Lifelike Bot Builders

Rewa Rammal
Elias Charbel Salameh

Outline

01 Problem

02 Data Preparation

03 Model Training and testing

04 Results and challenges

Introduction

Implementing a CNN model to accurately classify animal images, trained on Kaggle Animals-10 dataset, containing 10 different animal classes, using deep learning and data augmentation techniques.

To access our dataset: https://www.kaggle.com/datasets/alessiocorrado99/animals10/data

Problem

We aim to solve the problem of accurately identifying and classifying animal images from diverse categories.

This will benefit applications in wildlife monitoring, veterinary diagnostics, and educational resources, providing enhanced accuracy and efficiency in image-based animal recognition tasks.

Data Preparation

❖ EDA

Project (3).ipynb

```python
# Initialize a dictionary to store the count of images in each class
class_counts = {}

# Loop through each class directory and count the images
for class_name in classes:
    class_dir = os.path.join(data_dir, class_name)
    num_images = len(os.listdir(class_dir))
    class_counts[class_name] = num_images

# Convert the dictionary to a DataFrame for easy plotting
class_counts_df = pd.DataFrame(list(class_counts.items()), columns=['Class', 'Number of Images'])

# Plot the class distribution
plt.figure(figsize=(10, 6))
sns.barplot(x='Class', y='Number of Images', data=class_counts_df)
plt.title('Class Distribution')
plt.xticks(rotation=45)
plt.show()
```

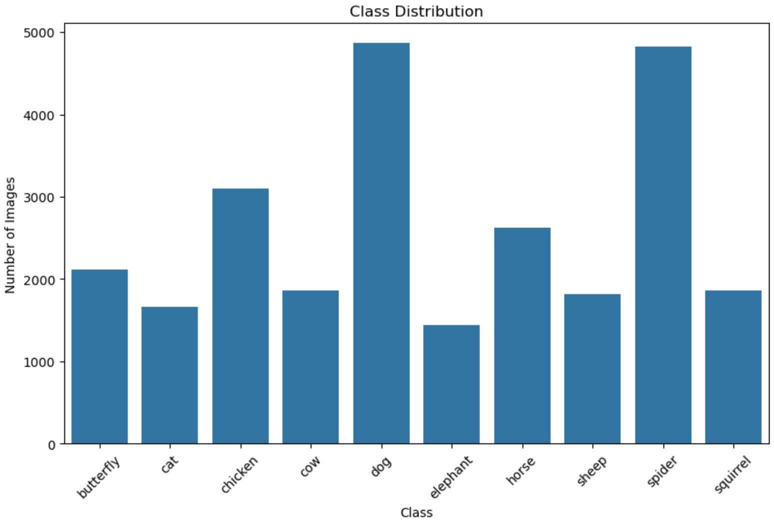
The class distribution plot shows the number of images in each class, highlighting any class imbalances.

Project (3).ipynb

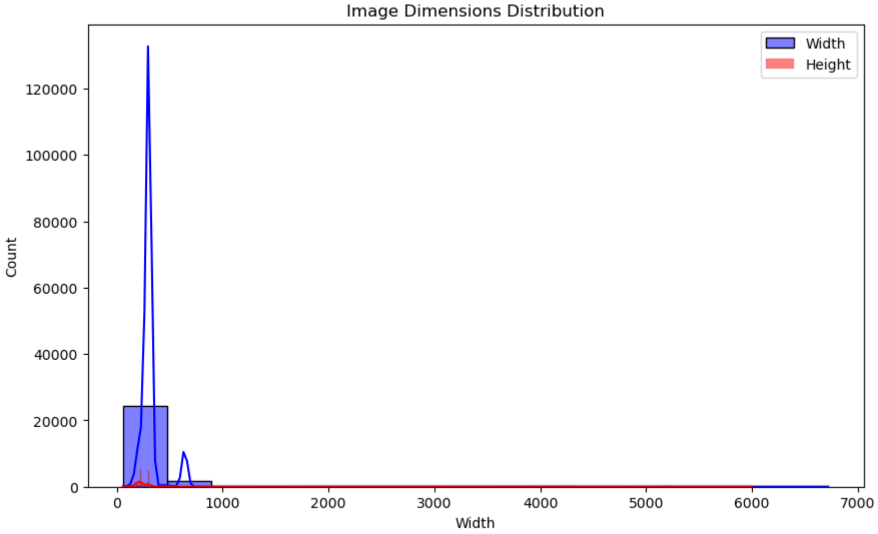
```python
# Checking image dimensions
image_shapes = []

for class_name in classes:
    class_dir = os.path.join(data_dir, class_name)
    for image_name in os.listdir(class_dir):
        image_path = os.path.join(class_dir, image_name)
        image = Image.open(image_path)
        image_shapes.append(image.size)

# Convert to DataFrame for analysis
image_shapes_df = pd.DataFrame(image_shapes, columns=['Width', 'Height'])

# Plot image dimensions
plt.figure(figsize=(10, 6))
sns.histplot(data=image_shapes_df, x='Width', kde=True, color='blue', label='Width')
sns.histplot(data=image_shapes_df, x='Height', kde=True, color='red', label='Height')
plt.title('Image Dimensions Distribution')
plt.legend()
plt.show()
```

The image dimensions distribution plot displays the range and frequency of image widths and heights.

Data Preparation



Plot of class distribution



Plot of image dimension variety

❖ Data Preparation

From **tensorflow.keras**:

- **ImageDataGenerator** is used to generate data. Data was resized and underwent augmentation techniques such as rotation, shifting, shearing, zooming, and horizontal flipping for the training set to enhance diversity.

- **flow_from_directory** is used to load images from the specified directories. Data was resized, batched for efficiency, categorized for training, and shuffled for diversity, ensuring proper formatting and augmentation for robust model training and evaluation.

Project (3).ipynb

```
target_size = 264

train_data_generator = ImageDataGenerator(rescale=1./255,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
train_generator = train_data_generator.flow_from_directory(
        train_dir,
        target_size=(target_size, target_size),
        batch_size=64,
        class_mode='categorical',
        shuffle=True)

val_data_generator = ImageDataGenerator(rescale=1./255)
val_generator = val_data_generator.flow_from_directory(
        val_dir,
        target_size=(target_size, target_size),
        batch_size=64,
        class_mode='categorical',
        shuffle=False)

test_data_generator = ImageDataGenerator(rescale=1./255)
test_set = test_data_generator.flow_from_directory(
        test_dir,
        target_size=(target_size, target_size),
        batch_size=64,
        class_mode='categorical',
        shuffle=False)
```

Model Architecture

The model has multiple Conv2D layers with increasing filters (32 to 1024), each followed by batch normalization and max pooling.

After flattening, it includes a dense layer with 512 neurons, batch normalization, dropout, and a final dense layer with 10 neurons using softmax for classification.

It is compiled with the Adam optimizer and categorical crossentropy loss for multi-class classification.

Augmentation proved to solve overfitting better than the dropout layer method. Adding the two methods had a minimal impact on our model.

Project (3).ipynb

```
model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(target_size, target_size, 3)),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Conv2D(256, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Conv2D(512, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Conv2D(1024, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(512, activation='relu'),
        BatchNormalization(),
        Dense(10, activation='softmax')
])
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 262, 262, 32)      896

 batch_normalization (BatchN  (None, 262, 262, 32)     128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 131, 131, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 129, 129, 64)      18496

 batch_normalization_1 (Batc  (None, 129, 129, 64)     256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 64, 64, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 62, 62, 128)       73856

...
Total params: 8,401,098
Trainable params: 8,396,042
Non-trainable params: 5,056
```

Model training and testing

Project (3).ipynb

```python
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Define callbacks:
early_stopping = EarlyStopping(monitor='val_accuracy', patience=8, restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model2.h5', monitor='val_accuracy', save_best_only=True)
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.1, patience=4, verbose=1, mode='auto', min_delta=0.0001, cooldown=0,
min_lr=0)

print("Model initialized and Callbacks defined.")
```

Callbacks included, ModelCheckpoint, and ReduceLROnPlateau to stop training once the validation accuracy is not improving, saving the best model and reduce learning rate.

Project (3).ipynb

```python
# Train the model with callbacks
history = model.fit(train_generator, epochs=50, validation_data=val_generator,
                    callbacks=[early_stopping, model_checkpoint, reduce_lr])
```

The model was trained using train_generator for 50 epochs, with val_generator as the validation set, and the best model based on validation accuracy was saved.

Project (3).ipynb

```python
# Create a figure
plt.figure(figsize=(12, 5))

# Subplot for Loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

# Subplot for Accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

# Show the plots
plt.tight_layout()
plt.show()
```

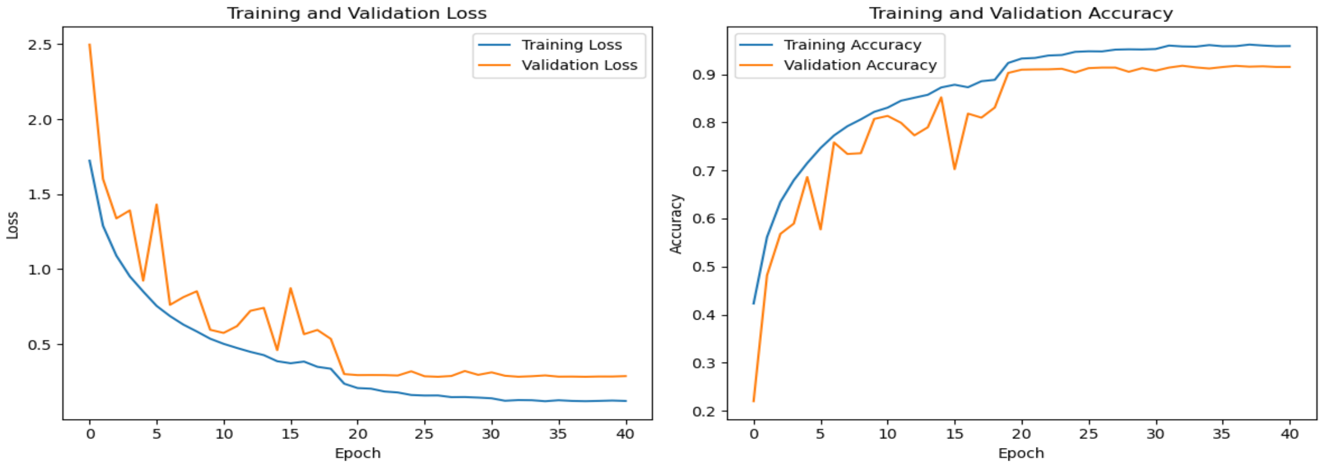
Visual inspections of model predictions were conducted using sample images.

Project (3).ipynb

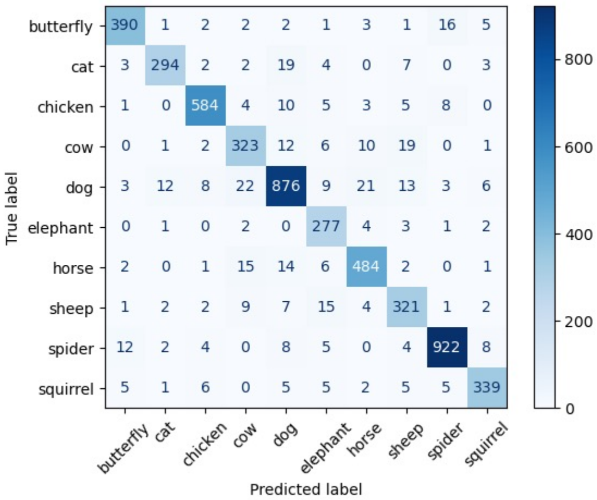
```python
# Evaluate the model
loss, accuracy = model.evaluate(val_generator)
print("Validation Loss:", loss)
print("Validation Accuracy:", accuracy)
```

After training, the model was evaluated on the validation set to determine final validation loss and accuracy

Results of Training and validation



Results of validation

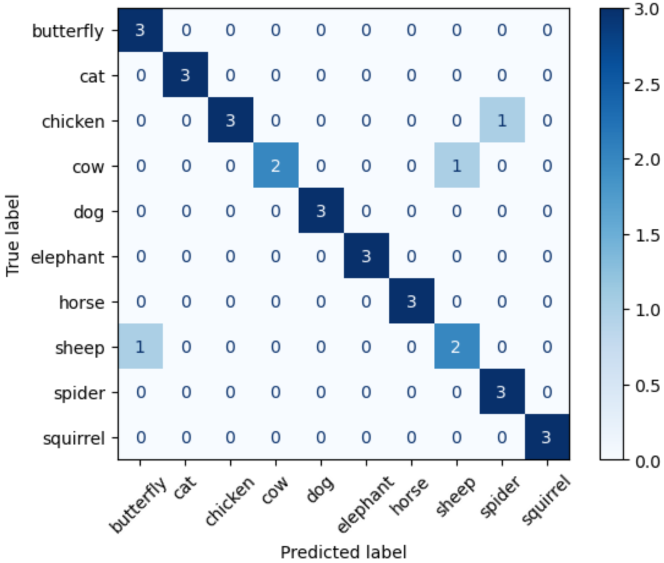


Validation confusion matrix

```
82/82 [==============================] - 7s 89ms/step
Classification Report:
=====================
              precision    recall  f1-score   support

   butterfly       0.94      0.92      0.93       423
         cat       0.94      0.88      0.91       334
     chicken       0.96      0.94      0.95       620
         cow       0.85      0.86      0.86       374
         dog       0.92      0.90      0.91       973
    elephant       0.83      0.96      0.89       290
       horse       0.91      0.92      0.92       525
       sheep       0.84      0.88      0.86       364
      spider       0.96      0.96      0.96       965
    squirrel       0.92      0.91      0.92       373

    accuracy                           0.92      5241
   macro avg       0.91      0.91      0.91      5241
weighted avg       0.92      0.92      0.92      5241
```

The classification report

Results of testing



The accuracy on these 25 images is: 0.9032258064516129

Challenges

- Time constraints
- Imbalanced dataset
- Image sizes selection
- Variety of hyperparameters
- Prevent overfitting

Next step

Gathering more data to balance the dataset and create a more accurate model

Implement additional classes to cover more species

Optimize our model to accurately predict more images

Implementing our model in an environment to solve a reallife problem

Thank you!