International Journal of Computer and Information System (IJCIS)
Peer Reviewed – International Journal
Vol : Vol. 05, Issue 02, June 2024
e-ISSN : 2745-9659
https://ijcis.net/index.php/ijcis/index

Classification of Cattle Diseases in Semin District Using Convolutional Neural Network (CNN)

1st Xvan Erik Kobar Permana, 2nd Nendy Akbar Rozaq Rais, 3rd Muqorobin
[123]Program S1 Informatika, Institut Teknologi Bisnis AAS Indonesia
[123]Institut Teknologi Bisnis AAS Indonesia, Sukoharjo, Indonesia
e-mail : [1]xvanerikkobarpermana@gmail.com, [2]ab.terate@gmail.com, [3]robbyaullah@gmail.com

Abstract—Cattle farming is a crucial sector for the economy and food security in Semin District. However, cattle diseases pose a serious threat, leading to economic losses and animal welfare issues. Farmers' lack of understanding about cattle diseases hinders effective disease management, and some solutions implemented by farmers can worsen the condition of the animals. Therefore, this study aims to implement a disease classification system for cattle using Convolutional Neural Network (CNN). The diseases targeted in this study include three common threats to cattle in this region: Bovine Ephemeral Fever (BEF), Mastitis, and Scabies. With the advancement of technology, it is expected that cattle farmers in Semin District can minimize errors in diagnosing cattle diseases through the application of artificial intelligence (AI) for disease classification. The study utilized a dataset consisting of 864 training data and 216 validation data, achieving an accuracy of 1.0000 and a loss of 0.0040. For testing, the system achieved an accuracy of 0.9306 and a loss of 0.4430.

Keywords : Classification, Cattle, Convolutional Neural Network

I. INTRODUCTION

Livestock farming is a promising sector for the economy and the provision of animal protein. Among the various livestock, cattle are the most prominent in Semin District, followed by chickens and goats. According to the Gunungkidul Regency Central Statistics Agency, the number of cattle in Semin District increased from 13,164 in 2019 to 12,592 in 2020, consisting of both dairy and beef cattle (source: gunungkidulkab.bps.go.id).

Despite the increase, cattle production has decreased, mainly due to diseases. In 2023, three major cattle diseases in Semin District were Bovine Ephemeral Fever (BEF), also known as Three Days Sickness, Mastitis (inflammation of the mammary gland tissues), and Scabies (a contagious skin disease). Farmers' lack of knowledge about these diseases often leads to incorrect handling, exacerbating the problem.

The advancement of technology can assist farmers in accurately diagnosing cattle diseases using image processing and detection technologies. Deep learning techniques, particularly Convolutional Neural Networks (CNN), have shown remarkable results in image classification. CNNs offer benefits such as weight-sharing, which reduces the number of trainable parameters, improving generalization and avoiding overfitting. CNNs also learn feature extraction and classification layers simultaneously, leading to highly organized outputs based on extracted features.

Previous studies on cattle disease diagnosis include "Identification of Lumpy Skin Disease in Cattle Using Convolutional Neural Network Image Classification" [1], which focused on a single disease class. In contrast, this study classifies three diseases. Other relevant studies include "Classification of Tomato Plant Diseases Using Convolutional Neural Network (CNN)" [2], achieving an accuracy range of 82% to 98%, and "Classification of Corn Plant Diseases Using Convolutional Neural Network (CNN)" [3], achieving a training accuracy of 97.5% and a testing accuracy of 94%.

Additionally, research on fruit ripeness classification such as "Classification of Sweet Orange Fruit Maturity Based on Brightness Levels Using Deep Learning Convolutional Neural Network" [4], and plant classification like "Classification of Aglaonema Plants Based on Leaf Images Using Convolutional Neural Network (CNN)" [5] showed high accuracies, with the latter achieving a 99% testing accuracy. Another relevant study is "Classification of Brain Tumors Based on MRI Images Using Convolutional Neural Network (CNN)" [6], using the MobileNetV2 model. This study differs by focusing on cattle disease classification and implementing the system in a mobile application.

Groundbreaking research such as "ImageNet Classification with Deep Convolutional Neural Networks" [7], "Going Deeper with Convolutions" [8], "Fully Convolutional Networks for Semantic Segmentation" [9], and "Deep Residual Learning for Image Recognition" [10] have laid the foundation for CNN applications in various image classification tasks. This study aims to implement a disease classification system for cattle using CNN to aid farmers in accurately diagnosing cattle diseases and improving cattle health management.

II. RESEARCH METHODS

The system design for cattle disease classification in Semin District requires a software model that simplifies understanding of the design flow, using a Flowchart System for logical comprehension (Figure 1).

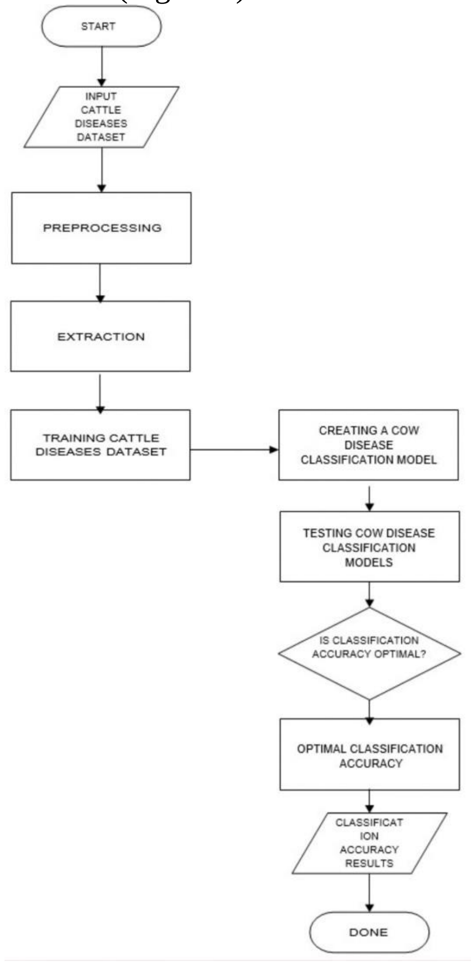

START
INPUT CATTLE DISEASES DATASET
PREPROCESSING
EXTRACTION
TRAINING CATTLE DISEASES DATASET
CREATING A COW DISEASE CLASSIFICATION MODEL
TESTING COW DISEASE CLASSIFICATION MODELS
IS CLASSIFICATION ACCURACY OPTIMAL?
OPTIMAL CLASSIFICATION ACCURACY
CLASSIFICAT ION ACCURACY RESULTS
DONE

Figure 1: Flowchart Design

In Figure 1, the first step is inputting the dataset, sourced from three categories of cattle diseases provided by the Semin Veterinary Health Center in Gunungkidul, Yogyakarta. The next step is preprocessing, where images are converted into vectors for easier convolution processing and image input handling. Following this, the dataset is divided into training and testing sets in the extraction process.

The training process aims to optimize the model. Once trained, the classification model is built and tested for accuracy. If the model meets the desired accuracy, it proceeds to the next step; otherwise, it returns to the training phase for further optimization. When optimal accuracy is achieved, the final classification accuracy results in a system for classifying cattle diseases in Semin District.

2.1 CNN Architecture Workflow

The workflow of the Convolutional Neural Network (CNN) architecture can be understood through the following steps:

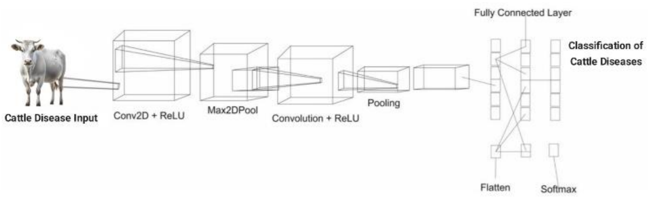


Figure 2: CNN Architecture Workflow

a) Input : input consists of a dataset of cattle diseases, categorized into three types: Bovine Ephemeral Fever (BEF), Mastitis, and Scabies.
b) Convolution + ReLU : is a process that combines convolution and activation used for training a model. Convolution performs summation of multiplications between a filter or kernel and the input. In this research, Conv2D is used for image classification with a filter parameter of 32, indicating 32 feature detectors with a 5x5 kernel. The ReLU activation function is used to address the vanishing gradient problem and efficiently compute activations.
c) Pooling : After convolution, pooling enhances and improves the performance of the CNN. Max pooling, a common pooling method, is used in this research. MaxPool2D is applied to the dataset images to reduce the feature map size by selecting the maximum value, aiding in the classification of cattle diseases in Semin District.

d) Fully Connected Layer : After obtaining the feature map, flattening is performed to convert it into a vector, which serves as input for the Fully Connected Layer, resulting in the output classification of cattle diseases.

III. RESULT AND ANALYSIS

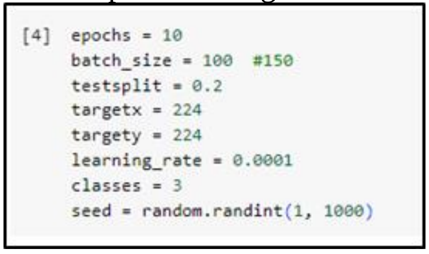
In this process, the dataset is divided into two parts: Training and Testing. The Training data is used to predict a model that has been created, as well as to run the functions of a Deep Learning algorithm, guiding the model to independently find correlations through the provided data. Testing is the dataset that will be used to test the accuracy or, in other words, to evaluate the performance of the given model.

3.1 Import Library

Python libraries are imported into the Google Colaboratory notebook as needed to build the application. These libraries facilitate the development process by reducing the need to write extensive code. Libraries used include numpy, pandas, tensorflow, and keras, among others.

3.2 Variables

Variables used in creating the application model include the number of epochs set to 10, batch_size of 100 for the training process, and Testsplit for dataset division. Variables targetx and targety are used to resize images to 244x244 pixels. The learning rate is set at 0.0001 to adjust weight correction during training. The classes variable determines the number of classes in the dataset, and random.randint returns an integer element selected from a specified range.

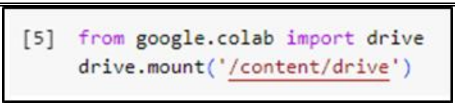


```
[4]  epochs = 10
     batch_size = 100  #150
     testsplit = 0.2
     targetx = 224
     targety = 224
     learning_rate = 0.0001
     classes = 3
     seed = random.randint(1, 1000)
```

Figure 3: Variables

3.3 Import Dataset from Google Drive

The next step is to import Google Drive to connect it to the Google Colaboratory notebook.

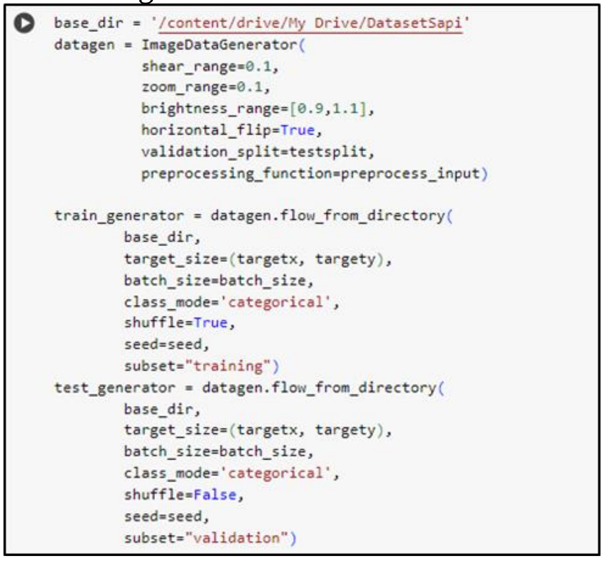


```
[5]  from google.colab import drive
     drive.mount('/content/drive')
```

Figure 4 : Import Dataset

3.4 Image Data Processing

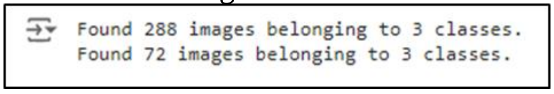
This process involves augmenting the dataset. The first line stores the Google Drive location in base_dir. The second line, image generator, augments the dataset, adding images in real-time during model training. The third and fourth lines divide the dataset into Training and Testing sets.



```
base_dir = '/content/drive/My Drive/DatasetSapi'
datagen = ImageDataGenerator(
          shear_range=0.1,
          zoom_range=0.1,
          brightness_range=[0.9,1.1],
          horizontal_flip=True,
          validation_split=testsplit,
          preprocessing_function=preprocess_input)

train_generator = datagen.flow_from_directory(
        base_dir,
        target_size=(targetx, targety),
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=True,
        seed=seed,
        subset="training")
test_generator = datagen.flow_from_directory(
        base_dir,
        target_size=(targetx, targety),
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=False,
        seed=seed,
        subset="validation")
```

Figure 5: Image Data Processing

The output after running the above program shows the result of the Image Data Processing, displaying the division of the true dataset into 288 Training images across 3 disease categories and 72 Testing images across 3 disease categories.

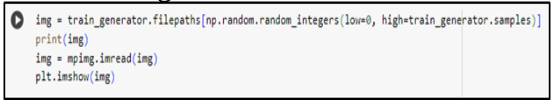


```
Found 288 images belonging to 3 classes.
Found 72 images belonging to 3 classes.
```

Figure 6: Output Result

3.5 Sample Image

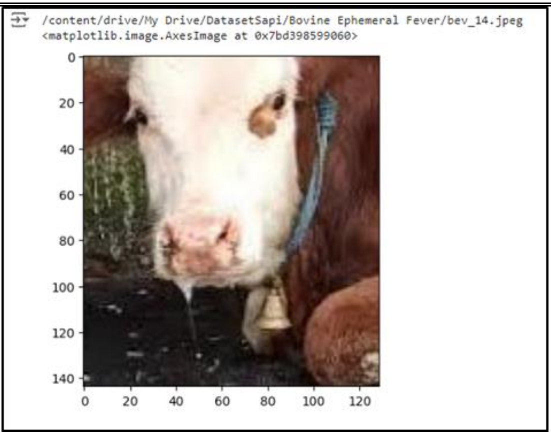
This method involves selecting one image from the dataset as a sample. The source code below is used to take one sample image from the various categories.



```
img = train_generator.filepaths[np.random.random_integers(low=0, high=train_generator.samples)]
print(img)
img = mpimg.imread(img)
plt.imshow(img)
```

Figure 7: Sample Image

The output of the sample image can be seen in the following figure.



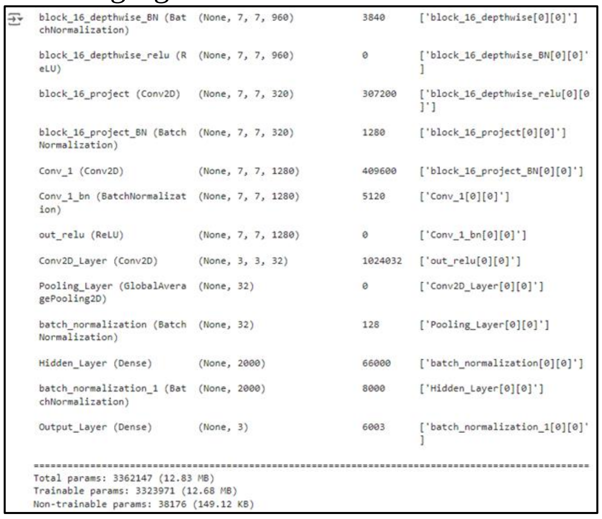
```
/content/drive/My Drive/DatasetSapi/Bovine Ephemeral Fever/bev_14.jpeg
<matplotlib.image.AxesImage at 0x7bd398599060>
```

Figure 8: Output Sample Image

3.6 MobileNetV2 Model

MobileNetV2 is a CNN architecture designed to meet computing needs. This model can be implemented in applications as required. The MobileNetV2 model built for classifying cattle diseases into three categories: BEV, Mastitis, and Scabies.

The MobileNetV2 model begins with Conv2D, initiating convolution with a filter size of 32, using ReLU activation and kernel_regularizers to manage hidden units. GlobalAveragePooling2D consolidates the average values to determine input width, height, and channels. Batch_Normalization speeds up and stabilizes the CNN through input layer normalization. Model_input is used to input the MobileNetV2 model for predictions. Optimizer_adam minimizes the model parameters, and Categorical_Crossentropy classifies multi-class outputs. The MobileNetV2 model output is displayed in the following figure.



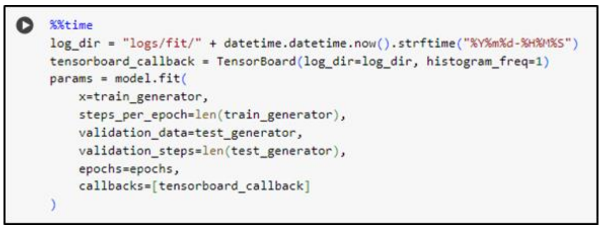
```
block_16_depthwise_BN (Bat  (None, 7, 7, 960)        3840      ['block_16_depthwise[0][0]']
chNormalization)

block_16_depthwise_relu (R  (None, 7, 7, 960)        0         ['block_16_depthwise_BN[0][0]'
eLU)                                                            ]

block_16_project (Conv2D)   (None, 7, 7, 320)        307200    ['block_16_depthwise_relu[0][0
                                                                ]']

block_16_project_BN (Batch  (None, 7, 7, 320)        1280      ['block_16_project[0][0]']
Normalization)

Conv_1 (Conv2D)             (None, 7, 7, 1280)       409600    ['block_16_project_BN[0][0]']

Conv_1_bn (BatchNormalizat  (None, 7, 7, 1280)       5120      ['Conv_1[0][0]']
ion)

out_relu (ReLU)             (None, 7, 7, 1280)       0         ['Conv_1_bn[0][0]']

Conv2D_Layer (Conv2D)       (None, 3, 3, 32)         1024032   ['out_relu[0][0]']

Pooling_Layer (GlobalAvera  (None, 32)               0         ['Conv2D_Layer[0][0]']
gePooling2D)

batch_normalization (Batch  (None, 32)               128       ['Pooling_Layer[0][0]']
Normalization)

Hidden_Layer (Dense)        (None, 2000)             66000     ['batch_normalization[0][0]']

batch_normalization_1 (Bat  (None, 2000)             8000      ['Hidden_Layer[0][0]']
chNormalization)

Output_Layer (Dense)        (None, 3)                6003      ['batch_normalization_1[0][0]'
                                                                ]

==================================================================================================
Total params: 3362147 (12.83 MB)
Trainable params: 3323971 (12.68 MB)
Non-trainable params: 38176 (149.12 KB)
```

Figure 9: Output MobileNetV2

The output shows a total of “3,362,147” parameters, with “3,323,971” trainable parameters and “38,176” non-trainable parameters.

3.7 Training Model and Validation

The source code for Training and Validation of the model is shown below.



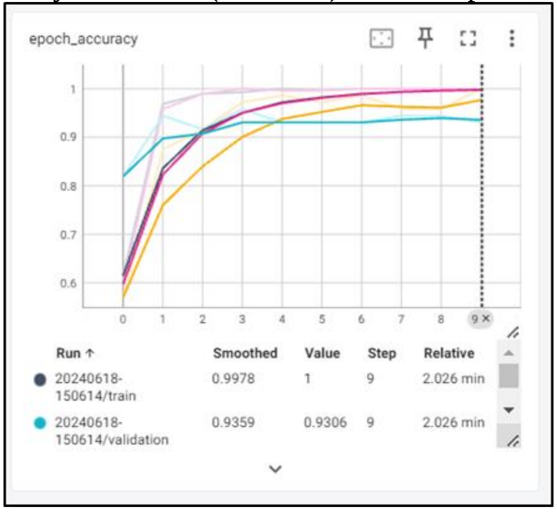
```
%%time
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=log_dir, histogram_freq=1)
params = model.fit(
    x=train_generator,
    steps_per_epoch=len(train_generator),
    validation_data=test_generator,
    validation_steps=len(test_generator),
    epochs=epochs,
    callbacks=[tensorboard_callback]
)
```

Figure 10: Training Model and Validation

This source code performs model Training and Validation. The first line %%time and Log fit measure the training process time. The second line, tensorboard_callback, monitors each training step, resulting in a performance graph. The third line, params with model.fit_generator, trains the model. Step_per_epoch displays iterations before training completion, validation_steps show the Testing dataset results, and Epoch functions as a single iteration return during CNN training. The output shows that after 10 epochs, Training accuracy is 1.0000 with a loss of 0.0040, and Testing accuracy is 0.9306 with a loss of 0.4430.

3.8 Accuracy Graph in TensorBoard

The performance graph shows Training accuracy of 0.9768 (orange line) and Testing accuracy of 0.9359 (blue line) over 10 epochs.

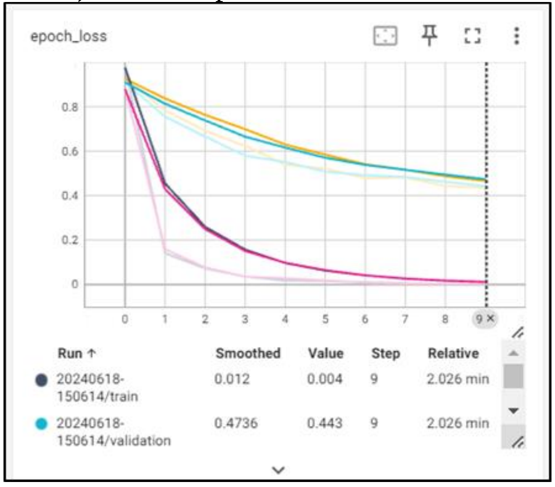


	Run ↑	Smoothed	Value	Step	Relative
●	20240618-150614/train	0.9978	1	9	2.026 min
●	20240618-150614/validation	0.9359	0.9306	9	2.026 min

Figure 11: Accuracy Graph in TensorBoard

3.9 Loss Graph in TensorBoard

The loss graph shows a Training loss of 0.466 (orange line) and Testing loss of 0.4736 (blue line) over 10 epochs.

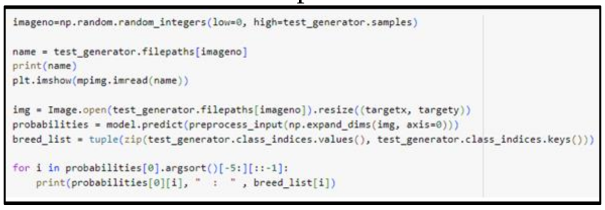


Run ↑	Smoothed	Value	Step	Relative
● 20240618-150614/train	0.012	0.004	9	2.026 min
● 20240618-150614/validation	0.4736	0.443	9	2.026 min

Figure 12: Loss Graph in TensorBoard

3.10 Sample Prediction

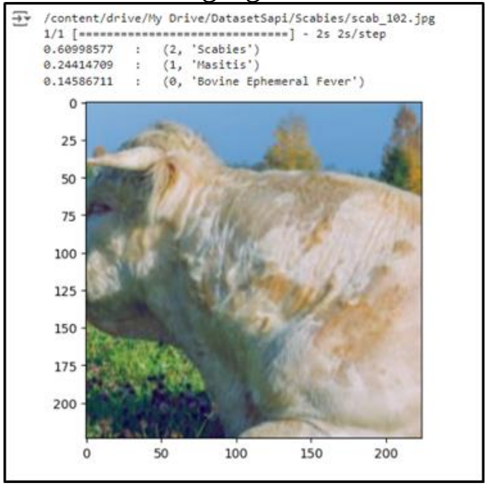
After training, the model's accuracy is tested on the Testing data. The source code below displays the image location, size, and a random value used for prediction.



```
imageno=np.random.random_integers(low=0, high=test_generator.samples)

name = test_generator.filepaths[imageno]
print(name)
plt.imshow(mpimg.imread(name))

img = Image.open(test_generator.filepaths[imageno]).resize((targetx, targety))
probabilities = model.predict(preprocess_input(np.expand_dims(img, axis=0)))
breed_list = tuple(zip(test_generator.class_indices.values(), test_generator.class_indices.keys()))

for i in probabilities[0].argsort()[-5:][::-1]:
    print(probabilities[0][i], "  :  " , breed_list[i])
```

Figure 13: Sample Prediction

The output of the sample prediction is shown in the following figure.

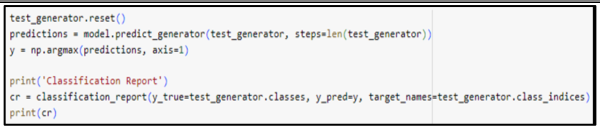


```
/content/drive/My Drive/DatasetSapi/Scabies/scab_102.jpg
1/1 [==============================] - 2s 2s/step
0.60998577   :   (2, 'Scabies')
0.24414709   :   (1, 'Masitis')
0.14586711   :   (0, 'Bovine Ephemeral Fever')
```

Figure 14: Output Sample Prediction

3.11 Classification Report

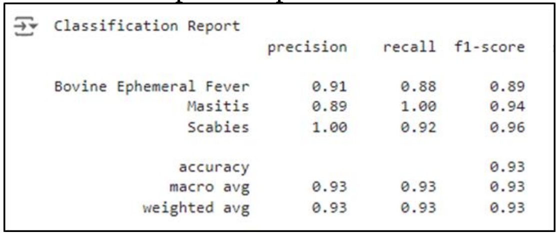
The source code for generating a Classification Report is shown below.



```
test_generator.reset()
predictions = model.predict_generator(test_generator, steps=len(test_generator))
y = np.argmax(predictions, axis=1)

print('Classification Report')
cr = classification_report(y_true=test_generator.classes, y_pred=y, target_names=test_generator.class_indices)
print(cr)
```

Figure 15: Classification Report

The first line directs to the Testing dataset, the second line predicts the Testing dataset, the third line displays the prediction results, the fourth line prints the text, the fifth line saves the prediction results, and the sixth line prints the saved results, showing Accuracy, Precision, Recall, and F1-score. The Classification Report output is shown below.



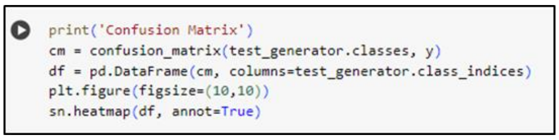
```
Classification Report
                        precision    recall  f1-score

Bovine Ephemeral Fever       0.91      0.88      0.89
               Masitis       0.89      1.00      0.94
               Scabies       1.00      0.92      0.96

              accuracy                           0.93
             macro avg       0.93      0.93      0.93
          weighted avg       0.93      0.93      0.93
```

Figure 16: Output Classification Report

Output shows an Accuracy of 0.93, Precision for Bovine Ephemeral Fever 0.91, Mastitis 0.89, Scabies 1.00, Recall for Bovine Ephemeral Fever 0.88, Mastitis 1.00, Scabies 0.96, and F1-Score for Bovine Ephemeral Fever 0.89, Mastitis 0.94, Scabies 0.96.

3.12 Confusion Matrix

The source code for generating a Confusion Matrix is shown below.



```
print('Confusion Matrix')
cm = confusion_matrix(test_generator.classes, y)
df = pd.DataFrame(cm, columns=test_generator.class_indices)
plt.figure(figsize=(10,10))
sn.heatmap(df, annot=True)
```

Figure 17: Confusion Matrix

The Confusion Matrix shows 67 correct predictions and 5 incorrect predictions, with the evaluation results displayed below.

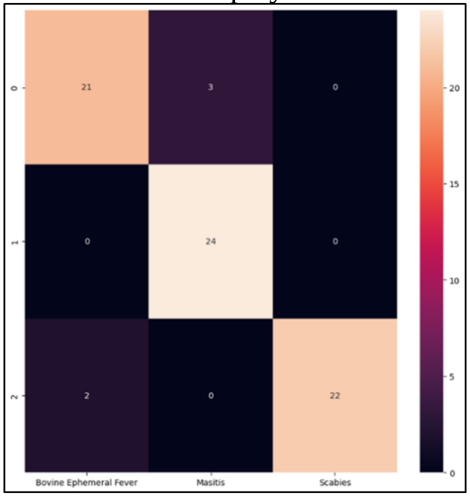
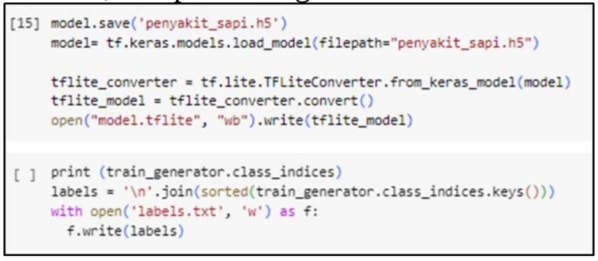


Figure 18: Confusion Matrix Model

3.13 Export Model & Label

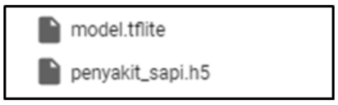
Export Model and Label source code is used to implement the model into an application. The first line saves the model, the second line calls the model to save into variable penyakit_sapi.h5, and the third to fifth lines convert penyakit_sapi.h5 to model.tflite for use in Android Studio. The next lines create Labels, saving Training data into the Labels variable, and processing it into a labels.txt file.



```
[15] model.save('penyakit_sapi.h5')
     model= tf.keras.models.load_model(filepath="penyakit_sapi.h5")

     tflite_converter = tf.lite.TFLiteConverter.from_keras_model(model)
     tflite_model = tflite_converter.convert()
     open("model.tflite", "wb").write(tflite_model)

[ ]  print (train_generator.class_indices)
     labels = '\n'.join(sorted(train_generator.class_indices.keys()))
     with open('labels.txt', 'w') as f:
       f.write(labels)
```

Figure 19: Export Model & Label

The output of the Export Model & Label process is shown below.



model.tflite
penyakit_sapi.h5

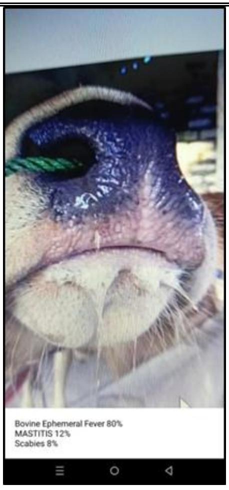
Figure 20: Output Result

3.14 Application Testing

Application testing process was conducted to assess the quality of the cattle disease analysis results obtained from the designed system model. The classification of cattle diseases in Semin District has been implemented in an app that can be downloaded from https://acesse.one/penyakitsapisemin.

This application has undergone testing processes to show the predicted results for three categories of cattle diseases, namely Bovine Ephermal Fever, Mastitis, and Scabies based on the developed model. The classification results displayed in the application are as follows:

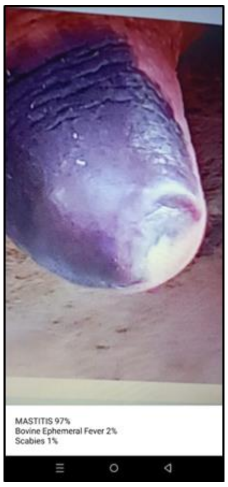
a) Classification of Bovine Ephermal Fever : The application can predict Bovine Ephermal Fever with an accuracy rate of 80%, as shown in



```
Bovine Ephemeral Fever 80%
MASTITIS 12%
Scabies 8%
```

Figure 21: Cattle Disease Bovine Ephermal Fever.

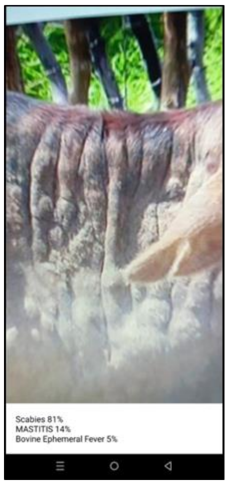
b) Classification of Mastitis : The application can predict Mastitis with an accuracy rate of 97%, as shown in



```
MASTITIS 97%
Bovine Ephemeral Fever 2%
Scabies 1%
```

Figure 22: Cattle Disease Mastitis.

c) Classification of Scabies : The application can predict Scabies with an accuracy rate of 81%, as shown in



```
Scabies 81%
MASTITIS 14%
Bovine Ephemeral Fever 5%
```

Figure 23: Cattle Disease Scabies.

IV. CONCLUSION

The application developed is beneficial for the community, especially in the cattle farming sector in Semin sub-district, for distinguishing among three categories of cattle diseases. In the field of veterinary health, not all institutions have such a system. This application facilitates Veterinary Medical Officers in the field to differentiate between the three categories of cattle diseases, and also aids the community in minimizing errors in diagnosing cattle diseases.

REFERENCES

[1] Thedjo Sentoso, Fachri Ardiansyah, Virginia Tamuntuan, Sabda Sastra Wangsa, Kusrini, Kusnawi. 2024. "Identification of Lumpy Skin Disease in Cattle using Image Classification with Convolutional Neural Network," S2 Distance Learning Program in Informatics Engineering, Amikom University Yogyakarta.

[2] Rendra Soekarta, Nirwana Nurdjan, Ardian Syah. 2023. "Classification of Tomato Plant Diseases using Convolutional Neural Network (CNN)," Informatics Engineering. Sorong Muhammadiyah University.

[3] E. Rasywir, R. Sinaga, Y. Pratama. 2020. "Classification of Corn Plant Diseases using Convolutional Neural Network (CNN)," Paradig - Journal of Computing and Informatics, vol. 22, no. 2, pp. 117-123, doi: 10.31294/p.v22i2.8907.

[4] B. Yanto, L. Fimawahib, A. Supriyanto, B. H. Hayadi, R. R. Pratama. 2021. "Classification of Sweet Orange Fruit Maturity Texture based on Brightness Color Intensity using Deep Learning Convolutional Neural Network," INOVTEK Polbeng - Informatics Series, vol. 6, no. 2, p. 259, doi: 10.35314/isi.v6i2.2104.

[5] S. Muhammad, A. T. Wibowo. 2021. "Classification of Aglaonema Plants based on Leaf Images using Convolutional Neural Network (CNN)," vol. 8, no. 5, pp. 10621-10636.

[6] Muhammad Bargas Ishak. 2023. "Classification of Brain Tumors based on MRI Images using CNN (Convolution Neural Network)," Informatics Engineering. Respati University Yogyakarta.

[7] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. 2018. "ImageNet Classification with Deep Convolutional Neural Networks," Neural Information Processing Systems. University of Toronto.

[8] V Wiley, Thomas Lucas 2018. "Going Deeper with Convolutions Neural Network," Computer Vision and Pattern Recognition. TSCC Jakarta, Indonesia.

[9] Rajshree Jodha, Priyanka Mishra, J. Angel Arul Jothi, Sameer Saifi, and Abhishek. 2023. "Fully Convolutional Networks for Semantic Segmentation," International Conference on Bioinformatics, Computational Biology, and Health Informatics.

[10] Sarfaraz Masood, Umer Ahsan, Fatima Munawwar, Danish Raza Rizvi, Mumtaz Ahmed. 2020. "Scene Recognition from Image Using Convolutional Neural Network," Department of Computer Engineering, Jamia Millia Islamia New Delhi 110025, INDIA.