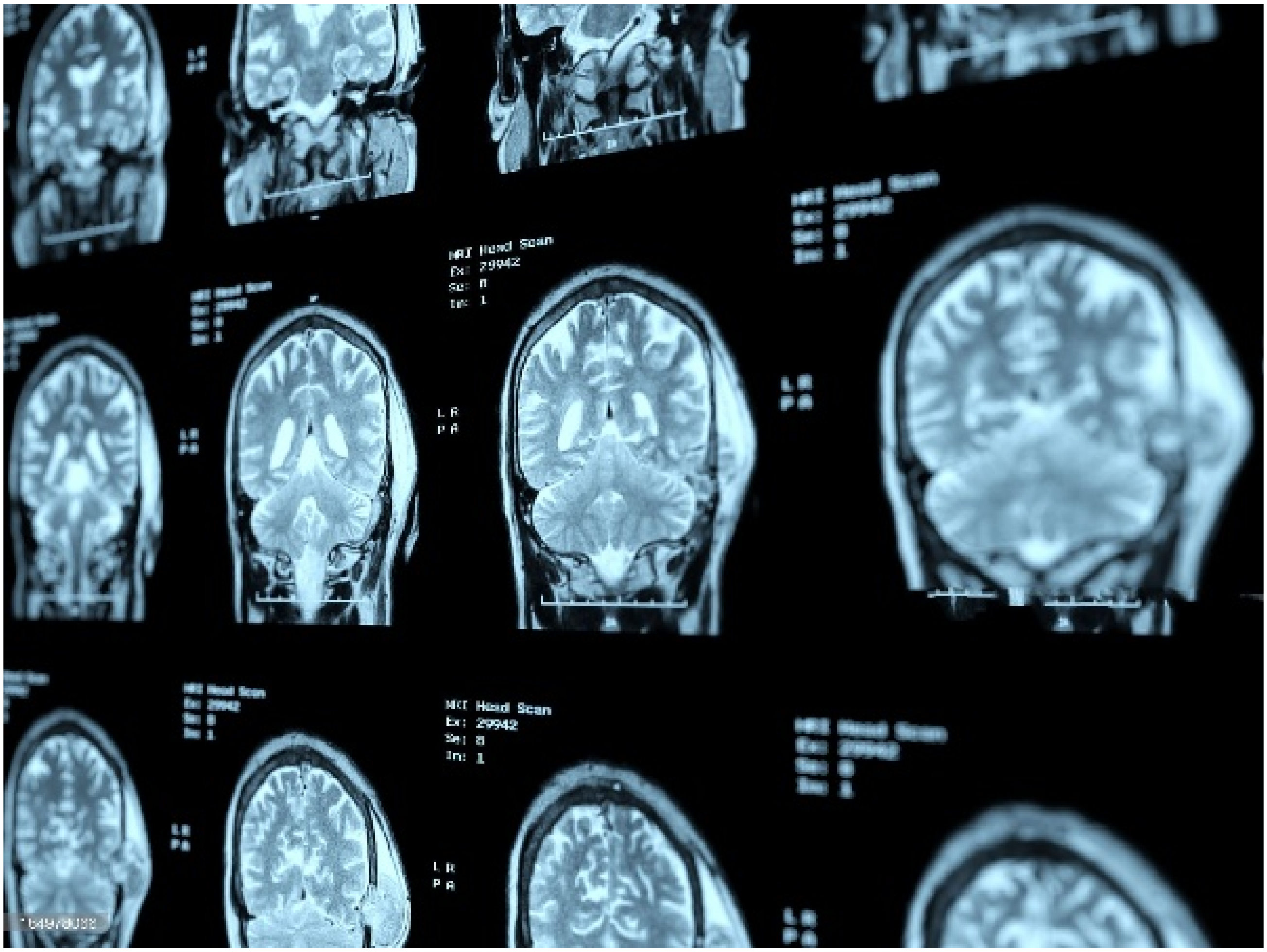
Detection and Classification of Brain Tumor through Brain MRI Images Using Deep Learning



1.Introduction

- The occurrence of brain tumor patients in India is steadily rising, more and more number of cases are reported each year in India across various age groups.
- The International Association of Cancer Registries (IARC) reported that there are over $28,000$ cases of brain tumours reported in India each year and more than $24,000$ people reportedly i.e. 85.72% of the total reported die due to brain tumours annually. Brain tumour's are a serious condition and in most cases fatal if not detected & treated in early stages.

2. Setting Up Local Storage for Dataset

2.1 Giving Access To Google Drive

In [ ]:

```
from google.colab import drive
drive.mount('/content/gdrive/')
```

```
Mounted at /content/gdrive/
```

2.2 Checking OS Version and Details

In [ ]:

```
print("OS Version & Details: ")
!lsb_release -a
```

```
OS Version & Details:
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.5 LTS
Release:        18.04
Codename:       bionic
```

3. Importing Required Libraries

In [ ]:

```
import sys
import os
import math

import numpy as np
import pandas as pd

from matplotlib import pyplot as plt
from matplotlib import rcParams
rcParams['figure.dpi'] = 300
%matplotlib inline
import seaborn as sns
import missingno as msno
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import *

from PIL import Image, ImageEnhance
from tensorflow.keras.preprocessing.image import *

print(f'Tensorflow Version: {tf.__version__}.')
```

```
Tensorflow Version: 2.5.0.
```

4. Setting Up the Environment

In [ ]:

```python
gpu_device_location = tpu_device_location = cpu_device_location = None

if os.environ['COLAB_GPU'] == '1':
    print("Allocated GPU Runtime Details:")
    !nvidia-smi
    print()
    try:
        import pynvml
        pynvml.nvmlInit()
        handle = pynvml.nvmlDeviceGetHandleByIndex(0)
        gpu_device_name = pynvml.nvmlDeviceGetName(handle)

        if gpu_device_name not in {b'Tesla T4', b'Tesla P4', b'Tesla P100-PCIE-16GB'}:
            raise Exception("Unfortunately this instance does not have a T4, P4 or P100 GPU.
 ↪\nSometimes Colab allocates a Tesla K80 instead of a T4, P4 or P100.\nIf you get Tesla␣
 ↪K80 then you can factory reset your runtime to get another GPUs.")
    except Exception as hardware_exception:
        print(hardware_exception, end = '\n\n')
    gpu_device_location = tf.test.gpu_device_name()
    print(f"{gpu_device_name.decode('utf-8')} is allocated sucessfully at location:␣
 ↪{gpu_device_location}")
elif 'COLAB_TPU_ADDR' in os.environ:
    tpu_device_location = f"grpc://{os.environ['COLAB_TPU_ADDR']}"
    print(f"TPU is allocated successfully at location: {tpu_device_location}.")
    resolver = tf.distribute.cluster_resolver.TPUClusterResolver(tpu_location)
    tf.config.experimental_connect_to_cluster(resolver)
    tf.tpu.experimental.initialize_tpu_system(resolver)
    tpu_strategy = tf.distribute.TPUStrategy()
else:
    cpu_device_location = "/cpu:0"
    print("GPUs and TPUs are not allocated successfully, hence runtime fallbacked to CPU.")
```

```
Allocated GPU Runtime Details:
Tue Jun  8 20:01:14 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 465.27       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   57C    P8    10W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
```

```
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+

Tesla T4 is allocated sucessfully at location: /device:GPU:0
```

4.1 Installation of tree Utility Using Bash.

In [ ]:

```
%%bash
RED_COLOR='\033[0;31m'
NO_COLOR='\033[0m'
pkg_name=tree
dpkg -s $pkg_name &> /dev/null
if [ "$?" -ne "0" ]
    then
        echo "Installing tree utility..."
        apt-get autoclean
        apt-get autoremove
        apt-get install $pkg_name
        if [ "$?" -eq "0" ]
            then
                echo -e ${RED_COLOR}"tree utility installed sucessfully.\n"${NO_COLOR}
        fi
    else
        echo "tree utility is already installed."
fi
tree --version
```

```
Installing tree utility…
Reading package lists…
Building dependency tree…
Reading state information…
Reading package lists…
Building dependency tree…
Reading state information…
0 upgraded, 0 newly installed, 0 to remove and 39 not upgraded.
Reading package lists…
Building dependency tree…
Reading state information…
The following NEW packages will be installed:
  tree
0 upgraded, 1 newly installed, 0 to remove and 39 not upgraded.
Need to get 40.7 kB of archives.
After this operation, 105 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/universe amd64 tree amd64 1.7.0-5 [40.7 kB]
Fetched 40.7 kB in 1s (73.4 kB/s)
Selecting previously unselected package tree.
(Reading database … 160772 files and directories currently installed.)
Preparing to unpack …/tree_1.7.0-5_amd64.deb …
Unpacking tree (1.7.0-5) …
Setting up tree (1.7.0-5) …
```

```
Processing triggers for man-db (2.8.3-2ubuntu0.1) …
tree utility installed sucessfully.


tree v1.7.0 (c) 1996 - 2014 by Steve Baker, Thomas Moore, Francesc Rocher, Florian
Sesser, Kyosuke Tokoro
```

4.2 Display of File Structure

In [ ]:

```
!tree -d "gdrive/MyDrive/Deep_Learning_Course_Project"
```

```
gdrive/MyDrive/Deep_Learning_Course_Project
  Brain-Tumor-Dataset
     Brain-Tumor-Images-Mat-Files
     Testing
        No
        Yes
     Training
        glioma
        meningioma
        no_tumor
        pituitary_tumor
     Tumor-Mask
        glioma
        meningioma
        pituitary_tumor
  Model-Checkpoints
     AlexNet-CNN
     InceptionV3
     Multi-Layer-Perceptron
  Research-Papers

19 directories
```

4.3 Setting Up Paths to Root and Data Directories

In [ ]:

```
ROOT_DIR = r"gdrive/MyDrive/Deep_Learning_Course_Project/"
DATA_ROOT_DIR = os.path.join(ROOT_DIR, "Brain-Tumor-Dataset")
TRAIN_DIR = os.path.join(DATA_ROOT_DIR, 'Training')
MASK_DIR = os.path.join(DATA_ROOT_DIR, 'Tumor-Mask')
assert os.path.isdir(ROOT_DIR) and os.path.isdir(DATA_ROOT_DIR) and os.path.
 ↪isdir(TRAIN_DIR) and os.path.isdir(MASK_DIR)
TUMOR_CLASS = ['meningioma', 'glioma', 'pituitary_tumor', 'no_tumor']
IMAGE_DATA_PATHS = [os.path.join(TRAIN_DIR, tumor_class) for tumor_class in TUMOR_CLASS]
MASK_DATA_PATHS = [os.path.join(MASK_DIR, tumor_name) for tumor_name in TUMOR_CLASS[:-1]]
```

5. Data Preprocessing and Exploratory Data Analysis

In [ ]:

```
data_distribution_count = pd.Series([len(os.listdir(path)) for path in IMAGE_DATA_PATHS if␣
↪os.path.exists(path) and os.path.isdir(path)],
                                    index = TUMOR_CLASS)
data_distribution_count
```

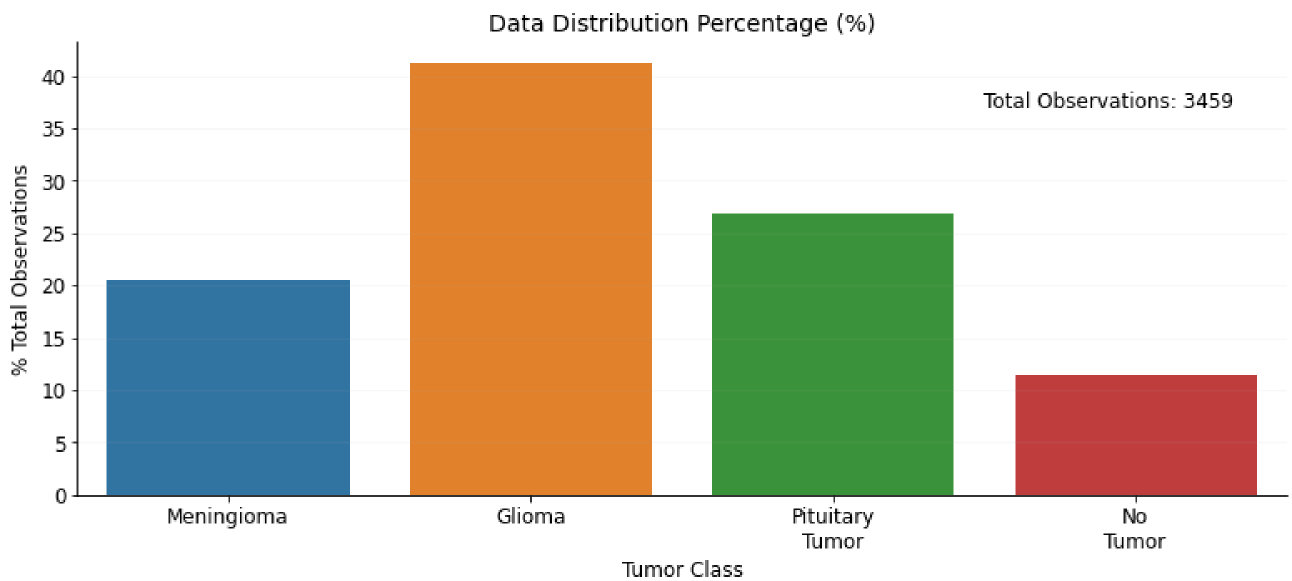
Out [ ]:

```
meningioma         708
glioma            1426
pituitary_tumor    930
no_tumor           395
dtype: int64
```

5.1 Data Distribution Visualization

In [ ]:

```
fig, axis = plt.subplots(figsize = (13, 5))
axis.grid(True, alpha = 0.1)
axis.set_title("Data Distribution Percentage (%)", fontsize = 14)
sns.barplot(x = ['\n'.join(curr_index.strip().split('_')).title() for curr_index in␣
↪data_distribution_count.index],
            y = 100 * (data_distribution_count / data_distribution_count.sum()), ax = axis)
axis.set_xlabel("Tumor Class", fontsize = 12)
axis.set_ylabel("% Total Observations", fontsize = 12)
axis.tick_params(which = 'major', labelsize = 12)
axis.text(2.5, 37, f'Total Observations: {data_distribution_count.sum()}', fontdict =␣
↪dict(size = 12))
sns.despine()
```



5.2 Visualisation of Brain MRI Dataset

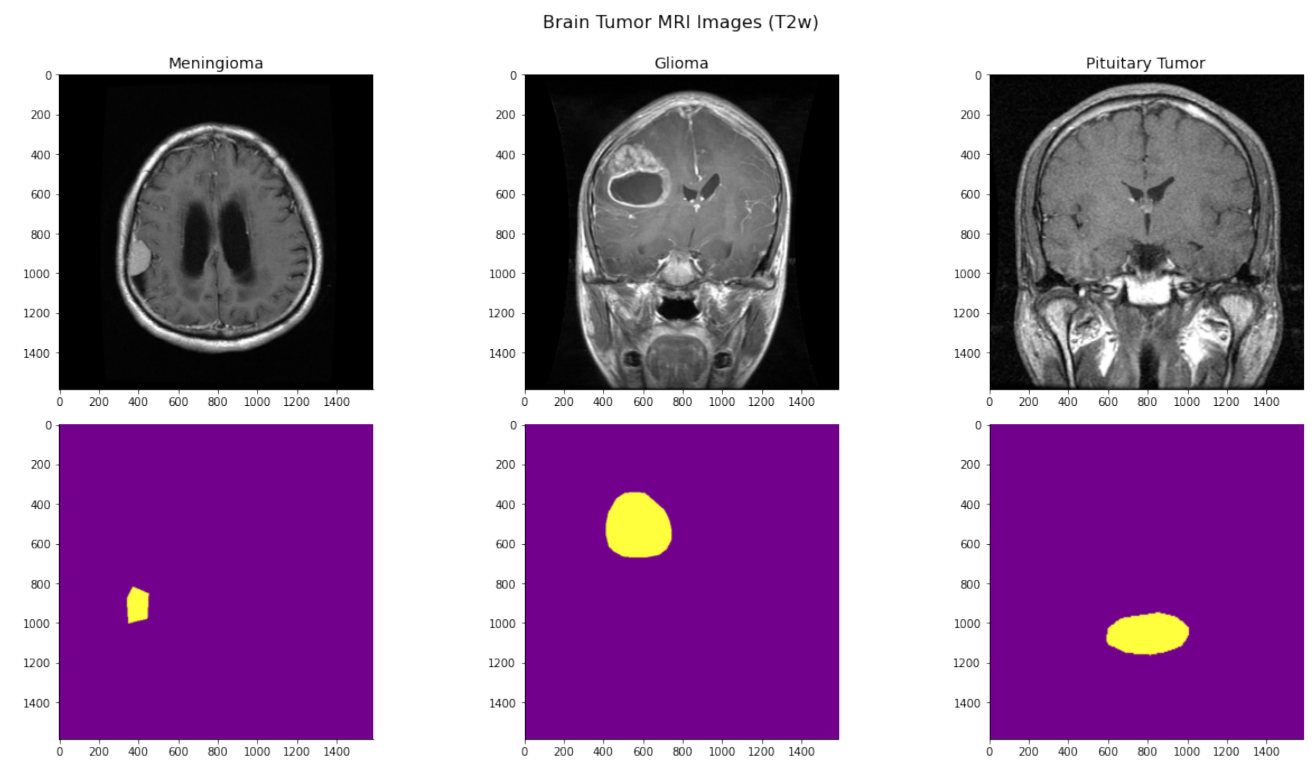
Dataset Source: https://figshare.com/articles/dataset/brain_tumor_dataset/1512427

Source Code for Conversion of .mat file to .jpg: Google Colab Notebook Link

Final Dataset Link: https://drive.google.com/drive/folders/11QIC82FBdAyq0PUwLVNd22i-oq6lcat1?usp=sharing

In [ ]:

```
BRIGHTNESS_FACTOR = 1.7
fig, axes = plt.subplots(nrows = 2, ncols = 3, figsize = (18, 9))
axes = axes.flatten()
fig.suptitle("Brain Tumor MRI Images (T2w)", fontsize = 16, fontdict = dict(weight =␣
 ↪'bold'), y = 1.04)
for curr_title, filename, curr_axis in zip(TUMOR_CLASS[:-1], IMAGE_DATA_PATHS[:-1], axes[:
 ↪3]):
    curr_image = Image.open(os.path.join(filename, os.listdir(filename)[2]))
    img_enhancer = ImageEnhance.Brightness(curr_image)
    curr_axis.imshow(img_enhancer.enhance(BRIGHTNESS_FACTOR))
    curr_axis.set_title(" ".join(curr_title.split('_')).title(), fontsize = 14)

for filename, curr_axis in zip(MASK_DATA_PATHS, axes[3:]):
    curr_image = Image.open(os.path.join(filename, os.listdir(filename)[2]))
    mask_enhancer = ImageEnhance.Brightness(curr_image)
    curr_axis.imshow(mask_enhancer.enhance(BRIGHTNESS_FACTOR))
fig.tight_layout()
sns.despine()
```



6. Development of Training, Validation & Testing Dataset

In [ ]:

```
image_data_paths = []
for curr_path, tumor_name in zip(IMAGE_DATA_PATHS, TUMOR_CLASS):
    if os.path.exists(curr_path) and os.path.isdir(curr_path):
        image_data_paths.extend(map(lambda filename: (os.path.join(curr_path, filename),␣
↪tumor_name), os.listdir(curr_path)))
```

In [ ]:

```
image_data_paths_df = pd.DataFrame(image_data_paths, columns = ['image_filepaths',␣
↪'tumor_class']).sample(frac = 1, random_state = 42).reset_index(drop = True)
image_data_paths_df.head()
```

Out [ ]:

	image_filepaths	tumor_class
0	gdrive/MyDrive/Deep_Learning_Course_Project/Br…	meningioma
1	gdrive/MyDrive/Deep_Learning_Course_Project/Br…	meningioma
2	gdrive/MyDrive/Deep_Learning_Course_Project/Br…	pituitary_tumor
3	gdrive/MyDrive/Deep_Learning_Course_Project/Br…	pituitary_tumor
4	gdrive/MyDrive/Deep_Learning_Course_Project/Br…	pituitary_tumor

In [ ]:

```
image_data_paths_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3459 entries, 0 to 3458
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   image_filepaths  3459 non-null   object
 1   tumor_class      3459 non-null   object
dtypes: object(2)
memory usage: 54.2+ KB
```

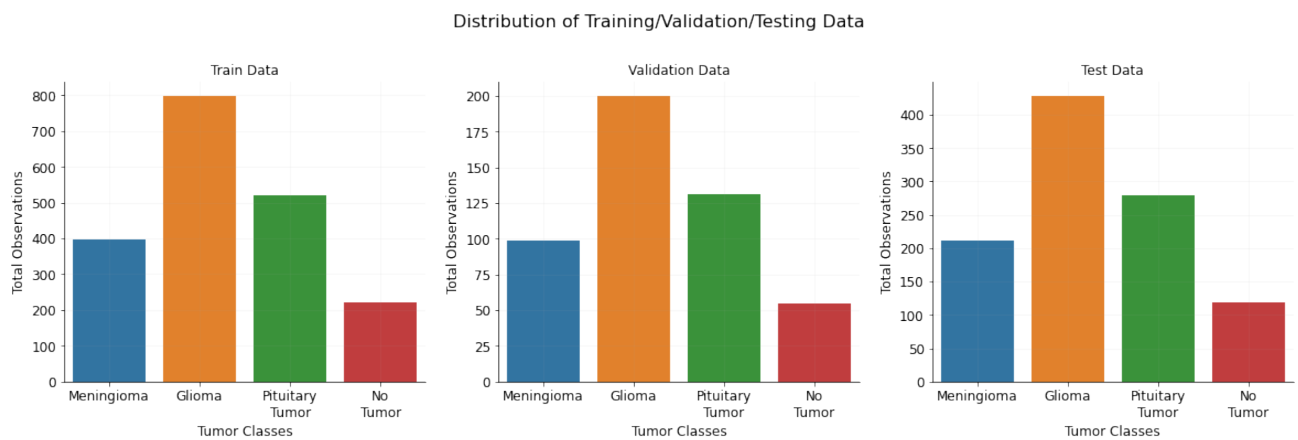
In [ ]:

```
intermediate_train_data, test_data = train_test_split(image_data_paths_df,
                                                      train_size = 0.70,
                                                      random_state = 42,
                                                      stratify = image_data_paths_df.
↪tumor_class)

train_data, validation_data = train_test_split(intermediate_train_data,
                                               train_size = 0.80,
                                               random_state = 42,
                                               stratify = intermediate_train_data.
↪tumor_class)
```

6.1 Training, Validation and Testing Dataset Data Distribution Visualization

In [ ]:

```
fig, axes = plt.subplots(ncols = 3, figsize = (20, 5))
fig.suptitle("Distribution of Training/Validation/Testing Data", fontsize = 16, fontdict =␣
↪dict(weight = 'bold'), y = 1.05)
sns.countplot(x = train_data.tumor_class, order = TUMOR_CLASS, ax = axes[0])
sns.countplot(x = validation_data.tumor_class, order = TUMOR_CLASS, ax = axes[1])
sns.countplot(x = test_data.tumor_class, order = TUMOR_CLASS, ax = axes[2])
for curr_axis, curr_title in zip(axes, ['Train Data', 'Validation Data', 'Test Data']):
    curr_axis.grid(False, alpha = 0.1)
    curr_axis.set_title(curr_title, fontsize = 12)
    curr_axis.set_xlabel("Tumor Classes", fontsize = 12)
    curr_axis.set_ylabel("Total Observations", fontsize = 12)
    curr_axis.tick_params(which = 'major', labelsize = 12)
    curr_axis.set_xticklabels(["\n".join(xtick.split("_")).title() for xtick in␣
↪TUMOR_CLASS])
sns.despine()
```



7. Data/Image Augmentation

- Image augmentation is usually used to increase the image dataset and also to make the network more robust against translation invariance. Image augmentation is defined as creating duplicates of the original image datasets by flipping, rotating, zooming, and adjusting brightness.
- We will use data/image augmentation using `ImageDataGenerator` class to train the model on different types of combinations formed by rotation, flipping, changing the brightness etc of an image so as to increase our model accuracy.

In [ ]:

```
image_size = 128
batch_size = 32

image_datagen_kwargs = dict(rescale = 1 / 255,
                            rotation_range = 15,
                            width_shift_range = 0.1,
```

```
                          zoom_range = 0.01,
                          shear_range = 0.01,
                          brightness_range = [0.3, 1.5],
                          horizontal_flip = True,
                          vertical_flip = True)
```

In [ ]:

```
train_image_datagen = ImageDataGenerator(**image_datagen_kwargs)
validation_image_datagen = ImageDataGenerator(**image_datagen_kwargs)
test_image_datagen = ImageDataGenerator(**image_datagen_kwargs)
```

In [ ]:

```
train_dataset = train_image_datagen.flow_from_dataframe(train_data,
                                                        x_col = 'image_filepaths',
                                                        y_col = 'tumor_class',
                                                        seed = 42,
                                                        batch_size = batch_size,
                                                        target_size = (image_size,␣
 ↪image_size),
                                                        color_mode = 'rgb')
validation_dataset = validation_image_datagen.flow_from_dataframe(validation_data,
                                                                  x_col = 'image_filepaths',
                                                                  y_col = 'tumor_class',
                                                                  seed = 42,
                                                                  batch_size = batch_size,
                                                                  target_size =␣
 ↪(image_size, image_size),
                                                                  color_mode = 'rgb')
test_dataset = test_image_datagen.flow_from_dataframe(test_data,
                                                      x_col = 'image_filepaths',
                                                      y_col = 'tumor_class',
                                                      seed = 42,
                                                      batch_size = batch_size,
                                                      target_size = (image_size,␣
 ↪image_size),
                                                      color_mode = 'rgb')
```

```
Found 1936 validated image filenames belonging to 4 classes.
Found 485 validated image filenames belonging to 4 classes.
Found 1038 validated image filenames belonging to 4 classes.
```

In [ ]:

```
print("Information about Training Dataset:")
print(train_dataset.class_indices)
print(train_dataset.image_shape, end = '\n\n')

print("Information about Validation Dataset:")
print(validation_dataset.class_indices)
print(validation_dataset.image_shape, end = '\n\n')
```

```
print("Information about Testing Dataset:")
print(test_dataset.class_indices)
print(test_dataset.image_shape)
```

```
Information about Training Dataset:
{'glioma': 0, 'meningioma': 1, 'no_tumor': 2, 'pituitary_tumor': 3}
(128, 128, 3)

Information about Validation Dataset:
{'glioma': 0, 'meningioma': 1, 'no_tumor': 2, 'pituitary_tumor': 3}
(128, 128, 3)

Information about Testing Dataset:
{'glioma': 0, 'meningioma': 1, 'no_tumor': 2, 'pituitary_tumor': 3}
(128, 128, 3)
```

7.1 Training Data Images Glimpse

In [ ]:

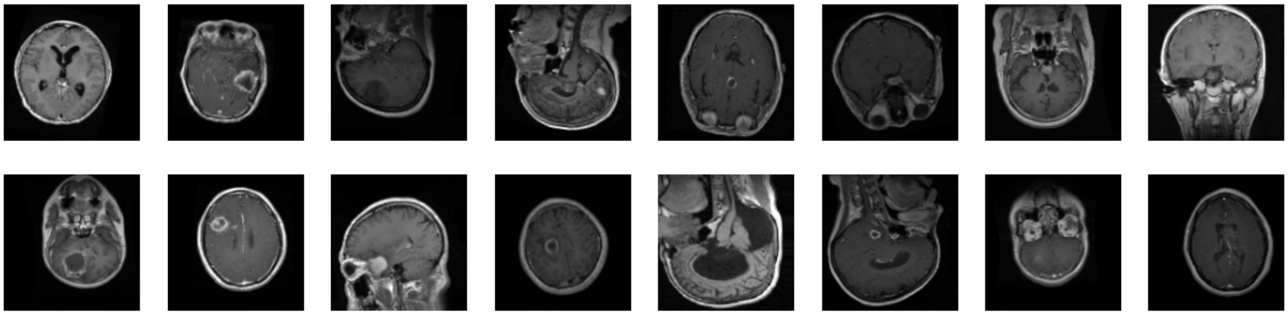
```
fig, axes = plt.subplots(nrows = 2, ncols = 8, figsize = (20, 5))
fig.suptitle("Samples from Training Set Batch", fontsize = 16, fontdict = dict(weight =␣
 ↪'bold'))
for curr_axis, curr_image in zip(axes.flatten(), train_dataset[0][0][:16]):
    curr_axis.imshow(tf.squeeze(curr_image), cmap = 'gray')
    curr_axis.axis(False)
```

7.2 Validation Data Images Glimpse

In [ ]:

```
fig, axes = plt.subplots(nrows = 2, ncols = 8, figsize = (20, 5))
fig.suptitle("Samples from Validation Set Batch", fontsize = 16, fontdict = dict(weight =␣
 ↪'bold'))
for curr_axis, curr_image in zip(axes.flatten(), validation_dataset[0][0][:16]):
    curr_axis.imshow(tf.squeeze(curr_image), cmap = 'gray')
    curr_axis.axis(False)
```

Samples from Validation Set Batch

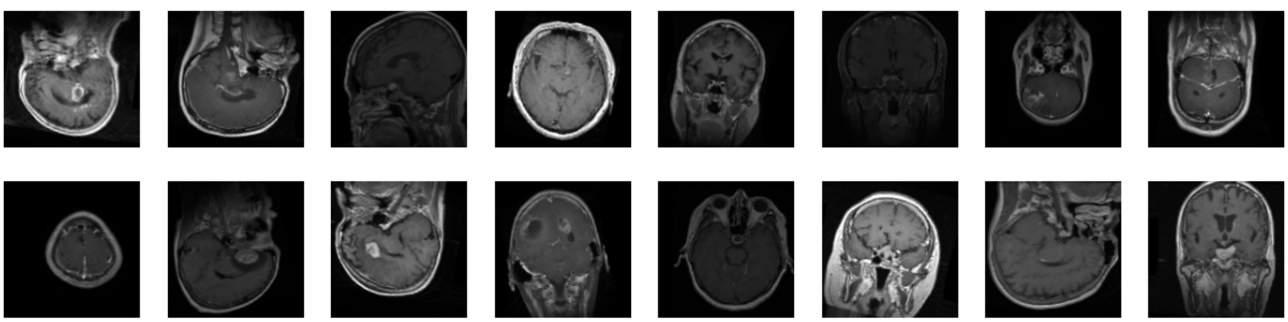


7.3 Testing Data Images Glimpse

In [ ]:

```
fig, axes = plt.subplots(nrows = 2, ncols = 8, figsize = (20, 5))
fig.suptitle("Samples from Testing Set Batch", fontsize = 16, fontdict = dict(weight = 'bold'))
for curr_axis, curr_image in zip(axes.flatten(), test_dataset[0][0][:16]):
    curr_axis.imshow(tf.squeeze(curr_image), cmap = 'gray')
    curr_axis.axis(False)
```

Samples from Testing Set Batch



8. Model Development

In [ ]:

```
early_stopping = EarlyStopping(monitor = 'val_accuracy', patience = 10)
```

In [ ]:

```
ROOT_CHECKPOINT_DIR_PATH = os.path.join(ROOT_DIR, "Model-Checkpoints")
MLP_CHECKPOINT_DIR_PATH = os.path.join(ROOT_CHECKPOINT_DIR_PATH, "Multi-Layer-Perceptron")
ALEXNET_CHECKPOINT_DIR_PATH = os.path.join(ROOT_CHECKPOINT_DIR_PATH, "AlexNet-CNN")
INCEPTIONV3_CHECKPOINT_DIR_PATH = os.path.join(ROOT_CHECKPOINT_DIR_PATH, "InceptionV3")
```

```
assert os.path.isdir(ROOT_CHECKPOINT_DIR_PATH) and os.path.isdir(MLP_CHECKPOINT_DIR_PATH)␣
↪and os.path.isdir(ALEXNET_CHECKPOINT_DIR_PATH) and os.path.
↪isdir(INCEPTIONV3_CHECKPOINT_DIR_PATH)
```

In [ ]:

```
mlp_cp_callback = ModelCheckpoint(MLP_CHECKPOINT_DIR_PATH,
                                  monitor = 'val_accuracy',
                                  verbose = 1,
                                  save_weights_only = True,
                                  save_freq =  'epoch')

alexnet_cp_callback = ModelCheckpoint(ALEXNET_CHECKPOINT_DIR_PATH,
                                      monitor = 'val_accuracy',
                                      verbose = 1,
                                      save_weights_only = True,
                                      save_freq = 'epoch')

inceptionv3_cp_callback = ModelCheckpoint(INCEPTIONV3_CHECKPOINT_DIR_PATH,
                                          monitor = 'val_accuracy',
                                          verbose = 1,
                                          save_weights_only = True,
                                          save_freq = 'epoch')
```

In [ ]:

```
def training_process_viz(training_stats: pd.DataFrame, **plot_kwargs) -> None:
    fig, axes = plt.subplots(ncols = 2, figsize = (15, 5))
    fig.suptitle(plot_kwargs['plot_title'], fontsize = 16, fontdict = dict(weight =␣
↪'bold'), y = 1.08)
    for curr_axis, col_name in zip(axes, ['accuracy', 'loss']):
        curr_axis.grid(True, alpha = 0.3)
        curr_axis.set_title(f"Model {col_name}".title(), fontsize = 14)
        sns.lineplot(x = range(1, 1 + training_stats.shape[0]), y =␣
↪training_stats[col_name], color = 'blue', ax = curr_axis)
        sns.lineplot(x = range(1, 1 + training_stats.shape[0]), y =␣
↪training_stats[f"val_{col_name}"], color = 'red', ax = curr_axis)
        curr_axis.set_xlabel("Epochs", fontsize = 12)
        curr_axis.set_ylabel(col_name.title(), fontsize = 12)
        curr_axis.tick_params(which = 'major', labelsize = 12)
        curr_axis.legend([col_name.title(), f"validation {col_name}".title()], title =␣
↪col_name.title())
    fig.tight_layout()
    sns.despine()
```

In [ ]:

```
def confusion_matrix_viz(model, test_dataset, **plot_kwargs) -> None:
    assert isinstance(model, Sequential)
    model_preds = [np.argmax(curr_row) for curr_row in model.predict(test_dataset)]
    fig, axis = plt.subplots(figsize = (8, 6))
```

```
    class_names = ['Glioma', 'Meningioma', 'No-Tumor', 'Pituitary\nTumor']
    sns.heatmap(confusion_matrix(test_dataset.classes, model_preds), annot = True, cmap =␣
↪plt.cm.Reds, ax = axis)
    axis.set_title(plot_kwargs['plot_title'], fontsize = 14)
    axis.tick_params(which = 'major', labelsize = 12)
    axis.set_xlabel("Pedicted Class", fontsize = 12)
    axis.set_ylabel("Actual Class", fontsize = 12)
    axis.set_xticklabels(class_names, fontdict = dict(fontsize = 12))
    axis.set_yticklabels(class_names, fontdict = dict(fontsize = 12))
    fig.tight_layout()
    sns.despine()
```

In [ ]:

```
def generate_report(*models, test_dataset, row_indexes) -> pd.DataFrame:
    assert len(models)
    report_df = pd.DataFrame(columns = ['MAE', 'MSE', 'RMSE', 'Loss', 'Accuracy',␣
↪'F1-Score'])
    y_hat = test_dataset.classes # y_hat = ground_truth
    for curr_index, curr_model in enumerate(models):
        assert isinstance(curr_model, Sequential)
        curr_model_loss, curr_model_accuracy = curr_model.evaluate(test_dataset)
        y_preds = [np.argmax(curr_preds) for curr_preds in curr_model.predict(test_dataset)]
        report_df.loc[curr_index] = [mean_absolute_error(y_hat, y_preds),␣
↪mean_squared_error(y_hat, y_preds), mean_squared_error(y_hat, y_preds, squared = False),␣
↪curr_model_loss, curr_model_accuracy, f1_score(y_hat, y_preds, average = "micro")]
    report_df.index = row_indexes
    return report_df
```

8.1 Multi-Layer Perceptron

8.1.1 Development of Multi-Layer Perceptron Model

In [ ]:

```
mlp_model = Sequential()
mlp_model.add(Flatten(input_shape = (image_size, image_size, 3), name = 'Flatten-Layer'))
mlp_model.add(Dense(2048, activation = 'relu', name = 'Hidden-Layer-1'))
mlp_model.add(Dropout(rate = 0.2, name = 'Dropout-Layer-1'))
mlp_model.add(Dense(1024, activation = 'relu', name = 'Hidden-Layer-2'))
mlp_model.add(Dropout(rate = 0.2, name = 'Dropout-Layer-2'))
mlp_model.add(Dense(512, activation = 'relu', name = 'Hidden-Layer-3'))
mlp_model.add(Dropout(rate = 0.2, name = 'Dropout-Layer-3'))
mlp_model.add(Dense(4, activation = 'softmax', name = 'Output-Layer-1'))
mlp_model.compile(optimizer = 'Adam', loss = 'categorical_crossentropy', metrics =␣
↪['accuracy'])
mlp_model.summary()
```

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
Flatten-Layer (Flatten)      (None, 49152)            0
_________________________________________________________________
Hidden-Layer-1 (Dense)       (None, 2048)             100665344
_________________________________________________________________
Dropout-Layer-1 (Dropout)    (None, 2048)             0
_________________________________________________________________
Hidden-Layer-2 (Dense)       (None, 1024)             2098176
_________________________________________________________________
Dropout-Layer-2 (Dropout)    (None, 1024)             0
_________________________________________________________________
Hidden-Layer-3 (Dense)       (None, 512)              524800
_________________________________________________________________
Dropout-Layer-3 (Dropout)    (None, 512)              0
_________________________________________________________________
Output-Layer-1 (Dense)       (None, 4)                2052
=================================================================
Total params: 103,290,372
Trainable params: 103,290,372
Non-trainable params: 0
_________________________________________________________________
```

8.1.2 Training and Validation of Multi-Layer Perceptron Based Model

In [ ]:

```python
with tf.device(gpu_device_location) if gpu_device_location else tpu_strategy.scope() if␣
 ↪tpu_device_location else tf.device(cpu_device_location):
    mlp_train_history = mlp_model.fit(train_dataset,
                                      batch_size = batch_size,
                                      validation_data = validation_dataset,
                                      epochs = 100,
                                      callbacks = [early_stopping])
```

```
Epoch 1/100
61/61 [==============================] - 847s 14s/step - loss: 4.7656 - accuracy: 0.3957
- val_loss: 1.3696 - val_accuracy: 0.3299
Epoch 2/100
61/61 [==============================] - 77s 1s/step - loss: 1.2743 - accuracy: 0.4917 -
val_loss: 1.1744 - val_accuracy: 0.5629
Epoch 3/100
61/61 [==============================] - 75s 1s/step - loss: 1.1763 - accuracy: 0.5243 -
val_loss: 1.1415 - val_accuracy: 0.5505
Epoch 4/100
61/61 [==============================] - 74s 1s/step - loss: 1.1731 - accuracy: 0.5232 -
val_loss: 1.1290 - val_accuracy: 0.5670
Epoch 5/100
61/61 [==============================] - 74s 1s/step - loss: 1.1487 - accuracy: 0.5320 -
val_loss: 1.1370 - val_accuracy: 0.5423
Epoch 6/100
61/61 [==============================] - 74s 1s/step - loss: 1.1226 - accuracy: 0.5579 -
val_loss: 1.1002 - val_accuracy: 0.5773
Epoch 7/100
61/61 [==============================] - 74s 1s/step - loss: 1.1489 - accuracy: 0.5403 -
```

```
val_loss: 1.1865 - val_accuracy: 0.5423
Epoch 8/100
61/61 [==============================] - 73s 1s/step - loss: 1.1077 - accuracy: 0.5635 -
val_loss: 1.1172 - val_accuracy: 0.5691
Epoch 9/100
61/61 [==============================] - 73s 1s/step - loss: 1.1134 - accuracy: 0.5708 -
val_loss: 1.0647 - val_accuracy: 0.5732
Epoch 10/100
61/61 [==============================] - 73s 1s/step - loss: 1.1059 - accuracy: 0.5697 -
val_loss: 1.0703 - val_accuracy: 0.5835
Epoch 11/100
61/61 [==============================] - 73s 1s/step - loss: 1.0625 - accuracy: 0.5826 -
val_loss: 1.0690 - val_accuracy: 0.5835
Epoch 12/100
61/61 [==============================] - 73s 1s/step - loss: 1.0535 - accuracy: 0.5821 -
val_loss: 1.0487 - val_accuracy: 0.5835
Epoch 13/100
61/61 [==============================] - 74s 1s/step - loss: 1.0763 - accuracy: 0.5713 -
val_loss: 1.0513 - val_accuracy: 0.5918
Epoch 14/100
61/61 [==============================] - 74s 1s/step - loss: 1.0516 - accuracy: 0.5785 -
val_loss: 1.0400 - val_accuracy: 0.5856
Epoch 15/100
61/61 [==============================] - 74s 1s/step - loss: 1.0251 - accuracy: 0.5919 -
val_loss: 1.0699 - val_accuracy: 0.5856
Epoch 16/100
61/61 [==============================] - 74s 1s/step - loss: 1.0348 - accuracy: 0.5852 -
val_loss: 1.0210 - val_accuracy: 0.6041
Epoch 17/100
61/61 [==============================] - 74s 1s/step - loss: 1.0266 - accuracy: 0.5847 -
val_loss: 1.1187 - val_accuracy: 0.6124
Epoch 18/100
61/61 [==============================] - 74s 1s/step - loss: 1.0384 - accuracy: 0.5888 -
val_loss: 1.0726 - val_accuracy: 0.5629
Epoch 19/100
61/61 [==============================] - 73s 1s/step - loss: 1.0218 - accuracy: 0.5935 -
val_loss: 1.0249 - val_accuracy: 0.5918
Epoch 20/100
61/61 [==============================] - 73s 1s/step - loss: 1.0167 - accuracy: 0.6049 -
val_loss: 0.9846 - val_accuracy: 0.6021
Epoch 21/100
61/61 [==============================] - 73s 1s/step - loss: 1.0130 - accuracy: 0.6012 -
val_loss: 0.9933 - val_accuracy: 0.6062
Epoch 22/100
61/61 [==============================] - 73s 1s/step - loss: 0.9902 - accuracy: 0.6162 -
val_loss: 0.9511 - val_accuracy: 0.6289
Epoch 23/100
61/61 [==============================] - 73s 1s/step - loss: 1.0167 - accuracy: 0.5966 -
val_loss: 1.0133 - val_accuracy: 0.5835
Epoch 24/100
61/61 [==============================] - 73s 1s/step - loss: 0.9807 - accuracy: 0.5992 -
val_loss: 1.0058 - val_accuracy: 0.6000
```

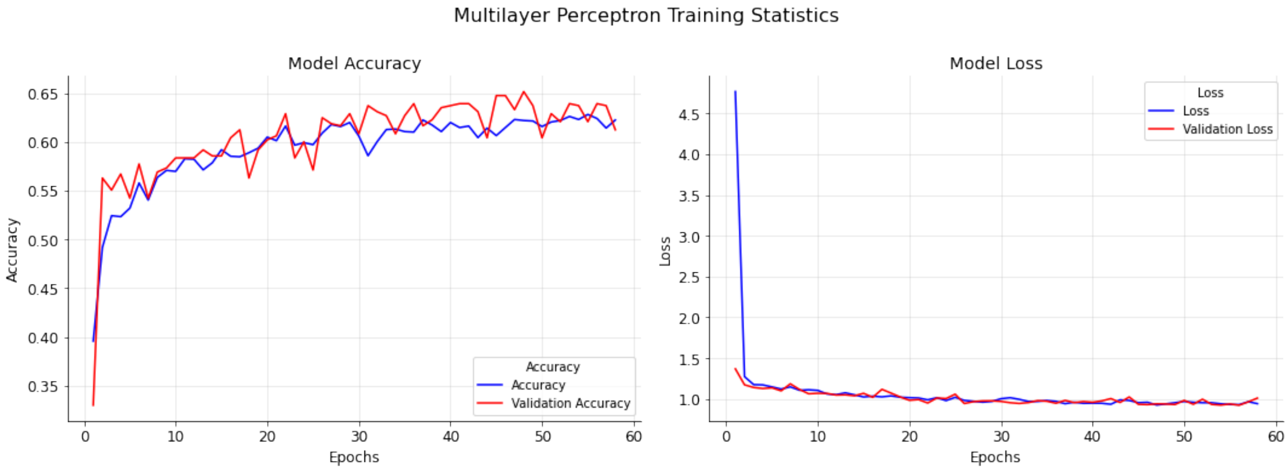
```
Epoch 25/100
61/61 [==============================] - 73s 1s/step - loss: 1.0206 - accuracy: 0.5971 -
val_loss: 1.0592 - val_accuracy: 0.5711
Epoch 26/100
61/61 [==============================] - 73s 1s/step - loss: 0.9835 - accuracy: 0.6090 -
val_loss: 0.9448 - val_accuracy: 0.6247
Epoch 27/100
61/61 [==============================] - 73s 1s/step - loss: 0.9723 - accuracy: 0.6178 -
val_loss: 0.9680 - val_accuracy: 0.6186
Epoch 28/100
61/61 [==============================] - 74s 1s/step - loss: 0.9614 - accuracy: 0.6157 -
val_loss: 0.9766 - val_accuracy: 0.6165
Epoch 29/100
61/61 [==============================] - 73s 1s/step - loss: 0.9707 - accuracy: 0.6198 -
val_loss: 0.9779 - val_accuracy: 0.6289
Epoch 30/100
61/61 [==============================] - 73s 1s/step - loss: 1.0039 - accuracy: 0.6059 -
val_loss: 0.9709 - val_accuracy: 0.6082
Epoch 31/100
61/61 [==============================] - 73s 1s/step - loss: 1.0166 - accuracy: 0.5857 -
val_loss: 0.9550 - val_accuracy: 0.6371
Epoch 32/100
61/61 [==============================] - 73s 1s/step - loss: 0.9962 - accuracy: 0.6002 -
val_loss: 0.9459 - val_accuracy: 0.6309
Epoch 33/100
61/61 [==============================] - 73s 1s/step - loss: 0.9699 - accuracy: 0.6126 -
val_loss: 0.9566 - val_accuracy: 0.6268
Epoch 34/100
61/61 [==============================] - 73s 1s/step - loss: 0.9707 - accuracy: 0.6131 -
val_loss: 0.9805 - val_accuracy: 0.6082
Epoch 35/100
61/61 [==============================] - 73s 1s/step - loss: 0.9817 - accuracy: 0.6105 -
val_loss: 0.9733 - val_accuracy: 0.6268
Epoch 36/100
61/61 [==============================] - 74s 1s/step - loss: 0.9715 - accuracy: 0.6100 -
val_loss: 0.9485 - val_accuracy: 0.6392
Epoch 37/100
61/61 [==============================] - 73s 1s/step - loss: 0.9430 - accuracy: 0.6224 -
val_loss: 0.9829 - val_accuracy: 0.6165
Epoch 38/100
61/61 [==============================] - 73s 1s/step - loss: 0.9607 - accuracy: 0.6173 -
val_loss: 0.9563 - val_accuracy: 0.6227
Epoch 39/100
61/61 [==============================] - 73s 1s/step - loss: 0.9476 - accuracy: 0.6105 -
val_loss: 0.9691 - val_accuracy: 0.6351
Epoch 40/100
61/61 [==============================] - 73s 1s/step - loss: 0.9499 - accuracy: 0.6198 -
val_loss: 0.9605 - val_accuracy: 0.6371
Epoch 41/100
61/61 [==============================] - 73s 1s/step - loss: 0.9481 - accuracy: 0.6147 -
val_loss: 0.9770 - val_accuracy: 0.6392
Epoch 42/100
```

```
61/61 [==============================] - 74s 1s/step - loss: 0.9364 - accuracy: 0.6162 -
val_loss: 1.0055 - val_accuracy: 0.6392
Epoch 43/100
61/61 [==============================] - 73s 1s/step - loss: 0.9902 - accuracy: 0.6043 -
val_loss: 0.9579 - val_accuracy: 0.6309
Epoch 44/100
61/61 [==============================] - 74s 1s/step - loss: 0.9823 - accuracy: 0.6142 -
val_loss: 1.0254 - val_accuracy: 0.6041
Epoch 45/100
61/61 [==============================] - 74s 1s/step - loss: 0.9551 - accuracy: 0.6064 -
val_loss: 0.9354 - val_accuracy: 0.6474
Epoch 46/100
61/61 [==============================] - 74s 1s/step - loss: 0.9600 - accuracy: 0.6147 -
val_loss: 0.9321 - val_accuracy: 0.6474
Epoch 47/100
61/61 [==============================] - 73s 1s/step - loss: 0.9266 - accuracy: 0.6229 -
val_loss: 0.9412 - val_accuracy: 0.6330
Epoch 48/100
61/61 [==============================] - 74s 1s/step - loss: 0.9405 - accuracy: 0.6219 -
val_loss: 0.9390 - val_accuracy: 0.6515
Epoch 49/100
61/61 [==============================] - 74s 1s/step - loss: 0.9551 - accuracy: 0.6214 -
val_loss: 0.9330 - val_accuracy: 0.6371
Epoch 50/100
61/61 [==============================] - 73s 1s/step - loss: 0.9704 - accuracy: 0.6157 -
val_loss: 0.9832 - val_accuracy: 0.6041
Epoch 51/100
61/61 [==============================] - 73s 1s/step - loss: 0.9604 - accuracy: 0.6204 -
val_loss: 0.9360 - val_accuracy: 0.6289
Epoch 52/100
61/61 [==============================] - 73s 1s/step - loss: 0.9545 - accuracy: 0.6219 -
val_loss: 0.9978 - val_accuracy: 0.6206
Epoch 53/100
61/61 [==============================] - 73s 1s/step - loss: 0.9547 - accuracy: 0.6260 -
val_loss: 0.9343 - val_accuracy: 0.6392
Epoch 54/100
61/61 [==============================] - 73s 1s/step - loss: 0.9417 - accuracy: 0.6229 -
val_loss: 0.9262 - val_accuracy: 0.6371
Epoch 55/100
61/61 [==============================] - 73s 1s/step - loss: 0.9335 - accuracy: 0.6281 -
val_loss: 0.9395 - val_accuracy: 0.6206
Epoch 56/100
61/61 [==============================] - 73s 1s/step - loss: 0.9317 - accuracy: 0.6240 -
val_loss: 0.9245 - val_accuracy: 0.6392
Epoch 57/100
61/61 [==============================] - 73s 1s/step - loss: 0.9670 - accuracy: 0.6142 -
val_loss: 0.9657 - val_accuracy: 0.6371
Epoch 58/100
61/61 [==============================] - 73s 1s/step - loss: 0.9434 - accuracy: 0.6224 -
val_loss: 1.0117 - val_accuracy: 0.6124
```

8.1.3 Multi-Layer Perceptron Based Model Training Process Statistics

In [ ]:

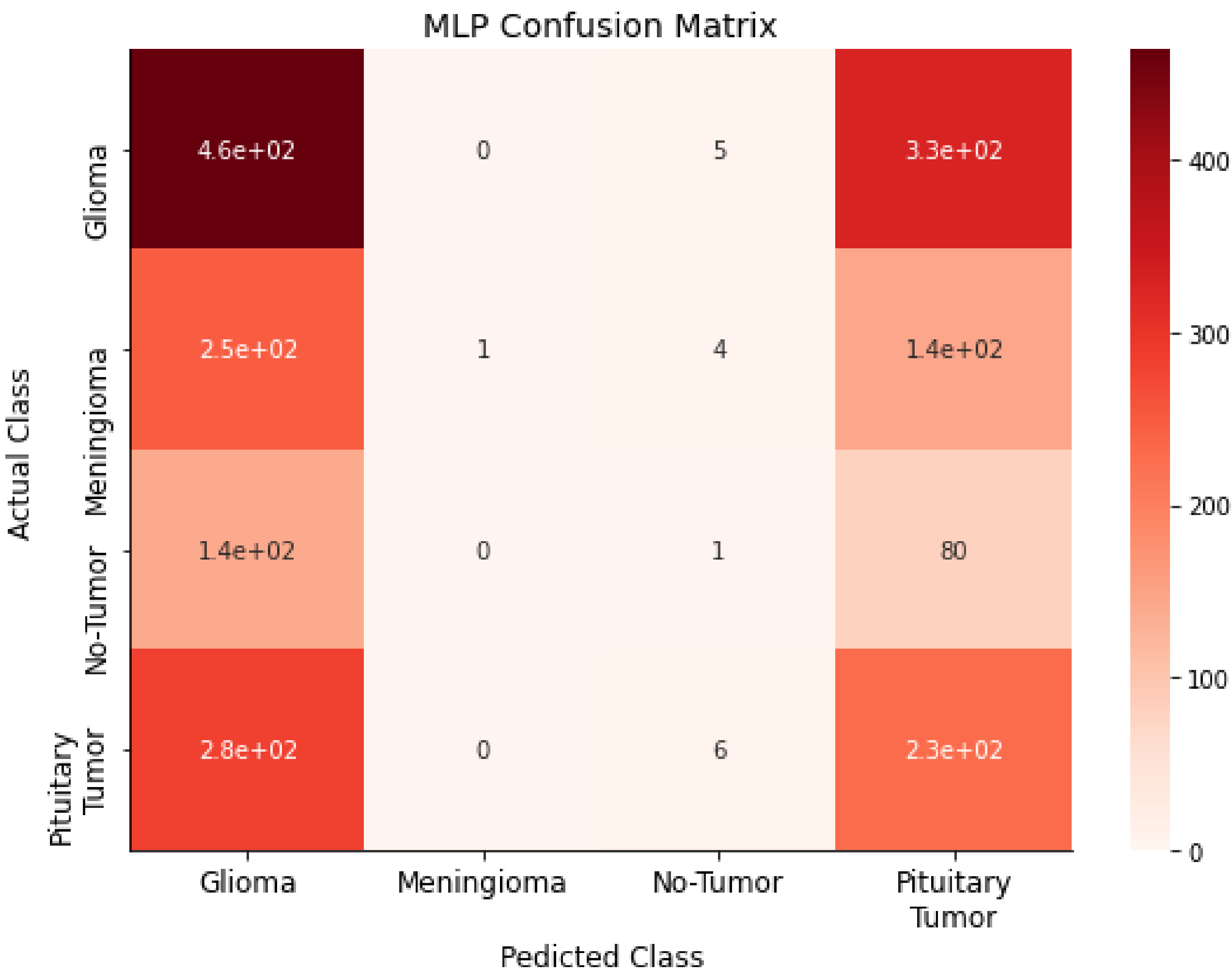
```
training_process_viz(pd.DataFrame(mlp_train_history.history),
                     plot_title = 'Multilayer Perceptron Training Statistics')
```



8.1.4 Confusion Matrix for Multi-Layer Perceptron Based Model

In [ ]:

```
confusion_matrix_viz(mlp_model, train_dataset, plot_title = "MLP Confusion Matrix")
```



In [ ]:

```
mlp_report_df = generate_report(mlp_model,
                                test_dataset = test_dataset,
                                row_indexes = ("Multi-Layer-Perceptron Model",))
mlp_report_df
```

```
33/33 [==============================] - 33s 991ms/step - loss: 0.9263 - accuracy: 0.6243
```

Out [ ]:

```
                                  MAE       MSE  ...  Accuracy  F1-Score
Multi-Layer-Perceptron Model  1.39499  3.533719  ...  0.624277  0.368979

[1 rows x 6 columns]
```

8.2 AlexNet CNN

8.2.1 Develoment of AlexNet CNN Model

In [ ]:

```python
alexnet_cnn = Sequential()
alexnet_cnn.add(Conv2D(96, kernel_size = 11, strides = 4, activation = 'relu', input_shape␣
 ↪= (image_size, image_size, 3), name = 'Conv2D-1'))
alexnet_cnn.add(BatchNormalization(name = 'Batch-Normalization-1'))
alexnet_cnn.add(MaxPool2D(pool_size = 3, strides = 2, name = 'Max-Pooling-1'))
alexnet_cnn.add(Conv2D(256, kernel_size = 5, padding = 'same', activation = 'relu', name =␣
 ↪'Conv2D-2'))
alexnet_cnn.add(BatchNormalization(name = 'Batch-Normalization-2'))
alexnet_cnn.add(MaxPool2D(pool_size = 3, strides = 2, name = 'Max-Pooling-2'))
alexnet_cnn.add(Conv2D(384, kernel_size = 3, padding = 'same', activation = 'relu', name =␣
 ↪'Conv2D-3'))
alexnet_cnn.add(BatchNormalization(name = 'Batch-Normalization-3'))
alexnet_cnn.add(Conv2D(384, kernel_size = 3, padding = 'same', activation = 'relu', name =␣
 ↪'Conv2D-4'))
alexnet_cnn.add(BatchNormalization(name = 'Batch-Normalization-4'))
alexnet_cnn.add(Conv2D(256, kernel_size = 3, padding = 'same', activation = 'relu', name =␣
 ↪'Conv2D-5'))
alexnet_cnn.add(BatchNormalization(name = 'Batch-Normalization-5'))
alexnet_cnn.add(MaxPool2D(pool_size = 3, strides = 2, name = 'Max-Pooling-3'))
alexnet_cnn.add(Flatten(name = 'Flatten-Layer-1'))
alexnet_cnn.add(Dense(128, activation = 'relu', name = 'Hidden-Layer-1'))
alexnet_cnn.add(Dropout(rate = 0.5, name = 'Dropout-Layer-1'))
alexnet_cnn.add(Dense(64, activation = 'relu', name = 'Hidden-Layer-2'))
alexnet_cnn.add(Dropout(rate = 0.5, name = 'Dropout-Layer-2'))
alexnet_cnn.add(Dense(4, activation = 'softmax', name = 'Output-Layer'))
alexnet_cnn.compile(optimizer = 'Adam', loss = 'categorical_crossentropy', metrics =␣
 ↪['accuracy'])
alexnet_cnn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Conv2D-1 (Conv2D)            (None, 30, 30, 96)        34944
_________________________________________________________________
Batch-Normalization-1 (Batch (None, 30, 30, 96)        384
_________________________________________________________________
Max-Pooling-1 (MaxPooling2D) (None, 14, 14, 96)        0
_________________________________________________________________
Conv2D-2 (Conv2D)            (None, 14, 14, 256)       614656
_________________________________________________________________
Batch-Normalization-2 (Batch (None, 14, 14, 256)       1024
_________________________________________________________________
Max-Pooling-2 (MaxPooling2D) (None, 6, 6, 256)         0
_________________________________________________________________
Conv2D-3 (Conv2D)            (None, 6, 6, 384)         885120
_________________________________________________________________
Batch-Normalization-3 (Batch (None, 6, 6, 384)         1536
_________________________________________________________________
Conv2D-4 (Conv2D)            (None, 6, 6, 384)         1327488
_________________________________________________________________
```

```
Batch-Normalization-4 (Batch (None, 6, 6, 384)         1536
_________________________________________________________________
Conv2D-5 (Conv2D)            (None, 6, 6, 256)         884992
_________________________________________________________________
Batch-Normalization-5 (Batch (None, 6, 6, 256)         1024
_________________________________________________________________
Max-Pooling-3 (MaxPooling2D) (None, 2, 2, 256)         0
_________________________________________________________________
Flatten-Layer-1 (Flatten)    (None, 1024)              0
_________________________________________________________________
Hidden-Layer-1 (Dense)       (None, 128)               131200
_________________________________________________________________
Dropout-Layer-1 (Dropout)    (None, 128)               0
_________________________________________________________________
Hidden-Layer-2 (Dense)       (None, 64)                8256
_________________________________________________________________
Dropout-Layer-2 (Dropout)    (None, 64)                0
_________________________________________________________________
Output-Layer (Dense)         (None, 4)                 260
=================================================================
Total params: 3,892,420
Trainable params: 3,889,668
Non-trainable params: 2,752
_________________________________________________________________
```

8.2.2 Training and Validation of AlexNet CNN Model

In [ ]:

```
with tf.device(gpu_device_location) if gpu_device_location else tpu_strategy.scope() if␣
↪tpu_device_location else tf.device(cpu_device_location):
    alexnet_train_history = alexnet_cnn.fit(train_dataset,
                                            batch_size = batch_size,
                                            validation_data = validation_dataset,
                                            epochs = 100,
                                            callbacks = [early_stopping,␣
↪alexnet_cp_callback])
```

```
Epoch 1/100
61/61 [==============================] - 106s 1s/step - loss: 1.9368 - accuracy: 0.3745 -
val_loss: 1.3511 - val_accuracy: 0.3361

Epoch 00001: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 2/100
61/61 [==============================] - 75s 1s/step - loss: 1.3039 - accuracy: 0.4236 -
val_loss: 1.4838 - val_accuracy: 0.2701

Epoch 00002: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 3/100
61/61 [==============================] - 75s 1s/step - loss: 1.2819 - accuracy: 0.4396 -
val_loss: 1.4569 - val_accuracy: 0.2825
```

```
Epoch 00003: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 4/100
61/61 [==============================] - 75s 1s/step - loss: 1.2649 - accuracy: 0.4824 -
val_loss: 1.2596 - val_accuracy: 0.4433

Epoch 00004: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 5/100
61/61 [==============================] - 75s 1s/step - loss: 1.1811 - accuracy: 0.5269 -
val_loss: 1.9028 - val_accuracy: 0.4124

Epoch 00005: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 6/100
61/61 [==============================] - 75s 1s/step - loss: 1.1224 - accuracy: 0.5532 -
val_loss: 1.1000 - val_accuracy: 0.5464

Epoch 00006: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 7/100
61/61 [==============================] - 75s 1s/step - loss: 1.0914 - accuracy: 0.5656 -
val_loss: 1.3219 - val_accuracy: 0.5093

Epoch 00007: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 8/100
61/61 [==============================] - 74s 1s/step - loss: 1.0605 - accuracy: 0.5816 -
val_loss: 1.0640 - val_accuracy: 0.5711

Epoch 00008: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 9/100
61/61 [==============================] - 75s 1s/step - loss: 1.0418 - accuracy: 0.5925 -
val_loss: 1.0312 - val_accuracy: 0.6041

Epoch 00009: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 10/100
61/61 [==============================] - 74s 1s/step - loss: 0.9587 - accuracy: 0.6095 -
val_loss: 1.0401 - val_accuracy: 0.5711

Epoch 00010: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 11/100
61/61 [==============================] - 74s 1s/step - loss: 0.9697 - accuracy: 0.5992 -
val_loss: 1.3941 - val_accuracy: 0.4969

Epoch 00011: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 12/100
```

```
61/61 [==============================] - 74s 1s/step - loss: 0.9032 - accuracy: 0.6426 -
val_loss: 1.0185 - val_accuracy: 0.5773

Epoch 00012: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 13/100
61/61 [==============================] - 74s 1s/step - loss: 0.8973 - accuracy: 0.6426 -
val_loss: 1.2468 - val_accuracy: 0.5546

Epoch 00013: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 14/100
61/61 [==============================] - 74s 1s/step - loss: 0.8586 - accuracy: 0.6426 -
val_loss: 0.8579 - val_accuracy: 0.6639

Epoch 00014: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 15/100
61/61 [==============================] - 74s 1s/step - loss: 0.8298 - accuracy: 0.6668 -
val_loss: 1.0546 - val_accuracy: 0.6206

Epoch 00015: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 16/100
61/61 [==============================] - 74s 1s/step - loss: 0.8412 - accuracy: 0.6606 -
val_loss: 0.8214 - val_accuracy: 0.6825

Epoch 00016: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 17/100
61/61 [==============================] - 74s 1s/step - loss: 0.8241 - accuracy: 0.6606 -
val_loss: 0.8492 - val_accuracy: 0.6825

Epoch 00017: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 18/100
61/61 [==============================] - 74s 1s/step - loss: 0.7722 - accuracy: 0.6834 -
val_loss: 1.1481 - val_accuracy: 0.5320

Epoch 00018: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 19/100
61/61 [==============================] - 74s 1s/step - loss: 0.8068 - accuracy: 0.6736 -
val_loss: 0.8517 - val_accuracy: 0.6660

Epoch 00019: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 20/100
61/61 [==============================] - 74s 1s/step - loss: 0.7517 - accuracy: 0.6942 -
val_loss: 1.0671 - val_accuracy: 0.5856

Epoch 00020: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
```

```
Checkpoints/AlexNet-CNN
Epoch 21/100
61/61 [==============================] - 74s 1s/step - loss: 0.7691 - accuracy: 0.6854 -
val_loss: 0.7761 - val_accuracy: 0.6825

Epoch 00021: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 22/100
61/61 [==============================] - 74s 1s/step - loss: 0.7211 - accuracy: 0.7030 -
val_loss: 0.7683 - val_accuracy: 0.6845

Epoch 00022: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 23/100
61/61 [==============================] - 74s 1s/step - loss: 0.7188 - accuracy: 0.7113 -
val_loss: 0.8285 - val_accuracy: 0.6412

Epoch 00023: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 24/100
61/61 [==============================] - 74s 1s/step - loss: 0.6995 - accuracy: 0.7102 -
val_loss: 0.9983 - val_accuracy: 0.6062

Epoch 00024: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 25/100
61/61 [==============================] - 74s 1s/step - loss: 0.6989 - accuracy: 0.7118 -
val_loss: 0.7648 - val_accuracy: 0.6907

Epoch 00025: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 26/100
61/61 [==============================] - 74s 1s/step - loss: 0.6745 - accuracy: 0.7082 -
val_loss: 0.7971 - val_accuracy: 0.6701

Epoch 00026: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 27/100
61/61 [==============================] - 74s 1s/step - loss: 0.6660 - accuracy: 0.7206 -
val_loss: 0.7770 - val_accuracy: 0.6866

Epoch 00027: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 28/100
61/61 [==============================] - 73s 1s/step - loss: 0.6556 - accuracy: 0.7273 -
val_loss: 0.8068 - val_accuracy: 0.6701

Epoch 00028: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 29/100
61/61 [==============================] - 74s 1s/step - loss: 0.6195 - accuracy: 0.7350 -
val_loss: 0.9703 - val_accuracy: 0.5814
```

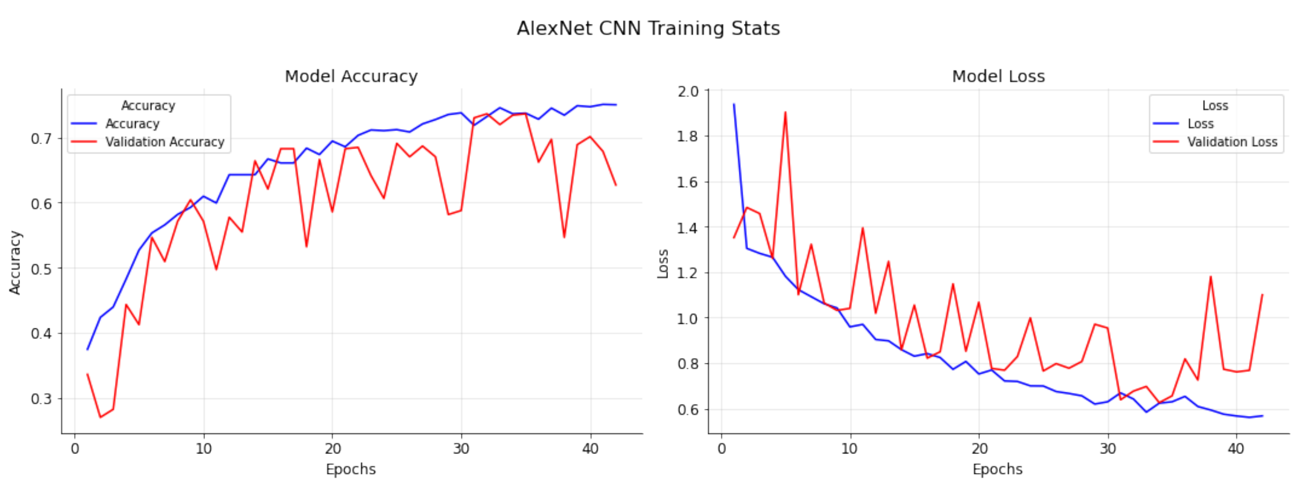
```
Epoch 00029: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 30/100
61/61 [==============================] - 74s 1s/step - loss: 0.6294 - accuracy: 0.7376 -
val_loss: 0.9537 - val_accuracy: 0.5876

Epoch 00030: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 31/100
61/61 [==============================] - 74s 1s/step - loss: 0.6685 - accuracy: 0.7185 -
val_loss: 0.6383 - val_accuracy: 0.7299

Epoch 00031: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 32/100
61/61 [==============================] - 74s 1s/step - loss: 0.6416 - accuracy: 0.7319 -
val_loss: 0.6763 - val_accuracy: 0.7361

Epoch 00032: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 33/100
61/61 [==============================] - 74s 1s/step - loss: 0.5837 - accuracy: 0.7454 -
val_loss: 0.6968 - val_accuracy: 0.7196

Epoch 00033: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 34/100
61/61 [==============================] - 74s 1s/step - loss: 0.6233 - accuracy: 0.7361 -
val_loss: 0.6255 - val_accuracy: 0.7340

Epoch 00034: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 35/100
61/61 [==============================] - 74s 1s/step - loss: 0.6294 - accuracy: 0.7371 -
val_loss: 0.6555 - val_accuracy: 0.7361

Epoch 00035: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 36/100
61/61 [==============================] - 74s 1s/step - loss: 0.6532 - accuracy: 0.7278 -
val_loss: 0.8182 - val_accuracy: 0.6619

Epoch 00036: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 37/100
61/61 [==============================] - 74s 1s/step - loss: 0.6086 - accuracy: 0.7448 -
val_loss: 0.7252 - val_accuracy: 0.6969

Epoch 00037: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 38/100
```

```
61/61 [==============================] - 74s 1s/step - loss: 0.5932 - accuracy: 0.7340 -
val_loss: 1.1803 - val_accuracy: 0.5464

Epoch 00038: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 39/100
61/61 [==============================] - 74s 1s/step - loss: 0.5749 - accuracy: 0.7485 -
val_loss: 0.7726 - val_accuracy: 0.6887

Epoch 00039: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 40/100
61/61 [==============================] - 74s 1s/step - loss: 0.5670 - accuracy: 0.7469 -
val_loss: 0.7610 - val_accuracy: 0.7010

Epoch 00040: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 41/100
61/61 [==============================] - 74s 1s/step - loss: 0.5610 - accuracy: 0.7505 -
val_loss: 0.7677 - val_accuracy: 0.6784

Epoch 00041: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
Epoch 42/100
61/61 [==============================] - 74s 1s/step - loss: 0.5674 - accuracy: 0.7500 -
val_loss: 1.0997 - val_accuracy: 0.6268

Epoch 00042: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/AlexNet-CNN
```

8.2.3 AlexNet CNN Model Training Process Statistics

In [ ]:

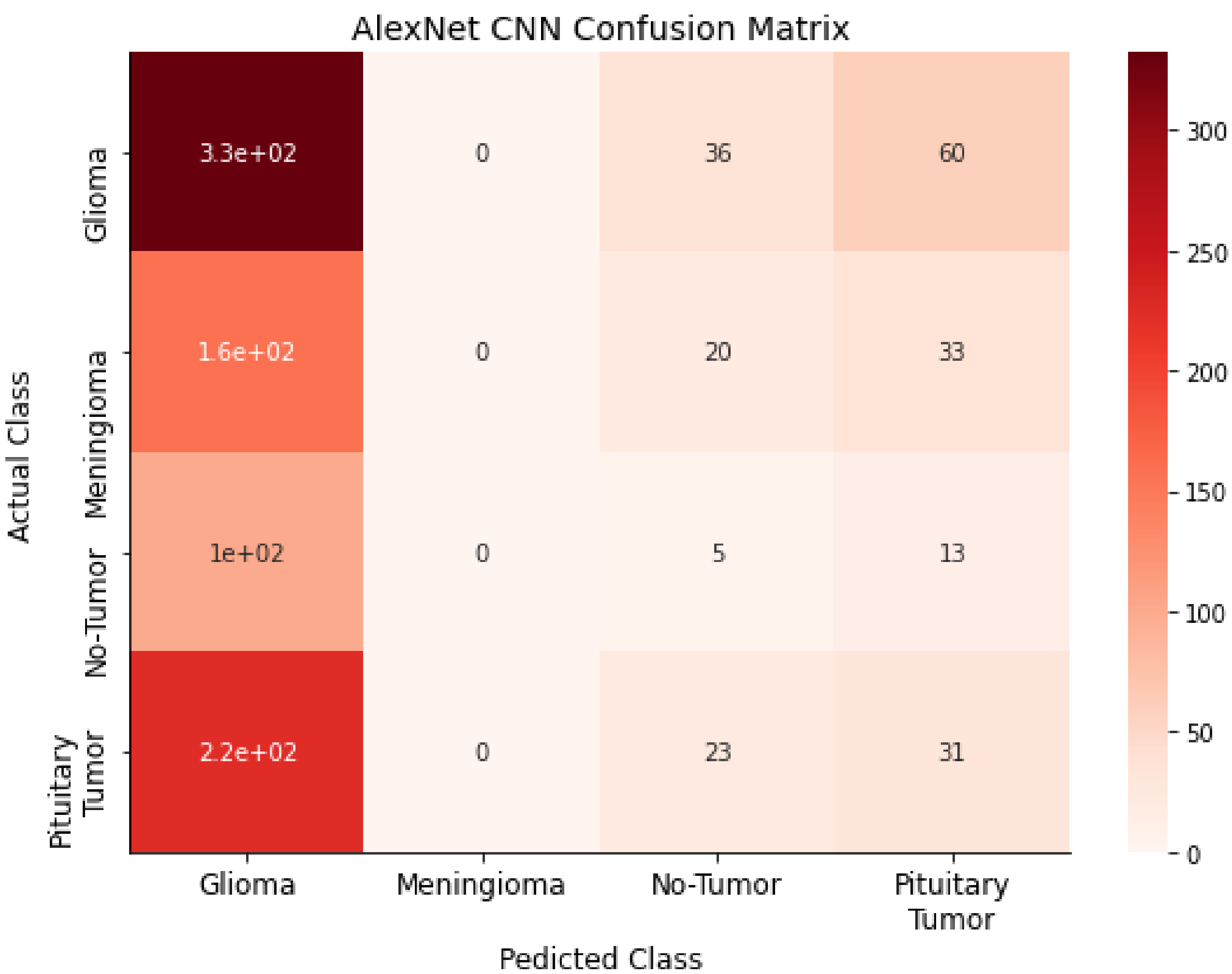
```
training_process_viz(pd.DataFrame(alexnet_train_history.history), plot_title = 'AlexNet CNN␣
↪Training Stats')
```



8.2.4 Confusion Matrix for AlexNet CNN Model

In [ ]:

```
with tf.device(gpu_device_location) if gpu_device_location else tpu_strategy.scope() if␣
↪tpu_device_name else tf.device(cpu_device_location):
    confusion_matrix_viz(alexnet_cnn,
                         test_dataset,
                         plot_title = "AlexNet CNN Confusion Matrix")
```



In [ ]:

```
alexnet_report_df = generate_report(alexnet_cnn, test_dataset = test_dataset, row_indexes =␣
↪['AlexNet CNN'])
alexnet_report_df
```

```
33/33 [==============================] - 33s 988ms/step - loss: 1.1357 - accuracy: 0.6079
```

Out [ ]:

```
              MAE       MSE      RMSE      Loss  Accuracy  F1-Score
AlexNet CNN  1.301541  3.205202  1.790308  1.135692    0.6079  0.382466
```

8.3 Inception V3

8.3.1 Developement of InceptionV3

In [ ]:

```
inception_v3_model = InceptionV3(include_top = False,
                                 input_shape = (image_size, image_size, 3),
                                 pooling = 'avg')
inception_v3_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 1s 0us/step
```

In [ ]:

```
inception_cnn_model = Sequential()
inception_cnn_model.add(inception_v3_model)
inception_cnn_model.add(Flatten())
inception_cnn_model.add(Dense(1024, activation = 'relu', name = 'Hidden-Layer-1'))
inception_cnn_model.add(Dense(4, activation = 'softmax', name = 'Output-Layer'))
inception_cnn_model.compile(optimizer = 'Adam', loss = 'categorical_crossentropy', metrics␣
 ↪= ['accuracy'])
inception_cnn_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 2048)              21802784
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
Hidden-Layer-1 (Dense)       (None, 1024)              2098176
_________________________________________________________________
Output-Layer (Dense)         (None, 4)                 4100
=================================================================
Total params: 23,905,060
Trainable params: 2,102,276
Non-trainable params: 21,802,784
_________________________________________________________________
```

8.3.2 Training and Validation of InceptionV3 Model

In [ ]:

```
with tf.device(gpu_device_location) if gpu_device_location else tpu_strategy.scope() if␣
 ↪tpu_device_location else tf.device(cpu_device_location):
    inception_model_train_history = inception_cnn_model.fit(train_dataset,
```

```
                                                   batch_size = batch_size,
                                                   validation_data =␣
↪validation_dataset,

                                                   epochs = 100,
                                                   callbacks = [early_stopping,␣
↪inceptionv3_cp_callback])
```

```
Epoch 1/100
61/61 [==============================] - 83s 1s/step - loss: 1.8779 - accuracy: 0.6012 -
val_loss: 0.7379 - val_accuracy: 0.7113

Epoch 00001: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 2/100
61/61 [==============================] - 76s 1s/step - loss: 0.6863 - accuracy: 0.7335 -
val_loss: 0.6218 - val_accuracy: 0.7979

Epoch 00002: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 3/100
61/61 [==============================] - 76s 1s/step - loss: 0.6144 - accuracy: 0.7670 -
val_loss: 0.6042 - val_accuracy: 0.7835

Epoch 00003: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 4/100
61/61 [==============================] - 76s 1s/step - loss: 0.5593 - accuracy: 0.7774 -
val_loss: 0.8480 - val_accuracy: 0.6536

Epoch 00004: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 5/100
61/61 [==============================] - 76s 1s/step - loss: 0.5345 - accuracy: 0.7887 -
val_loss: 0.5721 - val_accuracy: 0.7732

Epoch 00005: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 6/100
61/61 [==============================] - 75s 1s/step - loss: 0.4957 - accuracy: 0.8135 -
val_loss: 0.6748 - val_accuracy: 0.7485

Epoch 00006: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 7/100
61/61 [==============================] - 75s 1s/step - loss: 0.4660 - accuracy: 0.8156 -
val_loss: 0.5046 - val_accuracy: 0.8000

Epoch 00007: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 8/100
61/61 [==============================] - 75s 1s/step - loss: 0.4784 - accuracy: 0.8171 -
val_loss: 0.6136 - val_accuracy: 0.7649
```

```
Epoch 00008: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 9/100
61/61 [==============================] - 75s 1s/step - loss: 0.4769 - accuracy: 0.8032 -
val_loss: 0.5283 - val_accuracy: 0.7876

Epoch 00009: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 10/100
61/61 [==============================] - 75s 1s/step - loss: 0.4529 - accuracy: 0.8223 -
val_loss: 0.5484 - val_accuracy: 0.8000

Epoch 00010: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 11/100
61/61 [==============================] - 75s 1s/step - loss: 0.4472 - accuracy: 0.8275 -
val_loss: 0.4995 - val_accuracy: 0.8062

Epoch 00011: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 12/100
61/61 [==============================] - 75s 1s/step - loss: 0.3815 - accuracy: 0.8456 -
val_loss: 0.5364 - val_accuracy: 0.7979

Epoch 00012: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 13/100
61/61 [==============================] - 75s 1s/step - loss: 0.4216 - accuracy: 0.8388 -
val_loss: 0.5230 - val_accuracy: 0.7835

Epoch 00013: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 14/100
61/61 [==============================] - 75s 1s/step - loss: 0.4312 - accuracy: 0.8285 -
val_loss: 0.5177 - val_accuracy: 0.7979

Epoch 00014: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 15/100
61/61 [==============================] - 75s 1s/step - loss: 0.3779 - accuracy: 0.8419 -
val_loss: 0.5387 - val_accuracy: 0.7979

Epoch 00015: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 16/100
61/61 [==============================] - 74s 1s/step - loss: 0.3753 - accuracy: 0.8476 -
val_loss: 0.5662 - val_accuracy: 0.7814

Epoch 00016: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 17/100
```

```
61/61 [==============================] - 75s 1s/step - loss: 0.3746 - accuracy: 0.8461 -
val_loss: 0.5143 - val_accuracy: 0.7938

Epoch 00017: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 18/100
61/61 [==============================] - 74s 1s/step - loss: 0.3375 - accuracy: 0.8647 -
val_loss: 0.5568 - val_accuracy: 0.7959

Epoch 00018: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 19/100
61/61 [==============================] - 74s 1s/step - loss: 0.3707 - accuracy: 0.8507 -
val_loss: 0.4909 - val_accuracy: 0.8206

Epoch 00019: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 20/100
61/61 [==============================] - 75s 1s/step - loss: 0.3774 - accuracy: 0.8507 -
val_loss: 0.6110 - val_accuracy: 0.7423

Epoch 00020: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 21/100
61/61 [==============================] - 75s 1s/step - loss: 0.3636 - accuracy: 0.8476 -
val_loss: 0.5006 - val_accuracy: 0.7959

Epoch 00021: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 22/100
61/61 [==============================] - 74s 1s/step - loss: 0.3153 - accuracy: 0.8822 -
val_loss: 0.4776 - val_accuracy: 0.8206

Epoch 00022: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 23/100
61/61 [==============================] - 75s 1s/step - loss: 0.3407 - accuracy: 0.8776 -
val_loss: 0.4448 - val_accuracy: 0.8206

Epoch 00023: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 24/100
61/61 [==============================] - 75s 1s/step - loss: 0.3756 - accuracy: 0.8538 -
val_loss: 0.4913 - val_accuracy: 0.8227

Epoch 00024: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 25/100
61/61 [==============================] - 75s 1s/step - loss: 0.3489 - accuracy: 0.8621 -
val_loss: 0.5133 - val_accuracy: 0.8206

Epoch 00025: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
```

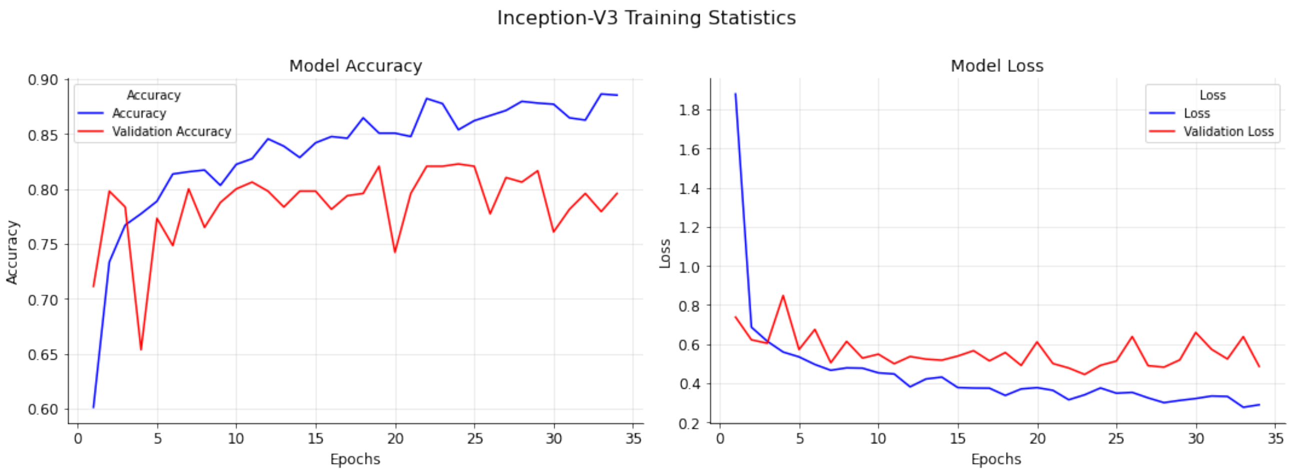
```
Checkpoints/InceptionV3
Epoch 26/100
61/61 [==============================] - 74s 1s/step - loss: 0.3531 - accuracy: 0.8667 -
val_loss: 0.6384 - val_accuracy: 0.7773

Epoch 00026: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 27/100
61/61 [==============================] - 75s 1s/step - loss: 0.3250 - accuracy: 0.8714 -
val_loss: 0.4896 - val_accuracy: 0.8103

Epoch 00027: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 28/100
61/61 [==============================] - 75s 1s/step - loss: 0.3006 - accuracy: 0.8796 -
val_loss: 0.4820 - val_accuracy: 0.8062

Epoch 00028: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 29/100
61/61 [==============================] - 75s 1s/step - loss: 0.3121 - accuracy: 0.8781 -
val_loss: 0.5193 - val_accuracy: 0.8165

Epoch 00029: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 30/100
61/61 [==============================] - 75s 1s/step - loss: 0.3217 - accuracy: 0.8771 -
val_loss: 0.6595 - val_accuracy: 0.7608

Epoch 00030: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 31/100
61/61 [==============================] - 75s 1s/step - loss: 0.3345 - accuracy: 0.8647 -
val_loss: 0.5732 - val_accuracy: 0.7814

Epoch 00031: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 32/100
61/61 [==============================] - 75s 1s/step - loss: 0.3324 - accuracy: 0.8626 -
val_loss: 0.5237 - val_accuracy: 0.7959

Epoch 00032: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 33/100
61/61 [==============================] - 75s 1s/step - loss: 0.2770 - accuracy: 0.8864 -
val_loss: 0.6379 - val_accuracy: 0.7794

Epoch 00033: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
Epoch 34/100
61/61 [==============================] - 75s 1s/step - loss: 0.2898 - accuracy: 0.8853 -
val_loss: 0.4858 - val_accuracy: 0.7959
```

```
Epoch 00034: saving model to gdrive/MyDrive/Deep_Learning_Course_Project/Model-
Checkpoints/InceptionV3
```

8.3.3 InceptionV3 Model Training Process Statistics

In [ ]:

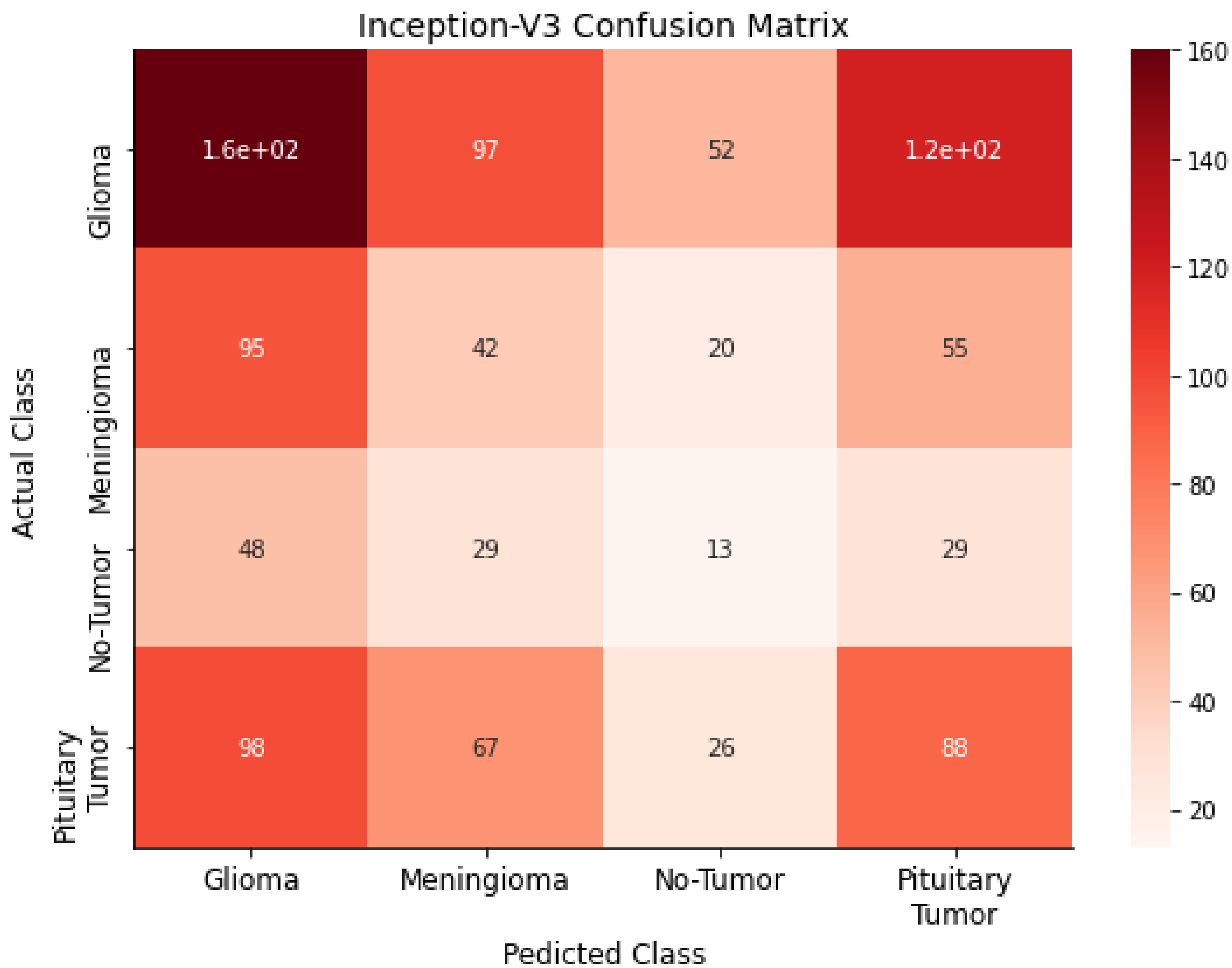
```
training_process_viz(pd.DataFrame(inception_model_train_history.history),
                     plot_title = 'Inception-V3 Training Statistics')
```



8.3.4 Confusion Matrix for InceptionV3 Model

In [ ]:

```
with tf.device(gpu_device_location) if gpu_device_location else tpu_strategy.scope() if␣
↪tpu_device_location else tf.device(cpu_device_name):
    confusion_matrix_viz(inception_cnn_model,
                         test_dataset,
                         plot_title = "Inception-V3 Confusion Matrix")
```



In [ ]:

```
inceptionv3_report_df = generate_report(inception_cnn_model, test_dataset = test_dataset,␣
↪row_indexes = ['InceptionV3'])
inceptionv3_report_df
```

```
33/33 [==============================] - 32s 962ms/step - loss: 0.4522 - accuracy: 0.8256
```

Out [ ]:

	MAE	MSE	RMSE	Loss	Accuracy	F1-Score
InceptionV3	1.365125	3.101156	1.76101	0.452249	0.825626	0.2842

9. Conclusions

- The **pre-trained (imagenet) InceptionV3** model has performed the best among Multi-Layer perceptron and AlexNet CNN models with an accuracy of 82.57% (Refer the following table).

In [ ]:

```
final_report_df = pd.concat([mlp_report_df, alexnet_report_df, inceptionv3_report_df])
final_report_df
```

Out [ ]:

```
                                MAE       MSE  …  Accuracy  F1-Score
Multi-Layer-Perceptron Model  1.394990  3.533719  …  0.624277  0.368979
AlexNet CNN                   1.301541  3.205202  …  0.607900  0.382466
InceptionV3                   1.365125  3.101156  …  0.825626  0.284200

[3 rows x 6 columns]
```

10. Future Works

- To incorporate a Data Augmentation pipeline to efficiently generate various different variants of the iamges to make the model more roboust.
- Training process will be migrated to TPUs (Tensor Processing Units) by representing the data in TFRecord format for significant reduction in training time.
- Implementation of R-CNN to not only detect a image which has a tumor in it but to also label the location of the tumor in the image.