CNN PROJECT

(CAT , COW, DEER CLASSIFICATION)

[18]:

**from google.colab import** drive drive.mount('/content/drive')

[ ]:

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

*# Paste this entire block into a new Colab cell and run it.*

# import tensorflow as tf

**from tensorflow.keras import** layers, models

# import numpy as np

**from sklearn.metrics import** classification\_report

# import os

*# --- Setup: Path to your dataset folder inside Google Drive ---*

*# This assumes your 'Dataset' folder is in the main 'My Drive' directory.*

*# If it's inside another folder, adjust the path e.g., '/content/drive/My Drive/*

𝗌*Colab\_Data/Dataset'*

dataset\_path = '/content/drive/My Drive/Dataset'

*# --- Sanity Check: Verify the path exists ---*

**if not** os.path.exists(dataset\_path):

print(f"Error: The directory '**{**dataset\_path**}**' does not exist.") print("Please make sure you have uploaded the 'Dataset' folder to your␣

𝗌Google Drive's main directory.")

# else:

*# --- Parameters ---*

img\_size = (128, 128)

batch\_size = 32

*# --- 1. Load Datasets with an 80/20 split ---*

*# validation\_split=0.2 uses 80% for training and 20% for validation.*

print("--- Loading Datasets (80% Train, 20% Validation) ---")

train\_ds = tf.keras.utils.image\_dataset\_from\_directory( dataset\_path,

validation\_split=0.2, subset="training", seed=123, image\_size=img\_size, batch\_size=batch\_size

)

val\_ds = tf.keras.utils.image\_dataset\_from\_directory( dataset\_path,

validation\_split=0.2, subset="validation", seed=123, image\_size=img\_size, batch\_size=batch\_size

)

*# Get class names*

class\_names = train\_ds.class\_names num\_classes = len(class\_names) print(f"Found classes: **{**class\_names**}**")

*# --- 2. Optimize dataset performance ---*

AUTOTUNE = tf.data.AUTOTUNE

train\_ds = train\_ds.cache().prefetch(buffer\_size=AUTOTUNE) val\_ds = val\_ds.cache().prefetch(buffer\_size=AUTOTUNE)

*# --- 3. Define the CNN model (same as before) ---*

model = models.Sequential([

layers.Rescaling(1./255, input\_shape=img\_size + (3,)), layers.Conv2D(16, (3, 3), padding='same', activation='relu'), layers.MaxPooling2D(),

layers.Conv2D(32, (3, 3), padding='same', activation='relu'), layers.MaxPooling2D(),

layers.Conv2D(64, (3, 3), padding='same', activation='relu'), layers.MaxPooling2D(),

layers.Flatten(),

layers.Dense(128, activation='relu'), layers.Dense(num\_classes, activation='softmax')

])

*# --- 4. Compile the model ---*

model.compile( optimizer='adam',

loss='sparse\_categorical\_crossentropy', metrics=['accuracy']

)

*# --- 5. Train the model ---*

epochs = 10

print("**\n**--- Starting Model Training ---") history = model.fit(

train\_ds,

validation\_data=val\_ds, *# The model is tested on the 20% validation*␣

𝗌*data each epoch*

epochs=epochs

)

print("--- Model Training Finished ---")

*# --- 6. Save the final model to Google Drive for permanent storage ---* model\_save\_path = '/content/drive/My Drive/animal\_cnn\_model.h5' print(f"**\n**--- Saving Model to: **{**model\_save\_path**}** ---") model.save(model\_save\_path)

print("Model saved successfully to Google Drive.")

*# --- 7. Evaluate on the 20% validation data and show Classification Report*␣

𝗌*---*

print("**\n**--- Final Evaluation on 20% Validation Data ---") y\_pred = []

y\_true = []

**for** image\_batch, label\_batch **in** val\_ds: y\_true.extend(label\_batch.numpy())

preds = model.predict(image\_batch, verbose=0) y\_pred.extend(np.argmax(preds, axis=1))

print("**\n**--- Classification Report ---") print(classification\_report(y\_true, y\_pred, target\_names=class\_names))

--- Loading Datasets (80% Train, 20% Validation) --- Found 381 files belonging to 3 classes.

Using 305 files for training.

Found 381 files belonging to 3 classes. Using 76 files for validation.

Found classes: ['Cat', 'Cow', 'Deer']

/usr/local/lib/python3.11/dist- packages/keras/src/layers/preprocessing/tf\_data\_layer.py:19: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

super(). init (\*\*kwargs)

--- Starting Model Training ---

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Epoch 1/10 |  |  |  | | | |
| **10/10** |  | **74s** 7s/step - |
| accuracy: | 0.2846 | - loss: 1.2399 | - val\_accuracy: | 0.2895 | - val\_loss: | 1.0865 |
| Epoch 2/10 |  |  |  |  |  |  |
| **10/10** |  | **0s** 10ms/step - |  |  |  |  |
| accuracy: | 0.3521 | - loss: 1.0967 | - val\_accuracy: | 0.3026 | - val\_loss: | 1.0626 |
| Epoch 3/10 |  |  |  |  |  |  |
| **10/10** |  | **0s** 10ms/step - |  |  |  |  |
| accuracy: | 0.3817 | - loss: 1.0423 | - val\_accuracy: | 0.5263 | - val\_loss: | 0.9647 |
| Epoch 4/10 |  |  |  |  |  |  |
| **10/10** |  | **0s** 10ms/step - |  |  |  |  |
| accuracy: | 0.5848 | - loss: 0.9418 | - val\_accuracy: | 0.6579 | - val\_loss: | 0.8736 |
| Epoch 5/10 |  |  |  |  |  |  |
| **10/10** |  | **0s** 10ms/step - |  |  |  |  |
| accuracy: | 0.6857 | - loss: 0.7990 | - val\_accuracy: | 0.7237 | - val\_loss: | 0.7577 |
| Epoch 6/10 |  |  |  |  |  |  |
| **10/10** |  | **0s** 10ms/step - |  |  |  |  |
| accuracy: | 0.8095 | - loss: 0.5884 | - val\_accuracy: | 0.7632 | - val\_loss: | 0.6277 |
| Epoch 7/10 |  |  |  |  |  |  |
| **10/10** |  | **0s** 10ms/step - |  |  |  |  |
| accuracy: | 0.8622 | - loss: 0.4428 | - val\_accuracy: | 0.7763 | - val\_loss: | 0.6328 |
| Epoch 8/10 |  |  |  |  |  |  |
| **10/10** |  | **0s** 11ms/step - |  |  |  |  |
| accuracy: | 0.8818 | - loss: 0.3429 | - val\_accuracy: | 0.8158 | - val\_loss: | 0.5436 |
| Epoch 9/10 |  |  |  |  |  |  |
| **10/10** |  | **0s** 10ms/step - |  |  |  |  |
| accuracy: | 0.9112 | - loss: 0.2873 | - val\_accuracy: | 0.7368 | - val\_loss: | 0.7605 |

Epoch 10/10

**10/10 0s** 11ms/step -

accuracy: 0.8911 - loss: 0.3301 - val\_accuracy: 0.7895 - val\_loss: 0.5826

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or

`keras.saving.save\_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g.

`model.save('my\_model.keras')` or `keras.saving.save\_model(model, 'my\_model.keras')`.

--- Model Training Finished ---

--- Saving Model to: /content/drive/My Drive/animal\_cnn\_model.h5 --- Model saved successfully to Google Drive.

--- Final Evaluation on 20% Validation Data ---

--- Classification Report ---

precision recall f1-score support

Cat 0.88 0.79 0.83 28

Cow 0.66 0.86 0.75 22

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Deer | 0.86 | 0.73 | 0.79 | 26 |
| accuracy |  |  | 0.79 | 76 |
| macro avg | 0.80 | 0.79 | 0.79 | 76 |
| weighted avg | 0.81 | 0.79 | 0.79 | 76 |

import tensorflow as tf import numpy as np

**from tensorflow.keras.utils import** load\_img, img\_to\_array

# import os

*# --- 1. Setup: Paths point to files in your Google Drive ---*

*# Path to your saved model in Google Drive*

MODEL\_PATH = '/content/drive/My Drive/animal\_cnn\_model.h5'

*# IMPORTANT: Change this to the path of the test image you uploaded to Drive*

IMAGE\_PATH = '/content/drive/My Drive/my\_test\_cat1.jpg'

*# Class names must match the training order*

CLASS\_NAMES = ['Cat', 'Cow', 'Deer']

*# Image size must match training*

IMG\_SIZE = (128, 128)

*# --- 2. Load the model and image, then predict ---*

**if not** os.path.exists(IMAGE\_PATH):

print(f"Error: Test image not found at '**{**IMAGE\_PATH**}**'")

print("Please upload your test image to Google Drive and check the path.")

# else:

print("--- Loading Model and Making Prediction ---") model = tf.keras.models.load\_model(MODEL\_PATH)

img = load\_img(IMAGE\_PATH, target\_size=IMG\_SIZE) img\_array = img\_to\_array(img)

img\_batch = np.expand\_dims(img\_array, axis=0)

predictions = model.predict(img\_batch)

predicted\_class\_index = np.argmax(predictions[0]) predicted\_class\_name = CLASS\_NAMES[predicted\_class\_index] confidence = np.max(predictions[0]) \* 100

print(f"**\n**--- Prediction Result ---")

print(f"The image is most likely a: **{**predicted\_class\_name.upper()**}**") print(f"Confidence: **{**confidence**:**.2f**}**%")

[ ]:

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile\_metrics` will be empty until you train or evaluate the model.

--- Loading Model and Making Prediction ---

**1/1 0s** 269ms/step

--- Prediction Result ---

The image is most likely a: CAT Confidence: 98.39%

# import matplotlib.pyplot as plt

*# --- 6. Visualize training results ---*

print("**\n**--- Generating Training History Graphs ---")

*# Get the accuracy and loss values from the history object*

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy'] loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs\_range = range(epochs)

*# Create a figure with two subplots*

plt.figure(figsize=(14, 6))

*# Subplot 1: Training and Validation Accuracy*

plt.subplot(1, 2, 1)

plt.plot(epochs\_range, acc, label='Training Accuracy') plt.plot(epochs\_range, val\_acc, label='Validation Accuracy') plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy') plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.grid(**True**) *# Add grid for better readability*

*# Subplot 2: Training and Validation Loss*

plt.subplot(1, 2, 2)

plt.plot(epochs\_range, loss, label='Training Loss')

plt.plot(epochs\_range, val\_loss, label='Validation Loss') plt.legend(loc='upper right')

plt.title('Training and Validation Loss') plt.xlabel('Epoch')

plt.ylabel('Loss')

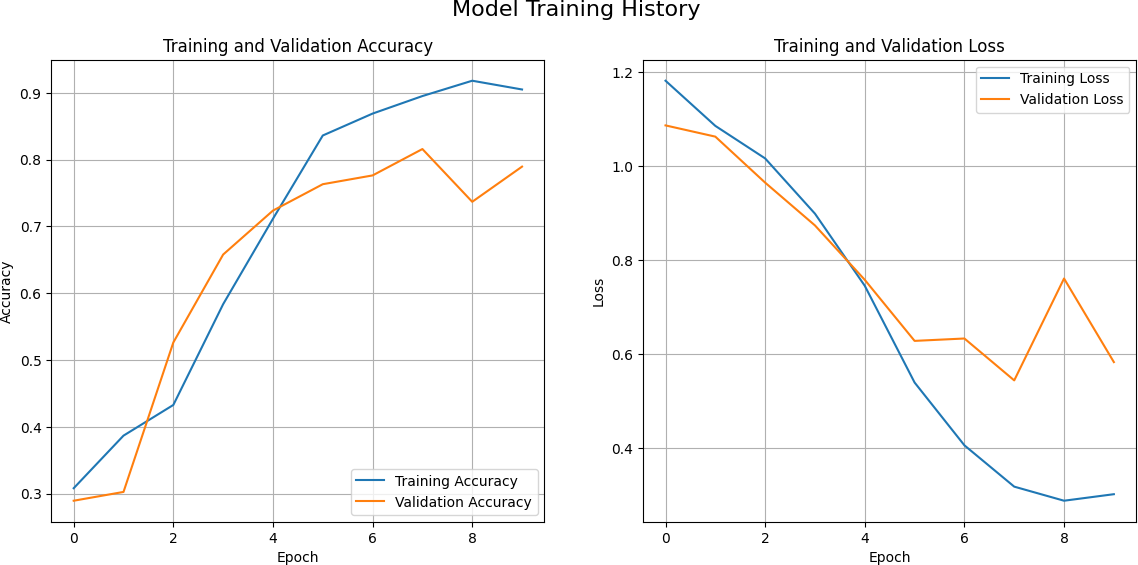
plt.grid(**True**) *# Add grid for better readability*

*# Display the plots*

plt.suptitle('Model Training History', fontsize=16) plt.show()

print("--- Graphs Generated Successfully ---")

--- Generating Training History Graphs ---



--- Graphs Generated Successfully ---

[ ]:

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**from sklearn.metrics import** confusion\_matrix

*# --- 8. Generate and Display Confusion Matrix ---*

print("**\n**--- Generating Confusion Matrix ---")

*# Compute the confusion matrix*

cm = confusion\_matrix(y\_true, y\_pred)

*# Create a figure for the plot*

plt.figure(figsize=(8, 6))

*# Create a heatmap using Seaborn*

*# annot=True: displays the numbers in each cell # fmt='d': formats the numbers as integers*

*# cmap='Blues': sets the color scheme*

sns.heatmap(cm, annot=**True**, fmt='d', cmap='Blues', xticklabels=class\_names, yticklabels=class\_names)

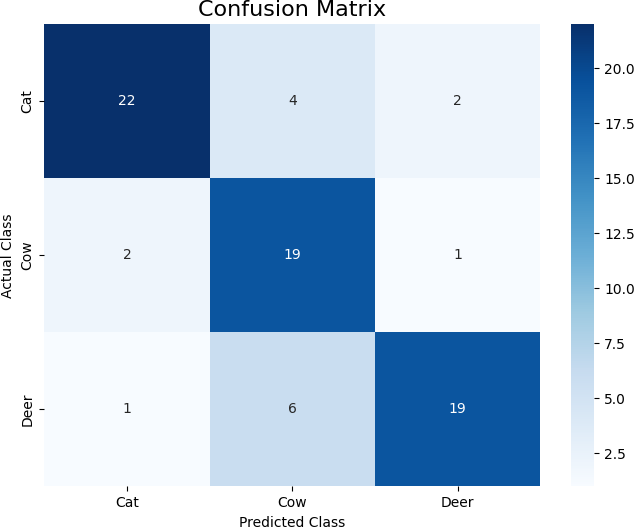
*# Add titles and labels for clarity* plt.title('Confusion Matrix', fontsize=16) plt.ylabel('Actual Class') plt.xlabel('Predicted Class')

*# Display the plot*

plt.show()

print("--- Confusion Matrix Generated Successfully ---")

--- Generating Confusion Matrix ---



CNN PROJECT

(LION , PANDA , TIGER CLASSIFICATION)

[12]:

*# Paste this entire block into a new Colab cell and run it.*

**import tensorflow as tf**

**from tensorflow.keras import** layers, models

**import numpy as np**

**from sklearn.metrics import** classification\_report

**import os**

*# --- Setup: Path to your dataset folder inside Google Drive ---*

*# This assumes your 'Dataset' folder is in the main 'My Drive' directory.*

*# If it's inside another folder, adjust the path e.g., '/content/drive/My Drive/*

𝗌*Colab\_Data/Dataset'*

dataset\_path = '/content/drive/My Drive/CNN/Dataset4'

*# --- Sanity Check: Verify the path exists ---*

**if not** os.path.exists(dataset\_path):

print(f"Error: The directory '**{**dataset\_path**}**' does not exist.") print("Please make sure you have uploaded the 'Dataset' folder to your␣

𝗌Google Drive's main directory.")

# else:

*# --- Parameters ---*

img\_size = (128, 128)

batch\_size = 32

*# --- 1. Load Datasets with an 80/20 split ---*

*# validation\_split=0.2 uses 80% for training and 20% for validation.* print("--- Loading Datasets (80% Train, 20% Validation) ---") train\_ds = tf.keras.utils.image\_dataset\_from\_directory(

dataset\_path, validation\_split=0.2, subset="training", seed=123, image\_size=img\_size, batch\_size=batch\_size

)

val\_ds = tf.keras.utils.image\_dataset\_from\_directory( dataset\_path,

validation\_split=0.2, subset="validation", seed=123, image\_size=img\_size, batch\_size=batch\_size

)

*# Get class names*

class\_names = train\_ds.class\_names num\_classes = len(class\_names) print(f"Found classes: **{**class\_names**}**")

*# --- 2. Optimize dataset performance ---*

AUTOTUNE = tf.data.AUTOTUNE

train\_ds = train\_ds.cache().prefetch(buffer\_size=AUTOTUNE) val\_ds = val\_ds.cache().prefetch(buffer\_size=AUTOTUNE)

*# --- 3. Define the CNN model (same as before) ---*

model = models.Sequential([

layers.Rescaling(1./255, input\_shape=img\_size + (3,)), layers.Conv2D(16, (3, 3), padding='same', activation='relu'), layers.MaxPooling2D(),

layers.Conv2D(32, (3, 3), padding='same', activation='relu'), layers.MaxPooling2D(),

layers.Conv2D(64, (3, 3), padding='same', activation='relu'),

layers.MaxPooling2D(), layers.Flatten(),

layers.Dense(128, activation='relu'), layers.Dense(num\_classes, activation='softmax')

])

*# --- 4. Compile the model ---*

model.compile( optimizer='adam',

loss='sparse\_categorical\_crossentropy', metrics=['accuracy']

)

*# --- 5. Train the model ---*

epochs = 10

print("**\n**--- Starting Model Training ---") history = model.fit(

train\_ds,

validation\_data=val\_ds, *# The model is tested on the 20% validation*␣

𝗌*data each epoch*

epochs=epochs

)

print("--- Model Training Finished ---")

*# --- 6. Save the final model to Google Drive for permanent storage ---* model\_save\_path = '/content/drive/My Drive/animal\_cnn\_model2.h5' print(f"**\n**--- Saving Model to: **{**model\_save\_path**}** ---") model.save(model\_save\_path)

print("Model saved successfully to Google Drive.")

*# --- 7. Evaluate on the 20% validation data and show Classification Report*␣

𝗌*---*

print("**\n**--- Final Evaluation on 20% Validation Data ---") y\_pred = []

y\_true = []

**for** image\_batch, label\_batch **in** val\_ds: y\_true.extend(label\_batch.numpy())

preds = model.predict(image\_batch, verbose=0) y\_pred.extend(np.argmax(preds, axis=1))

print("**\n**--- Classification Report ---") print(classification\_report(y\_true, y\_pred, target\_names=class\_names))

--- Loading Datasets (80% Train, 20% Validation) --- Found 405 files belonging to 3 classes.

Using 324 files for training.

Found 405 files belonging to 3 classes.

Using 81 files for validation.

Found classes: ['Lion', 'Panda', 'Tiger']

--- Starting Model Training --- Epoch 1/10

/usr/local/lib/python3.11/dist- packages/keras/src/layers/preprocessing/tf\_data\_layer.py:19: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

super(). init (\*\*kwargs)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11/11** |  | **7s** 341ms/step - |  | | |
| accuracy: | 0.4220 | - loss: 1.1620 - val\_accuracy: | 0.5185 | - val\_loss: | 0.9724 |
| Epoch 2/10 |  |  |  |  |  |
| **11/11** |  | **0s** 15ms/step - |  |  |  |
| accuracy: | 0.8255 | - loss: 0.7835 - val\_accuracy: | 0.7160 | - val\_loss: | 0.7595 |
| Epoch 3/10 |  |  |  |  |  |
| **11/11** |  | **0s** 13ms/step - |  |  |  |
| accuracy: | 0.8749 | - loss: 0.4727 - val\_accuracy: | 0.7901 | - val\_loss: | 0.5224 |
| Epoch 4/10 |  |  |  |  |  |
| **11/11** |  | **0s** 11ms/step - |  |  |  |
| accuracy: | 0.8947 | - loss: 0.2934 - val\_accuracy: | 0.7654 | - val\_loss: | 0.6115 |
| Epoch 5/10 |  |  |  |  |  |
| **11/11** |  | **0s** 11ms/step - |  |  |  |
| accuracy: | 0.9471 | - loss: 0.1938 - val\_accuracy: | 0.7778 | - val\_loss: | 0.5767 |
| Epoch 6/10 |  |  |  |  |  |
| **11/11** |  | **0s** 10ms/step - |  |  |  |
| accuracy: | 0.9622 | - loss: 0.1342 - val\_accuracy: | 0.7901 | - val\_loss: | 0.7037 |
| Epoch 7/10 |  |  |  |  |  |
| **11/11** |  | **0s** 11ms/step - |  |  |  |
| accuracy: | 0.9688 | - loss: 0.1093 - val\_accuracy: | 0.8272 | - val\_loss: | 0.4769 |
| Epoch 8/10 |  |  |  |  |  |
| **11/11** |  | **0s** 10ms/step - |  |  |  |
| accuracy: | 0.9610 | - loss: 0.1105 - val\_accuracy: | 0.8148 | - val\_loss: | 0.7453 |
| Epoch 9/10 |  |  |  |  |  |
| **11/11** |  | **0s** 10ms/step - |  |  |  |
| accuracy: | 0.9303 | - loss: 0.1630 - val\_accuracy: | 0.8148 | - val\_loss: | 0.4855 |

Epoch 10/10

**11/11 0s** 10ms/step -

accuracy: 0.9951 - loss: 0.0730 - val\_accuracy: 0.8889 - val\_loss: 0.3331

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or

`keras.saving.save\_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g.

`model.save('my\_model.keras')` or `keras.saving.save\_model(model, 'my\_model.keras')`.

--- Model Training Finished ---

--- Saving Model to: /content/drive/My Drive/animal\_cnn\_model2.h5 --- Model saved successfully to Google Drive.

**import matplotlib.pyplot as plt**

*# --- 6. Visualize training results ---*

print("**\n**--- Generating Training History Graphs ---")

*# Get the accuracy and loss values from the history object*

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy'] loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs\_range = range(epochs)

*# Create a figure with two subplots*

plt.figure(figsize=(14, 6))

*# Subplot 1: Training and Validation Accuracy*

plt.subplot(1, 2, 1)

--- Final Evaluation on 20% Validation Data --- WARNING:tensorflow:5 out of the last 7 calls to <function

TensorFlowTrainer.make\_predict\_function.<locals>.one\_step\_on\_data\_distributed at 0x7fad0ff5bb00> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce\_retracing=True option that can avoid unnecessary retracing. For (3), please refer to [https://www.tensorflow.org/guide/function#controlling\_retracing](http://www.tensorflow.org/guide/function#controlling_retracing) and [https://www.tensorflow.org/api\_docs/python/tf/function](http://www.tensorflow.org/api_docs/python/tf/function) for more details.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | --- Classification Report  precision | ---  recall | f1-score | support |
| Lion 0.92 | 0.80 | 0.86 | 30 |
| Panda 0.86 | 0.95 | 0.90 | 20 |
| Tiger 0.88 | 0.94 | 0.91 | 31 |
| accuracy |  | 0.89 | 81 |
| macro avg 0.89 | 0.90 | 0.89 | 81 |
| weighted avg 0.89 | 0.89 | 0.89 | 81 |
| [13]: |  |  |  |  |

plt.plot(epochs\_range, acc, label='Training Accuracy') plt.plot(epochs\_range, val\_acc, label='Validation Accuracy') plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy') plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.grid(**True**) *# Add grid for better readability*

*# Subplot 2: Training and Validation Loss*

plt.subplot(1, 2, 2)

plt.plot(epochs\_range, loss, label='Training Loss') plt.plot(epochs\_range, val\_loss, label='Validation Loss') plt.legend(loc='upper right')

plt.title('Training and Validation Loss') plt.xlabel('Epoch')

plt.ylabel('Loss')

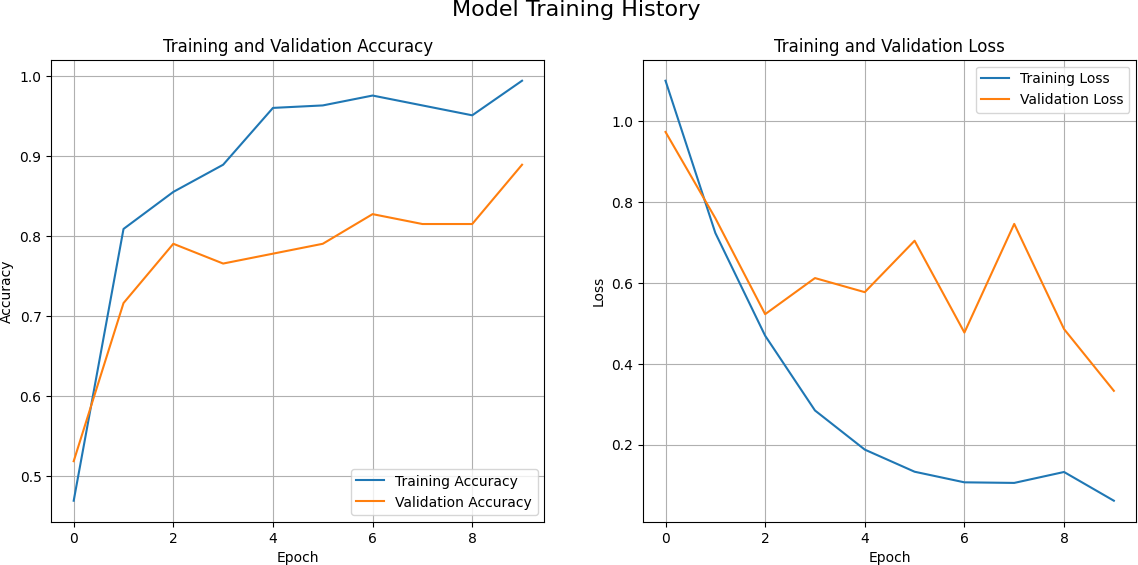
plt.grid(**True**) *# Add grid for better readability*

*# Display the plots*

plt.suptitle('Model Training History', fontsize=16) plt.show()

print("--- Graphs Generated Successfully ---")

--- Generating Training History Graphs ---



--- Graphs Generated Successfully ---

[14]:

# import seaborn as sns

**import matplotlib.pyplot as plt**

**from sklearn.metrics import** confusion\_matrix

*# --- 8. Generate and Display Confusion Matrix ---*

print("**\n**--- Generating Confusion Matrix ---")

*# Compute the confusion matrix*

cm = confusion\_matrix(y\_true, y\_pred)

*# Create a figure for the plot*

plt.figure(figsize=(8, 6))

*# Create a heatmap using Seaborn*

*# annot=True: displays the numbers in each cell # fmt='d': formats the numbers as integers*

*# cmap='Blues': sets the color scheme*

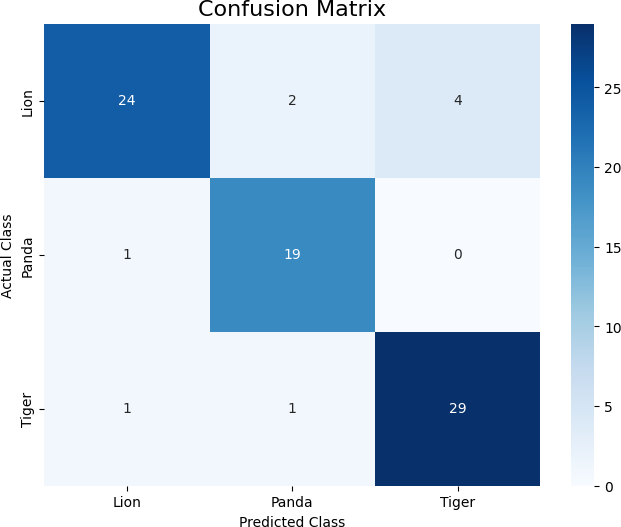
sns.heatmap(cm, annot=**True**, fmt='d', cmap='Blues', xticklabels=class\_names, yticklabels=class\_names)

*# Add titles and labels for clarity* plt.title('Confusion Matrix', fontsize=16) plt.ylabel('Actual Class') plt.xlabel('Predicted Class')

*# Display the plot*

plt.show()

print("--- Confusion Matrix Generated Successfully ---")

-

--- Confusion Matrix Generated Successfully ---

[19]:

*# Paste this entire block into a new Colab cell and run it.*

**import tensorflow as tf import numpy as np**

**from tensorflow.keras.utils import** load\_img, img\_to\_array

**import os**

*# --- 1. Setup: Paths point to files in your Google Drive ---*

*# Path to your saved model in Google Drive*

MODEL\_PATH = '/content/drive/My Drive/animal\_cnn\_model2.h5'

*# IMPORTANT: Change this to the path of the test image you uploaded to Drive*

IMAGE\_PATH = '/content/drive/My Drive/Tiger.jpg'

[ ]:

*# Class names must match the training order*

CLASS\_NAMES = ['Lion', 'Panda', 'Tiger']

*# Image size must match training*

IMG\_SIZE = (128, 128)

*# --- 2. Load the model and image, then predict ---*

**if not** os.path.exists(IMAGE\_PATH):

print(f"Error: Test image not found at '**{**IMAGE\_PATH**}**'")

print("Please upload your test image to Google Drive and check the path.")

# else:

print("--- Loading Model and Making Prediction ---") model = tf.keras.models.load\_model(MODEL\_PATH)

img = load\_img(IMAGE\_PATH, target\_size=IMG\_SIZE) img\_array = img\_to\_array(img)

img\_batch = np.expand\_dims(img\_array, axis=0)

predictions = model.predict(img\_batch) predicted\_class\_index = np.argmax(predictions[0]) predicted\_class\_name = CLASS\_NAMES[predicted\_class\_index] confidence = np.max(predictions[0]) \* 100

print(f"**\n**--- Prediction Result ---")

print(f"The image is most likely a: **{**predicted\_class\_name.upper()**}**") print(f"Confidence: **{**confidence**:**.2f**}**%")

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile\_metrics` will be empty until you train or evaluate the model.

--- Loading Model and Making Prediction ---

**1/1 0s** 313ms/step

--- Prediction Result ---

The image is most likely a: TIGER Confidence: 99.85%