

```
!pip install tensorflow matplotlib numpy scikit-learn seaborn kaggle -q
```

```
import os
import shutil
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.applications import ResNet50, EfficientNetB0, InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    confusion_matrix, accuracy_score,
    precision_score, recall_score,
    f1_score, classification_report
)

print('TensorFlow version:', tf.__version__)
print('All libraries imported successfully!')
```

TensorFlow version: 2.19.0
All libraries imported successfully!

```
from google.colab import files

# Upload your kaggle.json API key
uploaded = files.upload()

# Setup Kaggle credentials
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# Download dataset
!kaggle datasets download -d masoudnickparvar/brain-tumor-mri-dataset

# Unzip
!unzip -q brain-tumor-mri-dataset.zip

print('Dataset downloaded successfully!')
```

Choose files kaggle (1).json
kaggle (1).json(application/json) - 67 bytes, last modified: 28/03/2026 - 100% done
 Saving kaggle (1).json to kaggle (1).json
 cp: cannot stat 'kaggle.json': No such file or directory
 chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
 Dataset URL: <https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset>
 License(s): Attribution 4.0 International (CC BY 4.0)
 Downloading brain-tumor-mri-dataset.zip to /content
 100% 157M/157M [00:01<00:00, 108MB/s]
 Dataset downloaded successfully!

```
# Merge Training and Testing folders into one
original_dataset_dir = 'brain_tumor_dataset_merged'
os.makedirs(original_dataset_dir, exist_ok=True)

for split in ['Training', 'Testing']:
    split_path = split
    if not os.path.exists(split_path):
        continue
    for class_name in os.listdir(split_path):
        class_path = os.path.join(split_path, class_name)
        if not os.path.isdir(class_path):
            continue
        dest = os.path.join(original_dataset_dir, class_name)
        os.makedirs(dest, exist_ok=True)
        for img in os.listdir(class_path):
            src = os.path.join(class_path, img)
            dst = os.path.join(dest, f'{split}_{img}')
            shutil.copy(src, dst)

print('Classes found:')
for cls in os.listdir(original_dataset_dir):
```

```
count = len(os.listdir(os.path.join(original_dataset_dir, cls)))
print(f' {cls}: {count} images')
```

```
Classes found:
meningioma: 1800 images
notumor: 1800 images
glioma: 1800 images
pituitary: 1800 images
```

```
base_dir      = 'dataset'
train_dir     = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir      = os.path.join(base_dir, 'test')

for d in [train_dir, validation_dir, test_dir]:
    os.makedirs(d, exist_ok=True)

train_ratio, validation_ratio, test_ratio = 0.7, 0.1, 0.2

def split_and_copy_files(class_dir, dest_base_dir):
    all_images = [
        os.path.join(class_dir, f) for f in os.listdir(class_dir)
        if os.path.isfile(os.path.join(class_dir, f))
    ]

    n = len(all_images)
    class_name = os.path.basename(class_dir)

    if n < 3:
        dest_dir = os.path.join(dest_base_dir, 'train', class_name)
        os.makedirs(dest_dir, exist_ok=True)
        for file in all_images:
            shutil.copy(file, dest_dir)
        return

    train_images, temp_images = train_test_split(
        all_images, test_size=(1 - train_ratio), random_state=42)

    if len(temp_images) < 2:
        validation_images = temp_images
        test_images = []
    else:
        validation_images, test_images = train_test_split(
            temp_images,
            test_size=test_ratio / (validation_ratio + test_ratio),
            random_state=42)

    for images, folder_name in zip(
        [train_images, validation_images, test_images],
        ['train', 'validation', 'test']
    ):
        if not images:
            continue
        dest_dir = os.path.join(dest_base_dir, folder_name, class_name)
        os.makedirs(dest_dir, exist_ok=True)
        for file in images:
            shutil.copy(file, dest_dir)

# Apply to all class subfolders
for class_dir in os.listdir(original_dataset_dir):
    full_class_dir = os.path.join(original_dataset_dir, class_dir)
    if os.path.isdir(full_class_dir):
        split_and_copy_files(full_class_dir, base_dir)

# Verify split
print('Dataset split completed!')
for split in ['train', 'validation', 'test']:
    split_path = os.path.join(base_dir, split)
    if not os.path.exists(split_path):
        continue
    total = sum(
        len(os.listdir(os.path.join(split_path, c)))
        for c in os.listdir(split_path)
        if os.path.isdir(os.path.join(split_path, c))
    )
    print(f' {split}: {total} images')
```

```
Dataset split completed!
train: 5036 images
validation: 720 images
test: 1444 images
```

```
def create_data_generators(train_dir, val_dir, test_dir,
                           img_size=(240, 240), batch_size=32):

    # Data augmentation for training dataset
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=360,
        horizontal_flip=True,
        vertical_flip=True,
        width_shift_range=0.1,
        height_shift_range=0.1,
    )

    # Only rescale for validation and testing
    val_test_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=img_size,
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=True
    )

    val_generator = val_test_datagen.flow_from_directory(
        val_dir,
        target_size=img_size,
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=False
    )

    test_generator = val_test_datagen.flow_from_directory(
        test_dir,
        target_size=img_size,
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=False
    )

    return train_generator, val_generator, test_generator

train_gen, val_gen, test_gen = create_data_generators(
    'dataset/train',
    'dataset/validation',
    'dataset/test'
)

input_shape = (240, 240, 3)
num_classes = len(train_gen.class_indices)
labels = list(test_gen.class_indices.keys())
print('Classes:', labels)
```

```
Found 5036 images belonging to 4 classes.
Found 720 images belonging to 4 classes.
Found 1444 images belonging to 4 classes.
Classes: ['glioma', 'meningioma', 'notumor', 'pituitary']
```

```
def evaluate_model(model_name, y_true, y_pred):

    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted', zero_division=0)
    recall = recall_score(y_true, y_pred, average='weighted', zero_division=0)
    f1 = f1_score(y_true, y_pred, average='weighted', zero_division=0)
    cm = confusion_matrix(y_true, y_pred)

    print(f'\n===== {model_name} Results =====')
    print('Accuracy: ', accuracy)
    print('Precision:', precision)
    print('Recall: ', recall)
    print('F1-score: ', f1)

    print('\nClassification Report:')
    print(classification_report(y_true, y_pred,
                               target_names=labels,
                               zero_division=0))

    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Label')
```

```
plt.ylabel('True Label')
plt.title(f'Confusion Matrix - {model_name}')
plt.tight_layout()
plt.show()
```

```
def build_cnn_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D(2, 2),

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

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

        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model

model = build_cnn_model(input_shape, num_classes)
history = model.fit(train_gen, epochs=10, validation_data=val_gen)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
158/158 ————— 744s 5s/step - accuracy: 0.5185 - loss: 1.1058 - val_accuracy: 0.5917 - val_loss: 0.9733
Epoch 2/10
158/158 ————— 727s 5s/step - accuracy: 0.6315 - loss: 0.8994 - val_accuracy: 0.6278 - val_loss: 0.9101
Epoch 3/10
158/158 ————— 722s 5s/step - accuracy: 0.6865 - loss: 0.8026 - val_accuracy: 0.6528 - val_loss: 0.8272
Epoch 4/10
158/158 ————— 743s 5s/step - accuracy: 0.6956 - loss: 0.7883 - val_accuracy: 0.6778 - val_loss: 0.8088
Epoch 5/10
158/158 ————— 718s 5s/step - accuracy: 0.7045 - loss: 0.7590 - val_accuracy: 0.6694 - val_loss: 0.8653
Epoch 6/10
158/158 ————— 733s 5s/step - accuracy: 0.7214 - loss: 0.7185 - val_accuracy: 0.7111 - val_loss: 0.7332
Epoch 7/10
158/158 ————— 726s 5s/step - accuracy: 0.7210 - loss: 0.7152 - val_accuracy: 0.7042 - val_loss: 0.7817
Epoch 8/10
158/158 ————— 727s 5s/step - accuracy: 0.7246 - loss: 0.6922 - val_accuracy: 0.6583 - val_loss: 0.8286
Epoch 9/10
158/158 ————— 734s 5s/step - accuracy: 0.7313 - loss: 0.6786 - val_accuracy: 0.6333 - val_loss: 0.9763
Epoch 10/10
158/158 ————— 718s 5s/step - accuracy: 0.7361 - loss: 0.6699 - val_accuracy: 0.7111 - val_loss: 0.7257
```

```
y_pred = model.predict(test_gen)
y_true = test_gen.classes
y_pred_classes = np.argmax(y_pred, axis=1)
evaluate_model('CNN', y_true, y_pred_classes)
```

46/46 ————— 54s 1s/step

===== CNN Results =====

Accuracy: 0.7008310249307479

Precision: 0.7443072168856479

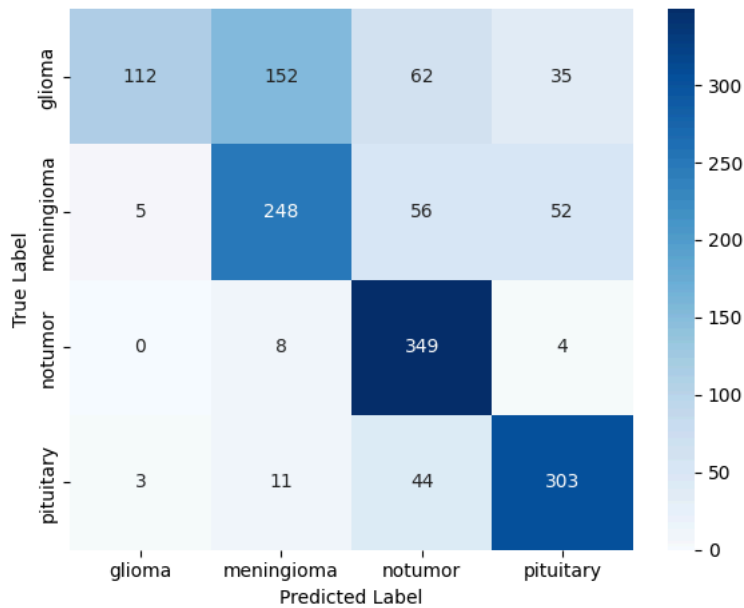
Recall: 0.7008310249307479

F1-score: 0.6761754059531871

Classification Report:

	precision	recall	f1-score	support
glioma	0.93	0.31	0.47	361
meningioma	0.59	0.69	0.64	361
notumor	0.68	0.97	0.80	361
pituitary	0.77	0.84	0.80	361
accuracy			0.70	1444
macro avg	0.74	0.70	0.68	1444
weighted avg	0.74	0.70	0.68	1444

Confusion Matrix - CNN



```
def build_resnet50_model(input_shape, num_classes):
    base_model = ResNet50(weights='imagenet', include_top=False,
                           input_shape=input_shape)
    base_model.trainable = False

    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

```
model = build_resnet50_model(input_shape, num_classes)
history = model.fit(train_gen, epochs=5, validation_data=val_gen)
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_1_94765736/94765736 ————— 1s 0us/step

```
Epoch 1/5
158/158 ————— 1269s 8s/step - accuracy: 0.3934 - loss: 1.2644 - val_accuracy: 0.5222 - val_loss: 1.2030
Epoch 2/5
158/158 ————— 1250s 8s/step - accuracy: 0.4061 - loss: 1.2139 - val_accuracy: 0.5306 - val_loss: 1.1840
Epoch 3/5
158/158 ————— 1247s 8s/step - accuracy: 0.4456 - loss: 1.1897 - val_accuracy: 0.5264 - val_loss: 1.1610
Epoch 4/5
158/158 ————— 1305s 8s/step - accuracy: 0.4404 - loss: 1.1813 - val_accuracy: 0.4597 - val_loss: 1.1645
Epoch 5/5
158/158 ————— 1259s 8s/step - accuracy: 0.4633 - loss: 1.1612 - val_accuracy: 0.5556 - val_loss: 1.1435
```

```
y_pred = model.predict(test_gen)
y_true = test_gen.classes
```

```
y_pred_classes = np.argmax(y_pred, axis=1)
evaluate_model('ResNet50', y_true, y_pred_classes)
```

46/46 ————— 127s 3s/step

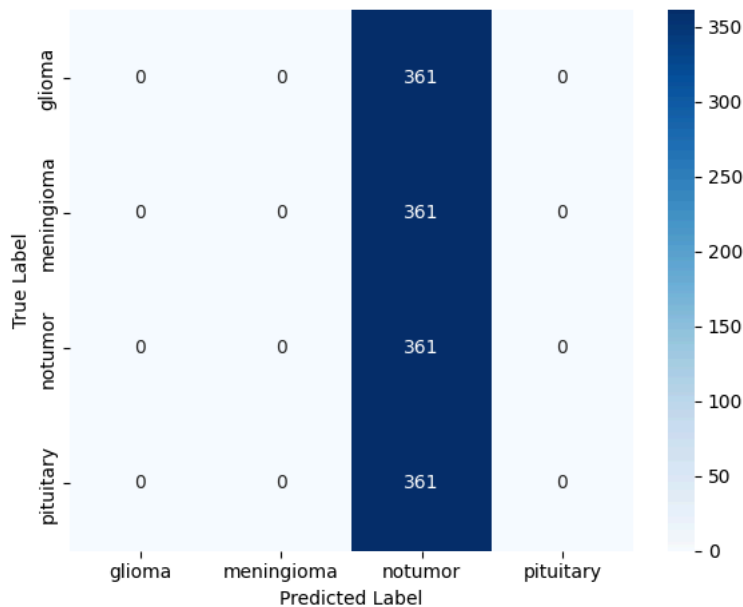
===== ResNet50 Results =====

Accuracy: 0.25
Precision: 0.0625
Recall: 0.25
F1-score: 0.1

Classification Report:

	precision	recall	f1-score	support
glioma	0.00	0.00	0.00	361
meningioma	0.00	0.00	0.00	361
notumor	0.25	1.00	0.40	361
pituitary	0.00	0.00	0.00	361
accuracy			0.25	1444
macro avg	0.06	0.25	0.10	1444
weighted avg	0.06	0.25	0.10	1444

Confusion Matrix - ResNet50



```
def build_efficientnetb0_model(input_shape, num_classes):
    base_model = EfficientNetB0(weights='imagenet', include_top=False,
                                input_shape=input_shape)
    base_model.trainable = False

    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

```
model = build_efficientnetb0_model(input_shape, num_classes)
history = model.fit(train_gen, epochs=5, validation_data=val_gen)
```

Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5

16705208/16705208 ————— 0s 0us/step

Epoch 1/5

158/158 ————— 565s 3s/step - accuracy: 0.2383 - loss: 1.3953 - val_accuracy: 0.2500 - val_loss: 1.3863

Epoch 2/5

158/158 ————— 548s 3s/step - accuracy: 0.2458 - loss: 1.3865 - val_accuracy: 0.2500 - val_loss: 1.3863

Epoch 3/5

158/158 ————— 545s 3s/step - accuracy: 0.2446 - loss: 1.3864 - val_accuracy: 0.2500 - val_loss: 1.3863

Epoch 4/5

158/158 ————— 543s 3s/step - accuracy: 0.2399 - loss: 1.3864 - val_accuracy: 0.2500 - val_loss: 1.3863

Epoch 5/5

158/158 ————— 569s 4s/step - accuracy: 0.2458 - loss: 1.3864 - val_accuracy: 0.2500 - val_loss: 1.3863

```
y_pred = model.predict(test_gen)
y_true = test_gen.classes
y_pred_classes = np.argmax(y_pred, axis=1)
evaluate_model('EfficientNetB0', y_true, y_pred_classes)
```

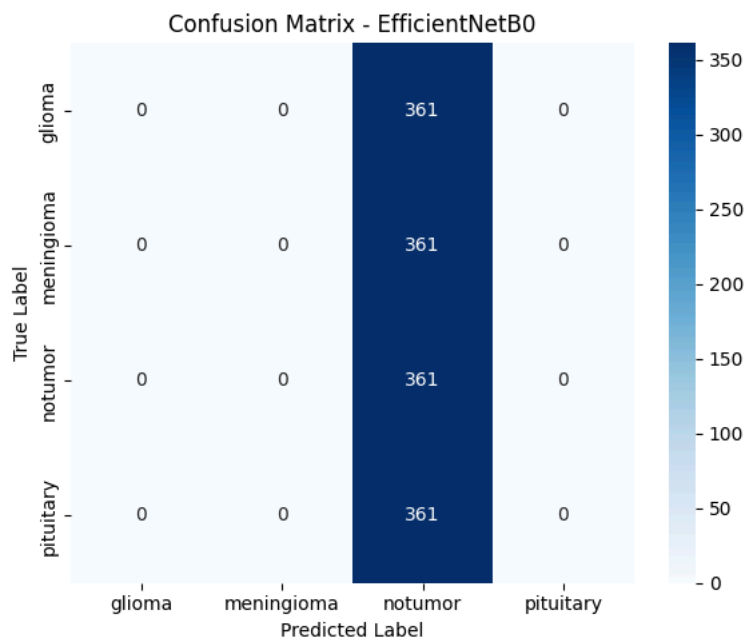
46/46 120s 3s/step

```
===== EfficientNetB0 Results =====
```

```
Accuracy: 0.25
Precision: 0.0625
Recall: 0.25
F1-score: 0.1
```

Classification Report:

	precision	recall	f1-score	support
glioma	0.00	0.00	0.00	361
meningioma	0.00	0.00	0.00	361
notumor	0.25	1.00	0.40	361
pituitary	0.00	0.00	0.00	361
accuracy			0.25	1444
macro avg	0.06	0.25	0.10	1444
weighted avg	0.06	0.25	0.10	1444



```
def build_inceptionv3_model(input_shape, num_classes):
    base_model = InceptionV3(weights='imagenet', include_top=False,
                               input_shape=input_shape)
    base_model.trainable = False

    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

```
model = build_inceptionv3_model(input_shape, num_classes)
history = model.fit(train_gen, epochs=5, validation_data=val_gen)
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_data_format.h5 87910968/87910968 — 1s 0us/step

	Epoch	Time	6s/step	accuracy	loss	val_accuracy	val_loss
	Epoch 1/5						
	158/158	912s	6s/step	- accuracy: 0.6610	- loss: 0.8335	- val_accuracy: 0.7514	- val_loss: 0.6114
	Epoch 2/5						
	158/158	900s	6s/step	- accuracy: 0.7591	- loss: 0.6044	- val_accuracy: 0.7333	- val_loss: 0.5957
	Epoch 3/5						
	158/158	896s	6s/step	- accuracy: 0.7840	- loss: 0.5574	- val_accuracy: 0.7750	- val_loss: 0.5669
	Epoch 4/5						
	158/158	934s	6s/step	- accuracy: 0.7971	- loss: 0.5124	- val_accuracy: 0.7681	- val_loss: 0.5177
	Epoch 5/5						
	158/158	940s	6s/step	- accuracy: 0.8125	- loss: 0.4919	- val_accuracy: 0.8111	- val_loss: 0.4853

```

y_pred      = model.predict(test_gen)
y_true      = test_gen.classes
y_pred_classes = np.argmax(y_pred, axis=1)
evaluate_model('InceptionV3', y_true, y_pred_classes)

```

46/46 ————— 216s 5s/step

===== InceptionV3 Results =====

Accuracy: 0.8303324099722992

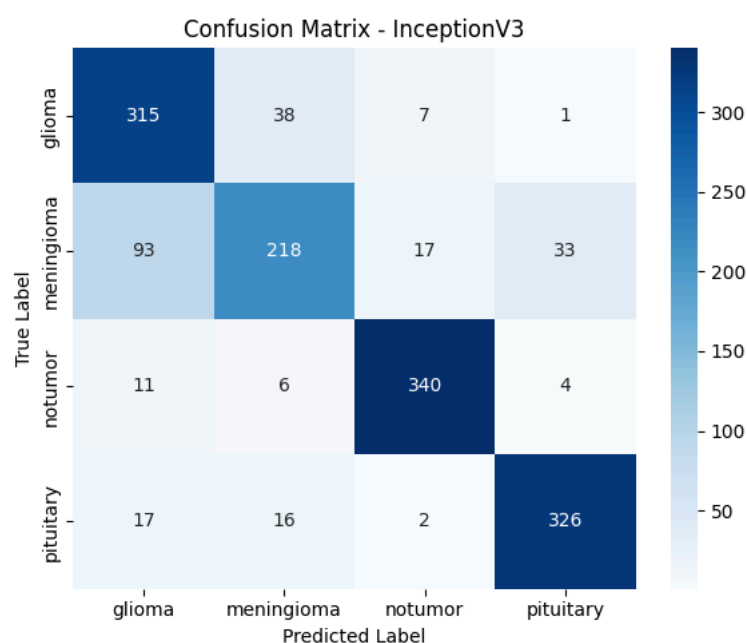
Precision: 0.8328039675822407

Recall: 0.8303324099722992

F1-score: 0.826860365300128

Classification Report:

	precision	recall	f1-score	support
glioma	0.72	0.87	0.79	361
meningioma	0.78	0.60	0.68	361
notumor	0.93	0.94	0.94	361
pituitary	0.90	0.90	0.90	361
accuracy			0.83	1444
macro avg	0.83	0.83	0.83	1444
weighted avg	0.83	0.83	0.83	1444



Step 8: Convert Notebook to PDF

```

!apt-get install -y texlive-xetex texlive-fonts-recommended texlive-plain-generic -q
!pip install nbconvert -q
print('Done. Run the next cell to export PDF.')

```

```
/sbin/ldconfig.real: /usr/local/lib/libumf.so.1 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libur_adapter_level_zero_v2.so.0 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libtbbmalloc.so.2 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libtcm.so.1 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libtcm_debug.so.1 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libtbbbind_2_5.so.3 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libhwloc.so.15 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libtbbbind.so.3 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libur_loader.so.0 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libtbbbind_2_0.so.3 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libtbbmalloc_proxy.so.2 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libtbb.so.12 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libur_adapter_level_zero.so.0 is not a symbolic link

Processing triggers for tex-common (6.17) ...
Running updpmap-sys. This may take some time... done.
Running mktexlsr /var/lib/texmf ... done.
Building format(s) --all.
    This may take some time... done.
Done. Run the next cell to export PDF.
```

```
import time, glob
time.sleep(2)

notebooks = glob.glob('/content/*.ipynb')
if notebooks:
    nb_path = notebooks[0]
    !jupyter nbconvert --to pdf "{nb_path}" --output /content/Assignment_BrainTumor.pdf
    from google.colab import files
```