



BIRD SPECIES CLASSIFICATION USING CNN IN TENSORFLOW

Gilang Wijanarko

Email : gilangwjnrk@gmail.com

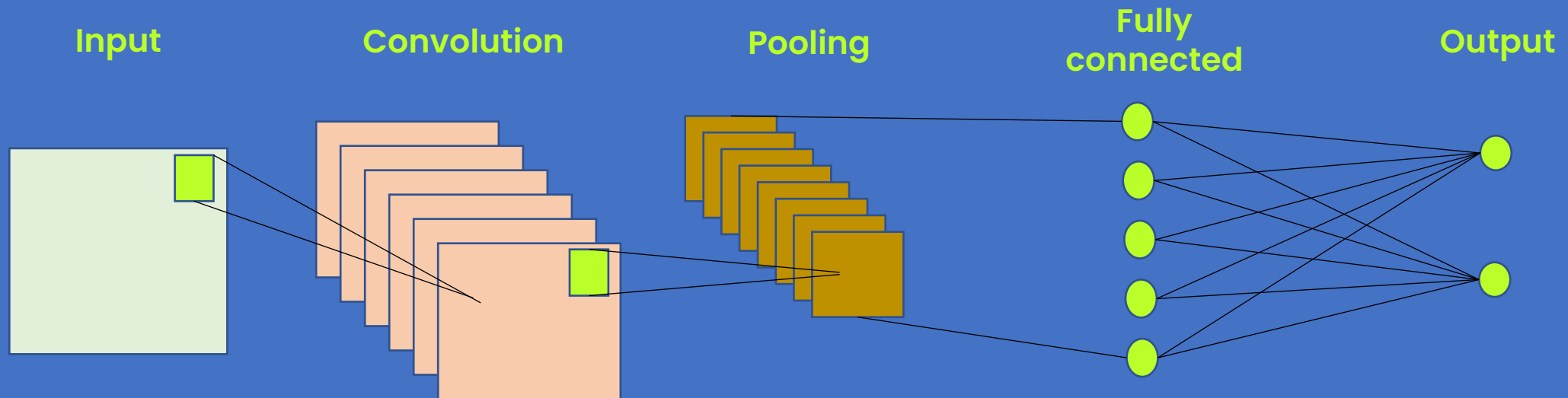
Whatsapp : +6288232217249

CONVOLUTION NEURAL NETWORK

Get to know

Convolutional Neural Network (CNN) is a type of neural network architecture designed to learn spatial patterns from image data such as bitmaps. It uses convolutional layers to detect pixel-level structures—like edges, textures, and shapes—and connects these features to nearby regions to build a hierarchical understanding of the image. This allows CNNs to extract meaningful representations from raw visual input, making them highly effective for tasks like image classification, object detection, and pattern recognition.

CONVOLUTION NEURAL NETWORK



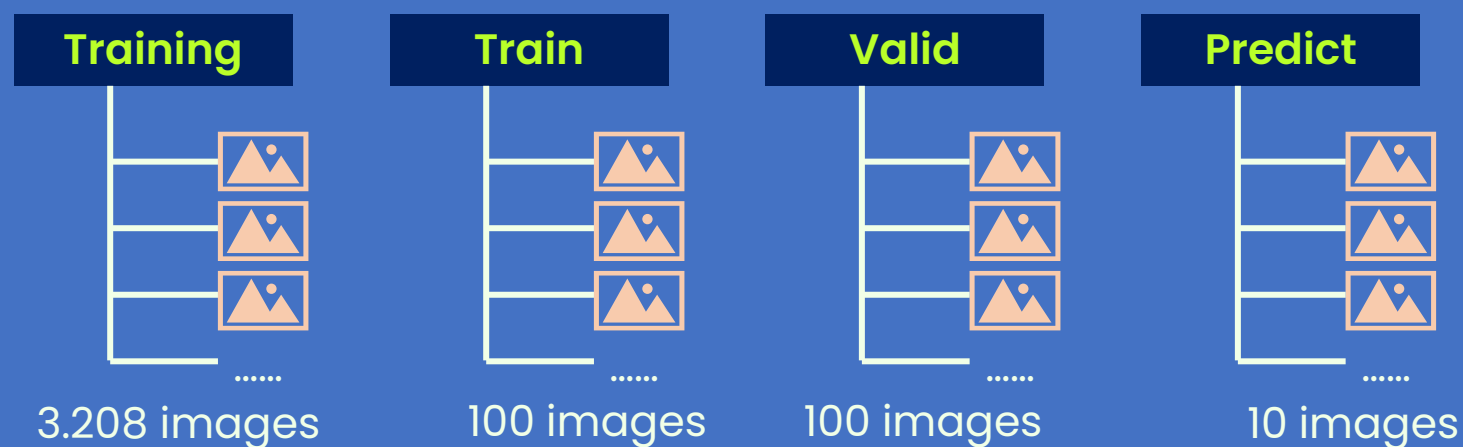
Convolutional Neural Network (CNN) is a neural network model designed to learn spatial patterns from images by detecting local pixel structures and building hierarchical feature representations.

DATASET

The dataset used in this project consists of 20 bird species classes, with 3,208 training images, 100 test images (5 images per species), and 100 validation images (5 images per species). All images are colored JPG files with a resolution of $224 \times 224 \times 3$.

The dataset can be downloaded from:

<https://www.kaggle.com/datasets/umairshahpirzada/birds-20-species-image-classification>



PICTURE EXAMPLE

Abbots Booby



African Pygmy Goose



Alphine Cough



American Goldfinch



The bird poses in the dataset are highly varied. Some images show the entire body, while others are close-up shots. Several images need to be cropped and resized to fit the required 224×224 pixel dimensions. The test and validation images were hand-selected as the "best" examples, so the model is likely to achieve a high accuracy score. The prediction will be performed on images that have never been seen by the model during training.

CNN WORKFLOW

Data Preparation

Data Preprocessing

Train Set
Preprocessing

Test Set
Preprocessing

Building CNN

Import CNN
algorithm

Add Convolution
Layer

Fully Connected
Layer

Output and
Compile Mode

Training with Model

Visualisation and Evaluation

Model
Accuracy

Multiple
Prediction

PYTHON LIBRARY

Tools and libraries included in this project are:



DATA PREPARATION



Goals:

- Mount the dataset from Google Drive, previously downloaded from Kaggle.com.
- Import Python libraries such as TensorFlow and ImageDataGenerator from Keras for data augmentation.
- Perform data augmentation to configure image data for machine learning compatibility.

DATA AUGMENTATION

```
#Data Augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    zoom_range=0.2,
    horizontal_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1,
    fill_mode='nearest'
)
```



Data augmentation is the process of transforming data sources to make them suitable for machine learning models. Various aspects can be adjusted, such as zoom, scale, rotation, and brightness.

Setup

Rescale = 1/255 → Transforms pixel values from 0–255 to a 0–1 scale, allowing the model to learn more efficiently.

Rotation range = 10 → Allows slight rotation (± 10 degrees) to account for bird heads tilted to the left or right.

Zoom range = 0.2 → Applies up to 20% zoom to simulate close-up shots of birds.

Horizontal flip = True → Flips images horizontally to reflect birds facing left or right.

Width shift range = 0.1 → Shifts images horizontally by up to 10% to simulate lateral movement.

Height shift range = 0.1 → Shifts images vertically by up to 10% to simulate vertical movement.

Fill mode = 'nearest' → Fills empty areas created by transformations using the nearest pixel values.

DATA PREPROCESSING



Goals:

- Configure data input pipelines for both training and testing sets to match the CNN architecture.
- Optimize the dataset into efficient batches to accelerate learning and improve memory usage.

PATH TO TRAIN SET

```
#path to drive for training set
training_set = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/Dataset/bird_classification/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)
```

Target size = (224, 224) → Resizes all imported images to 224×224 pixels, matching the input dimensions expected by the CNN architecture.

Batch size = 32 → Processes 32 images per training step, balancing memory efficiency and learning stability.

Class mode = 'categorical' → Specifies multi-class classification; the model will learn to categorize images into 20 distinct classes based on the training set.

20 labels are:

Class Label: {'ABBOTTS BABBLER': 0, 'ABBOTTS BOOBY': 1, 'ABYSSINIAN GROUND HORNBILL': 2, 'AFRICAN CROWNED CRANE': 3, 'AFRICAN EMERALD CUCKOO': 4, 'AFRICAN FIREFINCH': 5, 'AFRICAN OYSTER CATCHER': 6, 'AFRICAN PIED HORNBILL': 7, 'AFRICAN PYGMY GOOSE': 8, 'ALBATROSS': 9, 'ALBERTS TOWHEE': 10, 'ALEXANDRINE PARAKEET': 11, 'ALPINE CHOUGH': 12, 'ALTAMIRA YELLOWTHROAT': 13, 'AMERICAN AVOCET': 14, 'AMERICAN BITTERN': 15, 'AMERICAN COOT': 16, 'AMERICAN FLAMINGO': 17, 'AMERICAN GOLDFINCH': 18, 'AMERICAN KESTREL': 19}

PATH TO TEST SET

```
#path to drive for test set
test_datagen = ImageDataGenerator(rescale = 1./255)
test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/Dataset/bird_classification/test',
                                           target_size = (224, 224),
                                           batch_size = 32,
                                           class_mode = 'categorical')
```

Target size = (224, 224), Batch size = 32, Class mode = 'categorical' | Same as training set.

The test set doesn't require the same data configuration as the training set. Unlike the training set, it doesn't undergo any augmentation process and only needs rescaling to normalize pixel values. The test set contains fewer images (100) compared to the training set (3,208), as its main purpose is to evaluate the model's ability to correctly predict unseen data.

BUILDING CNN



Goals:

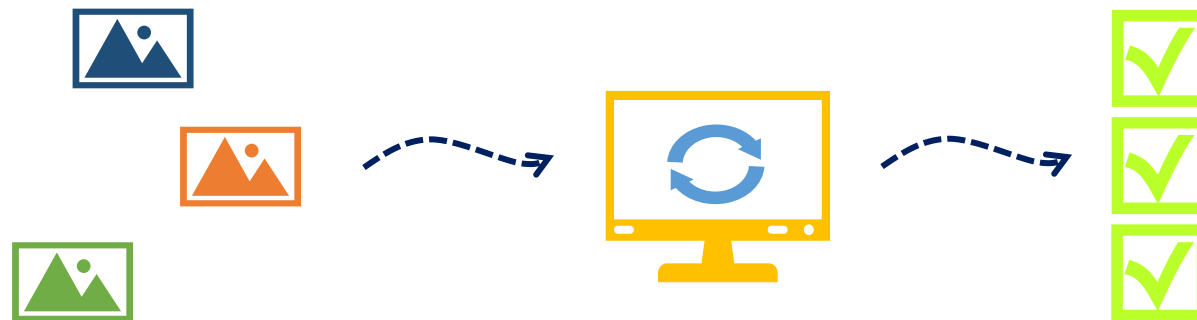
- Configure data input pipelines for both training and testing sets to match the CNN architecture.
- Optimize the dataset into efficient batches to accelerate learning and improve memory usage.

CREATE CNN ALGORITHM FROM SCRATCH

```
# Create CNN Algorithm  
cnn = tf.keras.models.Sequential()
```

Sequential() is one of the classes provided by Keras to build a layered neural network. It allows you to add layers one by one in a linear stack, where the output of each layer becomes the input for the next.

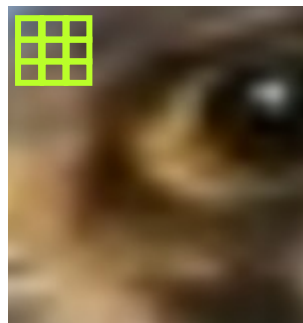
In this project, the CNN model is built from scratch, meaning it does not use transfer learning techniques or pre-trained architectures like ResNet or LeNet. Instead, the model is trained entirely on the current dataset. Since it will be deployed on new, unseen images, it must be capable of adapting to new data through full training.



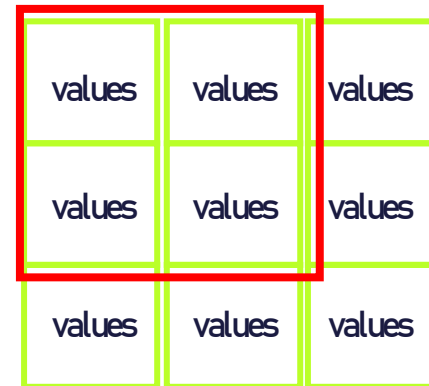
CONVOLUTION LAYER

The convolutional layer is the core component that defines a Convolutional Neural Network (CNN). It transforms local pixel regions into meaningful spatial patterns by applying learnable filters that capture features such as edges, textures, and shapes.

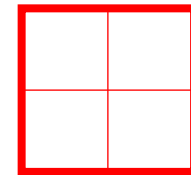
Each convolutional layer learns from individual pixel values and their surrounding context, gradually building feature maps that represent how the network interprets the image at different levels of abstraction from simple edges in early layers to complex objects in deeper layers. Although the configuration may be similar across layers, each convolutional layer uses different filters, allowing the model to recognize distinct pixel patterns and extract increasingly complex features.



Filter detection



Batch normalization



Pooling and dropout

MANAGE CONVOLUTION LAYER

Convolution Layer 1

```
# Manage Convolution Layer
cnn.add(tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(224, 224, 3)))
cnn.add(tf.keras.layers.BatchNormalization())
cnn.add(tf.keras.layers.MaxPooling2D(2))
cnn.add(tf.keras.layers.Dropout(0.1))
```

Setup

Conv2D → is a convolutional layer in Keras that applies a set of filters to the input image, converting local pixel patterns into spatial feature maps.

- ▶ The layer slides **32** different **3×3** filters across the input image. Each filter detects specific features (like edges, corners, textures).
- ▶ **Activation = 'relu'** used to efficiently extract features by introducing non-linearity into the model. It converts all negative values to zero, allowing only positive signals to pass through. This helps maintain stable gradients during backpropagation and speeds up learning.

Batch Normalization → technique that normalizes the inputs of a layer across the current batch of data

MaxPooling2D → layer that reduces the spatial dimensions (height and width) of the feature maps while keeping the most important information. In this case, a 2×2 pooling layer is used with maximum aggregation, which makes the process more efficient.

Dropout → regularization technique used to prevent overfitting in neural networks. The value 0.1 means 10% of the neurons in this layer will be "dropped out" during each training step.

Repeat same process in convolution layer 2 and convolution layer 3

```
# Manage Convolution Layer
cnn.add(tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(224, 224, 3)))
cnn.add(tf.keras.layers.BatchNormalization())
cnn.add(tf.keras.layers.MaxPooling2D(2))
cnn.add(tf.keras.layers.Dropout(0.1))

cnn.add(tf.keras.layers.Conv2D(64, 3, activation='relu'))
cnn.add(tf.keras.layers.BatchNormalization())
cnn.add(tf.keras.layers.MaxPooling2D(2))
cnn.add(tf.keras.layers.Dropout(0.1))

cnn.add(tf.keras.layers.Conv2D(128, 3, activation='relu'))
cnn.add(tf.keras.layers.BatchNormalization())
cnn.add(tf.keras.layers.MaxPooling2D(2))
cnn.add(tf.keras.layers.Dropout(0.1))
```

The convolutional layer is repeated across layer 1, 2, and 3 using the same configuration. Each convolutional layer has the ability to extract features from the image, allowing the model to learn through a hierarchical structure from low-level edges to high-level patterns. Every convolution layer separated by different filter to recognize pixels.

Although the configuration may be similar, each convolutional layer uses different filters (**32, 64, 128**), allowing the model to recognize distinct pixel patterns at each stage. This separation ensures that the network builds a richer and more diverse understanding of the visual data.

By stacking multiple convolutional layers, the model can renew its understanding and gain new perspectives from visual data.

EARLY STOPPING

```
from tensorflow.keras.callbacks import EarlyStopping

# Callback for early stopping
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True,
    verbose=1
)
```

Early stopping is a technique used to halt the training process when a certain condition is met typically when the model's performance on the validation set stops improving. This mechanism helps prevent overfitting, where the model learns the training data too well but performs poorly on unseen data.

Setup

Monitor = 'val_loss' → This tells the model to track the validation loss during training. If val_loss stops improving, early stopping will be triggered.

Patience = 3 → The model will wait for 3 epochs without improvement before stopping. It gives the model a chance to recover before deciding it's done

Restore best weight = True → After stopping, the model will revert to the best weights it had during training.

Verbose = 1 → Enables console output messages like "Epoch X: early stopping triggered" when it happens.

FULLY CONNECTED LAYER

```
#fully connected layer
cnn.add(tf.keras.layers.Flatten())
cnn.add(tf.keras.layers.Dense(128, activation='relu'))
cnn.add(tf.keras.layers.BatchNormalization())
cnn.add(tf.keras.layers.Dropout(0.3))
```

The **fully connected layer** (also known as a **dense layer**) is responsible for transforming the extracted features from previous layers — such as convolution and pooling — into a format suitable for classification. It connects every neuron from the previous layer to every neuron in the current layer, allowing the model to combine and interpret high-level features to make final predictions.

Setup

Flatten → Transforms the multi-dimensional spatial feature maps (e.g., 2D matrices) into a 1D array.

Dense (128, activation='relu') → Adds a fully connected layer with 128 neurons and ReLU activation. This layer learns complex combinations of features and helps the model make decisions based on extracted patterns.

Batch normalization → Normalizes the output stabilizes and speeds up training

Dropout → Randomly deactivates 30% of the neurons during training to prevent overfitting.

CREATE CNN ALGORITHM FROM SCRATCH

```
#output layer with 20 categories  
cnn.add(tf.keras.layers.Dense(20, activation='softmax'))
```

Dense(20) → Adds a fully connected layer with 20 output neurons, each representing one class.

Activation='softmax' → Converts raw scores (logits) into probabilities that sum to 1. The model will choose the class with the highest probability as its prediction.

```
cnn.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),  
            loss='categorical_crossentropy',  
            metrics=['accuracy'])
```

A neural network learns by adjusting its internal parameters through backpropagation, starting from the output layer and working backward, refining its understanding of data with every epoch.

Setup

Learning rate = 0.0005 → Controls how much the model updates its weights during training. A small learning rate like 0.0005 ensures slow but stable learning

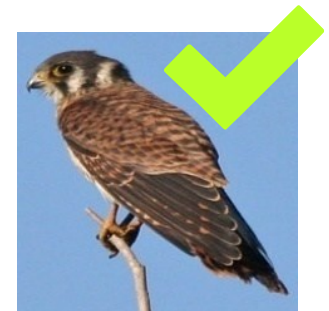
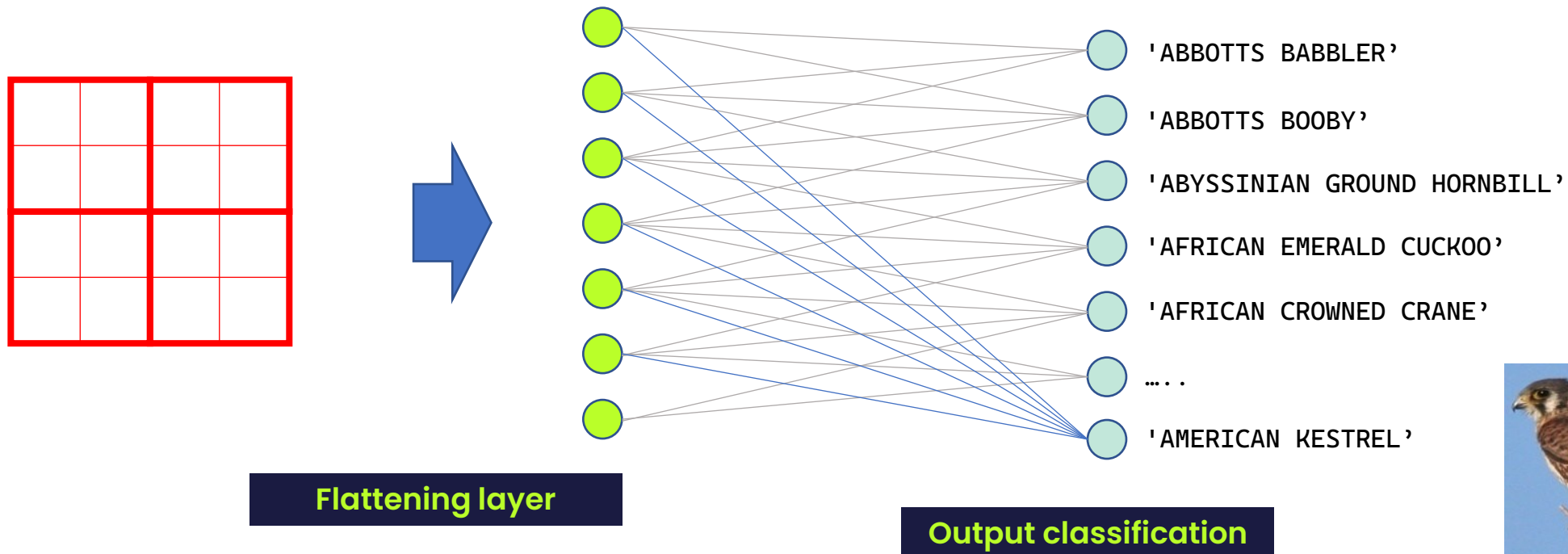
Loss = 'categorical crossentropy' → Used for multi-class classification problems where the target labels are one-hot encoded.

Metrics = 'accuracy' → Tracks how often the model's predictions match the true labels.

FULLY CONNECTED LAYER TO OUTPUT LAYER

The Flatten layer transforms the multi-dimensional output from the pooling layer into a 1D array, making it compatible with fully connected (Dense) layers. This flattened array represents the extracted features in a format that can be processed for classification.

Once the data reaches the **output layer**, the model uses backpropagation, a learning mechanism that adjusts weights across all layers, to minimize the loss function. This process is triggered during compile mode, where the model is configured with an optimizer, loss function, and evaluation metrics.



TRAINING WITH DATA



Goals:

- Train the CNN model using the provided training dataset.
- Monitor validation loss (`val_loss`) and accuracy during training to evaluate model performance.
- Use early stopping to prevent overfitting and determine the optimal number of training epochs.
- Analyze the training history to select the best model configuration for deployment or further testing.

FITTING MODEL AND TRAIN DATA

```
history = cnn.fit(  
    x=training_set,  
    validation_data=test_set,  
    epochs=30,  
    callbacks=[early_stop]  
)
```

The model is trained using the training_set, while its performance is validated on the test_set. The **training process** returns a 'History' object, which is stored in the variable history. This object contains metrics like loss and accuracy over each epoch, and can be used later for data visualization.

Setup

Validation data = test set → This means the model will evaluate its performance on the test_set after each training epoch.

Epoch = 30 → This sets the number of times the model will go through the entire training dataset.

Callbacks = early stop → This allows the training to stop early if a certain condition is met

```
# Callback for early stopping  
early_stop = EarlyStopping(  
    monitor='val_loss',  
    patience=3,  
    restore_best_weights=True,  
    verbose=1  
)
```

LEARNING PROCESS

```
Epoch 1/30
101/101 ————— 1222s 12s/step - accuracy: 0.2274 - loss: 2.7877 - val_accuracy: 0.0500 - val_loss: 11.5152
Epoch 2/30
101/101 ————— 46s 452ms/step - accuracy: 0.4663 - loss: 1.6951 - val_accuracy: 0.0400 - val_loss: 8.4469
Epoch 3/30
101/101 ————— 46s 458ms/step - accuracy: 0.5679 - loss: 1.4216 - val_accuracy: 0.1000 - val_loss: 5.9086
Epoch 4/30
101/101 ————— 45s 447ms/step - accuracy: 0.6244 - loss: 1.2364 - val_accuracy: 0.2200 - val_loss: 3.1436
Epoch 5/30
101/101 ————— 46s 457ms/step - accuracy: 0.6258 - loss: 1.2403 - val_accuracy: 0.4500 - val_loss: 1.9812
```

During the training process, the machine learning model learns step by step. After each epoch, it updates key metrics such as accuracy, loss, validation accuracy, and validation loss, which help track its performance and improvement over time.

Accuracy = Measures how often the model makes correct predictions on the training data (the percentage of correct predictions made by the model on the training set).

Loss = Represents how far off the model's predictions are from the actual labels. Lower is better (numerical value that indicates the model's error during training. It's what the model tries to minimize).

Val accuracy = Measures how often the model makes correct predictions on the validation (test) data (the percentage of correct predictions on unseen data, used to evaluate generalization).

Loss accuracy = Indicates how well the model performs on the validation set in terms of error (the error value on the validation data. If this increases while training loss decreases, it may signal overfitting).


```
Epoch 12/30
101/101 ————— 46s 453ms/step - accuracy: 0.7836 - loss: 0.6931 - val_accuracy: 0.6800 - val_loss: 0.9375
Epoch 13/30
101/101 ————— 48s 478ms/step - 
Epoch 14/30
101/101 ————— 46s 454ms/step - accuracy: 0.7965 - loss: 0.6340 - val_accuracy: 0.7700 - val_loss: 0.7136
Epoch 15/30
101/101 ————— 47s 463ms/step - accuracy: 0.8042 - loss: 0.6156 - val_accuracy: 0.6500 - val_loss: 1.1191
Epoch 16/30
101/101 ————— 46s 453ms/step - accuracy: 0.8229 - loss: 0.5730 - val_accuracy: 0.7200 - val_loss: 0.8826
Epoch 16: early stopping
Restoring model weights from the end of the best epoch: 13.
```

The model stopped training at **epoch 16**, even though it was set to run for **30 epochs**. This happened because the validation loss (val_loss) showed **no significant improvement for 3 consecutive epochs**, based on the early stopping configuration. As a result, the early stopping mechanism was triggered to prevent overfitting and save training time. The final model weights were restored from **epoch 13**, which had the best validation performance.

Accuracy = 0.7795 → After training, the model achieved 78% accuracy on the training data. This means the CNN correctly classified bird images nearly 78% of the time during training..

Loss = 0.7094 → The training loss was 0.7094, indicating the model's average error on the training set. Lower loss values suggest better learning and more confident predictions.

Val accuracy = 0.8000 → The model reached 81.00% accuracy on unseen validation data. This shows strong generalization, meaning the model performs well on new bird images it hasn't seen before.

Val loss = 0.5322 → The validation loss was 0.5131, which is lower than the training loss. This suggests the model is not overfitting and may even perform better on unseen data than on the training set.

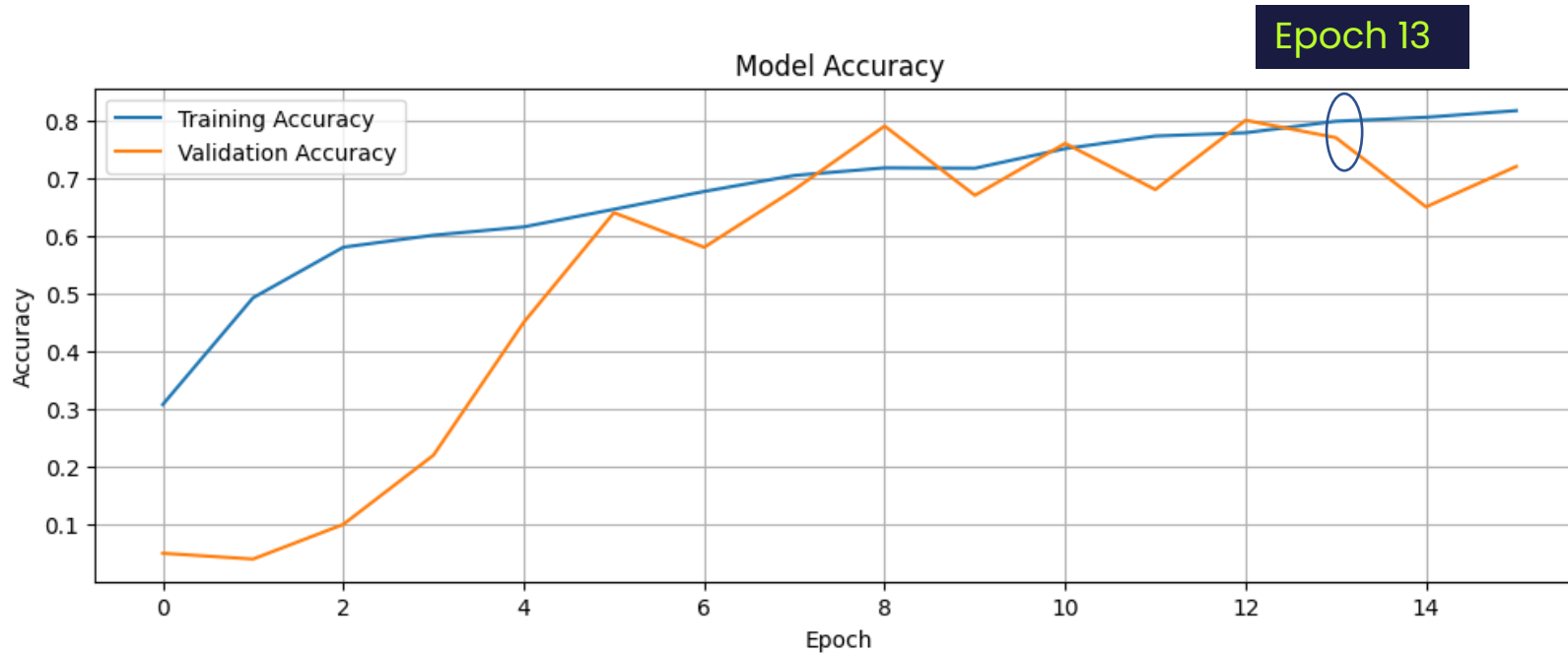
VISUALIZATION AND EVALUATION



Goals:

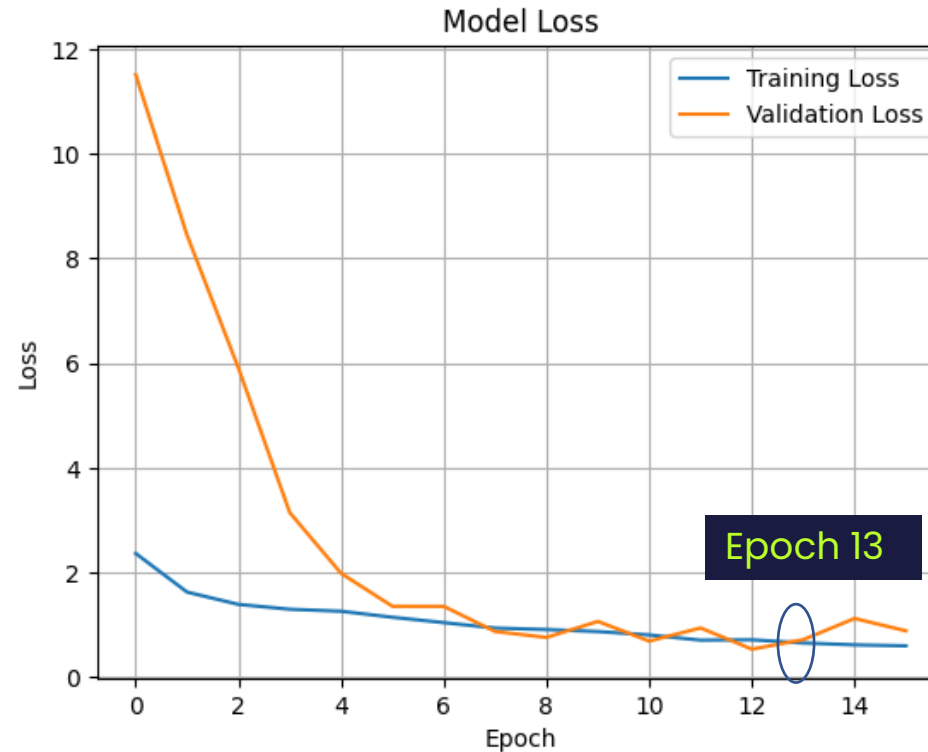
- To demonstrate how well your CNN model learned, visualize key metrics: (1) Accuracy over epochs shows how often the model made correct predictions during training and validation and (2) Loss over epochs reflects how far off the model's predictions were from the actual labels.
- Input images that were not part of the training or validation sets to evaluate generalization.

LEARNING VISUALIZATION



During training, the model's accuracy steadily improved across epochs, with training accuracy showing a consistent upward trend. However, validation accuracy began to fluctuate, signaling potential overfitting. To mitigate this, training was halted at epoch 16, with the best-performing model captured at epoch 13 (patience = 3) where validation accuracy peaked. This early stopping strategy ensures the model retains its generalization capability on unseen data.

LEARNING VISUALIZATION



Throughout training, the model's loss consistently declined, indicating a reduction in prediction error and improved learning. Notably, validation loss dropped sharply during the first four epochs—suggesting that early adjustments to weights and biases significantly enhanced the model's ability to generalize to unseen images. This rapid convergence between training and validation loss reflects a well-tuned learning process, where the model quickly grasps core patterns before fine-tuning its internal representations.

MULTIPLE PREDICTION

```
import os
import numpy as np
from tensorflow.keras.preprocessing import image
```

The tensorflow.keras.preprocessing.image library is used to simplify the process of loading and preparing new images for prediction in a machine learning model. By using functions like load_img() and img_to_array(), the model can receive input in the correct format without needing to repeat the entire augmentation pipeline from scratch.

```
1/1 _____ 0s 29ms/step
Gambar: 2.jpg → Prediksi: AFRICAN CROWNED CRANE
1/1 _____ 0s 29ms/step
Gambar: 3.jpg → Prediksi: AFRICAN CROWNED CRANE
1/1 _____ 0s 28ms/step
Gambar: 4.jpg → Prediksi: AFRICAN CROWNED CRANE
1/1 _____ 0s 29ms/step
Gambar: 1.jpg → Prediksi: AFRICAN CROWNED CRANE
1/1 _____ 0s 28ms/step
Gambar: 5.jpg → Prediksi: AFRICAN CROWNED CRANE
1/1 _____ 0s 35ms/step
Gambar: 7.jpeg → Prediksi: AMERICAN BITTERN
1/1 _____ 0s 28ms/step
Gambar: 10.jpg → Prediksi: ALEXANDRINE PARAKEET
1/1 _____ 0s 28ms/step
Gambar: 9.jpg → Prediksi: ALBATROSS
1/1 _____ 0s 30ms/step
Gambar: 8.jpg → Prediksi: ALBATROSS
1/1 _____ 0s 29ms/step
Gambar: 6.jpg → Prediksi: ABBOTTS BOOBY
```

After completing model configuration, the machine learning system was tested on 10 unseen bird images. It successfully predicted 9 out of 10 species correctly, demonstrating strong generalization performance. This result highlights the model's ability to extract meaningful features and apply learned patterns to new data—despite never encountering these specific images during training.

MULTIPLE PREDICTION

Prediksi: AFRICAN CROWNED CRANE



Correct
prediction

Prediksi: AFRICAN CROWNED CRANE



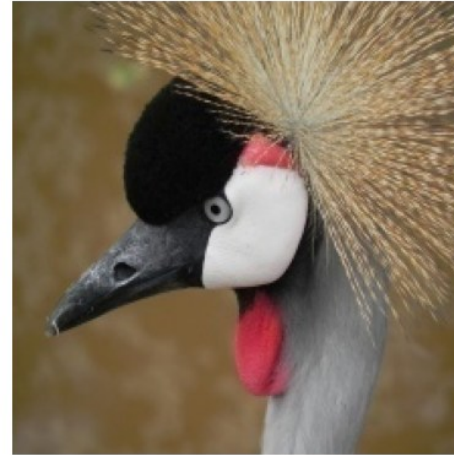
Correct
prediction

Prediksi: AFRICAN CROWNED CRANE



Correct
prediction

Prediksi: AFRICAN CROWNED CRANE



Correct
prediction

Prediksi: AFRICAN CROWNED CRANE



Correct
prediction

MULTIPLE PREDICTION

Prediksi: AMERICAN BITTERN



Correct
prediction

Prediksi: ALEXANDRINE PARAKEET



Correct
prediction

Prediksi: ALBATROSS



Correct
prediction

Prediksi: ALBATROSS



Correct
prediction

Prediksi: ABBOTTS BOOBY



Incorrect
prediction

DISCUSSION

This machine learning model was trained to classify bird species across 20 distinct classes. Despite the limited dataset just 3,208 training images, it managed to avoid overfitting, suggesting that the model learned generalizable patterns rather than memorizing fine-grained bird characteristics. Rather than focusing on detailed genus-level traits, the model appears to grasp broader visual cues, such as posture, silhouette, and plumage contrast.

While far from perfect, this prototype demonstrates how a simple Convolutional Neural Network (CNN) can begin to interpret biological diversity through pattern recognition. Given the vast visual variation among bird species, this experiment serves as a foundational step toward understanding how machines perceive nature not through taxonomy, but through texture, shape, and rhythm.



This model has shown promising results on a small dataset, but its true potential lies in scaling. With access to a larger and more diverse image set, the model could learn richer representations and improve classification accuracy across bird species with subtle visual differences.

Moreover, by leveraging transfer learning—using pre-trained CNN architectures like ResNet, VGG, or EfficientNet, the model can inherit powerful feature extractors trained on massive datasets (e.g., ImageNet). With adjusted configurations and fine tuning, it can adapt efficiently to bird classification tasks, even with limited new data. This approach is widely adopted by developers working at scale, allowing faster convergence, reduced training time, and better generalization especially in domains like biodiversity, remote sensing, and medical imaging.




EVALUATION

This project currently faces several challenges that limit its classification performance:

- The test set is too small to provide meaningful evaluation. With only ~7% overall accuracy and near-zero precision/recall across most classes, the model struggles to generalize. Increasing the test set size is critical for reliable performance metrics.
- The training and testing datasets are both limited, leading to poor representation of many bird species. The model disproportionately predicts “African Crowned Crane,” indicating overfitting to dominant classes and lack of diversity in predictors.
- The model struggles to distinguish species in the confusion matrix, likely due to insufficient training data and lack of full-body bird images. It performs better when birds are clearly visible and unobstructed, but gets confused when multiple objects appear in the frame.

To improve performance and robustness:

- Expand the dataset with more balanced class representation and varied bird poses.
 - Include full-body bird images to help the model learn holistic features like shape, posture, and plumage.
 - Apply transfer learning to leverage pre-trained models that already understand general visual patterns.
 - Use smarter augmentation to simulate occlusions, lighting changes, and background clutter.
 - Analyze misclassifications to identify which species are most confused and why—this can guide targeted data collection.
- 



THANK'S!

Let's get connected!

www.linkedin.com/in/gilang-wijanarko-1b6b4b16b