```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import cv2
import os
import zipfile
from tensorflow.keras.models import Sequential
```

```
!pip install kaggle

# Upload your Kaggle API credentials JSON file
from google.colab import files
files.upload()

!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# Download the dataset from Kaggle
!kaggle datasets download -d mohamedhanyyy/chest-ctscan-images
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/di
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-pack
```

Choose Files kaggle.json

- **kaggle.json**(application/json) - 63 bytes, last modified: 7/25/2023 - 100% done

```
Saving kaggle.json to kaggle.json
Downloading chest-ctscan-images.zip to /content
 82% 97.0M/119M [00:01<00:00, 68.9MB/s]
```

```
with zipfile.ZipFile("chest-ctscan-images.zip", "r") as zip_ref:
    zip_ref.extractall("/content/chest-ctscan-images")
```

```
import numpy as np
import cv2
import os
from sklearn.utils import shuffle
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Function to load the dataset
def load_dataset(dataset_folder_path):
    data = []
    labels = []

    label_to_int = {}  # Dictionary to map labels to integers
    int_label = 0

    for folder_name in os.listdir(dataset_folder_path):
        if folder_name not in label_to_int:
            label_to_int[folder_name] = int_label
            int_label += 1

        folder_path = os.path.join(dataset_folder_path, folder_name)
        for image_name in os.listdir(folder_path):
            image_path = os.path.join(folder_path, image_name)
            image = cv2.imread(image_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            image = cv2.resize(image, (224, 224))  # VGG19 input size
            data.append(image)
```

```
            labels.append(folder_name)

    # Convert the lists to numpy arrays
    data = np.array(data)
    labels = np.array(labels)

    return data, labels, label_to_int

# Load the test dataset
test_dataset_path = "/content/chest-ctscan-images/Data/test/"
test_data, test_labels, label_to_int = load_dataset(test_dataset_path)

# Load the train dataset
train_dataset_path = "/content/chest-ctscan-images/Data/train/"
train_data, train_labels, label_to_int = load_dataset(train_dataset_path)

# Load the validation dataset
valid_dataset_path = "/content/chest-ctscan-images/Data/valid/"
valid_data, valid_labels, label_to_int = load_dataset(valid_dataset_path)

# Shuffle the datasets
train_data, train_labels = shuffle(train_data, train_labels, random_state=42)
test_data, test_labels = shuffle(test_data, test_labels, random_state=42)
valid_data, valid_labels = shuffle(valid_data, valid_labels, random_state=42)

# Normalize the pixel values to [0, 1]
train_data = train_data.astype('float32') / 255.0
test_data = test_data.astype('float32') / 255.0
valid_data = valid_data.astype('float32') / 255.0

# Convert the labels to integers using label_to_int dictionary
train_labels_mapped = np.array([label_to_int[label] for label in train_labels])
# Convert the labels to integers using label_to_int dictionary
train_labels_mapped = np.array([label_to_int[label] for label in train_labels])

# Convert the labels to integers using label_to_int dictionary
test_labels_mapped = []
for label in test_labels:
    if label in label_to_int:
        test_labels_mapped.append(label_to_int[label])
    else:
        # Handle missing or unexpected labels (e.g., assign a unique integer or skip the sample)
        # You can also ignore the sample if needed.
        pass

test_labels_mapped = np.array(test_labels_mapped)

# Convert the labels to integers using label_to_int dictionary
valid_labels_mapped = np.array([label_to_int[label] for label in valid_labels])

valid_labels_mapped = np.array([label_to_int[label] for label in valid_labels])

# Convert the labels to one-hot encoded vectors
num_classes = len(label_to_int)
train_labels = tf.keras.utils.to_categorical(train_labels_mapped, num_classes)
test_labels = tf.keras.utils.to_categorical(test_labels_mapped, num_classes)
valid_labels = tf.keras.utils.to_categorical(valid_labels_mapped, num_classes)

# Rest of the code for building, training, and evaluating the model...
```

```
from sklearn.model_selection import train_test_split

# Split the original training dataset into new training and test datasets
new_train_data, new_test_data, new_train_labels, new_test_labels = train_test_split(
    train_data, train_labels, test_size=0.2, random_state=42
)

# Combine the new training and test datasets
combined_data = np.concatenate((new_train_data, new_test_data), axis=0)
combined_labels = np.concatenate((new_train_labels, new_test_labels), axis=0)

# Shuffle the combined dataset
combined_data, combined_labels = shuffle(combined_data, combined_labels, random_state=42)

# Split the shuffled dataset into final training and test datasets
train_data, test_data, train_labels, test_labels = train_test_split(
    combined_data, combined_labels, test_size=0.2, random_state=42
)
```

```
# Print the sizes of the final datasets
print("Final training dataset size:", len(train_data))
print("Final test dataset size:", len(test_data))
```

```
Final training dataset size: 490
Final test dataset size: 123
```

```
# Build VGG16 model
input_shape = (224, 224, 3)  # VGG16 input size
input_layer = Input(shape=input_shape)

# Block 1
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv1')(input_layer)
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv2')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)

# Block 2
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv1')(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv2')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)

# Block 3
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv1')(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv2')(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)

# Block 4
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv1')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)

# Block 5
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv1')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)

x = Flatten(name='flatten')(x)
x = Dense(4096, activation='relu', name='fc1')(x)
x = Dense(4096, activation='relu', name='fc2')(x)
output_layer = Dense(num_classes, activation='softmax', name='predictions')(x)

vgg16_model = Model(inputs=input_layer, outputs=output_layer)

# Compile the model
vgg16_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Print the model summary
vgg16_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808
```

```
 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

 predictions (Dense)         (None, 4)                 16388

=================================================================
Total params: 134,276,932
Trainable params: 134,276,932
Non-trainable params: 0
_________________________________________________________________
```

```
print(len(train_data), len(train_labels))
print(len(test_data), len(test_labels))
print(len(valid_data), len(valid_labels))
```

```
490 490
123 123
72 72
```

```
# Training the model
batch_size = 32
epochs = 10

history = vgg16_model.fit(train_data, train_labels, batch_size=batch_size, epochs=epochs, validation_data=(test_data, test_labels))
```

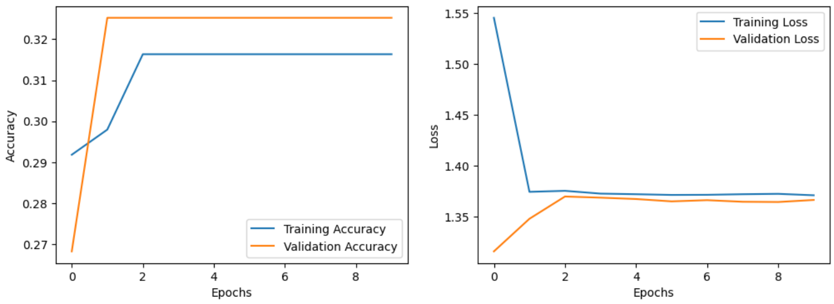
```
Epoch 1/10
16/16 [==============================] - 45s 1s/step - loss: 1.5452 - accuracy: 0.2918 - val_loss: 1.3161 - val_accuracy: 0.2683
Epoch 2/10
16/16 [==============================] - 7s 432ms/step - loss: 1.3745 - accuracy: 0.2980 - val_loss: 1.3480 - val_accuracy: 0.3252
Epoch 3/10
16/16 [==============================] - 7s 443ms/step - loss: 1.3755 - accuracy: 0.3163 - val_loss: 1.3699 - val_accuracy: 0.3252
Epoch 4/10
16/16 [==============================] - 7s 444ms/step - loss: 1.3727 - accuracy: 0.3163 - val_loss: 1.3688 - val_accuracy: 0.3252
Epoch 5/10
16/16 [==============================] - 7s 443ms/step - loss: 1.3721 - accuracy: 0.3163 - val_loss: 1.3674 - val_accuracy: 0.3252
Epoch 6/10
16/16 [==============================] - 7s 433ms/step - loss: 1.3714 - accuracy: 0.3163 - val_loss: 1.3651 - val_accuracy: 0.3252
Epoch 7/10
16/16 [==============================] - 7s 440ms/step - loss: 1.3716 - accuracy: 0.3163 - val_loss: 1.3663 - val_accuracy: 0.3252
Epoch 8/10
16/16 [==============================] - 7s 440ms/step - loss: 1.3721 - accuracy: 0.3163 - val_loss: 1.3647 - val_accuracy: 0.3252
Epoch 9/10
16/16 [==============================] - 7s 432ms/step - loss: 1.3725 - accuracy: 0.3163 - val_loss: 1.3645 - val_accuracy: 0.3252
Epoch 10/10
16/16 [==============================] - 7s 423ms/step - loss: 1.3711 - accuracy: 0.3163 - val_loss: 1.3665 - val_accuracy: 0.3252
```

```
test_loss, test_accuracy = vgg16_model.evaluate(test_data, test_labels, batch_size=batch_size)
print('Test Loss:', test_loss)
print('Test Accuracy:', test_accuracy)
```

```
4/4 [==============================] - 1s 129ms/step - loss: 1.3665 - accuracy: 0.3252
Test Loss: 1.36650550365448
Test Accuracy: 0.3252032399177551
```

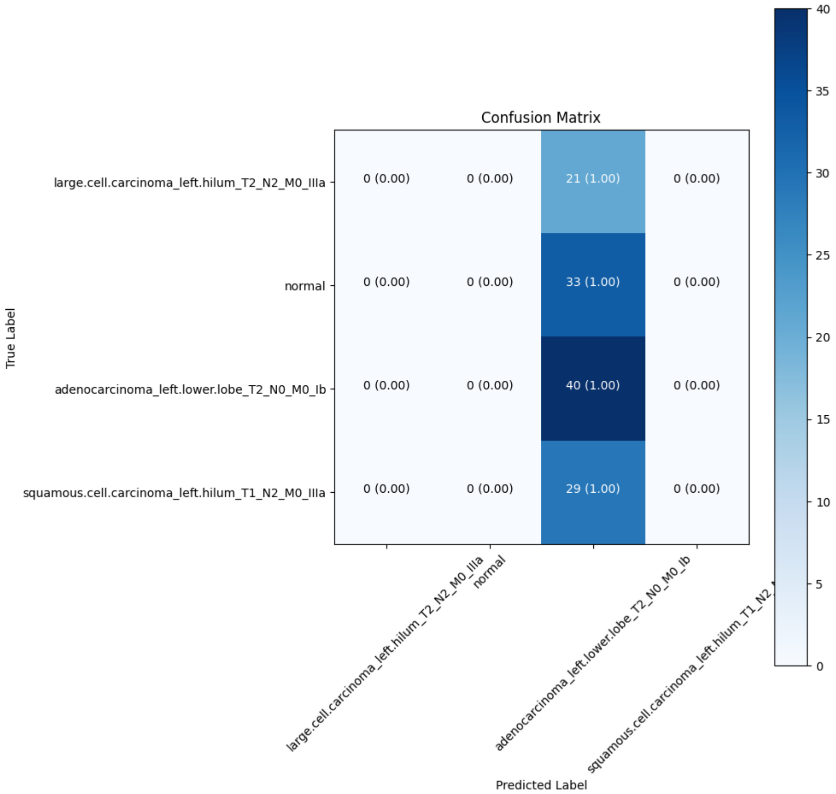
```
def plot_history(history):
    plt.figure(figsize=(12, 4))

    # Plot accuracy
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
```

```
    # Plot loss
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.show()

# Plot the training history
plot_history(history)
```




```
import itertools
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Assuming you have already trained your vgg19_model and obtained predictions on the test set
predictions = vgg16_model.predict(test_data)
predicted_labels = np.argmax(predictions, axis=1)

# Compute the confusion matrix
cm = confusion_matrix(np.argmax(test_labels, axis=1), predicted_labels)

# Get class labels from the label_to_int dictionary
label_to_int_inv = {v: k for k, v in label_to_int.items()}
classes = [label_to_int_inv[i] for i in range(num_classes)]

# Plot the confusion matrix
plt.figure(figsize=(10, 10))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

# Normalize the confusion matrix values for better readability
cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

# Use white text for darker cells, and black text otherwise
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], 'd') + " (" + format(cm_normalized[i, j], '.2f') + ")",
             horizontalalignment='center', color='white' if cm[i, j] > thresh else 'black')

plt.tight_layout()
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```

```
4/4 [==============================] - 1s 116ms/step
```




```
from sklearn.metrics import accuracy_score

# Assuming you have already trained your model and obtained predictions on the test set
# predictions = model.predict(test_data)
# predicted_labels = np.argmax(predictions, axis=1)

# Compute the accuracy
accuracy = accuracy_score(np.argmax(test_labels, axis=1), predicted_labels)

# Display the accuracy percentage
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

```
Accuracy: 32.52%
```

```
from sklearn.metrics import classification_report

# Assuming you have already trained your model and obtained predictions on the test set
# predictions = model.predict(test_data)
# predicted_labels = np.argmax(predictions, axis=1)

# Compute the classification report
report = classification_report(np.argmax(test_labels, axis=1), predicted_labels)

# Display the classification report
print("Classification Report:")
print(report)
```

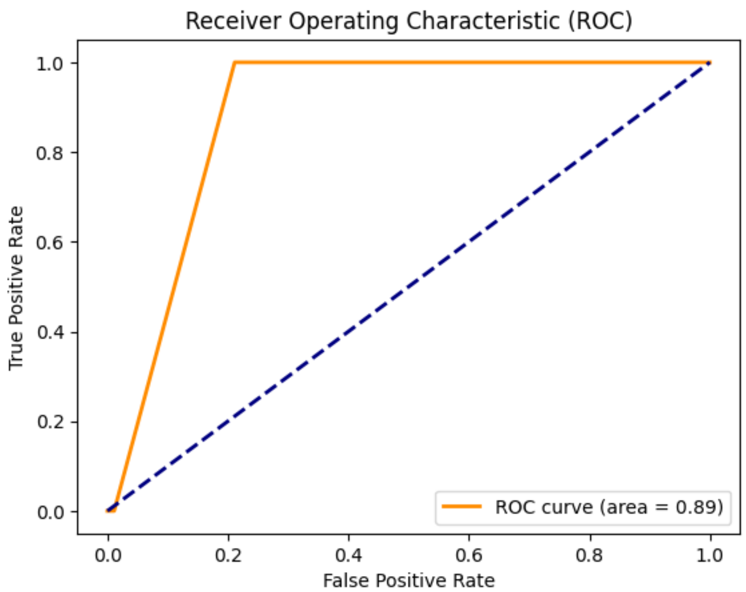
```
Classification Report:
              precision    recall  f1-score   support
```

```
           0       0.00      0.00      0.00        21
           1       0.00      0.00      0.00        33
           2       0.33      1.00      0.49        40
           3       0.00      0.00      0.00        29

    accuracy                           0.33       123
   macro avg       0.08      0.25      0.12       123
weighted avg       0.11      0.33      0.16       123

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
  _warn_prf(average, modifier, msg_start, len(result))
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(np.argmax(test_labels, axis=1), predicted_labels, average='weighted')
recall = recall_score(np.argmax(test_labels, axis=1), predicted_labels, average='weighted')
f1 = f1_score(np.argmax(test_labels, axis=1), predicted_labels, average='weighted')

print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-Score: {:.2f}".format(f1))
```

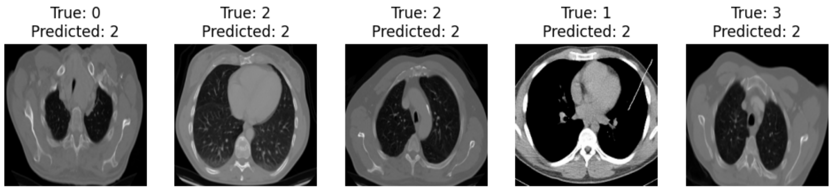
```
Precision: 0.11
Recall: 0.33
F1-Score: 0.16
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined a
  _warn_prf(average, modifier, msg_start, len(result))
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Assuming binary classification (two classes) for simplicity
# If your problem is multi-class, you need to modify the code accordingly

fpr, tpr, thresholds = roc_curve(test_labels[:, 1], predictions[:, 1])
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()
```



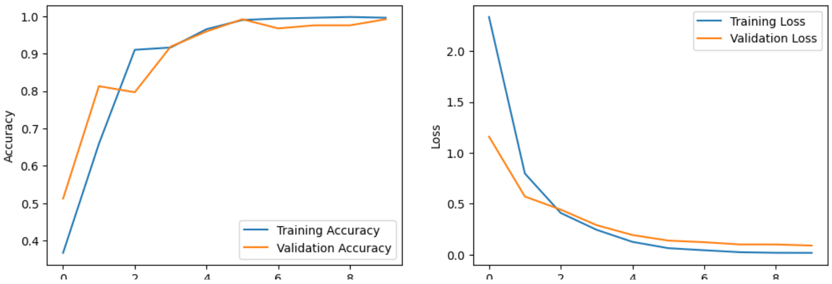

```
import random

# Assuming you have already trained your model and obtained predictions on the test set
```

```
# predictions = model.predict(test_data)
# predicted_labels = np.argmax(predictions, axis=1)

# Choose a random subset of test data and corresponding true and predicted labels
random_indices = random.sample(range(len(test_data)), 5)
sample_images = test_data[random_indices]
sample_true_labels = np.argmax(test_labels[random_indices], axis=1)
sample_predicted_labels = predicted_labels[random_indices]

# Visualize the sample images with true and predicted labels
plt.figure(figsize=(12, 6))
for i in range(len(sample_images)):
    plt.subplot(1, 5, i + 1)
    plt.imshow(sample_images[i])
    plt.title(f"True: {sample_true_labels[i]}\nPredicted: {sample_predicted_labels[i]}")
    plt.axis('off')
plt.show()
```




```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Flatten, Dense
from tensorflow.keras.optimizers import Adam

# Step 1: Load a pre-trained VGG16 model with pre-trained weights and without the final classification layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Step 2: Add your own classification layers on top of the pre-trained VGG19 model
x = base_model.output
x = Flatten()(x)
x = Dense(4096, activation='relu')(x)
x = Dense(4096, activation='relu')(x)
output_layer = Dense(num_classes, activation='softmax')(x)

# Step 3: Freeze the pre-trained layers to avoid overfitting on your limited dataset
for layer in base_model.layers:
    layer.trainable = False

# Step 4: Create the fine-tuned model by combining the base VGG19 model with your classification layers
fine_tuned_model = Model(inputs=base_model.input, outputs=output_layer)

# Step 5: Compile the model with a lower learning rate for fine-tuning
optimizer = Adam(learning_rate=0.0001)
fine_tuned_model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

# Fine-tune the model with your training data
batch_size = 32
epochs = 10

history = fine_tuned_model.fit(train_data, train_labels, batch_size=batch_size, epochs=epochs, validation_data=(test_data, test_labels))

# Evaluate the fine-tuned model on the test set
test_loss, test_accuracy = fine_tuned_model.evaluate(test_data, test_labels, batch_size=batch_size)
print('Test Loss:', test_loss)
print('Test Accuracy:', test_accuracy)

# Plot the training history
plot_history(history)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications
58889256/58889256 [==============================] - 0s 0us/step
Epoch 1/10
16/16 [==============================] - 5s 235ms/step - loss: 2.3342 - accuracy:
Epoch 2/10
16/16 [==============================] - 3s 185ms/step - loss: 0.7970 - accuracy:
Epoch 3/10
16/16 [==============================] - 3s 190ms/step - loss: 0.4095 - accuracy:
Epoch 4/10
16/16 [==============================] - 3s 186ms/step - loss: 0.2441 - accuracy:
Epoch 5/10
16/16 [==============================] - 3s 214ms/step - loss: 0.1257 - accuracy:
Epoch 6/10
16/16 [==============================] - 3s 196ms/step - loss: 0.0637 - accuracy:
Epoch 7/10
16/16 [==============================] - 3s 191ms/step - loss: 0.0432 - accuracy:
Epoch 8/10
16/16 [==============================] - 3s 199ms/step - loss: 0.0242 - accuracy:
Epoch 9/10
16/16 [==============================] - 3s 199ms/step - loss: 0.0183 - accuracy:
Epoch 10/10
16/16 [==============================] - 3s 195ms/step - loss: 0.0176 - accuracy:
4/4 [==============================] - 1s 135ms/step - loss: 0.0893 - accuracy: 0.
Test Loss: 0.08928515762090683
Test Accuracy: 0.9918699264526367
```




```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.applications import ResNet50, VGG16, ResNet101, VGG19, DenseNet201, EfficientNetB4, MobileNetV2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Flatten,Dropout,BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.layers import Input

from tensorflow.keras.utils import plot_model
from IPython.display import Image


train_path = "chest-ctscan-images/Data/train"
valid_path = "chest-ctscan-images/Data/valid"
test_path = "chest-ctscan-images/Data/test"


INPUT_SHAPE = (460,460,3)
NUM_CLASSES=4
train_datagen = ImageDataGenerator(
    dtype='float32',
    preprocessing_function=preprocess_input,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=False
)
val_datagen = ImageDataGenerator(
    dtype='float32',
    preprocessing_function=preprocess_input
)
test_datagen = ImageDataGenerator(
    dtype='float32',
    preprocessing_function=preprocess_input
)

train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=(460,460),
    batch_size=32,
    class_mode='categorical',
```

```
)


test_generator = test_datagen.flow_from_directory(
    test_path,
    target_size=(460,460),
    batch_size=32,
    class_mode='categorical',
)
validation_generator = val_datagen.flow_from_directory(
    valid_path,
    target_size=(460,460),
    batch_size=32,
    class_mode='categorical',
)
```

```
Found 613 images belonging to 4 classes.
Found 315 images belonging to 4 classes.
Found 72 images belonging to 4 classes.
```

```
#Using ResNet50
base_model = ResNet50(include_top=False,pooling='av',weights='imagenet',input_shape=(INPUT_SHAPE))
for layer in base_model.layers:
    layer.trainable = False
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(256,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(NUM_CLASSES,activation='softmax'))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 15, 15, 2048)      23587712

 flatten_3 (Flatten)         (None, 460800)            0

 batch_normalization_2 (Batc  (None, 460800)           1843200
 hNormalization)

 dense_7 (Dense)             (None, 256)               117965056

 dropout_2 (Dropout)         (None, 256)               0

 dense_8 (Dense)             (None, 4)                 1028

=================================================================
Total params: 143,396,996
Trainable params: 118,887,684
Non-trainable params: 24,509,312
_________________________________________________________________
```

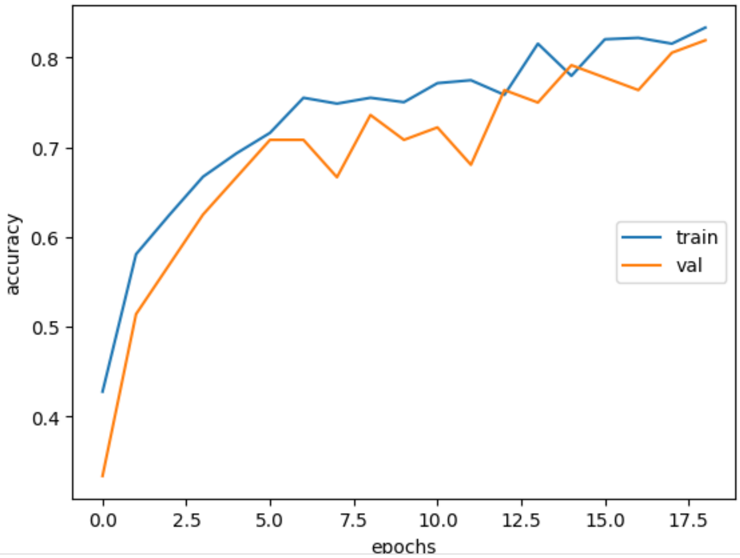
```
optimizer = tf.keras.optimizers.Adam(learning_rate= 0.00001)

model.compile(loss='categorical_crossentropy',optimizer=optimizer,metrics=['accuracy'])
checkpoint = ModelCheckpoint(
    filepath='Chest_CT_SCAN_ResNet50.h5',
    monitor='val_loss',
    save_best_only=True,
    verbose=1
)
earlystop = EarlyStopping(
    patience=10,
    verbose=1
)
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=30,
    callbacks=[checkpoint, earlystop],
    verbose=1
)
```

```
Epoch 7/30
20/20 [==============================] - ETA: 0s - loss: 0.7398 - accuracy: 0.7553
Epoch 7: val_loss improved from 0.93164 to 0.87358, saving model to Chest_CT_SCAN_ResNet50.h5
20/20 [==============================] - 65s 3s/step - loss: 0.7398 - accuracy: 0.7553 - val_loss: 0.8736 - val_accuracy: 0.7083
Epoch 8/30
20/20 [==============================] - ETA: 0s - loss: 0.6911 - accuracy: 0.7488
Epoch 8: val_loss did not improve from 0.87358
20/20 [==============================] - 44s 2s/step - loss: 0.6911 - accuracy: 0.7488 - val_loss: 0.9577 - val_accuracy: 0.6667
Epoch 9/30
20/20 [==============================] - ETA: 0s - loss: 0.7584 - accuracy: 0.7553
Epoch 9: val_loss improved from 0.87358 to 0.85581, saving model to Chest_CT_SCAN_ResNet50.h5
20/20 [==============================] - 54s 3s/step - loss: 0.7584 - accuracy: 0.7553 - val_loss: 0.8558 - val_accuracy: 0.7361
Epoch 10/30
20/20 [==============================] - ETA: 0s - loss: 0.7356 - accuracy: 0.7504
Epoch 10: val_loss did not improve from 0.85581
20/20 [==============================] - 44s 2s/step - loss: 0.7356 - accuracy: 0.7504 - val_loss: 0.8860 - val_accuracy: 0.7083
Epoch 11/30
20/20 [==============================] - ETA: 0s - loss: 0.6227 - accuracy: 0.7716
Epoch 11: val_loss did not improve from 0.85581
20/20 [==============================] - 44s 2s/step - loss: 0.6227 - accuracy: 0.7716 - val_loss: 0.9191 - val_accuracy: 0.7222
Epoch 12/30
20/20 [==============================] - ETA: 0s - loss: 0.6936 - accuracy: 0.7749
Epoch 12: val_loss did not improve from 0.85581
20/20 [==============================] - 43s 2s/step - loss: 0.6936 - accuracy: 0.7749 - val_loss: 1.1700 - val_accuracy: 0.6806
Epoch 13/30
20/20 [==============================] - ETA: 0s - loss: 0.7138 - accuracy: 0.7586
Epoch 13: val_loss did not improve from 0.85581
20/20 [==============================] - 44s 2s/step - loss: 0.7138 - accuracy: 0.7586 - val_loss: 1.0569 - val_accuracy: 0.7639
Epoch 14/30
20/20 [==============================] - ETA: 0s - loss: 0.5577 - accuracy: 0.8157
Epoch 14: val_loss did not improve from 0.85581
20/20 [==============================] - 43s 2s/step - loss: 0.5577 - accuracy: 0.8157 - val_loss: 1.0237 - val_accuracy: 0.7500
Epoch 15/30
20/20 [==============================] - ETA: 0s - loss: 0.6219 - accuracy: 0.7798
Epoch 15: val_loss did not improve from 0.85581
20/20 [==============================] - 44s 2s/step - loss: 0.6219 - accuracy: 0.7798 - val_loss: 0.8950 - val_accuracy: 0.7917
Epoch 16/30
20/20 [==============================] - ETA: 0s - loss: 0.5130 - accuracy: 0.8206
Epoch 16: val_loss did not improve from 0.85581
20/20 [==============================] - 43s 2s/step - loss: 0.5130 - accuracy: 0.8206 - val_loss: 0.8642 - val_accuracy: 0.7778
Epoch 17/30
20/20 [==============================] - ETA: 0s - loss: 0.5185 - accuracy: 0.8222
Epoch 17: val_loss did not improve from 0.85581
20/20 [==============================] - 43s 2s/step - loss: 0.5185 - accuracy: 0.8222 - val_loss: 0.9373 - val_accuracy: 0.7639
Epoch 18/30
20/20 [==============================] - ETA: 0s - loss: 0.5240 - accuracy: 0.8157
Epoch 18: val_loss did not improve from 0.85581
20/20 [==============================] - 44s 2s/step - loss: 0.5240 - accuracy: 0.8157 - val_loss: 0.8959 - val_accuracy: 0.8056
Epoch 19/30
20/20 [==============================] - ETA: 0s - loss: 0.5120 - accuracy: 0.8336
Epoch 19: val_loss did not improve from 0.85581
20/20 [==============================] - 43s 2s/step - loss: 0.5120 - accuracy: 0.8336 - val_loss: 0.9316 - val_accuracy: 0.8194
Epoch 19: early stopping
```

```
result = model.evaluate(test_generator)
```

```
10/10 [==============================] - 6s 596ms/step - loss: 0.4203 - accuracy: 0.8635
```

```
plt.plot(history.history['accuracy'], label = 'train',)
plt.plot(history.history['val_accuracy'], label = 'val')

plt.legend(loc = 'right')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.show()
```



✓ 0s completed at 2:02 AM