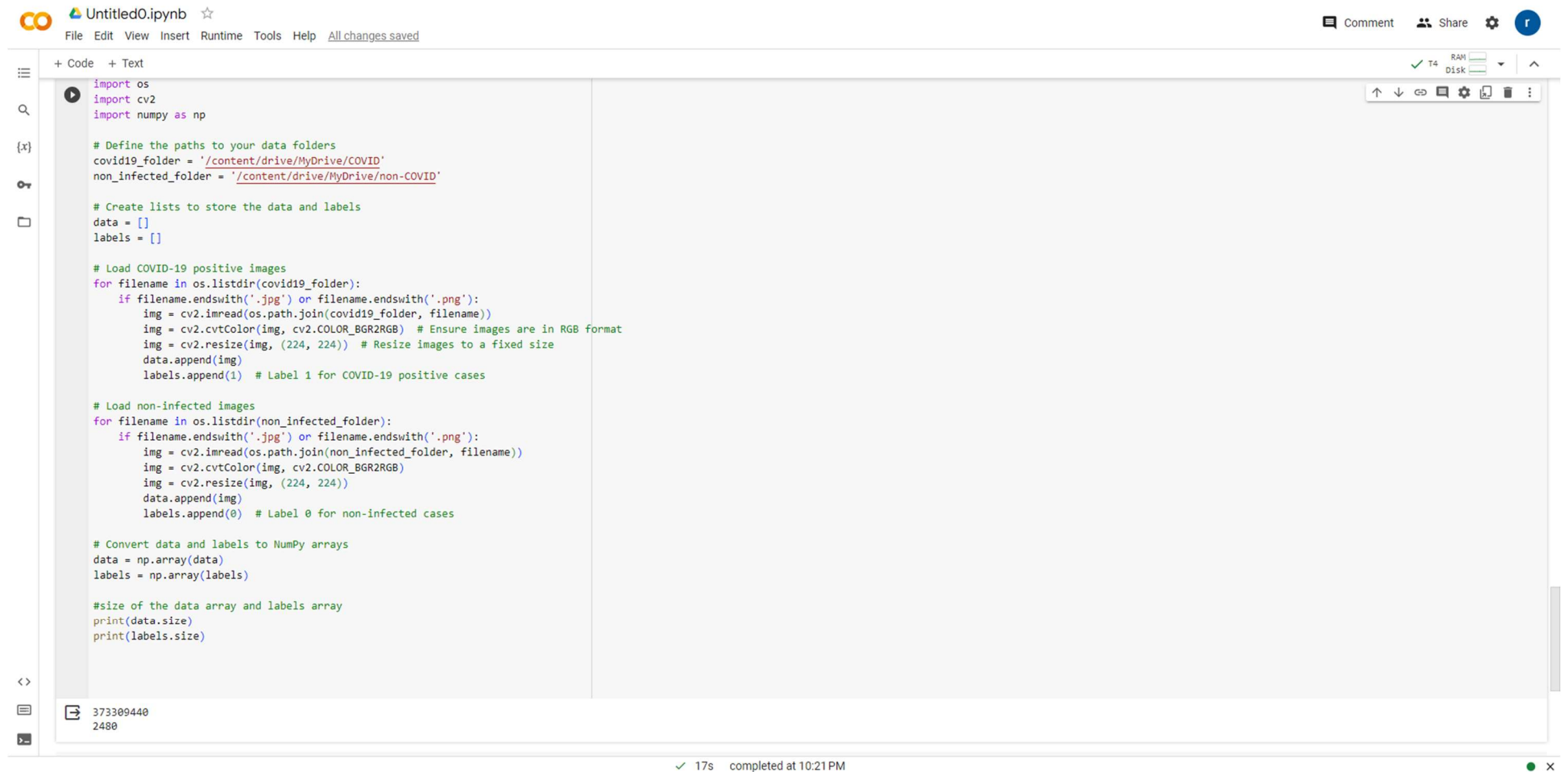


# FIRST SOLUTION CODE COMMIT:



The screenshot shows a Jupyter Notebook titled 'Untitled0.ipynb' with a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a status bar at the bottom indicating '17s completed at 10:21 PM'. The code is written in a light-themed editor and includes comments in green. It defines two folders, creates lists for data and labels, and loads images from both folders, converting them to RGB and resizing them to 224x224 pixels. The code also prints the size of the data and labels arrays.

```
import os
import cv2
import numpy as np

# Define the paths to your data folders
covid19_folder = '/content/drive/MyDrive/COVID'
non_infected_folder = '/content/drive/MyDrive/non-COVID'

# Create lists to store the data and labels
data = []
labels = []

# Load COVID-19 positive images
for filename in os.listdir(covid19_folder):
    if filename.endswith('.jpg') or filename.endswith('.png'):
        img = cv2.imread(os.path.join(covid19_folder, filename))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # Ensure images are in RGB format
        img = cv2.resize(img, (224, 224)) # Resize images to a fixed size
        data.append(img)
        labels.append(1) # Label 1 for COVID-19 positive cases

# Load non-infected images
for filename in os.listdir(non_infected_folder):
    if filename.endswith('.jpg') or filename.endswith('.png'):
        img = cv2.imread(os.path.join(non_infected_folder, filename))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (224, 224))
        data.append(img)
        labels.append(0) # Label 0 for non-infected cases

# Convert data and labels to NumPy arrays
data = np.array(data)
labels = np.array(labels)

#size of the data array and labels array
print(data.size)
print(labels.size)
```

Output: 373309440, 2480

## EXTENDED DESCRIPTION :

Here I am committing

1. Imports:

- Imported the necessary Python modules

2. Data Preparation:

- Load the data images folder, which contains 1252 CT scans of COVID-19-positive patients and 1230 CT scans of non-infected patients.

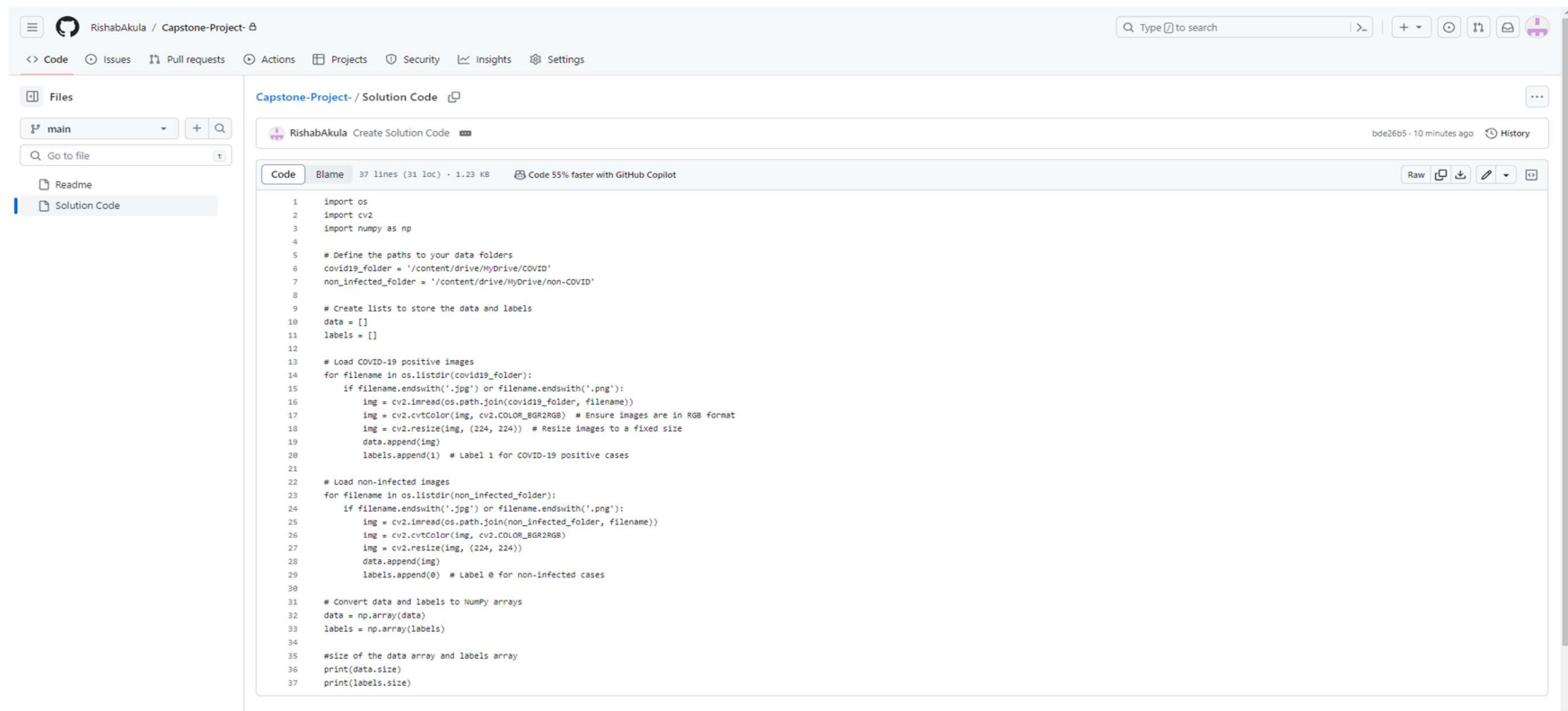
### 3. Load the data sets:

- Store data of both the data sets in the form of a data list and add label for each Covid-19 Positive image as 1 and non Covid 19 images as 0, and add into label list.

- Convert the data and labels list into NumPy Arrays

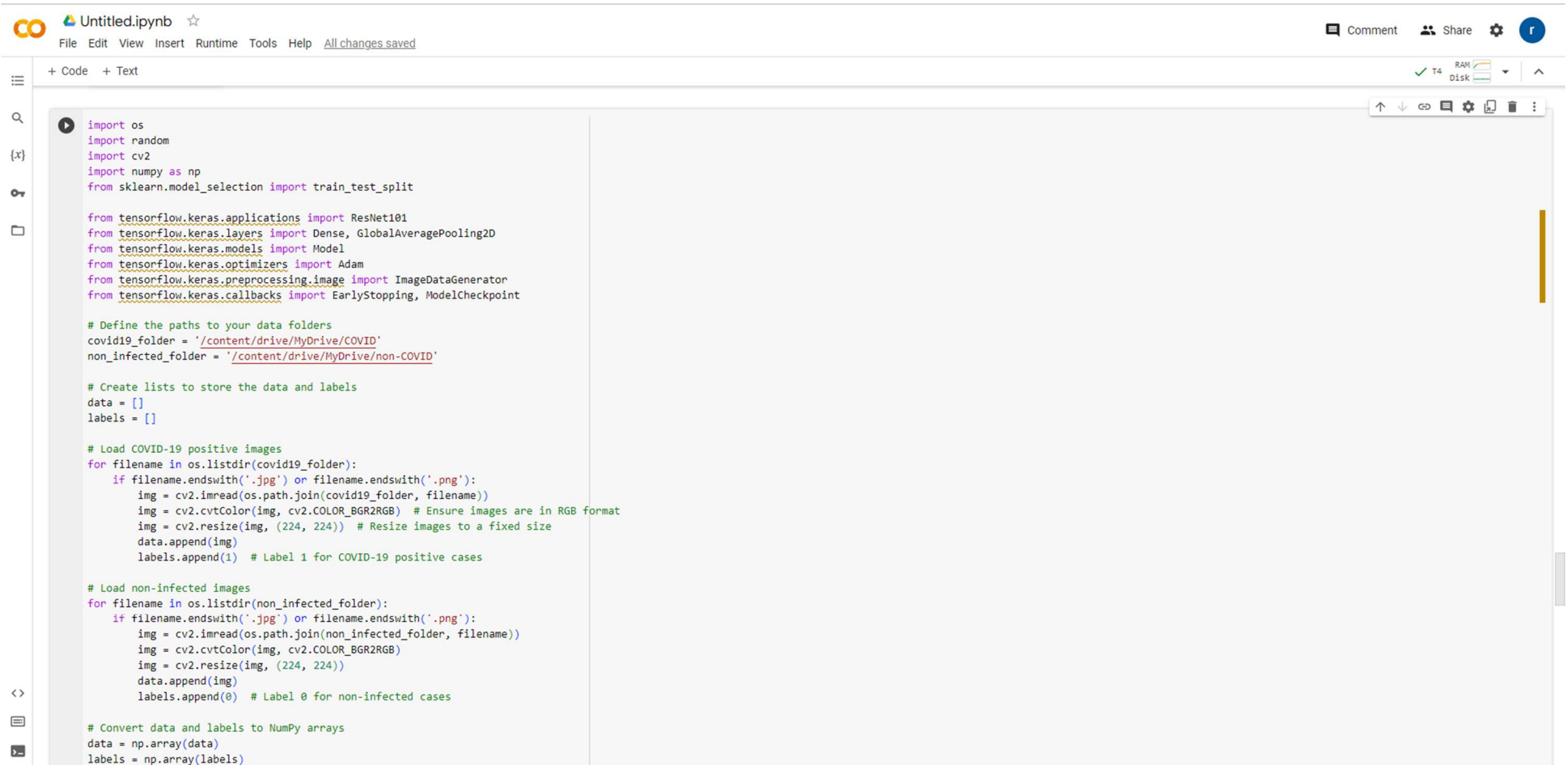
- Verified the size of the Data array and Labels array

## GITHUB UPLOADS :



```
1 import os
2 import cv2
3 import numpy as np
4
5 # Define the paths to your data folders
6 covid19_folder = '/content/drive/MyDrive/COVID'
7 non_infected_folder = '/content/drive/MyDrive/non-COVID'
8
9 # create lists to store the data and labels
10 data = []
11 labels = []
12
13 # Load COVID-19 positive images
14 for filename in os.listdir(covid19_folder):
15     if filename.endswith('.jpg') or filename.endswith('.png'):
16         img = cv2.imread(os.path.join(covid19_folder, filename))
17         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # Ensure images are in RGB format
18         img = cv2.resize(img, (224, 224)) # Resize images to a fixed size
19         data.append(img)
20         labels.append(1) # Label 1 for COVID-19 positive cases
21
22 # Load non-infected images
23 for filename in os.listdir(non_infected_folder):
24     if filename.endswith('.jpg') or filename.endswith('.png'):
25         img = cv2.imread(os.path.join(non_infected_folder, filename))
26         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
27         img = cv2.resize(img, (224, 224))
28         data.append(img)
29         labels.append(0) # Label 0 for non-infected cases
30
31 # Convert data and labels to Numpy arrays
32 data = np.array(data)
33 labels = np.array(labels)
34
35 #size of the data array and labels array
36 print(data.size)
37 print(labels.size)
```

## SECOND SOLUTION CODE COMMIT:



The screenshot shows a Jupyter Notebook titled "Untitled.ipynb" with a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a status bar indicating "All changes saved". The notebook is in "Code" mode. The code is as follows:

```
import os
import random
import cv2
import numpy as np
from sklearn.model_selection import train_test_split

from tensorflow.keras.applications import ResNet101
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

# Define the paths to your data folders
covid19_folder = '/content/drive/MyDrive/COVID'
non_infected_folder = '/content/drive/MyDrive/non-COVID'

# Create lists to store the data and labels
data = []
labels = []

# Load COVID-19 positive images
for filename in os.listdir(covid19_folder):
    if filename.endswith('.jpg') or filename.endswith('.png'):
        img = cv2.imread(os.path.join(covid19_folder, filename))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # Ensure images are in RGB format
        img = cv2.resize(img, (224, 224)) # Resize images to a fixed size
        data.append(img)
        labels.append(1) # Label 1 for COVID-19 positive cases

# Load non-infected images
for filename in os.listdir(non_infected_folder):
    if filename.endswith('.jpg') or filename.endswith('.png'):
        img = cv2.imread(os.path.join(non_infected_folder, filename))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (224, 224))
        data.append(img)
        labels.append(0) # Label 0 for non-infected cases

# Convert data and labels to NumPy arrays
data = np.array(data)
labels = np.array(labels)
```



+ Code + Text

✓ T4 RAM Disk



```
# Convert data and labels to NumPy arrays
data = np.array(data)
labels = np.array(labels)

#size of the data array and labels array
#print(data.size)
#print(labels.size)

# Split the data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(data, labels, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Define a list of ResNet architectures
#resnet_architectures = [ResNet18, ResNet34, ResNet50, ResNet101, ResNet152]
resnet_architectures = [ResNet101]

for resnet_model in resnet_architectures:
    # Load pre-trained ResNet model
    base_model = resnet_model(weights='imagenet', include_top=False)

    # Add custom top layers for binary classification
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(1, activation='sigmoid')(x)

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

    # Compile the model
    model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])

    # Create data generators with data augmentation
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        horizontal_flip=True
    )

    # Set up Early Stopping and Model Checkpoint callbacks
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)
    model_checkpoint = ModelCheckpoint(f'best_model {resnet_model.name}.h5', save_best_only=True, verbose=1)
```



+ Code + Text

✓ T4 RAM  
Disk

10m

{x}

Key

File

Find

Run

Stop

Help

About

Quit

Undo

Redo

Copy

Paste

Clear

Fullscreen

Terminal

Help

```
# Set up Early Stopping and Model Checkpoint callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)
model_checkpoint = ModelCheckpoint(f'best_model_{resnet_model.__name__}.h5', save_best_only=True, verbose=1)

# Train the model
history = model.fit_generator(
    train_datagen.flow(X_train, y_train, batch_size=32),
    validation_data=(X_val, y_val),
    epochs=100,
    callbacks=[early_stopping, model_checkpoint]
)

# Load the best model
model.load_weights(f'best_model_{resnet_model.__name__}.h5')

# Evaluate the model on the test set
test_predictions = model.predict(X_test)
test_predictions = (test_predictions > 0.5).astype(int)

# Selected Model Name
print(f"Model: {resnet_model.__name__}")
```

<ipython-input-2-707f49d8fbb8>:86: UserWarning: `Model.fit\_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.

```
history = model.fit_generator(
Epoch 1/100
55/55 [=====] - ETA: 0s - loss: 0.3197 - accuracy: 0.8606
Epoch 1: val_loss improved from inf to 0.50644, saving model to best_model_ResNet101.h5
55/55 [=====] - 119s 1s/step - loss: 0.3197 - accuracy: 0.8606 - val_loss: 0.5064 - val_accuracy: 0.7473
Epoch 2/100
55/55 [=====] - ETA: 0s - loss: 0.1971 - accuracy: 0.9217
Epoch 2: val_loss did not improve from 0.50644
55/55 [=====] - 37s 674ms/step - loss: 0.1971 - accuracy: 0.9217 - val_loss: 1.8342 - val_accuracy: 0.5188
Epoch 3/100
55/55 [=====] - ETA: 0s - loss: 0.1237 - accuracy: 0.9551
Epoch 3: val_loss did not improve from 0.50644
55/55 [=====] - 37s 659ms/step - loss: 0.1237 - accuracy: 0.9551 - val_loss: 1.9892 - val_accuracy: 0.4919
Epoch 4/100
55/55 [=====] - ETA: 0s - loss: 0.1174 - accuracy: 0.9603
Epoch 4: val_loss did not improve from 0.50644
55/55 [=====] - 38s 686ms/step - loss: 0.1174 - accuracy: 0.9603 - val_loss: 0.8140 - val_accuracy: 0.4409
Epoch 5/100
55/55 [=====] - ETA: 0s - loss: 0.0872 - accuracy: 0.9712
Epoch 5: val_loss did not improve from 0.50644
55/55 [=====] - 37s 665ms/step - loss: 0.0872 - accuracy: 0.9712 - val_loss: 6.2644 - val_accuracy: 0.4946
```



```
Epoch 1/100
55/55 [=====] - ETA: 0s - loss: 0.3197 - accuracy: 0.8606
Epoch 1: val_loss improved from inf to 0.50644, saving model to best_model_ResNet101.h5
55/55 [=====] - 119s 1s/step - loss: 0.3197 - accuracy: 0.8606 - val_loss: 0.5064 - val_accuracy: 0.7473
Epoch 2/100
55/55 [=====] - ETA: 0s - loss: 0.1971 - accuracy: 0.9217
Epoch 2: val_loss did not improve from 0.50644
55/55 [=====] - 37s 674ms/step - loss: 0.1971 - accuracy: 0.9217 - val_loss: 1.8342 - val_accuracy: 0.5188
Epoch 3/100
55/55 [=====] - ETA: 0s - loss: 0.1237 - accuracy: 0.9551
Epoch 3: val_loss did not improve from 0.50644
55/55 [=====] - 37s 659ms/step - loss: 0.1237 - accuracy: 0.9551 - val_loss: 1.9892 - val_accuracy: 0.4919
Epoch 4/100
55/55 [=====] - ETA: 0s - loss: 0.1174 - accuracy: 0.9603
Epoch 4: val_loss did not improve from 0.50644
55/55 [=====] - 38s 686ms/step - loss: 0.1174 - accuracy: 0.9603 - val_loss: 0.8140 - val_accuracy: 0.4409
Epoch 5/100
55/55 [=====] - ETA: 0s - loss: 0.0872 - accuracy: 0.9712
Epoch 5: val_loss did not improve from 0.50644
55/55 [=====] - 37s 665ms/step - loss: 0.0872 - accuracy: 0.9712 - val_loss: 6.2644 - val_accuracy: 0.4946
Epoch 6/100
55/55 [=====] - ETA: 0s - loss: 0.0671 - accuracy: 0.9764
Epoch 6: val_loss did not improve from 0.50644
55/55 [=====] - 36s 647ms/step - loss: 0.0671 - accuracy: 0.9764 - val_loss: 8.5598 - val_accuracy: 0.5054
Epoch 7/100
55/55 [=====] - ETA: 0s - loss: 0.0574 - accuracy: 0.9816
Epoch 7: val_loss did not improve from 0.50644
55/55 [=====] - 37s 673ms/step - loss: 0.0574 - accuracy: 0.9816 - val_loss: 4.4893 - val_accuracy: 0.5081
Epoch 8/100
55/55 [=====] - ETA: 0s - loss: 0.0789 - accuracy: 0.9747
Epoch 8: val_loss did not improve from 0.50644
55/55 [=====] - 37s 664ms/step - loss: 0.0789 - accuracy: 0.9747 - val_loss: 19.7425 - val_accuracy: 0.4946
Epoch 9/100
55/55 [=====] - ETA: 0s - loss: 0.0821 - accuracy: 0.9747
Epoch 9: val_loss did not improve from 0.50644
55/55 [=====] - 38s 689ms/step - loss: 0.0821 - accuracy: 0.9747 - val_loss: 39.0286 - val_accuracy: 0.4946
Epoch 10/100
55/55 [=====] - ETA: 0s - loss: 0.0364 - accuracy: 0.9879
Epoch 10: val_loss did not improve from 0.50644
55/55 [=====] - 37s 659ms/step - loss: 0.0364 - accuracy: 0.9879 - val_loss: 33.2096 - val_accuracy: 0.4946
Epoch 11/100
55/55 [=====] - ETA: 0s - loss: 0.0488 - accuracy: 0.9873
Epoch 11: val_loss did not improve from 0.50644
55/55 [=====] - 38s 683ms/step - loss: 0.0488 - accuracy: 0.9873 - val_loss: 197.0154 - val_accuracy: 0.4946
Epoch 11: early stopping
12/12 [=====] - 4s 169ms/step
Model: ResNet101
```

# EXTENDED DESCRIPTION:

Here I am updating:

1. Imports:
  - Import the necessary Python modules and ResNet models for training and prediction.
2. Split the Data:
  - Split the dataset into training, validation, and test sets. ( i.e. 70-20-10 ).
3. Load the pre-trained ResNet model :

- Load the pre-trained ResNet model using the ResNet architectures (ResNet101)

#### . 4. Model Building : -

Modify the ResNet model to suit your binary classification task ( i.e. COVID-19 detection). You may want to add additional fully connected layers on top of the ResNet model for fine-tuning.

- Compile the model using any Deep learning model such as Adam Optimization algorithm.

#### 5. Data Augmentation :

- Apply data augmentation to improve the model's generalization.

- You can use the `ImageDataGenerator` from Keras with various augmentation parameters, such as :

- `width\_shift\_range`

- `height\_shift\_range`

- `rotation\_range`

- `shear\_range`

- `zoom\_range`.

#### 6. Early Stopping and ModelCheckpoint :

- Implement early stopping and model checkpoint callbacks during training.

- Early stopping helps prevent overfitting, and ModelCheckpoint saves the best model during training.

#### 7. Training the model :

- Train the model using the augmented training dataset.

- Compile the model with appropriate loss function and optimizer for binary classification.



#### 8. Model Evaluation :

- Load the ResNet model. - Using the ResNet model predict whether the test image is Covid-19 positive or non-Covid-19.

#### 9. Selected model name : -

print the selected model name .

# GITHUB UPLOADS :



RishabAkula / Capstone-Project

<> Code

Issues

Pull requests

Actions

Projects

Security

Insights

Settings

Files

main

Go to file

Readme

Solution Code

Capstone-Project- / Solution Code

RishabAkula Update of Solution Code

74c3544 - 7 minutes ago History

Code Blame 101 lines (82 loc) · 3.62 KB Code 55% faster with GitHub Copilot

Raw Copy Download Edit View

```
1 import os
2 import random
3 import cv2
4 import numpy as np
5 from sklearn.model_selection import train_test_split
6
7 from tensorflow.keras.applications import ResNet101
8 from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
9 from tensorflow.keras.models import Model
10 from tensorflow.keras.optimizers import Adam
11 from tensorflow.keras.preprocessing.image import ImageDataGenerator
12 from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
13
14 # Define the paths to your data folders
15 covid19_folder = '/content/drive/MyDrive/COVID'
16 non_infected_folder = '/content/drive/MyDrive/non-COVID'
17
18 # Create lists to store the data and labels
19 data = []
20 labels = []
21
22 # Load COVID-19 positive images
23 for filename in os.listdir(covid19_folder):
24     if filename.endswith('.jpg') or filename.endswith('.png'):
25         img = cv2.imread(os.path.join(covid19_folder, filename))
26         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # Ensure images are in RGB format
27         img = cv2.resize(img, (224, 224)) # Resize images to a fixed size
28         data.append(img)
29         labels.append(1) # Label 1 for COVID-19 positive cases
30
31 # Load non-infected images
32 for filename in os.listdir(non_infected_folder):
33     if filename.endswith('.jpg') or filename.endswith('.png'):
34         img = cv2.imread(os.path.join(non_infected_folder, filename))
35         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
36         img = cv2.resize(img, (224, 224))
37         data.append(img)
38         labels.append(0) # Label 0 for non-infected cases
39
```



Files

main

Go to file

Readme

Solution Code

Capstone-Project / Solution Code

Code Blame 101 lines (82 loc) · 3.62 KB Code 55% faster with GitHub Copilot

Raw Copy Download Edit

```
37 # Append labels to the list
38 labels.append(0) # Label 0 for non-infected cases
39
40 # Convert data and labels to NumPy arrays
41 data = np.array(data)
42 labels = np.array(labels)
43
44 # Size of the data array and labels array
45 # print(data.size)
46 # print(labels.size)
47
48 # Split the data into training, validation, and test sets
49 X_train, X_temp, y_train, y_temp = train_test_split(data, labels, test_size=0.3, random_state=42)
50 X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
51
52 # Define a list of ResNet architectures
53 resnet_architectures = [ResNet18, ResNet34, ResNet50, ResNet101, ResNet152]
54 resnet_architectures = [ResNet101]
55
56 for resnet_model in resnet_architectures:
57     # Load pre-trained ResNet model
58     base_model = resnet_model(weights='imagenet', include_top=False)
59
60     # Add custom top layers for binary classification
61     x = base_model.output
62     x = GlobalAveragePooling2D()(x)
63     x = Dense(1024, activation='relu')(x)
64     predictions = Dense(1, activation='sigmoid')(x)
65
66     model = Model(inputs=base_model.input, outputs=predictions)
67
68     # Compile the model
69     model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
70
71     # Create data generators with data augmentation
72     train_datagen = ImageDataGenerator(
73         rescale=1./255,
74         rotation_range=20,
75         width_shift_range=0.2,
76         height_shift_range=0.2,
77         shear_range=0.2,
78         horizontal_flip=True
79     )
80
81     # Set up Early Stopping and Model Checkpoint callbacks
82     early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)
83     model_checkpoint = ModelCheckpoint(f'best_model_{resnet_model.__name__}.h5', save_best_only=True, verbose=1)
84
85     # Train the model
```



Files

main

Go to file

Readme

Solution Code

Capstone-Project / Solution Code

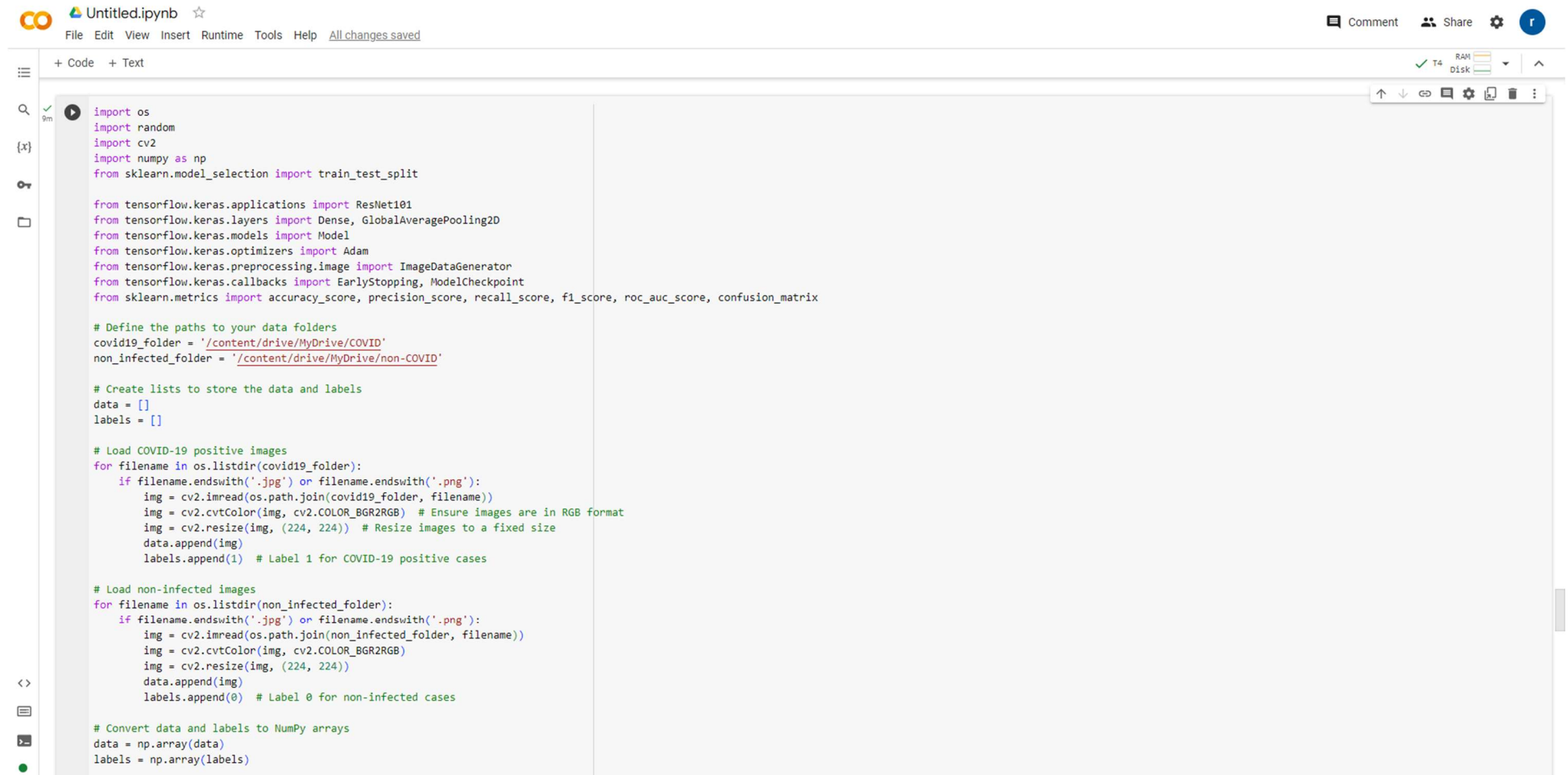
↑ Top

Code Blame 101 lines (82 loc) · 3.62 KB Code 55% faster with GitHub Copilot

Raw Copy Download Edit View

```
56 for resnet_model in resnet_architectures:
57     # Load pre-trained ResNet model
58     base_model = resnet_model(weights='imagenet', include_top=False)
59
60     # Add custom top layers for binary classification
61     x = base_model.output
62     x = GlobalAveragePooling2D()(x)
63     x = Dense(1024, activation='relu')(x)
64     predictions = Dense(1, activation='sigmoid')(x)
65
66     model = Model(inputs=base_model.input, outputs=predictions)
67
68     # Compile the model
69     model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
70
71     # Create data generators with data augmentation
72     train_datagen = ImageDataGenerator(
73         rescale=1./255,
74         rotation_range=20,
75         width_shift_range=0.2,
76         height_shift_range=0.2,
77         shear_range=0.2,
78         horizontal_flip=True
79     )
80
81     # Set up Early Stopping and Model Checkpoint callbacks
82     early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)
83     model_checkpoint = ModelCheckpoint(f'best_model_{resnet_model.__name__}.h5', save_best_only=True, verbose=1)
84
85     # Train the model
86     history = model.fit_generator(
87         train_datagen.flow(X_train, y_train, batch_size=32),
88         validation_data=(X_val, y_val),
89         epochs=100,
90         callbacks=[early_stopping, model_checkpoint]
91     )
92
93     # Load the best model
94     model.load_weights(f'best_model_{resnet_model.__name__}.h5')
95
96     # Evaluate the model on the test set
97     test_predictions = model.predict(X_test)
98     test_predictions = (test_predictions > 0.5).astype(int)
99
100     # Selected Model Name
101     print(f"Model: {resnet_model.__name__}")
```

# FINAL SOLUTION CODE :



The screenshot shows a Jupyter Notebook titled 'Untitled.ipynb' with a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a toolbar. The code is written in a light blue editor and includes imports for os, random, cv2, numpy, and sklearn. It defines paths for COVID-19 and non-infected folders, creates lists for data and labels, loads and processes images, and converts the data to NumPy arrays.

```
import os
import random
import cv2
import numpy as np
from sklearn.model_selection import train_test_split

from tensorflow.keras.applications import ResNet101
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score, confusion_matrix

# Define the paths to your data folders
covid19_folder = '/content/drive/MyDrive/COVID'
non_infected_folder = '/content/drive/MyDrive/non-COVID'

# Create lists to store the data and labels
data = []
labels = []

# Load COVID-19 positive images
for filename in os.listdir(covid19_folder):
    if filename.endswith('.jpg') or filename.endswith('.png'):
        img = cv2.imread(os.path.join(covid19_folder, filename))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # Ensure images are in RGB format
        img = cv2.resize(img, (224, 224)) # Resize images to a fixed size
        data.append(img)
        labels.append(1) # Label 1 for COVID-19 positive cases

# Load non-infected images
for filename in os.listdir(non_infected_folder):
    if filename.endswith('.jpg') or filename.endswith('.png'):
        img = cv2.imread(os.path.join(non_infected_folder, filename))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (224, 224))
        data.append(img)
        labels.append(0) # Label 0 for non-infected cases

# Convert data and labels to NumPy arrays
data = np.array(data)
labels = np.array(labels)
```



+ Code + Text

✓ T4 RAM Disk



```
# Convert data and labels to NumPy arrays
data = np.array(data)
labels = np.array(labels)

#size of the data array and labels array
#print(data.size)
#print(labels.size)

# Split the data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(data, labels, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Define a list of ResNet architectures
#resnet_architectures = [ResNet18, ResNet34, ResNet50, ResNet101, ResNet152]
resnet_architectures = [ResNet101]

for resnet_model in resnet_architectures:
    # Load pre-trained ResNet model
    base_model = resnet_model(weights='imagenet', include_top=False)

    # Add custom top layers for binary classification
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(1, activation='sigmoid')(x)

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

    # Compile the model
    model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])

    # Create data generators with data augmentation
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        horizontal_flip=True
    )

    # Set up Early Stopping and Model Checkpoint callbacks
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)
    model_checkpoint = ModelCheckpoint(f'best_model_{resnet_model.__name__}.h5', save_best_only=True, verbose=1)
```



+ Code + Text

✓ T4 RAM Disk

```
# Set up Early Stopping and Model Checkpoint callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)
model_checkpoint = ModelCheckpoint(f'best_model_{resnet_model.__name__}.h5', save_best_only=True, verbose=1)

# Train the model
history = model.fit_generator(
    train_datagen.flow(X_train, y_train, batch_size=32),
    validation_data=(X_val, y_val),
    epochs=100,
    callbacks=[early_stopping, model_checkpoint]
)

# Load the best model
model.load_weights(f'best_model_{resnet_model.__name__}.h5')

# Evaluate the model on the test set
test_predictions = model.predict(X_test)
test_predictions = (test_predictions > 0.5).astype(int)

# Selected Model Name
#print(f"Model: {resnet_model.__name__}")

# Calculate performance metrics
accuracy = accuracy_score(y_test, test_predictions)
precision = precision_score(y_test, test_predictions)
recall = recall_score(y_test, test_predictions)
f1 = f1_score(y_test, test_predictions)
roc_auc = roc_auc_score(y_test, test_predictions)
cm = confusion_matrix(y_test, test_predictions)

# Print or log the performance metrics for this model
print(f"Model: {resnet_model.__name__}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print(f"ROC AUC: {roc_auc}")
print(f"Confusion Matrix:\n{cm}")

<ipython-input-1-a38ebf77f820>:87: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
history = model.fit_generator(
Epoch 1/100
55/55 [=====] - ETA: 0s - loss: 0.3239 - accuracy: 0.8589
Epoch 1: val_loss improved from inf to 0.76100, saving model to best_model_ResNet101.h5
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Kera
saving_api.save_model(
```



+ Code + Text

✓ T4 RAM Disk

```
print(f"F1 Score: {f1}")
print(f"ROC AUC: {roc_auc}")
print(f"Confusion Matrix:\n{cm}")
```

```
<ipython-input-1-a38ebf77f820>:87: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
history = model.fit_generator(
Epoch 1/100
55/55 [=====] - ETA: 0s - loss: 0.3239 - accuracy: 0.8589
Epoch 1: val_loss improved from inf to 0.76100, saving model to best_model_ResNet101.h5
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Kera
saving_api.save_model(
55/55 [=====] - 128s 880ms/step - loss: 0.3239 - accuracy: 0.8589 - val_loss: 0.7610 - val_accuracy: 0.7151
Epoch 2/100
55/55 [=====] - ETA: 0s - loss: 0.1605 - accuracy: 0.9389
Epoch 2: val_loss did not improve from 0.76100
55/55 [=====] - 37s 667ms/step - loss: 0.1605 - accuracy: 0.9389 - val_loss: 0.7797 - val_accuracy: 0.6828
Epoch 3/100
55/55 [=====] - ETA: 0s - loss: 0.1202 - accuracy: 0.9505
Epoch 3: val_loss did not improve from 0.76100
55/55 [=====] - 37s 669ms/step - loss: 0.1202 - accuracy: 0.9505 - val_loss: 4.9981 - val_accuracy: 0.4946
Epoch 4/100
55/55 [=====] - ETA: 0s - loss: 0.1015 - accuracy: 0.9672
Epoch 4: val_loss did not improve from 0.76100
55/55 [=====] - 37s 654ms/step - loss: 0.1015 - accuracy: 0.9672 - val_loss: 1.0132 - val_accuracy: 0.5457
Epoch 5/100
55/55 [=====] - ETA: 0s - loss: 0.0599 - accuracy: 0.9758
Epoch 5: val_loss did not improve from 0.76100
55/55 [=====] - 39s 706ms/step - loss: 0.0599 - accuracy: 0.9758 - val_loss: 1.1852 - val_accuracy: 0.5457
Epoch 6/100
55/55 [=====] - ETA: 0s - loss: 0.0704 - accuracy: 0.9758
Epoch 6: val_loss did not improve from 0.76100
55/55 [=====] - 37s 677ms/step - loss: 0.0704 - accuracy: 0.9758 - val_loss: 12.4094 - val_accuracy: 0.4946
Epoch 7/100
55/55 [=====] - ETA: 0s - loss: 0.0671 - accuracy: 0.9724
Epoch 7: val_loss did not improve from 0.76100
55/55 [=====] - 36s 653ms/step - loss: 0.0671 - accuracy: 0.9724 - val_loss: 0.9489 - val_accuracy: 0.5887
Epoch 8/100
55/55 [=====] - ETA: 0s - loss: 0.0596 - accuracy: 0.9770
Epoch 8: val_loss did not improve from 0.76100
55/55 [=====] - 36s 648ms/step - loss: 0.0596 - accuracy: 0.9770 - val_loss: 43.4878 - val_accuracy: 0.4946
Epoch 9/100
55/55 [=====] - ETA: 0s - loss: 0.0388 - accuracy: 0.9862
Epoch 9: val_loss did not improve from 0.76100
55/55 [=====] - 38s 685ms/step - loss: 0.0388 - accuracy: 0.9862 - val_loss: 47.9529 - val_accuracy: 0.4946
Epoch 10/100
55/55 [=====] - ETA: 0s - loss: 0.0427 - accuracy: 0.9839
Epoch 10: val_loss did not improve from 0.76100
55/55 [=====] - 39s 698ms/step - loss: 0.0427 - accuracy: 0.9839 - val_loss: 99.0801 - val_accuracy: 0.4946
Epoch 11/100
```



```
Epoch 4: val_loss did not improve from 0.76100
55/55 [=====] - 37s 654ms/step - loss: 0.1015 - accuracy: 0.9672 - val_loss: 1.0132 - val_accuracy: 0.5457
Epoch 5/100
55/55 [=====] - ETA: 0s - loss: 0.0599 - accuracy: 0.9758
Epoch 5: val_loss did not improve from 0.76100
55/55 [=====] - 39s 706ms/step - loss: 0.0599 - accuracy: 0.9758 - val_loss: 1.1852 - val_accuracy: 0.5457
Epoch 6/100
55/55 [=====] - ETA: 0s - loss: 0.0704 - accuracy: 0.9758
Epoch 6: val_loss did not improve from 0.76100
55/55 [=====] - 37s 677ms/step - loss: 0.0704 - accuracy: 0.9758 - val_loss: 12.4094 - val_accuracy: 0.4946
Epoch 7/100
55/55 [=====] - ETA: 0s - loss: 0.0671 - accuracy: 0.9724
Epoch 7: val_loss did not improve from 0.76100
55/55 [=====] - 36s 653ms/step - loss: 0.0671 - accuracy: 0.9724 - val_loss: 0.9489 - val_accuracy: 0.5887
Epoch 8/100
55/55 [=====] - ETA: 0s - loss: 0.0596 - accuracy: 0.9770
Epoch 8: val_loss did not improve from 0.76100
55/55 [=====] - 36s 648ms/step - loss: 0.0596 - accuracy: 0.9770 - val_loss: 43.4878 - val_accuracy: 0.4946
Epoch 9/100
55/55 [=====] - ETA: 0s - loss: 0.0388 - accuracy: 0.9862
Epoch 9: val_loss did not improve from 0.76100
55/55 [=====] - 38s 685ms/step - loss: 0.0388 - accuracy: 0.9862 - val_loss: 47.9529 - val_accuracy: 0.4946
Epoch 10/100
55/55 [=====] - ETA: 0s - loss: 0.0427 - accuracy: 0.9839
Epoch 10: val_loss did not improve from 0.76100
55/55 [=====] - 39s 698ms/step - loss: 0.0427 - accuracy: 0.9839 - val_loss: 99.0801 - val_accuracy: 0.4946
Epoch 11/100
55/55 [=====] - ETA: 0s - loss: 0.0471 - accuracy: 0.9839
Epoch 11: val_loss did not improve from 0.76100
55/55 [=====] - 38s 690ms/step - loss: 0.0471 - accuracy: 0.9839 - val_loss: 80.1162 - val_accuracy: 0.4946
Epoch 11: early stopping
12/12 [=====] - 5s 166ms/step
Model: ResNet101
Accuracy: 0.7553763440860215
Precision: 0.9894736842105263
Recall: 0.5108695652173914
F1 Score: 0.6738351254480287
ROC AUC: 0.7527752081406106
Confusion Matrix:
[[187  1]
 [ 90 94]]
```

## EXTENDED DESCRIPTION :

1. import :

- imported necessary metrics for Accuracy, Precision, Confusion matrix and so on.

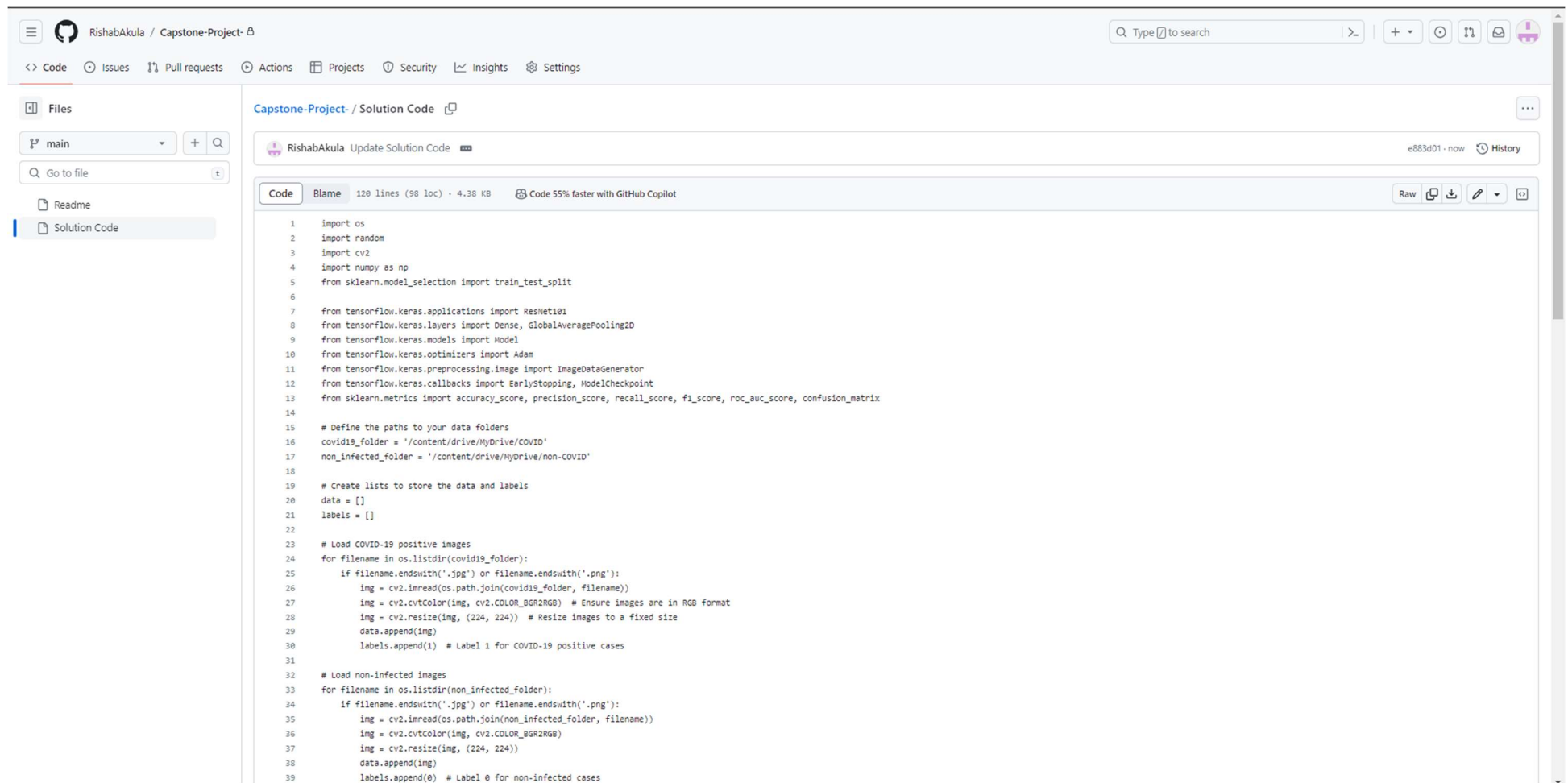
2. Performance Metrics :

- Calculated various performance metrics to assess the model's performance from scikit-learn library , such as:

- Accuracy.

- Precision.
- Recall.
- F1-score.
- ROC curve and AUC (Area Under the Curve).
- Confusion matrix.

## GITHUB UPLOADS :



The screenshot displays the GitHub interface for the 'Capstone-Project' repository. The left sidebar shows the repository structure with 'main' branch selected and 'Solution Code' file highlighted. The main content area shows the 'Solution Code' file, which is a Python script for image classification. The script includes imports for os, random, cv2, numpy, sklearn, tensorflow.keras, and tensorflow.keras.callbacks. It defines paths for COVID-19 and non-infected folders, creates lists for data and labels, and loads images from both folders, resizing them to 224x224 pixels.

```
1 import os
2 import random
3 import cv2
4 import numpy as np
5 from sklearn.model_selection import train_test_split
6
7 from tensorflow.keras.applications import ResNet101
8 from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
9 from tensorflow.keras.models import Model
10 from tensorflow.keras.optimizers import Adam
11 from tensorflow.keras.preprocessing.image import ImageDataGenerator
12 from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
13 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix
14
15 # Define the paths to your data folders
16 covid19_folder = '/content/drive/MyDrive/COVID'
17 non_infected_folder = '/content/drive/MyDrive/non-COVID'
18
19 # Create lists to store the data and labels
20 data = []
21 labels = []
22
23 # Load COVID-19 positive images
24 for filename in os.listdir(covid19_folder):
25     if filename.endswith('.jpg') or filename.endswith('.png'):
26         img = cv2.imread(os.path.join(covid19_folder, filename))
27         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # Ensure images are in RGB format
28         img = cv2.resize(img, (224, 224)) # Resize images to a fixed size
29         data.append(img)
30         labels.append(1) # Label 1 for COVID-19 positive cases
31
32 # Load non-infected images
33 for filename in os.listdir(non_infected_folder):
34     if filename.endswith('.jpg') or filename.endswith('.png'):
35         img = cv2.imread(os.path.join(non_infected_folder, filename))
36         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
37         img = cv2.resize(img, (224, 224))
38         data.append(img)
39         labels.append(0) # Label 0 for non-infected cases
```



Files

main

Go to file

Readme

Solution Code

Capstone-Project / Solution Code

Code Blame 120 lines (98 loc) · 4.38 KB Code 55% faster with GitHub Copilot

Raw Copy Download Edit

```
38 data.append(img)
39 labels.append(0) # Label 0 for non-infected cases
40
41 # Convert data and labels to NumPy arrays
42 data = np.array(data)
43 labels = np.array(labels)
44
45 #size of the data array and labels array
46 #print(data.size)
47 #print(labels.size)
48
49 # Split the data into training, validation, and test sets
50 X_train, X_temp, y_train, y_temp = train_test_split(data, labels, test_size=0.3, random_state=42)
51 X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
52
53 # Define a list of ResNet architectures
54 #resnet_architectures = [ResNet18, ResNet34, ResNet50, ResNet101, ResNet152]
55 resnet_architectures = [ResNet101]
56
57 for resnet_model in resnet_architectures:
58     # Load pre-trained ResNet model
59     base_model = resnet_model(weights='imagenet', include_top=False)
60
61     # Add custom top layers for binary classification
62     x = base_model.output
63     x = GlobalAveragePooling2D()(x)
64     x = Dense(1024, activation='relu')(x)
65     predictions = Dense(1, activation='sigmoid')(x)
66
67     model = Model(inputs=base_model.input, outputs=predictions)
68
69     # Compile the model
70     model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
71
72     # Create data generators with data augmentation
73     train_datagen = ImageDataGenerator(
74         rescale=1./255,
75         rotation_range=20,
76         width_shift_range=0.2,
77         height_shift_range=0.2,
78         shear_range=0.2,
79         horizontal_flip=True
80     )
81
82     # Set up Early Stopping and Model Checkpoint callbacks
83     early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)
84     model_checkpoint = ModelCheckpoint(f'best_model_{resnet_model.__name__}.h5', save_best_only=True, verbose=1)
85
```

Files

main

Go to file

Readme

Solution Code

Capstone-Project / Solution Code

↑ Top

Code Blame 120 lines (98 loc) · 4.38 KB Code 55% faster with GitHub Copilot

```
73 train_generator = ImageDataGenerator(
74     rescale=1./255,
75     rotation_range=20,
76     width_shift_range=0.2,
77     height_shift_range=0.2,
78     shear_range=0.2,
79     horizontal_flip=True
80 )
81
82 # Set up Early Stopping and Model Checkpoint callbacks
83 early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)
84 model_checkpoint = ModelCheckpoint(f'best_model_{resnet_model.__name__}.h5', save_best_only=True, verbose=1)
85
86 # Train the model
87 history = model.fit_generator(
88     train_datagen.flow(X_train, y_train, batch_size=32),
89     validation_data=(X_val, y_val),
90     epochs=100,
91     callbacks=[early_stopping, model_checkpoint]
92 )
93
94 # Load the best model
95 model.load_weights(f'best_model_{resnet_model.__name__}.h5')
96
97 # Evaluate the model on the test set
98 test_predictions = model.predict(X_test)
99 test_predictions = (test_predictions > 0.5).astype(int)
100
101 # Selected Model Name
102 #print(f"Model: {resnet_model.__name__}")
103
104 # Calculate performance metrics
105 accuracy = accuracy_score(y_test, test_predictions)
106 precision = precision_score(y_test, test_predictions)
107 recall = recall_score(y_test, test_predictions)
108 f1 = f1_score(y_test, test_predictions)
109 roc_auc = roc_auc_score(y_test, test_predictions)
110 cm = confusion_matrix(y_test, test_predictions)
111
112 # Print or log the performance metrics for this model
113 print(f"Model: {resnet_model.__name__}")
114 print(f"Accuracy: {accuracy}")
115 print(f"Precision: {precision}")
116 print(f"Recall: {recall}")
117 print(f"F1 Score: {f1}")
118 print(f"ROC AUC: {roc_auc}")
119 print(f"Confusion Matrix:\n{cm}")
```