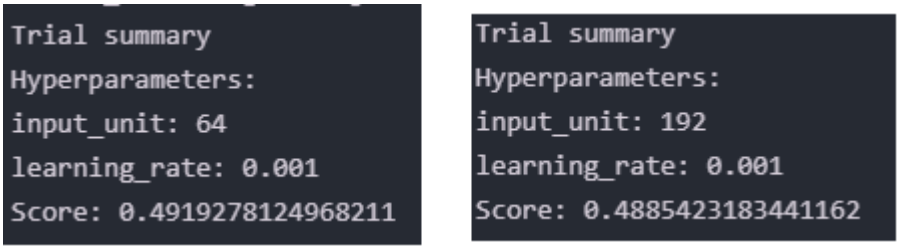
Lalu gambar dikonversi menjadi *numpy array* dengan bantuan *library numpy* yaitu fungsi *array ()* dan langkah terakhir adalah normalisasi warna pada gambar dari rentang 0 sampai 255 menjadi dari rentang 0 sampai 1, hasil pra-pemrosesan data dapat dilihat pada Gambar 4.5 sebagai salah satu contoh citra daun yang telah di proses sebelumnya untuk masuk ke dalam *deep learning model*.



Gambar 4.5*:* Hasil *Data Pre-processing*

Selanjutnya adalah proses eksperimen untuk menentukan struktur model *deep learning*, pada tahap ini dilakukan perbandingan antara 2 model yaitu 2 buah *layer LSTM* dan 1 *layer LSTM* lalu sekaligus dilakukan juga *hyperparameter tuning* dengan menggunakan bantuan dari *library keras tuner*. *Keras tuner* yang digunakan adalah dengan menggunakan algoritme *random search* untuk mencari parameter terbaik bagi model *deep learning LSTM*. Parameter yang akan dilakukan tuning adalah jumlah *neuron* pada *layer LSTM* dan berapa nilai dari *learning rate* yang sesuai untuk model, proses *hyperparameter tuning* ini dibagi menjadi 2 yaitu dari nilai node 32 sampai 128 dan dari 128 sampai 256, lalu dari learning rate diberikan pilihan untuk *tuner* adalah nilai 0.01, 0.001, dan 0.0001. Proses *tuning* dilakukan dengan tujuan untuk mengurangi variabel *validation loss* ditambah juga dengan *early stopping* yaitu untuk memberhentikan proses *tuning* pada iterasi tertentu saat *validation loss* tidak berkurang selama 10 *epoch*, setiap iterasi dilakukan sebanyak 100 *epochs* dan 100 *epochs* ini diulangi selama 3 kali untuk setiap iterasi *tuning*, dimana iterasi *tuning* akan dilakukan sebanyak 5 kali. Hasil dari eksperimen dengan

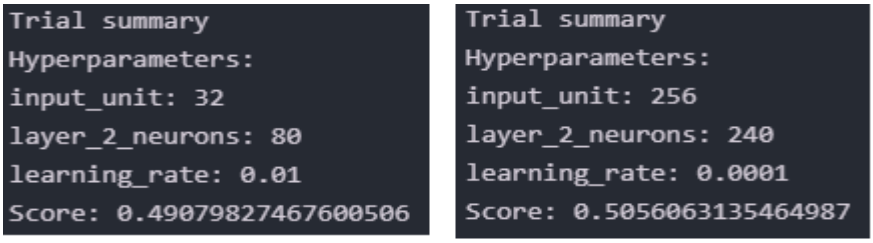
menggunakan 1 *layer LSTM* dapat dilihat pada Gambar 4.6 dimana yang sebelah kiri adalah hasil dari eksperimen 1 layer yang pertama yaitu model terbaik dengan pencarian jumlah *node* diantara nilai *node* 32 sampai 128 dan didapat hasil nilai skor *validation loss* terendah yaitu 0.4919278124968211 didapat untuk parameter *node LSTM* sejumlah 64 dan nilai *learning rate* 0.001. Pada gambar yang kanan adalah model terbaik hasil dari eksperimen 1 layer yang kedua yaitu dengan pencarian jumlah *node* diantara nilai *node* 128 sampai 256 dan didapat hasil nilai skor *validation loss* terendah yaitu 0.4885423183441162 didapat untuk parameter *node LSTM* sejumlah 192 dan nilai *learning rate* 0.001.



Gambar 4.6*:* Hasil Eksperimen 1 *Layer LSTM*

Selanjutnya dilanjutkan dengan eksperimen untuk menentukan nilai *validation loss* terbaik untuk 2 *layer LSTM,* proses eksperimen dilakukan sama dengan eksperimen 1 *layer LSTM* yaitu proses eksperimen terbagi menjadi 2 bagian yaitu antara jumlah *node* 32 sampai 128 dan 128 sampai 256, perbedaanya terletak pada jumlah *layer LSTM* yang digunakan, untuk eksperimen ini digunakan 2 *layer LSTM*. Hasil dari eksperimen 2 *layer LSTM* dapat dilihat pada Gambar 4.7 dimana yang sebelah kiri adalah hasil dari eksperimen 2 layer

yang pertama yaitu model terbaik dengan pencarian jumlah *node* diantara nilai *node* 32 sampai 128 dan didapat hasil nilai *validation loss* terendah yaitu 0.49079827467600506 didapat untuk parameter *node LSTM* yang pertama sejumlah 32, *layer LSTM* yang kedua 80 dan nilai *learning rate* 0.01. Pada gambar yang kanan adalah model terbaik hasil dari eksperimen 2 layer yang kedua yaitu dengan pencarian jumlah *node* diantara nilai *node* 128 sampai 256 dan didapat hasil nilai *validation loss* terendah yaitu 0.5056063135464987 didapat untuk parameter *node LSTM* yang pertama sejumlah 256, *layer LSTM* yang kedua 240 dan nilai *learning rate* 0.0001. Untuk rangkuman hasil eksperimen *hyperparameter tuning* dapat dilihat pada Tabel 4.2.



Gambar 4.7*:* Hasil Eksperimen 2 *Layer LSTM*

Tabel 4.2: Hasil Eksperimen *Hyperparameter Tuning*

No	Jumlah *Layer*	Rentang *nodes*	Jumlah Node		Hasil	
			Node layer 1	*Node layer 2*	*Learning rate*	*Validation loss*
1	1	32-128	64	-	0.001	0.491927
2		128-256	192	-	0.001	0.488542
3	2	32-128	32	80	0.01	0.490798
4		128-256	256	240	0.0001	0.505606

Berdasarkan hasil *hyperparameter tuning* pada Tabel 4.2 didapat struktur model terbaik yaitu banyaknya *layer LSTM* beserta jumlah *nodes* yang sesuai yaitu 1 *layer LSTM* dengan 192 nodes, setelah menentukan struktur untuk *hidden layer* pada *deep learning* karena nilai *validation loss* masih rendah maka dilakukan eksperimen untuk mengetahui apakah *image augmentation* yaitu proses augmentasi citra dapat membuat akurasi dan *loss* dari model *deep learning* yang digunakan pada penelitian ini menjadi semakin baik, eksperimen augmentasi citra ini dilakukan dengan menambahkan 1 lapisan lagi pada model dengan menggunakan *library keras* yaitu *RandomContrast* dengan nilai parameter 0.2, dan didapat hasil seperti pada Gambar 4.8 yang menyatakan bahwa dari proses augmentasi tidak ada peningkatan signifikan pada proses augmentasi maka augmentasi dapat dihilangkan pada lapisan model.

```
Epoch 11/100
124/124 [==============================] - 41s 333ms/step - loss: 0.6159 - accuracy:
0.7064 - val_loss: 0.6399 - val_accuracy: 0.7075
Epoch 12/100
124/124 [==============================] - 41s 335ms/step - loss: 0.6046 - accuracy:
0.7082 - val_loss: 0.5616 - val_accuracy: 0.7100
Epoch 13/100
...
Epoch 44/100
124/124 [==============================] - 56s 450ms/step - loss: 0.4229 - accuracy:
0.8032 - val_loss: 0.5096 - val_accuracy: 0.7750
Epoch 45/100
124/124 [==============================] - 56s 456ms/step - loss: 0.4267 - accuracy:
0.8070 - val_loss: 0.5094 - val_accuracy: 0.7450
```

Gambar 4.8*:* Hasil Eksperimen Augmentasi Data

Setelah proses sub-eksperimen untuk mencari ukuran citra yang tepat dan *hyperparameter tuning* untuk mencari struktur model yang tepat, maka didapatkan model yang digunakan untuk deteksi penyakit pada daun jagung.

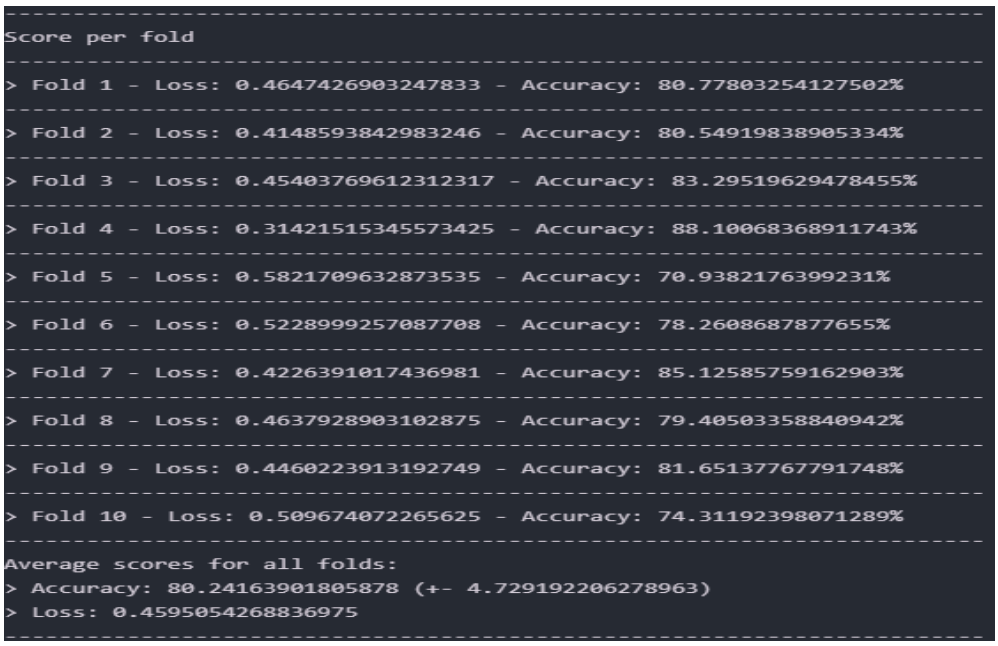
Struktur *final* dari model dapat dilihat pada Gambar 4.9, pada gambar tersebut *layer* pertama adalah *input layer* menyesuaikan dengan ukuran gambar yaitu 128 X 128, lalu *layer* kedua adalah *hidden layer* yaitu *LSTM* dengan jumlah *nodes* sebanyak 192, lalu *layer* terakhir adalah *output layer* yaitu *dense layer* dengan jumlah *nodes* 4 yaitu sesuai dengan jumlah kelas yang akan di prediksi dengan menggunakan fungsi aktivasi *softmax*, model di-*compile* dengan *sparse categorical crossentropy* untuk klasifikasi banyak kelas sebagai parameter *loss*, lalu *optimizer* yang digunakan adalah *Adam* dengan *learning rate* 0.001, dan metrik validasi model yang digunakan adalah *accuracy*.

```
earlyStop = tf.keras.callbacks.EarlyStopping(monitor='val_loss',patience=10)
model = keras.Sequential()
model.add(keras.Input(shape=(IMG_SIZE,IMG_SIZE)))
model.add(keras.layers.LSTM(192))
model.add(keras.layers.Dense(4, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=1e-3),
              metrics = ['accuracy'])
model.summary()
```

Gambar 4.9*:* Struktur *Final* Model

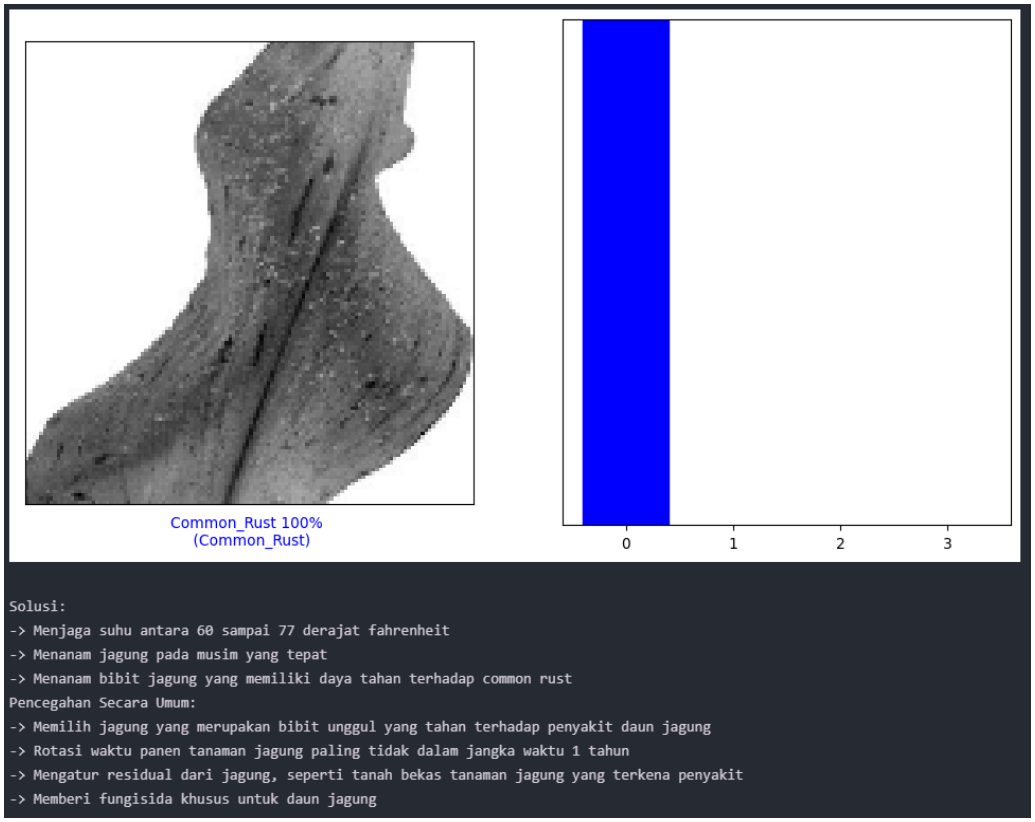
Setelah struktur model *deep learning* telah ditentukan maka dapat dilakukan eksperimen untuk optimisasi selanjutnya yaitu menggunakan teknik *k-fold cross validation* dengan bantuan *library sklearn.* Proses eksperimen *cross validation* ini menggunakan 10-*fold* dan data yang akan dilakukan *cross validation* adalah data dari *training* dan data dari *validation*. Hasil dari eksperimen *cross validation* ini dapat dilihat pada Gambar 4.10 yang merupakan 10 hasil dari setiap *fold* dari *cross validation*, dari keseluruhan *fold*

dihitung rata-rata *accuracy* dan *loss*, rata-rata tersebut adalah hasil evaluasi metode untuk metode *LSTM*.



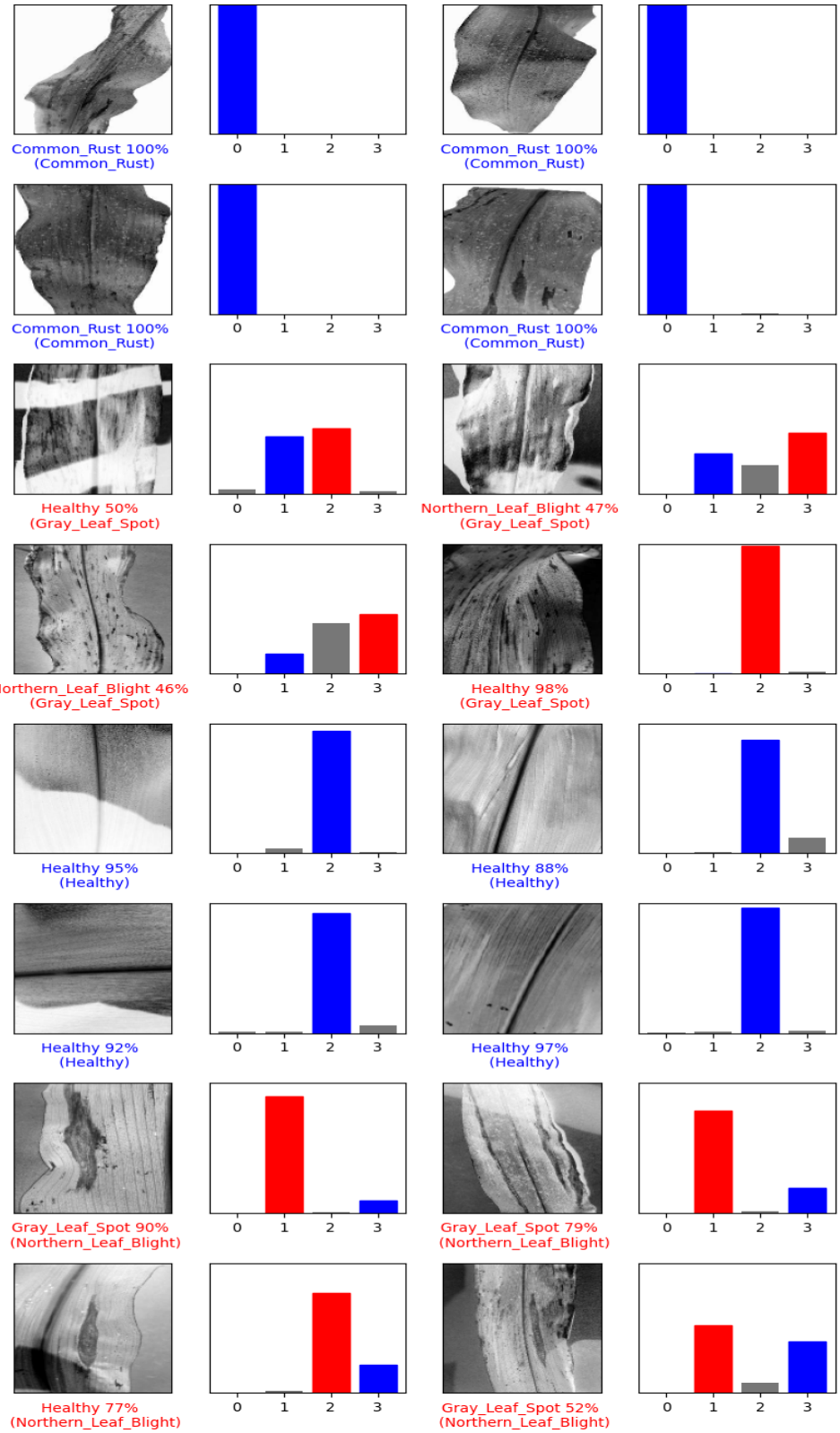
Gambar 4.10*:* Hasil K-fold Cross Validation

Selanjutnya dilakukan proses *training* dengan data *training* dengan data *validation* untuk menghasilkan model yang digunakan untuk memprediksi dari data *testing*. Prediksi untuk mendeteksi penyakit pada daun jagung yang dilakukan untuk 1 data dan seluruh data yang ada di dalam *testing dataset*, hasil dari prediksi dapat dilihat pada Gambar 4.11 yang merupakan hasil prediksi daun jagung yang terkena penyakit *common rust,* lalu apabila jagung tersebut benar terkena penyakit menurut model maka ditampilkan solusi dan cara untuk mencegah penyakit daun tersebut di bawah hasil prediksi. Untuk prediksi seluruh *testing data* dilakukan dalam waktu 1,59 detik , lalu hasilnya dapat dilihat pada lampiran.



Gambar 4.11*:* Hasil Prediksi 1 Data

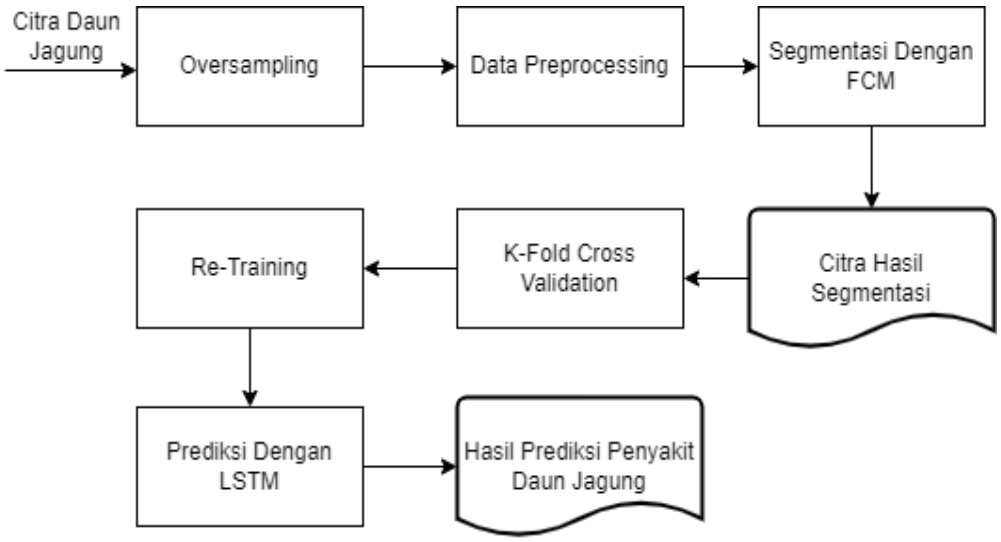
Selanjutnya dilakukan pengambilan sampel data dari dataset *testing* untuk dianalisis hasil prediksi dari algoritme *LSTM* yang dapat dilihat pada Gambar 4.12, dimana pada gambar dapat dilihat algoritme *LSTM* dapat memprediksi citra dari kelas *common rust* dan *healthy* dengan baik, namun untuk kelas *gray leaf spot* dan *northern leaf blight* model ini masih kurang akurat dalam memprediksinya.



Gambar 4.12: Hasil Prediksi Sampel Dataset *Testing*

4.2 Hasil Metode *FCM + LSTM*

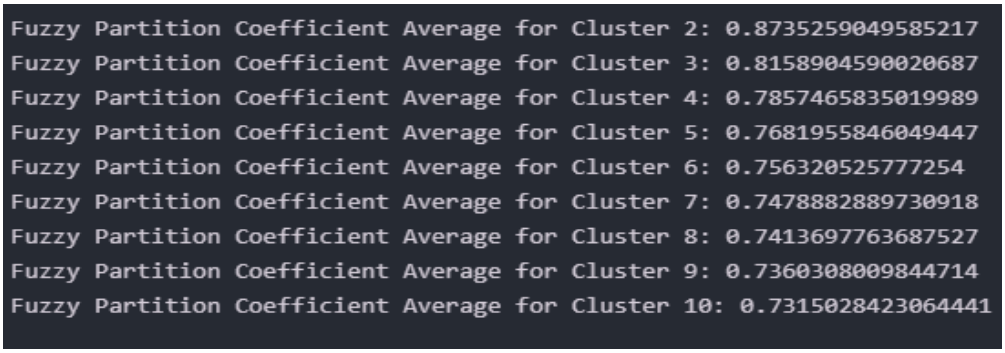
Untuk metode *Fuzzy C-Means* pertama-tama dilakukan *resize* citra menjadi 128 X 128 dan citra dikonversi menjadi citra *grayscale*. Setelah itu dilakukan normalisasi data agar rentang warna dari 0 sampai 255 menjadi 0 sampai 1. Untuk implementasi algoritme *Fuzzy C-Means* dilakukan dengan menggunakan *library* dari *python* yaitu *scikit-fuzzy* lalu untuk parameter yang digunakan untuk *Fuzzy C-Means* diambil dari referensi [17]. Pada Gambar 4.13 dapat dilihat *flowchart* yang menggambarkan proses deteksi penyakit daun jagung dengan menggunakan algoritme *Fuzzy C-Means*.



Gambar 4.13*: Flowchart* Eksperimen *FCM*

Sebelum memasuki proses *clustering* utama dilakukan eksperimen pada seluruh citra. Citra dilakukan sub-eksperimen *clustering* dengan jumlah *cluster* 2,3,4,5,6,7,8,9, dan 10 untuk melihat *cluster* yang memiliki hasil segmentasi yang paling baik berdasarkan nilai *fuzzy partition coefficient* dan citra hasil

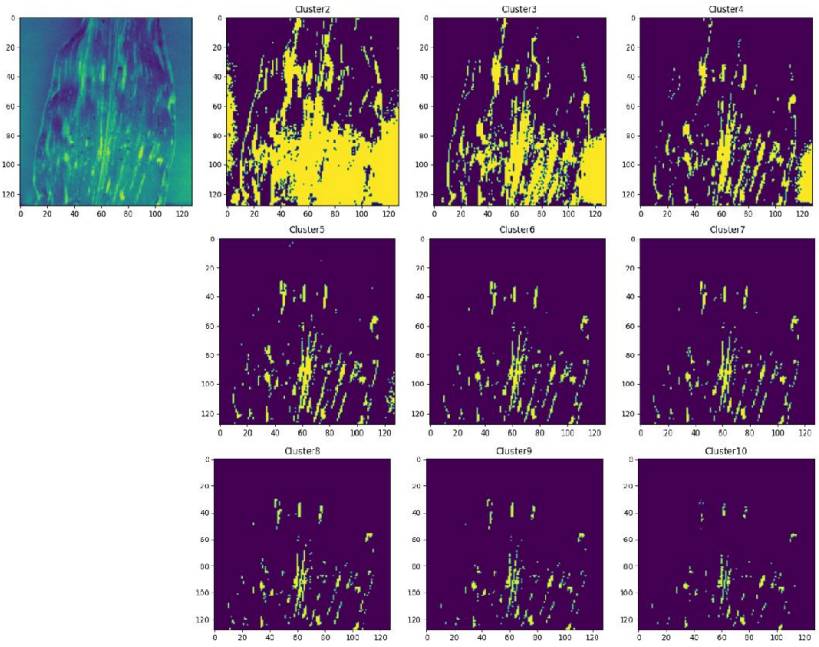
segmentasinya. Hasil dari eksperimen *fuzzy partition coefficient* dapat dilihat pada Gambar 4.14, pada gambar dapat dilihat rata-rata dari *fuzzy partition coefficient*, setiap citra memiliki 1 *fuzzy partition coefficient* jadi untuk mengetahui *cluster* yang terbaik berdasarkan *fuzzy partition coefficient* harus dihitung nilai rata-rata dari perhitungan *fuzzy partition coefficient* masing-masing citra*, clustering* dilakukan untuk seluruh citra pada dataset.



Gambar 4.14*:* Hasil Eksperimen *Fuzzy Partition Coefficient*

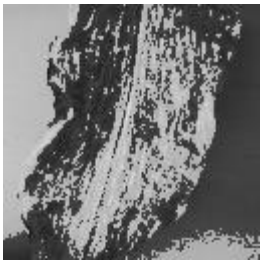
Setelah diketahui jumlah *fuzzy partition coefficient* untuk *cluster* 2 sampai 10, maka selanjutnya dilakukan eksperimen kecil lagi untuk melihat hasil segmentasi citra oleh *Fuzzy C-Means* secara visual. Eksperimen kecil ini dilakukan dengan menggunakan 3 data sampel dari masing-masing kelas citra daun jagung. Hasil segmentasi citra untuk *cluster* 2 sampai 10 dengan data sampel dapat dilihat pada Lampiran.

Hasil eksperimen segmentasi citra dengan data sampel dapat dilihat bahwa hasil segmentasi untuk *cluster* 2 sampai 5 kurang baik karena jika dilihat dari Gambar 4.15 terdapat bagian cahaya yang masuk ke hasil segmentasi, lalu mulai dari *cluster* 6 sampai 10 hasil segmentasi lebih baik.



Gambar 4.15*:* Hasil Eksperimen Segmentasi *FCM*

Setelah itu baru dilakukan eksperimen utama yaitu menggabungkan hasil segmentasi citra dari *FCM* dengan gambar *grayscale* yang aslinya untuk menonjolkan bagian dari citra yang dilakukan segmentasi. Setelah itu baru citra tersebut dimasukkan ke dalam algoritme *LSTM* untuk dilakukan klasifikasi, untuk contoh gambar hasil segmentasi *FCM* yang telah digabungkan dengan citra *grayscale* aslinya dapat dilihat pada Gambar 4.16.



Gambar 4.16*:* Hasil Segmentasi Citra *FCM*

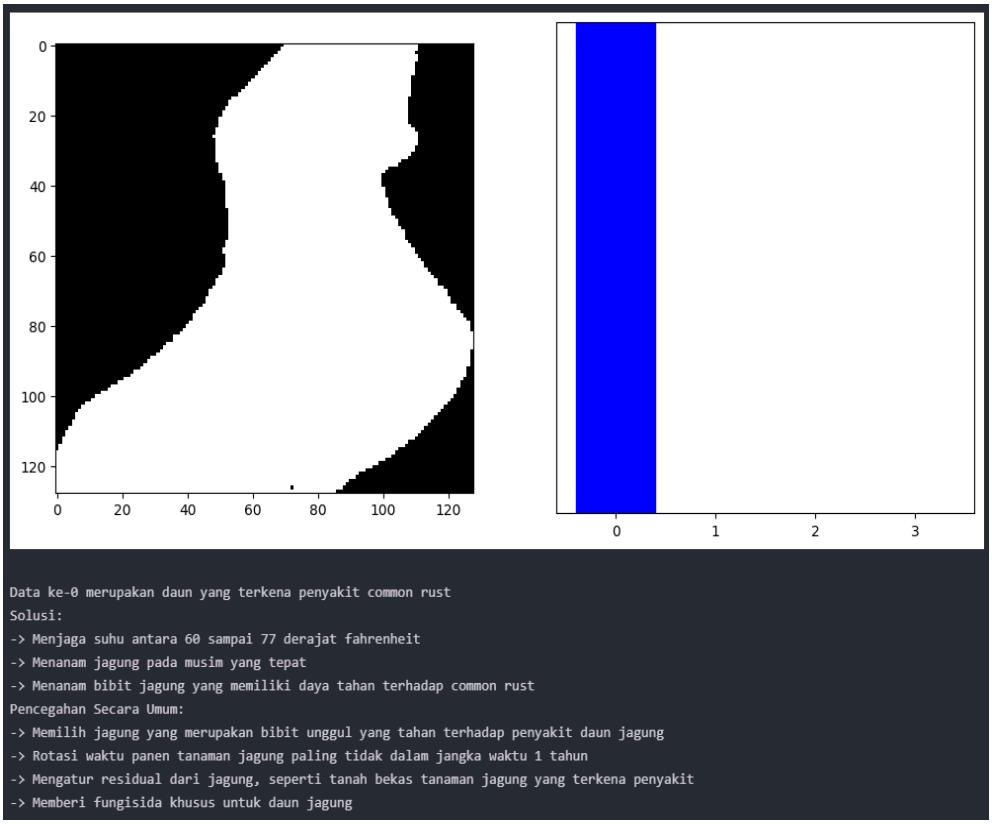
Seperti eksperimen pada *LSTM* data validasi dan data *training* digabung dan dilakukan *k-fold cross validation* dengan nilai $k = 10$, dan struktur model dan parameter yang digunakan sama seperti eksperimen pada *LSTM*. 10-*fold cross validation* dilakukan untuk *cluster* 1 sampai 10 dan dilihat rata-rata dari proses validasi seluruh *cluster* yang dapat dilihat pada Tabel 4.3. Untuk hasil dari validasi masing-masing *cluster* dapat dilihat pada lampiran.

Tabel 4.3: Hasil Evaluasi 10-*Fold Cross Validation* 10 *Cluster FCM*

Cluster	Rata-Rata *Accuracy*	Rata-Rata *Loss*
1	70,49%	0,62
2	86,06%	0,46
3	85,07%	0,48
4	83,06%	0,48
5	82,14%	0,48
6	77,20%	0,53
7	78,94%	0,51
8	76,58%	0,54
9	76,10%	0,54
10	76,99%	0,54
Rata-Rata	79,26%	0,52

Setelah itu dilakukan *training* ulang agar dapat dihasilkan model yang valid untuk dilakukan prediksi, proses *training* dilakukan dengan menggunakan

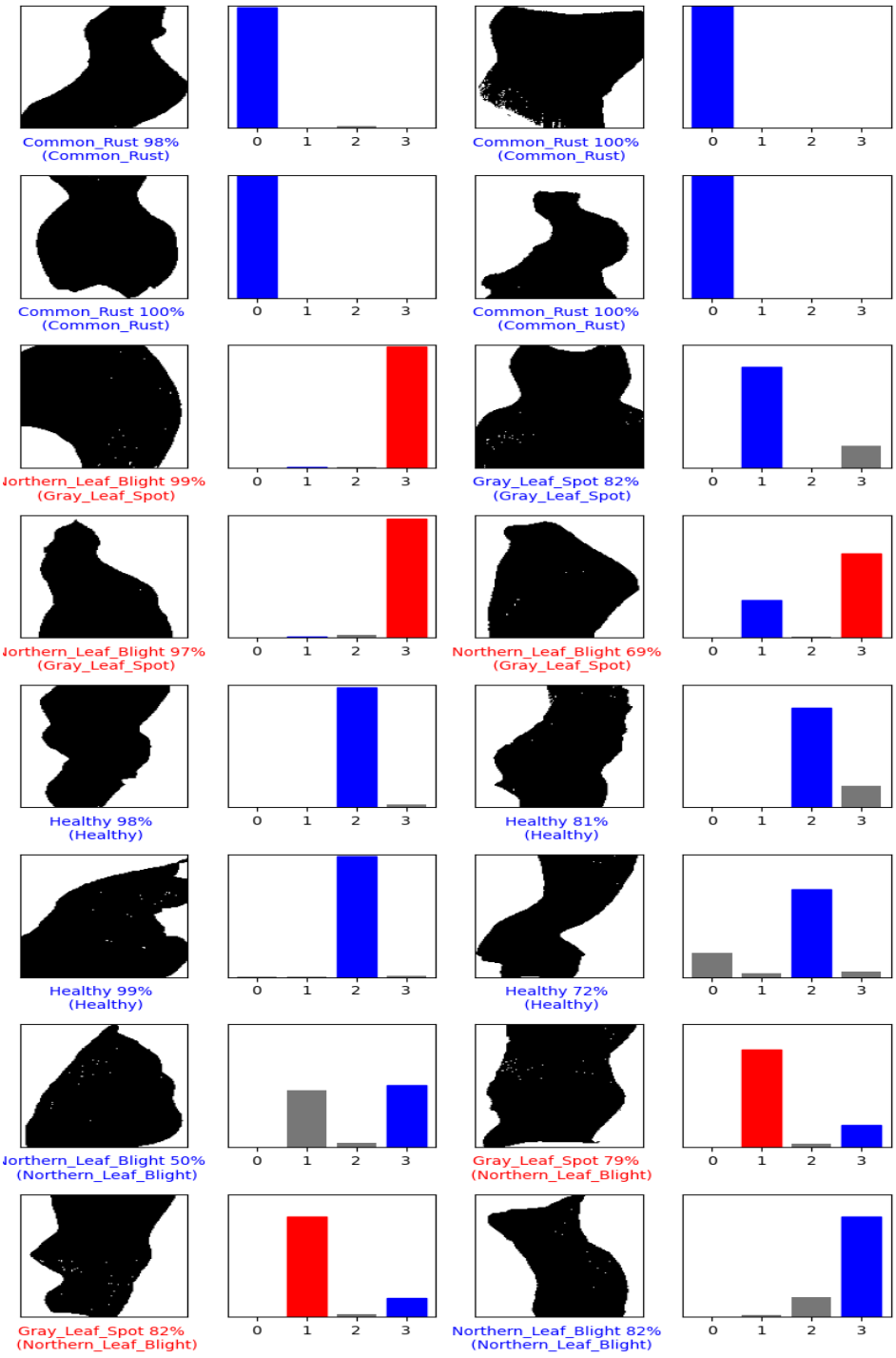
cluster 2 karena merupakan *cluster* yang paling optimal yaitu *cluster* yang memiliki akurasi yang paling tinggi. Model yang dihasilkan dari hasil *training* disimpan dalam format .h5 yang kemudian digunakan untuk prediksi data dari *test* dataset. Hasil dari prediksi pada *test* dataset dapat dilihat pada Gambar 4.17 dan hasil prediksi dari seluruh data pada *testing* dataset dapat dilihat pada lampiran.



Gambar 4.17*:* Hasil Prediksi 1 Citra Pada Dataset Testing

Selanjutnya dilakukan pengambilan sampel data dari dataset *testing* untuk dianalisis hasil prediksi dari algoritme *FCM+LSTM* yang dapat dilihat pada Gambar 4.18, dimana pada gambar dapat dilihat algoritme *FCM+LSTM* dapat

memprediksi citra hasil segmentasi dengan jumlah *cluster* 2 karena memiliki nilai akurasi tertinggi dari *cluster* lainnya.



Gambar 4.18*:* Hasil Prediksi Sampel Dataset Testing

4.3 Analisis Hasil dan Perbandingan Algoritme

Tabel perbandingan kedua algoritme yang digunakan dalam penelitian ini dapat dilihat pada Tabel 4.4. Untuk hasil akurasi algoritme *LSTM* dan *FCM+LSTM* dipengaruhi oleh beberapa faktor yaitu pertama adalah kualitas dan kualitas data , untuk penelitian ini dataset yang digunakan secara kuantitas masih kurang banyak jika dibandingkan dengan dataset yang sering digunakan untuk klasifikasi citra seperti dataset *MNIST handwritten digits* yang memiliki jumlah data 60000 maka jumlah data kurang lebih 3852 data yang digunakan masih kurang banyak, dan juga untuk hal kualitas dalam dataset yang digunakan pada penelitian ini untuk citra daun jagung kurang berkualitas dikarenakan masih ada seperti cahaya sinar matahari, bayangan, dan *midrib* yaitu bagian tengah pada daun jagung.

Faktor kedua yaitu arsitektur model yang digunakan, untuk *LSTM* walaupun sudah melalui *hyperparameter tuning* namun masih memungkinkan untuk dilakukan *tuning* dengan jumlah *node* untuk *LSTM* sampai 512 dan dapat juga dilakukan penambahan *layer* pada *LSTM*. Untuk algoritme *FCM* karena parameter yang didapatkan dari penelitian [17] memiliki akurasi rendah untuk dataset ini maka dapat dilakukan *hyperparameter tuning* untuk parameter algoritme *FCM* agar dapat ditemukan parameter yang lebih tepat untuk dataset yang digunakan pada penelitian ini.

Faktor ketiga yaitu waktu *training* (*epochs*) pada penelitian ini digunakan *epochs* dengan nilai 50 yang merupakan nilai yang didapat dari hasil sub-

eksperimen dengan menggunakan *early stopping* yaitu memberhentikan proses *training* jika akurasi dari validasi tidak meningkat lagi dan akurasi *training* terus meningkat, setelah *epochs* ke-50 mulai terjadi *overfitting.*

Faktor keempat yaitu *data preprocessing,* pada penelitian ini telah dilakukan proses *data preprocessing* seperti *resizing* yaitu memperkecil ukuran citra, *normalization* yaitu menstandarkan nilai piksel dari 0 sampai 255 menjadi 0 sampai 1, *grayscaling* mengkonversi citra menjadi citra *grayscale,* dan *data augmentation* yaitu telah dilakukan sub-eksperimen untuk augmentasi *random zoom* namun tidak ada peningkatan pada hasil akurasi. Masih ada banyak cara *data preprocessing* yang dapat dicoba seperti *image cropping* yaitu mengambil bagian dari citra yang merupakan objek yang akan diklasifikasi dan cara *data cleaning* yaitu dengan menghapus data yang tidak valid seperti citra yang terdapat bayangan.

Tabel 4.4: Perbandingan Algoritme

No.	Algoritme	*K-Fold Cross Validation*		Waktu *Re-Training*	Waktu Prediksi
		Rata-Rata *Accuracy*	Rata-Rata *Loss*		
1	LSTM	80,24%	0,46	13 menit 18 detik	1,59 detik
2	FCM+LSTM	86,06%	0,46	14 menit 28 detik	50,64 detik

Dapat dilihat dari Tabel 4.4 bahwa akurasi *FCM+LSTM* lebih tinggi yaitu 86,06% walaupun memakan waktu sekitar 1 menit lebih lama untuk *training*

dan 49 detik lebih lama dalam memprediksi, *FCM* memiliki akurasi lebih tinggi dikarenakan citra daun jagung telah disegmentasi dan lebih terfokus pada objek daun jagungnya. Algoritme *LSTM* sendiri sudah memiliki akurasi yang cukup bagus jika dibandingkan dengan penelitian [25] dimana hasil akurasi untuk *LSTM* adalah 80,5% dan hasil akurasi untuk *LSTM* dengan 10-*fold cross validation* adalah 80,24%. Berdasarkan penelitian [26] terdapat beberapa kelemahan dan tantangan dalam implementasi algoritme *FCM* untuk segmentasi citra antara lain *FCM* sensitif terhadap inisialisasi pusat *cluster* walaupun telah dilakukan berbagai penelitian tetapi permasalahan ini masih ada, kedua segmentasi citra dengan *FCM* sangat berpengaruh pada jumlah *cluster* untuk setiap citra sehingga diperlukan teknik otomatis untuk membuat agar *FCM* dapat menentukan berapa jumlah *cluster* yang optimal pada setiap citra, permasalahan yang ketiga adalah segmentasi citra dengan *FCM* sensitif terhadap *noise* yaitu piksel pada citra *grayscale* yang hampir mirip fiturnya dan hal tersebut menyebabkan algoritme *FCM* mengklasifikasikan fitur tersebut ke *cluster* yang salah.