```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import (Input, Conv2D, MaxPooling2D,
                                     Flatten, Dense, Dropout)
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, Conf
```

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

```

```
!pip install --upgrade kaggle
!kaggle --version

```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.11/dist-packages (1.7.4.
Requirement already satisfied: bleach in /usr/local/lib/python3.11/dist-packages (from k
Requirement already satisfied: certifi>=14.05.14 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: charset-normalizer in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: idna in /usr/local/lib/python3.11/dist-packages (from kag
Requirement already satisfied: protobuf in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: python-dateutil>=2.5.3 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: python-slugify in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: setuptools>=21.0.0 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: text-unidecode in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from kag
Requirement already satisfied: urllib3>=1.15.1 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (
Kaggle API 1.7.4.2
```

```
!kaggle datasets download -d masoudnickparvar/brain-tumor-mri-dataset -p /content

```

```
Dataset URL: https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset
License(s): CC0-1.0
```

```
def get_class_paths(base_path):
    classes, paths = [], []
    for label in os.listdir(base_path):
        full_dir = os.path.join(base_path, label)
        if os.path.isdir(full_dir):
            for fname in os.listdir(full_dir):
                classes.append(label)
                paths.append(os.path.join(full_dir, fname))
    return pd.DataFrame({"Class": classes, "Path": paths})

tr_df = get_class_paths("Training")
ts_df = get_class_paths("Testing")
```

```
#Split Testing into validation & test 50/50
valid_df, ts_df = train_test_split(
    ts_df, stratify=ts_df["Class"], train_size=0.5, random_state=42
)
#RANDOM STATE Sets a seed for reproducibility
#ensuring that the split is consistent across different runs
#every time run the code
# the data will be split in the same way ensuring consistency in result
batch_size = 16
img_size = (224, 224)

train_datagen = ImageDataGenerator(rescale=1/255, brightness_range=(0.8,1.2))
test_datagen  = ImageDataGenerator(rescale=1/255)

tr_gen = train_datagen.flow_from_dataframe(
    tr_df, x_col="Path", y_col="Class",
    target_size=img_size, batch_size=batch_size
)
valid_gen = train_datagen.flow_from_dataframe(
    valid_df, x_col="Path", y_col="Class",
    target_size=img_size, batch_size=batch_size
)
ts_gen = test_datagen.flow_from_dataframe(
    ts_df, x_col="Path", y_col="Class",
    target_size=img_size, batch_size=batch_size, shuffle=False
)
```

```
Found 5712 validated image filenames belonging to 4 classes.
Found 655 validated image filenames belonging to 4 classes.
Found 656 validated image filenames belonging to 4 classes.
```

```
#VGG-19 Architecture

inputs = Input(shape=(224, 224, 3))

```

```
# Block 1 (2 Conv64 + Pool)
x = Conv2D(64, (3,3), activation="relu", padding="same")(inputs)
x = Conv2D(64, (3,3), activation="relu", padding="same")(x)
x = MaxPooling2D((2,2), strides=2)(x)

# Block 2 (2 Conv128 + Pool)
x = Conv2D(128, (3,3), activation="relu", padding="same")(x)
x = Conv2D(128, (3,3), activation="relu", padding="same")(x)
x = MaxPooling2D((2,2), strides=2)(x)

# Block 3 (4 Conv256 + Pool)
for _ in range(4):
    x = Conv2D(256, (3,3), activation="relu", padding="same")(x)
x = MaxPooling2D((2,2), strides=2)(x)

# Block 4 (4 Conv512 + Pool)
for _ in range(4):
    x = Conv2D(512, (3,3), activation="relu", padding="same")(x)
x = MaxPooling2D((2,2), strides=2)(x)

# Block 5 (4 Conv512 + Pool)
for _ in range(4):
    x = Conv2D(512, (3,3), activation="relu", padding="same")(x)
x = MaxPooling2D((2,2), strides=2)(x)



```

```
# flatten layer
x = Flatten()(x)
x = Dense(4096, activation="relu")(x)
x = Dense(4096, activation="relu")(x) #Stacking dense layers enables the model to


outputs = Dense(len(tr_gen.class_indices), activation="softmax")(x)

model = Model(inputs, outputs, name="VGG19_Custom")
model.compile(
    optimizer=Adam(learning_rate=1e-4),#computes adaptive learning rates for each
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
model.summary()

```

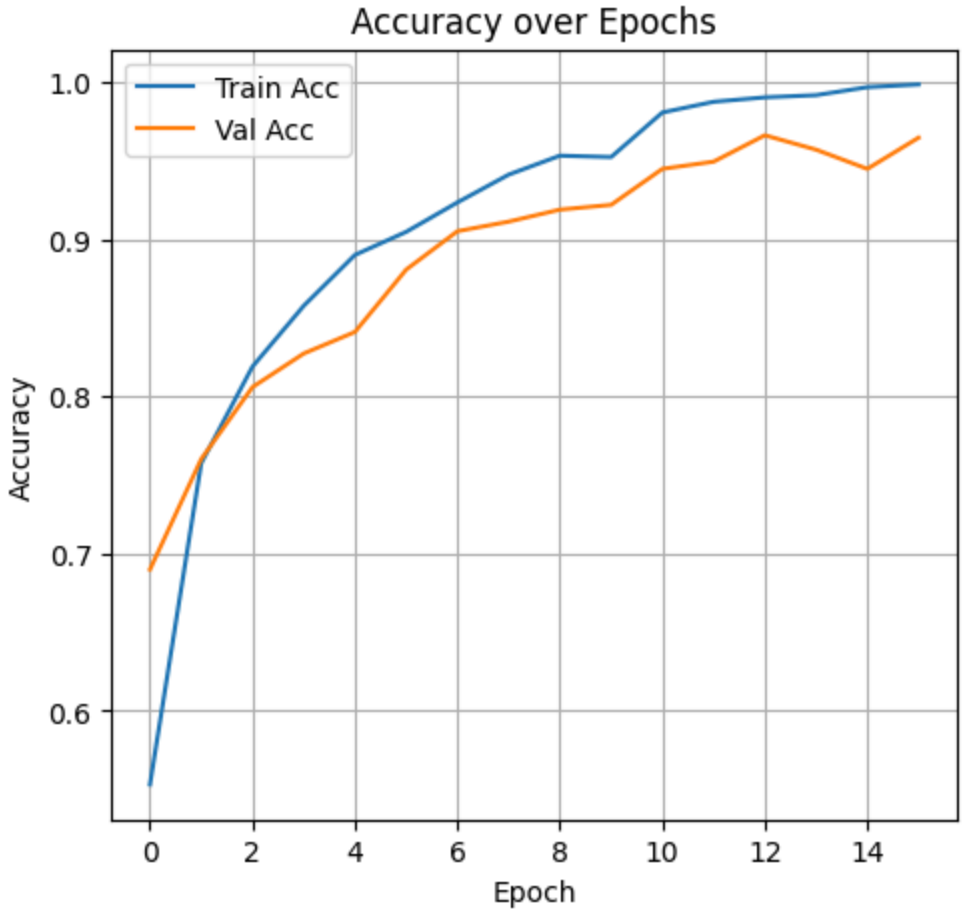
Model: "VGG19_Custom"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
conv2d (Conv2D)	(None, 224, 224, 64)	1,792
conv2d_1 (Conv2D)	(None, 224, 224, 64)	36,928
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_2 (Conv2D)	(None, 112, 112, 128)	73,856
conv2d_3 (Conv2D)	(None, 112, 112, 128)	147,584
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 128)	0
conv2d_4 (Conv2D)	(None, 56, 56, 256)	295,168
conv2d_5 (Conv2D)	(None, 56, 56, 256)	590,080
conv2d_6 (Conv2D)	(None, 56, 56, 256)	590,080
conv2d_7 (Conv2D)	(None, 56, 56, 256)	590,080
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 256)	0
conv2d_8 (Conv2D)	(None, 28, 28, 512)	1,180,160
conv2d_9 (Conv2D)	(None, 28, 28, 512)	2,359,808
conv2d_10 (Conv2D)	(None, 28, 28, 512)	2,359,808
conv2d_11 (Conv2D)	(None, 28, 28, 512)	2,359,808
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 512)	0
conv2d_12 (Conv2D)	(None, 14, 14, 512)	2,359,808
conv2d_13 (Conv2D)	(None, 14, 14, 512)	2,359,808
conv2d_14 (Conv2D)	(None, 14, 14, 512)	2,359,808
conv2d_15 (Conv2D)	(None, 14, 14, 512)	2,359,808
max_pooling2d_4 (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 4096)	102,764,544
dense_1 (Dense)	(None, 4096)	16,781,312
dense_2 (Dense)	(None, 4)	16,388

```python
#Train model

early_stop  = EarlyStopping(monitor="val_loss", patience=3, restore_best_weights=T
#EarlyStopping Callback set to monitor the val_loss if the validation loss doesn't
reduce_lr   = ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=2, verbos

history = model.fit(
    tr_gen,
    epochs=20,
    validation_data=valid_gen,
    callbacks=[early_stop, reduce_lr]
)
```

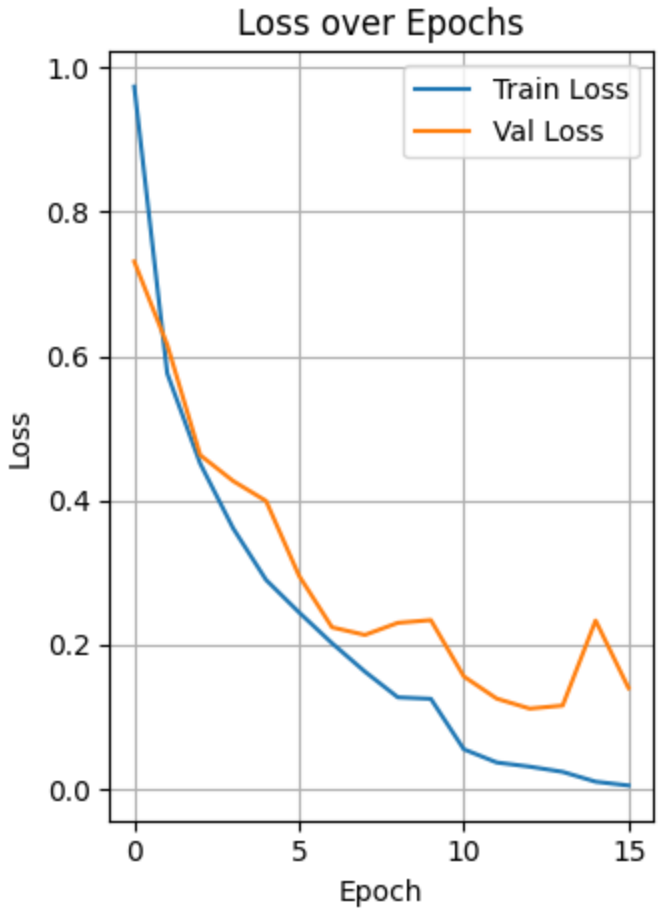
```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adap
  self._warn_if_super_not_called()
Epoch 1/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 182s 400ms/step - accuracy: 0.4367 - loss: 1.1712 - val_acc
Epoch 2/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 129s 361ms/step - accuracy: 0.7414 - loss: 0.6064 - val_acc
Epoch 3/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 129s 361ms/step - accuracy: 0.7981 - loss: 0.4822 - val_acc
Epoch 4/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 129s 361ms/step - accuracy: 0.8532 - loss: 0.3793 - val_acc
Epoch 5/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 142s 361ms/step - accuracy: 0.8897 - loss: 0.2988 - val_acc
Epoch 6/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 129s 360ms/step - accuracy: 0.9116 - loss: 0.2332 - val_acc
Epoch 7/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 129s 360ms/step - accuracy: 0.9304 - loss: 0.1821 - val_acc
Epoch 8/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 129s 361ms/step - accuracy: 0.9448 - loss: 0.1559 - val_acc
Epoch 9/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 128s 359ms/step - accuracy: 0.9567 - loss: 0.1206 - val_acc
Epoch 10/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 0s 344ms/step - accuracy: 0.9517 - loss: 0.1300
Epoch 10: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
357/357 ━━━━━━━━━━━━━━━━━━━━ 128s 359ms/step - accuracy: 0.9517 - loss: 0.1300 - val_acc
Epoch 11/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 128s 360ms/step - accuracy: 0.9757 - loss: 0.0692 - val_acc
Epoch 12/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 129s 360ms/step - accuracy: 0.9877 - loss: 0.0347 - val_acc
Epoch 13/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 129s 360ms/step - accuracy: 0.9927 - loss: 0.0250 - val_acc
Epoch 14/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 141s 358ms/step - accuracy: 0.9934 - loss: 0.0179 - val_acc
Epoch 15/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 0s 344ms/step - accuracy: 0.9986 - loss: 0.0069
Epoch 15: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-05.
357/357 ━━━━━━━━━━━━━━━━━━━━ 128s 358ms/step - accuracy: 0.9986 - loss: 0.0069 - val_acc
Epoch 16/20
357/357 ━━━━━━━━━━━━━━━━━━━━ 128s 359ms/step - accuracy: 0.9979 - loss: 0.0052 - val_acc
```

```
# Plot accuracy loss
plt.figure(figsize=(12,5))

# Accuracy
plt.subplot(1,2,1)
plt.plot(history.history["accuracy"], label="Train Acc")
plt.plot(history.history["val_accuracy"], label="Val Acc")
plt.title("Accuracy over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)


```




```
# Loss
plt.subplot(1,2,2)
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Val Loss")
plt.title("Loss over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
```

```
plt.tight_layout()
plt.show()

model.save("vgg19_brain_tumor_mri.h5")

```

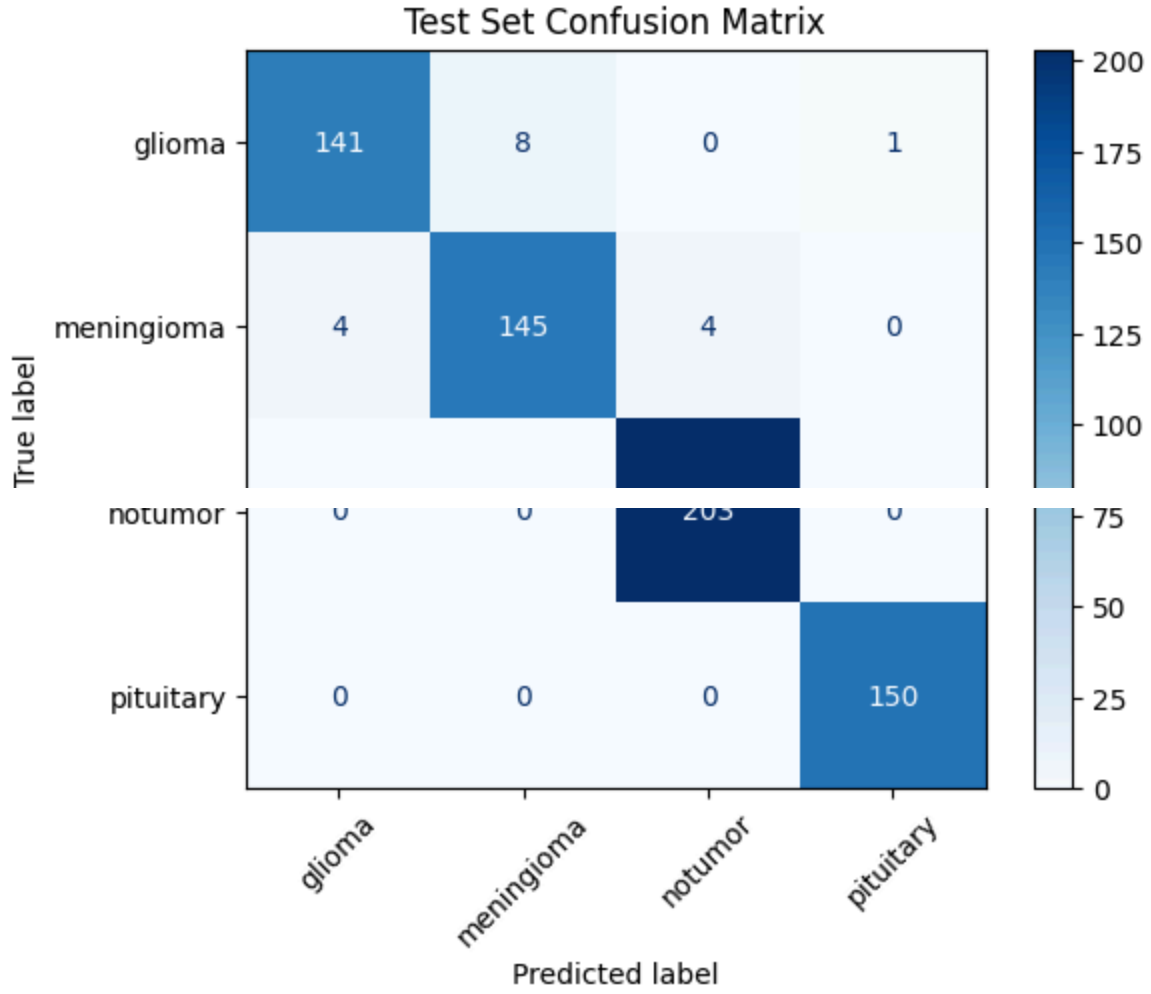



```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.savi
```

```
# Evaluate on test set

# predictions
y_true = ts_gen.classes
y_pred = np.argmax(model.predict(ts_gen), axis=1)

# confusion Matrix
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(cm, display_labels=list(tr_gen.class_indices.keys())

plt.figure(figsize=(7,6))
disp.plot(cmap="Blues", xticks_rotation=45)
plt.title("Test Set Confusion Matrix")
plt.grid(False)
plt.show()
```

```
16
17 print("Classification Report:\n")
18 print(classification_report(y_true, y_pred, target_names=list(tr_gen.class_indices
19
```

```
41/41 ━━━━━━━━━━━━━━━━━━━━ 6s 124ms/step
<Figure size 700x600 with 0 Axes>
```




```
Classification Report:

              precision    recall  f1-score   support

      glioma       0.97      0.94      0.96       150
  meningioma       0.95      0.95      0.95       153
     notumor       0.98      1.00      0.99       203
   pituitary       0.99      1.00      1.00       150

    accuracy                           0.97       656
   macro avg       0.97      0.97      0.97       656
weighted avg       0.97      0.97      0.97       656
```

```
1 #DISCUSSION
2 The role of each layer in the architecture
3 Input layer: Accepts images of size 224×224×3
4  CONV LAYER:3×3 filters to the input image, detecting various features su
```

```
ReLU activation: After each convolutional operation ReLU activation funct
Max pool:Max pooling involves sliding a fixed-size window 2x2 over the ir
Fully connected layer: At the end of the network fully connected layers i
Softmax layer:Softmax layer outputs a probability distribution over the p


Why VGG-19 uses small filters (3×3) [1]
Small filters allow the network to capture detailed spatial hierarchies i
```

```
Start coding or generate with AI.
```