Klasifikasi Sel Darah Putih Menggunakan Vision Transformer (ViT)

Jeremia Daud Halim[#1], Risal[*2]

[#]*Program Studi Teknik Informatika Fakultas Teknologi dan Rekayasa Cerdas, Universitas Kristen Maranatha*
Jalan Prof. Drg. Surya Sumantri No.65, Bandung 40164
[1]2072021@maranatha.ac.id
[2]laurentius.risal@it.maranatha.edu

Abstract — **White blood cells are important for humans to protect the immune system from bacteria or viruses. White blood cells have 5 types of blood cells which consist of eosinophils, lymphocytes, monocytes, neutrophils, and basophils. Each type has a different shape. The classification of white blood cells is important in the field of healthcare to diagnose diseases caused by white blood cells, such as anemia, leukemia, and others. The classification process is usually done manually using a hemocytometer, the process is prone to human error which can lead to errors in diseases diagnosis. Advancement in technology, especially deep learning, can provide the potential to facilitate and minimize human error in the classification process. This report describes the application of a Vision Transformer (ViT) with a high-performance model in classifying white blood cells based on images. The ViT model uses multi-head attention to process information from images at the same time. The model training process uses a dataset containing 12,500 images and 4 types of white blood cells (eosinophils, lymphocytes, monocytes, neutrophils) with each having about 3,000 images. The model that has been trained using the right combination of hyperparameter values gets a train accuracy result of 98% and validation accuracy result of 83.44%. The model was used on a simple website as a test platform for classification. The results show that the model can classify white blood cells based on images correctly and can be a potential for the medical field.**

Keywords— **Classification, Deep Learning, Vision Transformer, White Blood Cells.**

I. PENDAHULUAN

Sel darah putih merupakan bagian penting dalam sistem kekebalan tubuh manusia, berfungsi untuk melindungi tubuh dari infeksi, penyakit, dan benda yang dianggap asing oleh tubuh. Sel darah putih terbagi menjadi beberapa jenis diantaranya adalah basofil, neutrofil, eosinofil, monosit, dan limfosit [1]. Jenis tersebut dibedakan berdasarkan struktur sel dan jumlah lobus pada nukleus. Jumlah masing-masing jenis sel darah putih penting bagi dokter untuk mendiagnosa suatu penyakit seperti leukemia, anemia, dan lainnya. Dokter akan mendiagnosa penyakit dengan menganalisa jumlah masing-masing sel darah putih.

Proses klasifikasi sel darah putih umumnya dilakukan secara manual oleh dokter atau ahli laboratorium dalam rentang waktu 1x24 jam. Proses tersebut menggunakan alat hemocytometer, sel darah putih seorang pasien dilihat melalui mikroskop dengan menghitung masing-masing jumlah sel darah putih. Terdapat beberapa kekurangan dalam penghitungan secara manual menggunakan hemocytometer seperti sampel darah yang tumpang tindih membuat proses penghitungan menjadi lebih sulit, dan berdasarkan pengamatan visual manusia juga dinilai cukup sulit untuk mendapatkan hasil yang akurat [4], menghabiskan banyak waktu dan melelahkan, apabila proses terganggu maka harus diulang kembali dari awal, tetapi bagi seorang yang kurang pengalaman, maka harus dilakukan pemeriksaan secara berulang kali untuk memastikan hasil penghitungan yang akurat [5]. Maka dari itu, proses penghitungan secara manual rentan terhadap kesalahan manusia yang dapat mengakibatkan kesalahan dalam menghasilkan analisa maupun diagnosa penyakit.

Dengan kemajuan teknologi khususnya *deep learning*, tentu bisa menjadi salah satu solusi dan potensi untuk mempermudah proses klasifikasi atau penghitungan dibandingkan menggunakan metode secara manual. Dengan metode yang tepat untuk melatih model *deep learning* agar mendapatkan kinerja atau performa yang baik, maka proses klasifikasi dapat dilakukan dengan hasil yang tepat dan dapat mengurangi kesalahan manusia sehingga dapat menghasilkan analisa atau diagnosa yang lebih tepat.

Masalah utama pada penelitian ini berdasarkan latar belakang adalah bagaimana cara menerapkan Vision Transformer untuk mengklasifikasikan sel darah putih berdasarkan gambar, serta bagaimana cara agar mendapatkan kinerja atau performa yang tinggi pada model yang digunakan agar dapat mengklasifikasikan dengan tepat. Tujuan penelitian ini untuk

membuat model Vision Transformer untuk mengklasifikasi jenis sel darah putih berdasarkan gambar dan mendapatkan kinerja atau performa yang tinggi pada model tersebut agar hasil klasifikasi tepat.

II. KAJIAN TEORI

A. *Deep Learning*

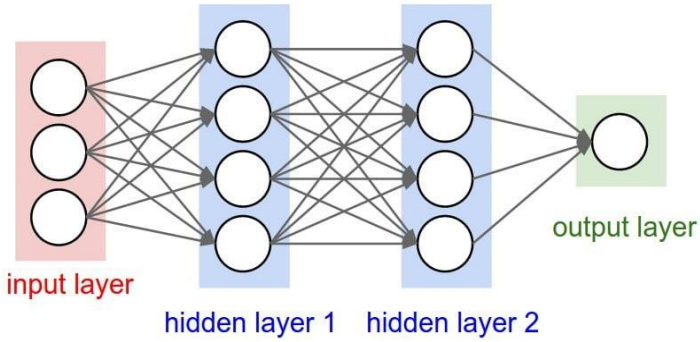
Deep learning merupakan subbidang khusus dari machine learning, sebuah cara baru dalam mempelajari data yang menekankan pada pembelajaran lapisan demi lapisan [6]. Deep learning mencoba untuk mempelajari abstraksi tingkat tinggi dalam data dengan memanfaatkan struktur arsitektur [7]. Kata "deep" dalam "deep learning" mengarah kepada jumlah *layers* atau lapisan yang digunakan [6]. Sederhananya, deep learning adalah sebuah metode dalam AI untuk mempelajari data dengan cara yang menyerupai jaringan saraf otak manusia. *Deep learning* dapat dipahami sebagai metode untuk meningkatkan hasil dan mengoptimasi waktu daripada pemrosesan dalam beberapa proses komputasi [8].

Deep learning merupakan pembelajaran mesin yang memungkinkan komputer untuk belajar dari pengalaman dan memahami dalam konsep secara hirarki [9]. Komputer atau mesin akan mengumpulkan pengetahuan yang dibutuhkan atau yang diinginkan berdasarkan pengalaman pembelajarannya sendiri. *Deep learning* melibatkan analisis *layers* dan metode dalam mengaplikasikannya [8].

Dalam *deep learning* terdapat beberapa metode dan arsitektur yang menggunakan *layers* dalam prosesnya yang dapat digunakan untuk kebutuhan dan tujuan tertentu diantaranya adalah CNN atau *Convolutional Neural Networks*, RNN atau *Recurrent Neural Networks*, LSTM atau *Long Short-Term Memory*, NLP atau *Natural Language Processing*, dan lain-lain [10].

B. *Multi-Layer Neural Networks*

Secara umum, arsitektur *deep learning* didasari oleh *deep neural networks* atau biasa dikenal dengan *multi-layer neural networks* [11]. Ilmuwan *neural network* dari Amerika Bernama Robert Hecht-Nielsen mendefinisikan *neural network* sebagai sistem komputer yang terdiri dari elemen-elemen pemrosesan yang sederhana dan saling berhubungan, yang memroses informasi dengan respons dinamis terhadap input [11]. Secara sederhana, *neural network* adalah sebuah sistem yang dibuat mendekati struktur otak manusia.



Gambar 1. Deep Neural Network [12]

Gambar 1 merupakan ilustrasi *deep neural networks*, terdapat istilah *multi-layer* yang mengacu kepada jumlah *hidden layer*. *Hidden layer* adalah kumpulan neuron yang mengirim data dan *training layer to layers* dan lapisan perantara antara *input layer* dan *output layer* [13]. *Input layer* berfungsi sebagai penerima data dan mengirimkannya ke *hidden layers* untuk diproses, setelah data diproses di dalam *hidden layers* maka data dikirimkan ke *output layer* sebagai hasil pemrosesan. Pemilihan jumlah *hidden layers* tidak bisa sembarangan, karena dapat menyebabkan model menjadi *overfitting* atau *underfitting* yang memengaruhi efektivitas, efisiensi, akurasi, maupun prediksi model dalam generalisasi data baru.

Overfitting adalah kondisi ketika jumlah *hidden layers* yang terlalu besar atau berlebihan dibandingkan tingkat kompleksitas masalah yang ada, sehingga menyebabkan proses *training* pada model menjadi sangat baik atau berlebihan, tetapi kinerja pada saat pengujian data dan uji coba menurun apabila model dimasukkan data baru yang sebelumnya belum pernah dikenal oleh model. Sebaliknya, *underfitting* adalah kondisi ketika jumlah *hidden layers* yang terlalu kecil atau kurang dibandingkan tingkat kompleksitas masalah yang ada, sehingga menyebabkan proses *training* pada model tidak bisa memroses data dengan maksimal, dan membuat kinerja buruk pada model pada saat pengujian data dilakukan [13].

C. TensorFlow dan Keras

TensorFlow merupakan Python-*based, free,* dan *open-source framework machine learning* yang dikembangkan oleh Google. Saat ini TensorFlow menjadi salah satu *framework* yang populer digunakan khususnya pada bidang pengembangan AI. TensorFlow memiliki cara unik dalam memecahkan suatu masalah, dimana cara tersebut dapat membantu kita dalam memecahkan masalah pada deep learning dengan sangat efisien [6] [14].

TensorFlow digunakan untuk membangun, melatih, dan menerapkan berbagai jenis model seperti *neural network* untuk klasifikasi gambar, deteksi objek, pemrosesan bahasa alami, dan lainnya [15]. TensorFlow mampu untuk melatih dan menjalankan *neural networks* yang sangat besar secara efisien dengan cara mendistribusikan komputasi ke ratusan server multi-GPU (Graphics Processing Unit) [15]. TensorFlow memiliki banyak fitur diantaranya adalah Keras [15].

Keras adalah API *deep learning* untuk Python yang dibangun di atas platform TensorFlow, yang menyediakan cara mudah untuk mendefinisikan dan melatih segala jenis model *deep learning* [6]. Melalui TensorFlow, Keras dapat dijalankan di berbagai jenis perangkat keras seperti GPU, TPU, atau CPU biasa [6]. TensorFlow sebagai *low-level tensor computing* dan Keras sebagai *high-level deep learning* API [6]. Dengan menggunakan Keras, kita dapat dengan fleksibel untuk membangun arsitektur *neural networks* yang luas [15].

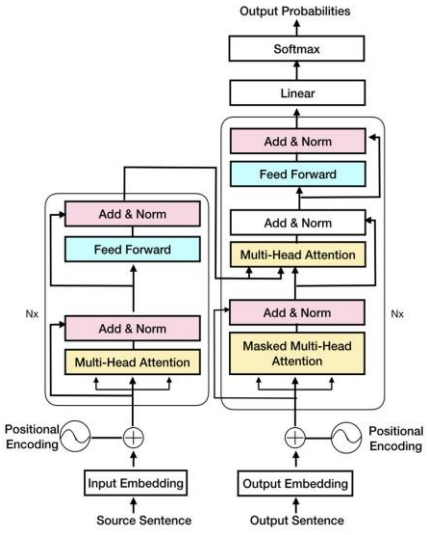
D. Layers

Keras memiliki berbagai macam *library* yang dapat digunakan, salah satunya adalah *library* "layers". Jenis *layers* yang berbeda disesuaikan untuk format tensor yang berbeda dan jenis pemrosesan data yang berbeda [6]. *Library* "layers" dalam Keras memiliki berbagai jenis diantaranya ada *Dense layers, Convolution layers, Pooling layers, Recurrent layers*, dan lain-lain.

Sebagai contoh, data vektor yang disimpan dalam tensor rank-2 seringkali diproses oleh *dense layers*, data *sequence* disimpan dalam tensor rank-3 seringkali diproses oleh *recurrent layers* atau 1D *Convolution layers* (Conv1D), dan untuk data berupa gambar yang disimpan pada tensor rank-4 biasanya diproses oleh 2D *Convolution layers* (Conv2D) [6]. Masing-masing "layers" dalam *library* Keras memiliki kegunaan yang berbeda sesuai dengan kebutuhan. Untuk *layers* yang digunakan dalam model ViT berbeda dengan metode *deep learning* lainnya, layers yang digunakan adalah *attention layers*.

E. Vision Transformer

ViT atau Vision Transformer adalah salah satu model untuk klasifikasi gambar yang menggunakan arsitektur Transformer melalui potongan-potongan gambar. Arsitektur pada Transformer model terbagi menjadi dua yaitu encoder dan decoder.

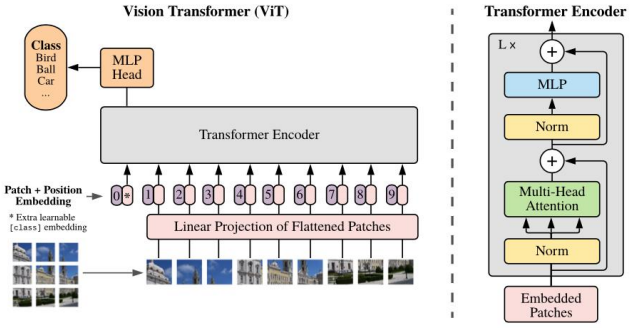


Gambar 2. Transformer Encoder dan Decoder

Dalam Gambar 2 encoder di sebelah kiri, berfungsi untuk memetakan input menjadi representasi yang berguna untuk tugas apapun yang sedang dilakukan, decoder di sebelah kanan, berfungsi untuk memecahkan representasi dari encoder dan menggabungkannya dengan input lainnya untuk membuat sebuah rangkaian prediksi [17]. Terdapat tiga varian yang populer yaitu *encoder-only*, *decoder-only*, dan *encoder-decoder*.

F. Model Arsitektur

Dalam Gambar 3 arsitektur ViT biasanya menggunakan encoder saja, encoder dalam ViT mengonversi gambar menjadi representasi fitur yang kemudian dapat diproses lebih lanjut untuk tugas-tugas *computer vision* seperti klasifikasi gambar [18]. Sederhananya cara kerja model adalah gambar yang utuh dibagi menjadi beberapa bagian potongan (*patch*) dengan ukuran yang sama, kemudian memasukkan masing-masing bagian secara linear, menambahkan posisi penyisipan, memasukkan urutan vektor yang dihasilkan ke encoder Transformer, melatih ViT model dengan label gambar, dan menyempurnakan dataset untuk klasifikasi gambar.

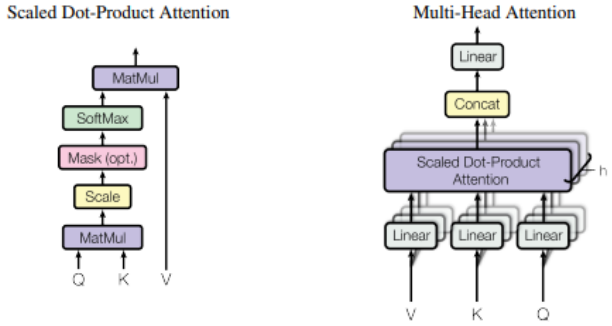


Gambar 3. Ilustrasi ViT Model

Output akhir dari arsitektur ViT berupa prediksi kelas yang diperoleh dengan melewati output dari blok transformer terakhir melalui klasifikasi utama.

G. Attention Layer

Layer yang digunakan dalam model ViT berbeda dengan *layer* pada pendekatan *deep learning* lainnya, *layer* dalam ViT disebut dengan *attention layer* yang dapat menyoroti bagian penting dari input. Fungsi attention dapat dideskripsikan sebagai pemetaan *query* dan satu set kombinasi *key-value* ke sebuah *output*, di mana *query*, *value*, dan *output* adalah vektor [17]. Terdapat dua jenis yang umum diketahui yaitu, scaled dot-product attention dan multi-head attention. Transformer encoder terdiri dari *layer* alternatif yang terdiri dari multihead self-attention.



Gambar 4 Scaled Dot-Product Attention dan Multi-Head Attention [17]

Self-attention memungkinkan ViT untuk mengintegrasikan informasi pada keseluruhan gambar bahkan pada *layers* paling dasar [18]. Attention menggunakan semua *patch* gambar untuk mengetahui gambar keseluruhannya dengan cara komunikasi antar *patch*. Dengan memproyeksikan setiap *patch* secara terpisah untuk menghasilkan *query*, *keys*, *values*. Input terdiri dari dimensi d_k dan dari dimensi d_v. Dimensi d_k adalah dimensi yang terdiri dari *query* dan *keys* sedangkan dimensi d_v adalah dimensi yang terdiri dari *values* [17].

H. Multi-Head Attention Layer

Multi-head attention layer pada ViT merupakan salah satu *layer* yang dapat digunakan untuk memproses informasi dari gambar. Multi-head attention terdiri dari beberapa *layer* attention yang dijalankan secara parallel [17]. Multi-head attention memungkinkan model untuk menangani informasi secara bersamaan dari representasi dan posisi yang berbeda [17]. Sederhananya, multi-head attention berfungsi agar *patch* gambar yang ada dapat berkomunikasi dengan mengirimkan informasi ganda dengan menggunakan proses *looping* yang memanfaatkan scale dot-product attention, *query*, *keys*, dan *value* yang dapat dieksekusi secara parallel.

I. Studi Literatur Penggunaan Deep Learning

Tabel 1 merupakan perbandingan hasil akurasi terbaik yang didapat oleh Yildirim dan Çinar [19], Hüseyin Kutlu et al. [20], César Cheuque et al. [21], Oguzhan Katar dan Ozal Yildirim [22], Shuwen Chen et al. [23], dan Shakib Mahmud Dipto et al. [24] dalam melakukan klasifikasi sel darah putih menggunakan CNN dan ViT.

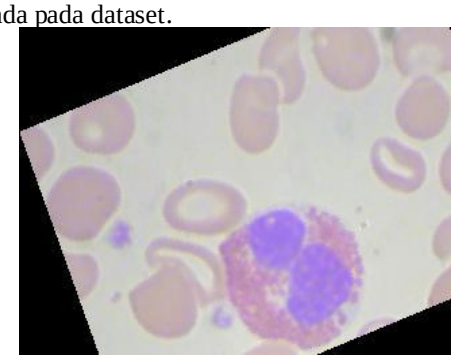
TABEL I
PERBANDINGAN HASIL STUDI LITERATUR

Studi Literatur	Metode	Hasil/Performa
Yildirim dan Çinar [19]	CNN DenseNet201	83.44%
Hüseyin Kutlu et al. [20]	CNN AlexNet	97%
César Cheuque et al. [21]	CNN MobileNet	98.36%
Oguzhan Katar dan Ozal Yildirim [22]	ViT pre-trained ImageNet-21k	99.27%
Shuwen Chen et al. [23]	ViT pre-trained Shifted Window	98.03%
Shakib Mahmud Dipto et al. [24]	ViT	84%

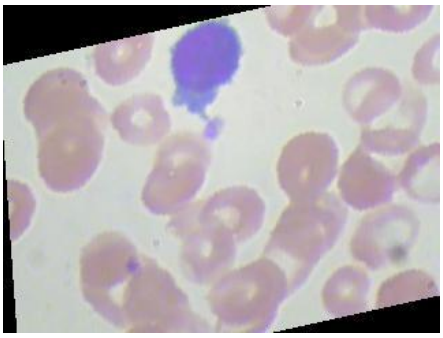
III. ANALISIS DAN RANCANGAN SISTEM

A. Dataset

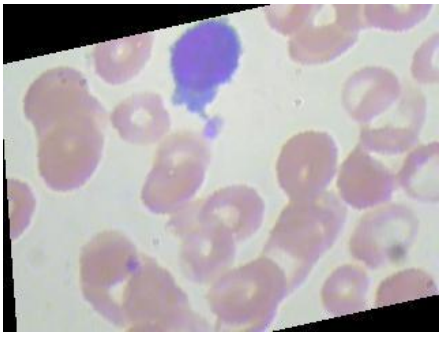
Dataset yang digunakan dalam proyek ini diambil dari Kaggle dengan nama "Blood Cell Images" yang terdiri dari 12.500 gambar sel darah putih yang diperbesar dan kurang lebih sekitar 3.000 gambar untuk masing-masing 4 jenis sel darah putih, semua gambar dalam dataset menggunakan ekstensi file JPEG dan disertai dengan ekstensi file CSV berisi label masing-masing jenis sel. Terdapat 4 jenis sel darah putih dalam dataset yang digunakan, yaitu eosinofil, limfosit, monosit, dan neutrofil. Gambar 5, Gambar 6, Gambar 7, dan Gambar 8 merupakan salah satu contoh dari masing-masing jenis sel yang ada pada dataset.



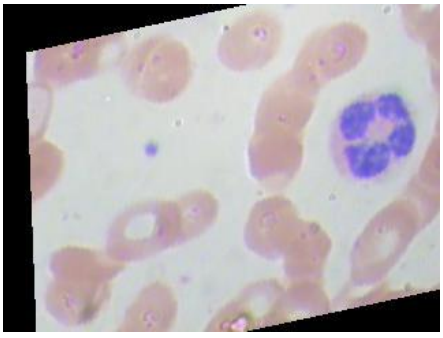
Gambar 5. Eosinofil



Gambar 6. Limfosit



Gambar 7. Monosit



Gambar 8. Neutrofil

B. Prepare Data

Data dalam dataset dipisahkan untuk *train, test, dan validation*. Memastikan seluruh gambar memiliki resolusi yang sama. Kemudian dataset dibersihkan dari data atau file yang rusak ataupun nilai yang hilang, sehingga pada saat proses *training* akan mendapatkan hasil yang maksimal karena tidak ada file yang rusak ataupun nilai yang hilang. Ukuran resolusi yang

digunakan dalam dataset adalah 240 x 320 x 3, dimensi pertama, yaitu 240 x 320 adalah resolusi daripada gambar dan dimensi kedua menjelaskan bahwa gambar menggunakan format warna RGB (Red, Green, Blue).

C. Rancangan Model

Model yang digunakan dalam proyek ini adalah Vision Transformer (ViT) model. Pertama akan dilakukan hyperparameters tuning menggunakan GridSearch tuner, dimana GridSearch tuner akan melakukan pencarian kombinasi nilai yang terbaik untuk digunakan pada saat proses pelatihan, seperti learning_rate, weight_decay, dan lain-lain, yang akan mempengaruhi kinerja atau performa model dalam proses pelatihan.

TABEL III
HYPERPARAMETER TUNING

Hyperparameter Tuning	Nilai
Learning rate	[0.0001, 0.001, 0.01, 0.1]
Weight decay	[0.0001, 0.001, 0.01, 0.1]
Projection dimension	[64, 128, 256]
Number of heads	[4, 6, 8]
Transformer layer	[8, 10, 12, 14]

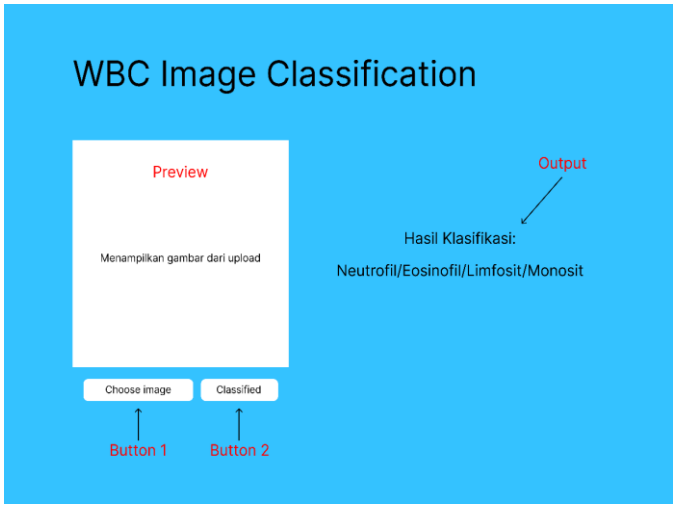
Nilai-nilai hyperparameter tersebut dipilih berdasarkan referensi daripada studi yang dilakukan oleh Oguzhan Katar dan Ozal Yildirim [22], referensi model Keras, dan referensi yang berada di internet, dimana studi Oguzhan Katar dan Ozal Yildirim menghasilkan nilai akurasi sebesar 99.27%, sedangkan referensi model Keras dan referensi di internet menghasilkan nilai akurasi diantara 93%-98%, oleh karena itu nilai tersebut coba diterapkan pada model ViT yang akan digunakan.

Melakukan augmentasi data untuk membuat variasi data untuk mencegah *overfitting* dengan melakukan perubahan kecil pada gambar asli, seperti membalikkan gambar, memutar gambar, dan memperbesar atau memperkecil gambar. Dengan dilakukannya augmentasi data, maka model dapat lebih maksimal dalam melakukan proses *training*. Dilakukan juga proses normalisasi dalam rancangan bertujuan untuk melakukan normalisasi pada setiap pixel dalam gambar.

Kemudian setiap gambar akan dilakukan proses pemecahan atau *patch* yang nantinya akan menjadi potongan-potongan gambar. Dari potongan-potongan gambar tersebut kemudian akan dikimkan ke encoder untuk diproses kembali yang semula potongan gambar menjadi sebuah vektor. Selanjutnya akan dilakukan pembuatan ViT model yang akan menggunakan input, data augmentasi, potongan gambar, dan potongan gambar yang sudah di encode.

Selanjutnya, setelah model dilatih dan divalidasi, maka akan dilakukan proses evaluasi terhadap model, dengan tujuan untuk menilai kinerja model yang telah dilatih terhadap data *test*. Dengan dilakukannya evaluasi model, maka dapat diketahui gambaran mengenai seberapa akurat atau seberapa baik model dalam melakukan klasifikasi pada data baru.

D. Desain Website Sederhana



Gambar 9. Desain Website

Website ini akan berfungsi sebagai wadah uji coba terhadap model yang sudah dilatih. *Website* ini akan memfasilitasi pengguna untuk dapat mengunggah gambar sel darah putih, kemudian gambar dari pengguna akan diproses oleh model ViT yang sudah dilatih dan akan menghasilkan *output* berupa teks klasifikasi dan probabilitas klasifikasi daripada gambar

tersebut. Gambar 9 merupakan desain *website* yang dibuat dengan antarmuka yang sederhana sehingga memudahkan pengguna dalam menggunakannya.

Desain *website* akan berisi judul, *preview* gambar dari unggahan pengguna, *button 1* yang berfungsi untuk mengunggah gambar, *button 2* untuk melakukan proses klasifikasi oleh model, dan *output* hasil klasifikasi dan probabilitas berupa teks.

IV. IMPLEMENTASI

A. *Pengambilan Data*

Dataset diambil dari *website* Kaggle dengan nama *Blood Cell Images* oleh Paul Mooney. Dataset terbagi menjadi dua folder, yaitu untuk *train* dan *test*, masing-masing folder berisi 4 kelas (limfosit, eosinofil, monosit, dan neutrofil). Untuk folder *train*, limfosit berisi 2483 gambar, eosinofil berisi 2497 gambar, monosit berisi 2478 gambar, dan neutrofil berisi 2499 gambar. Sedangkan folder *test*, limfosit berisi 620 gambar, eosinofi berisi 623 gambar, monosit berisi 620 gambar, dan neutrofil berisi 624 gambar. Dataset tersebut sudah bersih dari file yang rusak atau hilang, sehingga tidak memerlukan lagi tahap pembersihan dataset. Ukuran resolusi daripada setiap gambar adalah 240x320x3.

B. *Persiapan Data*

Setelah data berhasil diakses dari Google Drive, selanjutnya pada Gambar 10 dilakukan *load* data dari train_dir, test_dir, dan val_dir menggunakan utilitas dari TensorFlow, yaitu tf.keras.utils.image_dataset_from_directory().

```
train_ds = tf.keras.utils.image_dataset_from_directory(
  train_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size
)

test_ds = tf.keras.utils.image_dataset_from_directory(
  test_dir,
  validation_split=0.1,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    val_dir,
    validation_split=0.1,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)
```

Gambar 10. Pengambilan Data

Karena data tersebut berupa *prefetch* maka akan dikonversi terlebih dahulu menjadi array NumPy dengan langkah berikut, yaitu menggunakan as_numpy_iterator() yang menghasilkan iterator yang memuat batch data secara bertahap dan mengkonversinya menjadi array NumPy pada Gambar 11.

```
train_list = train_ds.as_numpy_iterator()
test_list = test_ds.as_numpy_iterator()
val_list = val_ds.as_numpy_iterator()
```

Gambar 11. Array NumPy

Kemudian pada Gambar 12 setiap batch yang ada pada train_list, test_list, dan val_list diproses menggunakan *for loop* untuk mengambil sampel acak sesuai dengan proporsi yang sudah ditentukan, yaitu 85% dari setiap batch.

```
train_x = []
train_y = []

for i, element in enumerate(train_list):
  sample_list = np.random.choice(len(element[0]), int(np.ceil(0.85 * len(element[0]))))
  for index in sample_list:
    train_x.append( element[0][index] )
    train_y.append( element[1][index] )

test_x = []
test_y = []

for i, element in enumerate(test_list):
  sample_list = np.random.choice(len(element[0]), int(np.ceil(0.85 * len(element[0]))))
  for index in sample_list:
    test_x.append( element[0][index] )
    test_y.append( element[1][index] )

val_x = []
val_y = []

for i, element in enumerate(val_list):
  sample_list = np.random.choice(len(element[0]), int(np.ceil(0.85 * len(element[0]))))
  for index in sample_list:
    val_x.append( element[0][index] )
    val_y.append( element[1][index] )
```

Gambar 12. Data Random Sampling

Gambar 13 merupakan bentuk data setelah pengambilan sampel acak.

```
x_train shape: (6971, 240, 320, 3) - y_train shape: (6971, 1)
x_test shape: (435, 240, 320, 3) - y_test shape: (435, 1)
x_val shape: (12, 240, 320, 3) - y_val shape: (12, 1)
```

Gambar 13. Bentuk Data

C. Hyperparameter Tuning

Berikut adalah beberapa tahap proses hyperparameter tuning:

```
learning_rate = hp.Choice("learning_rate", values=[0.0001, 0.001, 0.01, 0.1])
weight_decay = hp.Choice("weight_decay", values=[0.0001, 0.001, 0.01, 0.1])
projection_dim = hp.Choice("projection_dim", values=[64, 128, 256])
num_heads = hp.Choice("num_heads", values=[4, 6, 8])
transformer_layers = hp.Int("transformer_layers", min_value=8, max_value=14, step=2)
```

Gambar 14. Hyperparameter Tuning

```
tuner = kt.GridSearch(
    hypermodel=hypermodel,
    objective="val_accuracy",
    max_trials=10,
    seed=42,
    executions_per_trial=1,
    directory='grid_search_1',
    project_name='vision_transformer_1'
)
```

Gambar 15.GridSearch Tuner

```
tuner.search(X_train, y_train, epochs=40, validation_data=(X_val, y_val), callbacks=[early_stopping])
```

Gambar 16. Pencarian Hyperparameter Terbaik

Setelah melakukan hyperparameter tuning menggunakan GridSearch tuner, didapati kombinasi nilai hyperparameter terbaik dengan nilai akurasi sebesar 91.66%.

TABEL IIII
HYPERPARAMETER TERBAIK

Hyperparameter	Nilai
Learning rate	0.001
Weight decay	0.001
Projection dimension	128
Num of heads	6
Transformer layer	12

Gambar 17 merupakan parameter yang digunakan dalam proses *train*.

```
input_shape = [240, 320, 3]
batch_size = 256
num_classes = 4
num_epochs = 100
image_size= 720
patch_size = 60
num_patches = (image_size // patch_size) ** 2
transformer_units = [
    projection_dim * 2,
    projection_dim
]
mlp_head_units = [2048, 1024]
```

Gambar 17 Parameter Train

D. Augmentasi Data

Gambar 18 merupakan implementasi dalam melakukan augmentasi data.

```
augmentation_layer = keras.Sequential(
  [
      layers.Input(input_shape),
      layers.Normalization(),
      layers.Resizing(image_size, image_size),
      layers.RandomFlip("horizontal"),
      layers.RandomRotation(factor=0.02),
      layers.RandomZoom(height_factor=0.2, width_factor=0.2),
  ],
  name="augmentation_layer",
)

augmentation_layer.layers[0].adapt(X_train)
```

Gambar 18. Augmentasi Data

E. Fungsi MLP

Gambar 19 merupakan implementasi membuat fungsi MLP *custom* untuk mendefinisikan struktur lapisan MLP dengan jumlah unit dan *dropout* yang berbeda.

```
def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units, activation=keras.activations.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x
```

Gambar 19. Fungsi MLP

F. Proses Patch

Gambar 20 merupakan implementasi dalam memecah gambar menjadi potongan-potongan (*patch*).

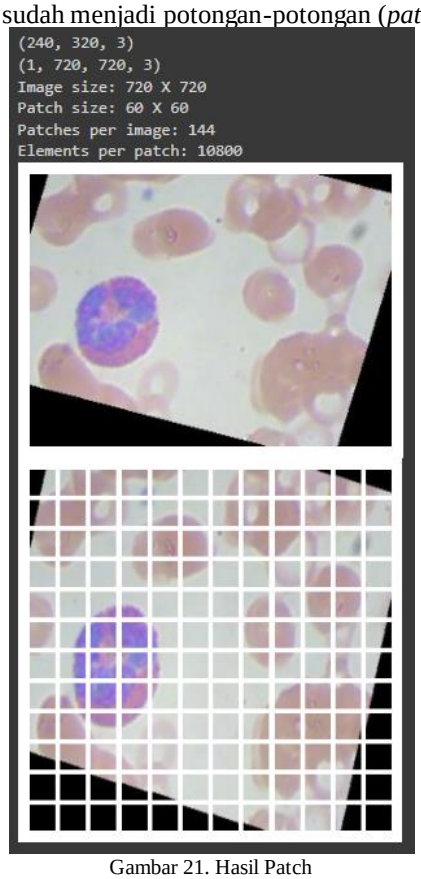
```python
class Patches(layers.Layer):
    def __init__(self, patch_size):
        super(Patches, self).__init__()
        self.patch_size = patch_size

    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(
            images = images,
            sizes=[1, self.patch_size, self.patch_size, 1],
            strides=[1, self.patch_size, self.patch_size, 1],
            rates=[1, 1, 1, 1],
            padding="VALID",
        )
        patch_dims = patches.shape[-1]
        patches = tf.reshape(patches, [batch_size, -1, patch_dims])
        return patches

    def get_config(self):
        config = super().get_config()
        config.update({"patch_size": self.patch_size})
        return config
```

Gambar 20. Class Patches

Gambar 21 merupakan hasil gambar yang sudah menjadi potongan-potongan (*patch*).



Gambar 21. Hasil Patch

G. Patch Encoder

Gambar 22 merupakan implementasi dalam mengubah setiap potongan gambar menjadi lapisan linear.

```python
class PatchEncoder(layers.Layer):

    def __init__(self, num_patches, projection_dim):
        super(PatchEncoder, self).__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim
        )
    def call(self, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) + self.position_embedding(positions)
        return encoded

    def get_config(self):
        config = super().get_config()
        config.update({"num_patches": self.num_patches})
        return config
```

Gambar 22. Patch Encoder

H. Pembuatan Model

Gambar 23 merupakan implementasi dalam pembuatan model ViT yang berisi *input*, hasil augmentasi data, potongan gambar, dan potongan gambar yang sudah di encode.

```python
def vit_model():
    # Inputs
    inputs = layers.Input(shape=input_shape)
    # Data Augmentation
    augmented = augmentation_layer(inputs)
    # Patches
    patches = Patches(patch_size)(augmented)
    encoder_patches = PatchEncoder(num_patches, projection_dim)(patches)

    for _ in range(transformer_layers):
        # Normalisasi
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoder_patches)

        # Multi-Head Attention Layer
        attention_output = layers.MultiHeadAttention(
            num_heads=num_heads,
            key_dim=projection_dim,
            dropout=0.1
        )(x1, x1)

        x2 = layers.Add()([attention_output, encoder_patches])

        # Normalisasi
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)

        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)

        encoder_patches = layers.Add()([x3, x2])

    representation = layers.LayerNormalization(epsilon=1e-6)(encoder_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)

    features = mlp(representation,
                   hidden_units=mlp_head_units,
                   dropout_rate=0.5)

    outputs = layers.Dense(num_classes)(features)

    model = keras.Model(inputs=inputs, outputs=outputs)
    return model
```

Gambar 23. Pembuatan Model

I. Pelatihan, Tes, dan Validasi Model

Berikut adalah tahapan implementasi dalam melatih dan validasi model:

```
model.compile(
    optimizer=keras.optimizers.AdamW(learning_rate=learning_rate, weight_decay=weight_decay),
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=["accuracy"],
)
```

Gambar 24. Compile Model

Gambar 25 merupakan tahapan *train* model menggunakan hyperparameter terbaik.

```
history = model.fit(
    X_train,
    y_train,
    epochs=100,
    batch_size=batch_size,
    validation_data=(X_val, y_val),
)
```

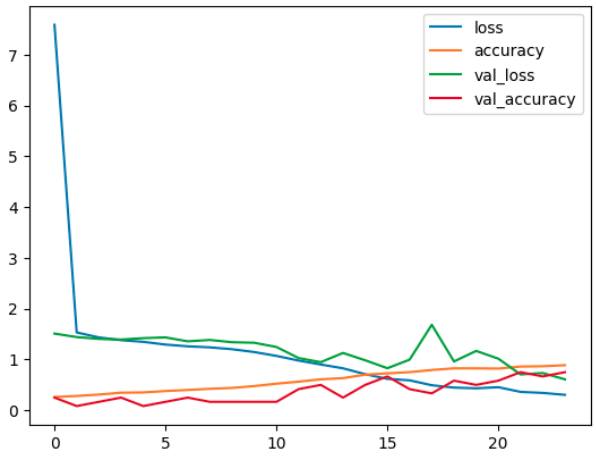
Gambar 25. Train Model

Tabel 4 merupakan hasil akurasi dari proses *train*.

TABEL IVII
HASIL TRAIN

Loss	Accuracy	Val_loss	Val_accuracy
0.0460	0.9841	0.6090	0.7500

Gambar 26 merupakan grafik akurasi dan loss dari proses *train*.



Gambar 26. Grafik Hasil Train

Gambar 27 merupakan tahapan validasi model yang sudah dilatih menggunakan data *test set*.

```
validation = model.evaluate(X_test, y_test)
print("[loss, accuracy]", validation)
```

Gambar 27. Validasi Model

TABEL VII
HASIL VALIDASI MODEL

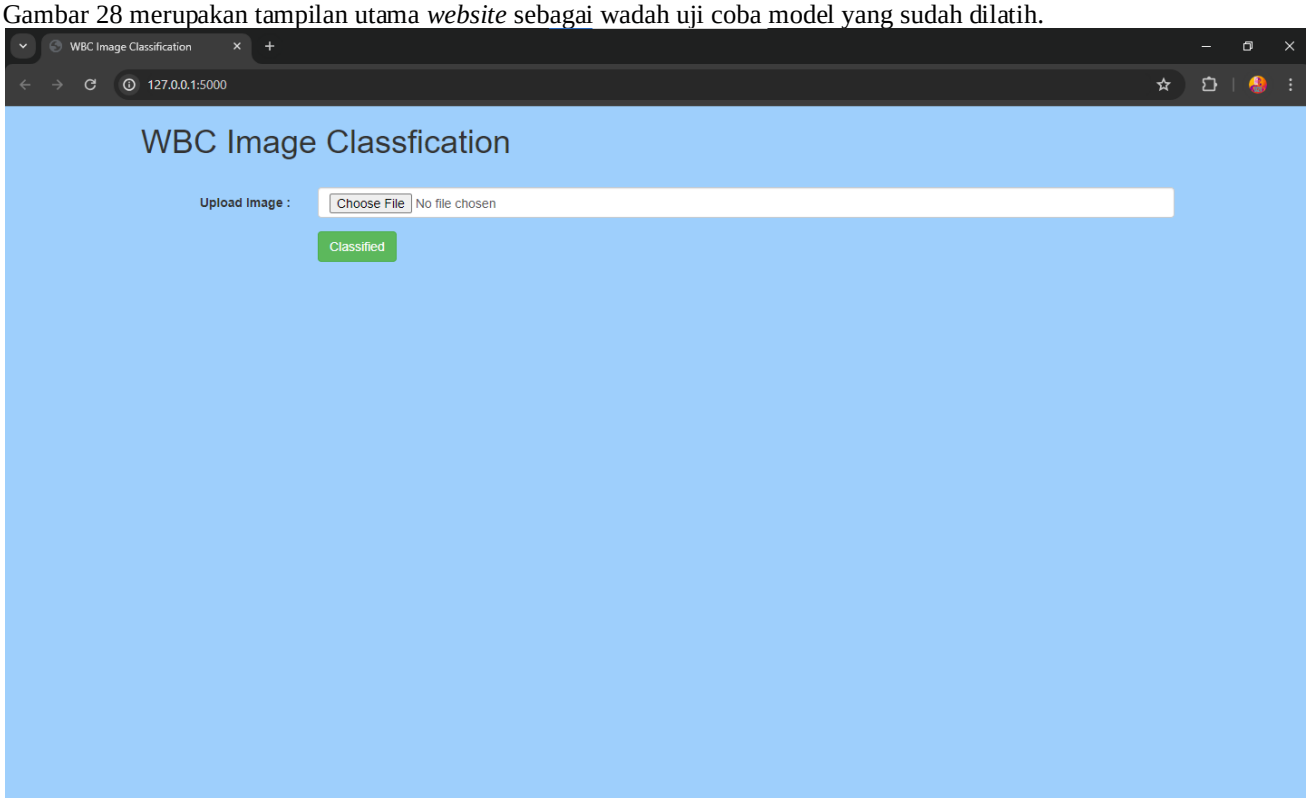
Validation loss	Validation accuracy
0.6170	0.8344

Hasil akurasi sebesar 83.44% tersebut didapati melalui hasil prediksi model terhadap data *test set*.

V. PENGUJIAN

A. Tampilan Website

Gambar 28 merupakan tampilan utama *website* sebagai wadah uji coba model yang sudah dilatih.



Gambar 28. Tampilan Website

Pada *website* tersebut user dapat menggunggah gambar yang akan diklasifikasi kemudian dapat melakukan proses klasifikasi dengan menekan tombol hijau.

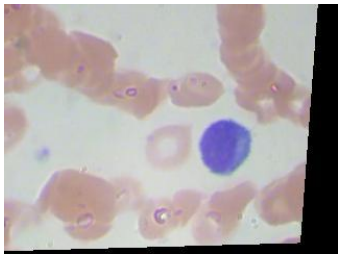
B. Gambar Pengujian

Berikut adalah beberapa gambar yang digunakan dalam proses uji coba pada *website*.

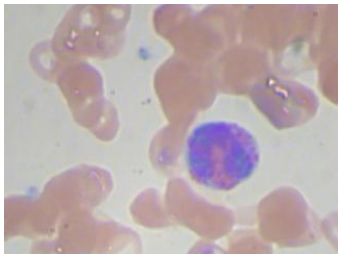
Gambar 29. Monosit (Validation Set)

Gambar 30. Limfosit (Validation Set)

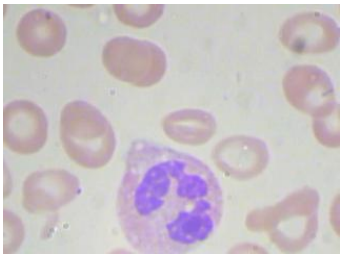
Gambar 31. Eosinofil (Test Set)



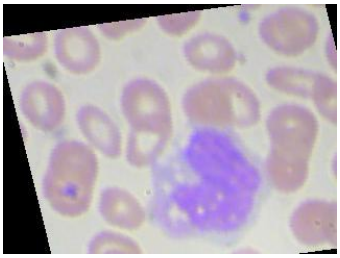
Gambar 32. Limfosit (Test Set)



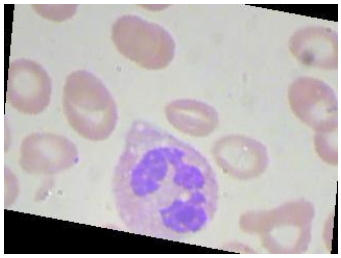
Gambar 33. Eosinofil (Validation Set)



Gambar 34. Neutrofil (Validation Set)



Gambar 35. Monosit (Test Set)

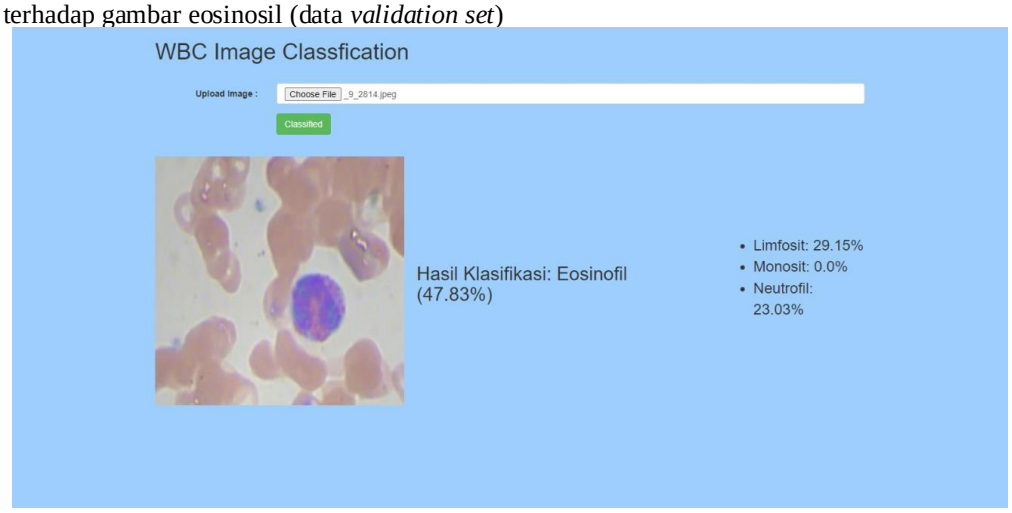


Gambar 36. Neutrofil (Test Set)

C. Pengujian Model

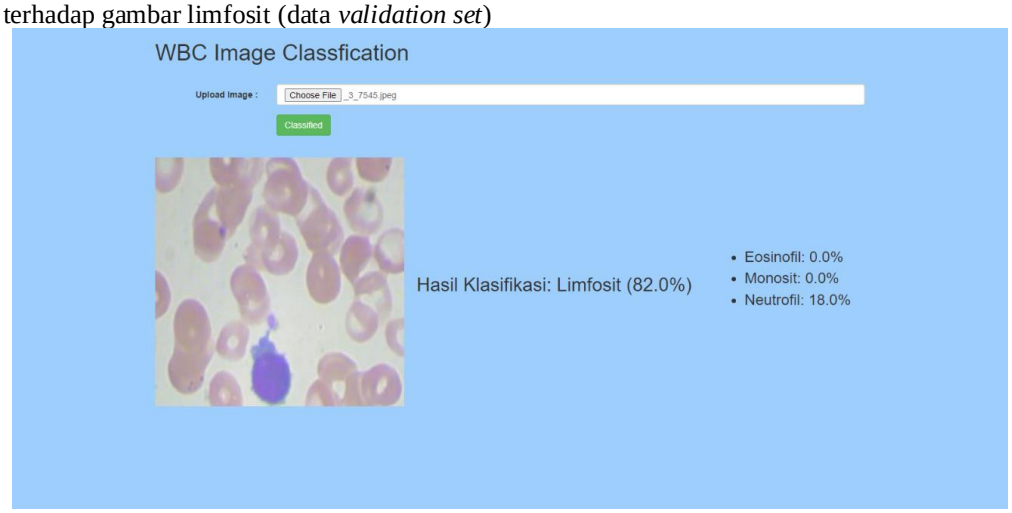
Berikut adalah hasil dari tahapan pengujian model pada *website*:

1. Pengujian terhadap gambar eosinosil (data *validation set*)



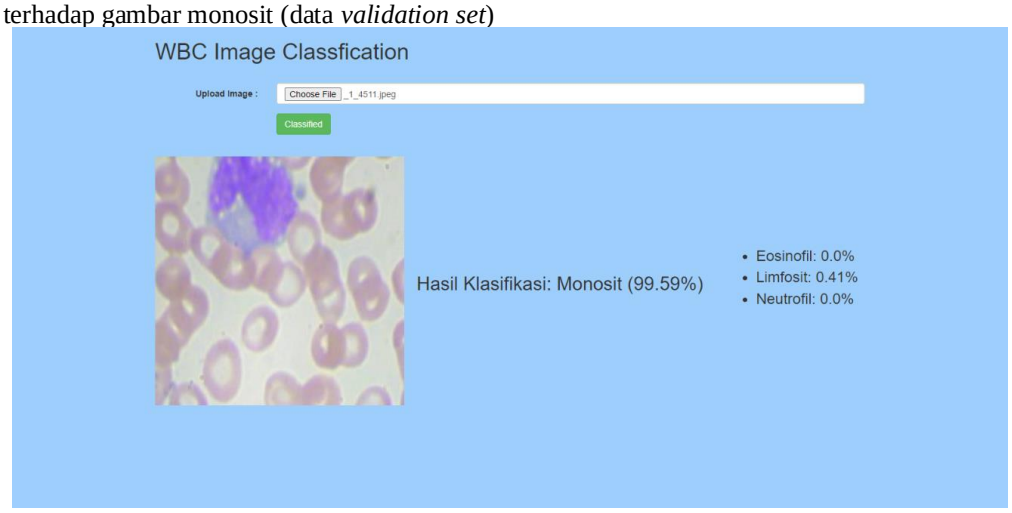
Gambar 37. Hasil Pengujian Eosinofil (Data Validation Set)

2. Pengujian terhadap gambar limfosit (data *validation set*)



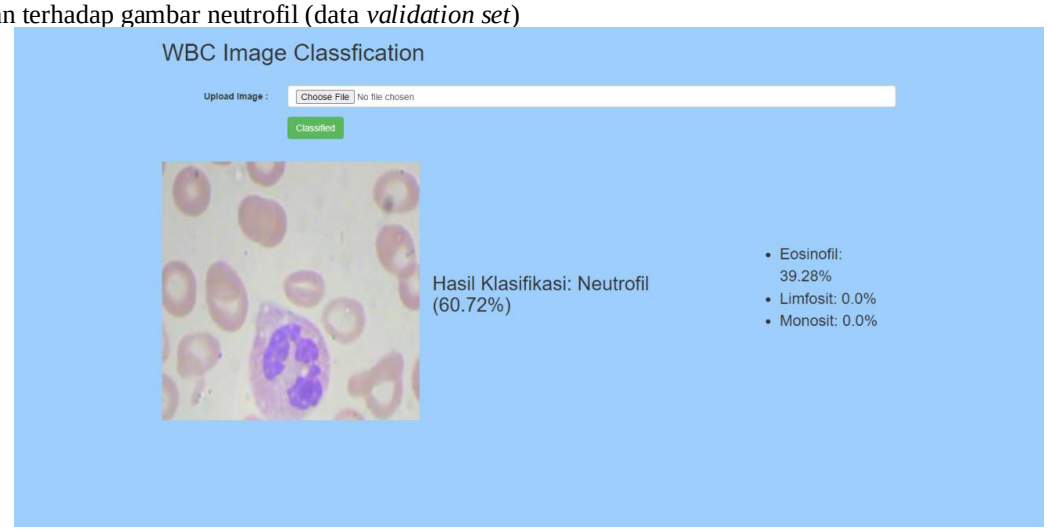
Gambar 38. Hasil Pengujian Limfosit (Data Validation Set)

3. Pengujian terhadap gambar monosit (data *validation set*)



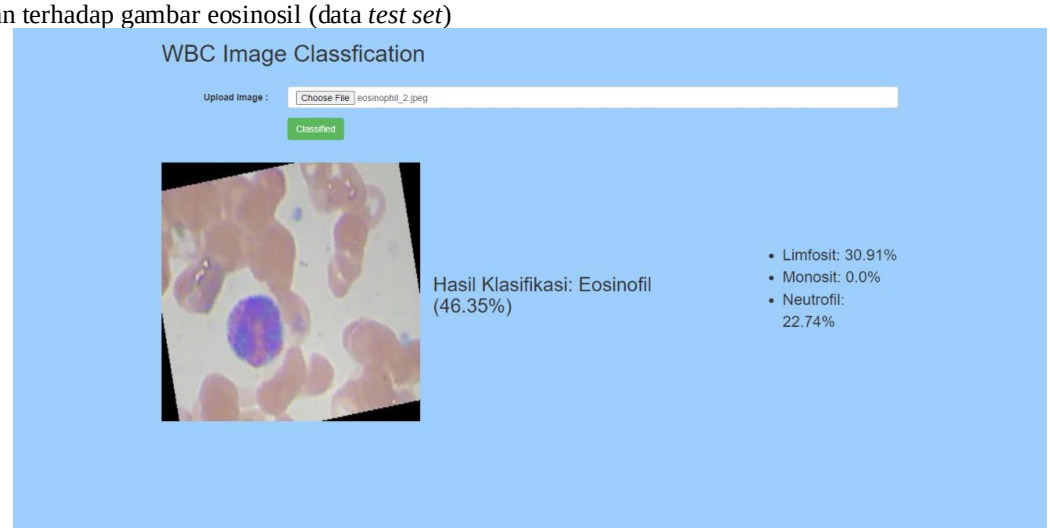
Gambar 39. Hasil Pengujian Monosit (Data Validation Set)

4. Pengujian terhadap gambar neutrofil (data *validation set*)



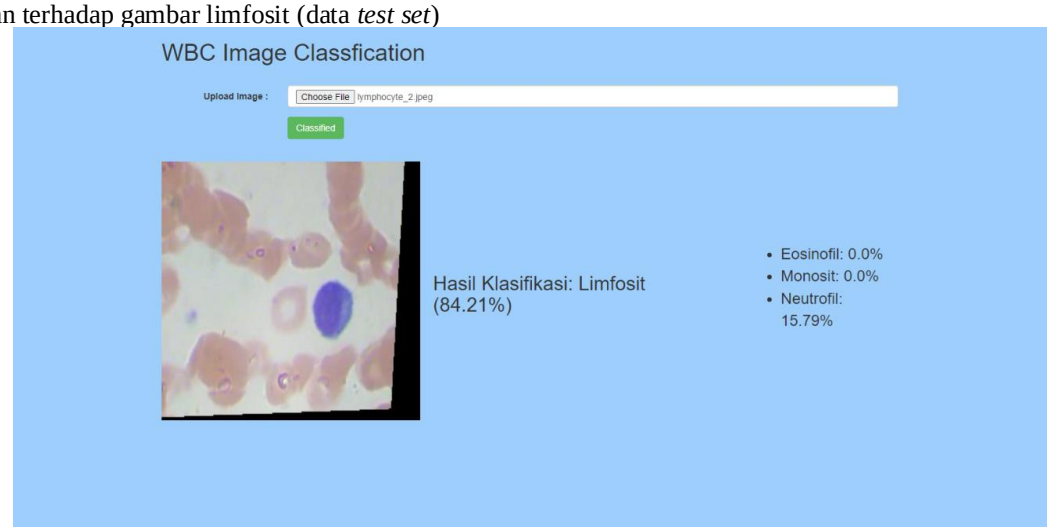
Gambar 40. Hasil Pengujian Neutrofil (Data Validation Set)

5. Pengujian terhadap gambar eosinosil (data *test set*)



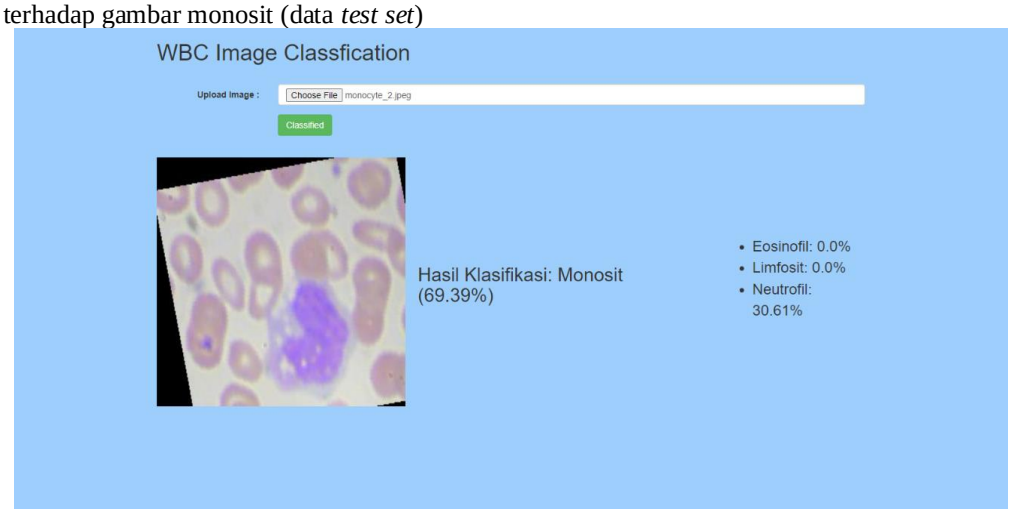
Gambar 41. Hasil Pengujian Eosinofil (Data Test Set)

6. Pengujian terhadap gambar limfosit (data *test set*)



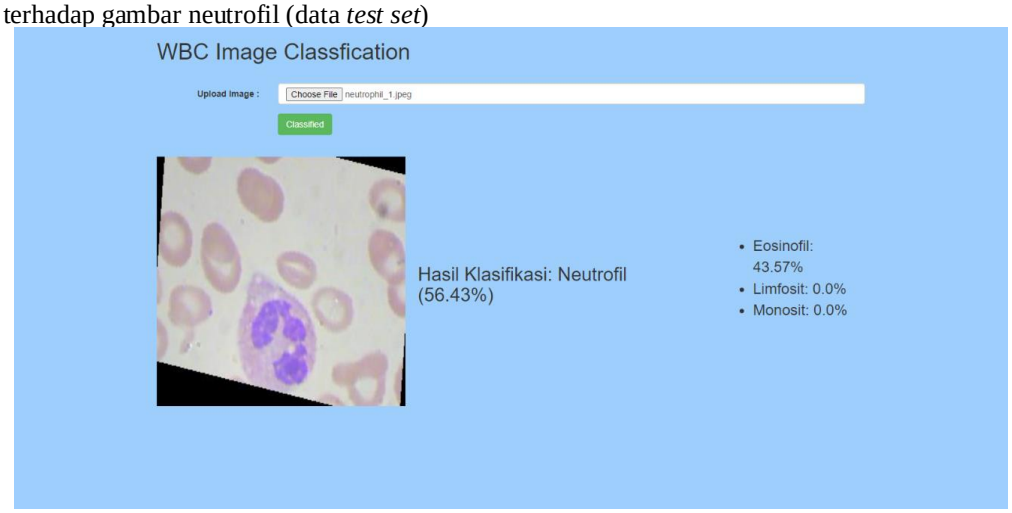
Gambar 42. Hasil Pengujian Limfosit (Data Test Set)

7. Pengujian terhadap gambar monosit (data *test set*)



Gambar 43. Hasil Pengujian Monosit (Data Test Set)

8. Pengujian terhadap gambar neutrofil (data *test set*)



Gambar 44. Hasil Pengujian Neutrofil (Data Test Set)

VI. SIMPULAN DAN SARAN

A. Simpulan

Penerapan Vision Transformer model untuk mengklasifikasi jenis sel darah putih berdasarkan gambar dapat diterapkan dan mendapatkan hasil yang baik pada saat dilakukan uji coba dan penggunaan kombinasi nilai hyperparameter terbaik dengan menggunakan GridSearch tuner dalam melakukan hyperparameter tuning dapat menghasilkan kinerja atau performa yang cukup baik dalam model melakukan klasifikasi, sehingga pada saat dilakukan uji coba melalui *website* yang berisi model, didapati hasil pengklasifikasiannya tepat sesuai dengan gambar.

B. Saran

Terdapat saran berdasarkan hasil dari laporan ini yaitu diharapkan dalam pengembangan penelitian Vision Transformer ini nantinya dapat dikembangkan lebih lanjut agar tidak hanya untuk mengklasifikasi gambar tetapi juga dapat menghitung jumlah sel darah putih pada sebuah gambar, sehingga akan memberikan potensi lebih baik dalam dunia medis.

DAFTAR PUSTAKA

[1] V. P. Maltsev, A. G. Hoekstra dan M. A. Yurkin, "Optics of white blood cells: optical models, simulations, and experiments," *Advanced Optical Flow Cytometry*, vol. 4, p. 3, 2011.

[2] A. Gautam dan H. Bhadauria, "Classification of white blood cells based on morphological features," *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 2363-2368, 2014.

[3] M. A'tourrohman, "Teknik Menghitung Jumlah Eritrosit dan leukosit Pada Manusia," *Jurnal Fisiologi, February,* pp. 1-8, 2019.

[4] P. Mali, V. P. Gejji dan V. K. Aithal, "A SURVEY FOR AUTOMATIC Identification And Counting Of Blood Cells," *Hemoglobin,* vol. 12, pp. 14-17, 2020.

[5] S. Khan, A. Khan, F. S. Khattak dan A. Naseem, "An accurate and cost effective approach to blood cell count," *International Journal of Computer Applications,* vol. 50, pp. 18-24, 2012.

[6] F. Chollet, Deep Learning with Python Second Edition, Simon and Schuster, 2021.

[7] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu dan M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing,* vol. 187, pp. 27-48, 2016.

[8] R. Vargas, M. Amir dan R. Ruiz, "Deep learning: a review," 2017.

[9] I. Goodfellow, Y. Bengio dan A. Courville, Deep Learning, MIT Press, 2016.

[10] M. Khan, B. Jan dan H. Farman, Deep Learning: Convergence to Big Data Analytics, Springer, 2019.

[11] Huawei Technologies Co., Ltd., Artificial Intelligence Technology, Hangzhou, China: Springer Singapore, 2023.

[12] J. Johnson, "What's a Deep Neural Network? Deep Nets Explained," BMC Software, Inc., [Online]. Available: https://www.bmc.com/blogs/deep-neural-network/. [Diakses 9 Maret 2024].

[13] M. Uzair dan N. Jamil, "Effects of Hidden Layers on the Efficiency of Neural networks," *2020 IEEE 23rd International Multitopic Conference (INMIC),* pp. 1-6, 2020.

[14] N. McClure, TensorFlow Machine Learning Cookbook, PACKT publishing Ltd, 2017.

[15] A. Géron, Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow, O'Reilly Media, Inc., 2022.

[16] M. A. Hasan, "Neural Machine Translation for the Bangla-English Language Pair," [Online]. Available: https://www.researchgate.net/figure/Transformer-Encoder-Decoder-architecture-taken-from-Vaswani-et-al-9-for-illustration_fig2_338223294. [Diakses 7 Maret 2024].

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser dan I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems,* vol. 30, 2017.

[18] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit dan N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929,* 2020.

[19] M. Yildirim dan A. Çinar, "Classification of white blood cells by deep learning methods for diagnosing disease," *Rev. d'Intelligence Artif.,* vol. 33, pp. 335-340, 2019.

[20] H. Kutlu, E. Avci dan F. Özyurt, "White blood cells detection and classification based on regional convolutional neural networks," *Medical hypotheses,* vol. 135, p. 109472, 2020.

[21] C. Cheuque, M. Querales, R. León, R. Salas dan R. Torres, "An efficient multi-level convolutional neural network approach for white blood cells classification," *Diagnostics,* vol. 12, p. 248, 2022.

[22] O. Katar dan O. Yildirim, "An explainable vision transformer model based white blood cells classification and localization," *Diagnostics,* vol. 13, p. 2459, 2023.

[23] S. Chen, S. Lu, S. Wang, Y. Ni dan Y. Zhang, "Shifted window vision transformer for blood cell classification," *Electronics,* vol. 12, p. 2442, 2023.

[24] S. M. Dipto, M. T. Reza, M. N. J. Rahman, M. Z. Parvez, P. D. Barua dan S. Chakraborty, "An XAI Integrated Identification System of White Blood Cell Type Using Variants of Vision Transformer," *Proceedings of the Second International Conference on Innovations in Computing Research (ICR'23),* pp. 303-315, 2023.