In [80]:
```
import tensorflow as tf
import json
import os
import math
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import glob
import numpy as np
```

In [82]:
```
# CREATE DATASET

# Define the folders
folders = {"Blanks": "small_animals/Blanks",
           "American_Toad": "small_animals/American_Toad", "Green_Frog": "sm
           "Northern_Leopard_Frog": "small_animals/Northern_Leopard_Frog", "
           "Indigo_Bunting": "small_animals/Indigo_Bunting", "Northern_House
           "Song_Sparrow": "small_animals/Song_Sparrow", "Sora": "small_anim
           "Common_Five-linked_skink": "small_animals/Common_Five-linked_ski
           "Eastern_Chipmunk": "small_animals/Eastern_Chipmunk", "Eastern_Co
           "Long_tailed_Weasel": "small_animals/Long_tailed_Weasel", "Masked
           "Meadow_Jumping_Mouse": "small_animals/Meadow_Jumping_Mouse", "Me
           "N._Short-tailed_Shrew": "small_animals/N._Short-tailed_Shrew", "
           "Star-nosed_mole": "small_animals/Star-nosed_mole", "Striped_Skur
           "Virginia_Opossum": "small_animals/Virginia_Opossum", "White-foot
           "Woodchuck": "small_animals/Woodchuck", "Woodland_Jumping_Mouse":
           "Butler's_Gartersnake": "small_animals/Butler's_Gartersnake", "De
           "Eastern_Gartersnake": "small_animals/Eastern_Gartersnake", "East
           "Eastern_Massasauga": "small_animals/Eastern_Massasauga", "Easter
           "Eastern_Racer_Snake": "small_animals/Eastern_Racer_Snake", "East
           "Gray_Ratsnake": "small_animals/Gray_Ratsnake", "Kirtland's_Snake
           "Northern_Watersnake": "small_animals/Northern_Watersnake", "Plai
           "Smooth_Greensnake": "small_animals/Smooth_Greensnake", "Turtle":
       }

# Define categories
blanks = {"Blanks"}
invertebrates = {"Invertebrate"}
lizards = {"Common_Five-linked_skink"}
turtles = {"Turtle" }
amphibians = {"American_Toad", "Green_Frog", "Northern_Leopard_Frog" }
birds = {"Common_Yellowthroat", "Indigo_Bunting", "Northern_House_Wren", "So
mammals = {"American_Mink", "Eastern_Chipmunk", "Eastern_Cottontail", "Long_
           "Meadow_Jumping_Mouse", "Meadow_Vole", "N._Short-tailed_Shrew", "
           "Striped_Skunk", "Virginia_Opossum", "White-footed_Mouse", "Woodc
snakes = {"Butler's_Gartersnake", "Dekay's_Brownsnake", "Eastern_Gartersnake
          "Eastern_Massasauga", "Eastern_Milksnake", "Eastern_Racer_Snake",
          "Gray_Ratsnake", "Kirtland's_Snake", "Northern_Watersnake", "Plain
          "Smooth_Greensnake"}

# Dictionary to hold the file paths with their labels
file_paths_with_labels = []
```

```
# Iterate through each folder
for label, folder_path in folders.items():
    # Get all file paths in the folder
    file_paths = glob.glob(os.path.join(folder_path, "*"))

    # Append the file paths with their labels
    file_paths_with_labels.extend([(file_path, label) for file_path in file_
```

In [84]:
```
# Print sample file paths to ensure correctness

print(file_paths_with_labels[1500:1502])
print(file_paths_with_labels[15000:15002])
print(file_paths_with_labels[15000:15002])
print(file_paths_with_labels[50000:50002])
print(file_paths_with_labels[100000:100002])
```

```
[('small_animals/Blanks/CBNP1N_2020-09-14_20-27-50.JPG', 'Blanks'), ('small_
animals/Blanks/CBNP1S_2020-10-22_10-13-15.JPG', 'Blanks')]
[('small_animals/Northern_House_Wren/FCM3__2019-08-29__11-28-44(7).JPG', 'No
rthern_House_Wren'), ('small_animals/Northern_House_Wren/FCM1__2019-08-18__1
2-16-28(2).JPG', 'Northern_House_Wren')]
[('small_animals/Northern_House_Wren/FCM3__2019-08-29__11-28-44(7).JPG', 'No
rthern_House_Wren'), ('small_animals/Northern_House_Wren/FCM1__2019-08-18__1
2-16-28(2).JPG', 'Northern_House_Wren')]
[('small_animals/Masked_Shrew/NOR3__2019-06-01__19-29-18(4).JPG', 'Masked_Sh
rew'), ('small_animals/Masked_Shrew/FCM1__2019-06-14__06-57-41(1).JPG', 'Mas
ked_Shrew')]
[('small_animals/Eastern_Gartersnake/NOR3__2019-08-31__16-07-08(5).JPG', 'Ea
stern_Gartersnake'), ('small_animals/Eastern_Gartersnake/KILC4S__2022-10-03_
_15-23-57(3)__Thamnophis_sirtalis.JPG', 'Eastern_Gartersnake')]
```

In [89]:
```
# CREATE AND RUN MODEL

# Define hyperparameters
IMG_HEIGHT = 128
IMG_WIDTH = 128
BATCH_SIZE = 32
EPOCHS = 10

# Split data into features (file paths) and labels
file_paths, labels = zip(*file_paths_with_labels)

# Split into training and testing data
train_file_paths, test_file_paths, train_labels, test_labels = train_test_sp

print("Training set size:", len(train_file_paths))
print("Test set size:", len(test_file_paths))

# Calculate steps per epoch, rounding up
#steps_per_epoch = math.floor(len(train_file_paths) / BATCH_SIZE)
steps_per_epoch = 2000

# Convert labels to integers using the label map
label_map = {label: idx for idx, label in enumerate(set(labels))}
train_labels = [label_map[label] for label in train_labels]
test_labels = [label_map[label] for label in test_labels]
```

```python
# Create TensorFlow dataset from file paths and labels
def create_tf_dataset(file_paths, labels, batch_size):
    def parse_image(file_path, label):
        try:
            # Read the image from file
            img = tf.io.read_file(file_path)
            # Decode the image
            img = tf.image.decode_jpeg(img, channels=3)
            # Resize the image to target size
            img = tf.image.resize(img, [IMG_HEIGHT, IMG_WIDTH])
            # Normalize pixel values to [0, 1]
            img = img / 255.0
            return img, label
        except tf.errors.InvalidArgumentError:
            # Return None if the image was invalid
            return None

    # Create a TensorFlow dataset
    ds = tf.data.Dataset.from_tensor_slices((file_paths, labels))

    # Remove invalid images using the filter method
    ds = ds.map(parse_image, num_parallel_calls=tf.data.AUTOTUNE)
    ds = ds.filter(lambda img, label: img is not None)  # Filter out None im

    # Shuffle, batch, and prefetch the dataset
    ds = ds.shuffle(buffer_size=1000).batch(batch_size).prefetch(buffer_size

    return ds

# Create training and testing datasets
train_dataset = create_tf_dataset(train_file_paths, train_labels, BATCH_SIZE
test_dataset = create_tf_dataset(test_file_paths, test_labels, BATCH_SIZE)

# Build the CNN model
model = models.Sequential([
    layers.InputLayer(shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    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.Dense(len(label_map), activation='softmax')
])

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

# Train the model
history = model.fit(
    train_dataset,
```

```
    validation_data=test_dataset,
    epochs=EPOCHS,
    steps_per_epoch=steps_per_epoch
)

# Get predictions for the test dataset
predictions = model.predict(test_dataset)
predictions = np.argmax(predictions, axis=1)

# Get the true labels from the test dataset
true_labels = np.array(test_labels)

# Print classification report
print("Classification Report:")
print(classification_report(true_labels, predictions, target_names=list(labe

# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_dataset)
print(f"Test Accuracy: {test_accuracy:.2f}")

# Save the model
tf.keras.models.save_model(model, 'CNN_model.keras')
```

```
Training set size: 92943
Test set size: 23236
Epoch 1/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.6133 - loss: 1.3273
2024-12-07 20:30:30.711204: I tensorflow/core/kernels/data/shuffle_dataset_o
p.cc:450] ShuffleDatasetV3:64: Filling up shuffle buffer (this may take a wh
ile): 566 of 1000
2024-12-07 20:30:31.366877: I tensorflow/core/kernels/data/shuffle_dataset_o
p.cc:480] Shuffle buffer filled.
```

2000/2000 ━━━━━━━━━━━━━━━━━━━━ **2581s** 1s/step - accuracy: 0.6134 - loss: 1.32
71 - val_accuracy: 0.8117 - val_loss: 0.6040
Epoch 2/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ **111s** 55ms/step - accuracy: 0.8235 - loss: 0.5
687 - val_accuracy: 0.8455 - val_loss: 0.4765
Epoch 3/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ **3964s** 2s/step - accuracy: 0.8611 - loss: 0.43
71 - val_accuracy: 0.8857 - val_loss: 0.3708
Epoch 4/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ **112s** 56ms/step - accuracy: 0.9054 - loss: 0.2
887 - val_accuracy: 0.8912 - val_loss: 0.3428
Epoch 5/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ **220s** 109ms/step - accuracy: 0.9225 - loss: 0.
2369 - val_accuracy: 0.9101 - val_loss: 0.2973
Epoch 6/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ **116s** 58ms/step - accuracy: 0.9425 - loss: 0.1
746 - val_accuracy: 0.9099 - val_loss: 0.2917
Epoch 7/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ **691s** 345ms/step - accuracy: 0.9489 - loss: 0.
1506 - val_accuracy: 0.9175 - val_loss: 0.3254
Epoch 8/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ **116s** 58ms/step - accuracy: 0.9575 - loss: 0.1
271 - val_accuracy: 0.9221 - val_loss: 0.2997
Epoch 9/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ **224s** 112ms/step - accuracy: 0.9605 - loss: 0.
1164 - val_accuracy: 0.9285 - val_loss: 0.2662
Epoch 10/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ **119s** 60ms/step - accuracy: 0.9676 - loss: 0.0
959 - val_accuracy: 0.9210 - val_loss: 0.2998
727/727 ━━━━━━━━━━━━━━━━━━━━ **33s** 44ms/step
Classification Report:

	precision	recall	f1-score	support
Green_Frog	0.00	0.00	0.00	15
Raccoon	0.00	0.00	0.00	12
Eastern_Gartersnake	0.28	0.28	0.28	6443
Meadow_Vole	0.11	0.10	0.11	2760
Eastern_Ribbonsnake	0.00	0.00	0.00	26
Sora	0.00	0.00	0.00	2
Plains_Gartersnake	0.01	0.01	0.01	266
Eastern_Hog-nosed_snake	0.00	0.00	0.00	11
Eastern_Milksnake	0.00	0.00	0.00	13
Common_Yellowthroat	0.01	0.01	0.01	151
Eastern_Chipmunk	0.00	0.00	0.00	35
Kirtland's_Snake	0.00	0.00	0.00	9
Indigo_Bunting	0.00	0.00	0.00	7
American_Toad	0.02	0.01	0.01	75
Northern_Leopard_Frog	0.00	0.00	0.00	32
Gray_Ratsnake	0.00	0.00	0.00	7
Butler's_Gartersnake	0.00	0.00	0.00	26
Song_Sparrow	0.13	0.13	0.13	2996
Woodchuck	0.00	0.00	0.00	9
Striped_Skunk	0.00	0.00	0.00	20
Woodland_Jumping_Mouse	0.00	0.00	0.00	292
Virginia_Opossum	0.00	0.00	0.00	194
Turtle	0.00	0.00	0.00	8

Blanks	0.10	0.11	0.10	2354
Invertebrate	0.04	0.04	0.04	978
Masked_Shrew	0.04	0.04	0.04	836
White-footed_Mouse	0.08	0.09	0.08	2043
Common_Five-linked_skink	0.04	0.04	0.04	1018
N._Short-tailed_Shrew	0.01	0.01	0.01	142
Northern_House_Wren	0.05	0.05	0.05	1206
Dekay's_Brownsnake	0.01	0.01	0.01	106
Meadow_Jumping_Mouse	0.00	0.00	0.00	28
Eastern_Cottontail	0.03	0.04	0.03	651
Eastern_Racer_Snake	0.00	0.00	0.00	54
Northern_Watersnake	0.00	0.00	0.00	22
Star-nosed_mole	0.00	0.00	0.00	18
Smooth_Greensnake	0.00	0.00	0.00	49
American_Mink	0.03	0.03	0.03	70
Eastern_Massasauga	0.00	0.00	0.00	252
accuracy			0.13	23236
macro avg	0.03	0.03	0.03	23236
weighted avg	0.13	0.13	0.13	23236

727/727 ━━━━━━━━━━━━━━━━━━━━ **33s** 44ms/step - accuracy: 0.9210 - loss: 0.3021
Test Accuracy: 0.92

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/trainers/epoch_iterator.py:151: UserWarning: Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches. You may need to use the `.repeat()` function when building your dataset.
  self._interrupted_warning()
```

In [ ]: