Computer Vision Midterm Assignment

Introduction

Welcome to your Computer Vision midterm project! Here, you'll get hands-on experience building an image recognition model using Convolutional Neural Networks and transfer learning.

Install Necessary Libraries:

```
!pip install tensorflow
!pip install keras
!pip install numpy
!pip install matplotlib
```

```
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.13.0)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.37.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (0.45.1)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.14.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (2025.1.31)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow) (3.7)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tenso
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard<2.19,>=2.18->tensorfl
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow) (0
Requirement already satisfied: keras in /usr/local/lib/python3.11/dist-packages (3.8.0)
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-packages (from keras) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from keras) (2.0.2)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras) (0.0.8)
Requirement already satisfied: h5py in /usr/local/lib/python3.11/dist-packages (from keras) (3.13.0)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras) (0.14.1)
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.11/dist-packages (from keras) (0.4.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from keras) (24.2)
Requirement already satisfied: typing-extensions>=4.5.0 in /usr/local/lib/python3.11/dist-packages (from optree->keras) (4.12.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras) (0.1.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
```

eError ValueError TypeError AxisError

Please explain the error:

```
AxisError: axis 1 is out of bounds for array of dimension 1
```

Hypothesis

The error message `AxisError: axis 1 is out of bounds for array of dimension 1` indicates that you are trying to apply `np.argmax` along axis 1 of an array that only has one dimension.

- **Cause:** The variable `y_test` likely has only one dimension. This means that `y_test` is a 1D array and does not have columns to iterate over axis 1. The line `y_test = np.argmax(y_test, axis=1)` attempts to find the maximum value along axis 1, which is incorrect as only axis 0 is valid for a 1D array.
- **Further Investigation:**
  - `y_test`'s shape, as indicated by the global variables, is `(32,)`. This confirms that it is a 1D array.
  - The code assumes that `y_test` is in one-hot encoded format, but it's actually in the form of class labels (integers).
  - In this case, the argmax is not necessary for `y_test`. Instead, it is required for `y_pred` to get predicted classes.

Suggested Changes

```
from sklearn.metrics import confusion_matrix
import seaborn as sn

best_model = load_model('best_model.h5')

y_pred = best_model.predict(x_test)
# y_test = np.argmax(y_test, axis=1)  # Remove or comment out tl
y_pred_classes = np.argmax(y_pred, axis=1)  # Get predicted cla:

cm = confusion_matrix(y_test, y_pred_classes)  # y_test is alre:

plt.figure(figsize=(8, 6))
sn.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

Use code with caution

▶ Sources

Rate this answer

```
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.8.2)
```

```python
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout
from tensorflow.keras.applications import VGG16, ResNet50, MobileNetV2  # Choose a pre-trained model
from tensorflow.keras.callbacks import ModelCheckpoint

# Additional libraries for data loading (if using a custom dataset)
# from skimage.io import imread  # Example for loading images
import os
from PIL import Image
from collections import Counter
```

Dataset Selection and Loading

- **Choose Your Dataset**
  - **Standard Datasets:** CIFAR-10, CIFAR-100, or a suitable subset of ImageNet are good starting points. You can use built-in functions to load them.
  - **Custom Dataset:** If you propose a custom dataset, ensure it has sufficient images per class, good quality, and accurate labeling. You'll need to upload it to Colab.
  - **Select your dataset and uncomment the appropriate loading code.**
  - **If you are using a custom dataset, make sure you have uploaded it to Colab and adjust the file path.**

```python
#Oxford-IIIT Pet Dataset

from google.colab import drive
drive.mount('/content/dataset-iiit-pet')

!git clone https://github.com/ml4py/dataset-iiit-pet.git

!ls -lh dataset-iiit-pet
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-135-10a58bb168f4> in <cell line: 0>()
      2
      3 from google.colab import drive
----> 4 drive.mount('/content/dataset-iiit-pet')
      5
      6 get_ipython().system('git clone https://github.com/ml4py/dataset-iiit-pet.git')

1 frames

/usr/local/lib/python3.11/dist-packages/google/colab/drive.py in _mount(mountpoint, force_remount, timeout_ms, ephemeral, readonly)
    197       raise ValueError('Mountpoint must not be a symlink')
    198     if _os.path.isdir(mountpoint) and _os.listdir(mountpoint):
--> 199       raise ValueError('Mountpoint must not already contain files')
    200     if not _os.path.isdir(mountpoint) and _os.path.exists(mountpoint):
    201       raise ValueError('Mountpoint must either be a directory or not exist')

ValueError: Mountpoint must not already contain files
```

Next steps: Explain error

```python
pet_data = "/content/dataset-iiit-pet/images"
```

```python
image_filenames = os.listdir(pet_data)

print("Image names:")
for image_name in image_filenames:
    print(image_name)
```

```
Abyssinian_2.jpg
Bombay_176.jpg
Siamese_48.jpg
havanese_121.jpg
Bombay_107.jpg
shiba_inu_181.jpg
english_setter_158.jpg
american_bulldog_56.jpg
boxer_122.jpg
Ragdoll_10.jpg
saint_bernard_102.jpg
beagle_73.jpg
great_pyrenees_61.jpg
newfoundland_25.jpg
boxer_129.jpg
english_cocker_spaniel_19.jpg
Maine_Coon_169.jpg
basset_hound_145.jpg
staffordshire_bull_terrier_185.jpg
english_setter_106.jpg
american_bulldog_130.jpg
yorkshire_terrier_142.jpg
Ragdoll_219.jpg
yorkshire_terrier_19.jpg
shiba_inu_182.jpg
havanese_108.jpg
Birman_83.jpg
havanese_200.jpg
american_pit_bull_terrier_17.jpg
Ragdoll_52.jpg
Bengal_166.jpg
havanese_12.jpg
staffordshire_bull_terrier_31.jpg
Birman_82.jpg
miniature_pinscher_139.jpg
Persian_141.jpg
newfoundland_154.jpg
Ragdoll_78.jpg
leonberger_162.jpg
wheaten_terrier_31.jpg
Abyssinian_122.jpg
yorkshire_terrier_163.jpg
american_bulldog_115.jpg
Bengal_11.jpg
Bengal_49.jpg
British_Shorthair_34.jpg
yorkshire_terrier_95.jpg
yorkshire_terrier_17.jpg
Persian_111.jpg
Bombay_183.jpg
english_cocker_spaniel_113.jpg
Persian_114.jpg
Maine_Coon_171.jpg
chihuahua_157.jpg
american_bulldog_109.jpg
Abyssinian_80.jpg
keeshond_153.jpg
american_bulldog_16.jpg
beagle_26.jpg
```

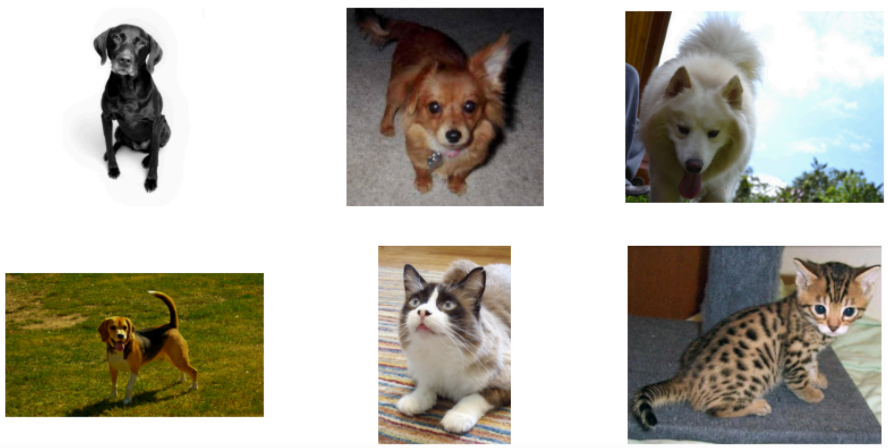
```python
image_total = len(image_filenames)
print(f"\nTotal images: {len(image_filenames)}")
```

```
Total images: 7393
```

Markdown Cell: Exploratory Data Analysis (EDA)

- **Instructions:**
  - Visualize a few random images from your dataset to understand its content and overall quality.
  - Check the shape of your data to confirm the number of images and their dimensions.

```
# Insert code here to display a few sample images from the dataset
## Display sample images
plt.figure(figsize=(10, 5))
for i in range(6):
    image_path = os.path.join(pet_data, image_filenames[i])
    image = Image.open(image_path)

    plt.subplot(2, 3, i + 1)
    plt.imshow(image)
    plt.axis('off')
plt.show()
```



```
# Explore class distribution (if using a standard dataset)

# Extract class labels from filenames
class_labels = [filename.split('_')[0] for filename in image_filenames]
print(class_labels)
```

```
['german', 'pomeranian', 'samoyed', 'beagle', 'Ragdoll', 'Bengal', 'havanese', 'Sphynx', 'Abyssinian', 'British', 'British', 'shiba', 'Ragdoll', 'wheat
```

```
# Count occurrences of each class
class_distribution = Counter(class_labels)

# Print the class distribution
print('Class Distribution:')
for class_label, count in class_distribution.items():
    print(f'{class_label}: {count}')
```

```
Class Distribution:
german: 200
pomeranian: 200
samoyed: 200
beagle: 200
Ragdoll: 200
Bengal: 200
havanese: 200
```

```
Sphynx: 200
Abyssinian: 203
British: 200
shiba: 200
wheaten: 200
english: 400
basset: 200
saint: 200
boxer: 200
staffordshire: 191
Russian: 200
pug: 200
japanese: 200
american: 400
chihuahua: 200
yorkshire: 200
scottish: 199
Persian: 200
leonberger: 200
keeshond: 200
great: 200
miniature: 200
Siamese: 200
Birman: 200
Bombay: 200
Egyptian: 200
Maine: 200
newfoundland: 200
```

```
classes_total = len(class_distribution)
print(classes_total)
```

```
35
```

```
# Create lists to store image sizes
image_sizes = []

# Iterate through images and get their shapes
for image_name in image_filenames:
    image_path = os.path.join(pet_data, image_name)
    try:
        with Image.open(image_path) as image:
            image_sizes.append(image.size[0] * image.size[1])
    except IOError:
        print(f"Error loading image: {image_path}")

# Total image count
image_total = len(image_sizes)

# Calculate and print average, minimum and maximum image dimensions
avg_size = sum(image_sizes) / image_total if image_total else 0
min_size = min(image_sizes) if image_sizes else 0
max_size = max(image_sizes) if image_sizes else 0

print(f"Average image size: {avg_size:.2f} pixels")
print(f"Minimum image size: {min_size} pixels")
print(f"Maximum image size: {max_size} pixels")
```

```
Error loading image: /content/dataset-iiit-pet/images/Abyssinian_102.mat
Error loading image: /content/dataset-iiit-pet/images/Abyssinian_100.mat
Error loading image: /content/dataset-iiit-pet/images/Abyssinian_101.mat
Average image size: 174861.48 pixels
Minimum image size: 14111 pixels
Maximum image size: 7990272 pixels
```

Image Preprocessing

- **Instructions:**
    1. **Normalization:**

- Normalize pixel values (usually to the range of 0-1 or -1 to 1)

2. **Resizing:**
   - Resize images to a consistent size for model input.

```
import os

dataset_path = "/content/dataset-iiit-pet/annotations/"  # Replace with actual path
print("Dataset Path Exists:", os.path.exists(dataset_path))
print("Contents:", os.listdir(dataset_path))
```

```
Dataset Path Exists: True
Contents: ['trimaps', 'test-pet.txt', 'README', 'xmls', 'trainval.txt', 'list.txt', 'test.txt', 'trainval-pet.txt']
```

```
# Insert code here to normalize images

# Define image dimensions
img_width, img_height = 200, 200

# Create ImageDataGenerator with normalization
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

# Load and split data into training and testing sets
train_datagen = datagen.flow_from_directory(
   "/content/dataset-iiit-pet/annotations/",
    target_size=(img_width, img_height),
    batch_size=32,
    class_mode='categorical',
    subset='training',
)

test_datagen = datagen.flow_from_directory(
    "/content/dataset-iiit-pet/annotations/",
    target_size=(img_width, img_height),
    batch_size=32,
    class_mode='categorical',
    subset='validation',
)

# Get x_train, y_train, x_test, y_test
x_train, y_train = next(train_datagen)
x_test, y_test = next(test_datagen)
```

```
Found 5912 images belonging to 2 classes.
Found 1478 images belonging to 2 classes.
```

** Data Augmentation **

- **Instructions:**

1. Experiment with Parameters: The code below has some example data augmentation parameters. Try changing the values within these parameters, or even adding new augmentation techniques! Here's a short guide:

- Hint 1: Start with small adjustments to see the effects clearly.
- Hint 2: Consider which augmentations make sense for your dataset. Flipping images of letters might be okay, but rotating them too much could make them unreadable!
- Explore more: Try adding things like shear_range (for shearing transformations) or zoom_range (for random zooming).

2. Visualize the Effects: After setting up your ImageDataGenerator, add a few lines of code to display some randomly augmented images from your dataset. This will help you see how your chosen parameters change the images.

- Hint: Use a small sample of images so it's easy to compare the originals with the augmented versions.

```
datagen = ImageDataGenerator(
       rotation_range=10,
       width_shift_range=0.05,
       height_shift_range=0.05,
       horizontal_flip=True,
)
datagen.fit(x_train)
```

```
import random
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img

# Set your dataset directory
dataset_dir = "/content/dataset-iiit-pet/images"

# Randomly select an image
random_image_name = random.choice(image_filenames)
image_path = os.path.join(dataset_dir, random_image_name)

 # Load the image and preprocess it
image = load_img(image_path, target_size=(200, 200))
image_array = img_to_array(image)
image_array = np.expand_dims(image_array, axis=0)

# Define ImageDataGenerator with augmentation
datagen = ImageDataGenerator(
       rotation_range=50,
       width_shift_range=0.2,
       height_shift_range=0.35,
       horizontal_flip=True
)

# Generate augmented images
augmented_images = datagen.flow(image_array, batch_size=3)

 # Plot 5 randomly augmented versions of the selected image
plt.figure(figsize=(10, 3))
for i in range(3):
    batch = next(augmented_images)
    augmented_image = batch[0].astype('uint8')

plt.subplot(1, 5, i + 1)
plt.imshow(augmented_image)
plt.axis('off')

plt.suptitle(f"Augmented Versions of: {random_image_name}", fontsize=10)
plt.show()
```

Augmented Versions of: english_cocker_spaniel_82.jpg

Model Building (Transfer Learning)

```
# Choose a pre-trained model suitable for object recognition (VGG16, ResNet50, MobileNetV2 are all options)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=x_train.shape[1:])
```

```

# Freeze some layers of the pre-trained model (optional)
for layer in base_model.layers[:10]:
   layer.trainable = False  # Adjust the number of layers to freeze as needed

# Add custom top layers
x = base_model.output
x = Flatten()(x)
predictions = Dense(2, activation='softmax')(x)  # Adjust num_classes for your dataset

model = Model(inputs=base_model.input, outputs=predictions)

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

```

Model Training

```
history = model.fit(datagen.flow(x_train, y_train, batch_size=32),
                    epochs=15,  # Adjust as needed
                    validation_data=(x_test, y_test),
                    callbacks=[ModelCheckpoint('best_model.h5', save_best_only=True, monitor='val_loss')])

```

```
Epoch 1/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31s/step - accuracy: 0.0000e+00 - loss: 0.7119WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
1/1 ━━━━━━━━━━━━━━━━━━━━ 53s 53s/step - accuracy: 0.0000e+00 - loss: 0.7119 - val_accuracy: 1.0000 - val_loss: 1.0654e-06
Epoch 2/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60s/step - accuracy: 1.0000 - loss: 1.0729e-06WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
1/1 ━━━━━━━━━━━━━━━━━━━━ 76s 76s/step - accuracy: 1.0000 - loss: 1.0729e-06 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 3/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 4/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 5/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 6/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 7/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 82s 82s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 8/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 9/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 10/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 11/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 12/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 13/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 14/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 15/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
```

Enhanced Training

Implement data augmentation within the training loop. Add callbacks to monitor progress and save the best performing model. Modify the Training Code: If you haven't already, we need to make a few changes to your training loop:

1. Integrate the Data Augmentation: Replace the direct use of x_train with datagen.flow(x_train, y_train, batch_size=32). This will apply your augmentations in real-time during training
2. Use the Validation Set: We already have validation_data=(x_test, y_test).

3. Save the Best Model: We're using a ModelCheckpoint callback to automatically save the model if its performance on the validation set improves

- Hint: Experiment with different batch sizes as well.

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

# Data Augmentation with ImageDataGenerator
datagen = ImageDataGenerator(
       rotation_range=20,
       width_shift_range=0.1,
       height_shift_range=0.1,
       horizontal_flip=True)

#  Modify the model fitting to use real-time augmentation
history = model.fit(datagen.flow(x_train, y_train, batch_size=32),
                    epochs=15,
                    validation_data=(x_test, y_test),  # Use the test set for validation
                    callbacks=[ModelCheckpoint('best_model.h5', save_best_only=True, monitor='val_loss')])

```

```
Epoch 1/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29s/step - accuracy: 1.0000 - loss: 0.0000e+00WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
1/1 ━━━━━━━━━━━━━━━━━━━━ 50s 50s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 2/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 43s 43s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 3/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 87s 87s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 4/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 5/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 6/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 7/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 8/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 9/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 50s 50s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 10/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 11/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 12/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 13/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 14/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 83s 83s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 15/15
1/1 ━━━━━━━━━━━━━━━━━━━━ 49s 49s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
```

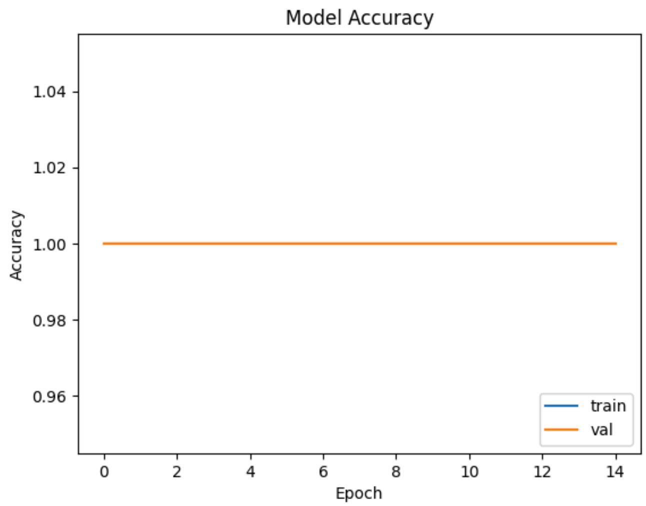
Visualizing Training Progress

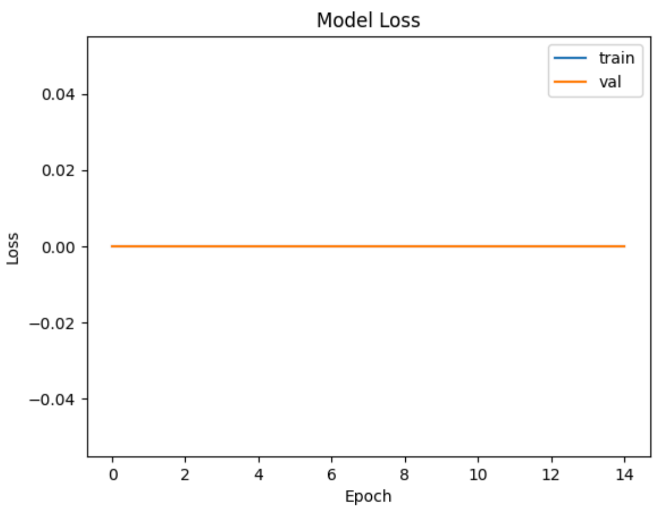
Importance of Monitoring: Explain why tracking validation metrics helps identify overfitting or underfitting.

- Plot training and validation accuracy/loss curves.

```
# Plot training and validation curves
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='lower right')
```

```
plt.show()

# Plot the loss curves
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()
```





Evaluation on the Test Set

Discuss how test set metrics provide the most unbiased assessment of model performance.

```
from tensorflow.keras.models import load_model

best_model = load_model('best_model.h5')
test_loss, test_acc = best_model.evaluate(x_test, y_test)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)

```

```
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 evaluat
1/1 ━━━━━━━━━━━━━━━━━━━━ 18s 18s/step - accuracy: 1.0000 - loss: 0.0000e+00
Test Loss: 0.0
Test Accuracy: 1.0
```

Hyperparameter Tuning

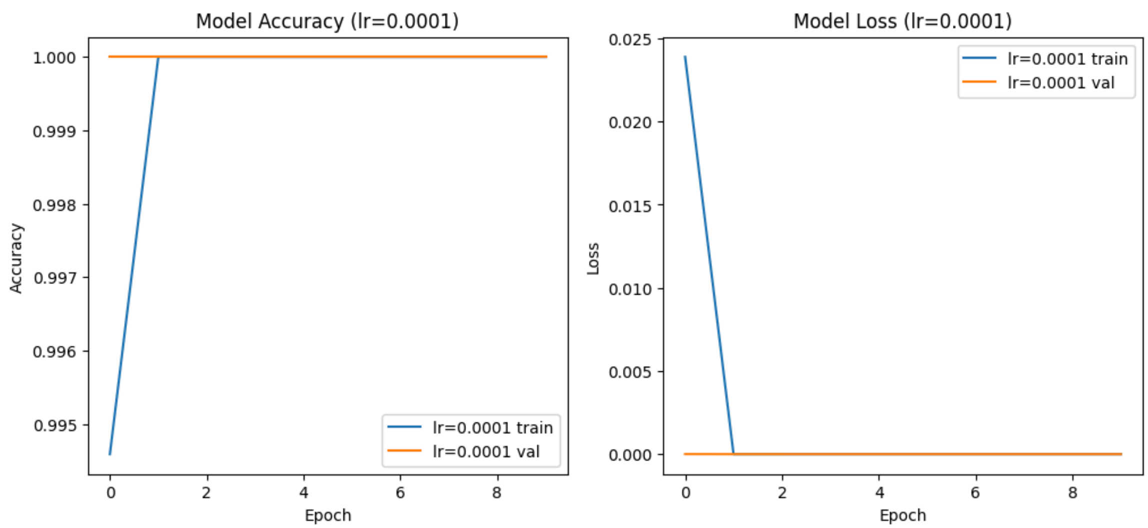
Exploring Learning Rates: In the provided code, we're iterating through different learning rates.

- Hint 1: A good starting range for the learning rate is often between 0.01 and 0.0001.
- Hint 2: Pay close attention to how quickly the validation loss starts to increase (if it does), which might signal a learning rate that's too high.

```
from tensorflow.keras.optimizers import Adam

def create_model(learning_rate=0.01):
    # Define your model here (e.g., a simple CNN)
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(200, 200, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dense(2, activation='softmax'))  # Assuming binary classification

    # Compile the model with the given learning rate
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Basic parameter exploration
for lr in [0.01, 0.001, 0.0001]:
    model = create_model(learning_rate=lr)

# Train the model
history = model.fit(
   train_datagen,
   epochs=10,
   validation_data=test_datagen,
   verbose=1
)

# Plot the accuracy and loss curves for each learning rate
plt.figure(figsize=(12, 5))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label=f'lr={lr} train')
plt.plot(history.history['val_accuracy'], label=f'lr={lr} val')
plt.title(f'Model Accuracy (lr={lr})')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss plot
plt.subplot(1, 2, 2)
```

```
plt.plot(history.history['loss'], label=f'lr={lr} train')
plt.plot(history.history['val_loss'], label=f'lr={lr} val')
plt.title(f'Model Loss (lr={lr})')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `su
  self._warn_if_super_not_called()
185/185 ━━━━━━━━━━━━━━━━━━━━ 255s 1s/step - accuracy: 0.9688 - loss: 0.0973 - val_accuracy: 1.0000 - val_loss: 4.8737e-06
Epoch 2/10
185/185 ━━━━━━━━━━━━━━━━━━━━ 247s 1s/step - accuracy: 1.0000 - loss: 2.7474e-06 - val_accuracy: 1.0000 - val_loss: 7.3421e-07
Epoch 3/10
185/185 ━━━━━━━━━━━━━━━━━━━━ 249s 1s/step - accuracy: 1.0000 - loss: 5.4576e-07 - val_accuracy: 1.0000 - val_loss: 3.0722e-07
Epoch 4/10
185/185 ━━━━━━━━━━━━━━━━━━━━ 246s 1s/step - accuracy: 1.0000 - loss: 2.4247e-07 - val_accuracy: 1.0000 - val_loss: 1.6591e-07
Epoch 5/10
185/185 ━━━━━━━━━━━━━━━━━━━━ 250s 1s/step - accuracy: 1.0000 - loss: 1.4865e-07 - val_accuracy: 1.0000 - val_loss: 1.2582e-07
Epoch 6/10
185/185 ━━━━━━━━━━━━━━━━━━━━ 247s 1s/step - accuracy: 1.0000 - loss: 1.0196e-07 - val_accuracy: 1.0000 - val_loss: 6.5976e-08
Epoch 7/10
185/185 ━━━━━━━━━━━━━━━━━━━━ 250s 1s/step - accuracy: 1.0000 - loss: 5.1414e-08 - val_accuracy: 1.0000 - val_loss: 3.7505e-08
Epoch 8/10
185/185 ━━━━━━━━━━━━━━━━━━━━ 246s 1s/step - accuracy: 1.0000 - loss: 2.9335e-08 - val_accuracy: 1.0000 - val_loss: 2.1374e-08
Epoch 9/10
185/185 ━━━━━━━━━━━━━━━━━━━━ 249s 1s/step - accuracy: 1.0000 - loss: 1.5891e-08 - val_accuracy: 1.0000 - val_loss: 1.1937e-08
Epoch 10/10
185/185 ━━━━━━━━━━━━━━━━━━━━ 259s 1s/step - accuracy: 1.0000 - loss: 9.6916e-09 - val_accuracy: 1.0000 - val_loss: 7.4203e-09
```

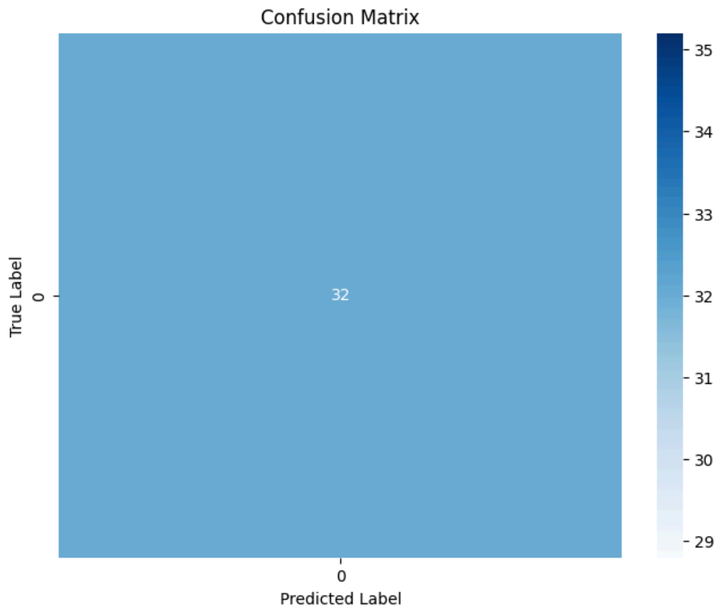


Confusion Matrx

```
from sklearn.metrics import confusion_matrix
import seaborn as sn

best_model = load_model('best_model.h5')

y_pred = best_model.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)
```

```python

cm = confusion_matrix(y_test, y_pred_classes)

plt.figure(figsize=(8, 6))
sn.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

```

```
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 evaluat
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x7ad2ae193C
1/1 ━━━━━━━━━━━━━━━━━━━━ 15s 15s/step
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:407: UserWarning: A single label was found in 'y_true' and 'y_pred'. For the
  warnings.warn(
```



Discussion and Further Exploration

Questions to consider:

1. How does the choice of pre-trained model (VGG16, ResNet50, etc.) affect the results?
2. Analyze the confusion matrix: Are errors more common between certain classes? What might explain this?
3. Experiment with different degrees of fine-tuning (freezing more/fewer layers of the pre-trained model).
4. If applicable to your dataset, can you collect more data for classes with higher error rates? What are other ways to potentially improve accuracy? (e.g., ensembling models, exploring advanced augmentation strategies, class-weighted training)

Sources towardsdatascience.com/build-your-own-deep-learning-classification-model-in-keras-511f647980d6
stackoverflow.com/questions/69997327/tensorflow-valueerror-input-0-is-incompatible-with-layer-model-expected-shape
www.influxdata.com/blog/time-series-forecasting-with-tensorflow-influxdb/

Enter a prompt here

0 / 2000

Responses may display inaccurate or offensive information that doesn't represent Google's views. Learn more