```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

```

train_folder = '/content/drive/MyDrive/Data/train'
test_folder = '/content/drive/MyDrive/Data/test'
validate_folder = '/content/drive/MyDrive/Data/valid'

normal_folder = '/normal'
adenocarcinoma_folder = '/adenocarcinoma_left.lower.lobe_T2_N0_M0_Ib'
large_cell_carcinoma_folder = '/large.cell.carcinoma_left.hilum_T2_N2_M0_IIIa'
squamous_cell_carcinoma_folder = '/squamous.cell.carcinoma_left.hilum_T1_N2_M0_IIIa'

```

```
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder

import tensorflow as tf
import tensorflow.keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, SpatialDropout2D, Activation, Lambda, Flatten, LSTM
from tensorflow.keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras import utils

print("Libraries Imported")

# Read data from the folders
IMAGE_SIZE = (350, 350)

print("Reading training images from:", train_folder)
print("Reading validation images from:", validate_folder)

train_datagen = ImageDataGenerator(rescale=1./255, horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

batch_size = 8

train_generator = train_datagen.flow_from_directory(
    train_folder,
    target_size=IMAGE_SIZE,
    batch_size=batch_size,
    color_mode="rgb",
    class_mode='categorical'
)

validation_generator = test_datagen.flow_from_directory(
    test_folder,
    target_size=IMAGE_SIZE,
    batch_size=batch_size,
    color_mode="rgb",
    class_mode='categorical'
)

```

```
Libraries Imported
Reading training images from: /content/drive/MyDrive/Data/train
Reading validation images from: /content/drive/MyDrive/Data/valid
Found 613 images belonging to 4 classes.
Found 323 images belonging to 4 classes.
```

```
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint

learning_rate_reduction = ReduceLROnPlateau(monitor='loss', patience=5, verbose=2, factor=0.5, min_lr=0.000001)
```

```
early_stops = EarlyStopping(monitor='loss', min_delta=0, patience=6, verbose=2, mode='auto')
checkpointer = ModelCheckpoint(filepath='best_model.weights.h5', verbose=2, save_best_only=True, save_weights_only=True)
```

```
OUTPUT_SIZE = 4

pretrained_model = tf.keras.applications.Xception(weights='imagenet', include_top=False, input_shape=[*IMAGE_SIZE, 3])
pretrained_model.trainable = False

model = Sequential()
model.add(pretrained_model)
model.add(GlobalAveragePooling2D())
model.add(Dense(OUTPUT_SIZE, activation='softmax'))

# print("Pretrained model used:")
# pretrained_model.summary()

# print("Final model created:")
# model.summary()

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

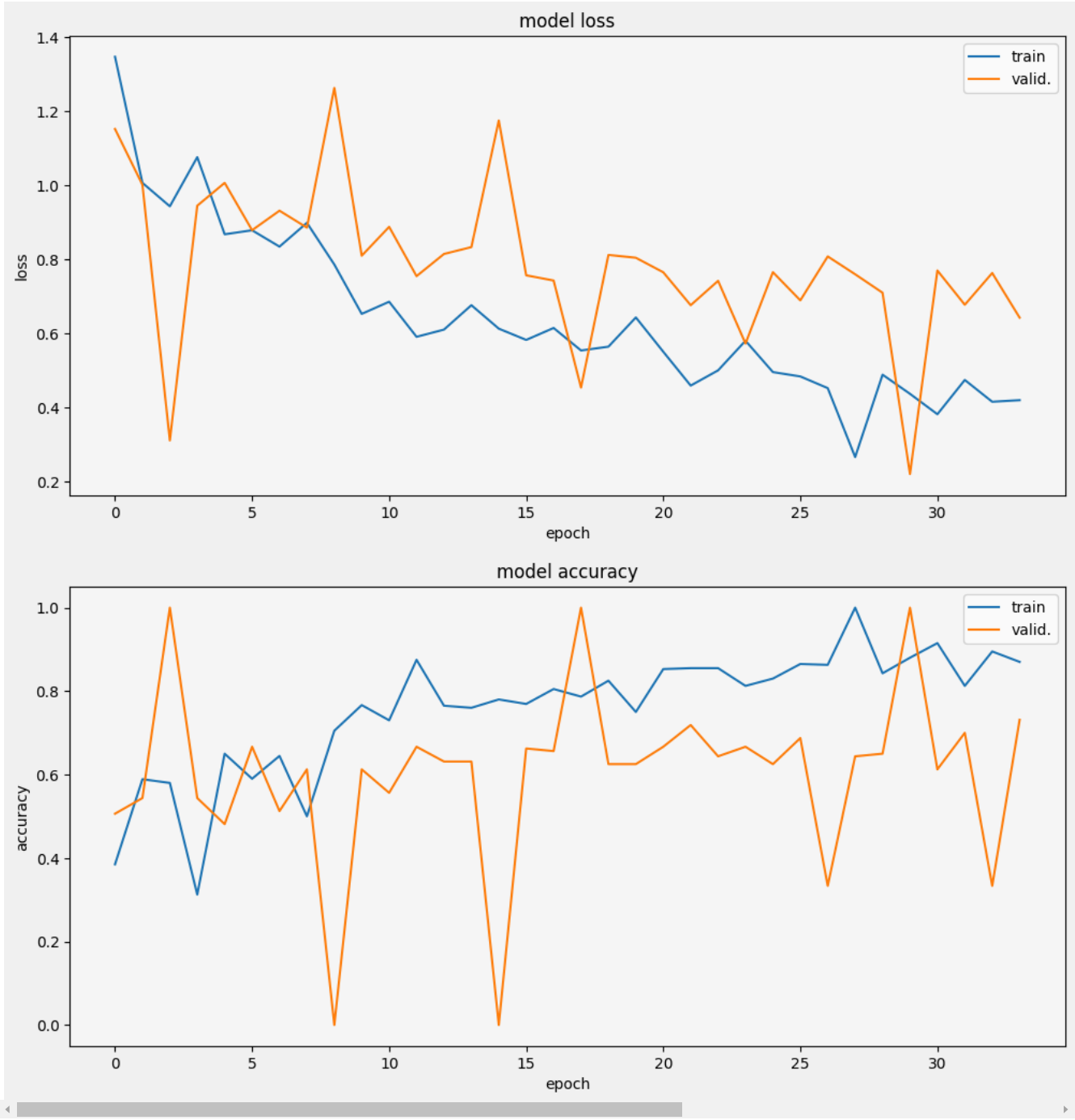
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kern
83683744/83683744 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
history = model.fit(
    train_generator,
    steps_per_epoch=25,
    epochs=50,
    callbacks=[learning_rate_reduction, early_stops, checkpointer],
    validation_data=validation_generator,
    validation_steps=20
)

print("Final training accuracy =", history.history['accuracy'][-1])
print("Final testing accuracy =", history.history['val_accuracy'][-1])
```

```
Epoch 1/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.3839 - loss: 1.3234
Epoch 1: val_loss improved from inf to 1.13811, saving model to best_model.weights.h5
25/25 ━━━━━━━━━━━━━━━━━━━━ 239s 9s/step - accuracy: 0.3850 - loss: 1.3217 - val_accuracy: 0.4250 - val_loss: 1.1381 - learning_ra
Epoch 2/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.5657 - loss: 1.0422
Epoch 2: val_loss improved from 1.13811 to 1.07243, saving model to best_model.weights.h5
25/25 ━━━━━━━━━━━━━━━━━━━━ 213s 9s/step - accuracy: 0.5654 - loss: 1.0415 - val_accuracy: 0.5250 - val_loss: 1.0724 - learning_ra
Epoch 3/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.5468 - loss: 1.0176
Epoch 3: val_loss improved from 1.07243 to 0.75630, saving model to best_model.weights.h5
25/25 ━━━━━━━━━━━━━━━━━━━━ 129s 5s/step - accuracy: 0.5486 - loss: 1.0134 - val_accuracy: 1.0000 - val_loss: 0.7563 - learning_ra
Epoch 4/50
 2/25 ━━━━━━━━━━━━━━━━━━━━ 2:11 6s/step - accuracy: 0.6250 - loss: 0.8424
Epoch 4: val_loss did not improve from 0.75630
25/25 ━━━━━━━━━━━━━━━━━━━━ 105s 4s/step - accuracy: 0.6250 - loss: 0.8450 - val_accuracy: 0.6062 - val_loss: 0.9247 - learning_ra
Epoch 5/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.6183 - loss: 0.8222
Epoch 5: val_loss did not improve from 0.75630
25/25 ━━━━━━━━━━━━━━━━━━━━ 222s 9s/step - accuracy: 0.6179 - loss: 0.8227 - val_accuracy: 0.6187 - val_loss: 0.9042 - learning_ra
Epoch 6/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.5736 - loss: 0.8768
Epoch 6: val_loss did not improve from 0.75630
25/25 ━━━━━━━━━━━━━━━━━━━━ 125s 5s/step - accuracy: 0.5754 - loss: 0.8750 - val_accuracy: 0.0000e+00 - val_loss: 1.4956 - learnin
Epoch 7/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.6852 - loss: 0.7895
Epoch 7: val_loss did not improve from 0.75630
25/25 ━━━━━━━━━━━━━━━━━━━━ 213s 9s/step - accuracy: 0.6867 - loss: 0.7886 - val_accuracy: 0.5375 - val_loss: 0.8901 - learning_ra
Epoch 8/50
 2/25 ━━━━━━━━━━━━━━━━━━━━ 2:22 6s/step - accuracy: 0.7500 - loss: 0.8337
Epoch 8: val_loss did not improve from 0.75630
25/25 ━━━━━━━━━━━━━━━━━━━━ 105s 4s/step - accuracy: 0.6350 - loss: 0.8446 - val_accuracy: 0.5688 - val_loss: 0.8368 - learning_ra
Epoch 9/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.7132 - loss: 0.6986
Epoch 9: val_loss improved from 0.75630 to 0.20142, saving model to best_model.weights.h5
25/25 ━━━━━━━━━━━━━━━━━━━━ 143s 5s/step - accuracy: 0.7118 - loss: 0.7002 - val_accuracy: 1.0000 - val_loss: 0.2014 - learning_ra
Epoch 10/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.7334 - loss: 0.6156
Epoch 10: val_loss did not improve from 0.20142
25/25 ━━━━━━━━━━━━━━━━━━━━ 212s 9s/step - accuracy: 0.7338 - loss: 0.6153 - val_accuracy: 0.6062 - val_loss: 0.8740 - learning_ra
Epoch 11/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.7511 - loss: 0.6805
Epoch 11: val_loss did not improve from 0.20142
25/25 ━━━━━━━━━━━━━━━━━━━━ 210s 9s/step - accuracy: 0.7512 - loss: 0.6793 - val_accuracy: 0.5312 - val_loss: 0.8676 - learning_ra
```

```
Epoch 12/50
 2/25 ━━━━━━━━━━━━━━━━━━━━ 1:29 4s/step - accuracy: 0.6875 - loss: 0.6441
Epoch 12: val_loss did not improve from 0.20142
25/25 ━━━━━━━━━━━━━━━━━━━━ 12s 223ms/step - accuracy: 0.6300 - loss: 0.7127 - val_accuracy: 1.0000 - val_loss: 0.6788 - learning_
Epoch 13/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.7515 - loss: 0.6438
Epoch 13: val_loss did not improve from 0.20142
25/25 ━━━━━━━━━━━━━━━━━━━━ 220s 9s/step - accuracy: 0.7526 - loss: 0.6439 - val_accuracy: 0.6000 - val_loss: 0.8278 - learning_ra
Epoch 14/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.8311 - loss: 0.5594
Epoch 14: val_loss did not improve from 0.20142
25/25 ━━━━━━━━━━━━━━━━━━━━ 212s 9s/step - accuracy: 0.8291 - loss: 0.5607 - val_accuracy: 0.6313 - val_loss: 0.8253 - learning_ra
Epoch 15/50
```

```
def display_training_curves(training, validation, title, subplot):
    if subplot % 10 == 1:
        plt.subplots(figsize=(10, 10), facecolor='#F0F0F0')
        plt.tight_layout()
    ax = plt.subplot(subplot)
    ax.set_facecolor('#F8F8F8')
    ax.plot(training)
    ax.plot(validation)
    ax.set_title('model ' + title)
    ax.set_ylabel(title)
    ax.set_xlabel('epoch')
    ax.legend(['train', 'valid.'])

display_training_curves(history.history['loss'], history.history['val_loss'], 'loss', 211)
display_training_curves(history.history['accuracy'], history.history['val_accuracy'], 'accuracy', 212)

```




```
model.save('/content/drive/MyDrive/Data/trained_lung_cancer_model.h5')

```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

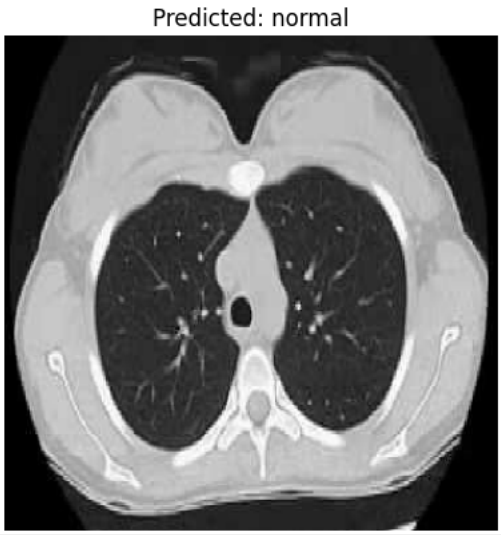
```
model.save('trained_lung_cancer_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

```
from tensorflow.keras.preprocessing import image
import numpy as np

# Define a function to load and preprocess the image
def load_and_preprocess_image(img_path, target_size=(350,350)):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0  # Rescale the image like the training images
    return img_array

# Load an image from your drive
img_path = '/content/drive/MyDrive/Data/train/normal/10.png'
img = load_and_preprocess_image(img_path)
```

```
15
16 # Make a prediction
17 predictions = model.predict(img)
18 predicted_class = np.argmax(predictions[0])
19
20 # Map the predicted class to the class label
21 class_labels = list(train_generator.class_indices.keys())
22 predicted_label = class_labels[predicted_class]
23
24 # Print the predicted class
25 print(f"The image belongs to class: {predicted_label}")
26
27 # Display the image
28 plt.imshow(image.load_img(img_path, target_size=IMAGE_SIZE))
29 plt.title(f"Predicted: {predicted_label}")
30 plt.axis('off')
31 plt.show()
32
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
The image belongs to class: normal
```

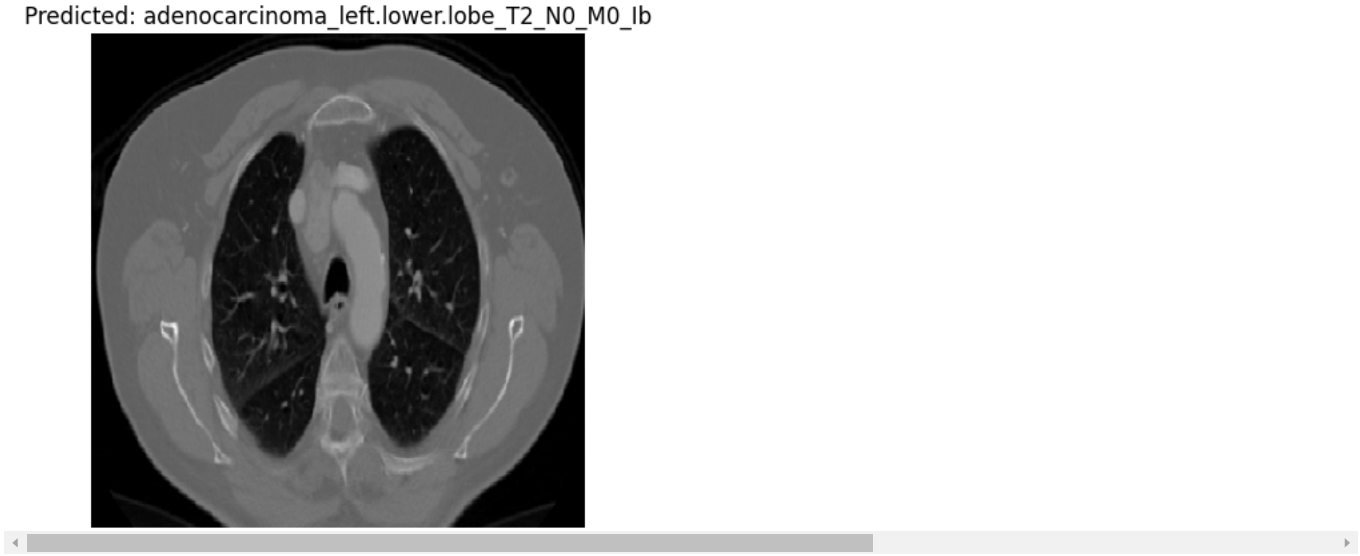



```
1 from tensorflow.keras.preprocessing import image
2 import numpy as np
3
4 # Define a function to load and preprocess the image
5 def load_and_preprocess_image(img_path, target_size=(350,350)):
6     img = image.load_img(img_path, target_size=target_size)
7     img_array = image.img_to_array(img)
8     img_array = np.expand_dims(img_array, axis=0)
9     img_array /= 255.0  # Rescale the image like the training images
10    return img_array
11
12 # Load an image from your drive
13 img_path = '/content/drive/MyDrive/Data/train/large.cell.carcinoma_left.hilum_T2_N2_M0_IIIa/000016 (4).png'
14 img = load_and_preprocess_image(img_path)
15
16 # Make a prediction
17 predictions = model.predict(img)
18 predicted_class = np.argmax(predictions[0])
19
20 # Map the predicted class to the class label
21 class_labels = list(train_generator.class_indices.keys())
22 predicted_label = class_labels[predicted_class]
23
24 # Print the predicted class
25 print(f"The image belongs to class: {predicted_label}")
26
27 # Display the image
28 plt.imshow(image.load_img(img_path, target_size=IMAGE_SIZE))
29 plt.title(f"Predicted: {predicted_label}")
30 plt.axis('off')
31 plt.show()
32
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
The image belongs to class: large.cell.carcinoma_left.hilum_T2_N2_M0_IIIa
```

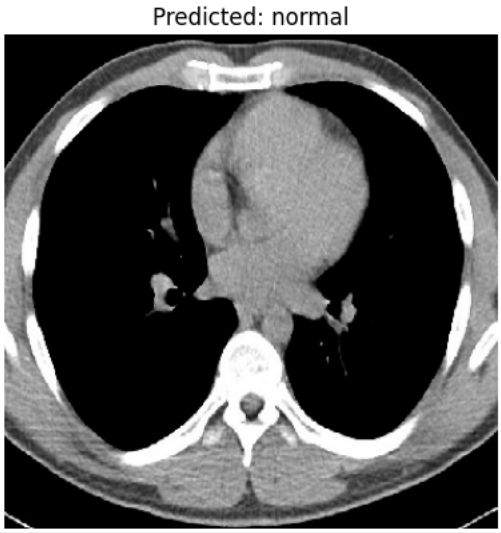

```
from tensorflow.keras.preprocessing import image
import numpy as np

# Define a function to load and preprocess the image
def load_and_preprocess_image(img_path, target_size=(350,350)):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0  # Rescale the image like the training images
    return img_array

# Load an image from your drive
img_path = '/content/drive/MyDrive/Data/train/adenocarcinoma_left.lower.lobe_T2_N0_M0_Ib/000000 (6).png'
img = load_and_preprocess_image(img_path)

# Make a prediction
predictions = model.predict(img)
predicted_class = np.argmax(predictions[0])

# Map the predicted class to the class label
class_labels = list(train_generator.class_indices.keys())
predicted_label = class_labels[predicted_class]

# Print the predicted class
print(f"The image belongs to class: {predicted_label}")

# Display the image
plt.imshow(image.load_img(img_path, target_size=IMAGE_SIZE))
plt.title(f"Predicted: {predicted_label}")
plt.axis('off')
plt.show()

```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 787ms/step
The image belongs to class: adenocarcinoma_left.lower.lobe_T2_N0_M0_Ib
```



```
from tensorflow.keras.preprocessing import image
import numpy as np

# Define a function to load and preprocess the image
def load_and_preprocess_image(img_path, target_size=(350,350)):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0  # Rescale the image like the training images
    return img_array

# Load an image from your drive
img_path = '/content/drive/MyDrive/Data/valid/normal/4 (2).png'
img = load_and_preprocess_image(img_path)

# Make a prediction
predictions = model.predict(img)
predicted_class = np.argmax(predictions[0])

# Map the predicted class to the class label
class_labels = list(train_generator.class_indices.keys())
predicted_label = class_labels[predicted_class]

# Print the predicted class
print(f"The image belongs to class: {predicted_label}")

# Display the image
plt.imshow(image.load_img(img_path, target_size=IMAGE_SIZE))
plt.title(f"Predicted: {predicted_label}")
plt.axis('off')
plt.show()

```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 772ms/step
The image belongs to class: normal
```




```
from tensorflow.keras.preprocessing import image
import numpy as np

# Define a function to load and preprocess the image
def load_and_preprocess_image(img_path, target_size=(350,350)):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0  # Rescale the image like the training images
    return img_array

# Load an image from your drive
img_path = '/content/drive/MyDrive/Data/valid/squamous.cell.carcinoma_left.hilum_T1_N2_M0_IIIa/000114 (4).png'
img = load_and_preprocess_image(img_path)

# Make a prediction
predictions = model.predict(img)
predicted_class = np.argmax(predictions[0])

# Map the predicted class to the class label
class_labels = list(train_generator.class_indices.keys())
predicted_label = class_labels[predicted_class]

```

```
# Print the predicted class
print(f"The image belongs to class: {predicted_label}")

# Display the image
plt.imshow(image.load_img(img_path, target_size=IMAGE_SIZE))
plt.title(f"Predicted: {predicted_label}")
plt.axis('off')
plt.show()

```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 429ms/step
The image belongs to class: squamous.cell.carcinoma_left.hilum_T1_N2_M0_IIIa
```


```
# prompt: write this all cell code fo running in jupiter note book

import warnings
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
import tensorflow.keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, SpatialDropout2D, Activation, Lambda, Flatten, LSTM
from tensorflow.keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras import utils
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
from tensorflow.keras.preprocessing import image

# Replace with your actual paths
train_folder = 'Data/train'
test_folder = 'Data/test'
validate_folder = 'Data/valid'

normal_folder = '/normal'
adenocarcinoma_folder = '/adenocarcinoma_left.lower.lobe_T2_N0_M0_Ib'
large_cell_carcinoma_folder = '/large.cell.carcinoma_left.hilum_T2_N2_M0_IIIa'
squamous_cell_carcinoma_folder = '/squamous.cell.carcinoma_left.hilum_T1_N2_M0_IIIa'

warnings.filterwarnings('ignore')

print("Libraries Imported")

# Read data from the folders
IMAGE_SIZE = (350, 350)

print("Reading training images from:", train_folder)
print("Reading validation images from:", validate_folder)

train_datagen = ImageDataGenerator(rescale=1./255, horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)
```

```
48
49 batch_size = 8
50
51 train_generator = train_datagen.flow_from_directory(
52     train_folder,
53     target_size=IMAGE_SIZE,
54     batch_size=batch_size,
55     color_mode="rgb",
56     class_mode='categorical'
57 )
58
59 validation_generator = test_datagen.flow_from_directory(
60     test_folder,
61     target_size=IMAGE_SIZE,
62     batch_size=batch_size,
63     color_mode="rgb",
64     class_mode='categorical'
65 )
66
67
68 learning_rate_reduction = ReduceLROnPlateau(monitor='loss', patience=5, verbose=2, factor=0.5, min_lr=0.000001)
69 early_stops = EarlyStopping(monitor='loss', min_delta=0, patience=6, verbose=2, mode='auto')
70 checkpointer = ModelCheckpoint(filepath='best_model.weights.h5', verbose=2, save_best_only=True, save_weights_only=True
71
72 OUTPUT_SIZE = 4
73
74 pretrained_model = tf.keras.applications.Xception(weights='imagenet', include_top=False, input_shape=[*IMAGE_SIZE, 3])
75 pretrained_model.trainable = False
76
77 model = Sequential()
78 model.add(pretrained_model)
79 model.add(GlobalAveragePooling2D())
80 model.add(Dense(OUTPUT_SIZE, activation='softmax'))
81
82 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
83
84 history = model.fit(
```