CNN Implementation

Assignment-2

Reg.no 24MSD7047

A Convolutional Neural Network (CNN) is a specialized type of deep learning neural network, primarily designed for processing data with a grid-like topology, such as images. They are particularly effective in tasks like image recognition, classification, object detection, and segmentation.

In [63]:

```
import os, glob
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter

import tensorflow as tf
from tensorflow.keras.utils import load_img, img_to_array
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation, MaxPooling2D, Dropout, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
```

In [65]:

```
data_dir = "/Users/pravalika/Downloads/PandasBears"
assert os.path.isdir(data_dir), "data_dir path is wrong or doesn't exist!"
```

In [67]:

```
IMAGE_EXTS = ('.jpg','.jpeg','.png','.bmp','.tif','.tiff','.gif','.webp')
IMG_SIZE = (32,32)

def load_dataset(data_dir, target_size=IMG_SIZE):
    files = glob.glob(os.path.join(data_dir,"**","*.*"), recursive=True)
    img_files = [f for f in files if f.lower().endswith(IMAGE_EXTS)]
    print("Total images found:", len(img_files))
    if len(img_files) == 0:
        raise ValueError("No images found. Check folder structure or file extensions.")

    X, y, class_names = [], [], []
    for f in img_files:
        label = os.path.basename(os.path.dirname(f))  # parent folder name used as class
        if label not in class_names:
            class_names.append(label)
        idx = class_names.index(label)

        try:
            img = load_img(f, target_size=target_size)
            arr = img_to_array(img)
            X.append(arr)
            y.append(idx)
        except Exception as e:
            print("Skipping", f, "error:", e)
            continue

    X = np.array(X, dtype=np.float32)/255.0
    y = np.array(y, dtype=np.int32)
    print("Dataset shape:", X.shape, y.shape)
    print("Classes:", class_names)
    print("Counts:", Counter(y))
    return X, y, class_names
```

In [69]:

```
X, y, class_names = load_dataset(data_dir)
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
for i, idx in enumerate(range(min(6, len(X)))):
    plt.subplot(2,3,i+1)
    plt.imshow(X[idx])
    plt.title(class_names[y[idx]])
    plt.axis("off")
plt.tight_layout()
plt.show()
```

```
Total images found: 1200
Dataset shape: (1200, 32, 32, 3) (1200,)
Classes: ['Bears', 'Pandas']
Counts: Counter({0: 600, 1: 600})
```

Loading [MathJax]/extensions/Safe.js

In [71]:

```
stratify_arg = y if len(np.unique(y)) > 1 else None

X_trainval, X_test, y_trainval, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=stratify_arg
)

stratify_arg2 = y_trainval if len(np.unique(y_trainval)) > 1 else None
X_train, X_val, y_train, y_val = train_test_split(
    X_trainval, y_trainval, test_size=0.2, random_state=42, stratify=stratify_arg2
)

print("Train:", X_train.shape, "Val:", X_val.shape, "Test:", X_test.shape)
```

```
Train: (768, 32, 32, 3) Val: (192, 32, 32, 3) Test: (240, 32, 32, 3)
```

In [73]:

```
from tensorflow.keras import Input

input_shape = (32,32,3)

model = Sequential([
    Input(shape=input_shape),

    Conv2D(32, (3,3), padding='same', use_bias=False),
    BatchNormalization(), Activation('relu'),
    MaxPooling2D((2,2)), Dropout(0.25),

    Conv2D(64, (3,3), padding='same', use_bias=False),
    BatchNormalization(), Activation('relu'),
    MaxPooling2D((2,2)), Dropout(0.25),

    Conv2D(128, (3,3), padding='same', use_bias=False),
    BatchNormalization(), Activation('relu'),
    MaxPooling2D((2,2)), Dropout(0.25),

    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(len(class_names), activation='softmax')
])

model.summary()
```

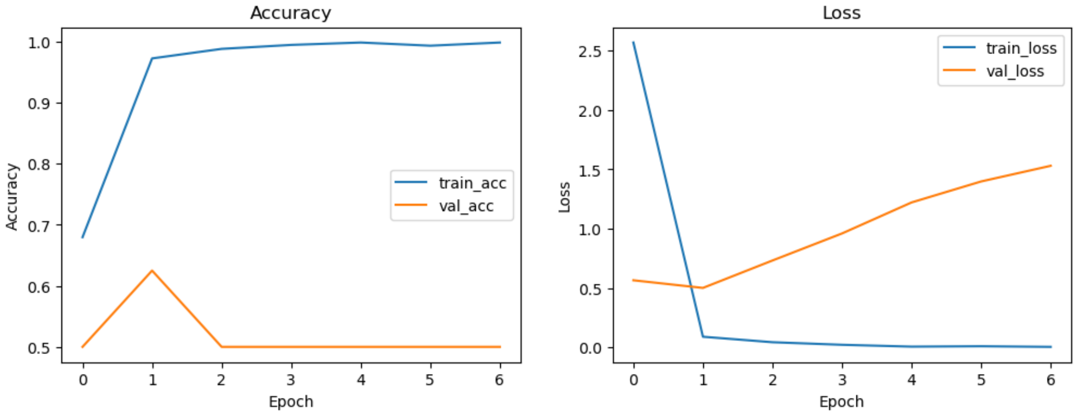
```
Model: "sequential_3"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape            ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d_9 (Conv2D)                │ (None, 32, 32, 32)      │           864 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ batch_normalization_9            │ (None, 32, 32, 32)      │           128 │
│ (BatchNormalization)             │                         │               │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ activation_9 (Activation)        │ (None, 32, 32, 32)      │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ max_pooling2d_9 (MaxPooling2D)   │ (None, 16, 16, 32)      │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ dropout_12 (Dropout)             │ (None, 16, 16, 32)      │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ conv2d_10 (Conv2D)               │ (None, 16, 16, 64)      │        18,432 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ batch_normalization_10           │ (None, 16, 16, 64)      │           256 │
│ (BatchNormalization)             │                         │               │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ activation_10 (Activation)       │ (None, 16, 16, 64)      │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ max_pooling2d_10 (MaxPooling2D)  │ (None, 8, 8, 64)        │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ dropout_13 (Dropout)             │ (None, 8, 8, 64)        │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ conv2d_11 (Conv2D)               │ (None, 8, 8, 128)       │        73,728 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ batch_normalization_11           │ (None, 8, 8, 128)       │           512 │
│ (BatchNormalization)             │                         │               │
│ activation_11 (Activation)           │ (None, 8, 8, 128)           │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_11 (MaxPooling2D)      │ (None, 4, 4, 128)           │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_14 (Dropout)                 │ (None, 4, 4, 128)           │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten_3 (Flatten)                  │ (None, 2048)                │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_6 (Dense)                      │ (None, 512)                 │       1,049,088 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_15 (Dropout)                 │ (None, 512)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_7 (Dense)                      │ (None, 2)                   │           1,026 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 1,144,034 (4.36 MB)
 Trainable params: 1,143,586 (4.36 MB)
 Non-trainable params: 448 (1.75 KB)
```

In [75]:
```python
model.compile(optimizer=Adam(1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

callbacks = [
    EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3)
]
```

In [77]:
```python
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=15,
    batch_size=64,
    callbacks=callbacks
)
```

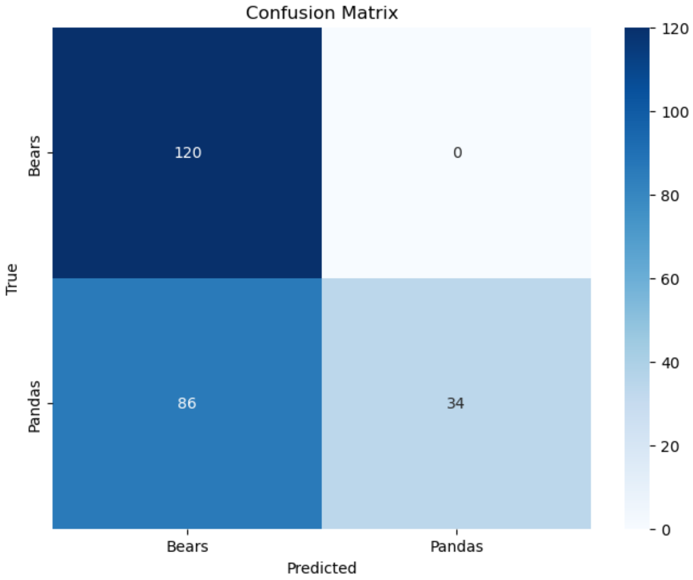
```
Epoch 1/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.5771 - loss: 4.0957 - val_accuracy: 0.5000 - val_loss: 0.5645 - learning_rate: 0.0010
Epoch 2/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.9611 - loss: 0.1289 - val_accuracy: 0.6250 - val_loss: 0.5002 - learning_rate: 0.0010
Epoch 3/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.9821 - loss: 0.0557 - val_accuracy: 0.5000 - val_loss: 0.7315 - learning_rate: 0.0010
Epoch 4/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.9939 - loss: 0.0198 - val_accuracy: 0.5000 - val_loss: 0.9596 - learning_rate: 0.0010
Epoch 5/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.9987 - loss: 0.0050 - val_accuracy: 0.5000 - val_loss: 1.2212 - learning_rate: 0.0010
Epoch 6/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.9899 - loss: 0.0142 - val_accuracy: 0.5000 - val_loss: 1.3981 - learning_rate: 5.0000e-04
Epoch 7/15
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.9962 - loss: 0.0059 - val_accuracy: 0.5000 - val_loss: 1.5302 - learning_rate: 5.0000e-04
```

In [78]:
```python
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='train_acc')
plt.plot(history.history['val_accuracy'], label='val_acc')
plt.legend(); plt.title("Accuracy"); plt.xlabel("Epoch"); plt.ylabel("Accuracy")

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.legend(); plt.title("Loss"); plt.xlabel("Epoch"); plt.ylabel("Loss")
plt.show()
```



In [85]:
```python
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print(f"Test accuracy: {test_acc:.4f}")

y_prob = model.predict(X_test)
y_pred = np.argmax(y_prob, axis=1)

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted"); plt.ylabel("True"); plt.title("Confusion Matrix")
plt.show()

print(classification_report(y_test, y_pred, target_names=class_names, digits=4))
```

```
Test accuracy: 0.6417
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```



```
              precision    recall  f1-score   support

       Bears     0.5825    1.0000    0.7362       120
      Pandas     1.0000    0.2833    0.4416       120

    accuracy                         0.6417       240
   macro avg     0.7913    0.6417    0.5889       240
weighted avg     0.7913    0.6417    0.5889       240
```

Loading [MathJax]/extensions/Safe.js