SKIN DISEASE USING CNN

Number of Classes - 23

DATASET

Training Dataset - 15557 Files in JPG Format

Testing Dataset - 4002 Files in JPG Format

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
batchsize = 32
imagesize = 128
train= tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/FINAL YEAR PROJECT/train",
    shuffle = True,
    image_size = (imagesize,imagesize),
    batch_size = batchsize
)
test= tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/FINAL YEAR PROJECT/test",
    shuffle = True,
    image_size = (imagesize,imagesize),
    batch_size =batchsize
)
```

```
Found 15569 files belonging to 23 classes.
Found 4002 files belonging to 23 classes.
```

```
print(len(train))
print(len(test))
```

```
487
126
```

```
type(train)
```

```
tensorflow.python.data.ops.batch_op._BatchDataset
```

```
class_names = train.class_names
class_names
```

```
['Acne and Rosacea Photos',
 'Actinic Keratosis Basal Cell Carcinoma and other Malignant Lesions',
 'Atopic Dermatitis Photos',
 'Bullous Disease Photos',
 'Cellulitis Impetigo and other Bacterial Infections',
 'Eczema Photos',
 'Exanthems and Drug Eruptions',
 'Hair Loss Photos Alopecia and other Hair Diseases',
 'Herpes HPV and other STDs Photos',
 'Light Diseases and Disorders of Pigmentation',
 'Lupus and other Connective Tissue diseases',
 'Melanoma Skin Cancer Nevi and Moles',
 'Nail Fungus and other Nail Disease',
 'Poison Ivy Photos and other Contact Dermatitis',
 'Psoriasis pictures Lichen Planus and related diseases',
 'Scabies Lyme Disease and other Infestations and Bites',
 'Seborrheic Keratoses and other Benign Tumors',
 'Systemic Disease',
 'Tinea Ringworm Candidiasis and other Fungal Infections',
 'Urticaria Hives',
 'Vascular Tumors',
 'Vasculitis Photos',
 'Warts Molluscum and other Viral Infections']
```

```
plt.figure(figsize = (15,15))
for image_batch, label_batch in train.take(1):
    for j in range(16):
        ax = plt.subplot(4,4,j+1)
        plt.imshow(image_batch[j].numpy().astype("uint8"))
        plt.title(class_names[label_batch[j]],fontsize = 8)
        plt.axis("off")
```


```
def dataset_partitions(ds, train_split = 0.9,val_split = 0.1,shuffle =
True,shuffle_size =10000):
    assert(train_split+val_split) == 1
    ds_size = len(ds)
    if shuffle:
        ds = ds.shuffle(shuffle_size, seed = 24)
    train_size = int(train_split*ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    return train_ds, val_ds

train_ds, val_ds =dataset_partitions(train)

len(val_ds)
```

```
48
```

```
train_ds =
train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds_ds =
val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds =
test.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)

resizerescale = tf.keras.Sequential([

layers.experimental.preprocessing.Resizing(imagesize,imagesize),

layers.experimental.preprocessing.Rescaling(1.0/255)
])

augmentation = tf.keras.Sequential([

layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"
),

layers.experimental.preprocessing.RandomRotation(0.2),
])

train_ds = train_ds.map(
    lambda x, y: (augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)

model = models.Sequential([
                            resizerescale,
                            layers.Conv2D(64, kernel_size = (3,3),
activation = 'relu', input_shape =
(batchsize,imagesize,imagesize,3) ),
                            layers.AveragePooling2D((2,2)),
                            layers.Conv2D(128, kernel_size = (3,3),
activation = 'relu'),
                            layers.AveragePooling2D((2,2)),
                            layers.Conv2D(128, kernel_size = (3,3),
activation = 'relu'),
                            layers.AveragePooling2D((2,2)),
                            layers.Conv2D(128, (3, 3),
activation='relu'),
                            layers.AveragePooling2D((2, 2)),
                            layers.Conv2D(128, (3, 3),
activation='relu'),
                            layers.AveragePooling2D((2, 2)),
                            layers.Flatten(),
                            layers.Dense(128, activation='relu'),
                            layers.Dense(23, activation='softmax')
])
model.build(input_shape = (batchsize,128,128,3))

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 128, 128, 3)         0

 conv2d (Conv2D)             (32, 126, 126, 64)        1792

 average_pooling2d (AverageP  (32, 63, 63, 64)         0
 ooling2D)

 conv2d_1 (Conv2D)           (32, 61, 61, 128)         73856

 average_pooling2d_1 (Averag  (32, 30, 30, 128)        0
 ePooling2D)

 conv2d_2 (Conv2D)           (32, 28, 28, 128)         147584

 average_pooling2d_2 (Averag  (32, 14, 14, 128)        0
 ePooling2D)

 conv2d_3 (Conv2D)           (32, 12, 12, 128)         147584

 average_pooling2d_3 (Averag  (32, 6, 6, 128)          0
 ePooling2D)

 conv2d_4 (Conv2D)           (32, 4, 4, 128)           147584

 average_pooling2d_4 (Averag  (32, 2, 2, 128)          0
 ePooling2D)

 flatten (Flatten)           (32, 512)                 0

 dense (Dense)               (32, 128)                 65664

 dense_1 (Dense)             (32, 23)                  2967

=================================================================
Total params: 587,031
Trainable params: 587,031
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(
    optimizer='adam',
    loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

```
epoch = 10
history = model.fit(
    train_ds,
    epochs = epoch,
    batch_size = batchsize,
    validation_data = val_ds,
    verbose = 1,
)
```

```
Epoch 1/10
438/438 [==============================] - 924s 2s/step - loss: 2.9651
- accuracy: 0.1009 - val_loss: 2.9095 - val_accuracy: 0.1296
Epoch 2/10
438/438 [==============================] - 864s 2s/step - loss: 2.8619
- accuracy: 0.1438 - val_loss: 2.8227 - val_accuracy: 0.1510
Epoch 3/10
438/438 [==============================] - 851s 2s/step - loss: 2.8091
- accuracy: 0.1677 - val_loss: 2.8128 - val_accuracy: 0.1479
Epoch 4/10
438/438 [==============================] - 851s 2s/step - loss: 2.7661
- accuracy: 0.1769 - val_loss: 2.7693 - val_accuracy: 0.1771
Epoch 5/10
438/438 [==============================] - 937s 2s/step - loss: 2.7279
- accuracy: 0.1887 - val_loss: 2.7000 - val_accuracy: 0.1849
Epoch 6/10
438/438 [==============================] - 925s 2s/step - loss: 2.6939
- accuracy: 0.1985 - val_loss: 2.6606 - val_accuracy: 0.2012
Epoch 7/10
438/438 [==============================] - 933s 2s/step - loss: 2.6637
- accuracy: 0.2062 - val_loss: 2.6146 - val_accuracy: 0.2116
Epoch 8/10
438/438 [==============================] - 934s 2s/step - loss: 2.6374
- accuracy: 0.2127 - val_loss: 2.6282 - val_accuracy: 0.2246
Epoch 9/10
438/438 [==============================] - 931s 2s/step - loss: 2.6042
- accuracy: 0.2235 - val_loss: 2.6102 - val_accuracy: 0.2337
Epoch 10/10
438/438 [==============================] - 913s 2s/step - loss: 2.5852
- accuracy: 0.2311 - val_loss: 2.5461 - val_accuracy: 0.2526
```

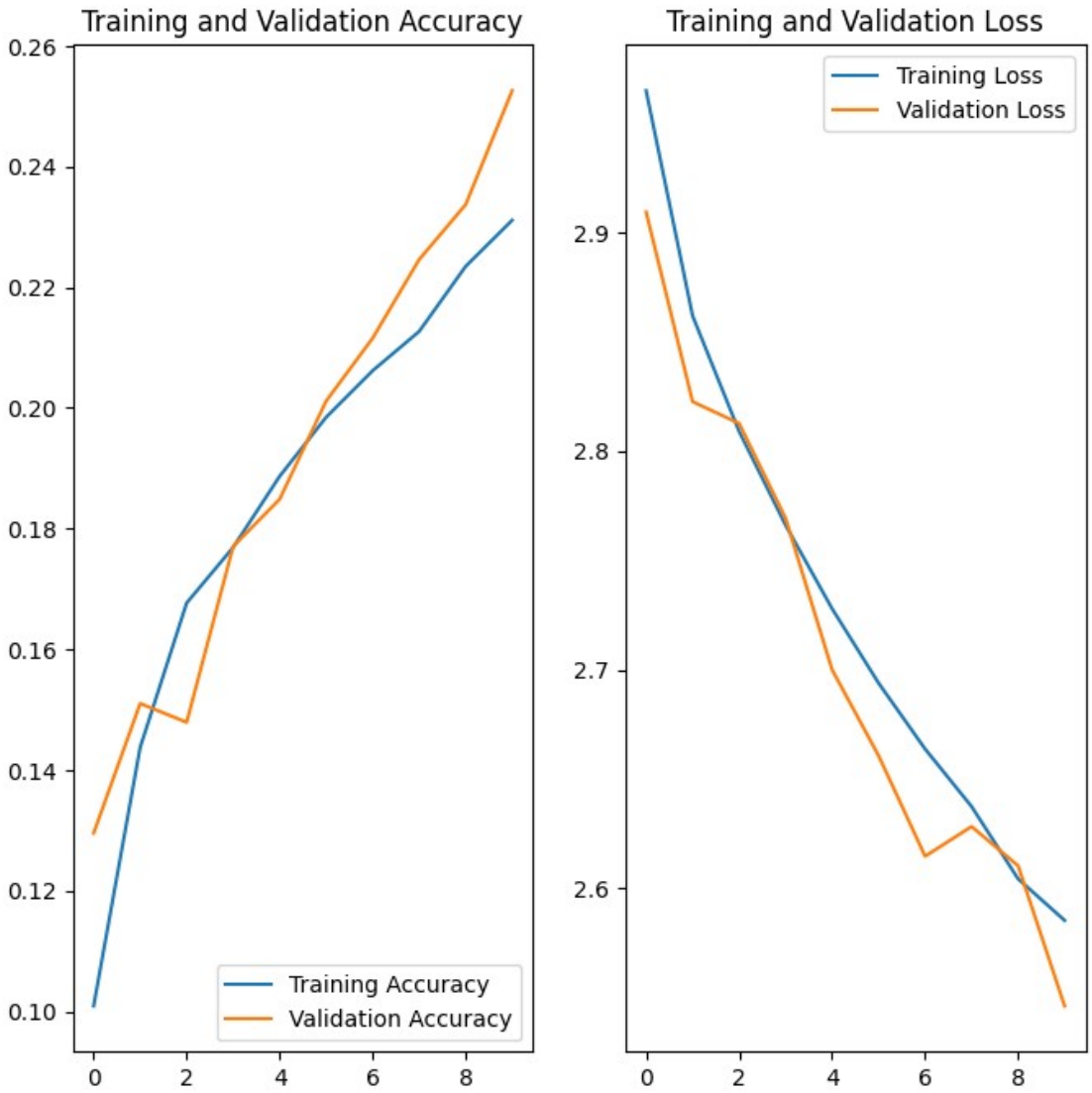
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(epoch), acc, label='Training Accuracy')
plt.plot(range(epoch), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
```

```
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(epoch), loss, label='Training Loss')
plt.plot(range(epoch), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



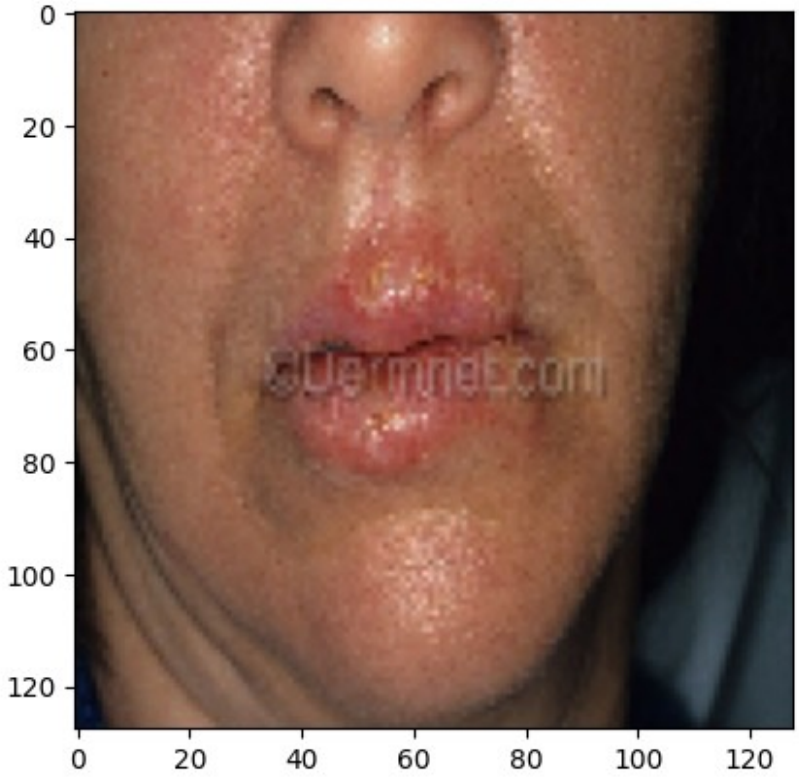

```
for images_batch, labels_batch in test_ds.take(1):

    first_image = images_batch[23].numpy().astype('uint8')
    first_label = labels_batch[25].numpy()

    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:",class_names[first_label])
```

```
    batch_prediction = model.predict(images_batch)
    print("predicted
label:",class_names[np.argmax(batch_prediction[23])])
```

```
first image to predict
actual label: Psoriasis pictures Lichen Planus and related diseases
1/1 [==============================] - 4s 4s/step
predicted label: Tinea Ringworm Candidiasis and other Fungal
Infections
```




```
def predict(model, img):
    img_array =
tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence

plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
```

```
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model,
images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted:
{predicted_class}.\n Confidence: {confidence}%")

        plt.axis("off")
```

```
1/1 [==============================] - 0s 241ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 36ms/step
```