BAB IV

IMPLEMENTASI DAN EVALUASI

4.1 Pelatihan Model

4.1.1 Augmentasi Data

Untuk meningkatkan kompleksitas dataset secara artifisial, diperlukan augmentasi data dengan melakukan transformasi secara acak kepada data pelatihan. Sebelum proses transformasi gambar, diterapkan *preprocessing function* yang mengkonversi gambar menjadi format yang dapat diterima oleh model *Convolutional Neural Network*. Salah satu *preprocessing* ini adalah proses *resize* gambar menjadi resolusi yang dapat diterima oleh tiap model. Tabel 4.1 menunjukkan konfigurasi ukuran input tiap model.

Tabel 4.1 Ukuran input tiap model Convolutional Neural Network

Arsitektur	Resolusi *Input*
Inception-V3	299 x 299
ResNet50	224 x 224
VGG-16	224 x 224
MobileNetV2	224 x 224

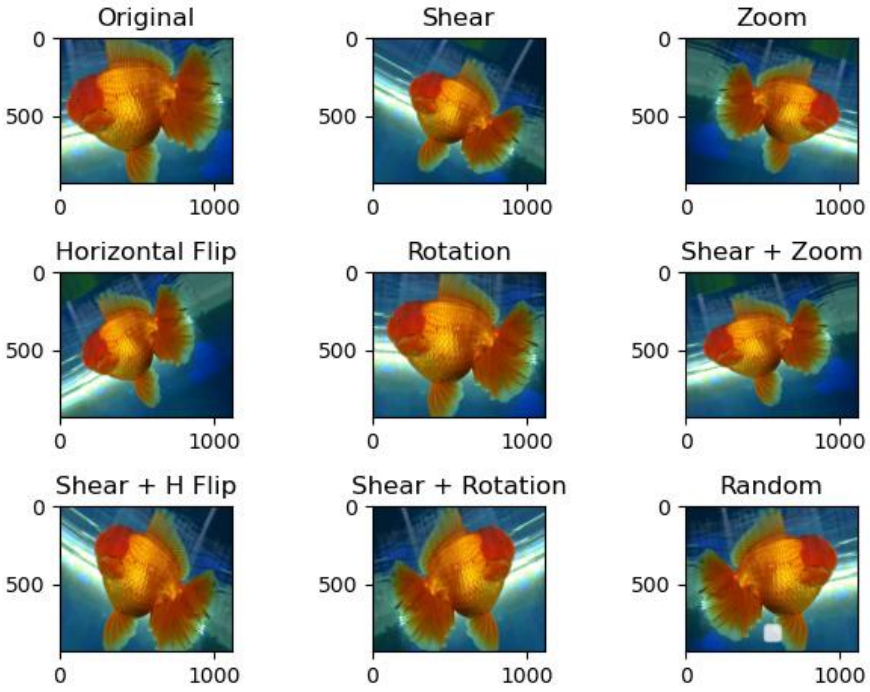
Transformasi yang diterapkan pada dataset melibatkan teknik-teknik seperti *shear, zoom, horizontal flip,* dan *rotation*. Melalui penerapan *shear*, gambar dapat mengalami pergeseran sudut tertentu, sementara *zoom* memungkinkan untuk mengecilkan atau memperbesar bagian tertentu dari gambar. *Horizontal flip* memberikan variasi tambahan dengan mencerminkan gambar secara horizontal, sedangkan *rotation* memungkinkan gambar diputar untuk memperkenalkan variasi orientasi. Gambar 4.1 menjabarkan kode yang

digunakan dalam melakukan augmentasi data terhadap dataset yang telah diakusisi.

```
train_datagen =
ImageDataGenerator(preprocessing_function=preprocess_input,
                   shear_range=0.2,
                   zoom_range=0.2,
                   horizontal_flip=True,
                    rotation_range=30)
```

Gambar 4.1 Kode augmentasi data sebelum pelatihan

Gambar 4.2 mengilustrasikan contoh augmentasi data yang dilakukan kepada satu sampel gambar ikan mas koki.

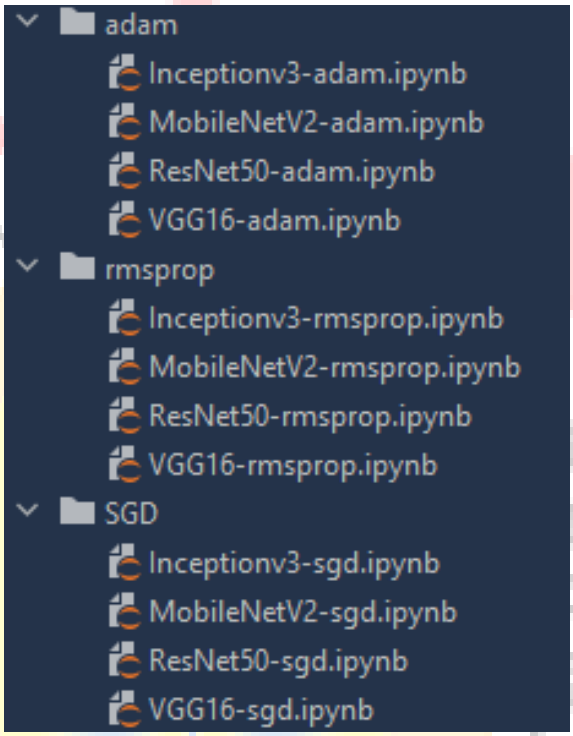


Gambar 4.2 Visualisasi Terhadap Augmentasi Data

4.1.2 Konfigurasi Model dan Optimizer

Konfigurasi model dan *optimizer* dilakukan pada file *Jupyter Notebook* masing-masing model. Struktur folder dikelompokkan berdasarkan *optimizer* yang digunakan, yaitu *ADAM*, *SGD*, dan *RMSProp*. Di dalam folder tersebut,

file *notebook* tiap arsitektur *Convolutional Neural Network* tersimpan. Penyusunan struktur folder ini memungkinkan file *notebook* menjadi lebih terorganisir dan memberikan akses cepat pada konfigurasi tiap model. Gambar 4.3 menunjukkan struktur folder yang digunakan untuk mengorganisir tiap model yang dilatih.



Gambar 4.3 Struktur folder notebook model

Arsitektur *Convolutional Neural Network* dideklarasikan pada variable base_model. Untuk melakukan proses *transfer learning*, bobot dikonfigurasi menjadi '*ImageNet*' serta layer.trainable diatur menjadi *false* untuk membekukan lapisan *Convolutional Neural Network* agar informasi yang telah didapatkan ketika *transfer learning* tidak hilang. Tabel 4.2 menyajikan rincian konfigurasi dari parameter yang digunakan pada tiap model.

Tabel 4.2 Konfigurasi parameter model Convolutional Neural Network

Parameter	*Inception-V3*	*VGG-16*	*MobileNetV2*	*ResNet50*
base_model	*Inception-V3()*	*VGG-16()*	*MobileNetV2()*	*ResNet50()*
Include_top	*False*	*False*	*False*	*False*
Weights	*ImageNet*	*ImageNet*	*ImageNet*	*ImageNet*
Jumlah neuron	1024	1024	1024	1024
Aktivasi *dense layer*	*Relu*	*Relu*	*Relu*	*Relu*
Layer.trainable	*False*	*False*	*False*	*False*

Optimizer diterapkan pada perintah kompilasi (*compile*) model. Setiap model dilatih sebanyak 20 *epoch* (iterasi) karena jumlah data yang relatif kecil, serta memberikan perbandingan yang seimbang antar model ketika tahap evaluasi. Pengukuran *loss* yang digunakan adalah *categorical crossentropy* yang merupakan metode pengukuran kesalahan akurasi (*loss*) yang sering digunakan pada model klasifikasi banyak kelas. Tabel 4.3 menjabarkan konfigurasi dari kompilasi model.

Tabel 4.3 Konfigurasi kompilasi model

Parameter	*ADAM*	*SGD*	*RMSProp*
Optimizer	*ADAM*	*SGD*	*RMSProp*
Loss	*Categorical_crossentropy*	*categorical_crossentropy*	*categorical_crossentropy*
Metrics	*Accuracy*	*Accuracy*	*Accuracy*
Epochs	20	20	20

4.1.1 Kompilasi Model

Proses *training* model dilakukan sesuai dengan konfigurasi masing-masing model. Berkat perangkat keras yang modern, model hanya memakan waktu

pelatihan dari 104 detik hingga 169 detik tergantung arsitektur *Convolutional Neural Network* yang dilatih. Tabel 4.4 hingga 4.7 merupakan akurasi keempat model *Convolutional Neural Network* dengan setiap konfigurasi *optimizer* dari awal *training* hingga selesai:

Tabel 4.4 Perbandingan Akurasi Inception-V3 dengan tiga konfigurasi optimizer

Epoch	***Inception-V3 + ADAM***	***Inception-V3 + SGD***	***Inception-V3 + RMSProp***
Epoch 1	0,3180	0,2984	0,3311
Epoch 2	0,5754	0,4131	0,4148
Epoch 3	0,6246	0,4885	0,5213
Epoch 4	0,6885	0,5607	0,5492
Epoch 5	0,7689	0,5787	0,6279
Epoch 6	0,8262	0,5869	0,6328
Epoch 7	0,8262	0,6590	0,6590
Epoch 8	0,8295	0,6623	0,7033
Epoch 9	0,8180	0,6738	0,7361
Epoch 10	0,8148	0,6803	0,7311
Epoch 11	0,8508	0,6869	0,7377
Epoch 12	0,8623	0,7311	0,7689
Epoch 13	0,8000	0,6852	0,7820
Epoch 14	0,8607	0,7541	0,7902
Epoch 15	0,8311	0,7443	0,7885
Epoch 16	0,8803	0,7525	0,7852
Epoch 17	0,9197	0,7623	0,8328

Epoch 18	0,8984	0,7787	0,8148
Epoch 19	0,8689	0,7770	0,8492
Epoch 20	0,8967	0,7754	0,8377

Tabel 4.5 Perbandingan Akurasi ResNet50 dengan tiga konfigurasi optimizer

Epoch	***ResNet50 + ADAM***	***ResNet50 + SGD***	***ResNet50 + RMSProp***
Epoch 1	0,4754	0,4197	0,3934
Epoch 2	0,7541	0,6295	0,5803
Epoch 3	0,8410	0,6574	0,6656
Epoch 4	0,9000	0,7607	0,7180
Epoch 5	0,9066	0,8279	0,7672
Epoch 6	0,9230	0,8131	0,8066
Epoch 7	0,9426	0,8262	0,8164
Epoch 8	0,9426	0,8820	0,8475
Epoch 9	0,9607	0,9016	0,8934
Epoch 10	0,9656	0,8918	0,8951
Epoch 11	0,9754	0,9213	0,8557
Epoch 12	0,9820	0,9426	0,9180
Epoch 13	0,9705	0,9426	0,9082
Epoch 14	0,9836	0,9344	0,9393
Epoch 15	0,9869	0,9410	0,9033
Epoch 16	0,9836	0,9525	0,9508

Epoch 17	0,9984	0,9131	0,9279
Epoch 18	0,9869	0,9492	0,9525
Epoch 19	0,9885	0,9525	0,9344
Epoch 20	0,9934	0,9557	0,9754

Tabel 4.6 Perbandingan Akurasi VGG-16 dengan tiga konfigurasi optimizer

Epoch	***VGG-16 + ADAM***	***VGG-16 + SGD***	***VGG-16 + RMSProp***
Epoch 1	0,4033	0,4525	0,4131
Epoch 2	0,6787	0,7246	0,6475
Epoch 3	0,8131	0,8295	0,7213
Epoch 4	0,8869	0,8246	0,7738
Epoch 5	0,9049	0,9180	0,8066
Epoch 6	0,9164	0,9197	0,8459
Epoch 7	0,9443	0,9459	0,8361
Epoch 8	0,9508	0,9344	0,9049
Epoch 9	0,9738	0,9721	0,9016
Epoch 10	0,9705	0,9607	0,8672
Epoch 11	0,9016	0,9787	0,8984
Epoch 12	0,9541	0,9787	0,9311
Epoch 13	0,9738	0,9885	0,9033
Epoch 14	0,9902	0,9820	0,9328
Epoch 15	0,9836	0,9787	0,9361

Epoch 16	0,9787	0,9869	0,8869
Epoch 17	0,9689	0,9721	0,9344
Epoch 18	0,9148	0,9820	0,9656
Epoch 19	0,9508	0,9852	0,9656
Epoch 20	0,9623	0,9902	0,9230

Tabel 4.7 Perbandingan Akurasi MobileNetV2 dengan tiga konfigurasi optimizer

Epoch	***MobileNetV2 + ADAM***	***MobileNetV2 + SGD***	***MobileNetV2 + RMSProp***
Epoch 1	0.5066	0.3525	0.4525
Epoch 2	0.7475	0.5934	0.6361
Epoch 3	0.8279	0.6443	0.7361
Epoch 4	0.8770	0.7049	0.7525
Epoch 5	0.9098	0.7492	0.7951
Epoch 6	0.9131	0.7639	0.8279
Epoch 7	0.9082	0.7754	0.8639
Epoch 8	0.9230	0.8082	0.8656
Epoch 9	0.9361	0.8082	0.8754
Epoch 10	0.9459	0.8246	0.8836
Epoch 11	0.9607	0.8525	0.9328
Epoch 12	0.9656	0.8623	0.9410
Epoch 13	0.9672	0.8672	0.9262
Epoch 14	0.9656	0.8902	0.9426

Epoch 15	0.9639	0.8836	0.9131
Epoch 16	0.9754	0.8656	0.9443
Epoch 17	0.9689	0.8967	0.9623
Epoch 18	0.9738	0.8967	0.9279
Epoch 19	0.9656	0.8951	0.9557
Epoch 20	0.9754	0.9016	0.9295

4.2 Evaluasi Model

4.2.1 Confusion Matrix

Model yang telah dilatih dan diuji menggunakan data *testing* akan menghasilkan *confusion matrix* yang mengilustrasikan performa model dengan menunjukkan jumlah *True Positive*, *True Negative*, *False Positive*, dan *False Negative*.

Dalam konteks klasifikasi banyak kelas, *True Positive* merujuk pada *cell* diagonal pada *confusion matrix*, *True Negative* merujuk pada jumlah tiap *cell* yang tidak berada pada kolom kelas n dan baris kelas n, *False Positive* merujuk pada jumlah kolom kelas n kecuali *cell True Positive*, sedangkan *False Negative* merujuk kepada jumlah baris kelas n kecuali *cell True Positive*.

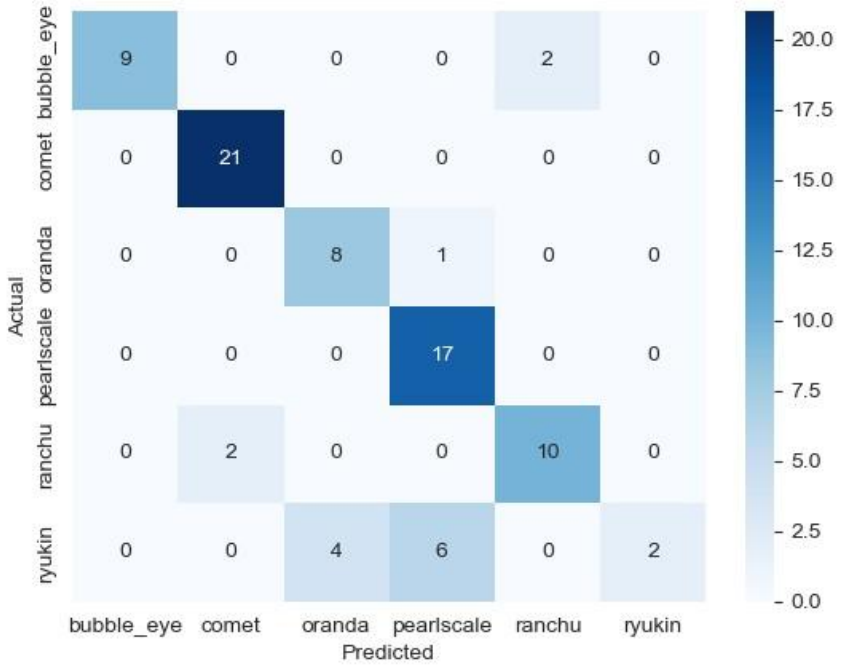
Untuk menyajikan data pengujian ke dalam bentuk *confusion matrix*, diperlukan visualisasi menggunakan library *pandas* dan *seaborn* di mana *pandas* berperan sebagai penyaji data tabular, sedangkan *seaborn* berperan untuk merepresentasikan frekuensi data menggunakan warna *heatmap* pada *confusion matrix* tersebut.

```
nb_samples = len(test_generator)
y_prob = []
y_act = []

for _ in range(nb_samples):
    X_test, Y_test = test_generator.next()
    y_prob.extend(model.predict(X_test))
    y_act.extend(Y_test)

predicted_class_indices = [i.argmax() for i in y_prob]
actual_class_indices = [i.argmax() for i in y_act]
predicted_class = [list(train_generator.class_indices.keys())[idx] for
idx in predicted_class_indices]
actual_class = [list(train_generator.class_indices.keys())[idx] for
idx in actual_class_indices]
out_df = pd.DataFrame(np.vstack([predicted_class,
actual_class]).T, columns=['predicted_class', 'actual_class'])
confusion_matrix = pd.crosstab(out_df['actual_class'],
out_df['predicted_class'], rownames=['Actual'],
colnames=['Predicted'])

sn.heatmap(confusion_matrix, cmap='Blues', annot=True, fmt='d')
plt.show()
```

Gambar 4.4 Kode penampilan confusion matrix

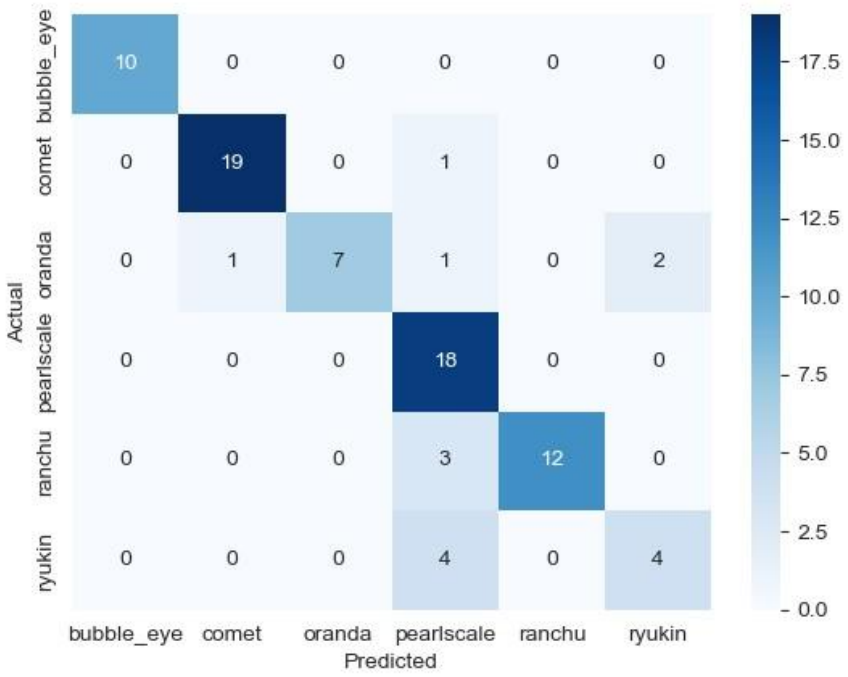
Gambar 4.4 menjabarkan algoritma yang digunakan untuk menyajikan data ke dalam bentuk *confusion matrix*. Setelah data diuji, hasil *testing* akan terbagi menjadi sumbu x yang merepresentasikan hasil prediksi model, dan sumbu y yang merepresentasikan nilai sesungguhnya dari sebuah sampel uji. Kedua sumbu ini kemudian divisualisasikan menjadi kolom dan baris menggunakan fungsi *DataFrame* milik *library pandas*. Setelah terbentuk kolom dan baris, dilakukan visualisasi *heatmap* menggunakan *library seaborn* di mana data dengan frekuensi yang lebih tinggi, maka warnanya akan semakin pekat di *confusion matrix*. Terakhir, untuk menampilkan hasil akhir dari *confusion matrix*, perlu dipanggil fungsi *plt.show()*.

Berikut merupakan *confusion matrix* masing-masing arsitektur *Convolutional Neural Netwotk* dengan tiga konfigurasi algoritma optimisasi yang berbeda:



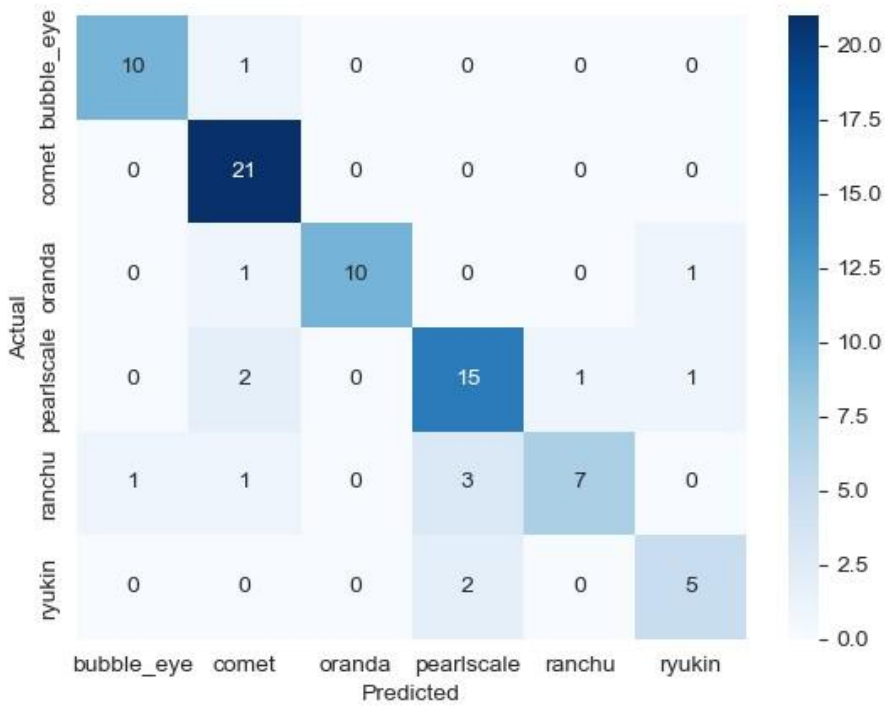
Gambar 4.5 Confusion Matrix untuk Inception-V3 dengan optimizer ADAM

Gambar 4.5 menunjukkan *confusion matrix* untuk arsitektur *Inception-V3* yang dilatih menggunakan optimisasi *ADAM*. Model ini mengalami kendala dalam mengidentifikasi tipe *Ryukin* dengan hanya dua klasifikasi tepat pada kelas tersebut. Model ini kerap mengidentifikasi tipe *Ryukin* sebagai *Oranda* maupun *Pearlscale*.



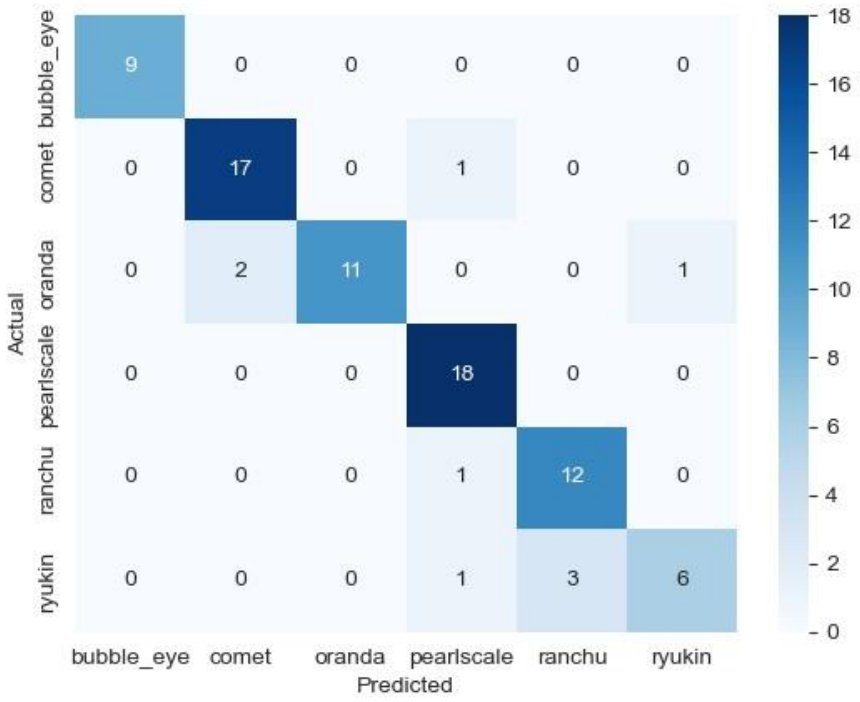
Gambar 4.6 Confusion Matrix untuk Inception-V3 dengan optimizer RMSProp

Gambar 4.6 menunjukkan *confusion matrix* untuk arsitektur *Inception-V3* yang dilatih menggunakan *optimizer RMSProp*. Model ini memiliki hasil yang lebih baik dalam mengidentifikasi tipe *Ryukin* dibanding *Inception-V3* dengan *ADAM*. Model mampu mengidentifikasi tipe *Ryukin* dengan tepat sebanyak empat kali dan tidak mengalami kesulitan dalam membedakan tipe *Ryukin* dengan tipe *Oranda*. Namun, Model masih mengalami kesulitan dalam membedakan tipe *Ryukin* dengan tipe *Pearlscale*.



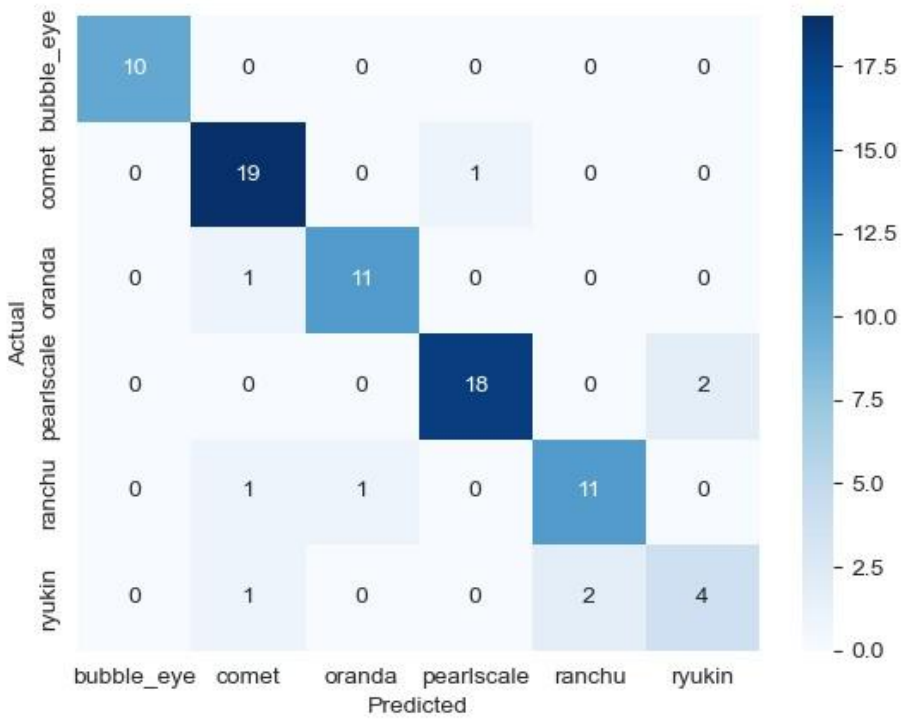
Gambar 4.7 Confusion Matrix untuk Inception-V3 dengan optimizer SGD

Gambar 4.7 menunjukkan *confusion matrix* untuk model *Inception-V3* yang dilatih menggunakan algoritma optimisasi *SGD*. Model ini mengalami kendala dalam mengidentifikasi tipe *Ranchu* dengan 7 klasifikasi tepat dibandingkan 10 dan 12 dari kedua *optimizer* sebelumnya.



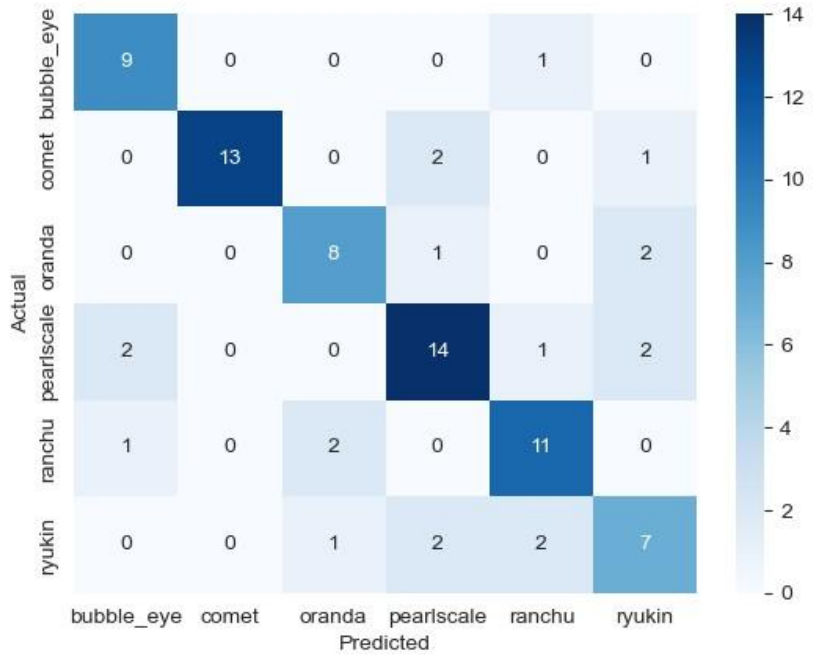
Gambar 4.8 Confusion Matrix untuk MobileNetV2 dengan optimizer ADAM

Gambar 4.8 menunjukkan *confusion matrix* untuk arsitektur *MobileNetV2* yang dilatih menggunakan optimisasi *ADAM*. Model ini menghasilkan klasifikasi yang lebih akurat dibandingkan model *Inception-V3*, dengan catatan model ini masih kesulitan dalam membedakan tipe *Ryukin* dengan tipe *Ranchu*.



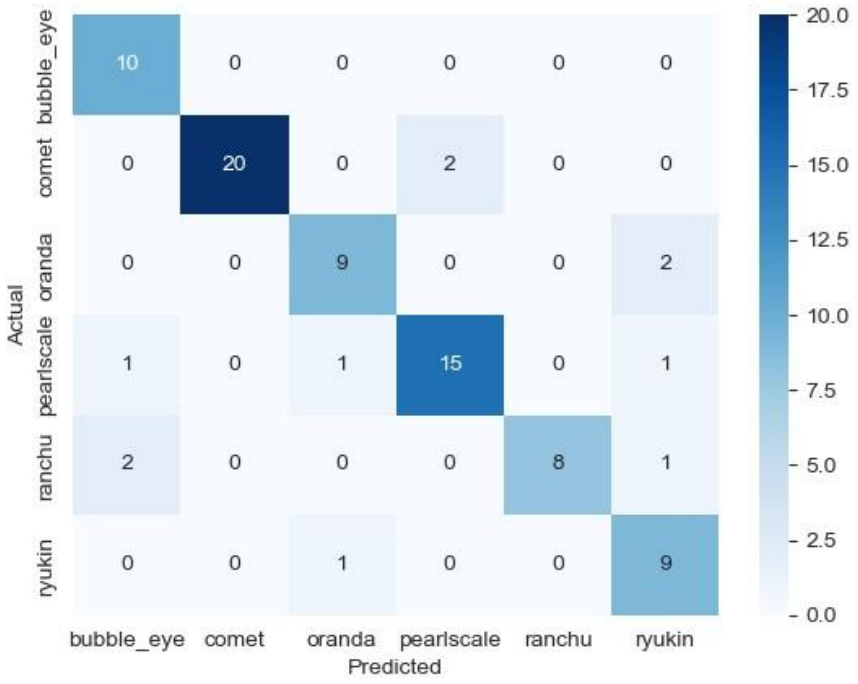
Gambar 4.9 Confusion Matrix untuk MobileNetV2 dengan optimizer RMSProp

Gambar 4.9 menunjukkan *confusion matrix* untuk arsitektur *MobileNetV2* yang dilatih menggunakan algoritma optimisasi *RMSProp*. Model ini mengalami kendala dalam mengidentifikasi tipe *Ryukin* dengan empat klasifikasi tepat pada kelas tersebut. Seperti *MobileNetV2* yang dilatih dengan *ADAM*, model ini juga kesulitan dalam membedakan tipe *Ryukin* dengan *Ranchu*.



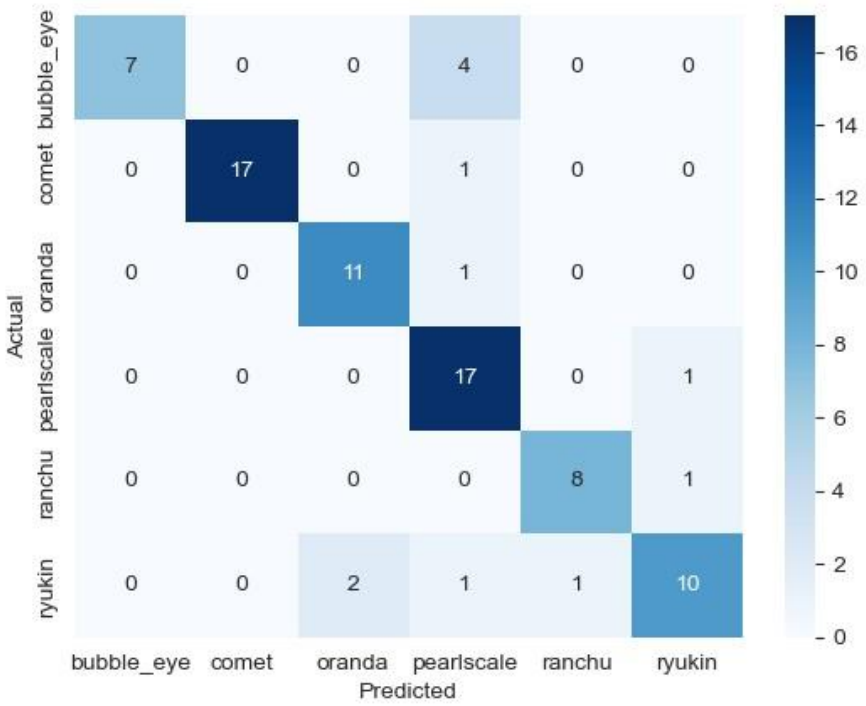
Gambar 4.10 Confusion Matrix untuk MobileNetV2 dengan optimizer SGD

Gambar 4.10 menunjukkan *confusion matrix* untuk arsitektur *MobileNetV2* yang dilatih menggunakan optimisasi *SGD*. Walaupun model ini lebih akurat dalam klasifikasi tipe *Ryukin*, namun tingkat *false positive* dan *false negative* secara keseluruhan lebih buruk dibandingkan dengan dua *optimizer* sebelumnya.



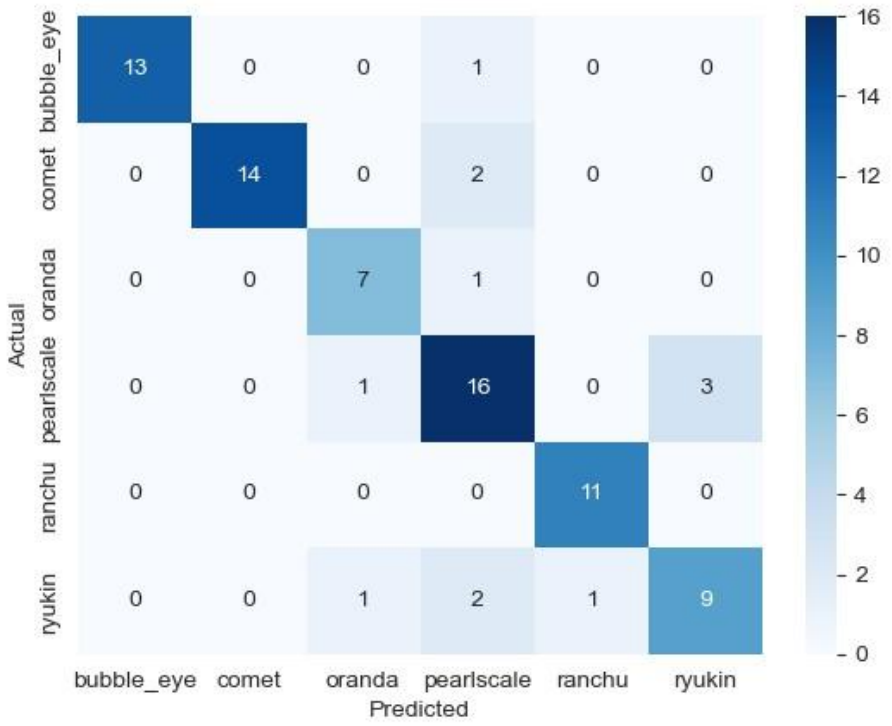
Gambar 4.11 Confusion Matrix untuk ResNet50 dengan optimizer ADAM

Gambar 4.11 menunjukkan *confusion matrix* untuk model *ResNet50* yang dilatih menggunakan *optimizer ADAM*. Arsitektur ini menghasilkan akurasi yang menjanjikan, akan tetapi model ini memiliki jumlah klasifikasi tepat yang lebih sedikit pada kelas *Ranchu* dibandingkan model-model yang telah dibahas sebelumnya. Tidak hanya itu, persentase *false positive* pada model ini lebih besar dibandingkan model sebelumnya.



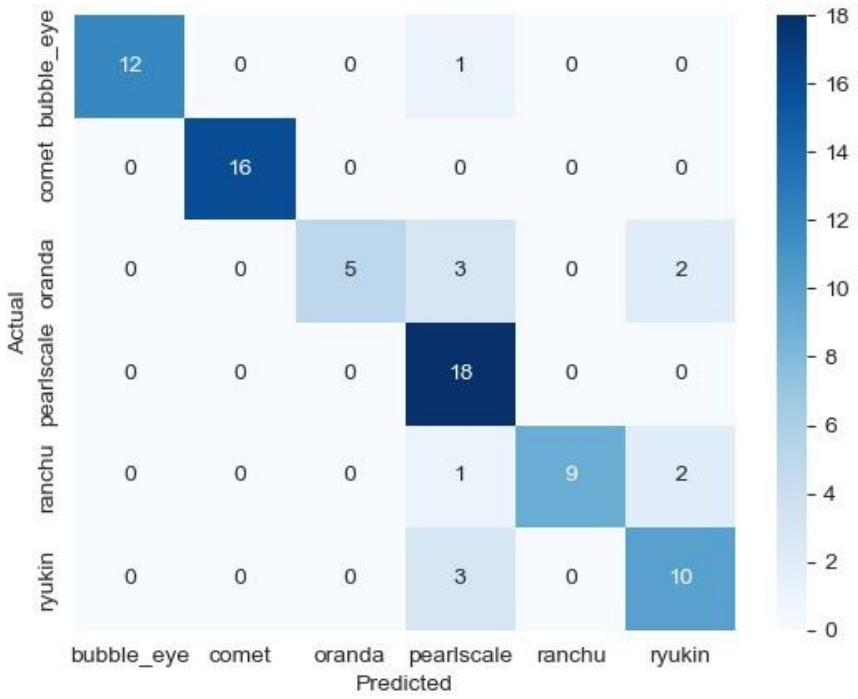
Gambar 4.12 Confusion Matrix untuk ResNet50 dengan optimizer RMSProp

Gambar 4.12 menunjukkan *confusion matrix* untuk model *ResNet50* yang dilatih menggunakan optimisasi *RMSProp*. Serupa dengan model *ResNet50* yang dilatih menggunakan *ADAM*, model ini memiliki persentase *false positive* yang lebih tinggi dibandingkan arsitektur sebelumnya.



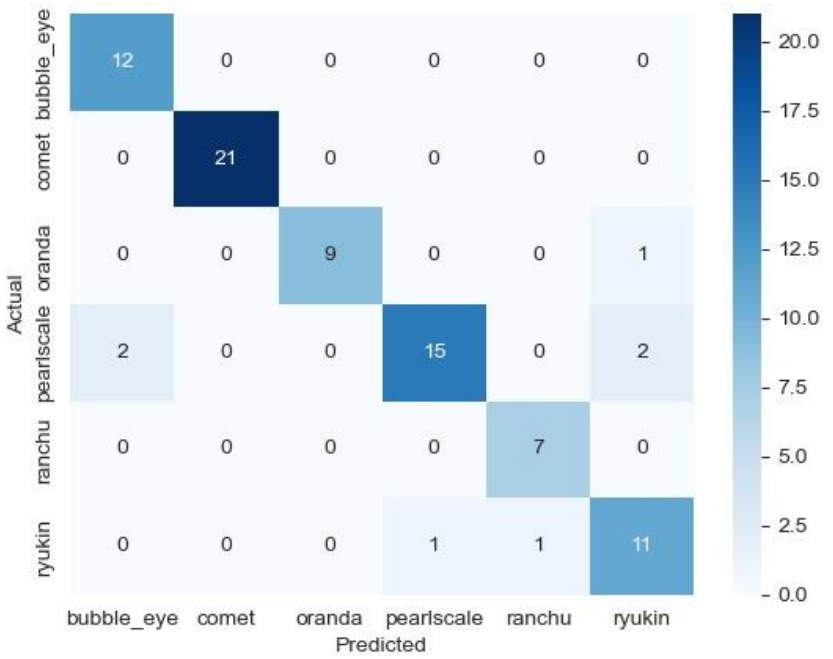
Gambar 4.13 Confusion Matrix untuk ResNet50 dengan optimizer SGD

Gambar 4.13 menunjukkan *confusion matrix* untuk model *ResNet50* yang dilatih menggunakan optimisasi *SGD*. Model ini mengklasifikasi tipe *Ranchu* lebih baik dibandingkan kedua algoritma optimisasi sebelumnya dengan hasil klasifikasi tepatnya yang mencapai 11 dibandingkan 8 dari *ADAM* dan *RMSProp*.



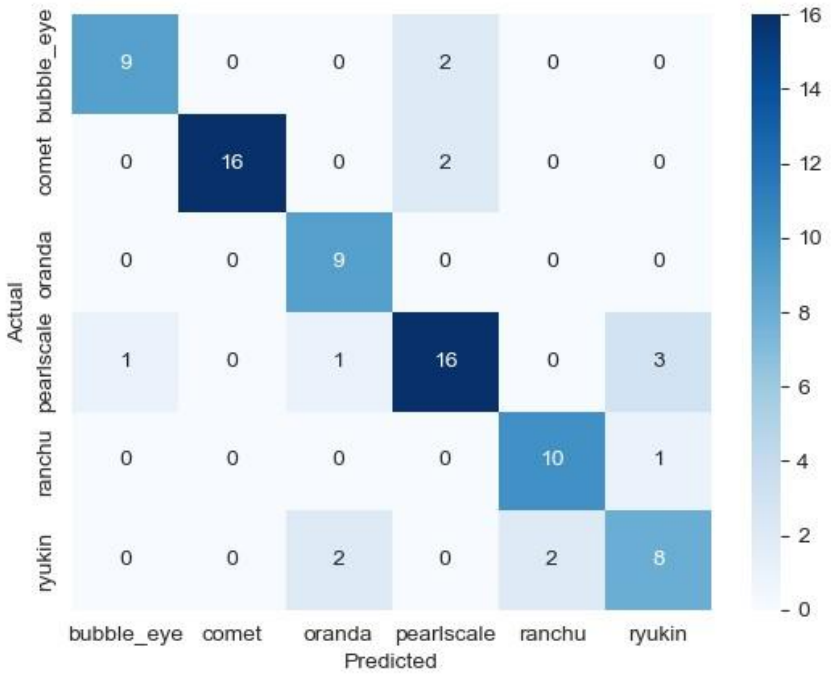
Gambar 4.14 Confusion Matrix untuk VGG-16 dengan optimizer ADAM

Gambar 4.14 menunjukkan *confusion matrix* untuk model *VGG-16* yang dilatih menggunakan optimisasi *ADAM*. Model ini mengalami kesulitan dalam mengidentifikasi tipe *Oranda* dengan hanya lima kali identifikasi *Oranda* yang tepat.



Gambar 4.15 Confusion Matrix untuk VGG-16 dengan optimizer RMSProp

Gambar 4.15 menunjukkan *confusion matrix* untuk model *VGG-16* yang dilatih menggunakan optimisasi *RMSProp*. Serupa dengan model *VGG-16* yang dilatih dengan *ADAM*, model ini mengalami kesulitan dalam mengklasifikasi satu kelas yang spesifik. Model ini berkendala dalam mengklasifikasi tipe *Ranchu* dengan identifikasi tepat hanya berjumlah tujuh. Namun, model ini memiliki persentase *false positive* terendah dibandingkan model *VGG-16* yang dilatih menggunakan *ADAM* dan *SGD*, dengan model ini hanya mengalami 3 kali *false positive* dibandingkan 8 oleh *optimizer* lain.



Gambar 4.16 Confusion Matrix untuk VGG-16 dengan optimizer SGD

Gambar 4.16 menunjukkan *confusion matrix* untuk model *VGG-16* yang dilatih menggunakan optimisasi *SGD*. Arsitektur ini memiliki pola kesalahan *false positive* maupun *false negative* yang lebih tersebar. Menandakan bahwa model ini lebih netral dan tidak mengalami kesulitan dalam mengklasifikasi satu kelas spesifik layaknya kedua *optimizer* sebelumnya.

Confusion matrix pada gambar 4.5 hingga 4.16 menunjukan bahwa algoritma optimisasi dapat mempengaruhi identifikasi kelas bahkan pada model yang sama. Pada gambar 4.6, model *Inception-V3* dengan *optimizer RMSProp* nampaknya kesulitan dalam membedakan variasi *Oranda* dengan variasi *Pearlscale*, menghasilkan *false negative* yang lebih banyak pada kelas tersebut. Fenomena yang sama dapat diamati pada *ResNet50* dengan *optimizer RMSProp* pada gambar 4.12. Pada gambar 4.9, dapat diobservasi bahwa *MobileNetV2* dengan optimisasi *RMSProp* kesulitan dalam

membedakan variasi *Oranda* dan *Pearlscale*, namun berbeda dengan dua model sebelumnya yang mengalami *false negative*, model ini mengalami *false positive*. Ketiga fenomena tersebut dapat mengimplikasikan salah satu dari dua kemungkinan. Kemungkinan pertama adalah *RMSProp* kurang cocok dalam klasifikasi banyak kelas karena ketiga fenomena tersebut terjadi ketika *optimizer RMSProp* digunakan dalam konfigurasi model. Kemungkinan kedua adalah data yang kurang seimbang menyebabkan model keliru antara variasi ikan mas koki *Oranda* dan *Pearlscale*.

Untuk mengubah hasil evaluasi dari *confusion matrix* menjadi metrik pengukuran seperti akurasi, presisi, *recall*, dan *F1 score*, dilakukan proses kalkulasi menggunakan *library SciKit Learn*. *SciKit Learn* merupakan *library* yang digunakan untuk melakukan kalkulasi dan penyajian data menggunakan bahasa pemograman *Python*. Gambar 4.17 menjabarkan cara mengkonversi *confusion matrix* dan menyajikannya menjadi metrik pengukuran seperti akurasi, presisi, *recall*, dan *F1 score*.

```
for _ in range(nb_samples):
    X_test, Y_test = test_generator.next()
    y_prob.extend(model.predict(X_test))  # Use extend instead of
append
    y_act.extend(Y_test)  # Use extend instead of append

predicted_class_indices = [i.argmax() for i in y_prob]
actual_class_indices = [i.argmax() for i in y_act]

# Convert class indices to class labels
predicted_class = [list(train_generator.class_indices.keys())[idx]
for idx in predicted_class_indices]
actual_class = [list(train_generator.class_indices.keys())[idx] for
idx in actual_class_indices]

out_df = pd.DataFrame(np.vstack([predicted_class,
actual_class]).T, columns=['predicted_class', 'actual_class'])
confusion_matrix = pd.crosstab(out_df['actual_class'],
out_df['predicted_class'], rownames=['Actual'],
colnames=['Predicted'])

from sklearn.metrics import classification_report,
accuracy_score, precision_score, recall_score, f1_score

# Calculate various metrics
accuracy = accuracy_score(actual_class_indices,
predicted_class_indices)
precision = precision_score(actual_class_indices,
predicted_class_indices, average='weighted')
recall = recall_score(actual_class_indices,
predicted_class_indices, average='weighted')
f1 = f1_score(actual_class_indices, predicted_class_indices,
average='weighted')

# Create a DataFrame for the metrics
metrics_df = pd.DataFrame({
    'Accuracy': [accuracy * 100],
    'Precision': [precision],
    'Recall': [recall],
    'F1 Score': [f1]
})
```

Gambar 4.17 Kode kalkulasi metrik pengukuran

Seperti yang ditunjukkan pada gambar 4.17, sumbu x dan sumbu y pada *confusion matrix* diubah menjadi angka frekuensi non-tabuler. Hasil konversi ini menghasilkan dua variabel, *actual_class_indices* dan

predicted_class_indices. *actual_class_indices* merupakan hasil konversi dari frekuensi kelas sesungguhnya (aktual) dari pengujian, sedangkan *predicted_class_indices* merupakan hasil konversi dari frekuensi kelas yang terprediksi oleh model *Convolutional Neural Network* saat proses pengujian,

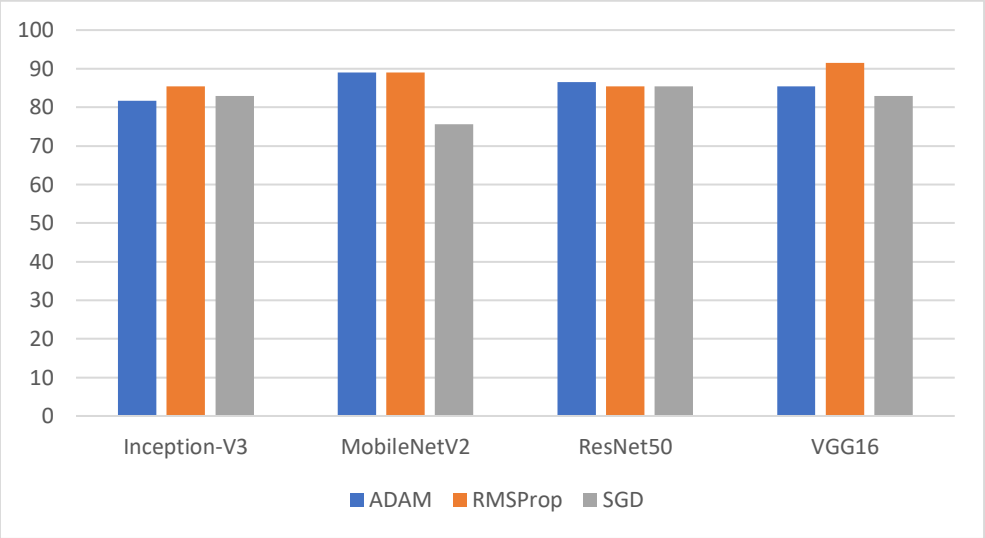
Library SciKit Learn menyediakan fungsi-fungsi untuk melakukan kalkulasi metrik-metrik pengukuran yang dibutuhkan untuk statistika, maupun evaluasi *machine learning* seperti akurasi, presisi, *recall*, dan *F1 score*. Fungsi *accuracy()* digunakan untuk mengkalkulasi akurasi model, fungsi *precision()* digunakan untuk menghitung presisi dari model tersebut, *recall()* berfungsi untuk perhitungan metrik *recall*, dan *f1_score()* adalah fungsi yang digunakan untuk mengkalkulasi nilai *F1 score* sebuah arsitektur *Convolutional Neural Network*.

4.2.2 Akurasi

Akurasi mengukur kebenaran model secara umum dengan cara menghitung rasio benar baik positif maupun negatif dibanding dengan jumlah sampel pengujian. Berikut merupakan hasil akurasi tiap model dengan masing-masing konfigurasi *optimizer*:

Tabel 4.8 Tabel akurasi model dengan konfigurasi optimizernya

Arsitektur	*ADAM*	*RMSProp*	*SGD*
Inception-V3	81,7073	85,3659	82,9268
MobileNetV2	89,0244	89,0244	75,6098
ResNet50	86,5854	85,3659	85,3659
VGG-16	85,3659	91,4634	82,9268



Gambar 4.18 Grafik perbandingan akurasi tiap model

Menurut tabel 4.8, arsitektur *VGG-16* dengan *optimizer RMSProp* mencapai akurasi tertinggi dengan nilai 91,46%. Hal ini merupkakan penurunan sebanyak 9,64% dibandingkan dengan penelitian oleh Xuefeng Liu dan kawan-kawan (2019) dalam klasifikasi binatang menggunakan model yang sama di mana penelitian sebelumnya mencapai akurasi sebesar 95%. Gambar 4.18 juga menunjukan bahwa arsitektur *Convolutional Neural Network* dan algoritma optimisasi merupakan faktor yang dapat mempengaruhi akurasi klasifikasi gambar. Arsitektur *Inception-V3* mendapatkan pengaruh terhadap performa model yang besar, hal ini dicerminkan pada akurasi tiap konfigurasi algoritma optimisasi yang memiliki selisih sebesar 17,07% antara nilai akurasi terendah dan akurasi tertinggi.

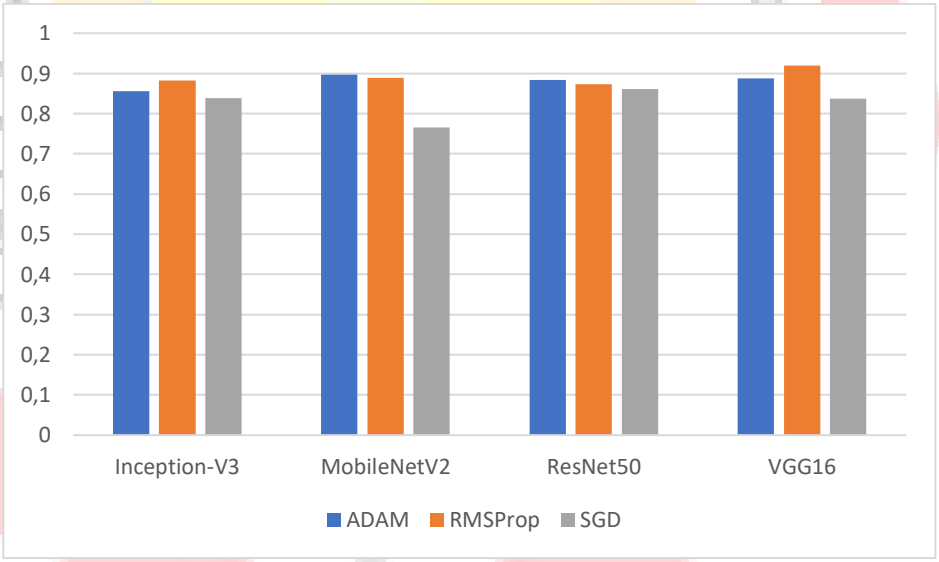
4.2.3 Presisi

Presisi mengindikasikan kualitas dari prediksi positif yang diberikan oleh model. Dengan kata lain, presisi menggambarkan berapa banyak identifikasi

positif dari model yang benar. Semakin tinggi nilai presisi, maka mengindikasikan model memiliki jumlah *false positive* yang rendah. Grafik di bawah ini mengilustrasikan perbandingan tingkat presisi setiap model berdasarkan konfigurasi *optimizer* yang digunakan:

Tabel 4.9 Tabel presisi model dengan konfigurasi optimizernya

Arsitektur	***ADAM***	***RMSProp***	***SGD***
Inception-V3	0,8563	0,8821	0,8379
MobileNetV2	0,8964	0,8889	0,7654
ResNet50	0,8841	0,8728	0,8610
VGG-16	0,8872	0,9200	0,8368



Gambar 4.19 Grafik perbandingan presisi tiap model

Berdasarkan tabel 4.9 dan grafik pada gambar 4.19, terlihat bahwa arsitektur *ResNet50* yang menggunakan *optimizer SGD* mencapai tingkat presisi tertinggi, mencapai nilai 0,8845, hal ini mengindikasikan model tersebut efektif dalam meminimalisir *false positive* dalam klasifikasinya. Gambar 4.16 juga mencerminkan bahwa faktor yang dapat memengaruhi

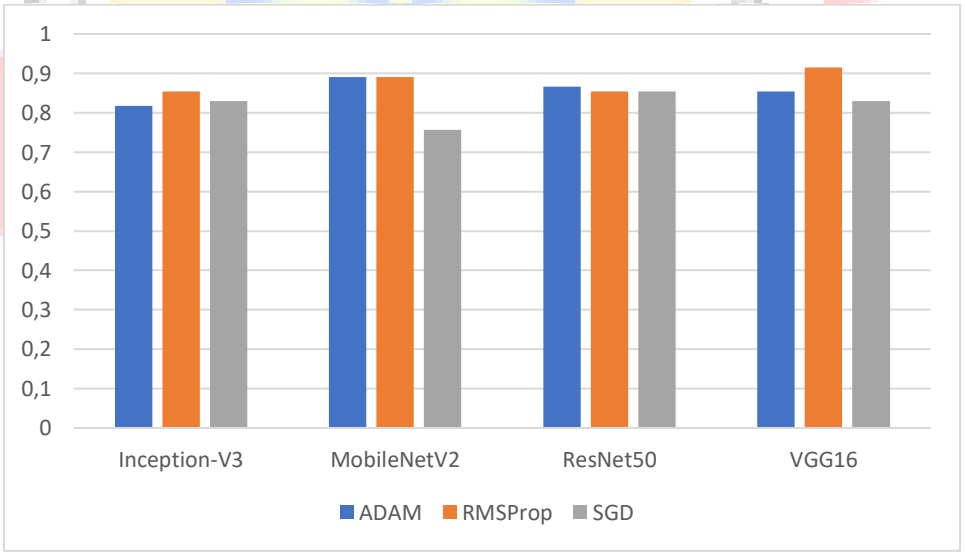
presisi klasifikasi gambar adalah arsitektur *Convolutional Neural Network* dan pilihan *optimizer*.

4.2.4 Recall

Recall mencerminkan sensitifitas model, *recall* mengukur berapa banyak positif yang diidentifikasi model dari keseluruhan kelas positif. Semakin tinggi nilai *recall*, maka mengindikasikan model memiliki jumlah *false negative* yang rendah. Grafik di bawah merupakan perbandingan nilai *recall* setiap model berdasarkan konfigurasi *optimizer* yang digunakan:

Tabel 4.10 Tabel nilai recall model dengan konfigurasi optimizernya

Arsitektur	*ADAM*	*RMSProp*	*SGD*
Inception-V3	0,8171	0,8537	0,8293
MobileNetV2	0,8902	0,8902	0,7561
ResNet50	0,8659	0,8537	0,8537
VGG-16	0,8537	0,9146	0,8293



Gambar 4.20 Grafik perbandingan nilai recall tiap model

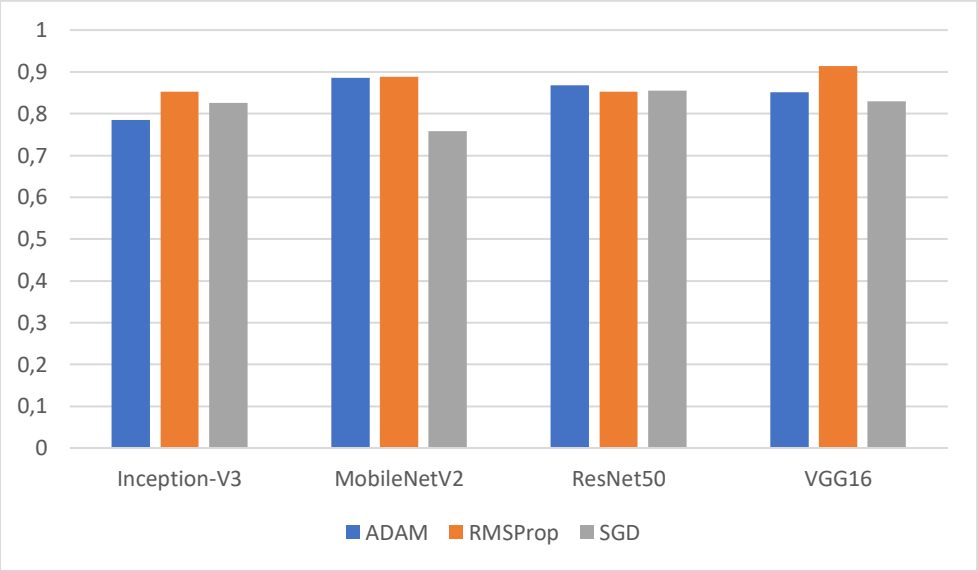
Dari tabel 4.10 dan juga visualisasi pada gambar 4.20, dapat diamati bahwa model *ResNet50* dengan *optimizer SGD* mencapai tingkat *recall* tertinggi, mencapai nilai 0,8659, menandakan model tersebut memiliki persentase tinggi dalam menangkap kelas positif pada klasifikasinya. Selain itu, gambar 4.20 juga menunjukkan bahwa faktor yang dapat memengaruhi *recall* klasifikasi gambar adalah arsitektur *Convolutional Neural Network* dan pilihan algoritma optimisasi.

4.2.5 F1 Score

F1 score merupakan *mean* harmonis dari presisi dan *recall*. Tingginya nilai *F1 score* mengindikasikan bahwa model memiliki presisi dan *recall* yang seimbang. Keseimbangan ini cukup penting Ketika distribusi data masing-masing kelas tidak sama rata. Grafik di bawah merupakan perbandingan nilai *F1 score* setiap model berdasarkan konfigurasi optimizer yang digunakan:

Tabel 4.11 Tabel F1 score model dengan konfigurasi optimizernya

Arsitektur	*ADAM*	*RMSProp*	*SGD*
Inception-V3	0,7845	0,8520	0,8255
MobileNetV2	0,8863	0,8886	0,7579
ResNet50	0,8679	0,8530	0,8552
VGG-16	0,8511	0,9142	0,8297



Gambar 4.21 Grafik perbandingan F1 score tiap model

Melalui tabel 4.11 dan representasi grafis pada gambar 4.21, terlihat bahwa model *ResNet50* dengan *optimizer SGD* berhasil mencapai *F1 score* paling optimal, dengan pencapaian nilai 0,8673, mengindikasikan bahwa model tersebut memiliki presisi dan *recall* yang sangat seimbang. Arsitektur *Inception-V3* cenderung memiliki *F1 score* yang rendah, mengimplikasikan bahwa presisi dan *recall* model tidak seimbang. Ketidakseimbangan tersebut dapat menyebabkan model terlalu berhati-hati dalam melakukan klasifikasi, meningkatkan resiko *false negative*. Atau kebalikannya, yaitu model melakukan klasifikasi terlalu bebas dan luas, yang meningkatkan resiko *false positive*. Selain itu, gambar 4.21 juga mencerminkan bahwa elemen-elemen seperti struktur *Convolutional Neural Network* dan seleksi algoritma optimisasi berperan penting dalam memengaruhi *F1 score* pada klasifikasi gambar.

4.2.6 Visualisasi *Heatmap*

Visualisasi persepsi *Convolutional Neural Network* menggunakan algoritma *GRAD-CAM* dapat memberitahu fitur atau titik mencolok yang dianggap penting oleh model dalam membuat identifikasi. Pada penelitian ini, dilakukan perbandingan persepsi antar model pada *convolutional layer* paling bawah guna mengevaluasi kemampuan tiap model dalam menangkap tiap ciri fisik masing-masing jenis ikan mas koki. Gambar 4.21 menjabarkan algoritma *GRAD-CAM* untuk menampilkan visualisasi *heatmap* untuk masing-masing model *Convolutional Neural Network*.

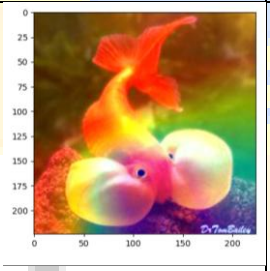
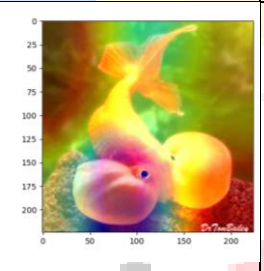
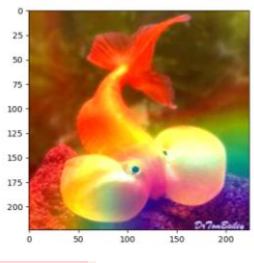
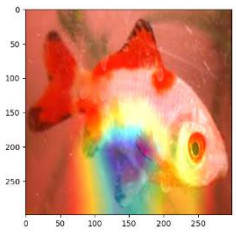
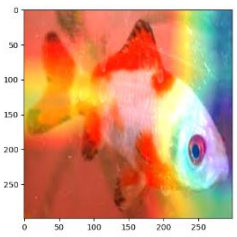
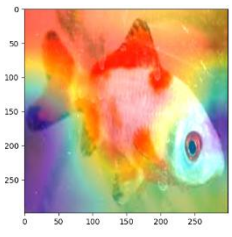
```
input = np.expand_dims(pixels, axis=0)
preds = model.predict(input)
print(species[np.argmax(preds[0])])
pred_index = preds

with tf.GradientTape() as tape:
    last_conv_layer = model.get_layer('conv2d_93')
    iterate = tf.keras.models.Model([model.inputs], [model.output, last_conv_layer.output])
    model_out, last_conv_layer = iterate(input)
    class_out = model_out[:, pred_index] # Use the pred_index instead of the argmax
    grads = tape.gradient(class_out, last_conv_layer)
    pooled_grads = K.mean(grads, axis=(0, 1, 2))

heatmap = tf.reduce_mean(tf.multiply(pooled_grads, last_conv_layer), axis=-1)
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
heatmap = heatmap.reshape((8, 8))

plt.matshow(heatmap)
plt.show()

INTENSITY = 0.5
raw = Image.open(img_path)
raw = raw.resize((299,299))
heatmap = cv2.resize(heatmap, (299, 299))
heatmap = cv2.applyColorMap(np.uint8(255*heatmap), cv2.COLORMAP_JET)
img = (heatmap * INTENSITY + np.array(raw))/255

# Display the original image and the overlay
plt.imshow(raw)
plt.show()
plt.imshow(img)
plt.show()
```

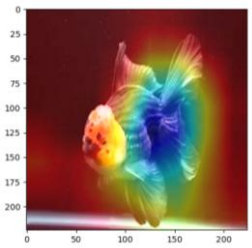
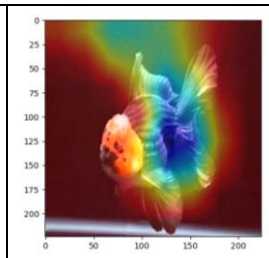
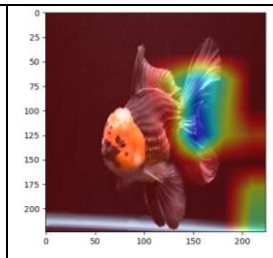
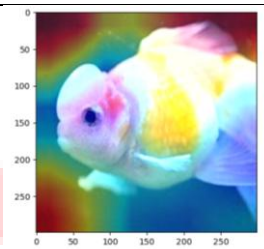
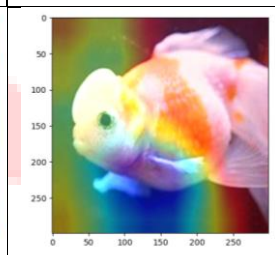
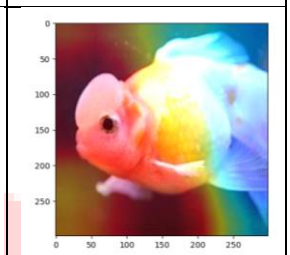
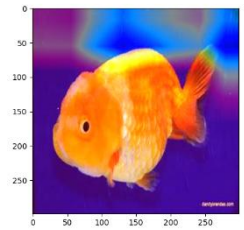
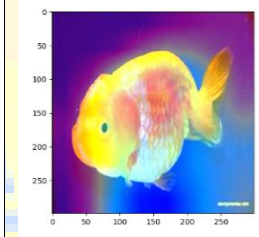
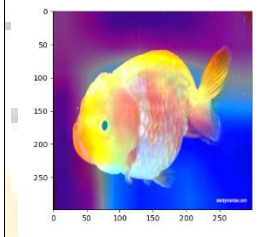
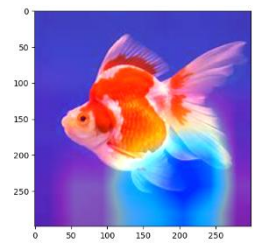
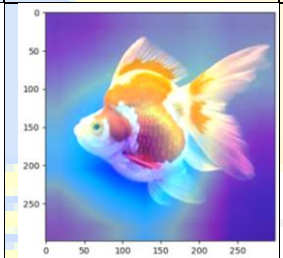
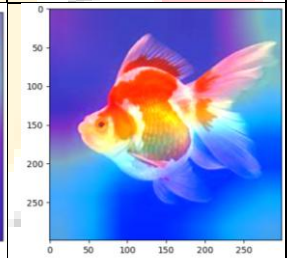
Gambar 4.22 Kode visualisasi GRAD-CAM pada Inception-v3

Kode pada gambar 4.21 menunjukkan bagaimana gambar diterima sebagai *input*, lalu kemudian dilakukannya proses klasifikasi menggunakan

arsitektur *Convolutional Neural Network* yang akan divisualisasikan. Setelah itu, diambillah lapisan konvolusi terakhir pada arsitektur tersebut. Dalam kasus ini, arsitektur *Inception-v3* memiliki lapisan konvolusi terakhir yang dinamakan "*conv2d_93*". Kemudian, dilakukanlah perhitungan *mean* antara lapisan konvolusi akhir dengan gradient kelas. Hasil perhitungan ini akan menghasilkan *heatmap* yang masih belum ternormalisasi. Normalisasikan *heatmap* tersebut, lalu dilakukan proses *overlay* di mana *heatmap* ditimpa kepada gambar sampel awal. Berikut merupakan hasil visualisasi *GRAD-CAM* masing-masing model pada tiap variasi ikan mas koki dengan tiga konfigurasi *optimizer*:

Tabel 4.12 Visualisasi Inception-V3 terhadap tiap variasi ikan mas koki menggunakan masing-masing optimizer

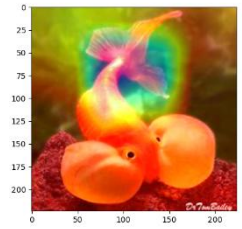
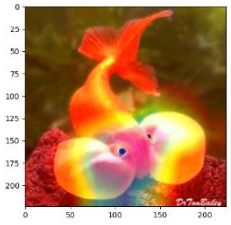
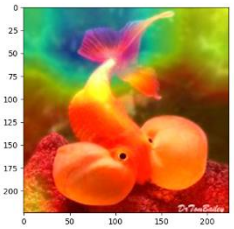
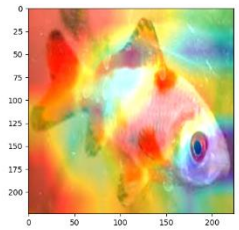
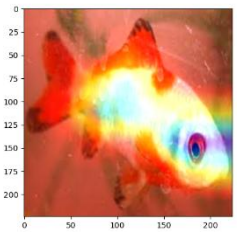
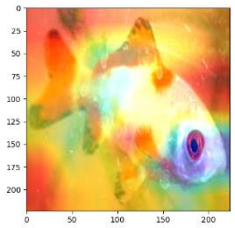
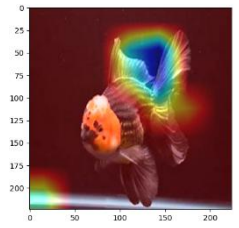
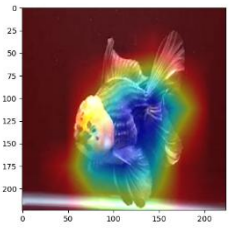
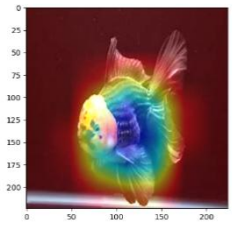
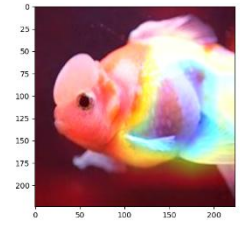
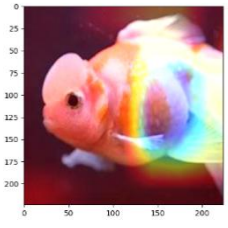
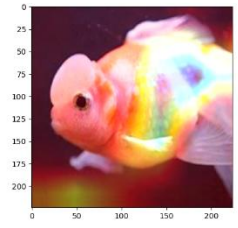
Kelas	*Inception-V3 + ADAM*	*Inception-V3 + SGD*	*Inception-V3 + RMSProp*
Bubble eyes			
Comet			

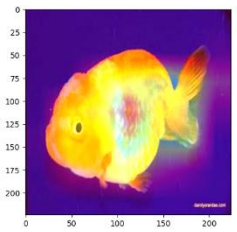
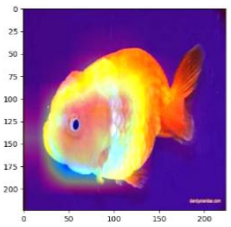
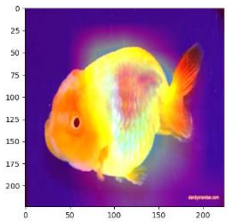
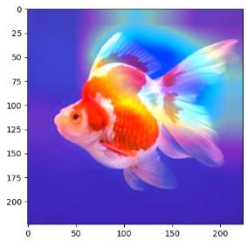
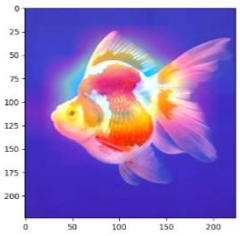
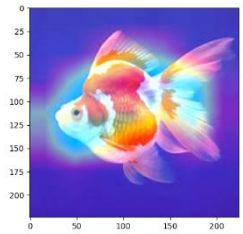
Oranda			
Pearlscale			
Ranchu			
Ryukin			

Tabel 4.12 menjabarkan perbandingan visualisasi *GRAD-CAM* antara model *Inception V3* yang dilatih oleh algoritma optimisasi *ADAM, RMSProp,* dan *SGD*. Ketiga model mengenali tipe ikan mas koki melalui cara yang berbeda-beda. Contohnya, pada jenis *Comet*, model yang dilatih dengan *ADAM* mengidentifikasi tipe ikan tersebut melalui perut bagian bawahnya yang panjang dan pipih, model yang dilatih dengan *SGD* mengenali *Comet* dari bentuk kepalanya, sedangkan *RMSProp* mengenali *Comet* dari bentuk tubuhnya yang panjang dan juga ekornya. Hal yang sama juga dapat diamati

pada tipe *Pearlscale*, di mana *ADAM* dan *SGD* mengenali tipe *Pearlscale* dari bentuk kepalanya, sedangkan model yang dilatih dengan *optimizer RMSProp* mengidentifikasi tipe tersebut melalui tubuhnya.

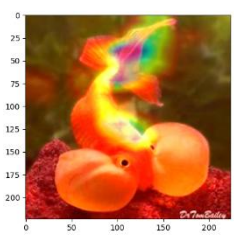
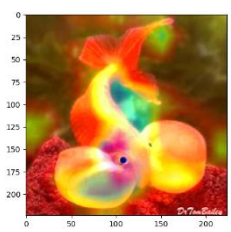
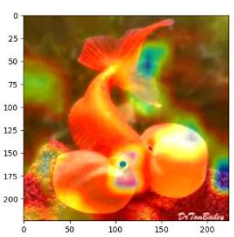
Tabel 4.13 Visualisasi ResNet50 terhadap tiap variasi ikan mas koki menggunakan masing-masing optimizer

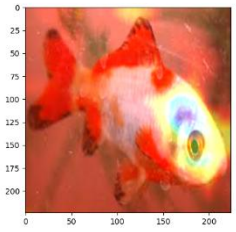
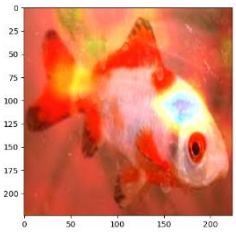
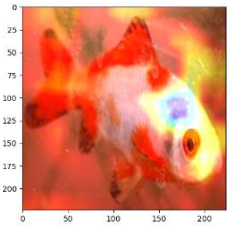
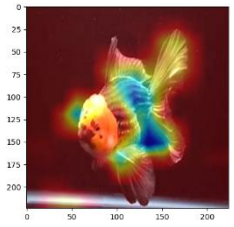
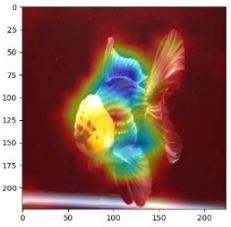
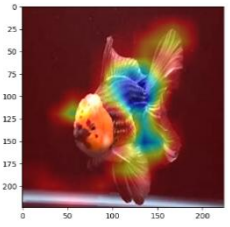
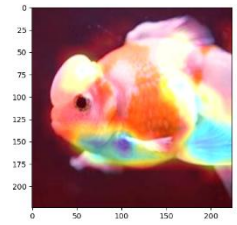
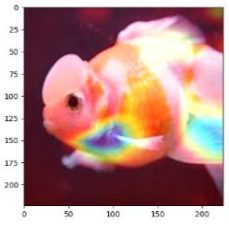
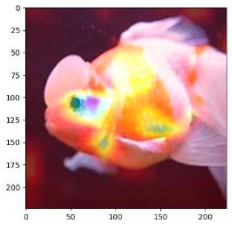
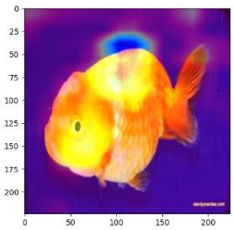
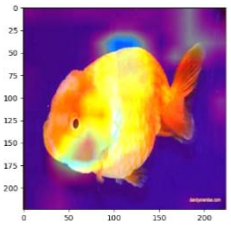
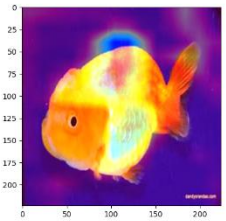
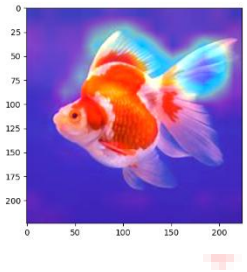
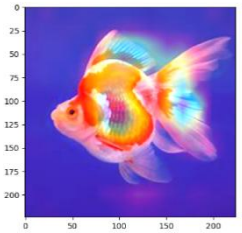
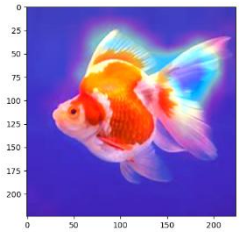
Kelas	ResNet 50 + *ADAM*	ResNet 50 + *SGD*	ResNet 50 + *RMSProp*
Bubble eyes			
Comet			
Oranda			
Pearlscale			

Ranchu			
Ryukin			

Tabel 4.13 menjabarkan perbandingan visualisasi *GRAD-CAM* antara model *ResNet50* yang dilatih oleh algoritma optimisasi *ADAM, RMSProp,* dan *SGD*. Model yang dilatih menggunakan optimisasi *ADAM* dan *RMSProp* mengidentifikasi tipe *Bubble eyes* dari bentuk ekornya, sedangkan *SGD* mengenali tipe tersebut melalui kantung matanya. Selain itu, model yang dilatih oleh *optimizer ADAM* mengenali tipe *Oranda* dari sirip renang atasnya (*dorsal fin*), sedangkan kedua model lainnya mengenali bentuk tuibuhnya.

Tabel 4.14 Visualisasi VGG-16 terhadap tiap variasi ikan mas koki menggunakan masing-masing optimizer

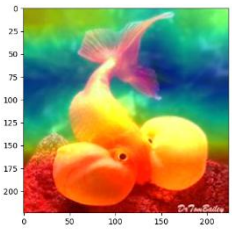
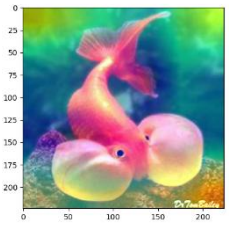
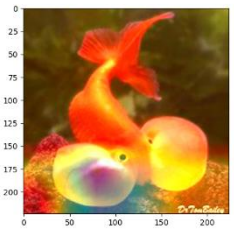
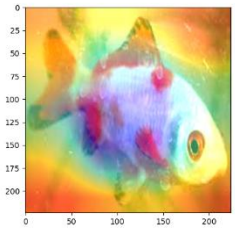
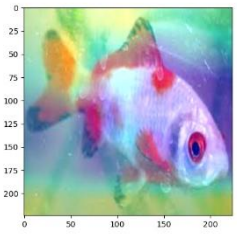
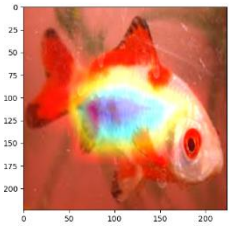
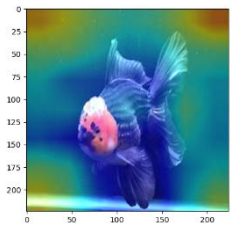
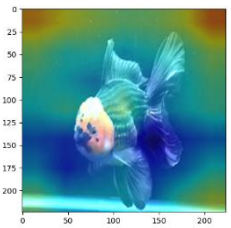
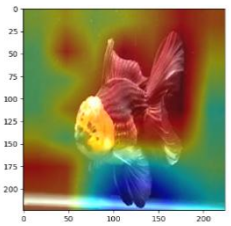
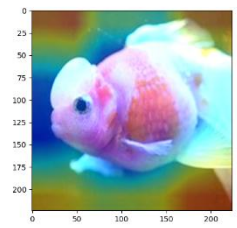
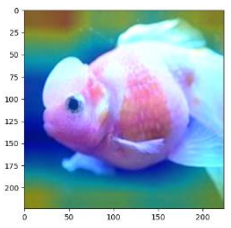
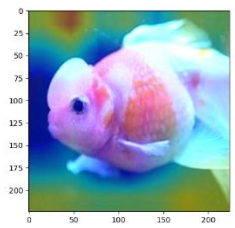
Kelas	*VGG-16 + ADAM*	*VGG-16 + SGD*	*VGG-16 + RMSProp*
Bubble eyes			

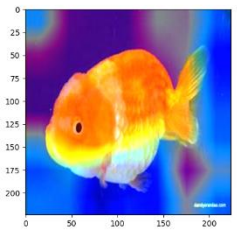
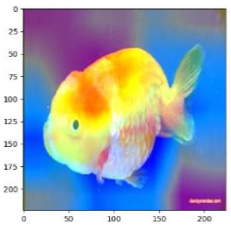
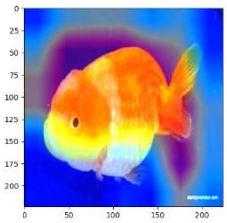
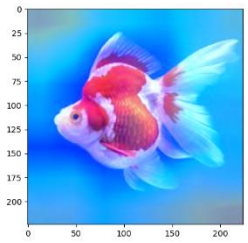
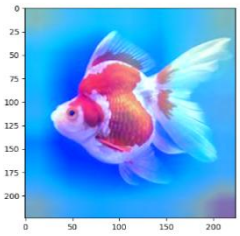
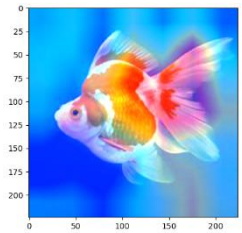
Comet			
Oranda			
Pearlscale			
Ranchu			
Ryukin			

Tabel 4.14 menjabarkan perbandingan visualisasi *GRAD-CAM* antara model *VGG-16* yang dilatih oleh algoritma optimisasi *ADAM, RMSProp,* dan *SGD*. Kedua model yang dilatih oleh *ADAM* dan *SGD* mengidentifikasi tipe

Pearlscale melalui perut bagian bawah ikan tersebut, sedangkan model yang dilatih *RMSProp* mengidentifikasi tubuh dan sisiknya.

Tabel 4.15 Visualisasi MobileNetV2 terhadap tiap variasi ikan mas koki menggunakan masing-masing optimizer

Kelas	*MobileNetV2 + ADAM*	*MobileNetV2 + SGD*	*MobileNetV2 + RMSProp*
Bubble eyes			
Comet			
Oranda			
Pearlscale			

Ranchu			
Ryukin			

Tabel 4.15 menjabarkan perbandingan visualisasi *GRAD-CAM* antara model *MobileNetV2* yang dilatih oleh algoritma optimisasi *ADAM*, *RMSProp*, dan *SGD*. Dibandingkan dengan ketiga arsitektur yang telah dibahas sebelumnya, lingkup identifikasi arsitektur *MobileNetV2* lebih tersebar. Hal ini menyebabkan *heatmap* yang lebih lebar dan luas. Hal ini mengimplikasikan bahwa arsitektur *MobileNetV2* merupakan model yang kurang ideal untuk divisualisasikan melalui metode *GRAD-CAM*.

Seperti yang digambarkan pada tabel 4.12 hingga 4.15, tidak hanya tiap model memiliki persepsi dan penangkapan fitur yang berbeda, algoritma *optimizer* yang digunakan pun dapat mempengaruhi visualisasi *GRAD-CAM* secara signifikan. Hal ini mengimplikasikan bahwa *learning rate* memiliki pengaruh yang besar dalam kemampuan model untuk menangkap ciri unik pada gambar.

4.2.7 Arsitektur Terpilih

Dari evaluasi model yang telah dilakukan, arsitektur *VGG-16* yang dilatih menggunakan algoritma optimisasi *RMSProp* dinilai sebagai model

yang memiliki performa tertinggi dari 12 model yang telah dilatih dan diuji. Pada evaluasi *confusion matrix*, model ini memiliki persentase *false positive* dan *false negative* yang cenderung rendah. Tidak hanya itu, model ini memiliki performa tertinggi pada akurasi, presisi, *recall*, serta *F1-score* dibandingkan dengan model lainnya. Pada visualisasi *GRAD-CAM*, model mampu mengenali ciri-ciri fisik berbagai jenis ikan mas koki seperti sirip atas, ekor, maupun bentuk badan.

4.3 Implementasi *Website*

4.3.1 Integrasi *SQLite*

Pengimplementasian *database* pada *website* dilakukan dengan mengimplementasikan *SQLite* dengan cara membuat file *database* yang kemudian akan dikoneksikan menggunakan fungsi koneksi *database* yang dijabarkan oleh gambar 4.19.

```
fishdb = 'goldfishdb.db'

def get_db():
    db = getattr(g, '_database', None)
    if db is None:
        db = g._database = sql.connect(fishdb)
    return db

@app.teardown_appcontext
def close_connection(exception):
    db = getattr(g, '_database', None)
    if db is not None:
        db.close()
```

Gambar 4.23 Kode koneksi database pada website

Pada gambar 4.23, fungsi *get_db()* digunakan untuk menghubungkan *database* kepada *website*, sedangkan fungsi *close_connection()* digunakan untuk menutup koneksi setelah *request* dari pengguna untuk mencegah

kebocoran data. Untuk menggunakan *database* yang terhubung, digunakan fungsi *get_db().cursor()* untuk mengeksekusikan *query SQL*. Contoh penggunaan *query* ditunjukkan pada gambar 4.20.

```
cur = get_db().cursor()
cur.execute('SELECT * FROM goldfishtable')
articles = cur.fetchall()
```

Gambar 4.24 Kode pemanggilan query SQL

Kode pada gambar 4.24 akan mengeksekusi *query* yang akan menyajikan seluruh data dari tabel *goldfishtable* yang kemudian dikirim ke *website* menggunakan fungsi *cur.fetchall().*

4.3.2 Integrasi *Convolutional Neural Network*

Untuk mengintegrasikan model *Convolutional Neural Network* pada *website*, dibuat sebuah fungsi bernama getPrediction() yang mana fungsi ini akan memuat model *Convolutional Neural Network* yang telah dikompilasi ke dalam format file .h5, lalu fungsi ini akan memuat gambar yang telah diunggah oleh pengguna untuk dilakukan langkah *resizing* dan *pre-processing* menggunakan fungsi preprocess_input untuk mengubah gambar menjadi bentuk yang dapat diterima oleh model. Setelah itu, model akan melakukan proses klasifikasi yang hasil label dan juga persentase akurasinya akan dijadikan *return* dari fungsi tersebut Kode yang digunakan dalam fungsi getPrediction dijabarkan pada gambar 4.25.

```
def getPrediction(filename):
    model = load_model('checkpoint_VGG-16_SGD.h5')

    image = load_img('static/uploads/' + filename, target_size=(224, 224))
    image = img_to_array(image)
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    image = preprocess_input(image)

    yhat = model.predict(image)

    class_index = yhat.argmax()
    label = class_labels[class_index]
    probability = yhat[0][class_index] * 100  # Convert to percentage

    print(f'{label} ({probability:.2f}%)')

    return label, probability
```

Gambar 4.25 Kode untuk fungsi getPrediction

4.3.3 Routing Website

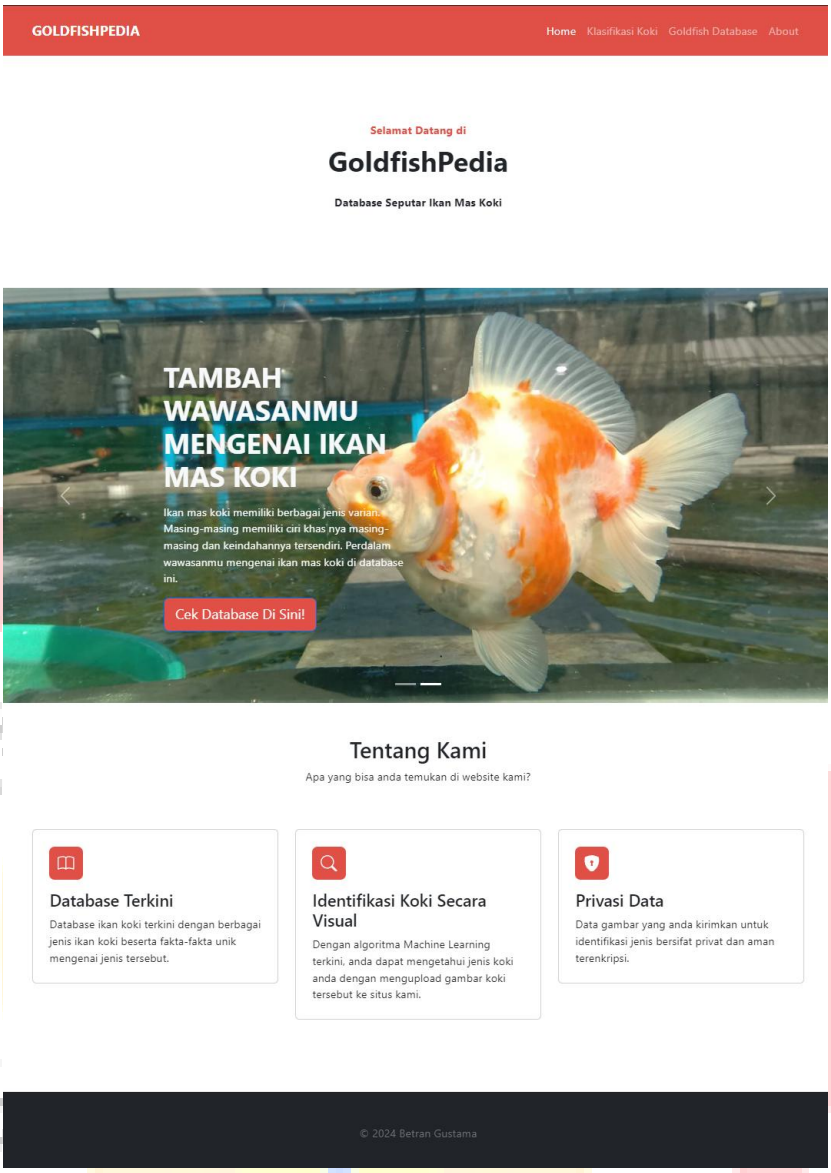
Routing merupakan proses yang dilakukan untuk memetakan rute *website* yang dapat diakses oleh pengguna berupa *URL*. Tanpa adanya *routing*, pengguna tidak akan dapat mengakses sebuah halaman *website*. Pada *flask*, proses *routing* dilakukan menggunakan *decorator* bernama "@app.route()" di dalam *decorator* tersebut, terdapat fungsi yang akan menampilkan halaman web yang dispesifikasikan. Contoh *routing* pada halaman index ditunjukkan pada gambar 4.26.

```
@app.route('/')
def index():
    return render_template('index.html')
```

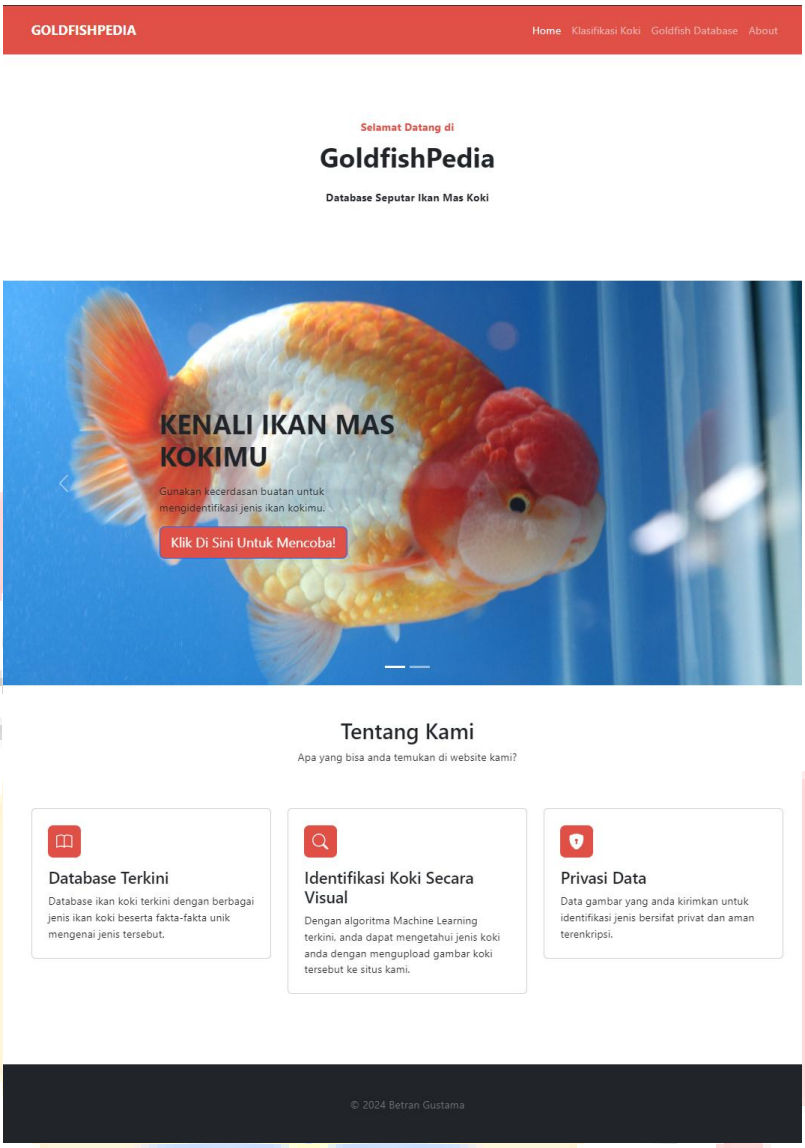
Gambar 4.26 Routing pada halaman utama

4.3.4 Tampilan Halaman Utama

Halaman utama atau *landing page* merupakan halaman yang tampil ketika pengguna mengunjungi *website*. Halaman ini merupakan pengenalan berbagai fitur yang dapat diakses oleh pengguna. Pada bagian atas halaman, terdapat *navbar* yang dapat menjadi fasilitas navigasi utama pengguna untuk mengakses berbagai fitur *website* secara cepat. Pada isi *landing page* terdapat ucapan selamat dating kepada user dan juga deskripsi fitur dari *website* yang disajikan dengan karousel yang mendeskripsikan fitur dari klasifikasi koki dan juga fitur *goldfish database* beserta tombol pintas untuk mengakses halaman itu secara langsung melalui karousel. Karousel ini dapat bergulir setelah beberapa detik, atau dengan meng-klik tombol *"next"* dan *"back"* pada sebelah kiri dan kanan karousel. Di bawah karousel, terdapat tiga deskripsi singkat mengenai *website* yang meliputi kelengkapan *database* ikan mas koki, penggunaan *machine learning* untuk membantu pengguna mengidentifikasi ikan mas koki melalui gambar yang diupload, serta bagaimana data yang diunggah oleh pengguna bersifat privat dan hanya digunakan dalam klasifikasi gambar saja. Gambar 4.27 dan 4.28 menunjukkan tampilan utama dari *landing page* dengan dua kondisi karousel di mana pada gambar 4.27 karousel sedang mendeskripsikan fitur *database* ikan mas koki, sedangkan gambar 4.28 menunjukan karousel yang sedang mendeskripsikan fitur identifikasi koki.



Gambar 4.27 Tampilan halaman utama (karousel a)

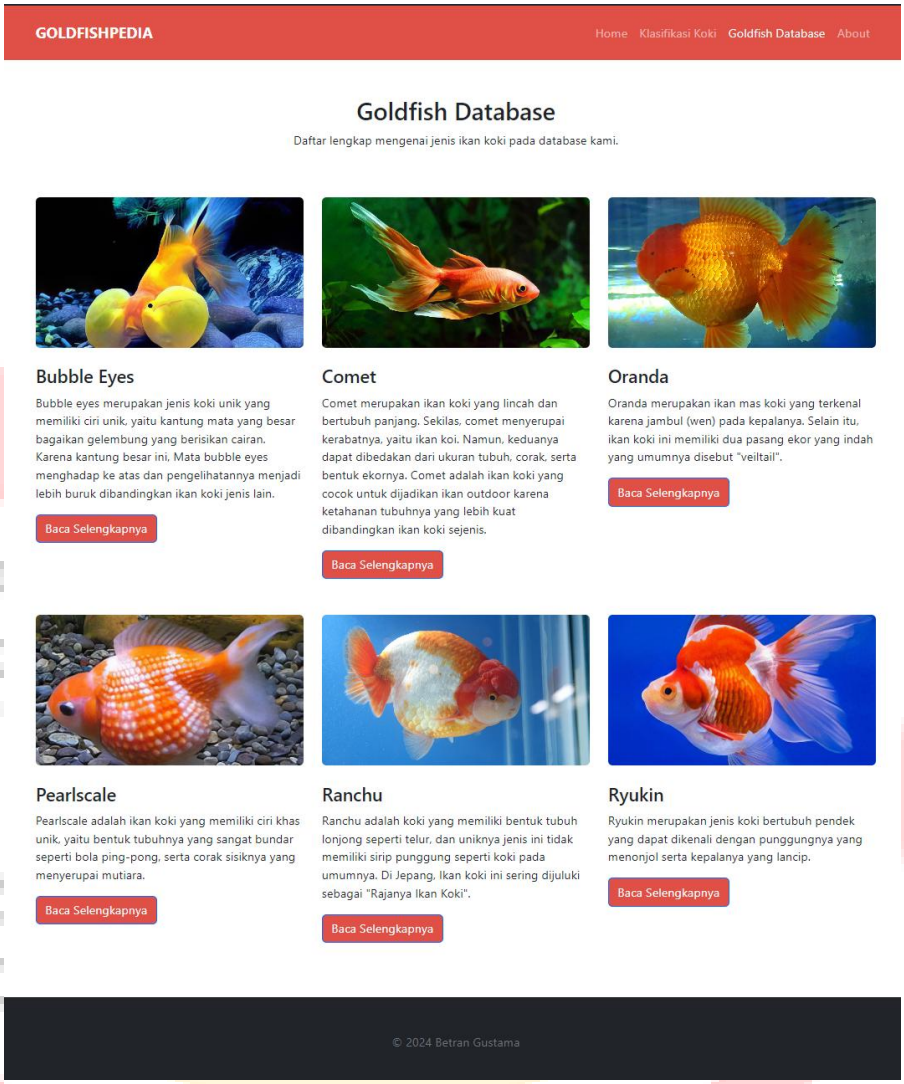


Gambar 4.28 Tampilan halaman utama (karousel b)

4.3.5 Tampilan Halaman *Database* Ikan Mas Koki

Goldfish database atau *database* ikan mas koki merupakan halaman yang menyajikan daftar jenis ikan mas koki yang dapat diakses informasinya pada situs ini. Masing-masing jenis ikan mas koki akan ditampilkan nama umumnya, gambar jenis ikan mas koki tersebut, deskripsi singkat mengenai jenis tersebut, serta tombol navigasi untuk mengalihkan pengguna ke halaman *entry* seperti yang dibahas pada poin 4.3.7. Gambar 4.29 menunjukkan

tampilan dari *database* ikan mas koki yang menyajikan enam jenis ikan koki yang dapat diakses informasinya.



Gambar 4.29 Tampilan halaman database ikan mas koki

4.3.6 Tampilan Halaman Klasifikasi Koki

Halaman klasifikasi koki merupakan halaman di mana pengguna dapat mengklasifikasi jenis ikan mas koki menggunakan gambar unggahan mereka sebagai input. Sesuai yang disajikan pada gambar 4.30, pada bagian atas halaman, terdapat bilah navigasi yang dapat menjadi sarana navigasi utama pengguna untuk mengakses berbagai fitur *website* secara cepat. Pada isi halaman, terdapat *file selector* yang ketika diklik akan membuka dialog untuk

memilih file gambar yang akan diunggah ke *website*. Di samping *file selector*, terdapat tombol "submit" yang akan mengunggah file tersebut, kemudian memproses gambar yang nantinya akan diklasifikasikan jenisnya oleh *Convolutional Neural Network*. Ketika *CNN* selesai mengklasifikasi gambar tersebut, maka pengguna akan dialihkan ke halaman *entry* seperti yang disajikan pada poin 4.3.7, sesuai dengan jenis yang diklasifikasikan oleh *CNN* sebelumnya. Tak hanya itu, halaman ini juga menyajikan sampel gambar ikan mas koki yang dapat digunakan apabila pengguna tidak memiliki gambar ikan koki yang tersedia.

Gambar 4.30 Tampilan halaman klasifikasi ikan mas koki

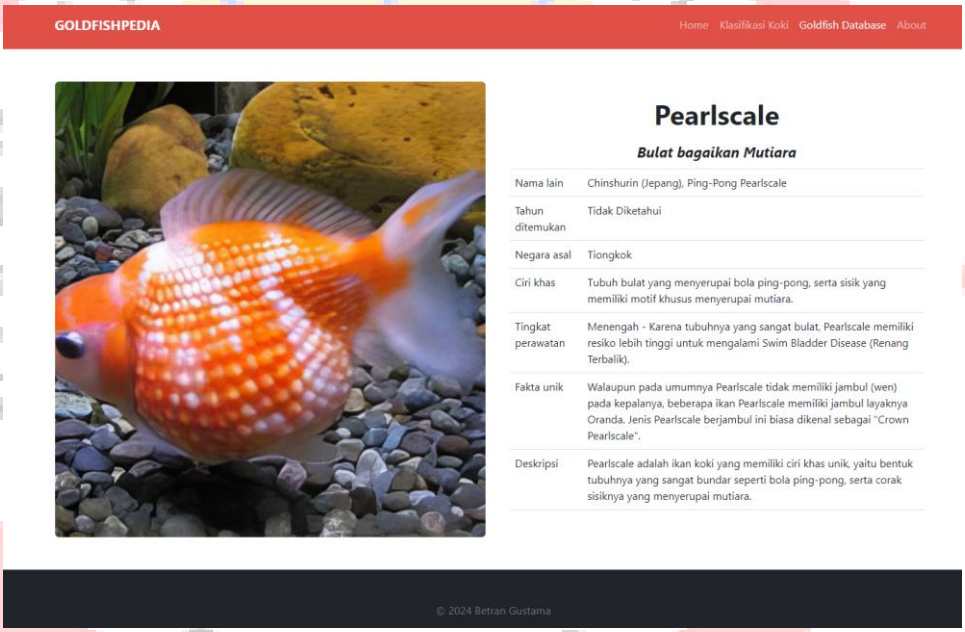
4.3.7 Tampilan Halaman *Entry*

Halaman *entry* merupakan halaman yang menyajikan secara spesifik informasi mengenai jenis ikan mas koki. Halaman ini dapat diakses melalui *database* ikan koki maupun melalui proses klasifikasi koki. Informasi yang disajikan berupa:

1. Nama jenis ikan mas koki.
2. Gambar ilustrasi jenis ikan mas koki.

3. Nama lain dari berbagai negara.
4. Tahun ditemukannya jenis ikan mas koki tersebut.
5. Negara asal jenis tersebut.
6. Tingkat kesulitan dalam perawatan serta tips mengenai perawatan jenis ikan mas koki tersebut.
7. *Trivia* atau fakta unik mengenai jenis tersebut.
8. Deskripsi singkat mengenai jenis ikan koki.

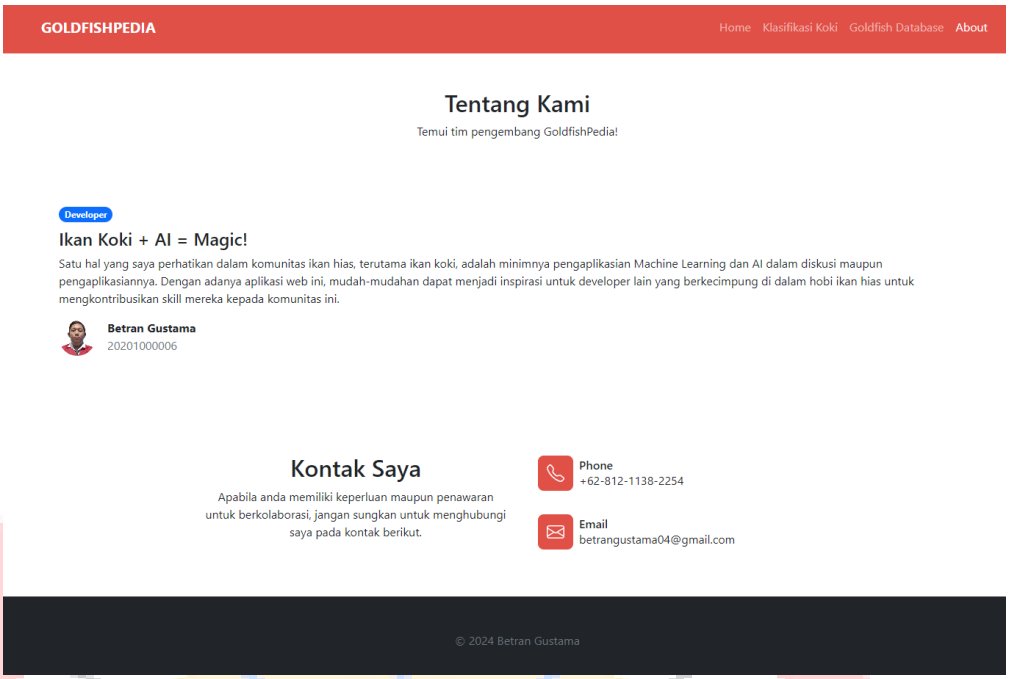
Gambar 4.31 menunjukan salah satu contoh *entry* dari jenis ikan mas koki berupa informasi mengenai ikan koki jenis *Pearlscale*.



Gambar 4.31 Tampilan halaman entry ikan mas koki

4.3.8 Tampilan Halaman *About*

Halaman *about* merupakan halaman yang menyajikan informasi mengenai pengembang *website*. Isi dari halaman ini adalah komentar singkat dari pengembang, serta kontak yang dapat dihubungi berupa nomor telepon dan juga Alamat email. Gambar 4.32 menunjukkan tampilan dari halaman *about*.



Gambar 4.32 Tampilan halaman about

4.4 Evaluasi *Black Box*

Evaluasi *Black Box* digunakan dalam menguji fungsi-fungsi pada implementasi *website* demi memastikan *website* bekerja sesuai dengan ekspetasi pengguna. Pengujian ini mencakup berbagai fungsi *website* seperti navigasi, integrasi model *Convolutional Neural Network* pada halaman *website*, serta integrasi *database* kepada *website* melalui *SQLite*. Tabel 4.16 menjabarkan hasil pengujian *Black Box* secara detail.

Tabel 4.16 Rincian evaluasi Black Box

No.	Deskripsi Uji	Input Uji	Output semestinya	Output yang dihasilkan	Hasil Uji
1	Menampiklan halaman utama	Memasukkan *URL website* pada *browser*	Halaman utama ditampilkan	Halaman utama ditampilkan	Lolos
2	Pengujian bilah navigasi	Tombol pada bilah navigasi diklik	Tombol navigasi mengarahkan	Tombol navigasi mengarahkan	Lolos

			pengguna ke halaman terpilih	pengguna ke halaman terpilih	
3	Menampilkan karousel halaman utama	Mengakses halaman utama	Karousel berganti dari karousel A menjadi karousel B setelah karousel tampil selama 10 detik atau ketika ditekan tombol *"next"* dan *"back"*	Karousel berganti dari karousel A menjadi karousel B setelah karousel tampil selama 10 detik atau ketika ditekan tombol *"next"* dan *"back"*	Lolos
4	Menampilkan halaman *goldfish database*	Mengakses halaman melalui tombol navigasi	Halaman menampilkan daftar ikan koki pada *database*	Halaman menampilkan daftar ikan koki pada *database*	Lolos
5	Menampilkan halaman klasifikasi koki	Mengakses halaman melalui tombol navigasi	Halaman klasifikasi koki berhasil ditampilkan	Halaman klasifikasi koki berhasil ditampilkan	Lolos
6	Menguji tombol form *upload*	Tombol *file selector* diklik dan dipilih file yang sesuai, kemudian klik tombol "*submit*"	File berhasil terunggah kepada direktori penyimpanan gambar pada *server*	File berhasil terunggah kepada direktori penyimpanan gambar pada *server*	Lolos

7	Menguji fungsi identifikasi ikan mas koki menggunakan *Convolutional Neural Network*	Tombol *file selector* diklik dan dipilih file yang sesuai, kemudian klik tombol "*submit*"	Pengguna diarahkan ke halaman *entry* dari jenis ikan mas koki yang teridentifikasi	Pengguna diarahkan ke halaman *entry* dari jenis ikan mas koki yang teridentifikasi	Lolos
8	Menampilkan halaman *entry*	Mengakses halaman melalui *goldfish database* dan melalui metode klasifikasi koki	Halaman *entry* ikan mas koki berhasil ditampilkan	Halaman *entry* ikan mas koki berhasil ditampilkan	Lolos
9	Menampilkan halaman *about*	Mengakses halaman melalui tombol navigasi	Halaman *about* koki berhasil ditampilkan	Halaman *about* koki berhasil ditampilkan	Lolos

4.5 Pembahasan

Penelitian ini menguji dua belas kombinasi model *Convolutional Neural Network* beserta dengan algoritma optimisasi guna menemukan kombinasi yang efektif dalam mengidentifikasi ikan mas koki. Sebagian besar kombinasi membuahkan hasil yang memuaskan dengan 11 dari 12 kombinasi mencapai nilai akurasi di atas 80%, dengan satu di antaranya mencapai skor di atas 90%. Arsitektur *VGG-16* yang dilatih menggunakan algoritma optimisasi *RMSProp* merupakan kombinasi model dan *optimizer* dengan performa tertinggi dari ke 12 kombinasi yang dilatih serta diuji.

Hasil evaluasi menunjukkan bahwa *VGG-16* dengan *optimizer RMSProp* memiliki performa yang kompetitif dengan penelitian sebelumnya dari nilai akurasi,

presisi, *recall*, serta *F1 score*. Adapun perbandingan hasil penelitian ini dibandingkan dengan penelitian-penelitian sebelumnya yang juga menggunakan *VGG-16* dijabarkan pada tabel 4.17.

Tabel 4.17 Perbandingan hasil penelitian dengan penelitian sebelumnya

Penelitian	Jumlah Dataset	Kelas	Akurasi
Penelitian ini	765	6	91,46%
(Iqbal *et al.*, 2021)	1334	6	89,97%
(Montalbo & Hernandez, 2019)	530	3	99%
(Banan *et al.*, 2020)	409	4	100%
(Ye *et al.*, 2020)	30000	5	81,40%
(Hasan *et al.*, 2021)	4000	4	97,25%
(Auliasari *et al.*, 2023)	9000	9	99%

Berdasarkan tabel 4.17, penelitian ini berhasil mencapai akurasi tertinggi ketiga dari lima penelitian yang menggunakan arsitektur serupa. Penelitian oleh Banan *et al.* (2020) berhasil mencapai akurasi sempurna, yaitu 100%. Hasil ini dicapai menggunakan sampel yang lebih sedikit, serta metode akusisi dataset yang terkontrol, sedangkan penelitian ini menggunakan dataset yang lebih bervariasi dari segi sudut pengambilan, pencahayaan, serta luasnya variasi warna dari ikan mas koki dibandingkan dengan ikan liar yang diteliti oleh penelitian sebelumnya. Sementara itu, penelitian oleh Ye *et al.* (2020) mencapai akurasi terendah (81,40%) dengan jumlah dataset terbanyak, sebesar 30 ribu sampel dataset. Hal ini mengindikasikan bahwa jumlah dataset tidak selalu meningkatkan performa model dalam mengidentifikasi objek.

Proses integrasi model terpilih ke dalam *website* berhasil dilakukan dengan hasil yang sesuai dengan ekspetasi. Pengguna dapat mengidentifikasi ikan mas koki

berdasarkan gambar yang diinput oleh pengguna. Akan tetapi, penelitian ini masih memiliki beberapa keterbatasan seperti jumlah sampel dataset yang lebih sedikit dan tidak seimbang dibandingkan dengan penelitian sebelumnya. Tentunya hal ini berdampak terhadap kemampuan tiap model dalam mengidentifikasi ikan mas koki. Beberapa kombinasi model dan *optimizer* tidak dirancang untuk beradaptasi terhadap ketidakseimbangan data tersebut secara optimal, sehingga muncullah kecenderungan atau *bias* pada proses klasifikasi gambar model tersebut. Terlepas dari keterbatasan tersebut, penelitian ini dinilai telah mendemonstrasikan kelebihan dan kekurangan setiap kombinasi model *Convolutional Neural Network* dan *optimizer* yang dapat dijadikan referensi untuk penelitian selanjutnya, atau peningkatan lebih lanjut yang dapat mengatasi keterbatasan pada penelitian ini.