skin cancer detection

October 26, 2025

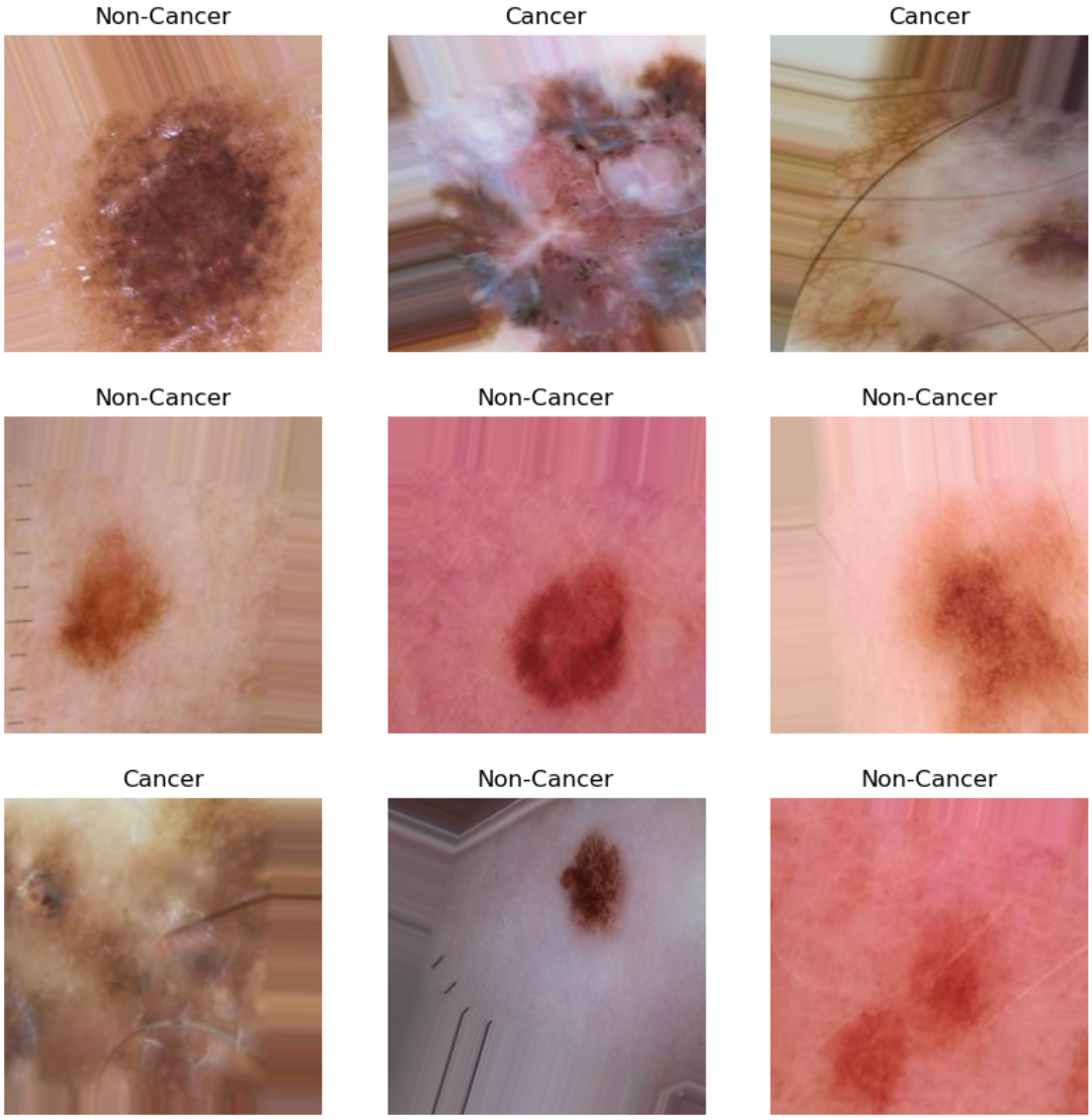
[ ]:
```
import os
import numpy as np

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout, BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
BASE_DIR = "C:\skin cancer detection"  # Change based on your dataset location
TRAIN_DIR = os.path.join(BASE_DIR, "train")
TEST_DIR = os.path.join(BASE_DIR, "test")

# Image size and batch size
IMG_SIZE = (224, 224)  # Optimal size for EfficientNetB0
BATCH_SIZE = 32

# Data Augmentation for Training
train_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalize pixel values to [0, 1]
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

# Rescaling only for Testing
test_datagen = ImageDataGenerator(rescale=1./255)

# Load Training Data
```

```
train_generator = train_datagen.flow_from_directory(
    directory=TRAIN_DIR,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="binary"  # Binary classification: Benign vs Malignant
)

# Load Testing Data
test_generator = test_datagen.flow_from_directory(
    directory=TEST_DIR,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="binary"
)

# Print class labels
print("Class Mapping:", train_generator.class_indices)
```

```
Found 11879 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
Class Mapping: {'Benign': 0, 'Malignant': 1}
```

[3]:
```
# Display some sample images from dataset
def plot_images(generator):
    images, labels = next(generator)  # Get a batch
    plt.figure(figsize=(10, 10))
    for i in range(9):
        plt.subplot(3, 3, i+1)
        plt.imshow(images[i])
        plt.title("Cancer" if labels[i] == 1 else "Non-Cancer")
        plt.axis("off")
    plt.show()

# Plot Training Images
plot_images(train_generator)
```



[4]:
```
from tensorflow.keras import layers, models

# Build the CNN model
model = models.Sequential([
    # 1st Convolution Block
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    layers.MaxPooling2D((2, 2)),

    # 2nd Convolution Block
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
```

```
    # 3rd Convolution Block
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    # Flattening Layer
    layers.Flatten(),

    # Fully Connected Layers
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.5),  # Dropout to reduce overfitting
    layers.Dense(1, activation='sigmoid')  # Output layer for binary␣
 ↪classification
])

# Compile the model
model.compile(optimizer='adam',  # Optimizer for faster convergence
              loss='binary_crossentropy',  # Binary classification loss
              metrics=['accuracy'])  # Accuracy metric

# Model Summary
model.summary()
```

```
C:\Users\adkel\anaconda3\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:113: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_2 (Conv2D)	(None, 52, 52, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 128)	0
flatten (Flatten)	(None, 86528)	0

```
 dense (Dense)                   (None, 512)                44,302,848

 dropout (Dropout)               (None, 512)                         0

 dense_1 (Dense)                 (None, 1)                         513


 Total params: 44,396,609 (169.36 MB)

 Trainable params: 44,396,609 (169.36 MB)

 Non-trainable params: 0 (0.00 B)
```

[5]:
```python
# EarlyStopping to stop training when the model stops improving
early_stopping = EarlyStopping(
    monitor="val_loss",
    patience=5,  # Stop training after 5 epochs with no improvement
    restore_best_weights=True,
    verbose=1
)

# ReduceLROnPlateau to reduce the learning rate if the model stops improving
lr_scheduler = ReduceLROnPlateau(
    monitor="val_loss",
    factor=0.3,
    patience=3,  # Reduce LR if no improvement after 3 epochs
    min_lr=1e-7,
    verbose=1
)


history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=30,  # Set number of epochs to train the model
    validation_data=test_generator,
    validation_steps=test_generator.samples // BATCH_SIZE,
    callbacks=[early_stopping, lr_scheduler]
)
```

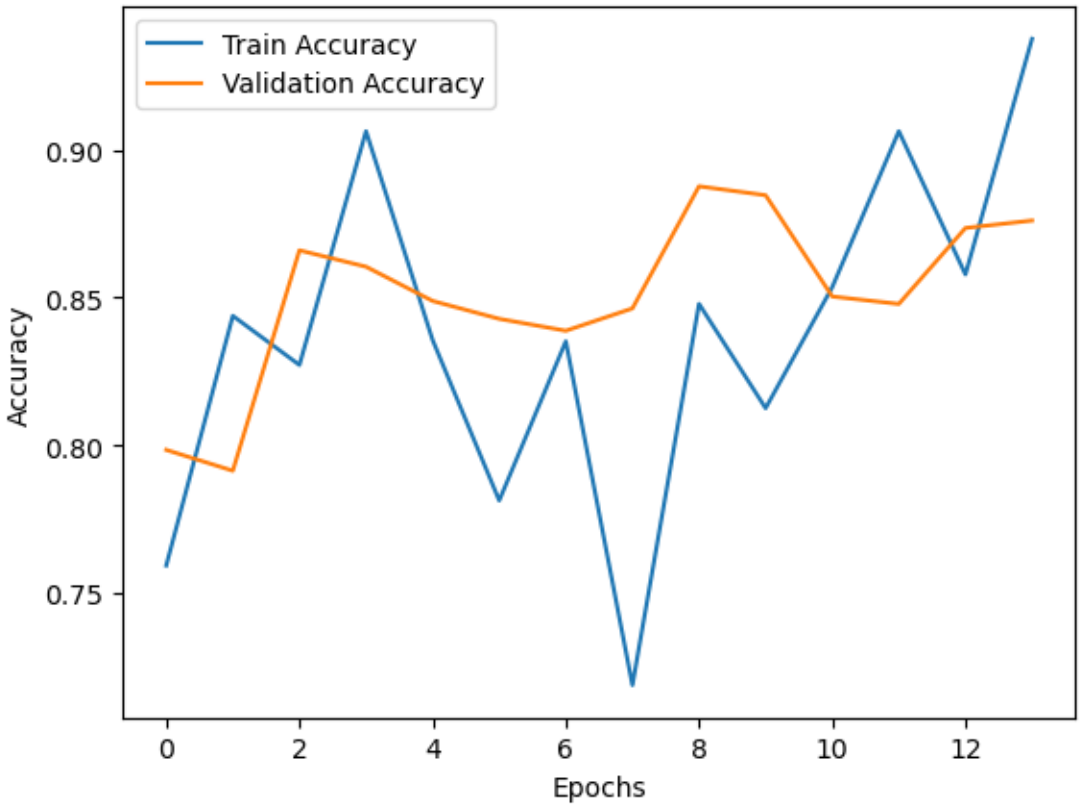
```
C:\Users\adkel\anaconda3\Lib\site-
packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
```

```
  self._warn_if_super_not_called()

Epoch 1/30
371/371             4194s 11s/step -
accuracy: 0.7593 - loss: 0.5132 - val_accuracy: 0.7984 - val_loss: 0.4473 -
learning_rate: 0.0010
Epoch 2/30
  1/371             24:43 4s/step - accuracy:
0.8438 - loss: 0.3878

C:\Users\adkel\anaconda3\Lib\site-
packages\keras\src\trainers\epoch_iterator.py:116: UserWarning: Your input ran
out of data; interrupting training. Make sure that your dataset or generator can
generate at least `steps_per_epoch * epochs` batches. You may need to use the
`.repeat()` function when building your dataset.
  self._interrupted_warning()

371/371             55s 138ms/step -
accuracy: 0.8438 - loss: 0.3878 - val_accuracy: 0.7913 - val_loss: 0.4508 -
learning_rate: 0.0010
Epoch 3/30
371/371             1829s 5s/step -
accuracy: 0.8271 - loss: 0.3965 - val_accuracy: 0.8659 - val_loss: 0.3753 -
learning_rate: 0.0010
Epoch 4/30
371/371             52s 130ms/step -
accuracy: 0.9062 - loss: 0.3144 - val_accuracy: 0.8604 - val_loss: 0.3865 -
learning_rate: 0.0010
Epoch 5/30
371/371             1863s 5s/step -
accuracy: 0.8356 - loss: 0.3791 - val_accuracy: 0.8488 - val_loss: 0.3276 -
learning_rate: 0.0010
Epoch 6/30
371/371             186s 494ms/step -
accuracy: 0.7812 - loss: 0.4253 - val_accuracy: 0.8427 - val_loss: 0.3378 -
learning_rate: 0.0010
Epoch 7/30
371/371             7897s 21s/step -
accuracy: 0.8351 - loss: 0.3710 - val_accuracy: 0.8387 - val_loss: 0.3579 -
learning_rate: 0.0010
Epoch 8/30
  1/371             12:33 2s/step - accuracy:
0.7188 - loss: 0.5647
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
371/371             41s 105ms/step -
accuracy: 0.7188 - loss: 0.5647 - val_accuracy: 0.8463 - val_loss: 0.3497 -
learning_rate: 0.0010
Epoch 9/30
371/371             933s 3s/step -
```

```
accuracy: 0.8478 - loss: 0.3451 - val_accuracy: 0.8876 - val_loss: 0.2933 -
learning_rate: 3.0000e-04
Epoch 10/30
371/371             31s 79ms/step -
accuracy: 0.8125 - loss: 0.4997 - val_accuracy: 0.8846 - val_loss: 0.2971 -
learning_rate: 3.0000e-04
Epoch 11/30
371/371             938s 3s/step -
accuracy: 0.8535 - loss: 0.3396 - val_accuracy: 0.8503 - val_loss: 0.3385 -
learning_rate: 3.0000e-04
Epoch 12/30
  1/371             12:59 2s/step - accuracy:
0.9062 - loss: 0.2965
Epoch 12: ReduceLROnPlateau reducing learning rate to 9.000000427477062e-05.
371/371             32s 80ms/step -
accuracy: 0.9062 - loss: 0.2965 - val_accuracy: 0.8478 - val_loss: 0.3383 -
learning_rate: 3.0000e-04
Epoch 13/30
371/371             922s 2s/step -
accuracy: 0.8578 - loss: 0.3291 - val_accuracy: 0.8735 - val_loss: 0.3172 -
learning_rate: 9.0000e-05
Epoch 14/30
371/371             30s 76ms/step -
accuracy: 0.9375 - loss: 0.2666 - val_accuracy: 0.8760 - val_loss: 0.3156 -
learning_rate: 9.0000e-05
Epoch 14: early stopping
Restoring model weights from the end of the best epoch: 9.
```

[7]:
```
model.save("skin_cancer_cnn.h5")

# Plot Training History
plt.plot(history.history["accuracy"], label="Train Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

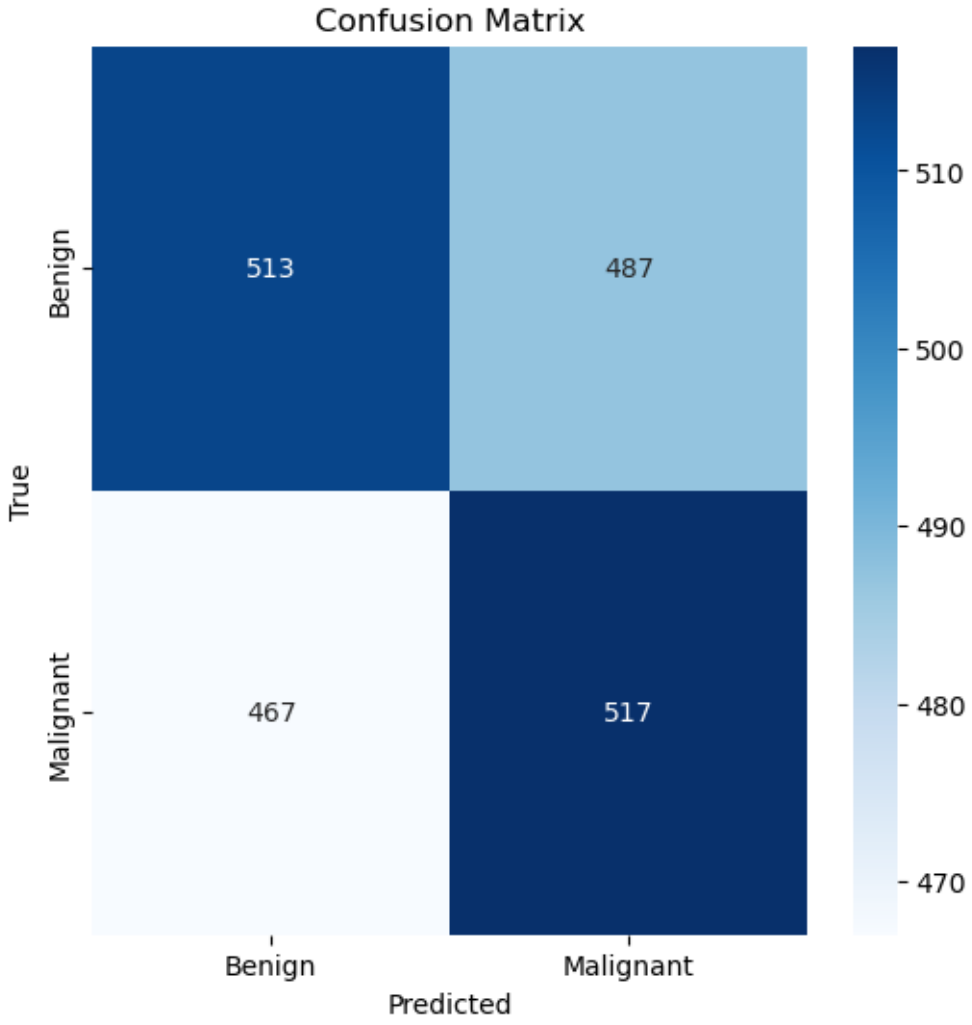
```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```



[8]:
```
# Predict on the test set
test_pred = model.predict(test_generator, steps=test_generator.samples //␣
 ↪BATCH_SIZE, verbose=1)

# Convert predictions to binary labels (0 or 1)
test_pred_labels = (test_pred > 0.5).astype("int32")

# Get the true labels
test_true_labels = test_generator.classes[:len(test_pred_labels)]  # Match␣
 ↪length to predictions

# Classification report
print("Classification Report:")
print(classification_report(test_true_labels, test_pred_labels))

# Confusion Matrix
cm = confusion_matrix(test_true_labels, test_pred_labels)

# Plot Confusion Matrix
plt.figure(figsize=(6, 6))
```

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=train_generator.
  ↪class_indices.keys(), yticklabels=train_generator.class_indices.keys())
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

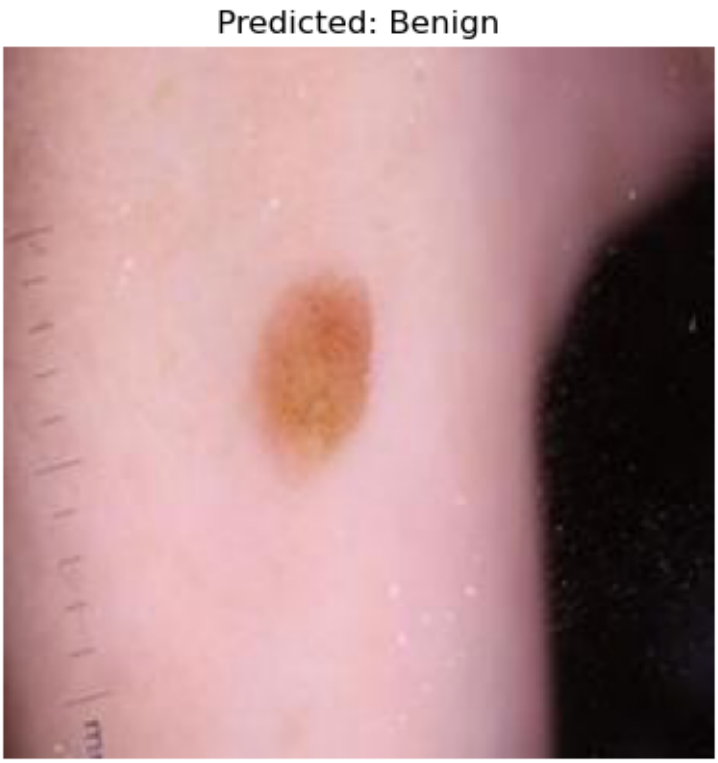
```
62/62              50s 816ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.52      0.51      0.52      1000
           1       0.51      0.53      0.52       984

    accuracy                           0.52      1984
   macro avg       0.52      0.52      0.52      1984
weighted avg       0.52      0.52      0.52      1984
```



[13]:
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model

# Load the entire model
model = load_model('skin_cancer_cnn.h5', compile=False)

def predict_skin_cancer(image_path, model):
    img = image.load_img(image_path, target_size=(224, 224))  # Load Image
    img_array = image.img_to_array(img) / 255.0  # Normalize
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

    # Make Prediction
    prediction = model.predict(img_array)
    class_label = "Malignant" if prediction > 0.5 else "Benign"

    # Show Image with Prediction
    plt.imshow(img)
    plt.title(f"Predicted: {class_label}")
    plt.axis("off")
    plt.show()
```

[17]:
```
predict_skin_cancer(r"C:\skin cancer detection\test\Benign\6331.jpg", model)
```

```
1/1             0s 115ms/step
```



[19]:
```
predict_skin_cancer(r"C:\skin cancer detection\train\Malignant\386.jpg", model)
```

```
1/1             0s 123ms/step
```


Predicted: Malignant

[ ]: