

VARICOSE VEINS DETECTION USING CNN

```
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing import image_dataset_from_directory

DATADIR = "/Users/kartiksolanki/Documents/vericose/dataset/Train"
```

First image from normal



First image from vericose



```
CATEGORIES = ["normal", "vericose"]
```

```
fig, axes = plt.subplots(1, len(CATEGORIES), figsize=(10, 5)) # Create subplots
```

```
for i, category in enumerate(CATEGORIES):
    path = os.path.join(DATADIR, category)
```

```
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path, img))
        img_rgb = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB) # Convert
BGR to RGB
```

```
        axes[i].imshow(img_rgb)
        axes[i].set_title(f"First image from {category}")
```

```

        axes[i].axis('off') # Hide axis

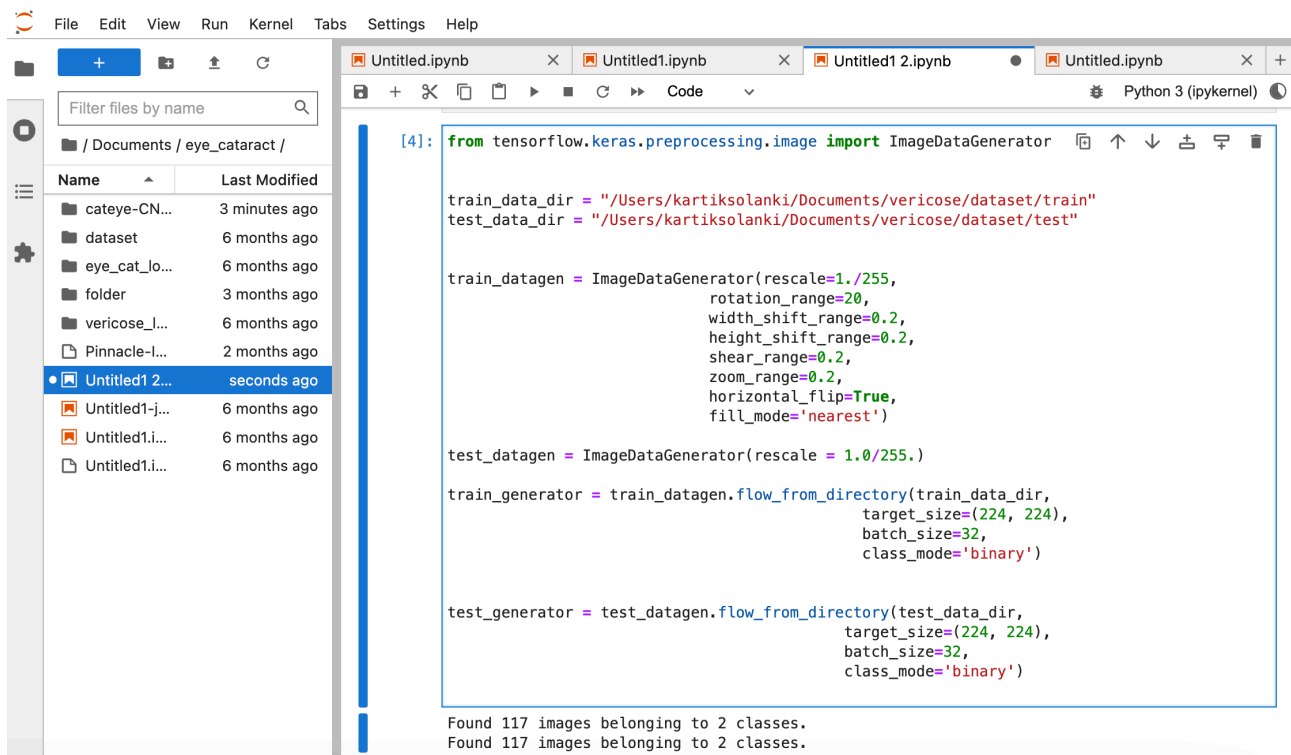
        break # Break the loop once the first image is printed

plt.show()

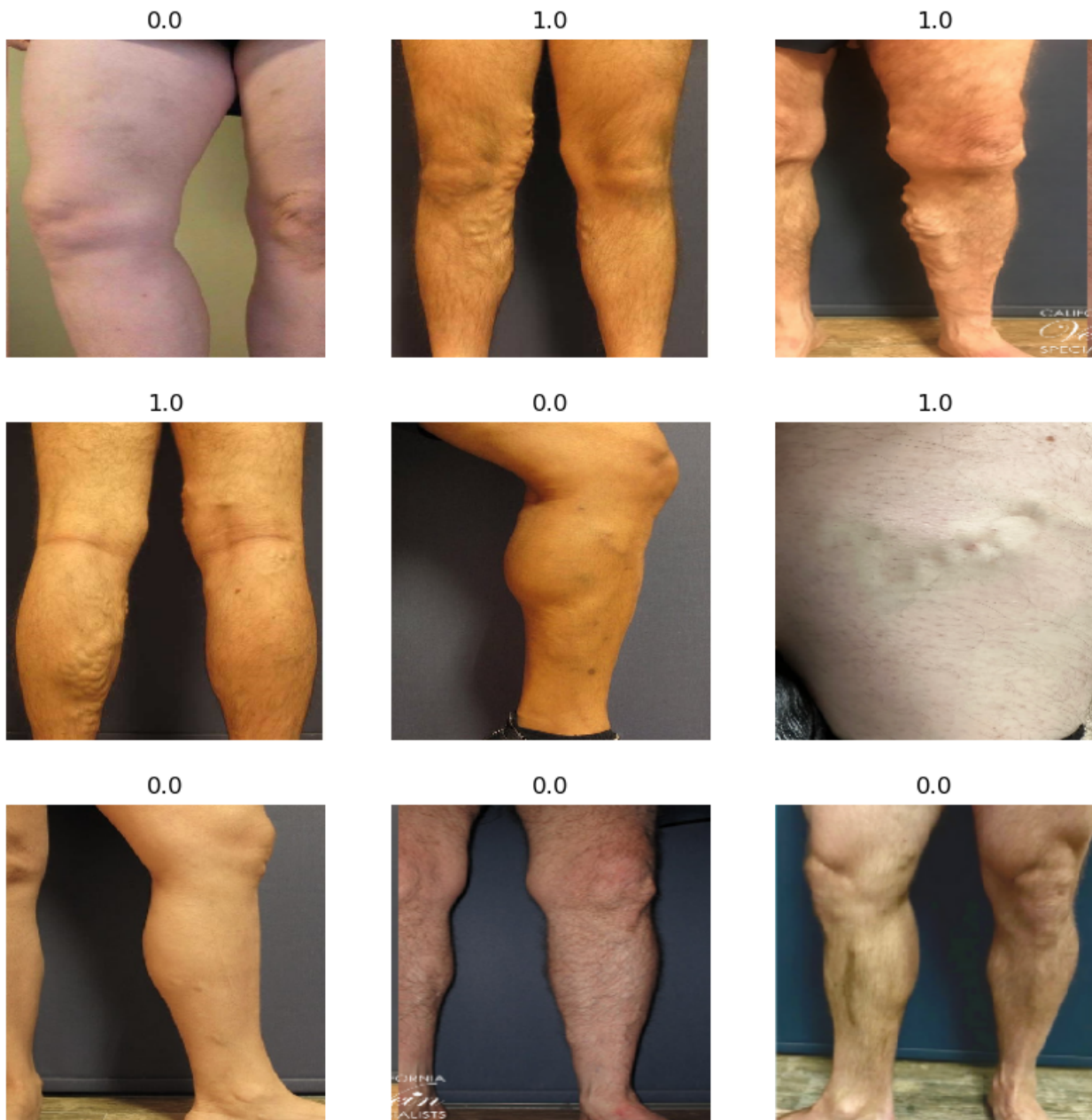
def create_data(path):
    data = []
    for category in CATEGORIES:
        path = os.path.join(DATADIR, category)
        class_num = CATEGORIES.index(category)
        for img in os.listdir(path):
            try:
                files = glob.glob(path+"/"+category+"/*")
                for f in files:
                    img_array = cv2.imread(f)
                    new_array = cv2.resize(img_array, (IMG_SIZE,
IMG_SIZE))

                    data.append([np.array(img_array),
CATEGORIES.index(category)])
            except Exception as e:
                pass
    np.random.shuffle(data)
    return data

```



```
import matplotlib.pyplot as plt
```



```

images, labels = next(test_generator)

plt.figure(figsize=(10, 10))

for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(images[i])
    plt.title(str(labels[i]) if labels[i] == 0 else str(labels[i]))
    plt.axis('off')

plt.show()

```

```

#Implementing convulational Neural Network
model = keras.Sequential([

```

```

layers.Conv2D(32, (3, 3), input_shape=(224, 224, 3),
activation='relu'),
layers.MaxPooling2D(2, 2),

layers.Conv2D(64, (3, 3), activation='relu'),
layers.MaxPooling2D(2, 2),

layers.Conv2D(128, (3, 3), activation='relu'),
layers.MaxPooling2D(2, 2),

layers.Conv2D(256, (3, 3), activation='relu'),#added a new
convulational layer
layers.MaxPooling2D(2, 2),

layers.Flatten(),
layers.Dense(256, activation='relu'),#raised the dense layers to 256

layers.Dropout(0.5),
layers.Dense(1, activation='sigmoid')

])

```

```

[*]: model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(train_generator, epochs=100, validation_data=test_generator, callbacks = [tensorboard_callback]) #at epoch =
model.save('cateye-CNN.model')

Epoch 1/100
4/4 [=====] - 5s 1s/step - loss: 1.0191 - accuracy: 0.5043 - val_loss: 0.6857 - val_accuracy: 0.5983
Epoch 2/100
4/4 [=====] - 3s 940ms/step - loss: 0.6671 - accuracy: 0.5897 - val_loss: 0.6613 - val_accuracy: 0.5983
Epoch 3/100
4/4 [=====] - 4s 898ms/step - loss: 0.6633 - accuracy: 0.5983 - val_loss: 0.6584 - val_accuracy: 0.5983
Epoch 4/100
4/4 [=====] - 4s 872ms/step - loss: 0.6604 - accuracy: 0.5983 - val_loss: 0.6454 - val_accuracy: 0.5983
Epoch 5/100
4/4 [=====] - 4s 994ms/step - loss: 0.6538 - accuracy: 0.5983 - val_loss: 0.6344 - val_accuracy: 0.5983
Epoch 6/100
4/4 [=====] - 3s 912ms/step - loss: 0.6261 - accuracy: 0.5983 - val_loss: 0.6350 - val_accuracy: 0.5983
Epoch 7/100
4/4 [=====] - 3s 900ms/step - loss: 0.6302 - accuracy: 0.5983 - val_loss: 0.6217 - val_accuracy: 0.5897
Epoch 8/100
4/4 [=====] - 3s 811ms/step - loss: 0.6131 - accuracy: 0.5897 - val_loss: 0.6043 - val_accuracy: 0.5897
Epoch 9/100
4/4 [=====] - 4s 858ms/step - loss: 0.5985 - accuracy: 0.5897 - val_loss: 0.5938 - val_accuracy: 0.5897
Epoch 10/100
4/4 [=====] - 4s 871ms/step - loss: 0.5741 - accuracy: 0.6325 - val_loss: 0.5840 - val_accuracy: 0.5726
Epoch 11/100
4/4 [=====] - 4s 859ms/step - loss: 0.7828 - accuracy: 0.6325 - val_loss: 0.6326 - val_accuracy: 0.6581
Epoch 12/100
4/4 [=====] - 4s 853ms/step - loss: 0.6571 - accuracy: 0.5726 - val_loss: 0.6933 - val_accuracy: 0.4786
Epoch 13/100
4/4 [=====] - 4s 977ms/step - loss: 0.6888 - accuracy: 0.5556 - val_loss: 0.6796 - val_accuracy: 0.5983
Epoch 14/100
4/4 [=====] - 4s 993ms/step - loss: 0.6804 - accuracy: 0.5983 - val_loss: 0.6666 - val_accuracy: 0.5983
Epoch 15/100
4/4 [=====] - 4s 877ms/step - loss: 0.6898 - accuracy: 0.5983 - val_loss: 0.6670 - val_accuracy: 0.5983
Epoch 16/100
4/4 [=====] - 4s 906ms/step - loss: 0.6684 - accuracy: 0.5983 - val_loss: 0.6645 - val_accuracy: 0.5983
Epoch 17/100
4/4 [=====] - 4s 894ms/step - loss: 0.6682 - accuracy: 0.5983 - val_loss: 0.6617 - val_accuracy: 0.5983
Epoch 18/100

```

```

test_loss, test_acc = model.evaluate(test_generator)
print(f"Test accuracy: {test_acc}")

```

```

4/4 [=====] - 1s 206ms/step - loss: 0.5581 - accuracy: 0.6838
Test accuracy: 0.6837607026100159

```

```

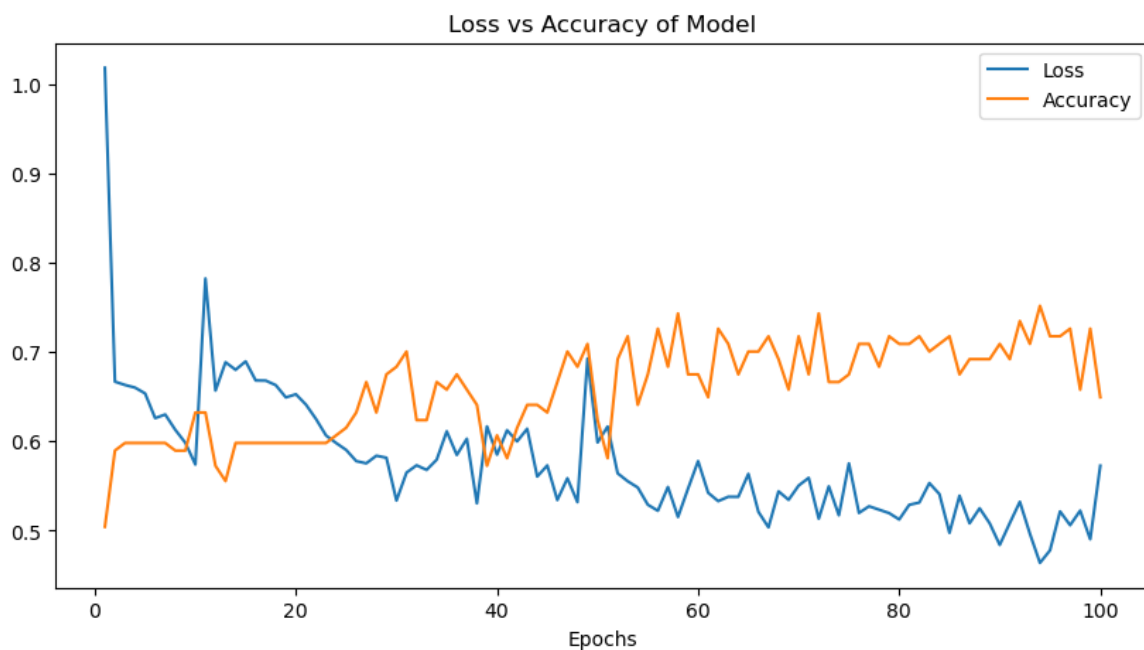
import matplotlib.pyplot as plt

epochs = range(1, 101)
plt.figure(figsize=(10, 5))
plt.title("Loss vs Accuracy of Model")
plt.plot(epochs, history.history['loss'][:100], label='Loss')
plt.plot(epochs, history.history['accuracy'][:100], label='Accuracy')
plt.grid()
plt.xlabel("Epochs")
plt.grid()
plt.legend()
plt.show()

```

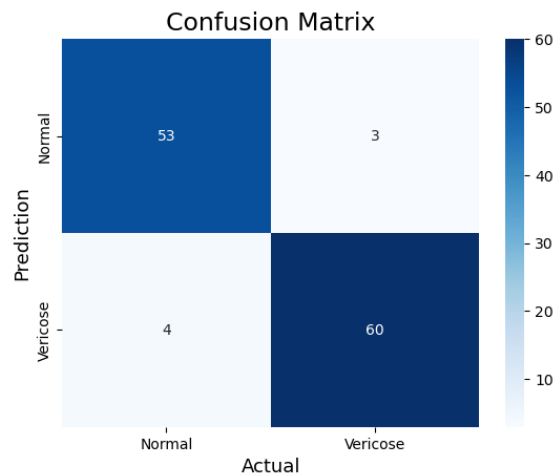
```
import matplotlib.pyplot as plt

epochs = range(1, 101)
plt.figure(figsize=(10, 5))
plt.title("Loss vs Accuracy of Model")
plt.plot(epochs, history.history['loss'][:100], label='Loss')
plt.plot(epochs, history.history['accuracy'][:100], label='Accuracy')
plt.grid()
plt.xlabel("Epochs")
plt.grid()
plt.legend()
plt.show()
```

[illegible]

[illegible]


```
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
```



The screenshot shows a Jupyter Notebook with two code cells. The first cell (index 24) processes a 'normal.webp' image and predicts 'Normal'. The second cell (index 27) processes a 'varicose.jpeg' image and predicts 'Cataract'.

```
[24]: from tensorflow.keras.preprocessing import image
test_image_path = '/Users/kartiksolanki/Documents/varicose/normal.webp'
img = image.load_img(test_image_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

prediction = model.predict(img_array)

binary_prediction = 1 if prediction[0] > 0.5 else 0

label_string = 'Normal' if binary_prediction == 1 else 'Varicose'
print("Actual: Normal")
print(f"Predicted: {label_string}")

1/1 [=====] - 0s 18ms/step
Actual: Normal
Predicted: Normal

[27]: test_image_path = '/Users/kartiksolanki/Documents/varicose/varicose.jpeg'
img = image.load_img(test_image_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

prediction = model.predict(img_array)

binary_prediction = 1 if prediction[0] > 0.5 else 0

label_string = 'Normal' if binary_prediction == 1 else 'Cataract'
print("Actual: Cataract")
print(f"Predicted: {label_string}")

1/1 [=====] - 0s 29ms/step
Actual: Cataract
Predicted: Cataract
```



(varicose.jpeg)



(normal.webp)