Breast Cancer Detection (Working Code Images)

Accuracy(86.62%)

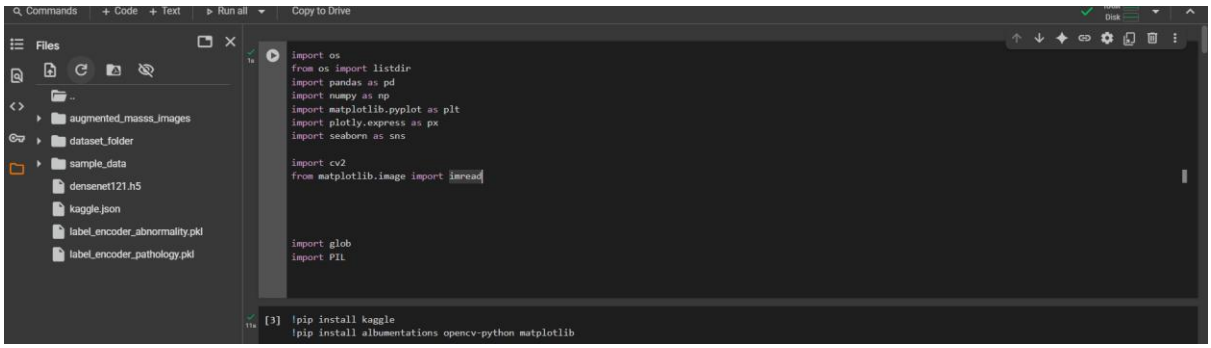
```
n_cls = len(le_path.classes_)
test_df["pathology_encoded"] = test_df["pathology"].map(lambda x: tf.keras.utils.to_cate

class Gen(tf.keras.utils.Sequence):
    def __init__(s, df, bs=32, sz=(256,256)):
        s.df, s.bs, s.sz = df.reset_index(drop=True), bs, sz
    def __len__(s): return (len(s.df)+s.bs-1)//s.bs
    def __getitem__(s, i):
        b = s.df.iloc[i*s.bs:(i+1)*s.bs]
        X = np.array([img_to_array(load_img(p, target_size=s.sz))/255. for p in b["image
        y = np.stack(b["pathology_encoded"].values)
        return X, y

gen = Gen(test_df)
model = load_model("densenet121_pathology.h5")
y_true = np.argmax(np.vstack(test_df["pathology_encoded"].values),1)
y_pred = np.argmax(model.predict(gen),1)
print("accuracy:", accuracy_score(y_true, y_pred))
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapt
  self._warn_if_super_not_called()
15/15 ━━━━━━━━━━━━━━━━━━━━ 101s 6s/step
accuracy: 0.8662420382165605
```

OVERALL CODE :



```
[9]
```

```
mass_case_df = pd.concat([mass_case_train_df, mass_case_test_df], ignore_index=True)
```

Double-click (or enter) to edit

```
[11] image_dir = os.path.join(dataset_folder, 'jpeg')  # Extract images here
```

```
[12] dicom_data.head()
```

	file_path	image_path	AccessionNumber	BitsAllocated	BitsStored	BodyPartExamined
0	CBIS-DDSM/dicom/1.3.6.1.4.1.9590.100.1.2.12930...	CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.129308...	NaN	16	16	BREAST
1	CBIS-DDSM/dicom/1.3.6.1.4.1.9590.100.1.2.24838...	CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.248386...	NaN	16	16	BREAST
2	CBIS-DDSM/dicom/1.3.6.1.4.1.9590.100.1.2.26721...	CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.267213...	NaN	16	16	BREAST
3	CBIS-DDSM/dicom/1.3.6.1.4.1.9590.100.1.2.38118...	CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.381187...	NaN	16	16	BREAST
4	CBIS-DDSM/dicom/1.3.6.1.4.1.9590.100.1.2.38118...	CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.381187...	NaN	8	8	Left Breast

5 rows × 38 columns

```
['cropped images' 'full mammogram images' nan 'ROI mask images']
```

```
[15] full_image_df = dicom_data[dicom_data['SeriesDescription'] == 'full mammogram images']
     full_image_df['image_path'] = full_image_df['image_path'].apply(lambda x: x.replace('CBIS-DDSM/jpeg', image_dir))
```

```
/tmp/ipython-input-15-234251504.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  full_image_df['image_path'] = full_image_df['image_path'].apply(lambda x: x.replace('CBIS-DDSM/jpeg', image_dir))
```

```
[16] mass_case_df['SeriesInstanceUID'] = mass_case_df['image file path'].apply(lambda path: path.split('/')[2])
```

```
[17] map_image_path_series_id = dict(zip(full_image_df['SeriesInstanceUID'].values,full_image_df['image_path'].values))
```

```
[18] mass_case_df['image_path'] = mass_case_df['SeriesInstanceUID'].apply(lambda seri: map_image_path_series_id[seri])
     mass_case_df.head(1)
```

	patient_id	breast_density	left or right breast	image view	abnormality id	abnormality type	mass shape	mass margins	assessment	pathology	subtle
0	P_00001	3	LEFT	CC	1	mass	IRREGULAR-ARCHITECTURAL_DISTORTION	SPICULATED	4	MALIGNANT	

```python
import os
import cv2
import pandas as pd
import albumentations as A
from tqdm import tqdm

ALLOWED_EXTENSIONS = {'jpg', 'jpeg', 'png', 'bmp', 'tiff'}

mammography_augmentations = {


    "GaussNoise": A.GaussNoise(var_limit=(5.0, 15.0), p=1.0),
    "RandomBrightnessContrast": A.RandomBrightnessContrast(brightness_limit=0.05, contrast_limit=0.05, p=1.0),
    "RandomGamma": A.RandomGamma(gamma_limit=(98, 102), p=1.0),
    "Equalize": A.Equalize(p=1.0),
    "GaussianBlur": A.GaussianBlur(blur_limit=1, p=1.0),

}

def apply_augmentations(image, augmentations):
    augmented_images = {}
    for aug_name, aug in augmentations.items():
        augmented = aug(image=image)
        augmented_images[aug_name] = augmented['image']
    return augmented_images

def balance_classes_mass(dataset_folder, images_folder, output_folder, dataframe):
    class_counts = dataframe["pathology"].value_counts()
    min_class_count = class_counts.min()
    max_class_count = class_counts.max()
```

```python
def balance_classes_mass(dataset_folder, images_folder, output_folder, dataframe):
    class_counts = dataframe["pathology"].value_counts()
    min_class_count = class_counts.min()
    max_class_count = class_counts.max()

    os.makedirs(output_folder, exist_ok=True)
    final_data = []

    for pathology in dataframe["pathology"].unique():
        df_class = dataframe[dataframe["pathology"] == pathology]
        real_images = df_class.sample(n=min_class_count, replace=False).reset_index(drop=True)
        real_images_copy = real_images.copy()

        num_augmented_needed = max_class_count - min_class_count
        augmented_data = []
        aug_index = 0

        print(f"{pathology}: Keeping {min_class_count} real images, generating {num_augmented_needed} augmented.")

        while len(augmented_data) < num_augmented_needed:
            row = real_images.iloc[aug_index % len(real_images)]
            image_path = row['image_path']

            if not os.path.exists(image_path):
                print(f"Image not found: {image_path}")
                aug_index += 1
                continue

            image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
            if image is None:
                print(f"Failed to load image: {image_path}")
                continue

            image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
            if image is None:
                print(f"Failed to load image: {image_path}")
                aug_index += 1
                continue

            image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)

            for aug_name, aug in mammography_augmentations.items():
                if len(augmented_data) >= num_augmented_needed:
                    break

                augmented = aug(image=image)['image']
                new_image_name = f"aug_{aug_name}_{aug_index}_{os.path.basename(image_path)}"
                new_image_path = os.path.join(output_folder, new_image_name)
                cv2.imwrite(new_image_path, augmented)

                new_row = row.copy()
                new_row['image_path'] = new_image_path
                augmented_data.append(new_row)

            aug_index += 1

        final_data.extend(real_images_copy.to_dict('records'))
        final_data.extend(augmented_data)

    final_df = pd.DataFrame(final_data)
    final_df.to_csv(os.path.join(dataset_folder, "augmented_balanced_mass_dataset.csv"), index=False)
    print("Balanced dataset saved with real = min count and total = max count per class.")
```

```
/tmp/ipython-input-21-3461631296.py:12: UserWarning: Argument(s) 'var_limit' are not valid for transform Gaus
  "GaussNoise": A.GaussNoise(var_limit=(5.0, 15.0), p=1.0),
```

```
balance_classes_mass(dataset_folder='/content/dataset_folder',
                     images_folder="os.path.join(dataset_folder, 'jpeg')",
                     output_folder="augmented_masss_images",
                     dataframe=mass_case_df)
```

```
MALIGNANT: Keeping 141 real images, generating 643 augmented.
BENIGN: Keeping 141 real images, generating 643 augmented.
BENIGN_WITHOUT_CALLBACK: Keeping 141 real images, generating 643 augmented.
Balanced dataset saved with real = min count and total = max count per class.
```

```
[23] aug_df = pd.read_csv("/content/dataset_folder/augmented_balanced_mass_dataset.csv")
```

```
[24] aug_df['pathology'].value_counts()
```

pathology	count
MALIGNANT	784
BENIGN	784
BENIGN_WITHOUT_CALLBACK	784

```
from sklearn.model_selection import train_test_split
import pandas as pd

aug_df = pd.read_csv("/content/dataset_folder/augmented_balanced_mass_dataset.csv")
train_df, test_df = train_test_split(aug_df, test_size=0.2, shuffle=True, random_state=42)
```

```
[26] train_df.head()
```

	patient_id	breast_density	left or right breast	image view	abnormality id	abnormality type	mass shape	mass margins	assessment	pathology	subtlety	
564	P_01799	2	LEFT	MLO	1	mass	IRREGULAR	ILL_DEFINED	4	MALIGNANT	4	Training_
297	P_00450	3	LEFT	MLO	1	mass	IRREGULAR	ILL_DEFINED	4	MALIGNANT	2	Training_
932	P_00044	2	RIGHT	MLO	2	mass	OVAL	CIRCUMSCRIBED	3	BENIGN	5	Training_
239	P_01151	2	RIGHT	CC	1	mass	LOBULATED	OBSCURED-ILL_DEFINED	3	MALIGNANT	3	Training
99	P_00543	2	RIGHT	MLO	1	mass	IRREGULAR	ILL_DEFINED	4	MALIGNANT	4	Training_

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import pickle


# Recreate and fit the encoders
label_encoder_abnormality = LabelEncoder()
label_encoder_pathology = LabelEncoder()

label_encoder_abnormality.fit(train_df["abnormality type"])
label_encoder_pathology.fit(train_df["pathology"])

# Save the encoders to disk
with open("label_encoder_abnormality.pkl", "wb") as f:
    pickle.dump(label_encoder_abnormality, f)
with open("label_encoder_pathology.pkl", "wb") as f:
    pickle.dump(label_encoder_pathology, f)

print("Encoders saved successfully as 'label_encoder_abnormality.pkl' and 'label_encoder_pathology.pkl'")
```

```
Encoders saved successfully as 'label_encoder_abnormality.pkl' and 'label_encoder_pathology.pkl'
```

```
[28] import tensorflow as tf
import pandas as pd
import numpy as np
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras import layers, Model
from tensorflow.keras.applications import DenseNet121
```

```
[28] model.fit(train_gen, epochs=10)
     model.save("densenet121.h5")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_kernels_notop.
29084464/29084464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should ca
  self._warn_if_super_not_called()
Epoch 1/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 276s 4s/step - accuracy: 0.5135 - loss: 1.3319
Epoch 2/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 238s 4s/step - accuracy: 0.6325 - loss: 0.9164
Epoch 3/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 231s 4s/step - accuracy: 0.6701 - loss: 0.7770
Epoch 4/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 234s 4s/step - accuracy: 0.7012 - loss: 0.7385
Epoch 5/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 232s 4s/step - accuracy: 0.7390 - loss: 0.6424
Epoch 6/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 265s 4s/step - accuracy: 0.7644 - loss: 0.5876
Epoch 7/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 232s 4s/step - accuracy: 0.7761 - loss: 0.5817
Epoch 8/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 274s 4s/step - accuracy: 0.7837 - loss: 0.5537
Epoch 9/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 237s 4s/step - accuracy: 0.8115 - loss: 0.4958
Epoch 10/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 239s 4s/step - accuracy: 0.8138 - loss: 0.4822
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 leg
```