```
!kaggle datasets download manasvi12/skin-cancer-isic-2020-segmented-both
```

```
Dataset URL: https://www.kaggle.com/datasets/manasvi12/skin-cancer-isic-2020-segmented-both
License(s): unknown
skin-cancer-isic-2020-segmented-both.zip: Skipping, found more recently modified local copy (use --force to force download)
```

```
import zipfile
zip_ref = zipfile.ZipFile('/content/skin-cancer-isic-2020-segmented-both.zip', 'r')
zip_ref.extractall('/content')
zip_ref.close()
```

```
import os
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.regularizers import l2
```

```
from PIL import Image
import cv2

# Paths to train and test directories
TRAIN_ORIGINAL_DIR = '/content/original images/Skin cancer ISIC The International Skin Imaging Collaboration/Train'
TRAIN_SEGMENTED_DIR = '/content/segmented images/segmented/Train'
TEST_ORIGINAL_DIR = '/content/original images/Skin cancer ISIC The International Skin Imaging Collaboration/Test'
TEST_SEGMENTED_DIR = '/content/segmented images/segmented/Test'

IMG_SIZE = (224, 224)  # Resize images to 224x224
CLASS_NAMES = sorted(os.listdir(TRAIN_ORIGINAL_DIR))  # Get class names from folder structure
NUM_CLASSES = len(CLASS_NAMES)

# Mapping class names to numeric labels
class_to_label = {class_name: idx for idx, class_name in enumerate(CLASS_NAMES)}
label_to_class = {idx: class_name for class_name, idx in class_to_label.items()}
```

```
import os
import cv2
import numpy as np

def load_combined_images(original_dir, segmented_dir, img_size=(224, 224)):
    images = []
    labels = []
    for class_name in os.listdir(original_dir):
        orig_class_path = os.path.join(original_dir, class_name)
        seg_class_path = os.path.join(segmented_dir, class_name)

        label = class_to_label[class_name]

        for img_name in os.listdir(orig_class_path):
            try:
                # Handle different formats for original and segmented images
                orig_img_path = os.path.join(orig_class_path, img_name)

                # Check for corresponding segmented image (e.g., with .tiff extension)
                base_name = os.path.splitext(img_name)[0]  # Remove extension
                seg_img_path = os.path.join(seg_class_path, f"{base_name}.tiff")

                # Load and resize original image
                orig_img = cv2.imread(orig_img_path)
                if orig_img is None:
                    raise FileNotFoundError(f"Original image not found: {orig_img_path}")
                orig_img = cv2.resize(orig_img, img_size)
                orig_img = orig_img / 255.0
```

```
                # Load and resize segmented image
                seg_img = cv2.imread(seg_img_path, cv2.IMREAD_GRAYSCALE)
                if seg_img is None:
                    raise FileNotFoundError(f"Segmented image not found: {seg_img_path}")
                seg_img = cv2.resize(seg_img, img_size)
                seg_img = seg_img / 255.0
                seg_img = np.expand_dims(seg_img, axis=-1)  # Add channel dimension

                # Combine original and segmented images
                combined_img = np.concatenate((orig_img, seg_img), axis=-1)

                images.append(combined_img)
                labels.append(label)
            except Exception as e:
                print(f"Error loading image {img_name}: {e}")

    return np.array(images), np.array(labels)



# Load training and testing data
X_train, y_train = load_combined_images(TRAIN_ORIGINAL_DIR, TRAIN_SEGMENTED_DIR)
X_test, y_test = load_combined_images(TEST_ORIGINAL_DIR, TEST_SEGMENTED_DIR)

# Convert labels to one-hot encoding
y_train = to_categorical(y_train, NUM_CLASSES)
y_test = to_categorical(y_test, NUM_CLASSES)

# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.3,
    height_shift_range=0.3,
    zoom_range=0.3,
    shear_range=0.2,
    horizontal_flip=True,
    vertical_flip=False,
    brightness_range=[0.8, 1.2]
)
datagen.fit(X_train)




model = Sequential([
    # First Convolutional Block
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE[0], IMG_SIZE[1], 4)),
    MaxPooling2D((2, 2)),

    # Second Convolutional Block
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    # Third Convolutional Block
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    # Fully Connected Layers
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),  # Higher dropout for regularization
    Dense(NUM_CLASSES, activation='softmax')
])
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Train the model
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6)
```

```
model.fit(X_train,y_train, epochs=50, callbacks=[early_stopping, reduce_lr])
```

```
Epoch 1/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 17s 137ms/step - accuracy: 0.2796 - loss: 1.9473 - learning_rate: 1.0000e-04
Epoch 2/50
 1/70 ━━━━━━━━━━━━━━━━━━━━ 8s 125ms/step - accuracy: 0.3750 - loss: 1.6184/usr/local/lib/python3.10/dist-packages/keras/src/callbacks
  current = self.get_monitor_value(logs)
/usr/local/lib/python3.10/dist-packages/keras/src/callbacks/callback_list.py:96: UserWarning: Learning rate reduction is conditioned
  callback.on_epoch_end(epoch, logs)
70/70 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.5091 - loss: 1.3479 - learning_rate: 1.0000e-04
Epoch 3/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 53ms/step - accuracy: 0.6162 - loss: 1.1135 - learning_rate: 1.0000e-04
Epoch 4/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - accuracy: 0.6852 - loss: 0.9477 - learning_rate: 1.0000e-04
Epoch 5/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 56ms/step - accuracy: 0.7103 - loss: 0.8408 - learning_rate: 1.0000e-04
Epoch 6/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - accuracy: 0.7813 - loss: 0.6971 - learning_rate: 1.0000e-04
Epoch 7/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 56ms/step - accuracy: 0.7695 - loss: 0.6967 - learning_rate: 1.0000e-04
Epoch 8/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 4s 55ms/step - accuracy: 0.8141 - loss: 0.5717 - learning_rate: 1.0000e-04
Epoch 9/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 4s 54ms/step - accuracy: 0.8340 - loss: 0.4879 - learning_rate: 1.0000e-04
Epoch 10/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 56ms/step - accuracy: 0.8601 - loss: 0.4219 - learning_rate: 1.0000e-04
Epoch 11/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 4s 55ms/step - accuracy: 0.8782 - loss: 0.3883 - learning_rate: 1.0000e-04
Epoch 12/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 53ms/step - accuracy: 0.8712 - loss: 0.3626 - learning_rate: 1.0000e-04
Epoch 13/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 4s 55ms/step - accuracy: 0.8882 - loss: 0.3275 - learning_rate: 1.0000e-04
Epoch 14/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - accuracy: 0.8952 - loss: 0.3157 - learning_rate: 1.0000e-04
Epoch 15/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - accuracy: 0.8992 - loss: 0.2907 - learning_rate: 1.0000e-04
Epoch 16/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 55ms/step - accuracy: 0.9049 - loss: 0.2585 - learning_rate: 1.0000e-04
Epoch 17/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 53ms/step - accuracy: 0.9249 - loss: 0.2115 - learning_rate: 1.0000e-04
Epoch 18/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 4s 55ms/step - accuracy: 0.9353 - loss: 0.1858 - learning_rate: 1.0000e-04
Epoch 19/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - accuracy: 0.9460 - loss: 0.1652 - learning_rate: 1.0000e-04
Epoch 20/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 4s 54ms/step - accuracy: 0.9510 - loss: 0.1674 - learning_rate: 1.0000e-04
Epoch 21/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 4s 55ms/step - accuracy: 0.9564 - loss: 0.1477 - learning_rate: 1.0000e-04
Epoch 22/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - accuracy: 0.9438 - loss: 0.1726 - learning_rate: 1.0000e-04
Epoch 23/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - accuracy: 0.9569 - loss: 0.1397 - learning_rate: 1.0000e-04
Epoch 24/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.9632 - loss: 0.1248 - learning_rate: 1.0000e-04
Epoch 25/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 5s 55ms/step - accuracy: 0.9701 - loss: 0.1146 - learning_rate: 1.0000e-04
Epoch 26/50
70/70 ━━━━━━━━━━━━━━━━━━━━ 4s 55ms/step - accuracy: 0.9645 - loss: 0.1129 - learning_rate: 1.0000e-04
Epoch 27/50
```

```
# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 432ms/step - accuracy: 0.5115 - loss: 5.8334
Test Accuracy: 53.39%
```

```
# Classification report
y_pred = np.argmax(model.predict(X_test), axis=1)
y_test_labels = np.argmax(y_test, axis=1)
print(classification_report(y_test_labels, y_pred, target_names=CLASS_NAMES))
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 88ms/step
                          precision    recall  f1-score   support

       actinic keratosis       0.57      0.25      0.35        16
    basal cell carcinoma       0.67      0.88      0.76        16
```

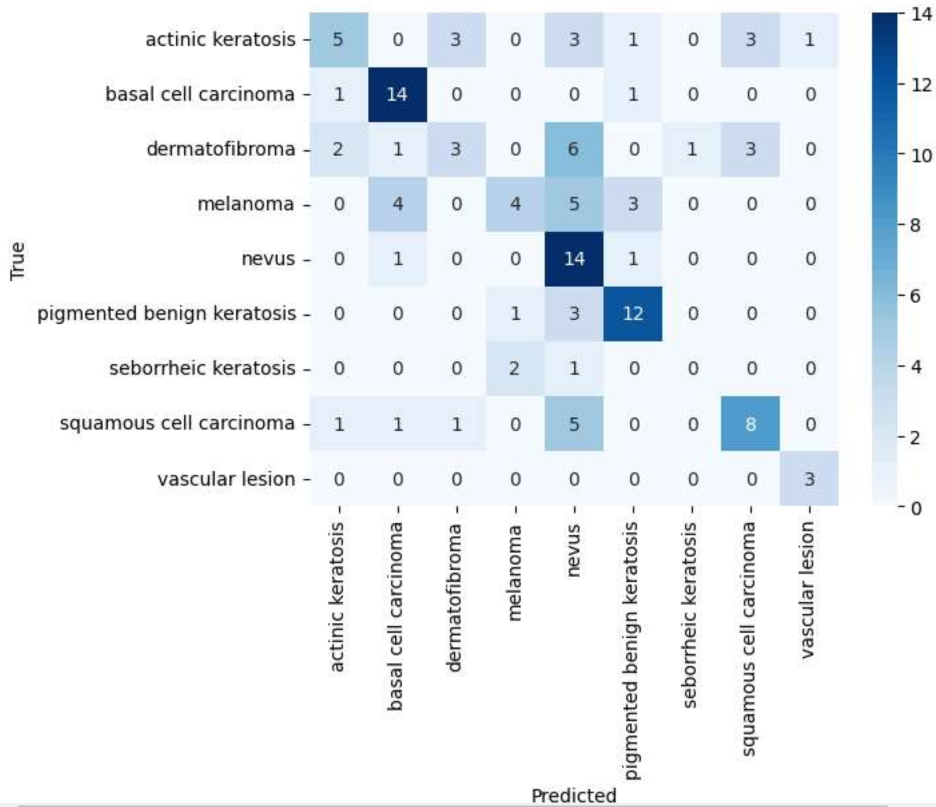
```
              dermatofibroma       0.43      0.19      0.26        16
                    melanoma       0.57      0.25      0.35        16
                       nevus       0.37      0.88      0.52        16
  pigmented benign keratosis       0.67      0.75      0.71        16
        seborrheic keratosis       0.00      0.00      0.00         3
     squamous cell carcinoma       0.60      0.56      0.58        16
             vascular lesion       0.75      1.00      0.86         3

                    accuracy                           0.53       118
                   macro avg       0.51      0.53      0.49       118
                weighted avg       0.54      0.53      0.50       118
```

```
# Confusion matrix
conf_matrix = confusion_matrix(y_test_labels, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Confusion Matrix:
 [[ 4  0  3  0  4  1  0  3  1]
 [ 1 14  0  0  0  1  0  0  0]
 [ 2  1  3  0  6  0  1  3  0]
 [ 0  4  0  4  5  3  0  0  0]
 [ 0  1  0  0 14  1  0  0  0]
 [ 0  0  0  1  3 12  0  0  0]
 [ 0  0  0  2  1  0  0  0  0]
 [ 0  1  1  0  5  0  0  9  0]
 [ 0  0  0  0  0  0  0  0  3]]
```

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

# Assuming 'classes' is defined elsewhere in your code
# ...

# Predict probabilities
y_pred_probs = model.predict(X_test)

# Convert probabilities to class labels (multilabel-indicator format)
# Assuming a threshold of 0.5 for assigning classes
threshold = 0.5
y_pred_multilabel = (y_pred_probs > threshold).astype(int)

# If y_test is also in multilabel-indicator format:
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_multilabel.argmax(axis=1))  # Compare argmax for both

# OR, if y_test is in multiclass format:
# cm = confusion_matrix(y_test, y_pred_multilabel.argmax(axis=1))  # Compare multiclass with argmax of multilabel

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=CLASS_NAMES, yticklabels=CLASS_NAMES)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
```



Double-click (or enter) to edit

```
test_generator = datagen.flow(X_test, y_test, batch_size=32)


# Predicting on test set
predictions = model.predict(test_generator)
predicted_classes = np.argmax(predictions, axis=1)

# Ensure y_test has the correct shape (e.g., one-hot encoded)
if len(y_test.shape) == 1:
    # If y_test is flattened, convert it back to one-hot encoding
    num_classes = predictions.shape[1]  # Get the number of classes from predictions
    y_test = np.eye(num_classes)[y_test]

true_classes = np.argmax(y_test, axis=1)

# Visualizing predictions
def plot_predictions(images, true_labels, predicted_labels, class_names, n=5):
    plt.figure(figsize=(15, 15))
    for i in range(n):
        plt.subplot(1, n, i + 1)
        plt.imshow(images[i])
        plt.title(f"True: {class_names[true_labels[i]]}\nPredicted: {class_names[predicted_labels[i]]}")
        plt.axis('off')
    plt.show()

# Plot the first 5 predictions
plot_predictions(X_test[:5], true_classes[:5], predicted_classes[:5], CLASS_NAMES)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 569ms/step
```