BAB 3
METODE PENELITIAN

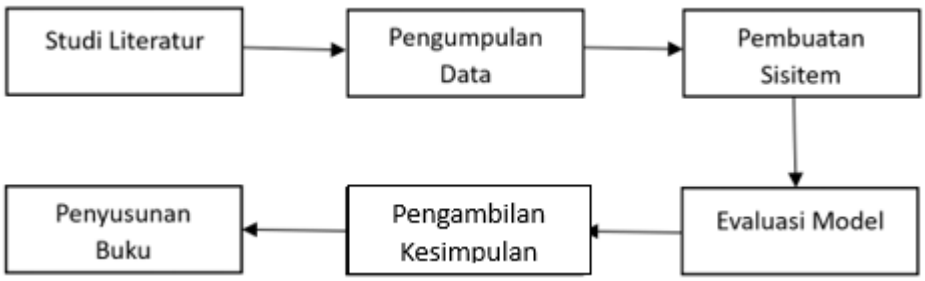
3.1 Alat dan Bahan

Dalam penelitian ini, digunakan perangkat lunak dan perangkat keras untuk membangun sistem. Perangkat lunak yang dimanfaatkan adalah *Google Colab*, sementara perangkat keras yang dipakai adalah laptop Lenovo IdeaPad. Laptop ini menjadi platform utama untuk mengakses dan menjalankan *Google Colab*. Berikut adalah spesifikasi dari perangkat yang digunakan:

1. Software yang digunakan *Google Colab.*
2. Hardware yang digunakan yaitu Laptop ideaPad
   Processor : Processor Intel(R) Core(TM) i3-10110U CPU @ 2.10GHz, 2592 Mhz, 2 Core(s), 4 Logical Processor(s)
   Sistem Operasi : Windows 11 Home Single 64-bit

3.2 Alur Penelitian

Untuk memudahkan pelaksanaan penelitian ini, langkah-langkah yang akan diambil akan dijelaskan secara detail. Alur dari penelitian ini direpresentasikan dalam gambar di bawah ini.



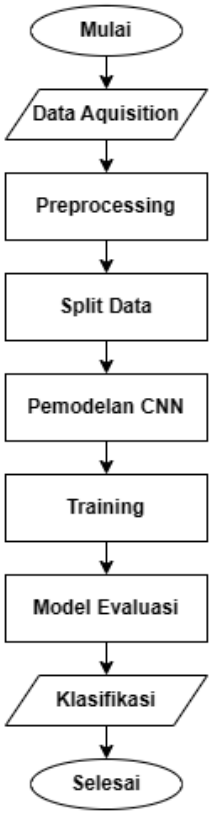
Gambar 3. 1 Alur Penelitian

Melihat dari gambar 3.1, terdapat serangkaian langkah yang harus diambil untuk mencapai hasil penelitian ini. Langkah awal adalah melakukan studi literatur guna memperluas pengetahuan melalui referensi. Selanjutnya, langkah berikutnya adalah mengumpulkan dataset berupa citra sel darah putih. Kemudian, proses pembuatan sistem dilakukan. Pada tahap ini, dilakukan implementasi algoritma *Convolutional*

Neural Network (CNN) untuk membangun struktur pemodelan dengan menggunakan arsitektur *MobileNetV3-Large* dan *EfficientNet-B0,* serta melakukan optimasi *hyperparameter*. Evealuasi model dilakukan sebagai langkah selanjutnya, yang mencakup *Confusion Matrix*, nilai akurasi, nilai presisi, dan *loss*. Serta mengambil kesimpulan tentang performa keseluruhan model.

3.3 Desain Sistem

Dalam perancangan desain sistem ini, dijelaskan secara keseluruhan tentang sistem yang akan dibuat. Pada proyek akhir ini, akan dilakukan perancangan sistem untuk klasifikasi penyakit *Acute Lymphoblastic Leukemia* (ALL) menggunakan *Convolutional Neural Network* (CNN) dengan arsitektur *EfficientNetB0* dan *MobileNetV3Large* untuk mengklasifikasi 4 kelas, yaitu *Benign*, *Early*, (Pre) *Precursor*, dan (Pro) *Progenitor*. Model dari desain sistem dapat dilihat pada Gambar 3.2.



Gambar 3. 2 Diagram Alir Model Sistem

Penelitian dilakukan dalam beberapa tahap yaitu akuisi data, selanjutnya data akan diolah melalui tahap *preprocessing*, tahap selanjutnya adalah *optimizing hyperparameter* dan *traning model*, setelah model dilatih tahap selanjutnya menyimpan model yang telah terlatih melalui *save model* tahap terakhir dari penelitian ini adalah *output image classification*.

3.3.1 Akuisisi Data

Pada tahap akuisisi data, dilakukan pengumpulan gambar atau dataset yang akan digunakan dalam melatih model. Dalam penelitian ini, dataset yang dimanfaatkan diambil dari sumber yang sudah tersedia, yaitu *"Acute Lymphoblastic Leukemia* (ALL) image dataset" yang disediakan oleh Mehrad Aria dan Mustafa Ghaderzadeh di Kaggle pada tahun 2021. Total dataset terdiri dari 3256 data, dengan rincian 504 data termasuk dalam kategori *benign*, 985 data dalam kategori *early*, 963 data dalam kategori *pre*, dan 804 data dalam kategori *pro.* Setelah akuisisi data, langkah selanjutnya adalah membagi dataset menjadi tiga subset yang berbeda, yaitu *training set*, *validation set*, dan *testing set*. Pendekatan yang diterapkan dalam pembagian dataset adalah dengan rasio 80-10-10, di mana 80% dari data digunakan sebagai *training set*, 10% sebagai *validation set*, dan 10% sebagai *testing set* [27]. Pembagian dataset dengan proporsi ini dilakukan untuk meningkatkan efisiensi proses pelatihan dan evaluasi model, serta memastikan bahwa model yang dihasilkan memiliki kemampuan yang baik dalam menggeneralisasi data baru yang belum pernah dilihat sebelumnya [28]. Dalam konteks penelitian ini, *training set* berjumlah 328 data, *validation set* berjumlah 2.604 data, dan *testing set* berjumlah 324 data.

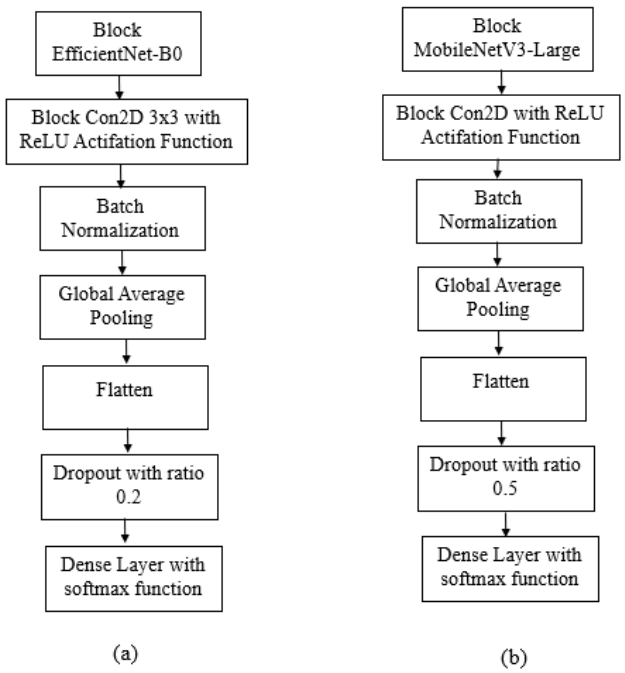
3.3.2 *Pre-processing*

Pada tahap *preprocessing* gambar, dilakukan modifikasi gambar agar model pelatihan dapat bekerja lebih baik Teknik *preprocessing* yang digunakan adalah *rescale*, untuk mengurangi nilai piksel agar lebih mudah diproses oleh model. Langkah-langkah *preprocessing* ini bertujuan memastikan gambar memiliki kualitas yang optimal sehingga model pelatihan mencapai hasil terbaik. Selain itu, juga dilakukan augmentasi pada gambar menggunakan *framework Keras* untuk mengurangi risiko

overfitting. *Overfitting* terjadi ketika model hanya mengenali pola pada data latih, sehingga kesulitan mengklasifikasikan gambar baru. Augmentasi data dilakukan dengan berbagai cara seperti *flipping*, *cropping*, *rotasi*, *brightness*, *contrast*, *colours*, saturasi, translasi, dan lainnya. Dalam penelitian ini, teknik augmentasi yang diterapkan meliputi *cropping*, *rotasi*, *flipping*, dan *shifting*.

3.3.3 Pemodelan CNN

Perancangan model CNN pada penelitian ini digunakan arsitektur *MobileNetV3-Large* dan *EfficientNet-B0*. Metode *transfer learning* kemudian diterapkan pada kedua arsitektur ini untuk melatih model dengan menggunakan parameter atau data baru. Kedua arsitektur tersebut akan mengklasifikasikan gambar menjadi 4 kelas, yaitu *Benign*, *Early*, *Pre* dan *Pro*. Langkah selanjutnya adalah merancang model CNN yang akan digunakan dalam proses pelatihan.



Gambar 3. 3 Usulan model *deep learning*. (a) arsitektur *EfficientNet-B0* (b) arsitektur *MobileNetV3-large*

Usulan model pada *EfficientNet-B0* dan *MobileNetV3-Large* yaitu dengan menambahkan layer setelah blok arsitetur *EfficientNet-B0* dan *MobileNetV3-Large*. Susunan model pada arsitektur *EfficientNet-B0* dapat

dilihat pada Gambar 3.3 kemudian citra hasil keluaran dari blok arsitektur *EfficientNet-B0* dan *MobileNetV3-Large* dilakukan konvolusi dengan filter berukuran 3x3 dengan penambahan fungsi aktivasi ReLU yang bekerja dengan mengubah seluruh input ke dalam nilai positif. Lalu akan diterapkan batch normalization untuk menormalisasi nilai input pda setiap layer *network* dengan tujuan meningkatkan akurasi pelatihan. Kemudian terdapat penambahan *Global Average pooling* dan *Flatten* yang akan mengubah citra dalam bentuk matriks menjadi fully connected layer. Setelah melalui *fully connected layer*, kemudian diberikan *Dense layer* berupa layer output dengan empat node yang jumlahnya menyesuaikan total kelas klasifikasi.

3.3.4 Optimizing *Hyperparameter* and *Training* Model

Tujuan dari penelitian ini adalah untuk mengklasifikasikan kanker sel darah putih dengan menggunakan dua arsitektur CNN, yakni *MobilenetV3-Large* dan *EfficientNet-B0.* Evaluasi dilakukan terhadap berbagai model. Dua jenis *optimzer* yang diuji adalah Adam dan RMSProp, serta dilakukan penelusuran nilai optimal untuk *hyperparameter* lain seperti jumlah *epoch* (iterasi pelatihan), tingkat pembelajaran (*learning rate*), dan ukuran *batch.* Dengan mengoptimalkan *hyperparameter* ini, diharapkan model mampu memberikan hasil yang lebih baik dalam mengklasifikasikan kanker sel darah putih.

3.3.5 Model *Deep Learning*

Setelah melalui proses pelatihan data, hasilnya adalah sebuah model *deep learning* yang mampu mengenali fitur pada citra penyakit ALL. Model ini digunakan untuk mengklasifikasikan citra berdasarkan jenis kelasnya. Berikutnya, model deep learning tersebut akan dinilai untuk mengevaluasi kinerjanya dalam mengklasifikasikan citra. Evaluasi dilakukan dengan memanfaatkan beragam metrik kinerja, termasuk matriks kebingungan (*confusion matrix*), akurasi (*accuracy*), kerugian (*loss*), presisi (*precision*), *recall*, dan F1-Score. *Confusion matrix* digunakan untuk mengevaluasi kualitas prediksi model dengan menghitung jumlah prediksi yang benar. *Accuracy* mengukur seberapa sering model memberikan prediksi yang benar, sementara *loss* mengukur seberapa baik model meminimalkan kesalahan

antara prediksi. *Precision* mengukur seberapa tepat model dalam memprediksi kelas positif, sedangkan *recall* mengukur seberapa baik model dapat menemukan kelas positif. *F1-Score* memberikan nilai yang seimbang tentang performa model secara keseluruhan. Setelah menemukan model dengan *hyperparameter* terbaik, dilakukan evaluasi menggunakan metrik-metrik tersebut untuk menilai performa secara menyeluruh.

3.4 Skenario Pengujian Hyperparameter

Pengujian *hyperparameter* adalah proses mencari nilai terbaik untuk beberapa bagian penting dalam model *deep learning*. Proses ini melibatkan tiga skenario utama: pengujian *optimizer*, pengujian *learning rate*, dan pengujian jumlah *epoch*.

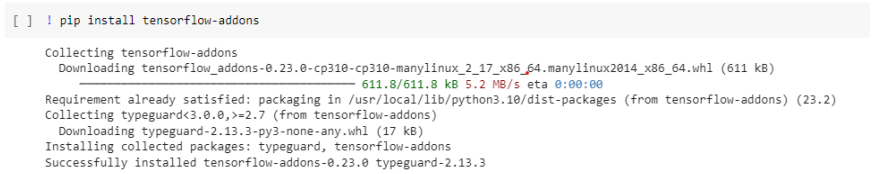
1. Pengujian *Optimizer*: Membandingkan dua jenis *optimizer*, yaitu Adam dan RMSProp, untuk melihat mana yang memberikan hasil terbaik.
2. Pengujian *Learning Rate*: Menguji empat *learning rate* berbeda (0.0001, 0.0003, 0.001, dan 0.003).
3. Pengujian Jumlah *Epoch* : Melakukan pengujian dengan tiga nilai *epoch* yang berbeda (20, 30, dan 50) untuk melihat mana yang menghasilkan model dengan akurasi tinggi dan loss yang rendah.

Tujuan dari pengujian ini adalah untuk menemukan kombinasi *optimizer*, *learning rate*, dan jumlah *epoch* yang dapat menghasilkan *model deep learning* dengan akurasi yang tinggi dan *loss* yang minimal.

3.5 Implementasi CNN menggunakan Python

Bahasa yang digunakan adalah *python.* Pemrograman menggunakan *Google Colab.* Berikut tahapan pemrograman yang dibuat:

- Tahap pertama adalah menginstall *TensorFlow Addons*.



Gambar 3. 4 Potongan code install tensorflow-addons

TensorFlow Addons menyediakan berbagai fungsionalitas tambahan yang sangat membantu dalam mengembangkan model. Di antaranya termasuk optimisasi, lapisan, dan fungsi aktivasi. Dengan menggunakan *TensorFlow Addons*, pengembang model memiliki lebih banyak opsi untuk menyesuaikan dan meningkatkan kinerja model yang akan dibuat. Adapun potongan *code* install *tensorflow* terdapat pada Gambar 3.4.

- Tahap kedua yaitu *import library* agar berbagai fungsi yang diperlukan dapat di akses dan dijalankan. Adapun potongan *code import library* terdapat pada Gambar 3.5.

Gambar 3. 5 Code *Import library*

- Tahap ketiga *connect Google Colab* dengan dataset yang tersimpan di *Google Drive*, yang akan digunakan sebagai lokasi penyimpanan dataset yang akan diakses. Adapun potongan *code connect to google drive* terdapat pada Gambar 3.6.

Gambar 3. 6 Code *connect* ke *google drive*

- Tahap keempat yaitu menentukan lokasi atau *path* dari direktori yang berisi dataset yang akan digunakan. Adapun potongan *code* menentukan *path* direktori terdapat pada Gambar 3.7.

```
TRAINING_DIR = 'train'
TEST_DIR = 'test'
VAL_DIR = 'val'
```

Gambar 3. 7 Code menentukan direktori dataset

Dataset dibagai menjadi tiga bagian yaitu, training, testing dan validasi. Data training digunakan untuk melatih model, data testing diguanakan untuk menguji performa model dan testing digunakan untuk memvalidasi model selama atau setelah proses pelatihan.

- Tahap Kelima yaitu *Augmentasi Data*, untuk meningkatkan variasi data pelatihan sehingga model yang akan dilatih dapat belajar dengan lebih baik dan dapat menggeneralisasi data yang belum pernah dilihat sebelumnya.

```
[ ] BATCH_SIZE = 32
    TARGET_SIZE = (224,224)

    datagen = ImageDataGenerator(
        rotation_range=15,
        horizontal_flip=True,
        width_shift_range=0.1,
        height_shift_range=0.1,
        #shear_range=0.2,
        #zoom_range=0.1,
        # preprocessing_function = myFunc
        # brightness_range=[0.8,0.2]
    )

    traingen = ImageDataGenerator()
```

Gambar 3. 8 Code *augmentasi data*

teknik augmentasi yang diterapkan meliputi *rotation, horozontal flip, width shift* dan *height shift. Rotation* yaitu gambar dapat diputar sebesar maksimum 15 derajat ke kiri atau ke kanan, *horozontal flip* gambar dapat dibalik secara *horizontal, width shift* gambar dapat digeser secara *horizontal* hingga 10% dari lebar gambar dan *height shift* gambar dapat digeser secara vertikal hingga 10% dari tinggi gambar. Adapun ptongan *code augmentasi data* terdapat pada Gambar 3.8.

- Tahap keenam yaitu menampilkan indeks kelas yang sesuai.

  Pada proses pelatihan ini, dataset dibagi menjadi empat kelas: *benign, early, pre, dan pro*. Selain itu, informasi tentang jumlah kelas yang terdapat dalam dataset juga ditampilkan. Adapun potongan *code* menampilkan indeks kelas yang sesuai terdapat pada Gambar 3.9.

```
[ ]  train_ds.class_indices

     {'Benign': 0, 'Early': 1, 'Pre': 2, 'Pro': 3}

[ ]  labels = {0 : 'Benign', 1 : 'Early', 2 : 'Pre', 3 : 'Pro'}

[ ]  class_names = list(train_ds.class_indices)
     class_names

     ['Benign', 'Early', 'Pre', 'Pro']

[ ]  n_class_names = len(class_names)
     n_class_names

     4
```

Gambar 3. 9 Code menampilkan indeks kelas

- Tahap ketujuh pembuatan model.

```
[ ]  efficientnet = EfficientNetB0(input_shape=(224,224,3),
                                   include_top=False,
                                   weights='imagenet')

     Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
     16705208/16705208 [==============================] - 0s 0us/step
```

```
model = Sequential([
    efficientnet,
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(4, activation='softmax')

])
```

Gambar 3. 10 Code pembuatan model *EfficientNetB0*

Untuk code pada Gambar 3.10 hanya menunjukkan arsitektur *EfficientNetB0*. Model *EfficientNetB0* menggunakan parameter seperti *input_shape*=(224, 224, 3) yang menunjukkan ukuran gambar input, *include_top=false* yang menunjukkan bahwa lapisan klasifikasi tidak akan disertakan dan *weights=imagenet* yang menunjukkan bahwa bobot pra-dilatih dari model yang digunakan.

- Tahap kedelapan adalah menentukan *learning rate*, jenis *optimizer* dan jumlah *epoch* yang digunakan dalam proses pelatihan model. Adapun potongan *code* untuk menentukan *learning rate*, *optimizer* dan jumlah *epoch* terdapat pada Gambar 3.11.

```
[ ]  model.compile(optimizer=optimizer.Adam(learning_rate=0.0001),
                   loss='categorical_crossentropy',
                   metrics=['acc', tf.keras.metrics.Precision(), tf.keras.metrics.Recall(), tfa.metrics.F1Score(num_classes=4)])
```

```
history = model.fit(
    train_ds,
    epochs = 30,
    batch_size = BATCH_SIZE,
    validation_data = val_ds,
    verbose = 1
)
```

Gambar 3. 11 code menentukan *learning rate*, *opimizer* dan jumah *epoch*

- Tahap kesembilan adalah menampilkan grafik perkembangan metrik pelatihan dan validasi selama proses pelathan model. Adapun potongan *code* untuk menampilkan kurva terdapat pada Gambar 3.12.

```
plt.plot(history.history["acc"],label="Akurasi Pelatihan")
plt.plot(history.history["val_acc"],label="Validasi Akurasi")
plt.legend()
plt.show()

plt.plot(history.history["loss"],label="Kesalahan Pelatihan")
plt.plot(history.history["val_loss"],label="Validasi Kesalahan")
plt.legend()
plt.show()
```

Gambar 3. 12 Menampilkan kurva

Grafik pertama menunjukkan perubahan akurasi pada setiap *epoch* selama pelatihan (akurasi pelatihan) dan pada setipa *epoch* selama validasi (validasi akurasi).

Grafik kedua menunjukkan peruahan nilai fungsi kerugian *(loss)* pada setiap *epoch* selama pelatihan (kesalahan pelatihan) dan pada setiap *epoch* selama validasi (validasi kesalahan).

```
[ ] scores = model.evaluate(train_ds)
82/82 [==============================] - 223s 3s/step - loss: 0.0023 - acc: 1.0000 - precision: 1.0000 - recall: 1.0000 - f1_score: 1.0000

[ ] scores = model.evaluate(val_ds)
11/11 [==============================] - 32s 3s/step - loss: 0.1000 - acc: 0.9784 - precision: 0.9783 - recall: 0.9753 - f1_score: 0.9747

[ ] scores = model.evaluate(test_ds)
11/11 [==============================] - 76s 7s/step - loss: 0.0681 - acc: 0.9848 - precision: 0.9848 - recall: 0.9848 - f1_score: 0.9833
```

Gambar 3. 13 code menampilkan tingkat akurasi

Pada Gambar 3.13 menampilkan performa model berdasarkan tingkat akurasi dalam melakukan prediksi terhadap data yang diberikan, yaitu pada data pelatihan *(train_ds)*, data validasi *(val_ds)*, dan data pengujian *(test_ds)*.

- Tahap kesepuluh adalah menampilkan *confusion matrix*

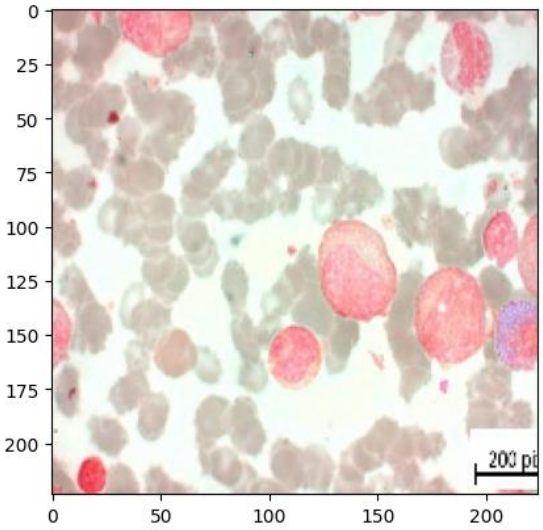
  Pada Gambar 3.14 merupakan code untuk melakukan evaluasi menggunakan *confusion matrix* terhadap model yang telah dilatih. *Confusion matrix* dibuat berdasarkan hasil prediksi untuk mengevaluasi seberapa baik model mengklasifikasikan data berdasarkan kelasnya.

```
predictions = model.predict_generator(test_ds, num_of_test_samples // BATCH_SIZE+1)
y_pred = np.argmax(predictions, axis=1)
true_classes = test_ds.classes
class_labels = list(test_ds.class_indices.keys())
cm = metrics.confusion_matrix(test_ds.classes, y_pred)
x_axis_labels = ['Benign','Early','Pre','Pro']
y_axis_labels = ['Benign','Early','Pre','Pro']
sns.heatmap(cm, xticklabels=x_axis_labels, yticklabels=y_axis_labels, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion matrix')
plt.tight_layout()
plt.show()
print('Classification Report')
print(classification_report(true_classes, y_pred, target_names=class_labels))
```

Gambar 3. 14 Code menampilkan *confusion matrix*

Confusion matrix tersebut ditampilkan dalam bentuk *heatmap* dengan *seaborn*. *Classification report* digunakan untuk mengevaluasi lebih lanjut kinerja model dengan menyajikan *precision, recall, f1-score*, dan *support* untuk setiap kelas dalam bentuk teks yang rinci.

- Tahap kesebelas Melakukan pengujian *(testing)* terhadap model menggunakan gambar yang telah dibaca sebelumnya dan kemudian memperoleh hasil prediksi dari model tersebut.



Gambar 3. 15 Memasukkan sel citra ALL kelas *Benign*

```
[ ]  output = model.predict(image_testing, 1)
     print(output)

     1/1 [==============================] - 1s 1s/step
     [[9.9999774e-01 1.9377640e-06 2.0982668e-07 6.4077184e-08]]
```

Gambar 3. 16 Hasil klasifikasi citra dengan menggunakan arsitektur *EfficientNet-B0*

Gambar 3.15 merupakan citra sel *acute limpoblastic leukimia* kelas *benign*, dan Gambar 3.16 merupakan hasil klasifikasi dari masukan citra tersebut

yang menunjukkan terdapat 4 nilai *array* yang mewakili jumlah kelas klasifikasi dengan urutan kelas yaitu *benign, early, pre* dan *pro.* Dapat dilihat 9.9% model telah memprediksi dengan benar bahwa masukan citra tersebut merupakan *acute limpoblastic leukimia* kelas *benign*