winters_2025_brain_tumor_cnn_clasification_v1.0

March 7, 2025

- Wiley Winters
- MSDS 686 Deep Learning
- Week 7-8 Kaggle Project — Brain Tumor Classification
- 2025-MAR-09

0.1 Requirements

0.1.1 Required for 80%

Complete project on *kaggle.com* using the skills learned in the Deep Learning class. The following are required: - Show/plot sample images or data with labels - Include at least one of the following - Convolution - Max Pooling - Batch Normalization - Dropout - LSTM - TF-IDf - Use validation data - Evaluate model on test data

---

0.2 Additional for another 20%

- Use data augmentation
- Use at least one of the following:
  - Kernels
  - Activation functions
  - Loss functions
  - Libraries
  - Methods
- Learning rate optimization
- Functional API model
- Transfer learning with or without trainable parameters
- Confusion matrix and / or ROC plots
- Plots of accuracy/loss vs epochs
- Show/plot sample incorrect prediction with labels and correct label

---

1.0 | Load Libraries and Packages

[1]:
```
# General Imports
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os, logging, random
from datetime import datetime

# Data prep and model scoring
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import classification_report

# TensorFlow likes to display a lot of debug information
# on my home system
# I will squash the messages
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
logging.getLogger('tensorFlow').setLevel(logging.FATAL)

# tensorflow and keras' API
import tensorflow as tf
from tensorflow import keras

# Model building
from tensorflow.keras import backend, optimizers, regularizers, models
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Model architecture visualization
from visualkeras import layered_view

# Model training
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.metrics import Precision, Recall, AUC

# Make plots have guidelines
plt.style.use('ggplot')

# Squash Python warnings
import warnings
warnings.filterwarnings('ignore')
```

```
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
E0000 00:00:1741353098.026237 1061755 cuda_dnn.cc:8310] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already
been registered
E0000 00:00:1741353098.032022 1061755 cuda_blas.cc:1418] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
```

```
already been registered
```

1.1 | Set Random Seed for Reproducibility

[2]:
```
tf.keras.utils.set_random_seed(42)
tf.random.set_seed(42)
np.random.seed(42)
random.seed(42)
```

1.2 | Declare Global Variables

[3]:
```
# Define training and testing image directories
home_dir = '/home/wiley'
trn_dir = home_dir+'/regis/dataScience/kaggleProject/images/data/training'
tst_dir = home_dir+'/regis/dataScience/kaggleProject/images/data/testing'

# Define classes
classes = ['negative', 'positive']

# Image size and shape
img_size = (224, 224)
img_shape = (224, 224, 3)

# Number of classes
num_classes = 2

# Declare batch size
batch_size = 64

# Flag to save weights
save = True
```

2.0 | Define Functions

2.1 | Load DataFrames - Join image filename and path information - Create labels from class directory names - Create dataframe - Randomize dataframe rows

[4]:
```
def load_dataframe(path):
    # Derive image file paths and labels from directory structure
    labels, paths = zip(*[(label, os.path.join(path, label, image))
                          for label in os.listdir(path)
                          if os.path.isdir(os.path.join(path, label))
                          for image in os.listdir(os.path.join(path, label))])

    # Create DataFrame
    df = pd.DataFrame({'paths': paths, 'labels': labels})

    # Randomize rows to help eliminate bias
```

```
    df = df.sample(frac=1, random_state=42).reset_index(drop=True)

    return df
```

2.2 | Plot Performance Metrics Plot the following: - Training loss - Validation loss - Training Accuracy - Validation Accuracy - Training Precision - Validation Precision - Training Recall - Validation Recall - Training AUC - Validation AUC

[5]:
```
def plot_history(history):
    epochs = range(1, len(history.history['accuracy']) + 1)

    # Plot training and validation loss
    plt.figure(figsize=(20,12))
    plt.subplot(2,2,1)
    plt.plot(epochs, history.history['loss'], 'b', label = 'Training Loss')
    plt.plot(epochs, history.history['val_loss'], 'r', label = 'Validation␣
↪Loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    # Plot training and validation accuracy
    plt.subplot(2,2,2)
    plt.plot(epochs, history.history['accuracy'], 'b', label = 'Training␣
↪Accuracy')
    plt.plot(epochs, history.history['val_accuracy'], 'r', label = 'Validation␣
↪Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.suptitle('Model Loss and Accuracy over Epochs', fontsize=16)
    plt.show()

    # Plot training and validation precision
    plt.figure(figsize=(20,12))
    plt.subplot(2,2,1)
    plt.plot(epochs, history.history['precision'], 'b', label='Training␣
↪Precision')
    plt.plot(epochs, history.history['val_precision'], 'r', label='Validation␣
↪Precision')
    plt.title('Training and Validation Precision')
    plt.xlabel('Epochs')
    plt.ylabel('Precision')
    plt.legend()
```

```
    # Plot training and validation recall
    plt.subplot(2,2,2)
    plt.plot(epochs, history.history['recall'], 'b', label='Training Recall')
    plt.plot(epochs, history.history['val_recall'], 'r', label='Validation␣
↪Recall')
    plt.title('Training and Validation Recall')
    plt.xlabel('Epochs')
    plt.ylabel('Recall')
    plt.legend()

    plt.suptitle('Model Precision and Recall over Epochs', fontsize=16)
    plt.show()

    # Plot training and validation AUC
    plt.figure(figsize=(5,3))
    plt.plot(epochs, history.history['auc'], 'b', label='Training AUC')
    plt.plot(epochs, history.history['val_auc'], 'r', label='Validation AUC')
    plt.title('Training and Validation AUC')
    plt.xlabel('Epochs')
    plt.ylabel('Recall')
    plt.legend()
    plt.show()
```

2.3 | Evaluate Model's Performance on Test DataSet - Infer loss, accuracy, precision, recall, and AUC from dataset - Compute F1 Score from precision and recall

[6]:
```
def score_model(model, ds):
    # Get metrics from test data
    loss, acc, auc, prec, recall = model.evaluate(ds)

    # Calculate F1 Score from precision and recall
    f1_score = 2 * (prec * recall) / (prec + recall)

    # Print results
    print('-' * 30)
    print(f'Loss:      {loss:.4f}')
    print(f'Accuracy:  {acc:.4f}')
    print(f'Precision: {prec:.4f}')
    print(f'Recall:    {recall:.4f}')
    print(f'AUC:       {auc:.4f}')
    print(f'F1 Score:  {f1_score:.4f}')
    print('-' * 30)
```

2.4 | Plot Confusion Matrix

[7]:
```
def plot_cm(model, ds):
    # Get predictions from dataset
```

```
    preds = np.argmax(np.round(model.predict(ds)), axis=1)

    # Create confusion matrix
    cm = confusion_matrix(ds.classes, preds)

    # Visualize confusion matrix
    plt.figure(figsize=(5,3))
    sns.heatmap(cm, annot=True, fmt='d', cmap='viridis',
                xticklabels=classes,
                yticklabels=classes)
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()
```

2.5 | Compute TPR and TNR

[8]:
```
def compute_tpr(model, ds):
    # get predictions from dataset
    preds = np.argmax(np.round(model.predict(ds)), axis=1)

    # Create confusion matrix
    cm = confusion_matrix(ds.classes, preds)

    # Extract required values from confusion matrix
    (tn, fp, fn, tp) = cm.flatten()

    # Calculate TPR
    tpr = tp / (tp + fn)

    # Calculate TNR
    tnr = tn / (tn + fp)

    # Print TPR and TNR
    print('-' * 30)
    print(f'True Positive Rate (TPR): {tpr:.4f}')
    print(f'True Negative Rate (TNR): {tnr:.4f}')
    print('-' * 30)
```

2.6 | Show True vs Predicted Labels

[9]:
```
def display_preds(model, ds):
    # Extract true and predicted labels from dataset
    images, labels = next(ds)
    preds = model.predict(images)
    pred_labs = np.argmax(preds, axis=1)
    dict = ds.class_indices
    tr_labels = list(dict.keys())
```

```
    # Plot the images with true and predicted labels
    plt.figure(figsize=(20,20))
    for i in range(16):
        img = images[i]
        label = labels[i]
        tr_label = classes[np.argmax(label)]
        pred_label = classes[pred_labs[i]]
        plt.subplot(4,4,i+1)
        plt.imshow(img)
        color = 'green' if tr_label == pred_label else 'red'
        plt.title('True:      %s \nPredict:  %s' % (tr_label, pred_label),␣
↪fontsize=15,
                  loc='left', color=color)
        plt.axis('off')

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

2.7 | Print Images

[10]:
```
def print_images(ds):
    # Pull images and labels out of the dataset
    images, labels = next(ds)

    # Create a dictionary of class indices
    dict = ds.class_indices

    # Form classes from the dictionary created in last step
    classes = list(dict.keys())

    # Plot the images and labels -- 16 images at a time
    plt.figure(figsize=(20,20))
    for i in range(16):
        img = images[i]
        label = labels[i]
        class_name = classes[np.argmax(label)]
        plt.subplot(4,4,i+1)
        plt.imshow(img)
        plt.title(class_name, loc='left', fontsize=15)
        plt.axis('off')

    plt.show()
```

3.0 | Load Data

3.1 | Create and Load DataFrame for EDA

[11]:
```
# Load training data
trn_df = load_dataframe(trn_dir)

# Load testing data
tst_df = load_dataframe(tst_dir)

# Take a look at the results
print('Training:   \n', trn_df.head(10).to_markdown())
print('Testing:    \n', tst_df.head(10).to_markdown())
```

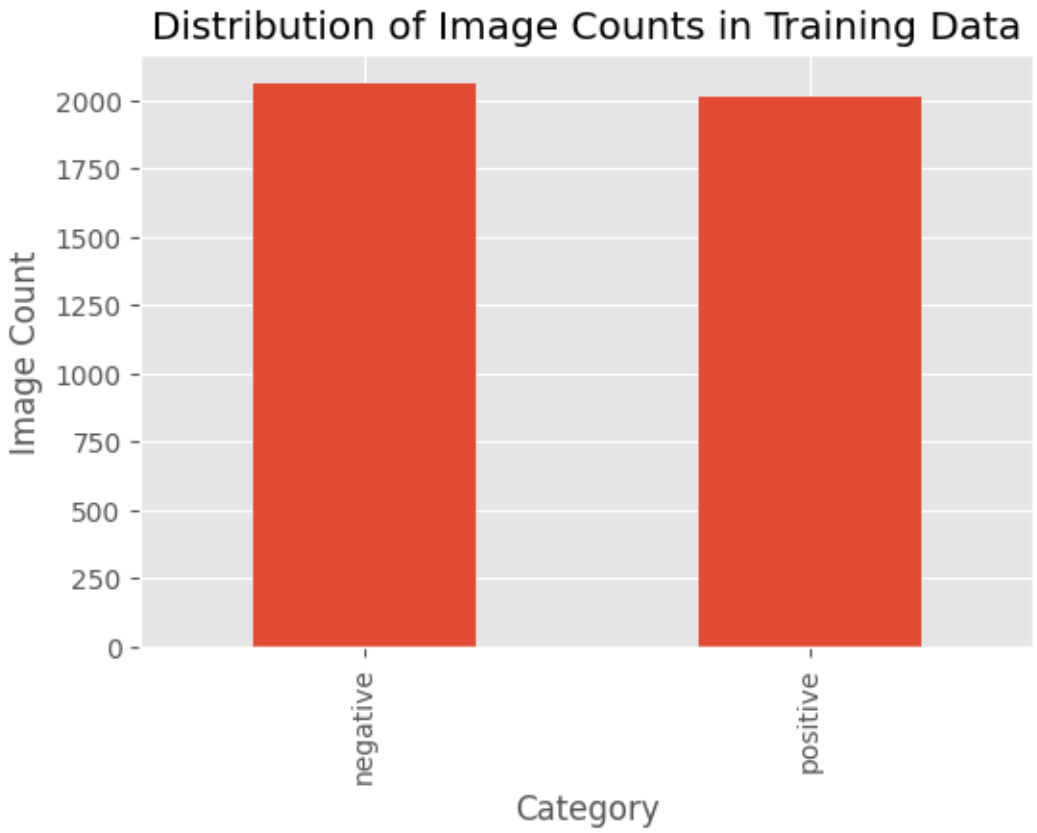
```
Training:
 |    | paths
| labels   |
|---:|:-----------------------------------------------------------------------
-----------------------|:---------|
|  0 | /home/wiley/regis/dataScience/kaggleProject/images/data/training/negative
/image(192).jpg         | negative |
|  1 | /home/wiley/regis/dataScience/kaggleProject/images/data/training/negative
/image(122).jpg         | negative |
|  2 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/negative/Not
Cancer  (769).jpg  | negative |
|  3 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/negative/Not
Cancer  (511).jpg  | negative |
|  4 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/positive/Cancer
(720).jpg       | positive |
|  5 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/negative/Not
Cancer  (468).jpg  | negative |
|  6 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/negative/Not
Cancer  (1108).jpg | negative |
|  7 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/negative/Not
Cancer  (360).jpg  | negative |
|  8 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/positive/Cancer
(402).jpg       | positive |
|  9 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/negative/Not
Cancer  (177).jpg  | negative |
Testing:
 |    | paths
| labels   |
|---:|:-----------------------------------------------------------------------
-----------------------|:---------|
```

```
|  0 |
/home/wiley/regis/dataScience/kaggleProject/images/data/testing/positive/Cancer
(1404).jpg      | positive |
|  1 |
/home/wiley/regis/dataScience/kaggleProject/images/data/testing/negative/Not
Cancer  (1168).jpg | negative |
|  2 | /home/wiley/regis/dataScience/kaggleProject/images/data/testing/negative/
image(30).jpg          | negative |
|  3 |
/home/wiley/regis/dataScience/kaggleProject/images/data/testing/negative/Not
Cancer  (720).jpg  | negative |
|  4 |
/home/wiley/regis/dataScience/kaggleProject/images/data/testing/positive/Cancer
(1559).jpg      | positive |
|  5 |
/home/wiley/regis/dataScience/kaggleProject/images/data/testing/positive/Cancer
(2272).jpg      | positive |
|  6 | /home/wiley/regis/dataScience/kaggleProject/images/data/testing/negative/
image(33).jpg          | negative |
|  7 |
/home/wiley/regis/dataScience/kaggleProject/images/data/testing/positive/Cancer
(714).jpg       | positive |
|  8 |
/home/wiley/regis/dataScience/kaggleProject/images/data/testing/negative/Not
Cancer  (1090).jpg | negative |
|  9 |
/home/wiley/regis/dataScience/kaggleProject/images/data/testing/positive/Cancer
(300).jpg       | positive |
```

4.0 | EDA

4.1 | Look at Training Images' Distribution

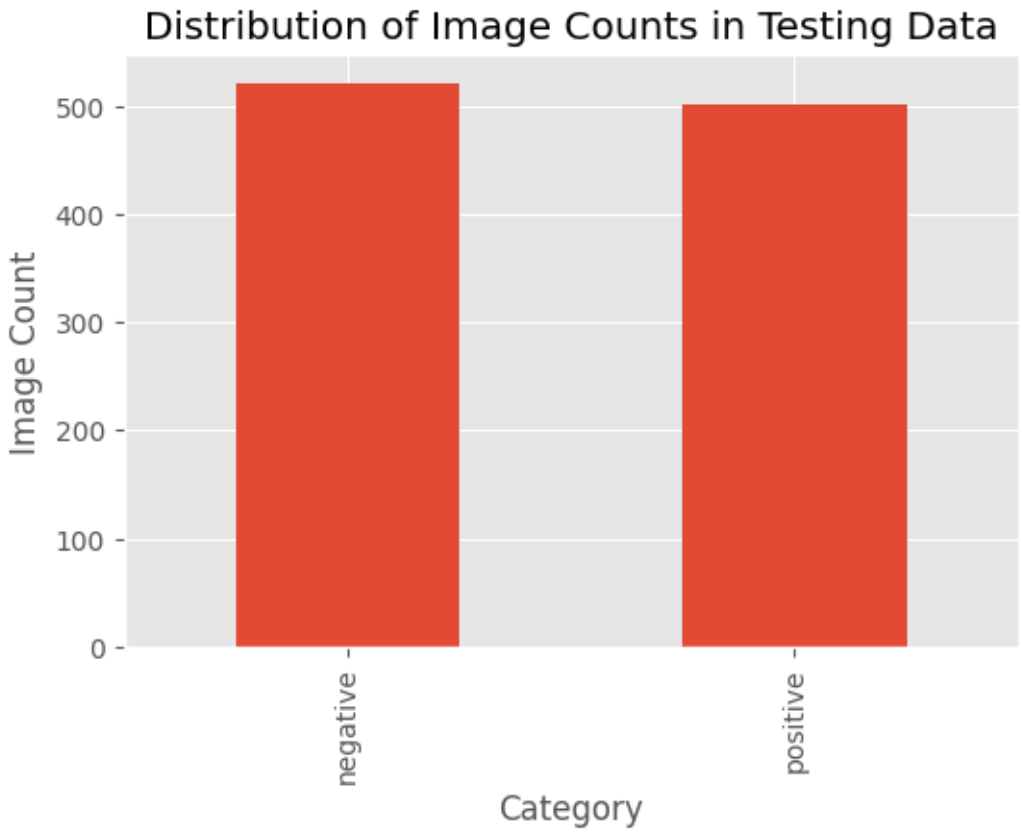
[12]:
```
plt.figure(figsize=(6,4))
trn_df['labels'].value_counts().plot(kind='bar')
plt.title('Distribution of Image Counts in Training Data')
plt.xlabel('Category')
plt.ylabel('Image Count')
plt.show()
```



Negative images slightly outnumber the positive ones, but are close enough to continue without additional data wrangling

4.2 | Look at Testing Images' Distribution

[13]:
```
plt.figure(figsize=(6,4))
tst_df['labels'].value_counts().plot(kind='bar')
plt.title('Distribution of Image Counts in Testing Data')
plt.xlabel('Category')
plt.ylabel('Image Count')
plt.show()
```



Distribution mirrors what the *training data* shows, but with less frequency.

4.3 | Examine Shape of Training and Testing DataFrames

[14]:
```
print('Training Shape: \n', trn_df.shape)
print('Testing Shape:  \n', tst_df.shape)
```

```
Training Shape:
 (4076, 2)
Testing Shape:
 (1024, 2)
```

NOTE: Since the dataframes are built from the contents of the image directories, there should be no missing values or duplicates.

4.0 | Data Wrangling

4.1 | Create a Validation Subset from Training Data I will use `flow_from_dataframe()` to create datasets for model training; therefore, no reason to create a new directory structure for validation data

[15]:
```
val_df, trn_df = train_test_split(trn_df, train_size=0.2, random_state=42,
                                  stratify=trn_df['labels'])
print(val_df.sample(10).to_markdown())
print(f'Validation Shape: {val_df.shape}')
```

```
|      | paths
| labels   |
|-----:|:-----------------------------------------------------------------------
-------------------------|:---------|
| 2341 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/positive/Cancer
(404).jpg       | positive |
| 1615 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/positive/Cancer
(830).jpg       | positive |
| 3949 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/negative/Not
Cancer  (595).jpg  | negative |
| 2158 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/positive/Cancer
(1965).jpg      | positive |
|   51 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/positive/Cancer
(1479).jpg      | positive |
| 1967 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/negative/Not
Cancer  (1274).jpg | negative |
| 3644 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/positive/Cancer
(2414).jpg      | positive |
| 2917 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/negative/Not
Cancer  (546).jpg  | negative |
| 1429 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/negative/Not
Cancer  (225).jpg  | negative |
| 2855 |
/home/wiley/regis/dataScience/kaggleProject/images/data/training/negative/Not
Cancer  (1673).jpg | negative |
Validation Shape: (815, 2)
```

4.2 | Process Images from DataFrames Image augmentation will be used on the training and validation datasets. The test images will just be normalized.

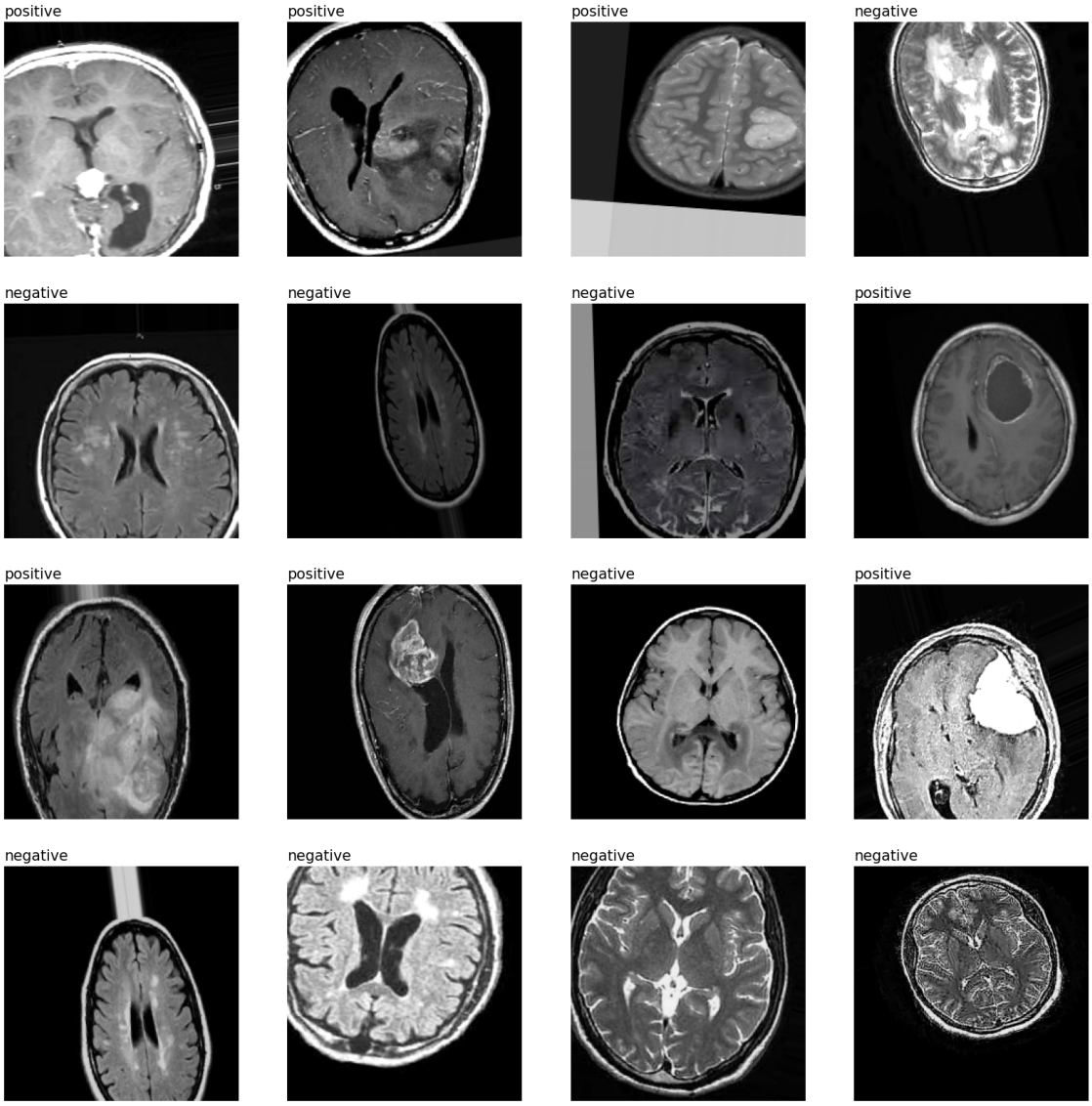
[16]:
```
# Apply image augmentation
gen = ImageDataGenerator(rescale=1./255,
                         brightness_range=(0.5, 1.5),
                         rotation_range=20,
```

```
                          width_shift_range=0.2,
                          height_shift_range=0.2,
                          shear_range=0.2,
                          zoom_range=0.2)

# The test dataset should not be augmented
# just rescaled
tst_gen = ImageDataGenerator(rescale=1./255)

# Create training datagen set
trn_gen = gen.flow_from_dataframe(trn_df, x_col='paths', y_col='labels',
                                  batch_size=batch_size, target_size=img_size,
                                  shuffle=True)

# Create validation datagen set
val_gen = gen.flow_from_dataframe(val_df, x_col='paths', y_col='labels',
                                  batch_size=batch_size, target_size=img_size,
                                  shuffle=True)

# Create test datagen set
tst_gen = tst_gen.flow_from_dataframe(tst_df, x_col='paths', y_col='labels',
                                      batch_size=16, target_size=img_size,
                                      shuffle=False)
```

```
Found 3261 validated image filenames belonging to 2 classes.
Found 815 validated image filenames belonging to 2 classes.
Found 1024 validated image filenames belonging to 2 classes.
```

4.3 | Examine a few Augmented Images and their Labels The images displayed have been augmented in the previous step. The appearance may not be consistent with non-augmented images.

[17]:
```
# Print augmented images from training dataset
print_images(trn_gen)
```



5.0 | Configure Training Values

5.1 | Basic Values

[18]:
```
# Number of training epochs
epochs = 50

# Steps per epoch
steps_per_ep = trn_gen.samples // batch_size

# Validation steps
val_steps = val_gen.samples // batch_size
```

```
print(f'Image shape:      {img_shape}')
print(f'Epochs:           {epochs}')
print(f'Batch size:       {batch_size}')
print(f'Steps per epoch:  {steps_per_ep}')
print(f'Validation steps: {val_steps}')
```

```
Image shape:      (224, 224, 3)
Epochs:           50
Batch size:       64
Steps per epoch:  50
Validation steps: 12
```

5.2 | Define Callbacks With these *callbacks* the model's training will stop if the validation loss stops decreasing (`EarlyStopping()`), and the learning rate will be reduced until the validation loss plateaus (`ReduceLROnPlateau()`)

[19]:
```
# Define early_stop callback
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.000000001,␣
 ↪patience=5,
                           baseline=None, restore_best_weights=True,␣
 ↪start_from_epoch=0)

# Define reduce LR on Plateau callback
reduceLRO = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=8,␣
 ↪mode='auto',
                              min_delta=0.0001, cooldown=0, min_lr=0)
```

6.0 | Baseline Model ### Define Model's Architecture

6.1 | Define Model's Architecture The CNN model is being defined by using `models.Sequential()` method. It consists of four convolution layers flattened into two fully connected layers with dropout. The output layer will use the *softmax* activation function instead of *relu*

[20]:
```
backend.clear_session()

model_cnn = models.Sequential([
    # Conv layer #1
    Conv2D(32, (4,4), activation='relu', input_shape=img_shape),
    MaxPooling2D(pool_size=(3,3)),

    # Conv layer #2
    Conv2D(64, (4,4), activation='relu'),
    MaxPooling2D(pool_size=(3,3)),

    # Conv layer #3
```

```
    Conv2D(128, (4,4), activation='relu'),
    MaxPooling2D(pool_size=(4,4)),

    # Conv layer #4
    Conv2D(128, (4,4), activation='relu'),
    Flatten(),

    # Fully connect layers
    Dense(512, activation='relu'),
    Dropout(0.5, seed=42),
    Dense(num_classes, activation='softmax')
])

model_cnn.summary()
```

```
I0000 00:00:1741353102.333307 1061755 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 9655 MB memory:  -> device: 0,
name: NVIDIA GeForce RTX 4070, pci bus id: 0000:09:00.0, compute capability: 8.9
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 221, 221, 32)	1,568
max_pooling2d (MaxPooling2D)	(None, 73, 73, 32)	0
conv2d_1 (Conv2D)	(None, 70, 70, 64)	32,832
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 64)	0
conv2d_2 (Conv2D)	(None, 20, 20, 128)	131,200
max_pooling2d_2 (MaxPooling2D)	(None, 5, 5, 128)	0
conv2d_3 (Conv2D)	(None, 2, 2, 128)	262,272
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 512)	262,656
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 2)	1,026

Total params: 691,554 (2.64 MB)

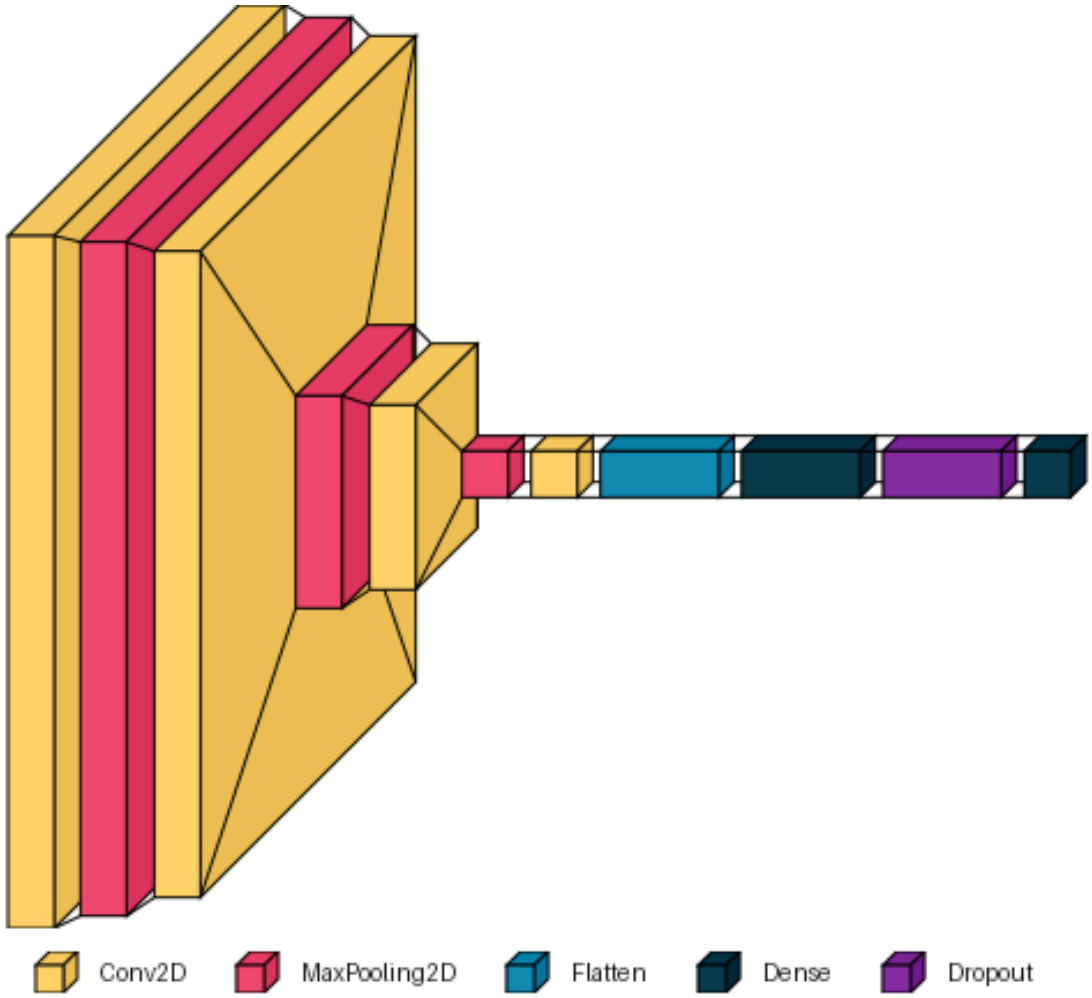
Trainable params: 691,554 (2.64 MB)

Non-trainable params: 0 (0.00 B)

6.2 | Visualize Layers

[21]:
```
layered_view(model_cnn, legend=True, max_xy=300)
```

[21]:



6.3 | Compile and Train Model The `Adam()` optimizer was selected for this model, since it is well suited to classification problems. The loss function `categorical_crossentropy()` was also selected for the same reason.

[22]:
```
# Configure Adam optimizer
opt = optimizers.Adam(learning_rate=0.001, beta_1=0.869, beta_2=0.995)

# Compile base model
model_cnn.compile(optimizer=opt, loss='categorical_crossentropy',
                  metrics=['accuracy', tf.keras.metrics.
 ↪Precision(name='precision'),
                           tf.keras.metrics.Recall(name='recall'),
                           tf.keras.metrics.AUC(curve='PR', name='auc')])

# Fit data to model and record training history
hist_cnn = model_cnn.fit(trn_gen, batch_size=batch_size,␣
 ↪steps_per_epoch=steps_per_ep,
                         epochs=epochs, validation_data=val_gen,
                         validation_steps=val_steps,
                         callbacks=[early_stop, reduceLRO])
```

```
Epoch 1/50

WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1741353106.745184 1061881 service.cc:148] XLA service 0x7f4df400e490
initialized for platform CUDA (this does not guarantee that XLA will be used).
Devices:
I0000 00:00:1741353106.745233 1061881 service.cc:156]   StreamExecutor device
(0): NVIDIA GeForce RTX 4070, Compute Capability 8.9
I0000 00:00:1741353107.080437 1061881 cuda_dnn.cc:529] Loaded cuDNN version
90300

 2/50            1s 30ms/step - accuracy:
0.5586 - auc: 0.5534 - loss: 0.6881 - precision: 0.5586 - recall: 0.5586

I0000 00:00:1741353112.356672 1061881 device_compiler.h:188] Compiled cluster
using XLA!  This line is logged at most once for the lifetime of the process.

50/50            51s 871ms/step -
accuracy: 0.5306 - auc: 0.5328 - loss: 0.6923 - precision: 0.5306 - recall:
0.5306 - val_accuracy: 0.6393 - val_auc: 0.6944 - val_loss: 0.6324 -
val_precision: 0.6393 - val_recall: 0.6393 - learning_rate: 0.0010
Epoch 2/50
50/50            2s 45ms/step -
accuracy: 0.5938 - auc: 0.6172 - loss: 0.6800 - precision: 0.5938 - recall:
0.5938 - val_accuracy: 0.6596 - val_auc: 0.6854 - val_loss: 0.6482 -
val_precision: 0.6596 - val_recall: 0.6596 - learning_rate: 0.0010
Epoch 3/50
50/50            36s 715ms/step -
accuracy: 0.6037 - auc: 0.6497 - loss: 0.6522 - precision: 0.6037 - recall:
0.6037 - val_accuracy: 0.6706 - val_auc: 0.7165 - val_loss: 0.6147 -
val_precision: 0.6706 - val_recall: 0.6706 - learning_rate: 0.0010
Epoch 4/50
```

```
50/50            0s 9ms/step -
accuracy: 0.6562 - auc: 0.7581 - loss: 0.5873 - precision: 0.6562 - recall:
0.6562 - val_accuracy: 0.6809 - val_auc: 0.7505 - val_loss: 0.5871 -
val_precision: 0.6809 - val_recall: 0.6809 - learning_rate: 0.0010
Epoch 5/50
50/50            37s 740ms/step -
accuracy: 0.6705 - auc: 0.7234 - loss: 0.6092 - precision: 0.6705 - recall:
0.6705 - val_accuracy: 0.6927 - val_auc: 0.7392 - val_loss: 0.5985 -
val_precision: 0.6927 - val_recall: 0.6927 - learning_rate: 0.0010
Epoch 6/50
50/50            0s 8ms/step -
accuracy: 0.6875 - auc: 0.7597 - loss: 0.5928 - precision: 0.6875 - recall:
0.6875 - val_accuracy: 0.6383 - val_auc: 0.7021 - val_loss: 0.6326 -
val_precision: 0.6383 - val_recall: 0.6383 - learning_rate: 0.0010
Epoch 7/50
50/50            36s 730ms/step -
accuracy: 0.7052 - auc: 0.7530 - loss: 0.5826 - precision: 0.7052 - recall:
0.7052 - val_accuracy: 0.7292 - val_auc: 0.8091 - val_loss: 0.5356 -
val_precision: 0.7292 - val_recall: 0.7292 - learning_rate: 0.0010
Epoch 8/50
50/50            0s 7ms/step -
accuracy: 0.7812 - auc: 0.8344 - loss: 0.5043 - precision: 0.7812 - recall:
0.7812 - val_accuracy: 0.7234 - val_auc: 0.7876 - val_loss: 0.5562 -
val_precision: 0.7234 - val_recall: 0.7234 - learning_rate: 0.0010
Epoch 9/50
50/50            37s 733ms/step -
accuracy: 0.7304 - auc: 0.8011 - loss: 0.5370 - precision: 0.7304 - recall:
0.7304 - val_accuracy: 0.7656 - val_auc: 0.8459 - val_loss: 0.4833 -
val_precision: 0.7656 - val_recall: 0.7656 - learning_rate: 0.0010
Epoch 10/50
50/50            0s 9ms/step -
accuracy: 0.7031 - auc: 0.8151 - loss: 0.5235 - precision: 0.7031 - recall:
0.7031 - val_accuracy: 0.7234 - val_auc: 0.8404 - val_loss: 0.4787 -
val_precision: 0.7234 - val_recall: 0.7234 - learning_rate: 0.0010
Epoch 11/50
50/50            37s 744ms/step -
accuracy: 0.7620 - auc: 0.8408 - loss: 0.4894 - precision: 0.7620 - recall:
0.7620 - val_accuracy: 0.8112 - val_auc: 0.8857 - val_loss: 0.4263 -
val_precision: 0.8112 - val_recall: 0.8112 - learning_rate: 0.0010
Epoch 12/50
50/50            0s 8ms/step -
accuracy: 0.8594 - auc: 0.9352 - loss: 0.3441 - precision: 0.8594 - recall:
0.8594 - val_accuracy: 0.7447 - val_auc: 0.8786 - val_loss: 0.4425 -
val_precision: 0.7447 - val_recall: 0.7447 - learning_rate: 0.0010
Epoch 13/50
50/50            37s 740ms/step -
accuracy: 0.7924 - auc: 0.8668 - loss: 0.4509 - precision: 0.7924 - recall:
0.7924 - val_accuracy: 0.8112 - val_auc: 0.8961 - val_loss: 0.4093 -
```

```
val_precision: 0.8112 - val_recall: 0.8112 - learning_rate: 0.0010
Epoch 14/50
50/50             0s 8ms/step -
accuracy: 0.9375 - auc: 0.9503 - loss: 0.3114 - precision: 0.9375 - recall:
0.9375 - val_accuracy: 0.7872 - val_auc: 0.8959 - val_loss: 0.4117 -
val_precision: 0.7872 - val_recall: 0.7872 - learning_rate: 0.0010
Epoch 15/50
50/50             37s 741ms/step -
accuracy: 0.8155 - auc: 0.8947 - loss: 0.4080 - precision: 0.8155 - recall:
0.8155 - val_accuracy: 0.8008 - val_auc: 0.8899 - val_loss: 0.4169 -
val_precision: 0.8008 - val_recall: 0.8008 - learning_rate: 0.0010
Epoch 16/50
50/50             0s 8ms/step -
accuracy: 0.8438 - auc: 0.9166 - loss: 0.3569 - precision: 0.8438 - recall:
0.8438 - val_accuracy: 0.8085 - val_auc: 0.8743 - val_loss: 0.4356 -
val_precision: 0.8085 - val_recall: 0.8085 - learning_rate: 0.0010
Epoch 17/50
50/50             36s 732ms/step -
accuracy: 0.8149 - auc: 0.9030 - loss: 0.3939 - precision: 0.8149 - recall:
0.8149 - val_accuracy: 0.8203 - val_auc: 0.9057 - val_loss: 0.3907 -
val_precision: 0.8203 - val_recall: 0.8203 - learning_rate: 0.0010
Epoch 18/50
50/50             0s 8ms/step -
accuracy: 0.8438 - auc: 0.9221 - loss: 0.3516 - precision: 0.8438 - recall:
0.8438 - val_accuracy: 0.7447 - val_auc: 0.8458 - val_loss: 0.5851 -
val_precision: 0.7447 - val_recall: 0.7447 - learning_rate: 0.0010
Epoch 19/50
50/50             37s 736ms/step -
accuracy: 0.8277 - auc: 0.8998 - loss: 0.4019 - precision: 0.8277 - recall:
0.8277 - val_accuracy: 0.8424 - val_auc: 0.9076 - val_loss: 0.3950 -
val_precision: 0.8424 - val_recall: 0.8424 - learning_rate: 0.0010
Epoch 20/50
50/50             0s 9ms/step -
accuracy: 0.8438 - auc: 0.8893 - loss: 0.4323 - precision: 0.8438 - recall:
0.8438 - val_accuracy: 0.8298 - val_auc: 0.9323 - val_loss: 0.3585 -
val_precision: 0.8298 - val_recall: 0.8298 - learning_rate: 0.0010
Epoch 21/50
50/50             36s 726ms/step -
accuracy: 0.8385 - auc: 0.9164 - loss: 0.3666 - precision: 0.8385 - recall:
0.8385 - val_accuracy: 0.8320 - val_auc: 0.9117 - val_loss: 0.3758 -
val_precision: 0.8320 - val_recall: 0.8320 - learning_rate: 0.0010
Epoch 22/50
50/50             0s 8ms/step -
accuracy: 0.8438 - auc: 0.9418 - loss: 0.3178 - precision: 0.8438 - recall:
0.8438 - val_accuracy: 0.8298 - val_auc: 0.9107 - val_loss: 0.3680 -
val_precision: 0.8298 - val_recall: 0.8298 - learning_rate: 0.0010
Epoch 23/50
50/50             36s 724ms/step -
```

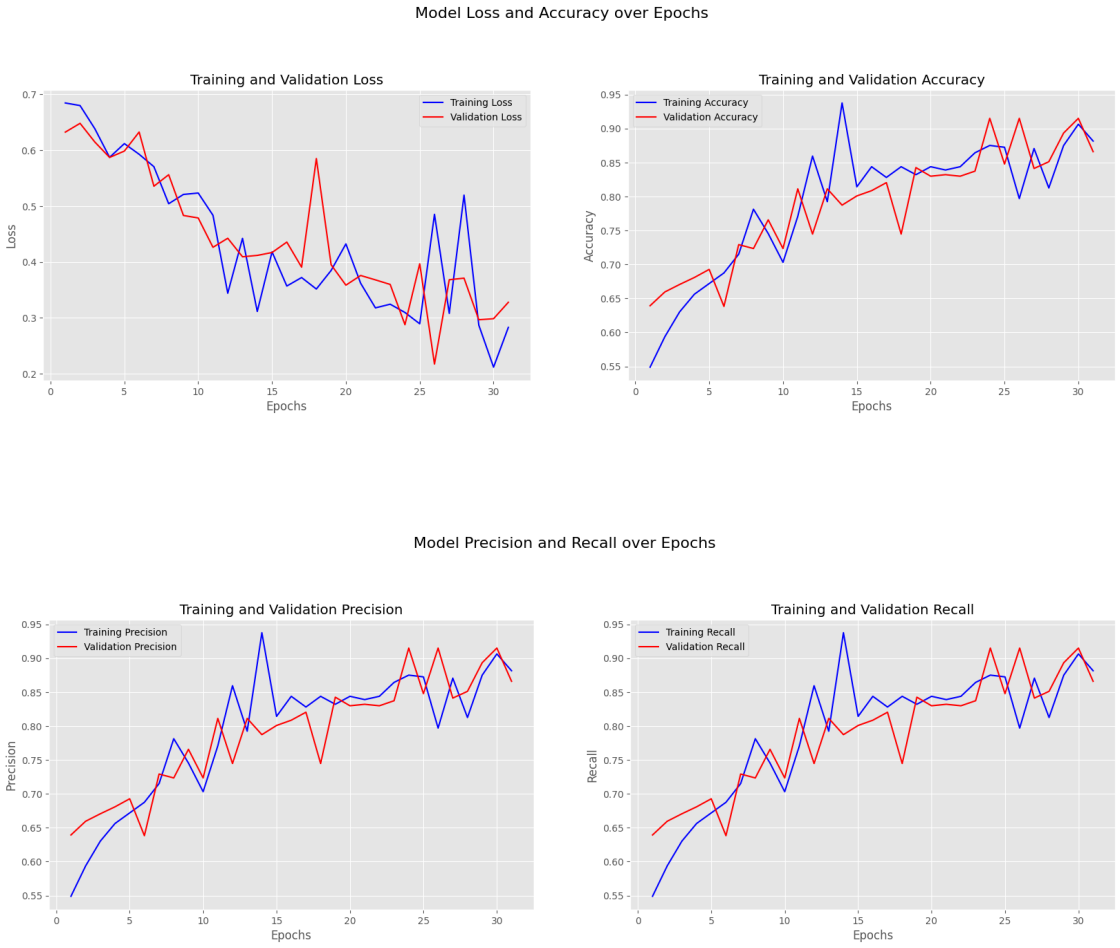
```
accuracy: 0.8650 - auc: 0.9297 - loss: 0.3365 - precision: 0.8650 - recall:
0.8650 - val_accuracy: 0.8372 - val_auc: 0.9212 - val_loss: 0.3598 -
val_precision: 0.8372 - val_recall: 0.8372 - learning_rate: 0.0010
Epoch 24/50
50/50            0s 8ms/step -
accuracy: 0.8750 - auc: 0.9436 - loss: 0.3096 - precision: 0.8750 - recall:
0.8750 - val_accuracy: 0.9149 - val_auc: 0.9532 - val_loss: 0.2877 -
val_precision: 0.9149 - val_recall: 0.9149 - learning_rate: 0.0010
Epoch 25/50
50/50            38s 758ms/step -
accuracy: 0.8708 - auc: 0.9478 - loss: 0.2942 - precision: 0.8708 - recall:
0.8708 - val_accuracy: 0.8477 - val_auc: 0.9152 - val_loss: 0.3967 -
val_precision: 0.8477 - val_recall: 0.8477 - learning_rate: 0.0010
Epoch 26/50
50/50            0s 8ms/step -
accuracy: 0.7969 - auc: 0.8602 - loss: 0.4852 - precision: 0.7969 - recall:
0.7969 - val_accuracy: 0.9149 - val_auc: 0.9760 - val_loss: 0.2173 -
val_precision: 0.9149 - val_recall: 0.9149 - learning_rate: 0.0010
Epoch 27/50
50/50            35s 694ms/step -
accuracy: 0.8734 - auc: 0.9456 - loss: 0.2979 - precision: 0.8734 - recall:
0.8734 - val_accuracy: 0.8411 - val_auc: 0.9185 - val_loss: 0.3682 -
val_precision: 0.8411 - val_recall: 0.8411 - learning_rate: 0.0010
Epoch 28/50
50/50            0s 8ms/step -
accuracy: 0.8125 - auc: 0.8774 - loss: 0.5196 - precision: 0.8125 - recall:
0.8125 - val_accuracy: 0.8511 - val_auc: 0.9133 - val_loss: 0.3710 -
val_precision: 0.8511 - val_recall: 0.8511 - learning_rate: 0.0010
Epoch 29/50
50/50            35s 694ms/step -
accuracy: 0.8758 - auc: 0.9538 - loss: 0.2780 - precision: 0.8758 - recall:
0.8758 - val_accuracy: 0.8932 - val_auc: 0.9438 - val_loss: 0.2967 -
val_precision: 0.8932 - val_recall: 0.8932 - learning_rate: 0.0010
Epoch 30/50
50/50            0s 8ms/step -
accuracy: 0.9062 - auc: 0.9808 - loss: 0.2117 - precision: 0.9062 - recall:
0.9062 - val_accuracy: 0.9149 - val_auc: 0.9371 - val_loss: 0.2984 -
val_precision: 0.9149 - val_recall: 0.9149 - learning_rate: 0.0010
Epoch 31/50
50/50            35s 694ms/step -
accuracy: 0.8754 - auc: 0.9506 - loss: 0.2864 - precision: 0.8754 - recall:
0.8754 - val_accuracy: 0.8659 - val_auc: 0.9352 - val_loss: 0.3278 -
val_precision: 0.8659 - val_recall: 0.8659 - learning_rate: 0.0010
```

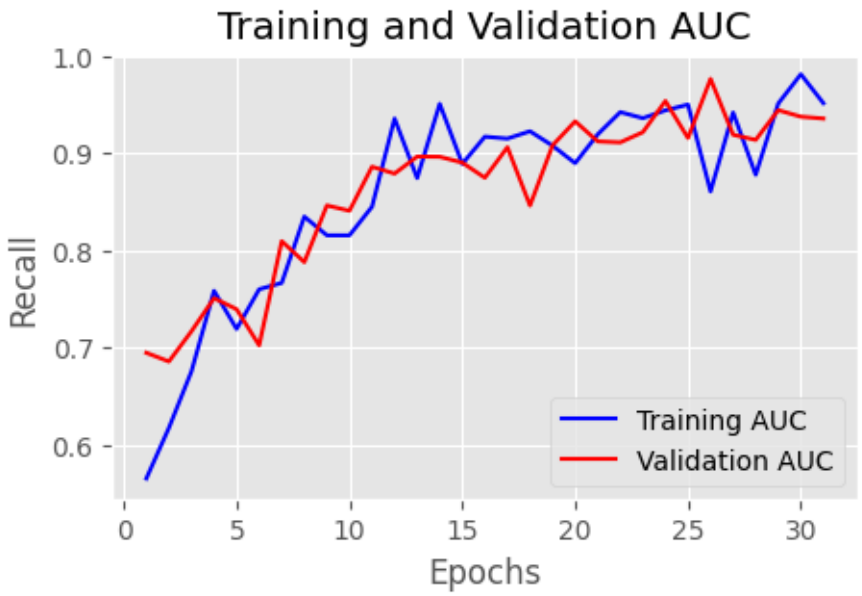
7.0 | Evaluate Performance

7.1 | Plot Training and Validation Metrics If the training and validation metrics diverge

significantly from each other, that can be an indication of overfitting.

[23]:
```
plot_history(hist_cnn)
```





7.2 | Score Model To evaluate the model's performance the following matrices will be evaluated against the test dataset: - **Model Loss** — gives a nuanced view of model optimization - **Model Accuracy** — provides the proportion of all classifications that were correct - **Precision** — shows how often a model is correct when predicting the target class - **Recall** — displays whether a model can find all objects of the target class - **Area Under Curve (AUC)** — compares the *true positive rate (TPR)* against the *false positive rate (FPR)* and shows how well the model distinguishes between the two classes - **F1 Score** — describes the harmonic mean of the precision and recall of the model

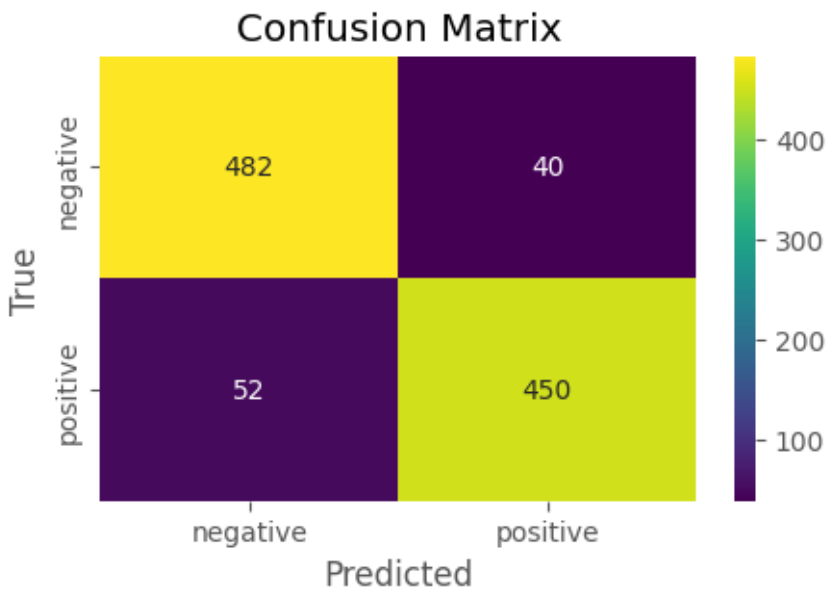
[24]:
```
score_model(model_cnn, tst_gen)
```

```
64/64              3s 22ms/step -
accuracy: 0.8981 - auc: 0.9610 - loss: 0.2521 - precision: 0.8981 - recall:
0.8981
------------------------------
Loss:      0.2208
Accuracy:  0.9102
Precision: 0.9102
Recall:    0.9696
AUC:       0.9102
F1 Score:  0.9389
------------------------------
```

7.3 | Plot Confusion Matrix A confusion matrix provides a visual representation of a model's performance when it comes to comparing true positives, false negatives, true negatives, and false positives.

[25]:
```
plot_cm(model_cnn, tst_gen)
```

```
64/64              2s 23ms/step
```



7.4 | Compute TPR and TNR The True Positive Rate (TPR) and True Negative Rate (TNR) are good indicators of how well the model is predicting positives (1s) and negatives (0s).

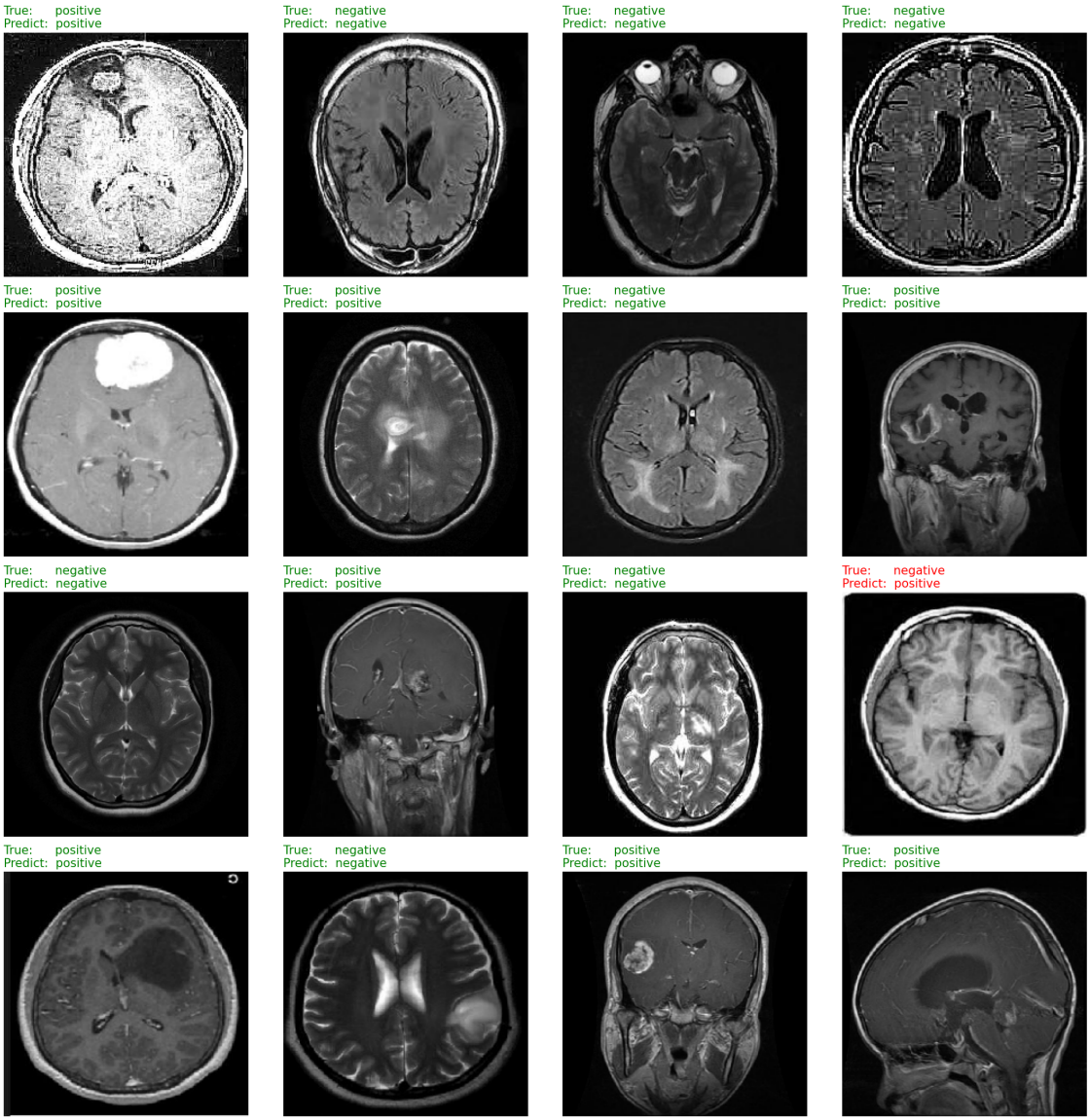
[26]:
```
compute_tpr(model_cnn, tst_gen)
```

```
64/64              1s 20ms/step
------------------------------
True Positive Rate (TPR): 0.8964
True Negative Rate (TNR): 0.9234
------------------------------
```

7.5 | View and Compare Predicted with True Labels If the ***True*** and ***Predict*** labels match, then the model accurately predicted the label of the image. This plot gives a visual representation of how well the model makes predictions between the two classes.

[27]:
```
display_preds(model_cnn, tst_gen)
```

```
1/1              0s 421ms/step
```



8.0 | Save Weights Weights can be used later for inference.

---

[28]:
```
# Use date and time to create a unique filename
if save:
    now = datetime.now()
    date = now.strftime('%Y%m%d')
    time = now.strftime('%H%M%S')
    filename = date+time+'_cnn_brain_tumor.keras'

    # Save weights in keras zip format
    model_cnn.save('../weights/'+filename)
```

9.0 | End of Notebook

A companion paper that discusses the methods used and results can be found here