# Classifier Model Training

1) Libraries

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import os
```

2) Paths for Images

```
#
======================================================================
=
# Step 1: Define Paths (Update with your actual paths)
#
======================================================================
=
 # Replace with your folder path
base_dir = r'C:\Users\BAPS\Documents\Classifier_Brain_Tumour'
train_dir = os.path.join(base_dir, 'Training')
test_dir = os.path.join(base_dir, 'Testing')

# Verify paths exist
assert os.path.exists(train_dir), f"Directory {train_dir} not
found!"
assert os.path.exists(test_dir), f"Directory {test_dir} not found!"
```

3) Loading the Dataset

```
#
======================================================================
=
# Step 2: Load Binary Dataset (Tumor vs. No Tumor)
```

```
#
===================================================================
=
# Load training data
train_ds = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    labels='inferred',
    label_mode='binary',  # Binary labels (tumor=1, no_tumor=0)
    image_size=(224, 224),
    batch_size=32,
    shuffle=True,
    color_mode='grayscale'  # Use 'rgb' if images are color
)

# Load testing data
test_ds = tf.keras.utils.image_dataset_from_directory(
    test_dir,
    labels='inferred',
    label_mode='binary',
    image_size=(224, 224),
    batch_size=32,
    shuffle=False,
    color_mode='grayscale'
)

#
===================================================================
=
#
===================================================================
=
```

Output:

```
Found 5712 files belonging to 2 classes.
Found 1311 files belonging to 2 classes.
```

4) Building the model

```python
# Step 3: Build Binary Classification Model
#
=====================================================================
=
def build_model():
    model = models.Sequential([

        layers.Input(shape=(224, 224, 1)),  # 1 channel for
grayscale
        # Preprocessing layers (resize + normalize)
        layers.Resizing(224, 224),  # Handles variable input sizes
        layers.Rescaling(1./255),   # Normalize pixels [0, 255] →
[0, 1]

        # Feature extraction
        layers.Conv2D(32, (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(64, (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(128, (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),

        # Classifier
        layers.Flatten(),
        layers.Dense(256, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(1, activation='sigmoid')  # Binary output
    ])
    return model

model = build_model()
model.summary()
```

Output:

```
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                        ┃ Output Shape              ┃
Param #  ┃
```

| | | |
|---|---|---|
| resizing_1 (Resizing) | (None, 224, 224, 1) | 0 |
| rescaling_1 (Rescaling) | (None, 224, 224, 1) | 0 |
| conv2d_3 (Conv2D) | (None, 222, 222, 32) | 320 |
| max_pooling2d_3 (MaxPooling2D) | (None, 111, 111, 32) | 0 |
| conv2d_4 (Conv2D) | (None, 109, 109, 64) | 18,496 |
| max_pooling2d_4 (MaxPooling2D) | (None, 54, 54, 64) | 0 |
| conv2d_5 (Conv2D) | (None, 52, 52, 128) | 73,856 |
| max_pooling2d_5 (MaxPooling2D) | (None, 26, 26, 128) | 0 |
| flatten_1 (Flatten) | (None, 86528) | 0 |
| dense_2 (Dense) | (None, 256) | 22,151,424 |
| dropout_1 (Dropout) | (None, 256) | 0 |

```
| dense_3 (Dense)                    | (None, 1)                 |
257 |
└────────────────────────────────────┴───────────────────────────┴─────────
────────┘
```

Total params: 22,244,353 (84.86 MB)

Trainable params: 22,244,353 (84.86 MB)

Non-trainable params: 0 (0.00 B)

5) Compile and Train

```
# Step 4: Compile and Train
#
======================================================================
=
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss='binary_crossentropy',
    metrics=['accuracy',
tf.keras.metrics.Precision(name='precision'),
tf.keras.metrics.Recall(name='recall')]
)

#
======================================================================
=
class_weight = None  # Set based on your data distribution
history = model.fit(
    train_ds,
    validation_data=test_ds,
    epochs=20,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(patience=3,
restore_best_weights=True),
        tf.keras.callbacks.ModelCheckpoint(
            filepath='brain_tumor_model.keras',  # <-- Added here
            monitor='val_loss',
            save_best_only=True,
            verbose=1
        )
```

```
    ],
    class_weight=class_weight
)
```

## 6) Evaluation on Testing Data

```
# Evaluate on test data
test_loss, test_acc, test_precision, test_recall =
model.evaluate(test_ds)
print(f"Test Accuracy: {test_acc:.2f}")
print(f"Test Precision: {test_precision:.2f}")
print(f"Test Recall: {test_recall:.2f}")

# Confusion matrix
y_true = []
y_pred = []
for images, labels in test_ds:
    y_true.extend(labels.numpy())
    y_pred.extend((model.predict(images) > 0.5).astype("int32"))

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:")
print(cm)
```

### Output

```
Test Accuracy: 1.00
Test Precision: 1.00
Test Recall: 1.00

Confusion Matrix:

  [ 405      0 ]
  [   4      902]
```

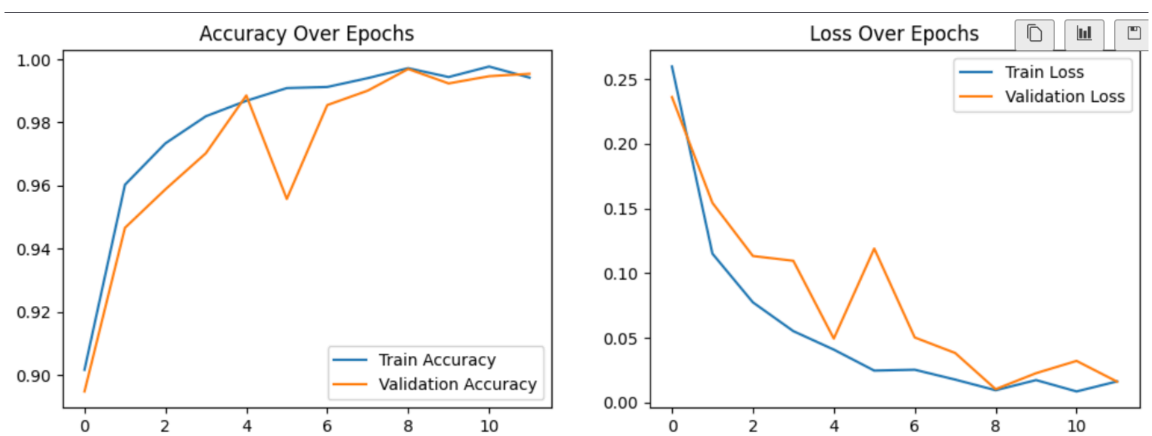
## 7) Training and Validation accuracy and loss

```
import matplotlib.pyplot as plt
# Plot training history
plt.figure(figsize=(12, 4))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
plt.title('Accuracy Over Epochs')
plt.legend()

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Over Epochs')
plt.legend()

plt.show()
```




## 8) Save the Entire Model

```
# Save entire model (including optimizer state)
model.save('final_brain_tumor_model.keras')
```

```python
# Optional: Save training history
import pickle
with open('training_history.pkl', 'wb') as f:
    pickle.dump(history.history, f)
```

9) Function for predicting the image

```python
def predict_tumor(image_path):
    # Load image (match your model's expected format)
    img = tf.keras.utils.load_img(
        image_path,
        color_mode='grayscale',  # Must match your training data
format
        target_size=(224, 224)   # Must match your model's input
size
    )

    # Convert to array and preprocess
    img_array = tf.keras.utils.img_to_array(img)
    img_array = tf.expand_dims(img_array, axis=0)  # Add batch
dimension

    # Make prediction
    prediction = model.predict(img_array)[0][0]

    return "Tumor" if prediction > 0.5 else "No Tumor"
```

10) Function for probability

```python
# Get probability scores
def get_tumor_probability(image_path):
    img = tf.keras.utils.load_img(image_path,
color_mode='grayscale', target_size=(224, 224))
    img_array = tf.keras.utils.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0)
    return model.predict(img_array)[0][0]
```

11) Predicting a tumour

```
test_image_path = "brain tumour prediction sathi.jpg"
print(predict_tumor(test_image_path))
```

Output:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
Tumor
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
Tumor probability: 99.29%
```

12) Predicting a non-tumour image

```
test_image_path = "notumourimage.jpeg"
print(f"Tumor probability:
{get_tumor_probability(test_image_path):.2%}")
```

Output:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
Tumor probability: 0.00%
```