

Table of contents:

- [Introduction and brief introduction of data](#)
- [1. Import Necessary Library](#)
- [2. Data Loading and Preprocessing](#)
- [3. DenseNet201 function Calling](#)
- [5. Now Plot the Training Loss, Validation Loss](#)
- [6. Now Predictions for random image](#)

Introduction

A Brain tumor is considered as one of the aggressive diseases, among children and adults. Brain tumors account for 85 to 90 percent of all primary Central Nervous System(CNS) tumors. Every year, around 11,700 people are diagnosed with a brain tumor. The 5-year survival rate for people with a cancerous brain or CNS tumor is approximately 34 percent for men and 36 percent for women. Brain Tumors are classified as: Benign Tumor, Malignant Tumor, Pituitary Tumor, etc. Proper treatment, planning, and accurate diagnostics should be implemented to improve the life expectancy of the patients. The best technique to detect brain tumors is Magnetic Resonance Imaging (MRI). A huge amount of image data is generated through the scans. These images are examined by the radiologist. A manual examination can be error-prone due to the level of complexities involved in brain tumors and their properties.

Application of automated classification techniques using Machine Learning(ML) and Artificial Intelligence(AI) has consistently shown higher accuracy than manual classification. Hence, proposing a system performing detection and classification by using Deep Learning Algorithms using Convolution Neural Network (CNN), Artificial Neural Network (ANN), and Transfer Learning (TL) would be helpful to doctors all around the world.

Summary

Brain Tumors are complex. There are a lot of abnormalities in the sizes and location of the brain tumor(s). This makes it really difficult for complete understanding of the nature of the tumor. Also, a professional Neurosurgeon is required for MRI analysis. Often times in developing countries the lack of skillful doctors and lack of knowledge about tumors makes it really challenging and time-consuming to generate reports from MRI'. So an automated system on Cloud can solve this problem.

Read More About Dataset: [Click](#)

The key folders in the dataset include:

1. GLIOMA TUMOR
2. MENINGIOMA TUMOR
3. NO TUMOR
4. PITUITARY TUMOR

Remote source:<https://github.com/sartajbhuvaji/brain-tumor-classification-dataset> The folder contains MRI data. The images are already split into Training and Testing folders. Each folder has more four subfolders. These folders have MRIs of respective tumor classes.

Table of Contents

```
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import DenseNet201
from tensorflow.keras.applications.densenet import preprocess_input,
decode_predictions
from tensorflow.keras.layers import Flatten, Dense, Dropout,
Activation
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
ZeroPadding2D, BatchNormalization
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.callbacks import ModelCheckpoint,
EarlyStopping , ReduceLR0nPlateau
from tensorflow.keras.utils import to_categorical, plot_model

import numpy as np
import os
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

Table of Contents

[illegible]

```

True,
"nearest")

validation_datasets_generator =
ImageDataGenerator(rescale=1.0/255)

train_datasets_generator_data =
train_datasets_generator.flow_from_directory(
    batch_size = batch_size,
    directory = train_datasets,
    shuffle = True,
    target_size = (image_size, image_size),
    class_mode = "categorical"
)

validation_datasets_generator_data =
validation_datasets_generator.flow_from_directory(
    batch_size = batch_size,
    directory = validation_datasets,
    shuffle = True,
    target_size = (image_size, image_size),
    class_mode = "categorical"
)

return train_datasets_generator_data,
validation_datasets_generator_data

train_data , validation_data = prepare_the_datasets(train_datasets,
validation_datasets, batch_size, image_size)

```

Table of Contents

Each Keras Application expects a specific kind of input preprocessing. For DenseNet, call `tf.keras.applications.densenet.preprocess_input` on your inputs before passing them to the model.

Arguments

- 1. include_top:** whether to include the fully-connected layer at the top of the network.
- 1. weights:** one of None (random initialization), 'imagenet' (pre-training on ImageNet), or the path to the weights file to be loaded.
- 1. input_tensor:** optional Keras tensor (i.e. output of `layers.Input()`) to use as image input for the model.

1. input_shape: optional shape tuple, only to be specified if include_top is False (otherwise the input shape has to be (224, 224, 3) (with 'channels_last' data format) or (3, 224, 224) (with 'channels_first' data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 32. E.g. (200, 200, 3) would be one valid value.

1. pooling: Optional pooling mode for feature extraction when include_top is False.

None means that the output of the model will be the 4D tensor output of the last convolutional block.

avg means that global average pooling will be applied to the output of the last convolutional block, and thus the output of the model will be a 2D tensor.

max means that global max pooling will be applied.

1. classes: optional number of classes to classify images into, only to be specified if include_top is True, and if no weights argument is specified.

1. classifier_activation: A str or callable. The activation function to use on the "top" layer. Ignored unless include_top=True. Set classifier_activation=None to return the logits of the "top" layer. When loading pretrained weights, classifier_activation can only be None or "softmax".

From [Keras Application website](#)

```
densenet= tf.keras.applications.DenseNet201(
    include_top=False,
    weights="imagenet",
    pooling=None,
    input_shape=(224,224,3)
)

for layer in densenet.layers:
    layer.trainable= False

""" Creates the top or head of the model that will be placed on top of
the layers """

def addTopModel(bottom_model, num_class, D=256):
    top_model= bottom_model.output
    top_model= Flatten(name="flatten") (top_model)
    top_model= Dense(D, activation="relu") (top_model)
    top_model= Dropout(0.3) (top_model)
    top_model= Dense(num_class, activation="softmax") (top_model)
    return top_model

num_classes= 4
Fc_Head= addTopModel(densenet, num_classes)
model= Model(inputs= densenet.input, outputs= Fc_Head)

checkpoint= ModelCheckpoint("/Trained Models/densenet101.h5",
                           monitor= "val_loss",
                           mode= "min",
                           save_best_only= True,
```

```

        verbose=1)

earlystop= EarlyStopping(monitor= "val_loss",
                        min_delta= 0,
                        patience= 3,
                        verbose=1,
                        restore_best_weights= True)

reduce_lr= ReduceLROnPlateau(monitor= "val_loss",
                            factor=0.2,
                            patience= 5,
                            verbose=1,
                            min_delta= 0.000001)

callbacks=[earlystop, checkpoint, reduce_lr]


model.compile(loss= 'categorical_crossentropy',
              optimizer= RMSprop(learning_rate=0.00001),
              metrics= ['accuracy'])

nb_train_samples= 2870
nb_validation_samples= 394
epochs= 20
batch_size= 128

history= model.fit(train_data,
                  steps_per_epoch= nb_train_samples//batch_size,
                  epochs= epochs,
                  callbacks=callbacks,
                  validation_data= validation_data,
                  validation_steps=
nb_validation_samples//batch_size)

model_evaluation= model.evaluate(validation_data, batch_size =
batch_size)

```

 Table of Contents

```

import numpy as np

def plot_training_curves(history):
    loss= np.array(history.history['loss'])
    val_loss= np.array(history.history['val_loss'])

    accuracy= np.array(history.history['accuracy'])
    val_accuracy= np.array(history.history['val_accuracy'])

```

```

epochs= range(len(history.history['loss']))

fig, (ax1, ax2)= plt.subplots(1,2,figsize=(10,3))

#plot loss
ax1.plot(epochs, loss, label='traing_loss', marker='o')

ax1.plot(epochs, val_loss, label='val_loss', marker='o')

ax1.fill_between(epochs,loss, val_loss,
where=(loss>val_loss),color='C0',alpha=0.3,interpolate=True)
ax1.fill_between(epochs,loss, val_loss,
where=(loss<val_loss),color='C1',alpha=0.3,interpolate=True)

ax1.set_title('Loss(Lower Means Better)',fontsize= 16)
ax1.set_xlabel('Epochs', fontsize=10)

ax1.legend()

#plot Accuracy
ax2.plot(epochs, accuracy, label='traing_accuracy', marker='o')

ax2.plot(epochs, val_accuracy, label='val_accuracy', marker='o')

ax2.fill_between(epochs,accuracy, val_accuracy,
where=(accuracy>val_accuracy),color='C0',alpha=0.3,interpolate=True)
ax2.fill_between(epochs,accuracy, val_accuracy,
where=(accuracy<val_accuracy),color='C1',alpha=0.3,interpolate=True)

ax2.set_title('Accuracy(Higher Means Better)',fontsize= 16)
ax2.set_xlabel('Epochs', fontsize=10)

ax2.legend()

plot_training_curves(history)

```

Table of Contents

```

from tensorflow.keras.applications.imagenet_utils import
preprocess_input
img=
image.load_img("../input/brain-tumor-classification-mri/Testing/no_tum
or/image(10).jpg",target_size=(224,224))

x= image.img_to_array(img)
x=x/255
x= np.expand_dims(x, axis=0)
img_data=preprocess_input(x)

```

```

img_data.shape
preds= model.predict(x)
preds= np.argmax(preds, axis=1)
if preds==1:
    preds="The image is glioma Tumor"
elif preds==2:
    preds="The image is meningioma Tumor"
elif preds==3:
    preds="The image is No Tumor"
else:
    preds="The image is pituitary tumor"
print(preds)

# Model Saving on directory

model.save("../working/densenet101.h5")
y_pred= model.predict(validation_data)
print(y_pred)
y_pred= np.argmax(y_pred, axis=1)

```

Conclusion

Please feel free to ask in the comment section if you have any confusion or questions.

Here are some of the contributions I've made on Kaggle:

1. [Pie Charts in Python](#)
2. [Scatter plots with Plotly Express](#)
3. [X-ray Image Classification using Transfer Learning](#)
4. [Flowers Classification by Using VGG16 Model 🌸🌸](#)
5. [Car Brand Prediction's by Using ResNet50 Model](#)
6. [Image Preprocessing-Morphological Analysis & Kernel](#)
7. [Image Similarity Index \(SSIM analysis\)](#)
8. [Image Preprocessing- Image Transformation & OpenCV](#)