In [2]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
2024-03-24 17:04:45.799744: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.c
c:9261] Unable to register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
2024-03-24 17:04:45.799862: E external/local_xla/xla/stream_executor/cuda/cuda_fft.c
c:607] Unable to register cuFFT factory: Attempting to register factory for plugin c
uFFT when one has already been registered
2024-03-24 17:04:45.964900: E external/local_xla/xla/stream_executor/cuda/cuda_blas.
cc:1515] Unable to register cuBLAS factory: Attempting to register factory for plugi
n cuBLAS when one has already been registered
```

In [3]:
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
)

training_set = train_datagen.flow_from_directory(
        '/kaggle/input/brain-tumour-dataset/Dataset/Training',
        target_size=(224, 224),
        batch_size=64,
        class_mode='categorical')
```

```
Found 5712 images belonging to 4 classes.
```

In [4]:
```
validation_datagen = ImageDataGenerator(rescale=1./255)

validation_set = validation_datagen.flow_from_directory(
        '/kaggle/input/brain-tumour-dataset/Dataset/validation',
        target_size=(224, 224),
        batch_size=64,
        class_mode='categorical',
        shuffle = False)
```

```
Found 1239 images belonging to 4 classes.
```

Building the CNN :

Customising the AlexNET architecture for better result. Here we have used strides=4 in the 3rd,4th and 5th Convolution layers and omitted the last 3rd Pooling Layer (Max) from the architecture, to acheive more efficiency by experiment. (Which is different from the classic AlexNET model)

In [5]:
```
# Initialising the CNN
cnn = tf.keras.models.Sequential()

# Convolutional Layer
cnn.add(tf.keras.layers.Conv2D(filters=96, kernel_size=11, strides=4, padding='same
cnn.add(tf.keras.layers.Lambda(tf.nn.local_response_normalization)) # Normalisation

# Pooling Layer (Max Pooling)
cnn.add(tf.keras.layers.MaxPooling2D(pool_size=3, strides=2))

# 2nd Convolutional Layer
cnn.add(tf.keras.layers.Conv2D(filters=256, kernel_size=5, strides = 4, padding='sa
```

```
cnn.add(tf.keras.layers.Lambda(tf.nn.local_response_normalization)) # Normalisation

# 2nd Pooling Layer (Max Pooling)
cnn.add(tf.keras.layers.MaxPooling2D(pool_size=3, strides=2))

# Convolutional Layer 3
cnn.add(tf.keras.layers.Conv2D(filters = 384, kernel_size = 3, strides=4, padding='

# Convolutional Layer 4
cnn.add(tf.keras.layers.Conv2D(filters = 384, kernel_size = 3, strides=4, padding='

# Convolucnnl Layer 5
cnn.add(tf.keras.layers.Conv2D(filters = 256, kernel_size = 3, strides=4, padding='

# Flattening Layer
cnn.add(tf.keras.layers.Flatten())

# Full Connection
cnn.add(tf.keras.layers.Dense(units=4096,activation="relu")) # 1st Hidden Layer
cnn.add(tf.keras.layers.Dropout(0.5))

cnn.add(tf.keras.layers.Dense(units=4096,activation="relu")) # 2nd Hidden Layer
cnn.add(tf.keras.layers.Dropout(0.5))

cnn.add(tf.keras.layers.Dense(units=4,activation="softmax")) # Output Layer

# Compiling the CNN :
cnn.compile(optimizer="adam", loss = "categorical_crossentropy", metrics=["accuracy
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/layers/convolutional/base_conv.py:
99: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(
```

Training the Model :

In [6]:
```
# Training and Evaluating the CNN :
history = cnn.fit(x = training_set, validation_data = validation_set, epochs=25)
```

```
Epoch 1/25
/opt/conda/lib/python3.10/site-packages/keras/src/trainers/data_adapters/py_dataset_
adapter.py:122: UserWarning: Your `PyDataset` class should call `super().__init__(**
kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing
`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignore
d.
  self._warn_if_super_not_called()
 2/90 ━━━━━━━━━━━━━━━━━━━━ 6s 70ms/step - accuracy: 0.3477 - loss: 1.3849
WARNING: All log messages before absl::InitializeLog() is called are written to STDE
RR
I0000 00:00:1711299928.898236     102 device_compiler.h:186] Compiled cluster using
XLA!  This line is logged at most once for the lifetime of the process.
W0000 00:00:1711299928.920726     102 graph_launch.cc:671] Fallback to op-by-op mode
because memset node breaks graph update
```

```
23/90 ━━━━━━━━━━━━━━━━━━━━ 1:00 908ms/step - accuracy: 0.2929 - loss: 1.4451
W0000 00:00:1711299948.780342     103 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
89/90 ━━━━━━━━━━━━━━━━━━━━ 0s 674ms/step - accuracy: 0.3138 - loss: 1.3627
W0000 00:00:1711299989.933804     103 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
```

```
90/90 ━━━━━━━━━━━━━━━━━━━━ 92s 821ms/step - accuracy: 0.3158 - loss: 1.3591 - val_ac
curacy: 0.6602 - val_loss: 0.8481
Epoch 2/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 24s 247ms/step - accuracy: 0.6900 - loss: 0.7167 - val_ac
curacy: 0.7659 - val_loss: 0.7633
Epoch 3/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 237ms/step - accuracy: 0.8262 - loss: 0.5204 - val_ac
curacy: 0.7869 - val_loss: 0.5701
Epoch 4/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 242ms/step - accuracy: 0.8709 - loss: 0.4036 - val_ac
curacy: 0.8176 - val_loss: 0.5554
Epoch 5/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 24s 244ms/step - accuracy: 0.8885 - loss: 0.3316 - val_ac
curacy: 0.8297 - val_loss: 0.4792
Epoch 6/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 240ms/step - accuracy: 0.9060 - loss: 0.2802 - val_ac
curacy: 0.8386 - val_loss: 0.4303
Epoch 7/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 24s 245ms/step - accuracy: 0.9071 - loss: 0.2718 - val_ac
curacy: 0.8370 - val_loss: 0.4134
Epoch 8/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 235ms/step - accuracy: 0.9271 - loss: 0.2249 - val_ac
curacy: 0.8378 - val_loss: 0.3432
Epoch 9/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 24s 252ms/step - accuracy: 0.9350 - loss: 0.1911 - val_ac
curacy: 0.8797 - val_loss: 0.2910
Epoch 10/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 242ms/step - accuracy: 0.9353 - loss: 0.1712 - val_ac
curacy: 0.9104 - val_loss: 0.2616
Epoch 11/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 240ms/step - accuracy: 0.9297 - loss: 0.1873 - val_ac
curacy: 0.8878 - val_loss: 0.3010
Epoch 12/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 240ms/step - accuracy: 0.9470 - loss: 0.1501 - val_ac
curacy: 0.8878 - val_loss: 0.2847
Epoch 13/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 239ms/step - accuracy: 0.9520 - loss: 0.1190 - val_ac
curacy: 0.9177 - val_loss: 0.2099
Epoch 14/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 238ms/step - accuracy: 0.9722 - loss: 0.0946 - val_ac
curacy: 0.9040 - val_loss: 0.2663
Epoch 15/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 243ms/step - accuracy: 0.9649 - loss: 0.1024 - val_ac
curacy: 0.9266 - val_loss: 0.1841
Epoch 16/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 237ms/step - accuracy: 0.9673 - loss: 0.0950 - val_ac
curacy: 0.9266 - val_loss: 0.2012
Epoch 17/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 245ms/step - accuracy: 0.9761 - loss: 0.0718 - val_ac
curacy: 0.8660 - val_loss: 0.3683
Epoch 18/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 239ms/step - accuracy: 0.9579 - loss: 0.1228 - val_ac
curacy: 0.9330 - val_loss: 0.1827
Epoch 19/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 236ms/step - accuracy: 0.9688 - loss: 0.0862 - val_ac
curacy: 0.9209 - val_loss: 0.2530
```

```
Epoch 20/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 240ms/step - accuracy: 0.9789 - loss: 0.0690 - val_ac
curacy: 0.9233 - val_loss: 0.2558
Epoch 21/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 240ms/step - accuracy: 0.9822 - loss: 0.0530 - val_ac
curacy: 0.9403 - val_loss: 0.1814
Epoch 22/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 239ms/step - accuracy: 0.9887 - loss: 0.0402 - val_ac
curacy: 0.9379 - val_loss: 0.2052
Epoch 23/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 241ms/step - accuracy: 0.9892 - loss: 0.0368 - val_ac
curacy: 0.9516 - val_loss: 0.2174
Epoch 24/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 239ms/step - accuracy: 0.9735 - loss: 0.0889 - val_ac
curacy: 0.9112 - val_loss: 0.2701
Epoch 25/25
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 245ms/step - accuracy: 0.9723 - loss: 0.0830 - val_ac
curacy: 0.9427 - val_loss: 0.2590
```

In [21]:
```
test_datagen = ImageDataGenerator(rescale=1./255)

test_set = test_datagen.flow_from_directory(
        '/kaggle/input/brain-tumour-dataset/Dataset/test',
        target_size=(224, 224),
        batch_size=64,
        class_mode='categorical',
        shuffle=False)
```

```
Found 72 images belonging to 4 classes.
```

In [22]:
```
import numpy as np

y_pred =cnn.predict(test_set)
y_pred = np.argmax(y_pred, axis=1)
y_true = test_set.classes
print(np.concatenate((y_true.reshape(len(y_true),1),y_pred.reshape(len(y_pred),1)),
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/trainers/data_adapters/py_dataset_
adapter.py:122: UserWarning: Your `PyDataset` class should call `super().__init__(**
kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing
`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignore
d.
  self._warn_if_super_not_called()
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [3 3]
```

```
[3 3]
[3 3]
[3 3]
[3 3]
[3 3]
[3 3]
[3 3]
[3 3]
[3 3]
[3 3]
[3 3]
[3 3]
[3 3]
[3 3]
[3 3]
[3 3]
[3 1]]
```

Evaluating the Model :

In [23]:
```
# Confusion Matrix :
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true,y_pred)
print(cm)
```

```
[[18  0  0  0]
 [ 0 17  1  0]
 [ 0  0 18  0]
 [ 0  1  0 17]]
```

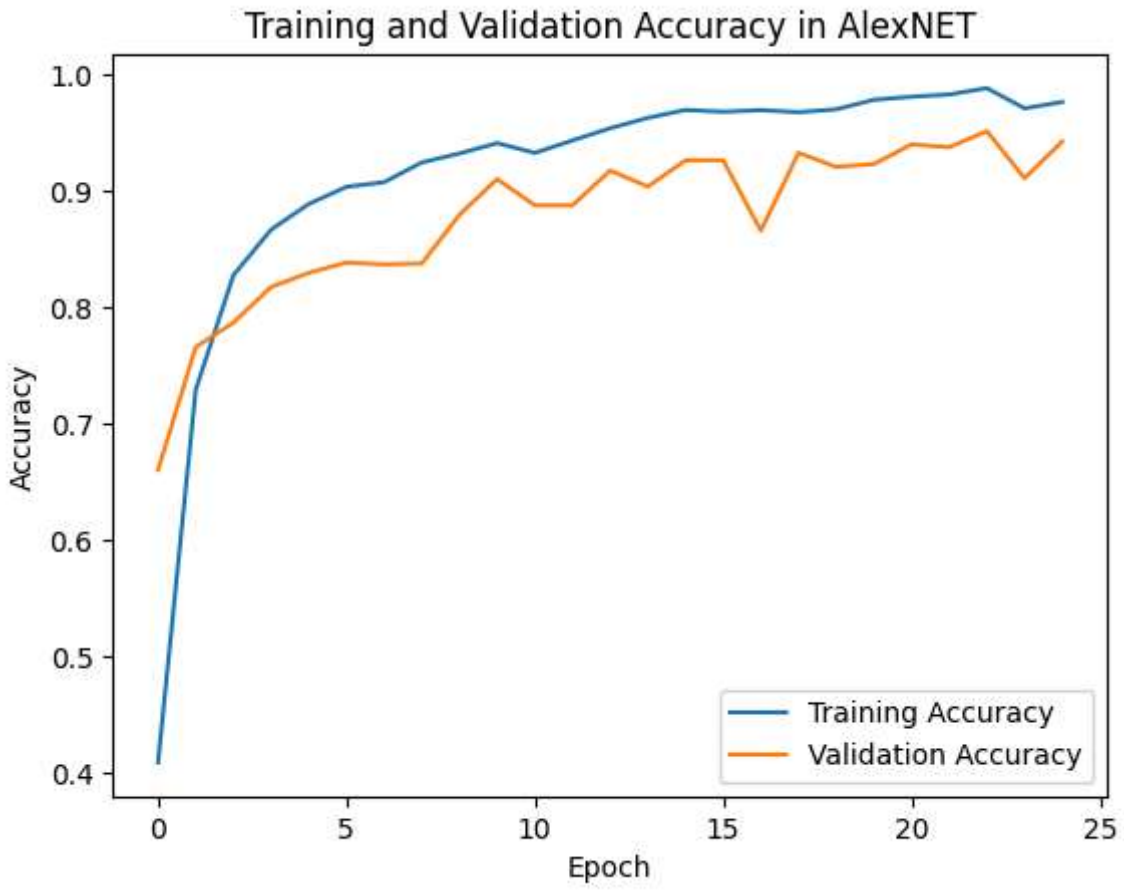
In [24]:
```
# Accuracy :
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_true,y_pred)
print(f"The Accuracy of the model is : {accuracy*100:.2f}%")
```

```
The Accuracy of the model is : 97.22%
```

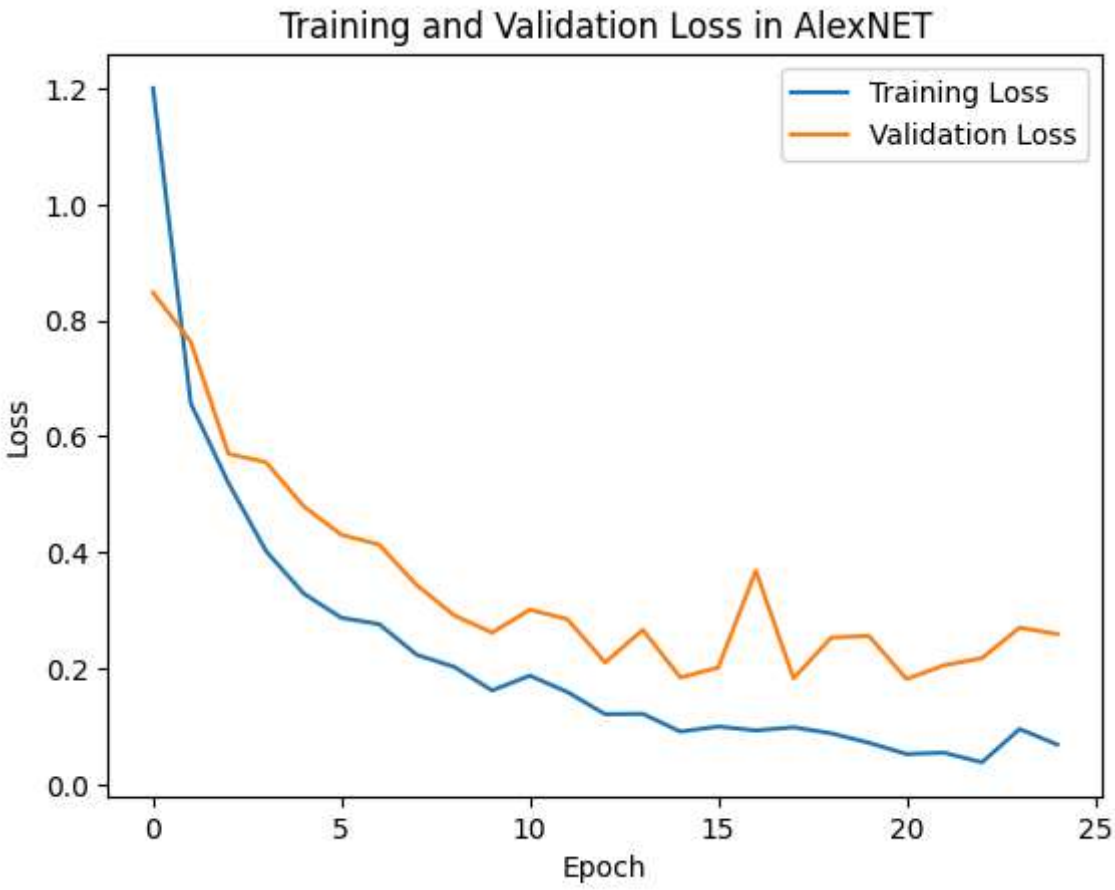
Visualisation of the Model, for perfect Evaluation :

In [25]:
```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy in AlexNET')
plt.show()
```



In [26]:
```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss in AlexNET')
plt.show()
```



In [28]:
```
from tensorflow.keras.models import load_model
cnn.save("/kaggle/working/Brain_tumour_prediction_alexnet.hdf5")
```

In [ ]:
```
```