```
from google.colab import files
```

```
uploaded = files.upload()
```

```
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving archive.zip to archive.zip
```

```
import zipfile
import os
```

```
with zipfile.ZipFile("archive.zip", 'r') as zip_ref:
    zip_ref.extractall("archive")
```

```
os.listdir("archive")
```

```
['name of the animals.txt', 'animals']
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
train_path = "animals_data/animals"
```

Code Text

```
datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
)
```

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    validation_split=0.2  # Split into train and validation sets
)
```

```
os.makedirs("'name of the animals.txt', 'animals'", exist_ok=True)
```

```
os.listdir("'name of the animals.txt', 'animals'")
```

```
[]
```

```
train_path = "'name of the animals.txt', 'animals'"



os.makedirs(" Images('name of the animals.txt', 'animals')", exist_ok=True)


train_generator = datagen.flow_from_directory(
    train_path,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)
```

```
Found 0 images belonging to 0 classes.
```

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import pandas as pd
from PIL import Image
import random
from tensorflow.keras.callbacks import EarlyStopping
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module='keras')


import kagglehub

# Download latest version
test_path = kagglehub.dataset_download("naykrit/animal-classification")
train_path = path = kagglehub.dataset_download("iamsouravbanerjee/animal-image-dataset-90

print("Path to test files:", test_path)
print("Path to train files:", train_path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/naykrit/animal-class
100%|██████████| 7.72M/7.72M [00:01<00:00, 7.40MB/s]Extracting files...

Path to test files: /root/.cache/kagglehub/datasets/naykrit/animal-classification/ver
Path to train files: /kaggle/input/animal-image-dataset-90-different-animals
```

```
def load_images_from_directory(directory, sel, image_size=(150, 150)):
    images = []
    labels = []
    class_names = os.listdir(directory)

    for class_name in sel:
        class_path = os.path.join(directory, class_name)

        if os.path.isdir(class_path):
            # Get all image files in the directory
            for img_name in os.listdir(class_path):
                img_path = os.path.join(class_path, img_name)
                if img_path.lower().endswith(('.png', '.jpg', '.jpeg')):
                    img = image.load_img(img_path, target_size=image_size)
                    img_array = image.img_to_array(img) / 255.0  # Normalize the image
                    images.append(img_array)
                    labels.append(sel.index(class_name))   # Index of class_name as label

    images = np.array(images)
    labels = np.array(labels)
    return images, labels


train_dir = train_path + '/animals/animals'
test_dir = test_path + '/test_set'
class_names = ["dog", "cat", "elephant", "lion", "tiger"]
num_classes = len(class_names)


print("Loading training data...")
train_images, train_labels = load_images_from_directory(train_dir, sel = class_names)
print("Loading test data...")
test_images, test_labels = load_images_from_directory(test_dir, sel = class_names)
train_images, e_images, train_labels, e_labels = train_test_split(train_images, train_lab
```

```
Loading training data...
Loading test data...
```

```
for class_name in class_names:
    class_path = os.path.join(train_dir, class_name)
    if os.path.isdir(class_path):
        num_images = len([f for f in os.listdir(class_path) if f.lower().endswith(('.png'
        print(f"Class {class_name}: {num_images} images")
```

```
Class dog: 60 images
Class cat: 60 images
Class elephant: 60 images
Class lion: 60 images
Class tiger: 60 images
```

```
def show_example_images(images, labels, class_names, target_class, num_images=5):
    class_index = class_names.index(target_class)  # Get the index for the target class
    class_images = [images[i] for i in range(len(images)) if labels[i] == class_index]
```

```
    # Randomly select a number of images from the filtered class
    selected_images = random.sample(class_images, min(num_images, len(class_images)))

    # Plot the selected images
    plt.figure(figsize=(15, 10))
    for i, img in enumerate(selected_images):
        plt.subplot(1, num_images, i+1)

        # Reverse normalization
        img = img * 255.0  # Reverse normalization to show the actual image
        img = img.astype(np.uint8)

        # Plot each image
        plt.imshow(img)
        plt.title(f"Label: {target_class}")
        plt.axis('off')

    plt.show()
```

```
show_example_images(train_images, train_labels, class_names, target_class="dog", num_imag
```


```
show_example_images(train_images, train_labels, class_names, target_class="cat", num_imag
```


```
show_example_images(train_images, train_labels, class_names, target_class="elephant", num
```


```
cnn_model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(150,150,3)),
    layers.MaxPooling2D(2,2),
```

```
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),
    layers.Conv2D(128, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])
cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['acc


cnn_history = cnn_model.fit(
    train_images, train_labels,
    validation_data=(e_images, e_labels),
    epochs=30,
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_b
)
```

```
Epoch 1/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 12s 766ms/step - accuracy: 0.2147 - loss: 1.7576 - val_accur
Epoch 2/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - accuracy: 0.2340 - loss: 1.5910 - val_accurac
Epoch 3/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.3281 - loss: 1.5383 - val_accurac
Epoch 4/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.4097 - loss: 1.3966 - val_accurac
Epoch 5/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.4553 - loss: 1.3631 - val_accurac
Epoch 6/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.4814 - loss: 1.1951 - val_accurac
Epoch 7/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.6108 - loss: 1.0029 - val_accurac
Epoch 8/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.6405 - loss: 0.9672 - val_accurac
Epoch 9/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.6860 - loss: 0.7899 - val_accurac
Epoch 10/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.7379 - loss: 0.7783 - val_accurac
Epoch 11/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.7856 - loss: 0.5935 - val_accurac
Epoch 12/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.7910 - loss: 0.5254 - val_accurac
Epoch 13/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.9091 - loss: 0.3013 - val_accurac
Epoch 14/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.8999 - loss: 0.2379 - val_accurac
Epoch 15/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.8675 - loss: 0.3656 - val_accurac
Epoch 16/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.9520 - loss: 0.1949 - val_accurac
Epoch 17/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.9279 - loss: 0.2060 - val_accurac
```

```
test_loss, test_acc = cnn_model.evaluate(test_images, test_labels)
print(f"Test Accuracy (CNN): {test_acc}")
```

```
test_predictions = np.argmax(cnn_model.predict(test_images), axis=1)
print(classification_report(test_labels, test_predictions, target_names=class_names))
(test_images, test_labels, test_predictions, class_names)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5092 - loss: 1.2680
Test Accuracy (CNN): 0.5699999928474426
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
              precision    recall  f1-score   support

         dog       0.47      0.35      0.40        20
         cat       0.26      0.30      0.28        20
    elephant       0.58      0.90      0.71        20
        lion       0.73      0.40      0.52        20
       tiger       0.90      0.90      0.90        20

    accuracy                           0.57       100
   macro avg       0.59      0.57      0.56       100
weighted avg       0.59      0.57      0.56       100

(array([[[[0.85882354, 0.88235295, 0.84313726],
          [0.88235295, 0.9019608 , 0.8862745 ],
          [0.8901961 , 0.90588236, 0.9019608 ],
          ...,
          [0.5921569 , 0.6       , 0.5137255 ],
          [0.5921569 , 0.6       , 0.5137255 ],
          [0.60784316, 0.6       , 0.5019608 ]],

         [[0.8666667 , 0.8901961 , 0.8509804 ],
          [0.8862745 , 0.90588236, 0.8901961 ],
          [0.90588236, 0.92156863, 0.91764706],
          ...,
          [0.6431373 , 0.6509804 , 0.5686275 ],
          [0.67058825, 0.6784314 , 0.59607846],
          [0.7019608 , 0.69803923, 0.6156863 ]],

         [[0.8745098 , 0.8980392 , 0.85882354],
          [0.89411765, 0.9137255 , 0.8980392 ],
          [0.9137255 , 0.92941177, 0.9254902 ],
          ...,
          [0.78431374, 0.7882353 , 0.7176471 ],
          [0.8039216 , 0.80784315, 0.7372549 ],
          [0.8235294 , 0.827451  , 0.7647059 ]],

         ...,

         [[0.87058824, 0.8352941 , 0.7058824 ],
          [0.8980392 , 0.8627451 , 0.73333335],
          [0.8901961 , 0.85490197, 0.7254902 ],
          ...,
          [0.92156863, 0.8862745 , 0.77254903],
          [0.92941177, 0.89411765, 0.78039217],
          [0.9254902 , 0.8901961 , 0.7764706 ]],

         [[0.8784314 , 0.84313726, 0.7137255 ],
          [0.8784314 , 0.84313726, 0.7137255 ],
          [0.8784314 , 0.84313726, 0.7137255 ],
          ...,
```

```
                [0.9254902 , 0.8901961 , 0.7764706 ],
                [0.9254902 , 0.8901961 , 0.7764706 ],
                [0.91764706, 0.88235295, 0.76862746]],
```

```
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```
cnn_aug_model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(150,150,3)),
    layers.MaxPooling2D(2,2),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),
    layers.Conv2D(128, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])
```

```
cnn_aug_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=[
```

```
cnn_aug_history = cnn_aug_model.fit(
    datagen.flow(train_images, train_labels, batch_size=32),
    validation_data=(e_images, e_labels),
    epochs=30,
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_b
)
```

```
Epoch 1/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 7s 559ms/step - accuracy: 0.1928 - loss: 2.3104 - val_accura
Epoch 2/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 155ms/step - accuracy: 0.1414 - loss: 1.6269 - val_accura
Epoch 3/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 153ms/step - accuracy: 0.2573 - loss: 1.6041 - val_accura
Epoch 4/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 150ms/step - accuracy: 0.2566 - loss: 1.6045 - val_accura
Epoch 5/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 152ms/step - accuracy: 0.2332 - loss: 1.5872 - val_accura
Epoch 6/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 154ms/step - accuracy: 0.2298 - loss: 1.5669 - val_accura
Epoch 7/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 156ms/step - accuracy: 0.3026 - loss: 1.5254 - val_accura
Epoch 8/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 191ms/step - accuracy: 0.3535 - loss: 1.4868 - val_accura
Epoch 9/30
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 151ms/step - accuracy: 0.2752 - loss: 1.5653 - val_accura
Epoch 10/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 150ms/step - accuracy: 0.3402 - loss: 1.5030 - val_accura
Epoch 11/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 156ms/step - accuracy: 0.4058 - loss: 1.4397 - val_accura
Epoch 12/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 153ms/step - accuracy: 0.4681 - loss: 1.3453 - val_accura
Epoch 13/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 152ms/step - accuracy: 0.4484 - loss: 1.2907 - val_accura
Epoch 14/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 156ms/step - accuracy: 0.4146 - loss: 1.3388 - val_accura
Epoch 15/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 150ms/step - accuracy: 0.5311 - loss: 1.2541 - val_accura
Epoch 16/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 161ms/step - accuracy: 0.4032 - loss: 1.3385 - val_accura
Epoch 17/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 151ms/step - accuracy: 0.4623 - loss: 1.2048 - val_accura
Epoch 18/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 154ms/step - accuracy: 0.5314 - loss: 1.1694 - val_accura
Epoch 19/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 152ms/step - accuracy: 0.5255 - loss: 1.1842 - val_accura
Epoch 20/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 149ms/step - accuracy: 0.5056 - loss: 1.1140 - val_accura
Epoch 21/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 152ms/step - accuracy: 0.5478 - loss: 1.1320 - val_accura
Epoch 22/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 149ms/step - accuracy: 0.6005 - loss: 1.0655 - val_accura
Epoch 23/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 151ms/step - accuracy: 0.5916 - loss: 0.9975 - val_accura
Epoch 24/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 147ms/step - accuracy: 0.5546 - loss: 1.0632 - val_accura
Epoch 25/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 248ms/step - accuracy: 0.5438 - loss: 1.0139 - val_accura
Epoch 26/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 147ms/step - accuracy: 0.6408 - loss: 0.8986 - val_accura
Epoch 27/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 156ms/step - accuracy: 0.5844 - loss: 1.0689 - val_accura
```

```
test_loss_aug, test_acc_aug = cnn_aug_model.evaluate(test_images, test_labels)
print(f"Test Accuracy (CNN with Augmentation): {test_acc_aug}")
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step - accuracy: 0.5287 - loss: 1.2679
Test Accuracy (CNN with Augmentation): 0.5899999737739563
```

```
cnn_aug_predictions = np.argmax(cnn_aug_model.predict(test_images), axis=1)
print(classification_report(test_labels, cnn_aug_predictions, target_names=class_names))
```

```
(test_images, test_labels, cnn_aug_predictions, class_names)
(cnn_aug_history)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
              precision    recall  f1-score   support

         dog       0.50      0.45      0.47        20
```

```
                cat       0.31      0.25      0.28        20
           elephant       0.53      0.90      0.67        20
               lion       0.89      0.40      0.55        20
              tiger       0.83      0.95      0.88        20

           accuracy                           0.59       100
          macro avg       0.61      0.59      0.57       100
       weighted avg       0.61      0.59      0.57       100

<keras.src.callbacks.history.History at 0x79aa10558d50>
```

https://colab.research.google.com/drive/1b71lvik2qex4KToGorl6W00rmWV885dN?usp=sharing