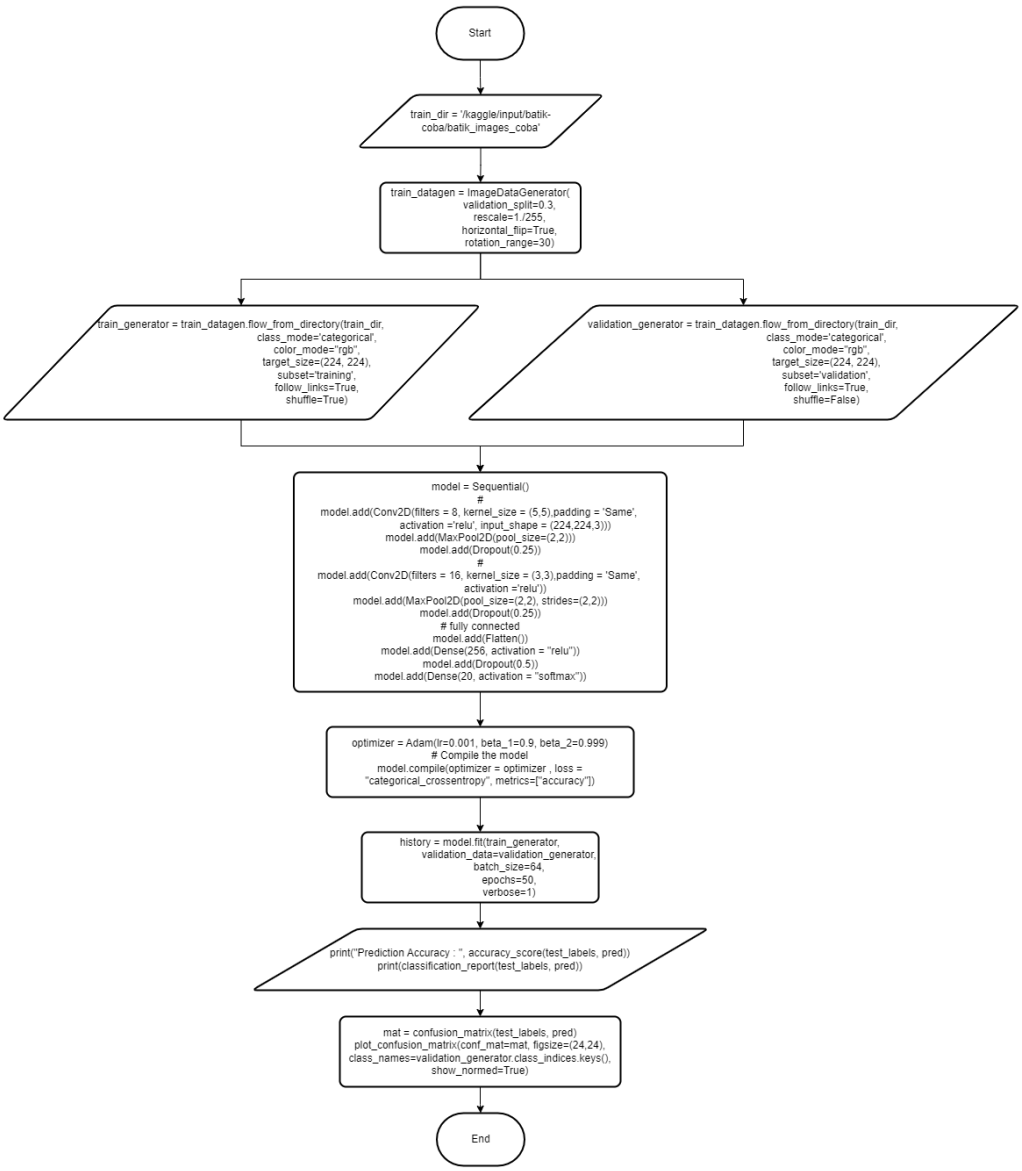
BAB V. IMPLEMENTASI DAN PENGUJIAN

5.1 Implementasi Uji Coba

Implementasi proses pelatihan yang ada pada pelatihan Implementasi *Image Classification* pada Jenis-Jenis Batik Menggunakan *Convolutional Neural Network* dengan Model *EfficientNet* dari beberapa potongan kode program sebagai berikut.

5.1.1 Skenario 1

Uji coba ini dilakukan dengan melakukan proses training dengan menggunakan Convolutional Neural Network tanpa menggunakan EfficientNet



Gambar 5. 1 *Flowchart Code* Skenario 1

Berikut merupakan alur jalan code atau *flowchart* code pada penelitian tentang Implementasi *Image Classification* pada Jenis-Jenis Batik Menggunakan Algoritma *Convolutional Neural Network* dengan Model *EfficientNet* untuk skenario 2. Pada alur pertama mesin akan melakukan input data berupa dataset citra batik yang berisikan 20 class dengan jumlah citra sebanyak 2000 gambar. Inputan dataset citra batik ini akan menginisialisasi objek dengan nama train_dir.

```
train_dir = '/kaggle/input/batik-coba/batik_images_coba'
```

Setelah mesin melakukan input dataset yang berisikan citra batik, mesin akan melakukan proses augmentasi dataset yang berisikan citra batik. Proses augmentasi ini memiliki fungsi untuk meningkatkan keanekaragaman dan jumlah data pelatihan tersedia. Dari proses augmentasi ini juga dapat meningkatkan performa model dan membantu untuk mencegah *overfitting*.

```
train_datagen = ImageDataGenerator(
                    validation_split=0.3,
                    rescale=1./255,
                    horizontal_flip=True,
                    rotation_range=30)
```

Pada proses augmentasi data ini akan menginisialisasi objek dengan nama train_datagen. Proses augmentasi data ini melibatkan kelas ImageDataGenerator dalam library TensorFlow. ImageDataGenerator merupakan alat yang berguna untuk mempersiapkan data gambar untuk pelatihan model jaringan saraf. Pada proses augmentasi ini, akan melakukan beberapa perlakuan untuk data seperti:

1. validation split yang memiliki fungsi untuk membagi data training dan data validation yang digunakan untuk pelatihan. Hasil dari pembagian data ini adalah 30% untuk data validasi dan 70% untuk data training.
2. rescale yang memiliki fungsi untuk menormalisasi data citra dengan membagi setiap nilai piksel dengan 255.
3. horizontal_flip yang memiliki fungsi untuk mengaktifan pembalikan citra pada datset secara horizontal.
4. rotation_range yang memiliki fungsi untuk mengaktifkan rotasi citra pada dataset senilai 30 derajat.

Setelah mesin melakukan proses augmentasi data, mesin akan membagi data citra menjadi 2, yaitu data train yang diargumenkan dengan train_genetaror dan data validasi yang diargumenkan dengan validation_generator. Proporsi untuk pembagian antara data train dengan data validasi sebanyak 70% untuk data train dan 30% untuk data validasi.

```
train_generator = train_datagen.flow_from_directory(
          train_dir,
          class_mode='categorical',
          color_mode="rgb",
          target_size=(224, 224),
          subset='training',
          follow_links=True,
          shuffle=True)
```

```
validation_generator = train_datagen.flow_from_directory(
          train_dir,
          class_mode='categorical',
          color_mode="rgb",
          target_size=(224, 224),
          subset='validation',
          follow_links=True,
          shuffle=False)
```

Setelah melakukan pembagian data antara data training dengan data validation, tahap selanjutnya adalah melakukan penambahan rancangan layer sesuai dengan arsitektur *Convolutional Neural Network*.

```
model = Sequential()
#
model.add(Conv2D(filters = 8, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (224,224,3)))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))
#
model.add(Conv2D(filters = 16, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))
# fully connected
model.add(Flatten())
```

```
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(20, activation = "softmax"))
```

Berikut merupakan rancangan layer untuk uji coba pelatihan menggunakan arsitektur Convolutional Neural Network

A. Conv2D

Conv2D merupakan lapisan konvolusi pertama pada uji coba pelatihan kali ini. Pada lapisan ini menggunakan 8 filter konvolusi dengan ukuran kernel 5x5. Statemet padding = ‘same’ digunakan untuk mempertahankan dimensi gambar input. Pada lapisan ini juga mengaktifkan fungsi aktivasi ReLu dan memiliki inputan gambar sebesar 224x224 piksel dan 3 saluran gambar (RGB).

B. MaxPool2D

MaxPool2D merupakan lapisan pooling pertama. Lapisan ini menggunakan *max pooling* dengan ukuran pool 2x2 yang memiliki fungsi untuk mengurangi ukuran representasi fitur.

C. Dropout

Dropout kali ini merupakan lapisan dropout pertama. Lapisan dropout digunakan untuk menghindari *overfitting*. Pada lapisan pertama dropout menggunakan tingkat dropout sebesar 0.25

D. Conv2D

Conv2D kali ini merupakan lapisan konvolosi yang kedua. Pada lapisan ini menggunakan 16 filter konvolusi dengan ukuran kernel 3x3.

E. MaxPool2D

MaxPool2D kali ini merupakan lapisan *max pooling* yang kedua. Lapisan ini menggunakan *max pooling* dengan ukuran pool 2x2 dan pergeseran (*strides*) 2x2 untuk mengurangi representasi fitur

F. Dropout

Droput kali ini merupakan lapisan dropout kedua. Pada lapisan kedua dropout menggunakan tingkat dropout sebesar 0.25

G. Flatten

Flatten merupakan lapisan *fully connected* yang digunakan untuk mengubah representasi matriks 2D yang dihasilkan oleh lapisan konvolusi sebelumnya menjadi vektor 1D

H. Dense

Dense merupakan lapisan output yang terhubung penuh (*fully connected layer*) dengan memiliki 256 unit neuron serta mengaktifkan fungsi aktivasi ReLu yang memungkinkan keluaran positif untuk melalui neuron dan mematikan keluaran negatif.

I. Dropout

Droput kali ini merupakan lapisan dropout ketiga. Pada lapisan ketiga dropout mennggunakan tingkat dropout sebesar 0.5

J. Dense

Lapisan output ini merupakan lapisan terhubung penuh (fully connected layer) dengan 20 unit neuron yang sesuai dengan jumlah kelas yang diharapkan. Fungsi aktivasi softmax digunakan untuk menghasilkan probabilitas prediksi untuk setiap kelas.

Setelah mesin melakukan perancangan layer untuk *Convolutional Neural Network*, proses selanjutnya yaitu penambahan optimizer dengan menggunakan optimizer Adam dan menggunakan *learning rate* sebesar 0.001

```
optimizer = Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
```

Setelah mesin melakukan penambahan *optimizer*, maka tahap selanjutnya adalah tahap pelatihan (*training*). Proses training data ini akan melibatkan seluruh proses yang telah dirancang. Pada proses training data ini akan dilakukan sebanyak 100 epoch

```
history = model.fit(train_generator,
                    validation_data=validation_generator,
                    batch_size=64,
                    epochs=100,
                    verbose=1)
```

Setelah mesin melakukan proses training, mesin akan mencetak hasil dari proses training data. Hasil dari proses training meliputi: grafik training, akurasi, precision, recall dan f1-score

1. Hasil proses training menggunakan grafik training

```
import matplotlib.pyplot as plt
metrics = history.history
plt.plot(history.epoch,            metrics['categorical_accuracy'],
metrics['val_categorical_accuracy'])
plt.legend(['categorical_accuracy', 'val_categorical_accuracy'])
plt.show()

plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.show()
```

2. Hasil proses training menggunakan akurasi, precision, recall dan f1-score

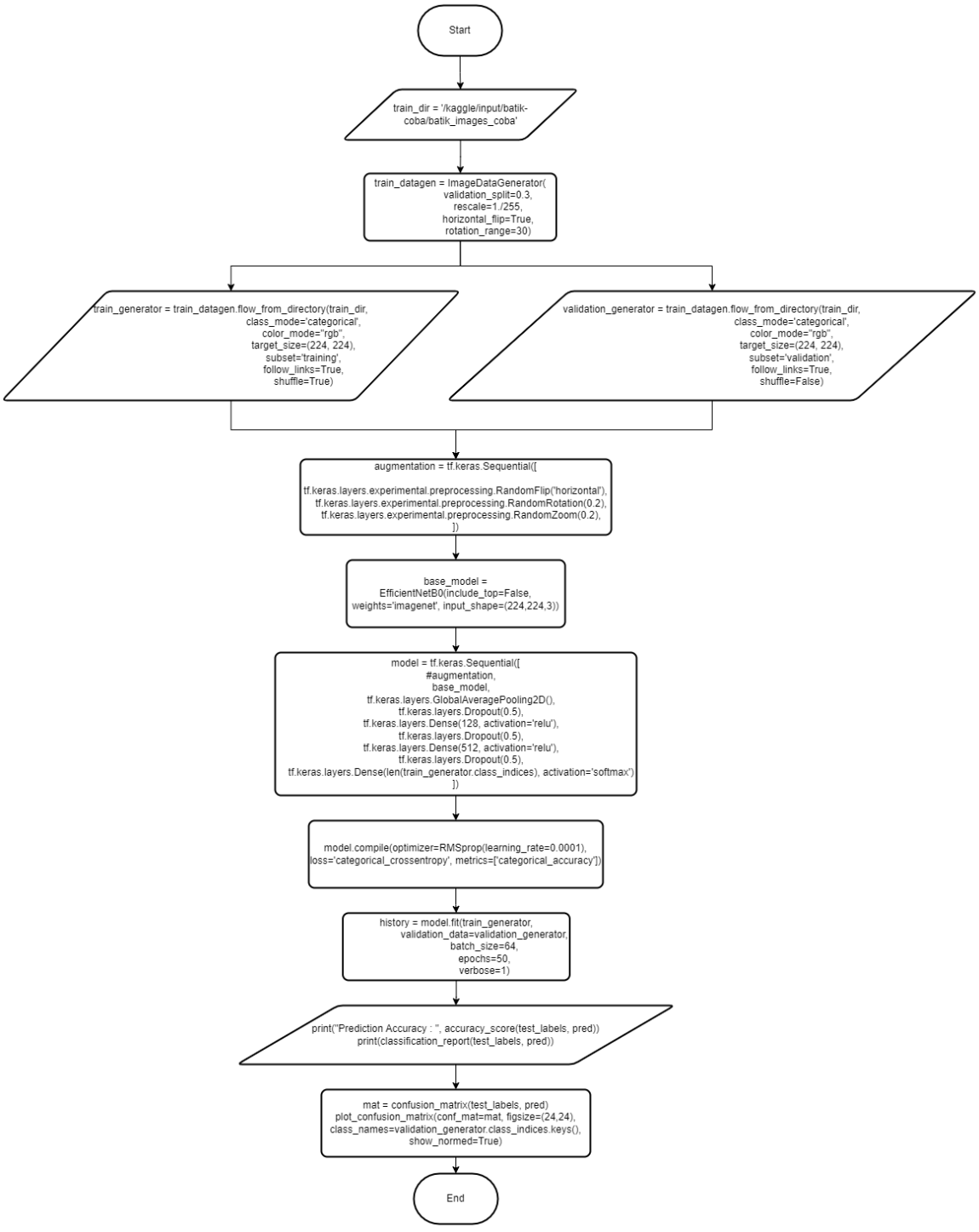
```
test_labels = validation_generator.classes
predictions = model.predict(validation_generator)
pred = np.argmax(predictions, axis=1)

from sklearn.metrics import accuracy_score
print("Prediction Accuracy : ", accuracy_score(test_labels, pred))

from sklearn.metrics import classification_report
print(classification_report(test_labels, pred))
```

Proses selanjutnya dari proses training citra batik adalah tahap pengujian. Uji coba pelatihan kali ini akan menggunakan metode *confusion matrix*.

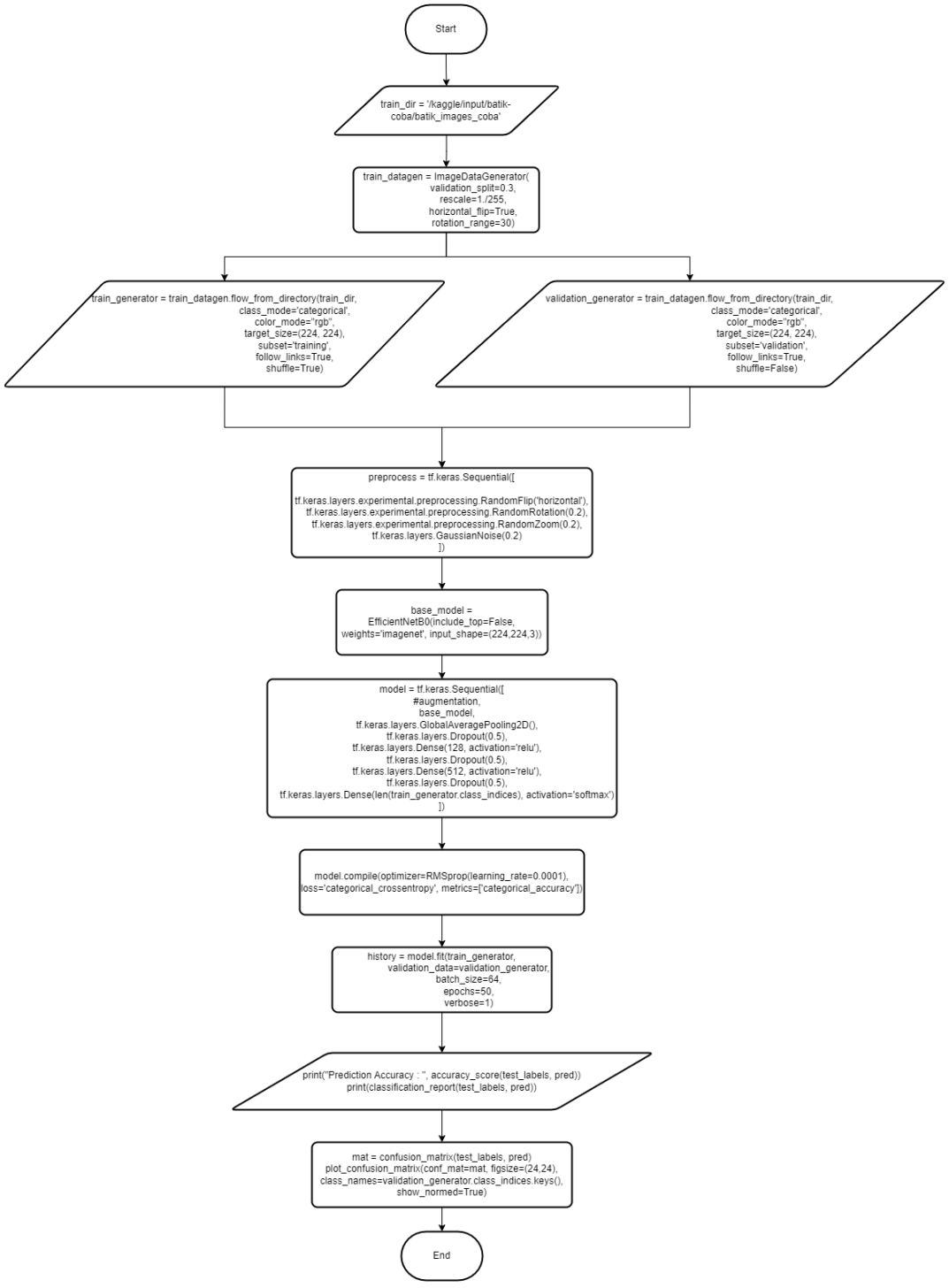
```
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix

mat = confusion_matrix(test_labels, pred)
plot_confusion_matrix(conf_mat=mat,              figsize=(24,24),
class_names=validation_generator.class_indices.keys(),
show_normed=True)
```

5.1.2 Skenario 2

Uji coba ini dilakukan dengan membandingkan model dengan menggunakan data *preprocessing* dengan model tanpa *preprocessing*. *Preprocessing* yang digunakan pada uji coba ini yaitu menggunakan filter GaussianNoise dan akan melakukan pelatihan sebanyak 100 *epoch*.



Gambar 5. 2 *Flowchart Code* Skenario 2 tanpa data *preprocessing*



Gambar 5. 3 *Flowchart Code* Skenario 2 dengan data *preprocessing*

Berikut merupakan alur jalan code atau *flowchart* code pada penelitian tentang Implementasi *Image Classification* pada Jenis-Jenis Batik Menggunakan *Convolutional Neural Network* dengan Model *EfficientNet* untuk skenario 2. Pada alur pertama mesin akan melakukan input data berupa dataset citra batik yang

berisikan 20 class dengan jumlah citra sebanyak 2000 gambar. Inputan dataset citra batik ini akan menginisialisasi objek dengan nama train_dir.

```
train_dir = '/kaggle/input/batik-coba/batik_images_coba'
```

Setelah mesin melakukan input dataset yang berisikan citra batik, mesin akan melakukan proses augmentasi dataset yang berisikan citra batik. Proses augmentasi ini memiliki fungsi untuk meningkatkan keanekaragaman dan jumlah data pelatihan tersedia. Dari proses augmentasi ini juga dapat meningkatkan performa model dan membantu untuk mencegah *overfitting*.

```
train_datagen = ImageDataGenerator(
                    validation_split=0.3,
                    rescale=1./255,
                    horizontal_flip=True,
                    rotation_range=30)
```

Pada proses augmentasi data ini akan menginisialisasi objek dengan nama train_datagen. Proses augmentasi data ini melibatkan kelas ImageDataGenerator dalam library TensorFlow. ImageDataGenerator merupakan alat yang berguna untuk mempersiapkan data gambar untuk pelatihan model jaringan saraf. Pada proses augmentasi ini, akan melakukan beberapa perlakuan untuk data seperti:

1. validation split yang memiliki fungsi untuk membagi data training dan data validation yang digunakan untuk pelatihan. Hasil dari pembagian data ini adalah 30% untuk data validasi dan 70% untuk data training.
2. rescale yang memiliki fungsi untuk menormalisasi data citra dengan membagi setiap nilai piksel dengan 255.
3. horizontal_flip yang memiliki fungsi untuk mengaktifan pembalikan citra pada datset secara horizontal.
4. rotation_range yang memiliki fungsi untuk mengaktifkan rotasi citra pada dataset senilai 30 derajat.

Setelah mesin melakukan proses augmentasi data, mesin akan membagi data citra menjadi 2, yaitu data train yang diargumenkan dengan train_genetaror dan data validasi yang diargumenkan dengan validation_generator. Proporsi untuk pembagian antara data train dengan data validasi sebanyak 70% untuk data train dan 30% untuk data validasi.

```
train_generator = train_datagen.flow_from_directory(
          train_dir,
          class_mode='categorical',
          color_mode="rgb",
          target_size=(224, 224),
          subset='training',
          follow_links=True,
          shuffle=True)
```

```
validation_generator = train_datagen.flow_from_directory(
          train_dir,
          class_mode='categorical',
          color_mode="rgb",
          target_size=(224, 224),
          subset='validation',
          follow_links=True,
          shuffle=False)
```

Setelah mesin melakukan proses pembagian data citra, mesin akan melakukan proses pre-processing. Pada skenario uji coba kali ini, mesin akan menambahkan data preprocessing filter GaussianNoise senilai 0,2 yang akan digunakan untuk menghasilkan noise dan tidak menggunakan data preprocessing.

1. Model tanpa menggunakan data *preprocessing*

```
augmentation = tf.keras.Sequential([
tf.keras.layers.experimental.preprocessing.RandomFlip('horizonta
l'),
tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
tf.keras.layers.experimental.preprocessing.RandomZoom(0.2)
])
```

2. Model menggunakan data *preprocessing* filter GaussianNoise senilai 0.2

```
preprocess = tf.keras.Sequential([
tf.keras.layers.experimental.preprocessing.RandomFlip('horizonta
l'),
tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
tf.keras.layers.experimental.preprocessing.RandomZoom(0.2),
tf.keras.layers.GaussianNoise(0.2)
])
```

Setelah mesin melakukan proses pre-processing, mesin akan menambahkan *EfficientNet* didalam objek base_model. Pada EfficientNet kali ini akan menerima

inputan citra dengan ukuran 224 x 224 piksel dan 3 saluran warna (RGB) yang diatur dalam input_shape.

```
base_model = EfficientNetB0(include_top=False, weights='imagenet',
input_shape=(224,224,3))
```

Setelah mesin melakukan penambahan layer untuk EfficientNet, mesin akan menambahkan racangan layer seperti dengan potongan code tersebut

```
      model = tf.keras.Sequential([
          #augmentation,
          base_model,
          tf.keras.layers.GlobalAveragePooling2D(),
          tf.keras.layers.Dropout(0.5),
          tf.keras.layers.Dense(128, activation='relu'),
          tf.keras.layers.Dropout(0.5),
          tf.keras.layers.Dense(512, activation='relu'),
          tf.keras.layers.Dropout(0.5),
          tf.keras.layers.Dense(len(train_generator.class_indices),
      activation='softmax')
])
```

Setelah mesin melakukan penambahan layer sesuai dengan rancangan, tahap selanjutnya adalah penambahan *optimizer*. Pada pelatihan kali ini menggunakan RMSprop dengan *learning rate* sebesar 0.0001

```
model.compile(optimizer=RMSprop(learning_rate=0.0001),
loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
```

Setelah mesin melakukan penambahan *optimizer*, maka tahap selanjutnya adalah tahap pelatihan (*training*). Proses training data ini akan melibatkan seluruh proses yang telah dirancang. Pada proses training data ini akan dilakukan sebanyak 100 epoch

```
      history = model.fit(train_generator,
                          validation_data=validation_generator,
                          batch_size=64,
                          epochs=50,
                          verbose=1)
```

Setelah mesin melakukan proses training, mesin akan mencetak hasil dari proses training data. Hasil dari proses training meliputi: grafik training, akurasi, precision, recall dan f1-score

1. Hasil proses training menggunakan grafik training

```
import matplotlib.pyplot as plt
metrics = history.history
plt.plot(history.epoch,            metrics['categorical_accuracy'],
metrics['val_categorical_accuracy'])
plt.legend(['categorical_accuracy', 'val_categorical_accuracy'])
plt.show()

plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.show()
```

2. Hasil proses training menggunakan akurasi, precision, recall dan f1-score

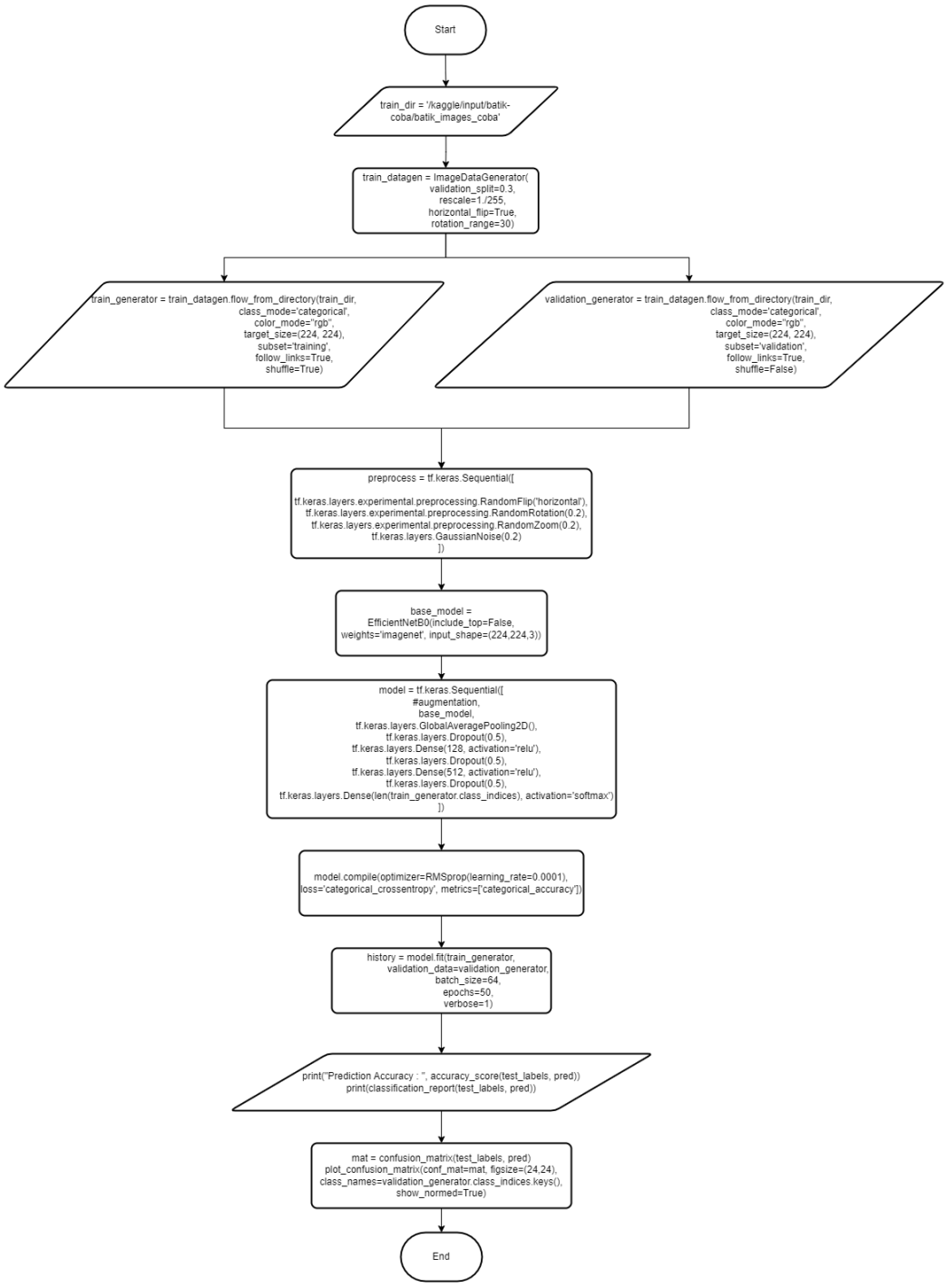
```
test_labels = validation_generator.classes
predictions = model.predict(validation_generator)
pred = np.argmax(predictions, axis=1)

from sklearn.metrics import accuracy_score
print("Prediction Accuracy : ", accuracy_score(test_labels, pred))

from sklearn.metrics import classification_report
print(classification_report(test_labels, pred))
```

Proses selanjutnya dari proses training citra batik adalah tahap pengujian. Uji coba pelatihan kali ini akan menggunakan metode *confusion matrix*.

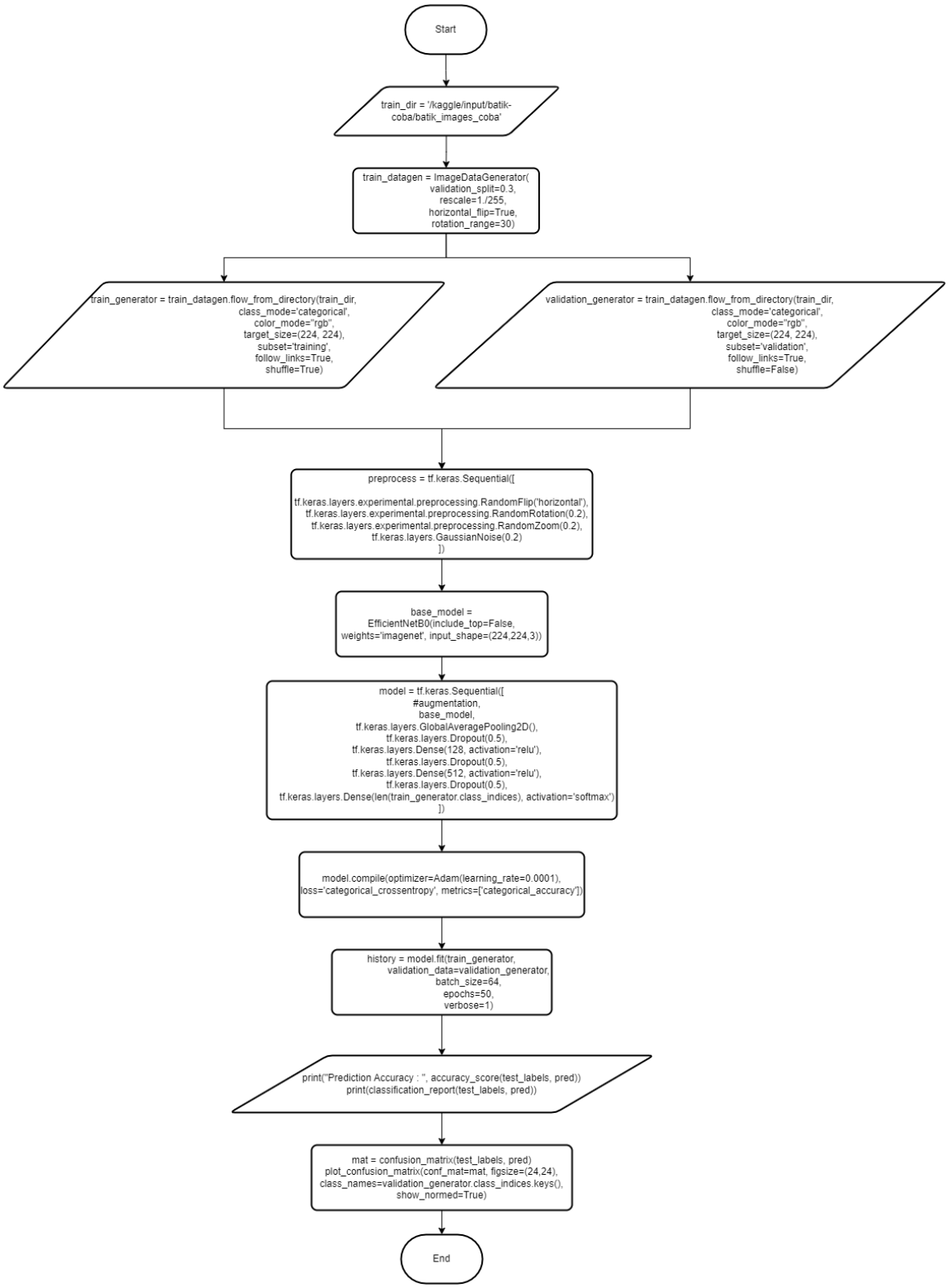
```
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix

mat = confusion_matrix(test_labels, pred)
plot_confusion_matrix(conf_mat=mat,              figsize=(24,24),
class_names=validation_generator.class_indices.keys(),
show_normed=True)
```

5.1.3 Skenario 3

Uji coba ini dilakukan dengan membandingkan model dengan menggunakan *optimizer* Adam dan RMSprop dengan *learning rate* sebesar 0.0001 dan melakukan pelatihan sebanyak 100 epoch.



Gambar 5. *4 Flowchart Code* Skenario 3 dengan *optimizer* RMSprop *learning rate* 0.0001



Gambar 5. 5 *Flowchart Code* Skenario 3 dengan *optimizer* Adam *learning rate* 0.0001

Berikut merupakan alur jalan code atau *flowchart* code pada penelitian tentang Implementasi *Image Classification* pada Jenis-Jenis Batik Menggunakan *Convolutional Neural Network* dengan Model *EfficientNet* untuk skenario 3. Pada alur pertama mesin akan melakukan input data berupa dataset citra batik yang berisikan 20 class dengan jumlah citra sebanyak 2000 gambar. Inputan dataset citra batik ini akan menginisialisasi objek dengan nama train_dir.

```
train_dir = '/kaggle/input/batik-coba/batik_images_coba'
```

Setelah mesin melakukan input dataset yang berisikan citra batik, mesin akan melakukan proses augmentasi dataset yang berisikan citra batik. Proses augmentasi ini memiliki fungsi untuk meningkatkan keanekaragaman dan jumlah data pelatihan tersedia. Dari proses augmentasi ini juga dapat meningkatkan performa model dan membantu untuk mencegah *overfitting*.

```
train_datagen = ImageDataGenerator(
                    validation_split=0.3,
                    rescale=1./255,
                    horizontal_flip=True,
                    rotation_range=30)
```

Setelah mesin melakukan proses augmentasi data, mesin akan membagi data citra menjadi 2, yaitu data train yang diargumenkan dengan train_genetaror dan data validasi yang diargumenkan dengan validation_generator. Proporsi untuk pembagian antara data train dengan data validasi sebanyak 70% untuk data train dan 30% untuk data validasi.

```
train_generator = train_datagen.flow_from_directory(
          train_dir,
          class_mode='categorical',
          color_mode="rgb",
          target_size=(224, 224),
          subset='training',
          follow_links=True,
          shuffle=True)
```

```
validation_generator = train_datagen.flow_from_directory(
          train_dir,
          class_mode='categorical',
          color_mode="rgb",
          target_size=(224, 224),
```

```
            subset='validation',
            follow_links=True,
            shuffle=False)
```

Setelah mesin melakukan proses pembagian data citra, mesin akan melakukan proses pre-processing. Pada tahap proses pre-processing kali ini, mesin akan menambahkan filter GaussianNoise senilai 0,2 yang akan digunakan untuk menghasilkan noise. Selain itu menambahkan filter Gaussian Noise, juga menambahkan Random Flip secara horizontal, Random Rotation, dan Random Zoom.

```
preprocess = tf.keras.Sequential([

tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
    tf.keras.layers.experimental.preprocessing.RandomZoom(0.2),
    tf.keras.layers.GaussianNoise(0.2)
])
```

Setelah mesin melakukan proses pre-processing, mesin akan menambahkan *EfficientNet* didalam objek base_model. Pada EfficientNet kali ini akan menerima inputan citra dengan ukuran 224 x 224 piksel dan 3 saluran warna (RGB) yang diatur dalam input_shape.

```
base_model = EfficientNetB0(include_top=False, weights='imagenet',
input_shape=(224,224,3))
```

Setelah mesin melakukan penambahan layer untuk EfficientNet, mesin akan menambahkan racangan layer seperti dengan potongan code tersebut

```
     model = tf.keras.Sequential([
         #augmentation,
         base_model,
         tf.keras.layers.GlobalAveragePooling2D(),
         tf.keras.layers.Dropout(0.5),
         tf.keras.layers.Dense(128, activation='relu'),
         tf.keras.layers.Dropout(0.5),
         tf.keras.layers.Dense(512, activation='relu'),
         tf.keras.layers.Dropout(0.5),
         tf.keras.layers.Dense(len(train_generator.class_indices),
     activation='softmax')
])
```

Setelah mesin melakukan penambahan layer sesuai dengan rancangan, tahap selanjutnya adalah penambahan *optimizer*. Pada skenario uji coba kali ini dilakukan dengan membandingkan model dengan menggunakan *optimizer* Adam dan RMSprop dengan *learning rate* sebesar 0.0001

1. *Optimizer* Adam dengan *learning rate* sebesar 0.0001

```
model.compile(optimizer=Adam(learning_rate=0.0001),
loss='categorical_crossentropy', metrics=['categorical_accuracy'])
```

2. *Optimizer* RMSProp dengan *learning rate* sebesar 0.0001

```
model.compile(optimizer=RMSprop(learning_rate=0.0001),
loss='categorical_crossentropy', metrics=['categorical_accuracy'])
```

Setelah mesin melakukan penambahan *optimizer*, maka tahap selanjutnya adalah tahap pelatihan (*training*). Proses training data ini akan melibatkan seluruh proses yang telah dirancang. Pada proses training data ini akan dilakukan sebanyak 100 epoch

```
    history = model.fit(train_generator,
                        validation_data=validation_generator,
                        batch_size=64,
                        epochs=50,
                        verbose=1)
```

Setelah mesin melakukan proses training, mesin akan mencetak hasil dari proses training data. Hasil dari proses training meliputi: grafik training, akurasi, precision, recall dan f1-score

1. Hasil proses training menggunakan grafik training

```
import matplotlib.pyplot as plt
metrics = history.history
plt.plot(history.epoch,            metrics['categorical_accuracy'],
metrics['val_categorical_accuracy'])
plt.legend(['categorical_accuracy', 'val_categorical_accuracy'])
plt.show()

plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.show()
```

2. Hasil proses training menggunakan akurasi, precision, recall dan f1-score

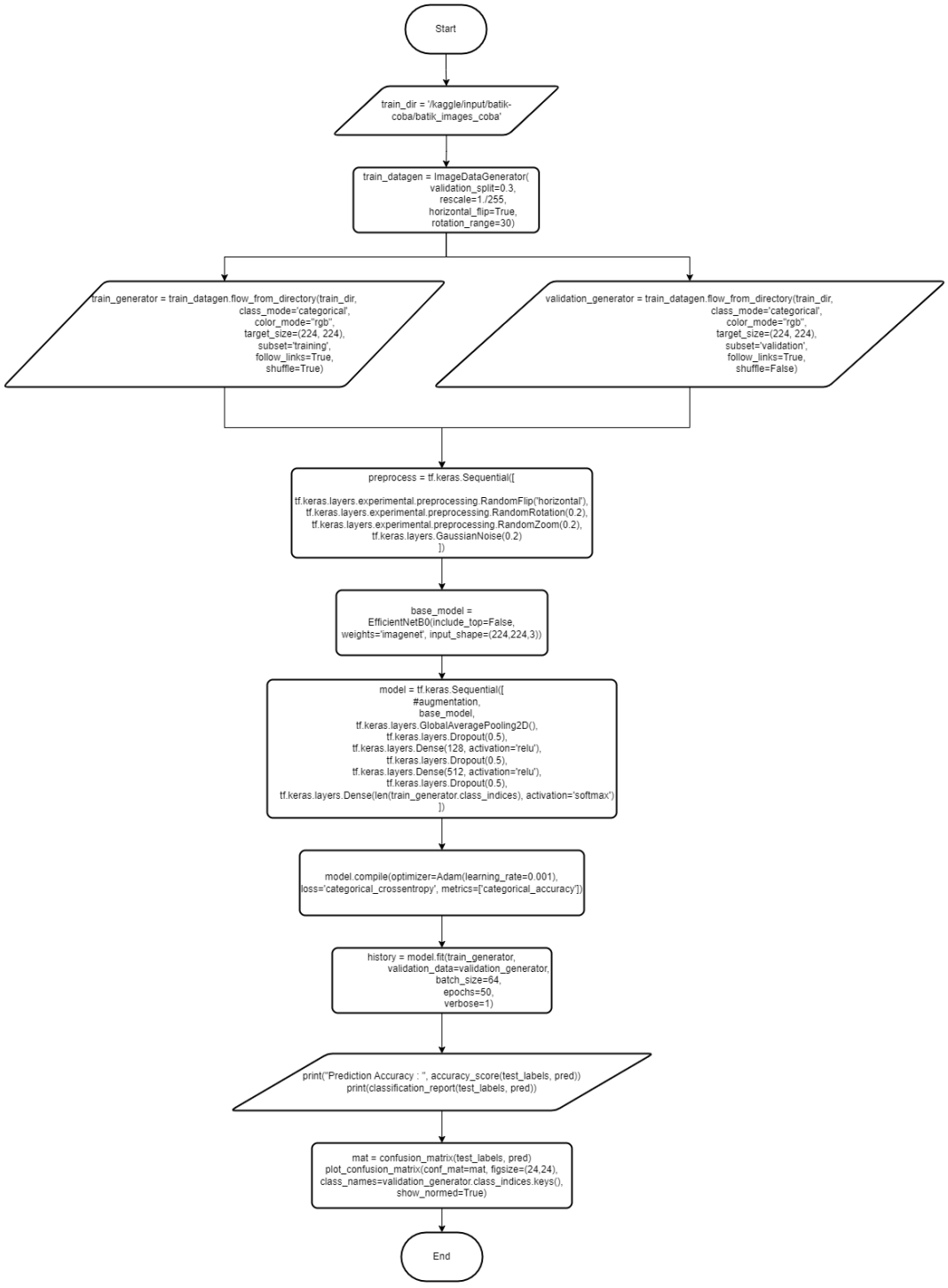
```
test_labels = validation_generator.classes
predictions = model.predict(validation_generator)
pred = np.argmax(predictions, axis=1)

from sklearn.metrics import accuracy_score
print("Prediction Accuracy : ", accuracy_score(test_labels, pred))

from sklearn.metrics import classification_report
print(classification_report(test_labels, pred))
```

Proses selanjutnya dari proses training citra batik adalah tahap pengujian. Uji coba pelatihan kali ini akan menggunakan metode *confusion matrix*.

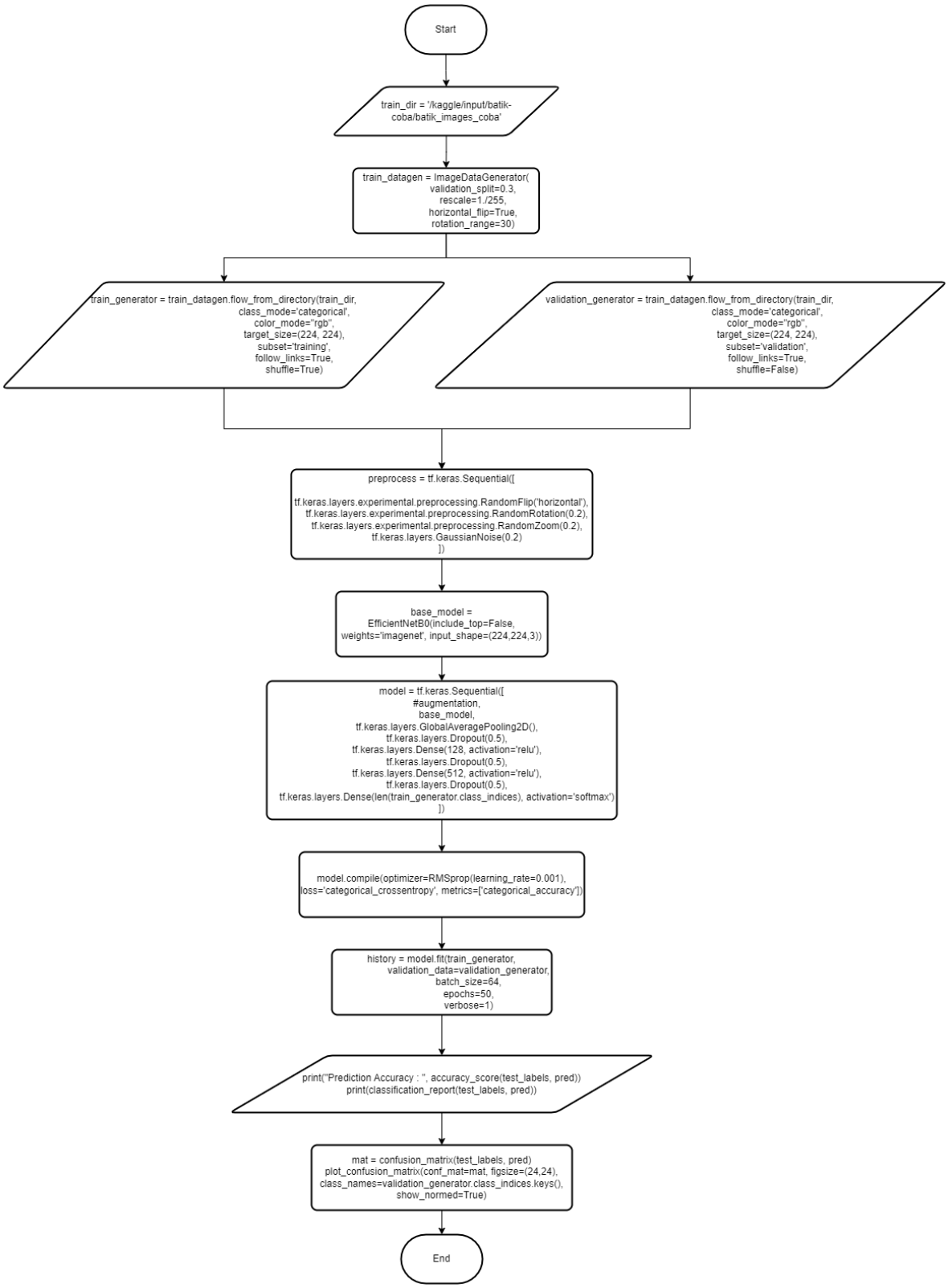
```
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix

mat = confusion_matrix(test_labels, pred)
plot_confusion_matrix(conf_mat=mat,              figsize=(24,24),
class_names=validation_generator.class_indices.keys(),
show_normed=True)
```

5.1.4 Skenario 4

Uji coba ini dilakukan dengan membandingkan model dengan menggunakan *optimizer* Adam dan RMSprop dengan *learning rate* sebesar 0.001 dan melakukan pelatihan sebanyak 100 epoch.



Gambar 5. 6 *Flowchart Code* Skenario 4 dengan *optimizer* Adam *learning rate* 0.001



Gambar 5. 7 *Flowchart Code* Skenario 4 dengan *optimizer* RMSprop *learning rate* 0.001

Berikut merupakan alur jalan code atau *flowchart* code pada penelitian tentang Implementasi *Image Classification* pada Jenis-Jenis Batik Menggunakan *Convolutional Neural Network* dengan Model *EfficientNet* untuk skenario 3. Pada alur pertama mesin akan melakukan input data berupa dataset citra batik yang berisikan 20 class dengan jumlah citra sebanyak 2000 gambar. Inputan dataset citra batik ini akan menginisialisasi objek dengan nama train_dir.

```
train_dir = '/kaggle/input/batik-coba/batik_images_coba'
```

Setelah mesin melakukan input dataset yang berisikan citra batik, mesin akan melakukan proses augmentasi dataset yang berisikan citra batik. Proses augmentasi ini memiliki fungsi untuk meningkatkan keanekaragaman dan jumlah data pelatihan tersedia. Dari proses augmentasi ini juga dapat meningkatkan performa model dan membantu untuk mencegah *overfitting*.

```
train_datagen = ImageDataGenerator(
                    validation_split=0.3,
                    rescale=1./255,
                    horizontal_flip=True,
                    rotation_range=30)
```

Setelah mesin melakukan proses augmentasi data, mesin akan membagi data citra menjadi 2, yaitu data train yang diargumenkan dengan train_genetaror dan data validasi yang diargumenkan dengan validation_generator. Proporsi untuk pembagian antara data train dengan data validasi sebanyak 70% untuk data train dan 30% untuk data validasi.

```
train_generator = train_datagen.flow_from_directory(
          train_dir,
          class_mode='categorical',
          color_mode="rgb",
          target_size=(224, 224),
          subset='training',
          follow_links=True,
          shuffle=True)
```

```
validation_generator = train_datagen.flow_from_directory(
          train_dir,
          class_mode='categorical',
          color_mode="rgb",
          target_size=(224, 224),
```

```
            subset='validation',
            follow_links=True,
            shuffle=False)
```

Setelah mesin melakukan proses pembagian data citra, mesin akan melakukan proses pre-processing. Pada tahap proses pre-processing kali ini, mesin akan menambahkan filter GaussianNoise senilai 0,2 yang akan digunakan untuk menghasilkan noise. Selain itu menambahkan filter Gaussian Noise, juga menambahkan Random Flip secara horizontal, Random Rotation, dan Random Zoom.

```
preprocess = tf.keras.Sequential([

tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
    tf.keras.layers.experimental.preprocessing.RandomZoom(0.2),
    tf.keras.layers.GaussianNoise(0.2)
])
```

Setelah mesin melakukan proses pre-processing, mesin akan menambahkan *EfficientNet* didalam objek base_model. Pada EfficientNet kali ini akan menerima inputan citra dengan ukuran 224 x 224 piksel dan 3 saluran warna (RGB) yang diatur dalam input_shape.

```
base_model = EfficientNetB0(include_top=False, weights='imagenet',
input_shape=(224,224,3))
```

Setelah mesin melakukan penambahan layer untuk EfficientNet, mesin akan menambahkan racangan layer seperti dengan potongan code tersebut

```
     model = tf.keras.Sequential([
         #augmentation,
         base_model,
         tf.keras.layers.GlobalAveragePooling2D(),
         tf.keras.layers.Dropout(0.5),
         tf.keras.layers.Dense(128, activation='relu'),
         tf.keras.layers.Dropout(0.5),
         tf.keras.layers.Dense(512, activation='relu'),
         tf.keras.layers.Dropout(0.5),
         tf.keras.layers.Dense(len(train_generator.class_indices),
     activation='softmax')
])
```

Setelah mesin melakukan penambahan layer sesuai dengan rancangan, tahap selanjutnya adalah penambahan *optimizer*. Pada skenario uji coba kali ini dilakukan dengan membandingkan model dengan menggunakan *optimizer* Adam dan RMSprop dengan *learning rate* sebesar 0.001

1. *Optimizer* Adam dengan *learning rate* sebesar 0.001

```
model.compile(optimizer=Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['categorical_accuracy'])
```

2. *Optimizer* RMSProp dengan *learning rate* sebesar 0.001

```
model.compile(optimizer=RMSprop(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['categorical_accuracy'])
```

Setelah mesin melakukan penambahan *optimizer*, maka tahap selanjutnya adalah tahap pelatihan (*training*). Proses training data ini akan melibatkan seluruh proses yang telah dirancang. Pada proses training data ini akan dilakukan sebanyak 100 epoch

```
    history = model.fit(train_generator,
                        validation_data=validation_generator,
                        batch_size=64,
                        epochs=100,
                        verbose=1)
```

Setelah mesin melakukan proses training, mesin akan mencetak hasil dari proses training data. Hasil dari proses training meliputi: grafik training, akurasi, precision, recall dan f1-score

3. Hasil proses training menggunakan grafik training

```
import matplotlib.pyplot as plt
metrics = history.history
plt.plot(history.epoch,            metrics['categorical_accuracy'],
metrics['val_categorical_accuracy'])
plt.legend(['categorical_accuracy', 'val_categorical_accuracy'])
plt.show()

plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.show()
```

4. Hasil proses training menggunakan akurasi, precision, recall dan f1-score

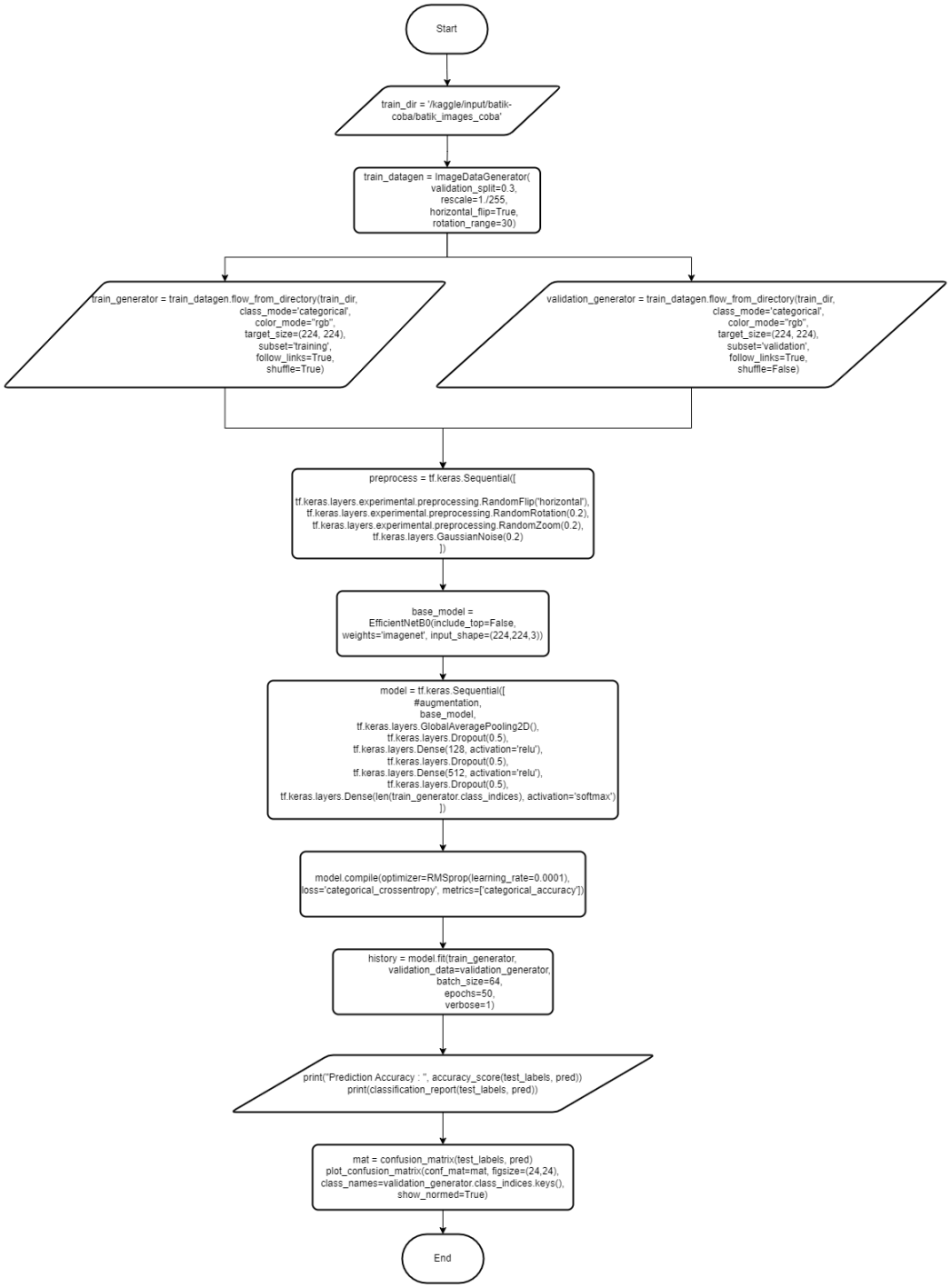
```
test_labels = validation_generator.classes
predictions = model.predict(validation_generator)
pred = np.argmax(predictions, axis=1)

from sklearn.metrics import accuracy_score
print("Prediction Accuracy : ", accuracy_score(test_labels, pred))

from sklearn.metrics import classification_report
print(classification_report(test_labels, pred))
```

Proses selanjutnya dari proses training citra batik adalah tahap pengujian. Uji coba pelatihan kali ini akan menggunakan metode *confusion matrix*.

```
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix

mat = confusion_matrix(test_labels, pred)
plot_confusion_matrix(conf_mat=mat,              figsize=(24,24),
class_names=validation_generator.class_indices.keys(),
show_normed=True)
```

5.1.5 Skenario 5

Uji coba ini dilakukan dengan membandingkan model dengan menggunakan *EfficientNetB1* hingga *EfficientNetB7 optimizer* RMSprop dengan *learning rate* sebesar 0.0001 dan melakukan pelatihan sebanyak 100 epoch.



Gambar 5. 8 *Flowchart Code* Skenario 5

Pada skenario ini, yang membedakan dengan skenario yang lainnya adalah penggunaan layer *EfficientNet*.

1. Layer *EfficientNetB1*

```
base_model = EfficientNetB1(include_top=False, weights='imagenet', input_shape=(224,224,3))
```

2. Layer *EfficientNetB2*

```
base_model = EfficientNetB2(include_top=False, weights='imagenet',
input_shape=(224,224,3))
```

3. Layer *EfficientNetB3*

```
base_model = EfficientNetB3(include_top=False, weights='imagenet',
input_shape=(224,224,3))
```

4. Layer *EfficientNetB4*

```
base_model = EfficientNetB4(include_top=False, weights='imagenet',
input_shape=(224,224,3))
```

5. Layer *EfficientNetB5*

```
base_model = EfficientNetB5(include_top=False, weights='imagenet',
input_shape=(224,224,3))
```

6. Layer *EfficientNetB6*

```
base_model = EfficientNetB6(include_top=False, weights='imagenet',
input_shape=(224,224,3))
```

7. Layer *EfficientNetB7*

```
base_model = EfficientNetB7(include_top=False, weights='imagenet',
input_shape=(224,224,3))
```