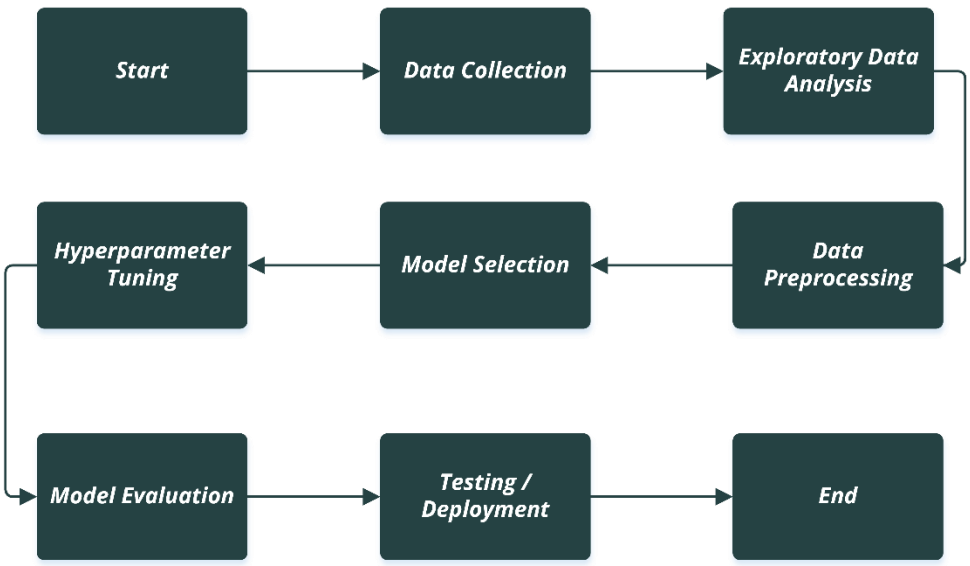
BAB III

METODOLOGI PENELITIAN

3.1 Alur Penelitian

Secara umum penelitian ini dilakukan untuk menyusun suatu model yang dapat dikembangkan untuk mempercepat dan meningkatkan akurasi dalam mendeteksi penyakit tumor otak berdasarkan gambar MRI. Adapun tahapan penelitian yang dilakukan untuk mengembangkan model *machine learning* yang dibutuhkan menurut Aurélien Géron [17] adalah sebagai berikut.



Gambar 3.1 Alur Penelitian

3.2 Pengumpulan Data

Penelitian ini menggunakan *dataset* gambar MRI yang diperoleh dari repositori

Kaggle. *Dataset* yang digunakan berjumlah 3000 gambar yang terdiri atas 1500 gambar yang terdiagnosis tumor otak meniongoma dan 1500 gambar dalam keadaan sehat. Gambar MRI yang digunakan dalam dataset memiliki rerata dimensi 300 x 300 *pixel* dengan jenis gambar *RGB*.

3.3 *Exploratory Data Analysis*

Data yang telah dikumpulkan perlu dianalis terlebih dahulu untuk mengetahui karakteristik dataset seperti sebaran gambar. Gambar hasil pindaian MRI yang terdiagnosis tumor otak dan sehat akan ditampilkan secara acak untuk mengetahui bentuk tumor secara umum.

3.4 *Data Preprocessing*

Sebelum dilatih, dataset akan diproses terlebih dahulu ke dalam format tertentu agar performa model menjadi lebih baik. *Dataset* akan dibagi menjadi *training*, *validation*, dan *testing* dengan rasio 70:20:10. Lalu, data gambar yang memiliki nilai piksel dalam rentang antara 0 – 255 akan dinormalisasi ke dalam rentang nilai 0 – 1 untuk mempermudah proses pembelajaran dan mempermudah model dalam mencapai konvergensi [9].

```
def split_data(source, target):
    '''Split Dataset From source to dataset with split_size proportion for
train-val-test : 70 - 20 - 10'''

    for dir in os.listdir(source):
        file_dir_path = os.path.join(source, dir)
        all_files_dir = []
        print(f"adding {dir} to variabel")

        for file in os.listdir(os.path.join(source,dir)):
            file_path = os.path.join(file_dir_path,file)
            if os.path.getsize(file_path):
                all_files_dir.append(file)
                print(f"Adding Non-Zero File {file} to variabel")
            else:
                print('{} is zero length, so ignoring'.format(file))

        sum_files = len(all_files_dir)
        split_point = int(sum_files * 0.7)
        split_point_mid = int(sum_files * 0.3)
        split_point_end = int(sum_files * 0.1)

        # Shuffle images for train-val-set within proportion
        shuffled = random.sample(all_files_dir, sum_files)
        train_set = shuffled[:split_point]
        val_set = shuffled[-split_point_mid:-split_point_end]
        test_set = shuffled[-split_point_end:]

        # Get Dir Path
        train_dir = os.path.join(target, os.path.join("train",dir))
        val_dir = os.path.join(target, os.path.join("val",dir))
        test_dir = os.path.join(target, os.path.join("test",dir))

        os.makedirs(train_dir)
        os.makedirs(val_dir)
        os.makedirs(test_dir)
```

```
        # Copy file to new dir according proportion
        for file in train_set:
            copyfile(os.path.join(file_dir_path, file),
os.path.join(train_dir,file))
            print(f"succes adding {file} to {train_dir}")

        for file in val_set:
            copyfile(os.path.join(file_dir_path, file),
os.path.join(val_dir,file))
            print(f"succes adding {file} to {val_dir}")

        for file in test_set:
            copyfile(os.path.join(file_dir_path, file),
os.path.join(test_dir,file))
            print(f"Succes adding {file} to {test dir}")
```

Gambar 3.2 Fungsi split data python

Untuk mengubah nilai tiap piksel pada gambar, penelitian ini menggunakan *library image data generator* dari *framework tensorflow* untuk menormalisasi nilai piksel ke dalam rentang 0 – 1 dan mengubah ukuran resolusi gambar menjadi 150 x 150 agar model dilatih dengan ukuran gambar yang konsisten.

```
# Apply data augmentation
train_datagen = ImageDataGenerator(
      rescale = 1./255, #normalization to change value from 0 - 255 to 0 -1
      # rotation_range = 30,
      fill_mode = 'nearest'
      )

validation_datagen = ImageDataGenerator(rescale=1./255) #normalization
test_datagen = ImageDataGenerator(rescale=1./255) #normalization

# Data Generator
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size = (img_height, img_width),  # All images will be resized to
150 x 150
        batch_size = BATCH_SIZE,
        shuffle = True,
        class_mode ='binary')

validation_generator = validation_datagen.flow_from_directory(
        validation_dir,  # This is the source directory for training images
        target_size = (img_height, img_width),  # All images will be resized to
150 x 150
        batch_size = BATCH_SIZE,
        class_mode ='binary')

test_generator = test_datagen.flow_from_directory(
        testing_dir,  # This is the source directory for training images
        target_size = (img_height, img_width),  # All images will be resized to
150 x 150
        batch_size = BATCH_SIZE,
        class_mode = 'binary')
```

Gambar 3.3 Kode *image augmentation* python

Pada gambar 3.3 *batch size* diatur dengan nilai 16 untuk *training data*, *validation data* dan *testing* data. *Bacth size* berfungsi sebagai *hyperparameter* yang mengatur jumlah data yang dimasukan sekaligus ke dalam model untuk dilatih.

Sedangkan *class mode* diatur menjadi nilai *binary* karena pengklasifikasian dua kelas.

3.5 *Model Selection*

Untuk mencapai nilai akurasi tinggi dengan nilai *loss* rendah, arsitektur model *deep neural network* perlu memiliki lapisan-lapisan yang ideal. Model dikatakan ideal atau (*good fit)* saat berada diantara *overfitting* dan *underfitting*, dan diantara *undercapacity* dan *overcapacity* [9]. Arsitektur model *deep neural network* yang diusulkan pada penelitian ini ialah *convolutional neural network* dengan detail tiap lapisan yang terdiri atas beberapa *block convolutional* seperti yang disajikan pada tabel 3.2.

Tabel 3.1. Block Convolutional

Block Convolutional	
Lapisan	**Jenis Layer**
1	*Convolutional 2D Layer*
2	*Batch Normalization Layer*
3	*Activation : ReLu Layer*
4	*Convolutional 2D Layer*
5	*Batch Normalization Layer*
6	*Activation : ReLu Layer*
7	*Max Pooling Layer*

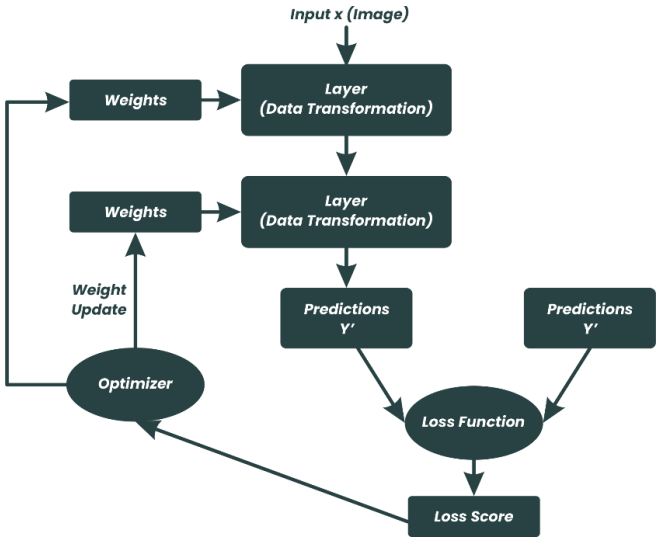
Tabel 3.2. Arsitektur yang diusulkan

Lapisan	Jenis Lapisan
1 - 7	Block Convolutional 1 : (128), Kernel Size : (3 x 3), Input shape : 150 x 150 x 3
8-14	Block Convolutional 2 : (64), Kernel Size : (3 x 3),
15-21	Block Convolutional 3 : (32), Kernel Size : (3 x 3),
29-28	Block Convolutional 4 : (16), Kernel Size : (3 x 3),
36 - 35	Block Convolutional 5 : (8), Kernel Size : (3 x 3),
36	Gloval Average Pooling
37	Dense Layer, Neruron : 128, *Activation Function : ReLU*
38	Dropout Layer (0.2)
39 (*output layer)*	Dense Layer, Neruron : 1, *Activation Function : Sigmoid*

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 150, 150, 3)]     0

conv2d (Conv2D)              (None, 150, 150, 128)     3584

batch_normalization (BatchN  (None, 150, 150, 128)     512
ormalization)

re_lu (ReLU)                 (None, 150, 150, 128)     0

conv2d_1 (Conv2D)            (None, 150, 150, 128)     147584

batch_normalization_1 (Batc  (None, 150, 150, 128)     512
hNormalization)

re_lu_1 (ReLU)               (None, 150, 150, 128)     0

max_pooling2d (MaxPooling2D  (None, 75, 75, 128)       0
)

conv2d_2 (Conv2D)            (None, 75, 75, 64)        73792

batch_normalization_2 (Batc  (None, 75, 75, 64)        256
hNormalization)

re_lu_2 (ReLU)               (None, 75, 75, 64)        0

conv2d_3 (Conv2D)            (None, 75, 75, 64)        36928

batch_normalization_3 (Batc  (None, 75, 75, 64)        256
hNormalization)

re_lu_3 (ReLU)               (None, 75, 75, 64)        0

max_pooling2d_1 (MaxPooling  (None, 37, 37, 64)        0
2D)

conv2d_4 (Conv2D)            (None, 37, 37, 32)        18464

batch_normalization_4 (Batc  (None, 37, 37, 32)        128
hNormalization)

re_lu_4 (ReLU)               (None, 37, 37, 32)        0

conv2d_5 (Conv2D)            (None, 37, 37, 32)        9248

batch_normalization_5 (Batc  (None, 37, 37, 32)        128
hNormalization)

re_lu_5 (ReLU)               (None, 37, 37, 32)        0

max_pooling2d_2 (MaxPooling  (None, 18, 18, 32)        0
2D)

conv2d_6 (Conv2D)            (None, 18, 18, 16)        4624

batch_normalization_6 (Batc  (None, 18, 18, 16)        64
hNormalization)

re_lu_6 (ReLU)               (None, 18, 18, 16)        0

conv2d_7 (Conv2D)            (None, 18, 18, 16)        2320

batch_normalization_7 (Batc  (None, 18, 18, 16)        64
hNormalization)

re_lu_7 (ReLU)               (None, 18, 18, 16)        0

max_pooling2d_3 (MaxPooling  (None, 9, 9, 16)          0
2D)

conv2d_8 (Conv2D)            (None, 9, 9, 8)           1160

batch_normalization_8 (Batc  (None, 9, 9, 8)           32
hNormalization)

re_lu_8 (ReLU)               (None, 9, 9, 8)           0

conv2d_9 (Conv2D)            (None, 9, 9, 8)           584

batch_normalization_9 (Batc  (None, 9, 9, 8)           32
hNormalization)

re_lu_9 (ReLU)               (None, 9, 9, 8)           0

max_pooling2d_4 (MaxPooling  (None, 4, 4, 8)           0
2D)

global_average_pooling2d (G  (None, 8)                 0
lobalAveragePooling2D)

dense (Dense)                (None, 128)               1152

dropout (Dropout)            (None, 128)               0

dense_1 (Dense)              (None, 1)                 129

=================================================================
Total params: 301,553
Trainable params: 300,561
Non-trainable params: 992
_________________________________________________________________
None
```

Gambar 3.4 Arsitektur model yang diusulkan

Selanjutnya model yang telah dirancang akan dilatih. Pada proses pelatihan atau *training model*, *loss function* dan *optimizer* perlu ditentukan terlebih dahulu untuk mengatur bagaimana algoritma *backpropagation* bekerja. *Loss function* berperan sebagai fungsi yang mengembalikan nilai jarak atau gap antara hasil prediksi dan hasil sebenarnya untuk memperbaiki bobot tiap neuron sehingga model dapat mengenali pola pada gambar [9]. Pada penelitian ini, kasus yang diteltiti memiliki dua kelas, yaitu gambar MRI yang terdiagnosis tumor dan sehat. Sehingga, *loss function* yang digunakan ialah *binary crossentrophy* (6). Selanjutnya, nilai *loss* akan diteruskan ke dalam fungsi *optimizer* sebagai *predictor*. *Optimizer* berperan untuk memprediksi dan mempelajari pola gambar dengan *learning rate* yang telah ditentukan dan nilai *loss* yang memperbaiki bagaimana *optimizer* dalam memprediksi gambar [9]. Pada penelitian ini *optimizer* yang digunakan ialah Adam (7) yang memberikan keluaran berupa probabilitas hasil prediksi.



Gambar 3.5 Hubungan antara bobot, lapisan, *optimizer* dan *loss*

$$-(y\log(p) + (1-y)\log(1-p)) \text{ ……..(6)}$$

Keterangan :

y : indikator biner (0 atau 1)

p : probabilitas prediksi

$$W_{t+1} = W_t - \alpha m_t \text{……..…… (7)}$$

Where,

$$m_t = \beta m_{t-1} + (1-\beta)[\frac{\partial L}{\partial W_t}]$$

Keterangan :

mt = aggregasi gradiens pada watktu t [berjalan] (inisialisasi, mt = 0)

mt-1 = aggregasi gradiens pada watktu t-1 [sebelumnya]

Wt = bobot waktu t

Wt+1 = bobot waktu t+1

αt = *learning rate* pada waktu t berjalan

∂L = fungsi loss

∂Wt = pembagian bobot pada waktu t

β = rata rata parameter bergerak (const, 0.9)

```
# Set training parameters
model.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.Adam(),
              metrics=METRICS)
```

Gambar 3.6 *Loss* dan *optimizer* fungsi yang diusulkan

Proses training akan dilakukan dengan *library keras* dengan mendefinisikan *loss* dan *optimizer* seperti pada gambar 3.6. Lalu, model akan dilatih sebanyak 50 *epoch* dengan *validation data*. *Hyperparameter epoch* berperan dalam menentukan berapa banyak iterasi yang dilakukan dari lapisan pertama ke akhir lalu kembali ke lapisan pertama pada proses pelatihan model.

```
History = model.fit(
      train_generator,
      epochs=EPOCHS,
      steps_per_epoch = STEPS_PER_EPOCH,
      validation_data=validation_generator,
      validation_steps = VALIDATON_STEPS,
      verbose=1)
```

Gambar 3.7 Fungsi *fit* untuk pelatihan model

3.6 *Hyperparameter tuning*

Proses *hyperparameter tuning* atau *hyperparameter optimization* berperan dalam menentukan serangkaian nilai *hyperparameter* yang optimal untuk model. Serangkaian nilai *hyperparameter* tersebut antara lain : jumlah *epoch*, jumlah neuron pada *dense layer* , jumlah filter *convolutional* dan *learning rate* yang tepat bagi model. Dalam menentukan *hyperparameter* yang paling optimal, algoritma *tuning* akan melakukan serangkaian uji coba yang mengkombinasikan beragam nilai pada tiap neuron, *learning rate*, dan *epoch* serta *convolutional filter* satu sama lain yang selanjutnya model tersebut akan disebut sebagai *hypermodel* [9]. Adapun nilai nilai yang akan diuji coba dapat dilihat pada tabel

Tabel 3.3. Serangkaian nilai hyperparameter yang diuji coba

Hyperparameter	*Value*
Learning rate	[1e-2, 1e-3, 1e-4]
Epoch	Maximum 30 *epoch* pada tiap ujicoba
Neuron	minimum 32 dan maksimum 128 dengan kenaikan 32 neuron tiap ujicoba.
Jumlah filter konvolusi	32 – 128

3.7 *Model Evaluation*

Metrik evaluasi yang digunakan untuk mengevaluasi tingkat akurasi model adalah *accuracy*, *sensitivity*, *specificity*, *precision* dan *F1-Score* serta *Dice similarity Coeficient*. Model yang telah dilatih akan memprediksi dataset *testing* untuk mendapatkan nilai metrik final tersebut dari total gambar yang berhasil terprediksi.

```
def Custom_Metrics(tp,tn,fp,fn):
  sensitivity = tp / (tp + fn)
  specificity = tn / (tn + fp)
  precision = tp / (tp + fp)
  recall = tp / (tp + fn)
  DSC = (2*tp) / (fp + (2*tp) + fn)
  F1_score =  (2 * precision * recall) / (precision + recall)
  return sensitivity, specificity, precision, recall, DSC, F1_score
METRICS = [
    tf.keras.metrics.BinaryAccuracy(name="accuracy"),
    tf.keras.metrics.TruePositives(name = "TP"),
    tf.keras.metrics.TrueNegatives(name = "TN"),
    tf.keras.metrics.FalsePositives(name = "FP"),
    tf.keras.metrics.FalseNegatives(name = "FN"),
    tf.keras.metrics.Precision(name="precision"),]
```

Gambar 3.8 *Metric Evaluation Code*

3.8 *Testing / Deployment*

Model yang telah dievaluasi akan diuji coba untuk memprediksi data baru langsung melalui *google colaboratory*.

```
img = load_img(path, target_size=(img_height, img_width))
x = img_to_array(img)
x = np.expand_dims(x, axis=0)

images = np.vstack([x])
prediction = model.predict(images)
if prediction == 0:
  # Show Image
  image = mpimg.imread(path)
  plt.imshow(image)
  plt.title("Healthy")
  print("Healthy")
elif prediction == 1:
  # Show Image
  image = mpimg.imread(path)
  plt.imshow(image)
  plt.title("Tumor")
```

Gambar 3.9 Kode python untuk memprediksi gambar