Animal Classification using Convolutional Neural Networks :-

Abstract:

This towering project is the enhancement of a Convolutional Neural Network (CNN) for animal images in fifteen types of animals. The only one-of-a-kind dataset having this smaller number of images is suited for beginner deep-learning applications. The images were pre-processed using TensorFlow utilities along with image augmentations that helped to minimize overfitting and better generalization of the model. The CNN architecture was made in Keras Sequential API. This CNN architecture was much simpler, but with notable effectiveness, composed of convolution, pooling, dropout, and dense layers. With approximately 96% training accuracy achieved and 98%+ validation accuracy barely requiring any GPU acceleration.

Introduction:

- Image classification is a very basic, but very important, task in computer vision discriminative categorizing of objects presented in an image. When deep learning became in the limelight, CNNs were considered just the right tools: presenting them with raw-pixel data, they would then extract and learn the appropriate features.
- With this task, having to build an image classifier based on CNNs to identify animals in a dataset containing fifteen classes-dogs, cats, lions, tigers, elephants, and so on-is enough. Since the dataset is smaller and well-structured, it forms a great motivation for any beginner trying to understand and practice deep learning concepts.
- The model embraced TensorFlow and Keras, following an almost naive, working architecture for CNN modelling-conv, pool, dropout, and dense layers. Data augmentation methods like random flipping, rotation, and zooming were considered to improve generalization.
- This whole pipeline is lightweight enough to run on normal CPU hardware without any GPU acceleration requirements. The main motivation here is to demonstrate that the simplest kind of neural network can yield impressive accuracy on well-prepared datasets.

Dataset Description:

- The dataset included 1,944 images divided into 15 animal categories: bear, bird, cat, cow, deer, dog, dolphin, elephant, giraffe, horse, kangaroo, lion, panda, tiger, and zebra. The TensorFlow tools automatically assign labels based on the class folders when loading data.
- All images were resized to 128×128 pixels. This keeps the size and shape consistent, which speeds up processing time. Since the dataset is small and balanced, it works well on computers that don't support GPUs.
- The image_dataset_from_directory function allows you to split the data into 80% for training and 20% for validation. It also handles batching and one-hot label encoding. Random flipping, rotation, zoom, and contrast adjustments improve the model's generalization.
- This well-organized and compact dataset provides a solid foundation for implementing and testing convolutional neural networks for multi-class image classification.

Technologies and Libraries Used:

- So, here's the rundown: I built this whole thing in Python—because, let's be real, what else would you use for machine learning these days? Everything happened inside a Jupyter Notebook. If you've ever used one, you know it's basically the Swiss Army knife for coding, plotting, and tinkering without losing your mind.
- For the heavy lifting, I stuck with TensorFlow 2.x and its buddy, the good old Keras API. Keras makes building and training deep learning models feel almost… fun? Or at least less soul-crushing. Here's what I leaned on:

1. TensorFlow / Keras: Built the CNN, handled data, did all the training jazz.
2. NumPy: Crunches numbers, wrangles arrays—classic stuff.
3. Matplotlib: Plots, charts, pretty graphs. Gotta see if your model's actually learning or just napping.
4. OS: Moved files around, kept my folders from turning into digital spaghetti.

- Honestly, I ran the whole thing on a regular CPU laptop. No fancy GPUs, no server farm humming in the background—just a humble machine and a small dataset. Means anyone can try it, even if you're not swimming in hardware.
- All these pieces came together to make a clean, modular setup for building and testing an animal classifier. Nothing convoluted. Nothing locked behind a $10,000 graphics card. Just straight-up code and some patience.

Data Preprocessing:

- Alright, so here's how I wrangled the data before tossing it into the model. First off, TensorFlow's got this handy `image dataset_from_directory () thing—super clutch. You just point it at your folders, and boom, it figures out your labels, squishes everything to 128x128, and even does the one-hot label jazz for you. No more manual label headaches.
- Next, I chopped up the data: 80% for training, 20% for validation. Gotta keep that model honest, right? Also, shuffled the batches—don't want the thing picking up on some weird order and thinking that's important.
- Now, let's talk about data augmentation. Because, honestly, if you don't mess with your images a bit, you're just asking for overfitting. I tossed in some random flips, spins, zooms, and even fiddled with the contrast. All that chaos only happens during training, though—validation data stays squeaky clean. No cheating.
- Oh, and I made sure all the pixel values landed between 0 and 1 (thank you, `Rescaling (1/255)). That makes the training smoother and stops things from exploding numerically.
- So yeah, without all these steps, you're basically asking your model to learn from a hot mess. This way, it actually finds the good stuff and generalizes, instead of just memorizing the training set like a parrot.

Model Architecture:

The image bracket model was erected using a successional Convolutional Neural Network (CNN) armature enforced via Keras. The network is designed to prize spatial scales from images and learn meaningful features for accurate beast bracket. The armature consists of the following layers:

- Data Augmentation Layers Applied arbitrary flips, reels, thrums, and discrepancy adaptations to increase training data variability.
- Rescaling Subcaste regularized image pixel values from (0, 255) to (0, 1) for briskly and more stable training.
- Convolutional Layers Three Conv2D layers with adding sludge sizes (e.g., 32, 64, 128), each followed by a ReLU activation and MaxPooling2D to reduce spatial confines.
- Powerhouse Layers Added between complication blocks to help overfitting by aimlessly disabling neurons during training.
- Flatten Layer Converted the 2D point maps into a 1D vector.

- Completely Connected (thick) Layers Included one or further thick layers with ReLU activation to learn complex patterns.
- Affair Subcaste A thick subcaste with 15 units (for each class) and a Softmax activation to produce class chances.

This armature was featherlight yet important enough to achieve high delicacy on the small dataset without taking GPU coffers.

Training Configuration:

The model was trained using the TensorFlow Keras API, with settings chosen to balance training effectiveness and delicacy. The following configuration was used:

- Loss Function CategoricalCrossentropy, suitable for multi-class bracket with one-hot decoded markers.
- Optimizer Adam, an adaptive literacy rate optimizer known for fast confluence and stability.
- Evaluation Metric Accuracy, to cover the proportion of rightly classified images.
- Batch Size 32 images per batch.
- Ages 200 training cycles, which handed sufficient time for confluence without overfitting.

The training process was run entirely on a CPU, as the dataset was small and the model featherlight. Data addition and rescaling were applied in real- time during each time using Keras preprocessing layers.

Confirmation delicacy and loss were covered after each time to estimate conception performance. No beforehand stopping or learning rate scheduling was needed due to the model's harmonious confluence.

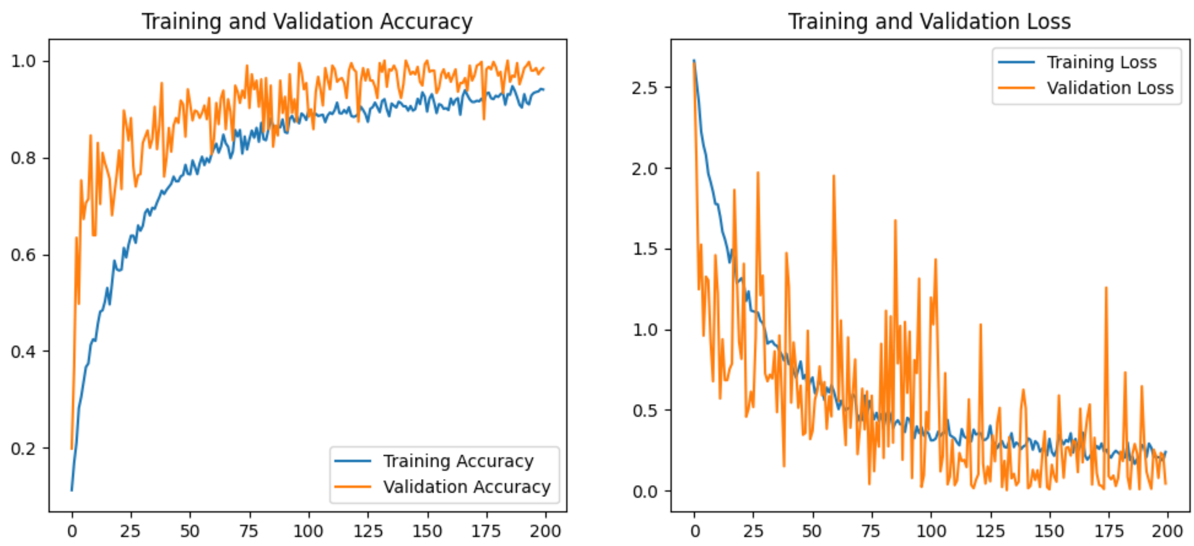
This configuration proved to be effective, achieving high training and confirmation delicacy with smooth loss angles, indicating stable and effective training behaviour throughout the process.

Results and Evaluation:

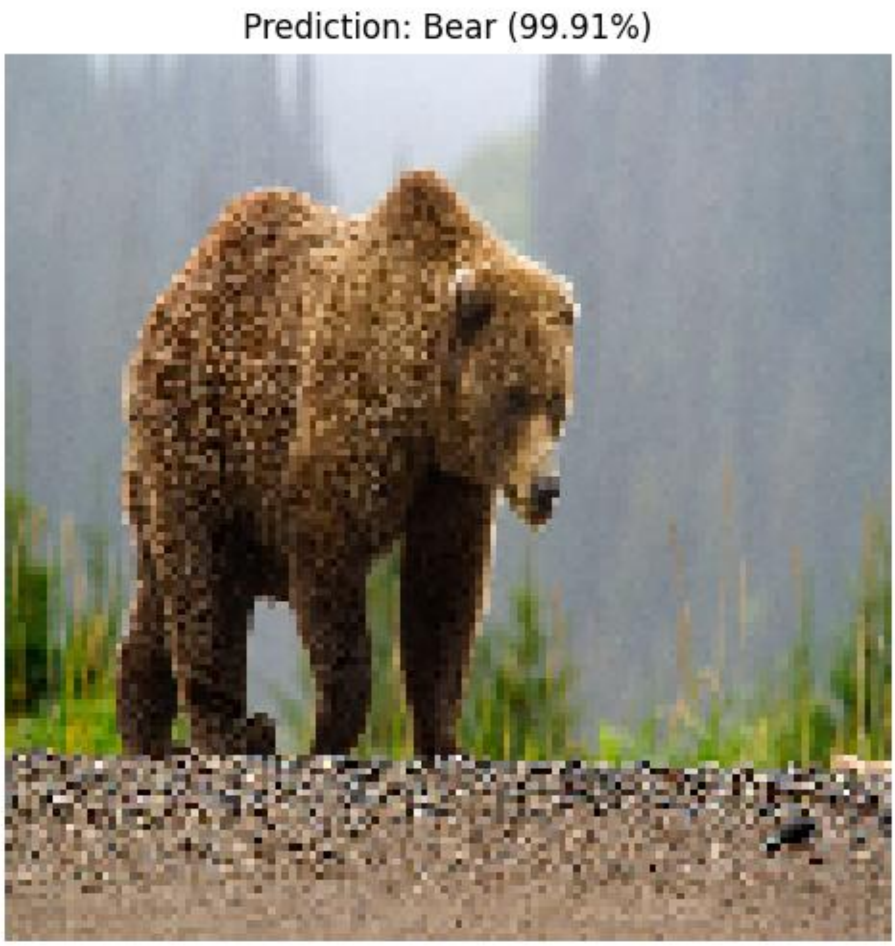
- The model performed very well on both the training and the evidence sets, indicating that it could have generalized from a rather small dataset.
- Final Training accuracy: 96
- Final evidence accuracy: 98

- In the course of the 200 training sessions, the loss for evidence-training and training datasets consistently decreased and the sensitivity gradually grew. There was no evidence of material overfitting in the models, due to the inclusion of hustler layers and data augmentation paths.
- Lineplots learned accuracy and loss of other criteria across epochs. These plots confirmed smooth convergence, as the evidence angles only slightly deviated from the training angles.
- Sample prognostications were also evaluated on evidence images, and the model accurately predicted most images, including those with different backgrounds and lightening.
- Although we did not make use of a confusion matrix, visual inspection of the predictions revealed that the position of class- position delicacy was high for all higher orders.
- In conclusion, this study confirms the efficiency of a well- tuned CNN architecture for multi- class image type based on feathery resources. The model is robust and can be applied to an extend-to-real-time or push- to-deploy operation.

Results Graph:



Sample Output:



Challenges Faced:

Although the dataset and model were fairly simple, many challenges arose during the development process:

1. Class Imbalance:

Some beast classes had slightly smaller images than others, which needed careful observation to ensure that the model was not prejudiced toward further frequent classes.

2. Overfitting threat:

Due to the small dataset size, the model snappily began to study training images. This was eased by introducing powerhouse layers and applying data addition to instinctively increase data diversity.

3. Augmentation Tuning:

Chancing the right balance in addition parameters was pivotal. Too important de formation affected the image quality and hurt delicacy,
while too little had no effect on conception.

4. Model Complexity:

Originally, deeper models were tested but redounded in slower training without significant delicacy earnings. The armature was optimized
to be featherlight and effective for CPU training.

5. Confirmation Monitoring:

Without using calls like
EarlyStopping, covering training manually was necessary to ensure the model di d not begin to overfit in after ages.

These challenges handed precious literacy gests and guided architectural and preprocessing opinions that led to a stable, high- performing model.

Future Scope:

- Though the present model does achieve great results on a small and clean dataset, there could be many improvements and extensions to the present project:
- Larger & More Diverse Data: Extending our data by including more animals or higher diversity in images (backgrounds, illumination, views) would evaluate how well the model would generalize and how robust it is.
- Transfer Learning: Using pretrained models will definitely help out (e.g., VGG16, ResNet, or MobileNet) especially when dealing with large datasets -- :) it will help to increase the accuracy and save time for training.
- Real-Time Deployment: Embed the trained model to a real-time application (running on either mobile app or web application) using TensorFlow Lite or TensorFlow. js size and inference time can be reduced using the techniques such as quantization or pruning, which can be helpful to make it work efficiently on edge devices or low-resource environments.
- Explainable AI (XAI): By employing models such as Grad-CAM or LIME, we can understand which areas of an image are being considered by CNNs, making model decisions transparent and trustworthy.

- Confusion Matrix and Metrics: Knowledge of a confusion matrix, precision, recall, and F1-score, would give more insight on how the model is doing per class, showing where it may apply more effort.

Conclusions:

This project effectively proves the strength and applicability of CNN in animal multi-class image classification with a small and clear animal dataset. Despite small dataset and being trained on a normal machine with limited power, the model had an impressive accuracy, confirming that with the correct preprocessing, data augmentation, and well-tuned architecture, a good performance can be achieved without GPU support.

The experiment also demonstrated the similar challenges like class imbalance, overfitting which was successfully handled with dropout layers and weak augmentation. Finally, this work demonstrates that even a small CNN can be meaningful, and it provides a strong basis for larger, scalable, deployable and interpretable models.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout, Rescaling
from tensorflow.keras.layers import RandomFlip, RandomRotation,
RandomZoom, RandomContrast
import numpy as np
import matplotlib.pyplot as plt
import os

DATA_DIR = 'E:\\Projects\\animal_classification\\Animal
Classification\\dataset'

IMAGE_SIZE = (128, 128)
BATCH_SIZE = 32
VALIDATION_SPLIT = 0.2

train_ds = tf.keras.utils.image_dataset_from_directory(
    DATA_DIR,
    labels='inferred',
    label_mode='categorical',
    image_size=IMAGE_SIZE,
    interpolation='nearest',
    batch_size=BATCH_SIZE,
    shuffle=True,
    validation_split=VALIDATION_SPLIT,
    subset='training',
    seed=123
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    DATA_DIR,
    labels='inferred',
    label_mode='categorical',
    image_size=IMAGE_SIZE,
    interpolation='nearest',
    batch_size=BATCH_SIZE,
    shuffle=False,
    validation_split=VALIDATION_SPLIT,
    subset='validation',
    seed=123
)
```

```
Found 1944 files belonging to 15 classes.
Using 1556 files for training.
Found 1944 files belonging to 15 classes.
Using 388 files for validation.
```

```python
class_names = train_ds.class_names
NUM_CLASSES = len(class_names)
```

```python
print(f"Detected {NUM_CLASSES} classes: {class_names}")

AUTOTUNE = tf.data.AUTOTUNE
train_ds =
train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

data_augmentation = Sequential([
    RandomFlip("horizontal_and_vertical"),
    RandomRotation(0.2),
    RandomZoom(0.2),
    RandomContrast(0.2),
], name="data_augmentation")
```

```
Detected 15 classes: ['Bear', 'Bird', 'Cat', 'Cow', 'Deer', 'Dog',
'Dolphin', 'Elephant', 'Giraffe', 'Horse', 'Kangaroo', 'Lion',
'Panda', 'Tiger', 'Zebra']
```

```python
model = Sequential([
    data_augmentation,
    Rescaling(1./255),
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_SIZE[0],
IMAGE_SIZE[1], 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(NUM_CLASSES, activation='softmax')
])

model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model.summary()

EPOCHS = 200

history = model.fit(
    train_ds,
    epochs=EPOCHS,
    validation_data=val_ds,
)
```

```python
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(EPOCHS)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

model.save('animal_classifier.keras')
print("Model saved as 'animal_classifier.keras'")
```

```
C:\Users\tanis\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\keras\src\layers\convolutional\
base_conv.py:99: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                         ┃ Output Shape              ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ data_augmentation (Sequential)       │ ?                         │    0
(unbuilt) │
├──────────────────────────────────────┼───────────────────────────┼──────────
─────┤
│ rescaling (Rescaling)                │ ?                         │    0
(unbuilt) │
├──────────────────────────────────────┼───────────────────────────┼──────────
─────┤
│ conv2d (Conv2D)                      │ ?                         │    0
(unbuilt) │
```

```
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ max_pooling2d (MaxPooling2D)    │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ conv2d_1 (Conv2D)               │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ max_pooling2d_1 (MaxPooling2D)  │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ conv2d_2 (Conv2D)               │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ max_pooling2d_2 (MaxPooling2D)  │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ flatten (Flatten)               │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ dense (Dense)                   │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ dropout (Dropout)               │ ?                      │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ dense_1 (Dense)                 │ ?                      │   0
(unbuilt) │
└─────────────────────────────────┴────────────────────────┴──────────
───────┘

 Total params: 0 (0.00 B)

 Trainable params: 0 (0.00 B)

 Non-trainable params: 0 (0.00 B)

Epoch 1/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 50s 677ms/step - accuracy: 0.0960 - loss:
2.7392 - val_accuracy: 0.1985 - val_loss: 2.6428
Epoch 2/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 613ms/step - accuracy: 0.1459 - loss:
```

```
2.5641 - val_accuracy: 0.3660 - val_loss: 2.0371
Epoch 3/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 593ms/step - accuracy: 0.2087 - loss:
2.4206 - val_accuracy: 0.6340 - val_loss: 1.2463
Epoch 4/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 620ms/step - accuracy: 0.2715 - loss:
2.2607 - val_accuracy: 0.4974 - val_loss: 1.5235
Epoch 5/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 596ms/step - accuracy: 0.2912 - loss:
2.1497 - val_accuracy: 0.7526 - val_loss: 0.9594
Epoch 6/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 594ms/step - accuracy: 0.3487 - loss:
2.0438 - val_accuracy: 0.6727 - val_loss: 1.3251
Epoch 7/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 604ms/step - accuracy: 0.3546 - loss:
1.9902 - val_accuracy: 0.7062 - val_loss: 1.3023
Epoch 8/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 600ms/step - accuracy: 0.3525 - loss:
1.9313 - val_accuracy: 0.7139 - val_loss: 0.9221
Epoch 9/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 616ms/step - accuracy: 0.4037 - loss:
1.8503 - val_accuracy: 0.8454 - val_loss: 0.6770
Epoch 10/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 607ms/step - accuracy: 0.4364 - loss:
1.7581 - val_accuracy: 0.6392 - val_loss: 1.4580
Epoch 11/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 617ms/step - accuracy: 0.4165 - loss:
1.7744 - val_accuracy: 0.6392 - val_loss: 1.2209
Epoch 12/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 598ms/step - accuracy: 0.4505 - loss:
1.7574 - val_accuracy: 0.8299 - val_loss: 0.5701
Epoch 13/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 598ms/step - accuracy: 0.4887 - loss:
1.5600 - val_accuracy: 0.7036 - val_loss: 0.9373
Epoch 14/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 599ms/step - accuracy: 0.4793 - loss:
1.5720 - val_accuracy: 0.8093 - val_loss: 0.6839
Epoch 15/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 623ms/step - accuracy: 0.5293 - loss:
1.4714 - val_accuracy: 0.7887 - val_loss: 0.6839
Epoch 16/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 598ms/step - accuracy: 0.5402 - loss:
1.4363 - val_accuracy: 0.7732 - val_loss: 0.7526
Epoch 17/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 593ms/step - accuracy: 0.5105 - loss:
1.5123 - val_accuracy: 0.7552 - val_loss: 0.7866
Epoch 18/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 603ms/step - accuracy: 0.5557 - loss:
1.4071 - val_accuracy: 0.6804 - val_loss: 1.8622
```

```
Epoch 19/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 616ms/step - accuracy: 0.5715 - loss:
1.3324 - val_accuracy: 0.7242 - val_loss: 1.3533
Epoch 20/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 621ms/step - accuracy: 0.5913 - loss:
1.2396 - val_accuracy: 0.7655 - val_loss: 0.9209
Epoch 21/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 601ms/step - accuracy: 0.5773 - loss:
1.2636 - val_accuracy: 0.8144 - val_loss: 0.8158
Epoch 22/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 603ms/step - accuracy: 0.5867 - loss:
1.2482 - val_accuracy: 0.7345 - val_loss: 1.4049
Epoch 23/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 602ms/step - accuracy: 0.6172 - loss:
1.1522 - val_accuracy: 0.8969 - val_loss: 0.4576
Epoch 24/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 609ms/step - accuracy: 0.5779 - loss:
1.3087 - val_accuracy: 0.8686 - val_loss: 0.5080
Epoch 25/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 621ms/step - accuracy: 0.6155 - loss:
1.0873 - val_accuracy: 0.8351 - val_loss: 0.6127
Epoch 26/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 618ms/step - accuracy: 0.6257 - loss:
1.1019 - val_accuracy: 0.8814 - val_loss: 0.5174
Epoch 27/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 595ms/step - accuracy: 0.6621 - loss:
1.0790 - val_accuracy: 0.7809 - val_loss: 0.9482
Epoch 28/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 604ms/step - accuracy: 0.6217 - loss:
1.0910 - val_accuracy: 0.7397 - val_loss: 1.9697
Epoch 29/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 599ms/step - accuracy: 0.6474 - loss:
1.0548 - val_accuracy: 0.7629 - val_loss: 1.2100
Epoch 30/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 595ms/step - accuracy: 0.6496 - loss:
1.0208 - val_accuracy: 0.7655 - val_loss: 1.3318
Epoch 31/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 608ms/step - accuracy: 0.6585 - loss:
1.0008 - val_accuracy: 0.8299 - val_loss: 0.7257
Epoch 32/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 594ms/step - accuracy: 0.6796 - loss:
0.9041 - val_accuracy: 0.8428 - val_loss: 0.6770
Epoch 33/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 593ms/step - accuracy: 0.6869 - loss:
0.9538 - val_accuracy: 0.8557 - val_loss: 0.7184
Epoch 34/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 599ms/step - accuracy: 0.6912 - loss:
0.8939 - val_accuracy: 0.8196 - val_loss: 0.6942
Epoch 35/200
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 595ms/step - accuracy: 0.6736 - loss:
0.9369 - val_accuracy: 0.8376 - val_loss: 0.8627
Epoch 36/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 612ms/step - accuracy: 0.7009 - loss:
0.9027 - val_accuracy: 0.9046 - val_loss: 0.4854
Epoch 37/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 618ms/step - accuracy: 0.7031 - loss:
0.8443 - val_accuracy: 0.8170 - val_loss: 0.9608
Epoch 38/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 601ms/step - accuracy: 0.7177 - loss:
0.8645 - val_accuracy: 0.8660 - val_loss: 0.6042
Epoch 39/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 592ms/step - accuracy: 0.7642 - loss:
0.7395 - val_accuracy: 0.9536 - val_loss: 0.1506
Epoch 40/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 600ms/step - accuracy: 0.7470 - loss:
0.7856 - val_accuracy: 0.7603 - val_loss: 1.4716
Epoch 41/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 605ms/step - accuracy: 0.7294 - loss:
0.7852 - val_accuracy: 0.7990 - val_loss: 1.2621
Epoch 42/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 616ms/step - accuracy: 0.7246 - loss:
0.8221 - val_accuracy: 0.8608 - val_loss: 0.5445
Epoch 43/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 599ms/step - accuracy: 0.7427 - loss:
0.7126 - val_accuracy: 0.8119 - val_loss: 0.9179
Epoch 44/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 600ms/step - accuracy: 0.7665 - loss:
0.6755 - val_accuracy: 0.8660 - val_loss: 0.7541
Epoch 45/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 603ms/step - accuracy: 0.7472 - loss:
0.7470 - val_accuracy: 0.8814 - val_loss: 0.5126
Epoch 46/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 604ms/step - accuracy: 0.7235 - loss:
0.8407 - val_accuracy: 0.8711 - val_loss: 0.6504
Epoch 47/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 611ms/step - accuracy: 0.7548 - loss:
0.7020 - val_accuracy: 0.9175 - val_loss: 0.3453
Epoch 48/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 31s 623ms/step - accuracy: 0.7787 - loss:
0.6798 - val_accuracy: 0.9098 - val_loss: 0.3581
Epoch 49/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 598ms/step - accuracy: 0.8010 - loss:
0.6008 - val_accuracy: 0.8428 - val_loss: 0.9900
Epoch 50/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 598ms/step - accuracy: 0.7849 - loss:
0.6697 - val_accuracy: 0.9407 - val_loss: 0.3188
Epoch 51/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 597ms/step - accuracy: 0.7717 - loss:
```

```
0.6830 - val_accuracy: 0.9098 - val_loss: 0.3688
Epoch 52/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 597ms/step - accuracy: 0.7997 - loss:
0.6036 - val_accuracy: 0.8763 - val_loss: 0.5629
Epoch 53/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 29s 599ms/step - accuracy: 0.8020 - loss:
0.6019 - val_accuracy: 0.8969 - val_loss: 0.6184
Epoch 54/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 30s 616ms/step - accuracy: 0.7550 - loss:
0.7506 - val_accuracy: 0.8943 - val_loss: 0.7691
Epoch 55/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 24s 495ms/step - accuracy: 0.7887 - loss:
0.6044 - val_accuracy: 0.8814 - val_loss: 0.5895
Epoch 56/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 16s 337ms/step - accuracy: 0.8080 - loss:
0.5545 - val_accuracy: 0.8918 - val_loss: 0.6723
Epoch 57/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 341ms/step - accuracy: 0.8092 - loss:
0.6214 - val_accuracy: 0.9072 - val_loss: 0.3825
Epoch 58/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 338ms/step - accuracy: 0.8110 - loss:
0.5531 - val_accuracy: 0.8789 - val_loss: 0.5847
Epoch 59/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 337ms/step - accuracy: 0.7992 - loss:
0.6265 - val_accuracy: 0.9227 - val_loss: 0.4595
Epoch 60/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 349ms/step - accuracy: 0.8050 - loss:
0.6373 - val_accuracy: 0.8067 - val_loss: 1.9501
Epoch 61/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 337ms/step - accuracy: 0.8235 - loss:
0.5564 - val_accuracy: 0.8376 - val_loss: 1.3702
Epoch 62/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 16s 332ms/step - accuracy: 0.8284 - loss:
0.4982 - val_accuracy: 0.9124 - val_loss: 0.5532
Epoch 63/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 16s 335ms/step - accuracy: 0.8192 - loss:
0.5469 - val_accuracy: 0.8686 - val_loss: 1.0538
Epoch 64/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 16s 336ms/step - accuracy: 0.8459 - loss:
0.4624 - val_accuracy: 0.9227 - val_loss: 0.5016
Epoch 65/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 346ms/step - accuracy: 0.8384 - loss:
0.5062 - val_accuracy: 0.9381 - val_loss: 0.2818
Epoch 66/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 369ms/step - accuracy: 0.8394 - loss:
0.4945 - val_accuracy: 0.8582 - val_loss: 0.9490
Epoch 67/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 363ms/step - accuracy: 0.8394 - loss:
0.4493 - val_accuracy: 0.9201 - val_loss: 0.3892
```

```
Epoch 68/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 353ms/step - accuracy: 0.7809 - loss:
0.6760 - val_accuracy: 0.8943 - val_loss: 0.6049
Epoch 69/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 346ms/step - accuracy: 0.8060 - loss:
0.6029 - val_accuracy: 0.8351 - val_loss: 0.8123
Epoch 70/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 338ms/step - accuracy: 0.8656 - loss:
0.4188 - val_accuracy: 0.9485 - val_loss: 0.2256
Epoch 71/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 342ms/step - accuracy: 0.8448 - loss:
0.4534 - val_accuracy: 0.9356 - val_loss: 0.3890
Epoch 72/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 346ms/step - accuracy: 0.8585 - loss:
0.4424 - val_accuracy: 0.9124 - val_loss: 0.6312
Epoch 73/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 339ms/step - accuracy: 0.8009 - loss:
0.6148 - val_accuracy: 0.9381 - val_loss: 0.3789
Epoch 74/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 346ms/step - accuracy: 0.8421 - loss:
0.4483 - val_accuracy: 0.9227 - val_loss: 0.6145
Epoch 75/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 350ms/step - accuracy: 0.8368 - loss:
0.4913 - val_accuracy: 0.9897 - val_loss: 0.0404
Epoch 76/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 353ms/step - accuracy: 0.8279 - loss:
0.5393 - val_accuracy: 0.9021 - val_loss: 0.5877
Epoch 77/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 351ms/step - accuracy: 0.8489 - loss:
0.4437 - val_accuracy: 0.9716 - val_loss: 0.1200
Epoch 78/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 367ms/step - accuracy: 0.8519 - loss:
0.4646 - val_accuracy: 0.9407 - val_loss: 0.4235
Epoch 79/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 359ms/step - accuracy: 0.8616 - loss:
0.4052 - val_accuracy: 0.9588 - val_loss: 0.2722
Epoch 80/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.8395 - loss:
0.4490 - val_accuracy: 0.9046 - val_loss: 0.9092
Epoch 81/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 357ms/step - accuracy: 0.8652 - loss:
0.4193 - val_accuracy: 0.9613 - val_loss: 0.2022
Epoch 82/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 357ms/step - accuracy: 0.8302 - loss:
0.5001 - val_accuracy: 0.8531 - val_loss: 1.1137
Epoch 83/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 354ms/step - accuracy: 0.8475 - loss:
0.4845 - val_accuracy: 0.9639 - val_loss: 0.2739
Epoch 84/200
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 353ms/step - accuracy: 0.8780 - loss:
0.4210 - val_accuracy: 0.8582 - val_loss: 1.0792
Epoch 85/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.8817 - loss:
0.3920 - val_accuracy: 0.9485 - val_loss: 0.2965
Epoch 86/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 369ms/step - accuracy: 0.8728 - loss:
0.4154 - val_accuracy: 0.8222 - val_loss: 1.6733
Epoch 87/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.8620 - loss:
0.3998 - val_accuracy: 0.8660 - val_loss: 0.7875
Epoch 88/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 354ms/step - accuracy: 0.8574 - loss:
0.4098 - val_accuracy: 0.8454 - val_loss: 1.0201
Epoch 89/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.8646 - loss:
0.3717 - val_accuracy: 0.9588 - val_loss: 0.1907
Epoch 90/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 352ms/step - accuracy: 0.8816 - loss:
0.3980 - val_accuracy: 0.8918 - val_loss: 1.0449
Epoch 91/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 357ms/step - accuracy: 0.8614 - loss:
0.4297 - val_accuracy: 0.9253 - val_loss: 0.6051
Epoch 92/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 355ms/step - accuracy: 0.8545 - loss:
0.4457 - val_accuracy: 0.8557 - val_loss: 0.9849
Epoch 93/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 19s 392ms/step - accuracy: 0.8748 - loss:
0.3957 - val_accuracy: 0.9716 - val_loss: 0.0778
Epoch 94/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 354ms/step - accuracy: 0.8874 - loss:
0.3135 - val_accuracy: 0.8918 - val_loss: 0.8085
Epoch 95/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 368ms/step - accuracy: 0.8697 - loss:
0.4037 - val_accuracy: 0.9201 - val_loss: 0.7278
Epoch 96/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 357ms/step - accuracy: 0.8892 - loss:
0.3317 - val_accuracy: 0.8737 - val_loss: 1.3129
Epoch 97/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.8510 - loss:
0.4480 - val_accuracy: 0.9948 - val_loss: 0.0228
Epoch 98/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 354ms/step - accuracy: 0.8791 - loss:
0.3459 - val_accuracy: 0.9768 - val_loss: 0.0960
Epoch 99/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.9061 - loss:
0.3020 - val_accuracy: 0.9407 - val_loss: 0.4869
Epoch 100/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 357ms/step - accuracy: 0.8838 - loss:
```

```
0.3276 - val_accuracy: 0.9536 - val_loss: 0.3760
Epoch 101/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.8896 - loss:
0.3061 - val_accuracy: 0.8840 - val_loss: 1.1959
Epoch 102/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 354ms/step - accuracy: 0.8989 - loss:
0.3093 - val_accuracy: 0.8995 - val_loss: 1.0291
Epoch 103/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 354ms/step - accuracy: 0.8876 - loss:
0.3172 - val_accuracy: 0.8582 - val_loss: 1.4319
Epoch 104/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 352ms/step - accuracy: 0.8934 - loss:
0.3619 - val_accuracy: 0.9356 - val_loss: 0.7789
Epoch 105/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 355ms/step - accuracy: 0.8825 - loss:
0.3246 - val_accuracy: 0.9665 - val_loss: 0.1189
Epoch 106/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.8996 - loss:
0.3251 - val_accuracy: 0.9407 - val_loss: 0.2137
Epoch 107/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 362ms/step - accuracy: 0.8848 - loss:
0.3677 - val_accuracy: 0.9072 - val_loss: 0.7277
Epoch 108/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 352ms/step - accuracy: 0.8809 - loss:
0.3995 - val_accuracy: 0.9845 - val_loss: 0.0387
Epoch 109/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 359ms/step - accuracy: 0.8654 - loss:
0.3919 - val_accuracy: 0.9613 - val_loss: 0.0863
Epoch 110/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 353ms/step - accuracy: 0.8892 - loss:
0.3642 - val_accuracy: 0.9407 - val_loss: 0.3160
Epoch 111/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 354ms/step - accuracy: 0.8992 - loss:
0.3037 - val_accuracy: 0.9897 - val_loss: 0.0333
Epoch 112/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 354ms/step - accuracy: 0.8934 - loss:
0.3187 - val_accuracy: 0.9820 - val_loss: 0.0622
Epoch 113/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 368ms/step - accuracy: 0.9154 - loss:
0.2671 - val_accuracy: 0.9562 - val_loss: 0.2333
Epoch 114/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 357ms/step - accuracy: 0.9012 - loss:
0.3509 - val_accuracy: 0.9613 - val_loss: 0.1826
Epoch 115/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 353ms/step - accuracy: 0.8849 - loss:
0.3758 - val_accuracy: 0.9639 - val_loss: 0.1904
Epoch 116/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 353ms/step - accuracy: 0.8995 - loss:
0.3152 - val_accuracy: 0.9536 - val_loss: 0.1457
```

```
Epoch 117/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.8953 - loss:
0.2802 - val_accuracy: 0.9227 - val_loss: 0.5656
Epoch 118/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 355ms/step - accuracy: 0.9065 - loss:
0.3016 - val_accuracy: 0.9794 - val_loss: 0.0353
Epoch 119/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 359ms/step - accuracy: 0.8994 - loss:
0.3260 - val_accuracy: 0.9948 - val_loss: 0.0148
Epoch 120/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 355ms/step - accuracy: 0.8815 - loss:
0.3624 - val_accuracy: 0.9820 - val_loss: 0.0666
Epoch 121/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 351ms/step - accuracy: 0.8941 - loss:
0.3037 - val_accuracy: 0.9768 - val_loss: 0.1016
Epoch 122/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 354ms/step - accuracy: 0.8805 - loss:
0.3731 - val_accuracy: 0.8737 - val_loss: 1.0286
Epoch 123/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 354ms/step - accuracy: 0.8991 - loss:
0.3542 - val_accuracy: 0.9562 - val_loss: 0.1673
Epoch 124/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 354ms/step - accuracy: 0.8975 - loss:
0.3114 - val_accuracy: 0.9845 - val_loss: 0.0435
Epoch 125/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 353ms/step - accuracy: 0.8803 - loss:
0.3307 - val_accuracy: 0.9588 - val_loss: 0.1499
Epoch 126/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 357ms/step - accuracy: 0.8882 - loss:
0.3494 - val_accuracy: 0.9820 - val_loss: 0.0572
Epoch 127/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 357ms/step - accuracy: 0.9038 - loss:
0.2999 - val_accuracy: 0.9588 - val_loss: 0.3420
Epoch 128/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.9209 - loss:
0.2476 - val_accuracy: 0.9588 - val_loss: 0.1812
Epoch 129/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 354ms/step - accuracy: 0.9280 - loss:
0.2308 - val_accuracy: 0.9459 - val_loss: 0.4237
Epoch 130/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 363ms/step - accuracy: 0.9077 - loss:
0.2795 - val_accuracy: 0.9227 - val_loss: 0.5126
Epoch 131/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 362ms/step - accuracy: 0.8978 - loss:
0.2843 - val_accuracy: 0.9948 - val_loss: 0.0221
Epoch 132/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.9054 - loss:
0.3496 - val_accuracy: 0.9768 - val_loss: 0.1839
Epoch 133/200
```

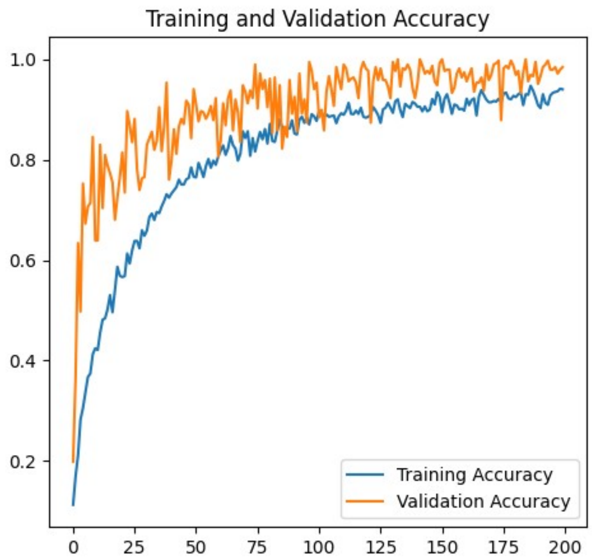
```
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 355ms/step - accuracy: 0.9228 - loss:
0.2703 - val_accuracy: 1.0000 - val_loss: 0.0029
Epoch 134/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 357ms/step - accuracy: 0.9115 - loss:
0.2843 - val_accuracy: 0.9459 - val_loss: 0.3315
Epoch 135/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 357ms/step - accuracy: 0.8854 - loss:
0.3631 - val_accuracy: 0.9820 - val_loss: 0.0759
Epoch 136/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 358ms/step - accuracy: 0.9129 - loss:
0.2653 - val_accuracy: 0.9794 - val_loss: 0.1044
Epoch 137/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.9235 - loss:
0.2680 - val_accuracy: 0.9897 - val_loss: 0.0333
Epoch 138/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 371ms/step - accuracy: 0.8897 - loss:
0.3498 - val_accuracy: 0.9794 - val_loss: 0.0547
Epoch 139/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 371ms/step - accuracy: 0.9241 - loss:
0.2330 - val_accuracy: 0.9433 - val_loss: 0.5016
Epoch 140/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 353ms/step - accuracy: 0.9249 - loss:
0.2402 - val_accuracy: 0.9227 - val_loss: 0.6250
Epoch 141/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 358ms/step - accuracy: 0.9117 - loss:
0.2718 - val_accuracy: 0.9510 - val_loss: 0.5088
Epoch 142/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 355ms/step - accuracy: 0.9075 - loss:
0.3041 - val_accuracy: 1.0000 - val_loss: 0.0147
Epoch 143/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 362ms/step - accuracy: 0.8967 - loss:
0.3152 - val_accuracy: 0.9871 - val_loss: 0.0296
Epoch 144/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.9078 - loss:
0.2538 - val_accuracy: 0.9716 - val_loss: 0.1290
Epoch 145/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 355ms/step - accuracy: 0.9035 - loss:
0.2724 - val_accuracy: 0.9768 - val_loss: 0.0658
Epoch 146/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.9060 - loss:
0.2732 - val_accuracy: 0.9716 - val_loss: 0.1267
Epoch 147/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 360ms/step - accuracy: 0.9254 - loss:
0.2392 - val_accuracy: 0.9871 - val_loss: 0.0220
Epoch 148/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 366ms/step - accuracy: 0.8977 - loss:
0.3027 - val_accuracy: 0.9562 - val_loss: 0.2223
Epoch 149/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 360ms/step - accuracy: 0.9289 - loss:
```

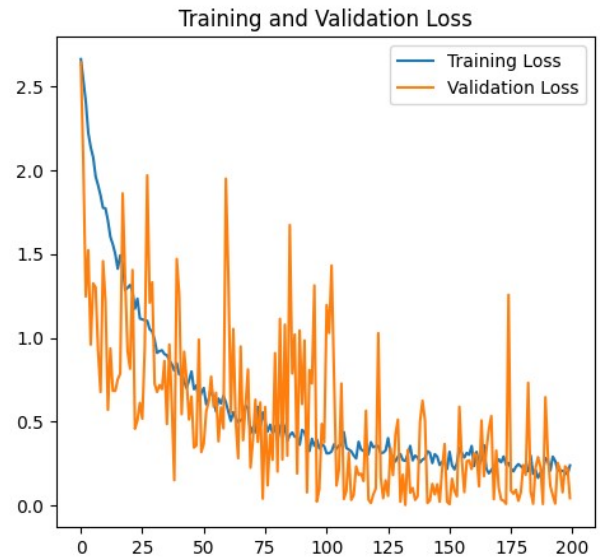
```
0.2725 - val_accuracy: 0.9485 - val_loss: 0.3669
Epoch 150/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 358ms/step - accuracy: 0.9223 - loss:
0.2288 - val_accuracy: 0.9923 - val_loss: 0.0217
Epoch 151/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 355ms/step - accuracy: 0.8924 - loss:
0.3316 - val_accuracy: 1.0000 - val_loss: 0.0076
Epoch 152/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 358ms/step - accuracy: 0.9256 - loss:
0.2392 - val_accuracy: 0.9768 - val_loss: 0.1598
Epoch 153/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 354ms/step - accuracy: 0.9315 - loss:
0.2137 - val_accuracy: 0.9794 - val_loss: 0.0875
Epoch 154/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 355ms/step - accuracy: 0.9147 - loss:
0.2321 - val_accuracy: 0.9794 - val_loss: 0.0548
Epoch 155/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.8884 - loss:
0.3556 - val_accuracy: 0.9330 - val_loss: 0.5892
Epoch 156/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 359ms/step - accuracy: 0.9205 - loss:
0.2652 - val_accuracy: 0.9433 - val_loss: 0.2727
Epoch 157/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.9114 - loss:
0.2597 - val_accuracy: 0.9716 - val_loss: 0.0795
Epoch 158/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 357ms/step - accuracy: 0.9017 - loss:
0.3223 - val_accuracy: 0.9820 - val_loss: 0.2644
Epoch 159/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 369ms/step - accuracy: 0.9052 - loss:
0.2728 - val_accuracy: 0.9639 - val_loss: 0.2694
Epoch 160/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 355ms/step - accuracy: 0.9078 - loss:
0.3011 - val_accuracy: 0.9742 - val_loss: 0.2167
Epoch 161/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 353ms/step - accuracy: 0.9119 - loss:
0.2435 - val_accuracy: 0.9562 - val_loss: 0.3107
Epoch 162/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 356ms/step - accuracy: 0.9137 - loss:
0.2730 - val_accuracy: 0.9691 - val_loss: 0.2991
Epoch 163/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 355ms/step - accuracy: 0.8981 - loss:
0.2893 - val_accuracy: 0.9820 - val_loss: 0.1145
Epoch 164/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 27s 546ms/step - accuracy: 0.9204 - loss:
0.2585 - val_accuracy: 0.9356 - val_loss: 0.5077
Epoch 165/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 42s 861ms/step - accuracy: 0.8931 - loss:
0.3987 - val_accuracy: 0.9536 - val_loss: 0.1747
Epoch 166/200
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 36s 743ms/step - accuracy: 0.9204 - loss:
0.2347 - val_accuracy: 0.9536 - val_loss: 0.3349
Epoch 167/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 38s 782ms/step - accuracy: 0.9423 - loss:
0.1808 - val_accuracy: 0.9639 - val_loss: 0.4547
Epoch 168/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 37s 754ms/step - accuracy: 0.9367 - loss:
0.1804 - val_accuracy: 0.9381 - val_loss: 0.5335
Epoch 169/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 35s 713ms/step - accuracy: 0.9188 - loss:
0.2452 - val_accuracy: 0.9897 - val_loss: 0.0372
Epoch 170/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 44s 899ms/step - accuracy: 0.9181 - loss:
0.2440 - val_accuracy: 0.9588 - val_loss: 0.3276
Epoch 171/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 42s 854ms/step - accuracy: 0.9147 - loss:
0.2964 - val_accuracy: 0.9691 - val_loss: 0.1067
Epoch 172/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 41s 826ms/step - accuracy: 0.9185 - loss:
0.2887 - val_accuracy: 0.9897 - val_loss: 0.0352
Epoch 173/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 38s 778ms/step - accuracy: 0.9247 - loss:
0.2582 - val_accuracy: 0.9923 - val_loss: 0.0286
Epoch 174/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 37s 754ms/step - accuracy: 0.9261 - loss:
0.2392 - val_accuracy: 0.9974 - val_loss: 0.0099
Epoch 175/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 36s 734ms/step - accuracy: 0.9253 - loss:
0.2377 - val_accuracy: 0.8789 - val_loss: 1.2572
Epoch 176/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 38s 780ms/step - accuracy: 0.9401 - loss:
0.2017 - val_accuracy: 0.9820 - val_loss: 0.0888
Epoch 177/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 37s 752ms/step - accuracy: 0.9395 - loss:
0.1776 - val_accuracy: 0.9871 - val_loss: 0.0704
Epoch 178/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 35s 717ms/step - accuracy: 0.9187 - loss:
0.2321 - val_accuracy: 0.9820 - val_loss: 0.0934
Epoch 179/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 35s 706ms/step - accuracy: 0.9189 - loss:
0.2597 - val_accuracy: 0.9974 - val_loss: 0.0282
Epoch 180/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 34s 689ms/step - accuracy: 0.9341 - loss:
0.2191 - val_accuracy: 0.9871 - val_loss: 0.0773
Epoch 181/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 35s 705ms/step - accuracy: 0.9203 - loss:
0.2256 - val_accuracy: 0.9691 - val_loss: 0.2415
Epoch 182/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 33s 664ms/step - accuracy: 0.9340 - loss:
```

```
0.1910 - val_accuracy: 0.9768 - val_loss: 0.1829
Epoch 183/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 34s 685ms/step - accuracy: 0.9319 - loss:
0.2004 - val_accuracy: 0.9330 - val_loss: 0.7322
Epoch 184/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 35s 711ms/step - accuracy: 0.9162 - loss:
0.2704 - val_accuracy: 0.9742 - val_loss: 0.0856
Epoch 185/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 36s 742ms/step - accuracy: 0.9254 - loss:
0.2111 - val_accuracy: 1.0000 - val_loss: 0.0102
Epoch 186/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 36s 731ms/step - accuracy: 0.9273 - loss:
0.2148 - val_accuracy: 0.9562 - val_loss: 0.2245
Epoch 187/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 36s 726ms/step - accuracy: 0.9467 - loss:
0.1592 - val_accuracy: 0.9691 - val_loss: 0.2889
Epoch 188/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 33s 672ms/step - accuracy: 0.9484 - loss:
0.1601 - val_accuracy: 0.9665 - val_loss: 0.2294
Epoch 189/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 34s 698ms/step - accuracy: 0.9275 - loss:
0.1938 - val_accuracy: 0.9948 - val_loss: 0.0086
Epoch 190/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 33s 674ms/step - accuracy: 0.9168 - loss:
0.2633 - val_accuracy: 0.9510 - val_loss: 0.6461
Epoch 191/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 33s 670ms/step - accuracy: 0.8864 - loss:
0.3120 - val_accuracy: 0.9665 - val_loss: 0.3026
Epoch 192/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 34s 690ms/step - accuracy: 0.9327 - loss:
0.2127 - val_accuracy: 0.9845 - val_loss: 0.1167
Epoch 193/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 35s 704ms/step - accuracy: 0.9209 - loss:
0.2678 - val_accuracy: 0.9897 - val_loss: 0.0604
Epoch 194/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 34s 689ms/step - accuracy: 0.9136 - loss:
0.2622 - val_accuracy: 0.9974 - val_loss: 0.0110
Epoch 195/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 33s 674ms/step - accuracy: 0.9336 - loss:
0.1965 - val_accuracy: 0.9794 - val_loss: 0.2545
Epoch 196/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 33s 665ms/step - accuracy: 0.9374 - loss:
0.1973 - val_accuracy: 0.9794 - val_loss: 0.2142
Epoch 197/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 34s 696ms/step - accuracy: 0.9432 - loss:
0.1837 - val_accuracy: 0.9845 - val_loss: 0.0786
Epoch 198/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 25s 512ms/step - accuracy: 0.9332 - loss:
0.2226 - val_accuracy: 0.9716 - val_loss: 0.2337
```

```
Epoch 199/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 16s 335ms/step - accuracy: 0.9362 - loss: 0.2030 - val_accuracy: 0.9794 - val_loss: 0.2209
Epoch 200/200
49/49 ━━━━━━━━━━━━━━━━━━━━ 17s 337ms/step - accuracy: 0.9600 - loss: 0.1611 - val_accuracy: 0.9845 - val_loss: 0.0438
```






```
Model saved as 'animal_classifier.keras'

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os

tf.config.set_visible_devices([], 'GPU')
print("Configured to use CPU only.")

MODEL_PATH = 'animal_classifier.keras'
IMAGE_PATH = "C:\\Users\\tanis\\Downloads\\bhalu.jpg"

IMAGE_SIZE = (128, 128)

CLASS_NAMES = sorted([
    'Bear', 'Bird', 'Cat', 'Cow', 'Deer', 'Dog', 'Dolphin', 'Elephant',
    'Giraffe', 'Horse', 'Kangaroo', 'Lion', 'Panda', 'Tiger', 'Zebra'
])

model = tf.keras.models.load_model(MODEL_PATH)

def preprocess_image(image_path, target_size):
    img = tf.keras.preprocessing.image.load_img(image_path,
```

```
target_size=target_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

test_image_array = preprocess_image(IMAGE_PATH, IMAGE_SIZE)

predictions = model.predict(test_image_array)

predicted_probabilities = predictions[0]
predicted_class_index = np.argmax(predicted_probabilities)
predicted_class_name = CLASS_NAMES[predicted_class_index]
confidence = predicted_probabilities[predicted_class_index] * 100

print(f"\nPredicted class: {predicted_class_name}")
print(f"Confidence: {confidence:.2f}%")

plt.figure(figsize=(6, 6))
display_img = tf.keras.preprocessing.image.load_img(IMAGE_PATH,
target_size=IMAGE_SIZE)
plt.imshow(display_img)
plt.title(f"Prediction: {predicted_class_name} ({confidence:.2f}%)")
plt.axis('off')
plt.show()
```

```
Configured to use CPU only.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 162ms/step

Predicted class: Bear
Confidence: 99.91%
```

Prediction: Bear (99.91%)