

Analisis Emosi Anjing Melalui Klasifikasi Citra untuk Deteksi Ekspresi Wajah Hewan Peliharaan

I Gede Surya Adi Pradana^{a1}, I Gede Santi Astawa^{a2}

^aProgram Studi Informatika, Fakultas Matematika dan Ilmu Pengetahuan Alam,
Universitas Udayana
Jalan Raya Kampus UNUD, Bukit Jimbaran, Kuta Selatan, Badung, Bali, Indonesia
¹pradana.2208561126@student.unud.ac.id
²santi.astawa@unud.ac.id

Abstract

Emotion Detection in Dogs through Image Classification for Pet Facial Expression Analysis utilizes convolutional neural networks (CNNs) to discern and quantify dogs' emotional states based on their facial expressions. This approach leverages advanced image processing techniques to recognize subtle cues indicative of various emotions like happiness, fear, or sadness in dogs. By applying CNNs to analyze image data, pet owners can gain valuable insights into their dogs' emotional well-being and address their needs accordingly. This technology offers a promising avenue for enhancing the bond between pets and owners by facilitating more informed and empathetic care tailored to each dog's emotional state and personality.

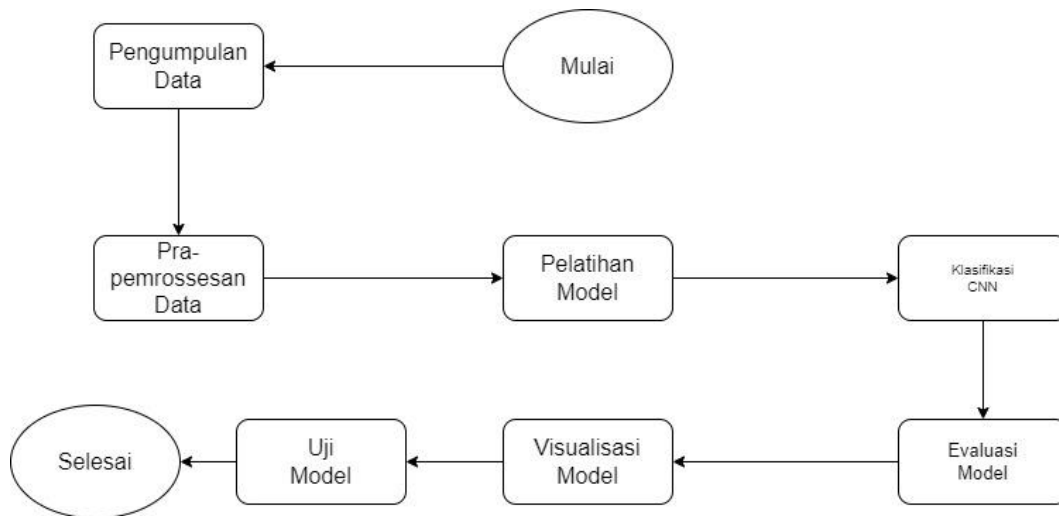
Keywords: Emotion Recognition, Dog Facial Expressions, Image Classification, Convolutional Neural Networks (CNNs), Pet Well-Being

1. Pendahuluan

Dalam dekade terkini, perkembangan teknologi telah mengarah pada peningkatan minat terhadap aplikasi kecerdasan buatan (AI) dalam berbagai bidang, termasuk ilmu komputer dan kedokteran hewan. Salah satu bidang yang menarik perhatian adalah penggunaan AI untuk menganalisis dan memahami ekspresi emosional hewan peliharaan, seperti anjing. Anjing, sebagai hewan peliharaan yang paling umum di dunia, sering kali menjadi anggota keluarga dan mendapatkan perhatian khusus dari pemiliknya. Penelitian sebelumnya telah menunjukkan bahwa anjing memiliki kemampuan unik untuk mengekspresikan emosi melalui ekspresi wajah mereka. Namun, interpretasi dan pemahaman terhadap ekspresi-ekspresi ini masih merupakan tantangan bagi banyak pemilik hewan. Dalam konteks ini, penggunaan teknologi AI, khususnya teknik klasifikasi citra dengan menggunakan jaringan saraf konvolusional (CNNs) [1], menawarkan potensi untuk meningkatkan pemahaman kita terhadap keadaan emosional anjing melalui analisis ekspresi wajah. Penelitian ini bertujuan untuk mengembangkan sistem yang dapat mendeteksi dan mengklasifikasikan ekspresi emosional pada anjing berdasarkan gambar wajah mereka. Dengan memanfaatkan kemajuan dalam bidang visi komputer dan pembelajaran mesin, teknik ini akan memberikan pemahaman yang lebih dalam tentang bagaimana anjing mengungkapkan perasaan mereka melalui wajah mereka. Keberhasilan proyek ini memiliki implikasi yang luas, termasuk memberikan pemilik hewan peliharaan wawasan yang lebih baik tentang kesejahteraan emosional anjing mereka. Informasi ini dapat membantu pemilik untuk merespons kebutuhan emosional hewan peliharaan mereka secara lebih efektif dan sensitif. Selain itu, penelitian ini juga dapat membuka jalan bagi pengembangan teknologi yang dapat digunakan dalam lingkungan klinik veteriner untuk membantu diagnosis dan perawatan hewan yang lebih baik. Dalam konteks lebih luas, penelitian ini juga mendorong pengembangan AI yang lebih humanistik, di mana teknologi tidak hanya digunakan untuk tujuan praktis tetapi juga untuk meningkatkan kualitas hidup dan hubungan antara manusia dan hewan peliharaan mereka. Dengan demikian, penelitian ini diharapkan dapat memberikan kontribusi positif terhadap bidang kecerdasan buatan dan kesejahteraan hewan secara keseluruhan.

2. Metode Penelitian

2.1. Alur Penelitian



Gambar 1. Alur penelitian

Penelitian ini dimulai dengan pengumpulan dataset citra ekspresi wajah anjing yang telah diberi label *angry* dan *happy*. Dataset ini terdiri dari berbagai gambar anjing dengan ekspresi wajah yang mencerminkan kedua emosi tersebut. Selanjutnya, data diproses dan dibagi secara acak menjadi dua bagian: data pelatihan untuk melatih model klasifikasi, dan data pengujian untuk mengevaluasi kinerja model. Selanjutnya, teknik klasifikasi citra menggunakan jaringan saraf konvolusional (CNN) diterapkan pada data pelatihan. CNN dilatih untuk mempelajari pola dan fitur khas dari citra anjing dengan ekspresi *angry* dan *happy*. Proses pelatihan dilakukan berulang kali untuk meningkatkan kemampuan model dalam mengenali perbedaan antara kedua ekspresi emosional tersebut.[2]. Langkah terakhir adalah evaluasi model klasifikasi menggunakan data pengujian yang terpisah. Metrik evaluasi seperti akurasi, presisi, dan recall digunakan untuk mengukur sejauh mana model dapat mengenali dan mengklasifikasikan ekspresi emosional pada citra wajah anjing[3]. Hasil evaluasi ini memberikan wawasan tentang kinerja model dan potensinya dalam menganalisis ekspresi emosional pada hewan peliharaan, khususnya anjing.

2.2. Pengumpulan Data

Dataset yang digunakan dalam proyek ini merupakan data sekunder yang diperoleh dari Kaggle, sebuah platform yang menyediakan berbagai dataset untuk keperluan machine learning dan analisis data. Dataset ini terdiri dari 800 gambar anjing yang dikategorikan berdasarkan dua emosi utama, yaitu "angry" dan "happy", dengan masing-masing kategori berisi 400 gambar. Pemilihan dataset ini bertujuan untuk membangun dan melatih model klasifikasi gambar yang dapat mengenali ekspresi emosional pada wajah anjing. Penggunaan dataset dari Kaggle dipilih karena dataset tersebut sudah terstruktur dengan baik dan menyediakan variasi gambar yang cukup, sehingga sangat cocok untuk keperluan pelatihan model deep learning.

2.3. Preprocessing Data

Data yang telah dikumpulkan mengalami penyesuaian ukuran dari ukuran asli yang beragam menjadi 150×150 piksel. Selanjutnya, dilakukan proses Augmentasi Data. Augmentasi Data adalah proses dalam pengolahan gambar yang bertujuan untuk meningkatkan variasi data dengan mengubah atau memodifikasi gambar sehingga komputer tetap mengenali gambar tersebut sebagai gambar yang sama. Pada penelitian ini, augmentasi data mencakup beberapa teknik: rotasi gambar secara acak hingga sudut maksimum 40°, pergeseran horizontal dan vertikal, shear atau distorsi gambar, zoom-in dan zoom-out, serta flipping horizontal. Selain itu,

semua gambar dinormalisasi dengan mereskalakan nilai pixel ke rentang 0-1. Proses-proses ini bertujuan untuk membuat model lebih tangguh terhadap berbagai variasi dalam data input, sehingga meningkatkan kemampuan generalisasi model saat dihadapkan dengan data baru.[4]

2.4. Pelatihan Model

Pelatihan model dilakukan setelah tahap preprocessing data selesai. Model yang digunakan adalah model Convolutional Neural Network (CNN), yang terdiri dari beberapa lapisan (layer) konvolusi dan pooling untuk ekstraksi fitur, serta lapisan fully connected untuk klasifikasi akhir. Pada penelitian ini, model CNN dimulai dengan lapisan konvolusi dengan 32 filter dan kernel berukuran 3x3, diikuti dengan lapisan pooling untuk mengurangi dimensi fitur. Proses ini dilanjutkan dengan beberapa lapisan konvolusi dan pooling tambahan, masing-masing dengan jumlah filter yang semakin meningkat (64 dan 128 filter), untuk menangkap fitur yang lebih kompleks. Setelah itu, lapisan flatten digunakan untuk mengubah matriks fitur menjadi vektor, yang kemudian diproses oleh beberapa lapisan fully connected, termasuk lapisan dropout untuk mengurangi overfitting, dan diakhiri dengan lapisan output menggunakan fungsi aktivasi softmax untuk menghasilkan probabilitas dari dua kelas (angry dan happy). Model ini dilatih menggunakan data yang telah di-augmentasi, dengan menggunakan optimizer Adam dan fungsi loss categorical crossentropy. Pelatihan dilakukan selama 20 epoch, dengan data validasi digunakan untuk memantau kinerja model dan menyesuaikan bobot secara bertahap. Selama pelatihan, kinerja model dievaluasi berdasarkan metrik akurasi dan loss baik pada data latih maupun data validasi. Hasil pelatihan menunjukkan peningkatan akurasi dan penurunan loss seiring bertambahnya epoch, yang menggambarkan kemampuan model dalam belajar dari data latih dan menggeneralisasikan pengetahuannya ke data validasi.

2.5. Klasifikasi Citra Dengan CNN

Klasifikasi citra dengan CNN adalah proses di mana model yang telah dilatih digunakan untuk mengidentifikasi kategori gambar baru. Dalam program ini, CNN yang telah dilatih pada gambar anjing dengan emosi "angry" dan "happy" digunakan untuk memprediksi emosi dari gambar baru. Gambar baru ini terlebih dahulu diproses untuk penyesuaian ukuran dan normalisasi, sama seperti data latih. Model kemudian mengeluarkan probabilitas untuk masing-masing kategori, dan kategori dengan probabilitas tertinggi menjadi hasil prediksi. Proses ini memungkinkan identifikasi otomatis dari emosi "angry" atau "happy" pada gambar anjing berdasarkan pola-pola yang telah dipelajari selama pelatihan. [5]

2.6. Evaluasi

Evaluasi model CNN dilakukan untuk mengevaluasi kinerja model dalam mengklasifikasikan gambar baru setelah dilatih. Terdapat beberapa metrik evaluasi yang umum digunakan, di antaranya:

- Akurasi (Accuracy): Metrik yang mengukur seberapa banyak prediksi yang benar dibuat oleh model dibandingkan dengan total jumlah prediksi.

$$\text{Accuracy} = \frac{\text{Total prediksi benar}}{\text{Total prediksi}}$$

- Loss Function: Fungsi yang mengukur seberapa baik model menghasilkan prediksi. Untuk klasifikasi multikelas, umumnya menggunakan cross-entropy loss function.

$$\text{Loss} = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^C y_{ij} \log(P_{ij})$$

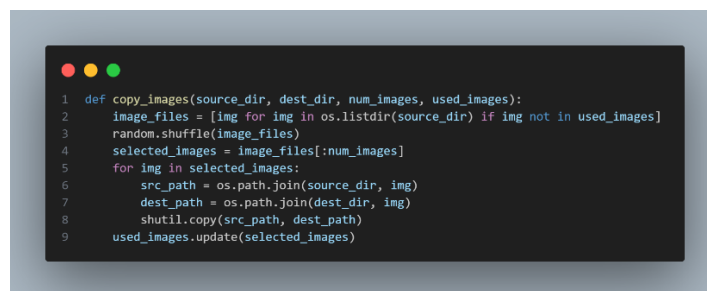
- N adalah jumlah sampel data
- C adalah jumlah kelas
- y_{ij} adalah label yang sebenarnya untuk sampel i dan kelas j
- P_{ij} adalah probabilitas model untuk sampel i dan kelas j

- c. Confusion Matrix: Matriks yang menunjukkan jumlah prediksi yang benar dan salah untuk setiap kelas.
- d. Precision, Recall, dan F1-score: Metrik evaluasi yang lebih rinci yang mengukur performa model untuk setiap kelas secara individual.

Evaluasi dilakukan menggunakan data validasi yang tidak pernah dilihat oleh model selama pelatihan. Dengan metrik-metrik tersebut, kita dapat memahami seberapa baik model CNN dapat mengidentifikasi emosi pada gambar anjing. Dalam kode, akurasi dan loss biasanya langsung tersedia sebagai output dari proses pelatihan model. Sedangkan untuk confusion matrix, precision, recall, dan F1-score, kita bisa menggunakan fungsi-fungsi dari library seperti scikit-learn di Python.

3. Hasil dan Diskusi

Bagian pertama yang menangani pembagian data terletak dalam fungsi `copy_images`, di mana gambar-gambar dari direktori sumber (angry_dir dan happy_dir) disalin ke direktori tujuan (train_angry, train_happy, validation_angry, validation_happy) dengan jumlah tertentu sesuai dengan jumlah data yang telah ditentukan. Selanjutnya, dilakukan inialisasi generator untuk data latih (train_generator) dan data validasi (validation_generator) menggunakan ImageDataGenerator. Proses ini sangat penting untuk mempersiapkan dataset sebelum proses pelatihan model dimulai. Penggunaan generator memungkinkan augmentasi data dilakukan secara dinamis selama proses pelatihan, yang membantu meningkatkan ketahanan dan kinerja model terhadap variasi data yang belum pernah dilihat sebelumnya.



```
1 def copy_images(source_dir, dest_dir, num_images, used_images):
2     image_files = [img for img in os.listdir(source_dir) if img not in used_images]
3     random.shuffle(image_files)
4     selected_images = image_files[:num_images]
5     for img in selected_images:
6         src_path = os.path.join(source_dir, img)
7         dest_path = os.path.join(dest_dir, img)
8         shutil.copy(src_path, dest_path)
9     used_images.update(selected_images)
```

Gambar 2. Pembagian Data

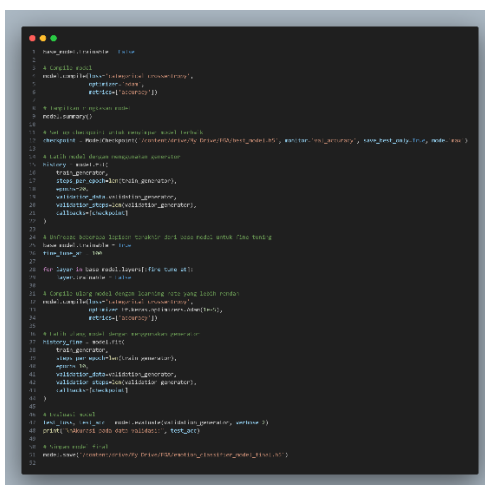
Tahap augmentasi dilakukan menggunakan ImageDataGenerator pada generator data latih. Dalam kode yang disediakan, train_datagen diinisialisasi dengan berbagai transformasi seperti rotasi, pergeseran horizontal dan vertikal, pemotongan, perbesaran, dan pencerminan. Berikut adalah inialisasi train_datagen



```
1 train_datagen = ImageDataGenerator(
2     rescale=1./255,
3     rotation_range=40,
4     width_shift_range=0.2,
5     height_shift_range=0.2,
6     shear_range=0.2,
7     zoom_range=0.2,
8     horizontal_flip=True,
9     fill_mode='nearest'
10 )
```

Gambar 3. Augmentasi

Kemudian, dibuatlah generator data latih (train_generator) menggunakan train_datagen dengan parameter-parameter yang telah ditentukan sebelumnya. Generator ini akan menghasilkan batch-batch citra yang telah dimodifikasi secara acak sesuai dengan transformasi yang telah ditentukan sebelumnya. Pendekatan ini memungkinkan model untuk melihat variasi yang lebih luas dari data latih yang tersedia, yang pada gilirannya membantu meningkatkan kemampuan model untuk menggeneralisasi pola dari data yang belum pernah dilihat sebelumnya. Ini adalah strategi penting dalam memperbaiki kinerja model dan mencegah overfitting. Selanjutnya merupakan proses pelatihan model menggunakan metode transfer learning dengan menggunakan base model VGG16. Pertama, model disetel agar lapisan-lapisan dasar tidak dapat di-train ulang dengan mengatur base_model.trainable = False. Kemudian, model dikompilasi dengan optimizer Adam dan fungsi loss categorical_crossentropy. Selanjutnya, model dilatih selama 20 epoch menggunakan generator data latih dan data validasi. Setelah itu, beberapa lapisan terakhir dari base model di-"unfreeze" untuk fine-tuning dengan mengatur base_model.trainable = True, dan lapisan-lapisan yang tidak ingin di-train kembali diatur ke False. Model kemudian dikompilasi ulang dengan learning rate yang lebih rendah, dan dilatih kembali selama 10 epoch. Akhirnya, model dievaluasi menggunakan data validasi, akurasi hasil evaluasi dicetak ke layar, dan model final disimpan ke dalam file.



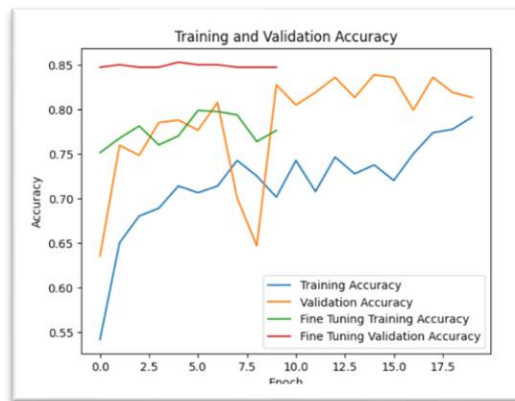
```
1 from keras.preprocessing import image
2 from keras.preprocessing.image import ImageDataGenerator
3 from keras.applications.vgg16 import VGG16
4 from keras.optimizers import Adam
5 from keras.losses import categorical_crossentropy
6 from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
7 from keras import metrics
8 import os
9 import numpy as np
10 import random
11 import sys
12 import time
13
14 # Load the VGG16 model
15 base_model = VGG16(weights='imagenet')
16
17 # Freeze the base model
18 for layer in base_model.layers:
19     layer.trainable = False
20
21 # Compile the model
22 model = base_model
23 optimizer = Adam()
24 model.compile(optimizer=optimizer, loss=categorical_crossentropy,
25             metrics=['accuracy'])
26
27 # Create the training and validation data generators
28 train_datagen = ImageDataGenerator(
29     rotation_range=20,
30     width_shift_range=0.1,
31     height_shift_range=0.1,
32     shear_range=0.1,
33     zoom_range=0.1,
34     horizontal_flip=True,
35     fill_mode='nearest')
36
37 validation_datagen = ImageDataGenerator(
38     rotation_range=0,
39     width_shift_range=0,
40     height_shift_range=0,
41     shear_range=0,
42     zoom_range=0,
43     horizontal_flip=False,
44     fill_mode='nearest')
45
46 # Create the training and validation data loaders
47 train_generator = train_datagen.flow_from_directory(
48     'train',
49     target_size=(224, 224),
50     batch_size=32)
51
52 validation_generator = validation_datagen.flow_from_directory(
53     'validation',
54     target_size=(224, 224),
55     batch_size=32)
56
57 # Train the model
58 model.fit_generator(train_generator,
59                   validation_data=validation_generator,
60                   epochs=20,
61                   validation_steps=100,
62                   callbacks=[ModelCheckpoint('model.h5'),
63                             EarlyStopping(monitor='val_loss',
64                                           patience=10,
65                                           verbose=1)])
66
67 # Load the model
68 model = VGG16(weights='imagenet')
69
70 # Compile the model
71 optimizer = Adam()
72 model.compile(optimizer=optimizer, loss=categorical_crossentropy,
73             metrics=['accuracy'])
74
75 # Create the training and validation data loaders
76 train_datagen = ImageDataGenerator(
77     rotation_range=20,
78     width_shift_range=0.1,
79     height_shift_range=0.1,
80     shear_range=0.1,
81     zoom_range=0.1,
82     horizontal_flip=True,
83     fill_mode='nearest')
84
85 validation_datagen = ImageDataGenerator(
86     rotation_range=0,
87     width_shift_range=0,
88     height_shift_range=0,
89     shear_range=0,
90     zoom_range=0,
91     horizontal_flip=False,
92     fill_mode='nearest')
93
94 # Create the training and validation data loaders
95 train_generator = train_datagen.flow_from_directory(
96     'train',
97     target_size=(224, 224),
98     batch_size=32)
99
100 validation_generator = validation_datagen.flow_from_directory(
101     'validation',
102     target_size=(224, 224),
103     batch_size=32)
104
105 # Train the model
106 model.fit_generator(train_generator,
107                   validation_data=validation_generator,
108                   epochs=10,
109                   validation_steps=100,
110                   callbacks=[ModelCheckpoint('model.h5'),
111                             EarlyStopping(monitor='val_loss',
112                                           patience=10,
113                                           verbose=1)])
114
115 # Load the model
116 model = VGG16(weights='imagenet')
117
118 # Compile the model
119 optimizer = Adam()
120 model.compile(optimizer=optimizer, loss=categorical_crossentropy,
121             metrics=['accuracy'])
122
123 # Create the training and validation data loaders
124 train_datagen = ImageDataGenerator(
125     rotation_range=20,
126     width_shift_range=0.1,
127     height_shift_range=0.1,
128     shear_range=0.1,
129     zoom_range=0.1,
130     horizontal_flip=True,
131     fill_mode='nearest')
132
133 validation_datagen = ImageDataGenerator(
134     rotation_range=0,
135     width_shift_range=0,
136     height_shift_range=0,
137     shear_range=0,
138     zoom_range=0,
139     horizontal_flip=False,
140     fill_mode='nearest')
141
142 # Create the training and validation data loaders
143 train_generator = train_datagen.flow_from_directory(
144     'train',
145     target_size=(224, 224),
146     batch_size=32)
147
148 validation_generator = validation_datagen.flow_from_directory(
149     'validation',
150     target_size=(224, 224),
151     batch_size=32)
152
153 # Train the model
154 model.fit_generator(train_generator,
155                   validation_data=validation_generator,
156                   epochs=10,
157                   validation_steps=100,
158                   callbacks=[ModelCheckpoint('model.h5'),
159                             EarlyStopping(monitor='val_loss',
160                                           patience=10,
161                                           verbose=1)])
162
163 # Load the model
164 model = VGG16(weights='imagenet')
165
166 # Compile the model
167 optimizer = Adam()
168 model.compile(optimizer=optimizer, loss=categorical_crossentropy,
169             metrics=['accuracy'])
170
171 # Create the training and validation data loaders
172 train_datagen = ImageDataGenerator(
173     rotation_range=20,
174     width_shift_range=0.1,
175     height_shift_range=0.1,
176     shear_range=0.1,
177     zoom_range=0.1,
178     horizontal_flip=True,
179     fill_mode='nearest')
180
181 validation_datagen = ImageDataGenerator(
182     rotation_range=0,
183     width_shift_range=0,
184     height_shift_range=0,
185     shear_range=0,
186     zoom_range=0,
187     horizontal_flip=False,
188     fill_mode='nearest')
189
190 # Create the training and validation data loaders
191 train_generator = train_datagen.flow_from_directory(
192     'train',
193     target_size=(224, 224),
194     batch_size=32)
195
196 validation_generator = validation_datagen.flow_from_directory(
197     'validation',
198     target_size=(224, 224),
199     batch_size=32)
200
201 # Train the model
202 model.fit_generator(train_generator,
203                   validation_data=validation_generator,
204                   epochs=10,
205                   validation_steps=100,
206                   callbacks=[ModelCheckpoint('model.h5'),
207                             EarlyStopping(monitor='val_loss',
208                                           patience=10,
209                                           verbose=1)])
210
211 # Load the model
212 model = VGG16(weights='imagenet')
213
214 # Compile the model
215 optimizer = Adam()
216 model.compile(optimizer=optimizer, loss=categorical_crossentropy,
217             metrics=['accuracy'])
218
219 # Create the training and validation data loaders
220 train_datagen = ImageDataGenerator(
221     rotation_range=20,
222     width_shift_range=0.1,
223     height_shift_range=0.1,
224     shear_range=0.1,
225     zoom_range=0.1,
226     horizontal_flip=True,
227     fill_mode='nearest')
228
229 validation_datagen = ImageDataGenerator(
230     rotation_range=0,
231     width_shift_range=0,
232     height_shift_range=0,
233     shear_range=0,
234     zoom_range=0,
235     horizontal_flip=False,
236     fill_mode='nearest')
237
238 # Create the training and validation data loaders
239 train_generator = train_datagen.flow_from_directory(
240     'train',
241     target_size=(224, 224),
242     batch_size=32)
243
244 validation_generator = validation_datagen.flow_from_directory(
245     'validation',
246     target_size=(224, 224),
247     batch_size=32)
248
249 # Train the model
250 model.fit_generator(train_generator,
251                   validation_data=validation_generator,
252                   epochs=10,
253                   validation_steps=100,
254                   callbacks=[ModelCheckpoint('model.h5'),
255                             EarlyStopping(monitor='val_loss',
256                                           patience=10,
257                                           verbose=1)])
258
259 # Load the model
260 model = VGG16(weights='imagenet')
261
262 # Compile the model
263 optimizer = Adam()
264 model.compile(optimizer=optimizer, loss=categorical_crossentropy,
265             metrics=['accuracy'])
266
267 # Create the training and validation data loaders
268 train_datagen = ImageDataGenerator(
269     rotation_range=20,
270     width_shift_range=0.1,
271     height_shift_range=0.1,
272     shear_range=0.1,
273     zoom_range=0.1,
274     horizontal_flip=True,
275     fill_mode='nearest')
276
277 validation_datagen = ImageDataGenerator(
278     rotation_range=0,
279     width_shift_range=0,
280     height_shift_range=0,
281     shear_range=0,
282     zoom_range=0,
283     horizontal_flip=False,
284     fill_mode='nearest')
285
286 # Create the training and validation data loaders
287 train_generator = train_datagen.flow_from_directory(
288     'train',
289     target_size=(224, 224),
290     batch_size=32)
291
292 validation_generator = validation_datagen.flow_from_directory(
293     'validation',
294     target_size=(224, 224),
295     batch_size=32)
296
297 # Train the model
298 model.fit_generator(train_generator,
299                   validation_data=validation_generator,
300                   epochs=10,
301                   validation_steps=100,
302                   callbacks=[ModelCheckpoint('model.h5'),
303                             EarlyStopping(monitor='val_loss',
304                                           patience=10,
305                                           verbose=1)])
306
307 # Load the model
308 model = VGG16(weights='imagenet')
309
310 # Compile the model
311 optimizer = Adam()
312 model.compile(optimizer=optimizer, loss=categorical_crossentropy,
313             metrics=['accuracy'])
314
315 # Create the training and validation data loaders
316 train_datagen = ImageDataGenerator(
317     rotation_range=20,
318     width_shift_range=0.1,
319     height_shift_range=0.1,
320     shear_range=0.1,
321     zoom_range=0.1,
322     horizontal_flip=True,
323     fill_mode='nearest')
324
325 validation_datagen = ImageDataGenerator(
326     rotation_range=0,
327     width_shift_range=0,
328     height_shift_range=0,
329     shear_range=0,
330     zoom_range=0,
331     horizontal_flip=False,
332     fill_mode='nearest')
333
334 # Create the training and validation data loaders
335 train_generator = train_datagen.flow_from_directory(
336     'train',
337     target_size=(224, 224),
338     batch_size=32)
339
340 validation_generator = validation_datagen.flow_from_directory(
341     'validation',
342     target_size=(224, 224),
343     batch_size=32)
344
345 # Train the model
346 model.fit_generator(train_generator,
347                   validation_data=validation_generator,
348                   epochs=10,
349                   validation_steps=100,
350                   callbacks=[ModelCheckpoint('model.h5'),
351                             EarlyStopping(monitor='val_loss',
352                                           patience=10,
353                                           verbose=1)])
354
355 # Load the model
356 model = VGG16(weights='imagenet')
357
358 # Compile the model
359 optimizer = Adam()
360 model.compile(optimizer=optimizer, loss=categorical_crossentropy,
361             metrics=['accuracy'])
362
363 # Create the training and validation data loaders
364 train_datagen = ImageDataGenerator(
365     rotation_range=20,
366     width_shift_range=0.1,
367     height_shift_range=0.1,
368     shear_range=0.1,
369     zoom_range=0.1,
370     horizontal_flip=True,
371     fill_mode='nearest')
372
373 validation_datagen = ImageDataGenerator(
374     rotation_range=0,
375     width_shift_range=0,
376     height_shift_range=0,
377     shear_range=0,
378     zoom_range=0,
379     horizontal_flip=False,
380     fill_mode='nearest')
381
382 # Create the training and validation data loaders
383 train_generator = train_datagen.flow_from_directory(
384     'train',
385     target_size=(224, 224),
386     batch_size=32)
387
388 validation_generator = validation_datagen.flow_from_directory(
389     'validation',
390     target_size=(224, 224),
391     batch_size=32)
392
393 # Train the model
394 model.fit_generator(train_generator,
395                   validation_data=validation_generator,
396                   epochs=10,
397                   validation_steps=100,
398                   callbacks=[ModelCheckpoint('model.h5'),
399                             EarlyStopping(monitor='val_loss',
400                                           patience=10,
401                                           verbose=1)])
402
403 # Load the model
404 model = VGG16(weights='imagenet')
405
406 # Compile the model
407 optimizer = Adam()
408 model.compile(optimizer=optimizer, loss=categorical_crossentropy,
409             metrics=['accuracy'])
410
411 # Create the training and validation data loaders
412 train_datagen = ImageDataGenerator(
413     rotation_range=20,
414     width_shift_range=0.1,
415     height_shift_range=0.1,
416     shear_range=0.1,
417     zoom_range=0.1,
418     horizontal_flip=True,
419     fill_mode='nearest')
420
421 validation_datagen = ImageDataGenerator(
422     rotation_range=0,
423     width_shift_range=0,
424     height_shift_range=0,
425     shear_range=0,
426     zoom_range=0,
427     horizontal_flip=False,
428     fill_mode='nearest')
429
430 # Create the training and validation data loaders
431 train_generator = train_datagen.flow_from_directory(
432     'train',
433     target_size=(224, 224),
434     batch_size=32)
435
436 validation_generator = validation_datagen.flow_from_directory(
437     'validation',
438     target_size=(224, 224),
439     batch_size=32)
440
441 # Train the model
442 model.fit_generator(train_generator,
443                   validation_data=validation_generator,
444                   epochs=10,
445                   validation_steps=100,
446                   callbacks=[ModelCheckpoint('model.h5'),
447                             EarlyStopping(monitor='val_loss',
448                                           patience=10,
449                                           verbose=1)])
450
451 # Load the model
452 model = VGG16(weights='imagenet')
453
454 # Compile the model
455 optimizer = Adam()
456 model.compile(optimizer=optimizer, loss=categorical_crossentropy,
457             metrics=['accuracy'])
458
459 # Create the training and validation data loaders
460 train_datagen = ImageDataGenerator(
461     rotation_range=20,
462     width_shift_range=0.1,
463     height_shift_range=0.1,
464     shear_range=0.1,
465     zoom_range=0.1,
466     horizontal_flip=True,
467     fill_mode='nearest')
468
469 validation_datagen = ImageDataGenerator(
470     rotation_range=0,
471     width_shift_range=0,
472     height_shift_range=0,
473     shear_range=0,
474     zoom_range=0,
475     horizontal_flip=False,
476     fill_mode='nearest')
477
478 # Create the training and validation data loaders
479 train_generator = train_datagen.flow_from_directory(
480     'train',
481     target_size=(224, 224),
482     batch_size=32)
483
484 validation_generator = validation_datagen.flow_from_directory(
485     'validation',
486     target_size=(224, 224),
487     batch_size=32)
488
489 # Train the model
490 model.fit_generator(train_generator,
491                   validation_data=validation_generator,
492                   epochs=10,
493                   validation_steps=100,
494                   callbacks=[ModelCheckpoint('model.h5'),
495                             EarlyStopping(monitor='val_loss',
496                                           patience=10,
497                                           verbose=1)])
498
499 # Load the model
500 model = VGG16(weights='imagenet')
501
502 # Compile the model
503 optimizer = Adam()
504 model.compile(optimizer=optimizer, loss=categorical_crossentropy,
505             metrics=['accuracy'])
506
507 # Create the training and validation data loaders
508 train_datagen = ImageDataGenerator(
509     rotation_range=20,
510     width_shift_range=0.1,
511     height_shift_range=0.1,
512     shear_range=0.1,
513     zoom_range=0.1,
514     horizontal_flip=True,
515     fill_mode='nearest')
516
517 validation_datagen = ImageDataGenerator(
518     rotation_range=0,
519     width_shift_range=0,
520     height_shift_range=0,
521     shear_range=0,
522     zoom_range=0,
523     horizontal_flip=False,
524     fill_mode='nearest')
525
526 # Create the training and validation data loaders
527 train_generator = train_datagen.flow_from_directory(
528     'train',
529     target_size=(224, 224),
530     batch_size=32)
531
532 validation_generator = validation_datagen.flow_from_directory(
533     'validation',
534     target_size=(224, 224),
535     batch_size=32)
536
537 # Train the model
538 model.fit_generator(train_generator,
539                   validation_data=validation_generator,
540                   epochs=10,
541                   validation_steps=100,
542                   callbacks=[ModelCheckpoint('model.h5'),
543                             EarlyStopping(monitor='val_loss',
544                                           patience=10,
545                                           verbose=1)])
546
547 # Load the model
548 model = VGG16(weights='imagenet')
549
550 # Compile the model
551 optimizer = Adam()
552 model.compile(optimizer=optimizer, loss=categorical_crossentropy,
553             metrics=['accuracy'])
554
555 # Create the training and validation data loaders
556 train_datagen = ImageDataGenerator(
557     rotation_range=20,
558     width_shift_range=0.1,
559     height_shift_range=0.1,
560     shear_range=0.1,
561     zoom_range=0.1,
562     horizontal_flip=True,
563     fill_mode='nearest')
564
565 validation_datagen = ImageDataGenerator(
566     rotation_range=0,
567     width_shift_range=0,
568     height_shift_range=0,
569     shear_range=0,
570     zoom_range=0,
571     horizontal_flip=False,
572     fill_mode='nearest')
573
574 # Create the training and validation data loaders
575 train_generator = train_datagen.flow_from_directory(
576     'train',
577     target_size=(224, 224),
578     batch_size=32)
579
580 validation_generator = validation_datagen.flow_from_directory(
581     'validation',
582     target_size=(224, 224),
583     batch_size=32)
584
585 # Train the model
586 model.fit_generator(train_generator,
587                   validation_data=validation_generator,
588                   epochs=10,
589                   validation_steps=100,
590                   callbacks=[ModelCheckpoint('model.h5'),
591                             EarlyStopping(monitor='val_loss',
592                                           patience=10,
593                                           verbose=1)])
594
595 # Load the model
596 model = VGG16(weights='imagenet')
597
598 # Compile the model
599 optimizer = Adam()
600 model.compile(optimizer=optimizer, loss=categorical_crossentropy,
601             metrics=['accuracy'])
602
603 # Create the training and validation data loaders
604 train_datagen = ImageDataGenerator(
605     rotation_range=20,
606     width_shift_range=0.1,
607     height_shift_range=0.1,
608     shear_range=0.1,
609     zoom_range=0.1,
610     horizontal_flip=True,
611     fill_mode='nearest')
612
613 validation_datagen = ImageDataGenerator(
614     rotation_range=0,
615     width_shift_range=0,
616     height_shift_range=0,
617     shear_range=0,
618     zoom_range=0,
619     horizontal_flip=False,
620     fill_mode='nearest')
621
622 # Create the training and validation data loaders
623 train_generator = train_datagen.flow_from_directory(
624     'train',
625     target_size=(224, 224),
626     batch_size=32)
627
628 validation_generator = validation_datagen.flow_from_directory(
629     'validation',
630     target_size=(224, 224),
631     batch_size=32)
632
633 # Train the model
634 model.fit_generator(train_generator,
635                   validation_data=validation_generator,
636                   epochs=10,
637                   validation_steps=100,
638                   callbacks=[ModelCheckpoint('model.h5'),
639                             EarlyStopping(monitor='val_loss',
640                                           patience=10,
641                                           verbose=1)])
642
643 # Load the model
644 model = VGG16(weights='imagenet')
645
646 # Compile the model
647 optimizer = Adam()
648 model.compile(optimizer=optimizer, loss=categorical_crossentropy,
649             metrics=['accuracy'])
650
651 # Create the training and validation data loaders
652 train_datagen = ImageDataGenerator(
653     rotation_range=20,
654     width_shift_range=0.1,
655     height_shift_range=0.1,
656     shear_range=0.1,
657     zoom_range=0.1,
658     horizontal_flip=True,
659     fill_mode='nearest')
660
661 validation_datagen = ImageDataGenerator(
662     rotation_range=0,
663     width_shift_range=0,
664     height_shift_range=0,
665     shear_range=0,
666     zoom_range=0,
667     horizontal_flip=False,
668     fill_mode='nearest')
669
670 # Create the training and validation data loaders
671 train_generator = train_datagen.flow_from_directory(
672     'train',
673     target_size=(224, 224),
674     batch_size=32)
675
676 validation_generator = validation_datagen.flow_from_directory(
677     'validation',
678     target_size=(224, 224),
679     batch_size=32)
680
681 # Train the model
682 model.fit_generator(train_generator,
683                   validation_data=validation_generator,
684                   epochs=10,
685                   validation_steps=100,
686                   callbacks=[ModelCheckpoint('model.h5'),
687                             EarlyStopping(monitor='val_loss',
688                                           patience=10,
689                                           verbose=1)])
689
```

Gambar 4. Pelatihan Model

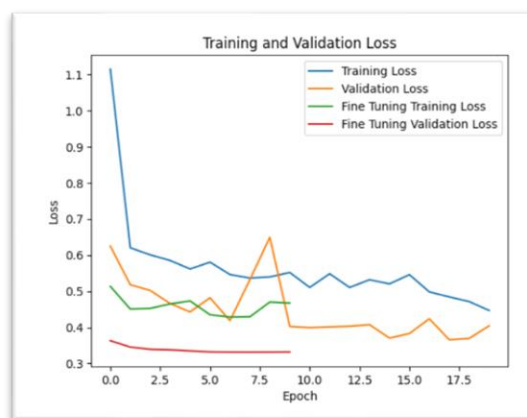
Akurasi pada data validasi: 0.8474576473236084

Gambar 5. Hasil Akurasi

Selanjutnya, untuk membuat plot akurasi dan loss selama proses pelatihan model. Pertama, plot akurasi pelatihan dan validasi dibuat dengan menggunakan data dari history.history['accuracy'] dan history.history['val_accuracy']. Hal ini membantu dalam memantau perkembangan akurasi model selama pelatihan dan mengevaluasi kinerja model pada data yang belum pernah dilihat sebelumnya. Selanjutnya, plot loss pelatihan dan validasi dibuat dengan menggunakan data dari history.history['loss'] dan history.history['val_loss']. Hal ini memungkinkan untuk melihat bagaimana loss model berubah seiring berjalannya proses pelatihan. Dengan memeriksa kedua grafik tersebut, kita dapat mengevaluasi performa model serta memastikan apakah terjadi overfitting atau underfitting. Dalam kasus ini, akurasi model telah mencapai 84% dengan total epoch yang ditentukan adalah 20, menunjukkan bahwa model telah belajar dengan baik dari data latih dan mampu melakukan klasifikasi dengan akurat.



Gambar 6. Training dan Validasi akurasi



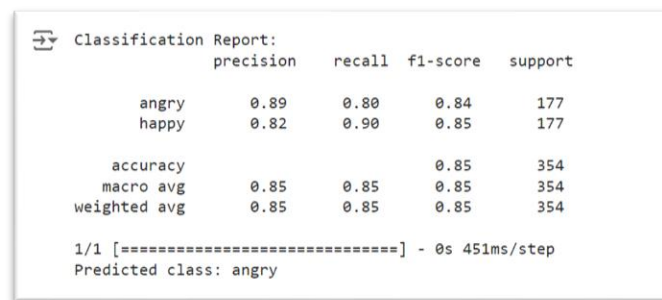
Gambar 7. Training dan Validasi loss

Selanjutnya, laporan klasifikasi dibuat menggunakan fungsi `classification_report` dari modul `sklearn.metrics`. Laporan ini memberikan detail tentang performa model dalam mengklasifikasikan data validasi, termasuk presisi, recall, dan nilai F1 untuk setiap kelas. Kemudian, sebuah fungsi bernama `classify_image` didefinisikan untuk melakukan klasifikasi pada gambar tunggal. Fungsi ini mengambil gambar dari path yang ditentukan, mengonversinya menjadi array gambar, melakukan prediksi menggunakan model yang telah dilatih, dan mengonversi hasil prediksi menjadi kelas yang sesuai menggunakan dictionary `class_indices`. Contoh penggunaan fungsi `classify_image` digunakan untuk memprediksi kelas dari satu gambar contoh yang diberikan.

```

1 # Buat laporan klasifikasi
2 class_report = classification_report(validation_generator.classes, y_pred, target_names=validation_generator.class_indices.keys())
3 print('Classification Report:\n', class_report)
4
5 # Fungsi untuk melakukan klasifikasi gambar tunggal
6 def classify_image(model, img_path, target_size=(150, 150)):
7     img = tf.keras.preprocessing.image.load_img(img_path, target_size=target_size)
8     img_array = tf.keras.preprocessing.image.img_to_array(img)
9     img_array = np.expand_dims(img_array, axis=0) / 255.0
10    prediction = model.predict(img_array)
11    class_indices = {v: k for k, v in validation_generator.class_indices.items()}
12    predicted_class = class_indices[np.argmax(prediction)]
13    return predicted_class
14
15 # Contoh penggunaan klasifikasi gambar
16 sample_image_path = "/content/drive/My Drive/H4(Nalidani)imgry/angry.jpg" # ganti dengan path gambar Anda
17 predicted_class = classify_image(model, sample_image_path)
18 print(f"Predicted class: {predicted_class}")
    
```

Gambar 8. Tahap Klasifikasi



```
Classification Report:
precision  recall  f1-score  support
angry      0.89   0.80   0.84     177
happy      0.82   0.90   0.85     177

accuracy          0.85     354
macro avg         0.85     354
weighted avg      0.85     354

1/1 [=====] - 0s 451ms/step
Predicted class: angry
```

Gambar 9. Hasil Klasifikasi

4. Kesimpulan

Berdasarkan penelitian ini, dapat disimpulkan bahwa Convolutional Neural Network (CNN) menunjukkan potensi yang baik dalam klasifikasi emosi pada citra anjing, terutama dalam membedakan antara emosi marah dan bahagia. Penggunaan dataset sekunder dari Kaggle yang terdiri dari 400 foto anjing untuk setiap kategori emosi telah memberikan hasil yang memuaskan. Proses awal melibatkan preprocessing data dengan menyesuaikan ukuran citra dan melakukan augmentasi data untuk meningkatkan variasi dataset. Model CNN yang dibangun menggunakan arsitektur VGG16 sebagai base model berhasil dilatih dengan baik menggunakan generator data latih. Hasil evaluasi menunjukkan bahwa model mencapai akurasi sebesar 84% pada data validasi setelah dilatih selama 20 epoch. Analisis lebih lanjut terhadap confusion matrix dan laporan klasifikasi mengonfirmasi kemampuan model dalam melakukan klasifikasi emosi dengan baik. Oleh karena itu, penelitian ini menunjukkan bahwa CNN memiliki potensi aplikasi yang luas dalam klasifikasi emosi pada citra anjing, dengan manfaat potensial dalam bidang seperti pemantauan hewan peliharaan dan pemrosesan citra medis.

Daftar Pustaka

- [1] S. Danish Arkansa and C. Lubis, "Klasifikasi Ras Anjing Menggunakan Metode Convolutional Neural Network Dengan Arsitektur Vgg-16."
- [2] K. O. Lauw, L. W. Santoso, and R. Intan, "Identifikasi Jenis Anjing Berdasarkan Gambar Menggunakan Convolutional Neural Network Berbasis Android."
- [3] K. Wajah Hewan Tresia Aprilia, "Jurnal Teknik Informatika dan Desain Komunikasi Visual Analisa Perbandingan Inception dan Xception Berbasis CNN untuk," *Universitas Selamat Sri*, vol. 3, no. 1, 2024.
- [4] R. Rafiif Amaanullah, G. Rizka Pasfica, S. Adi Nugraha, and M. Rifqi Zein, "Implementasi Convolutional Neural Network Untuk Deteksi Emosi Melalui Wajah (Implementation of Convolutional Neural Network for Emotion Detection Through Face)." [Online]. Available: <https://www.kaggle.com/shivambhardwaj0101/emo>
- [5] A. Arifandi, "Jurnal Terapan Sains & Teknologi Identifikasi Dan Prediksi Umur Serta Jenis Kelamin Berdasarkan Citra Wajah Menggunakan Algoritma Convolutional Neural Network (CNN)," *Fakultas Sains dan Teknologi-Universitas PGRI Kanjuruhan Malang*, vol. 4, no. 2, p. 2022.

Halaman ini sengaja dibiarkan kosong