

**Universidad Complutense de Madrid**  
**Máster Big Data, Data Science & Inteligencia Artificial**



Trabajo de Final de Máster

**Redes neuronales convolucionales (CNN) en radiología: Automatización de la  
clasificación de tumores cerebrales a partir de MRI.**

Presentado por  
Cristopher Marcial Ortiz Hernández

Septiembre 2025

## Índice

1.	Introducción .....	1
1.1	Contextualización .....	1
1.2	Objetivos .....	1
	Objetivo principal .....	1
	Objetivos específicos .....	1
2.	Marco teórico .....	2
2.1	Tumores cerebrales y diagnóstico por imágenes .....	2
2.2	Redes convolucionales y su aplicación en medicina .....	2
2.3	Explicabilidad en IA (XAI) .....	2
2.4	Modelos de lenguaje de gran tamaño en medicina .....	3
2.5	Integración clínica y consideraciones éticas .....	3
3.	Metodología .....	4
3.1	Exploración de imágenes MRI .....	4
3.2	Procesamiento de imágenes MRI .....	4
3.3	Experimentación redes neuronales convolucionales CNN .....	6
	Modelo 1 – Transfer learning EfficientNetV2B0 .....	6
	Modelo 2 – Transfer learning Xception .....	8
	Modelo 3 – CNN desde cero .....	10
3.4	Selección del mejor modelo .....	12
4.	Productivización .....	14
4.1	Inferencia y Grad-CAM .....	14
4.2	Modelo de lenguaje de gran tamaño para interpretación .....	15
4.3	Control de versiones .....	16
4.4	Interfaz Gráfica .....	16
5.	Conclusiones y trabajo futuro .....	19
5.1	Conclusiones .....	19
5.2	Trabajo a futuro .....	19
6.	Referencias .....	20

7. Anexos.....	21
7.1 Experimentos Notebook.....	21
7.2 Inferencia y Grad-CAM.....	57
7.3 Prompt.....	59
7.4 Explicabilidad LLM .....	60
7.5 Interfaz gráfica .....	61

## Índice de Figuras

Ilustración 1 Clases del conjunto de datos Brain Tumor MRI Dataset.....	5
Ilustración 2 Distribución de datos entre los subconjuntos entrenamiento, validación y pruebas.....	5
Ilustración 3 Capas del modelo 1 transfer learning EfficientNetV2B0.....	6
Ilustración 4 Modelo 1 EfficientNetV2B0 métricas durante entrenamiento. ....	7
Ilustración 5 Modelo 1 EfficientNetV2B0 matriz de confusión. ....	7
Ilustración 6 Capas del modelo 2 transfer learning Xception.....	8
Ilustración 7 Modelo 2 Xception métricas durante entrenamiento. ....	9
Ilustración 8 Modelo 2 Xception matriz de confusión.....	9
Ilustración 9 Capas del modelo 3 construido desde cero. ....	10
Ilustración 10 Modelo 3 CNN desde cero métricas durante entrenamiento.....	11
Ilustración 11 Modelo 3 CNN desde cero matriz de confusión. ....	12
Ilustración 12 Comparación de métricas entre experimentos. ....	13
Ilustración 13 Resultado de la Inferencia <b>Glioma</b> aplicado mapa de calor generado por Grad-CAM.....	14
Ilustración 14 Resultado de la Inferencia <b>Meningioma</b> aplicado mapa de calor generado por Grad-CAM.....	15
Ilustración 15 GUI - Generación de caso de análisis.....	17
Ilustración 16 GUI - Inferencia CNN .....	17
Ilustración 17 GUI - Interpretación de resultados por LLM .....	18

## Capítulo I

### 1. Introducción

#### 1.1 Contextualización

El diagnóstico oportuno y preciso de los tumores cerebrales a partir de imágenes de resonancia magnética (MRI) es un pilar crítico en el área de oncología moderna, teniendo impacto directo en el pronóstico y la planeación del tratamiento del paciente.

La interpretación de las MRI es una tarea compleja que requiere de un equipo de radiólogos y oncólogos altamente capacitados. La volumetría de estos estudios ha aumentado sustancialmente, esta carga de trabajo puede conducir a la fatiga y consecuentemente a errores de interpretación o la omisión de hallazgos sutiles [1].

#### 1.2 Objetivos

##### *Objetivo principal*

Desarrollar e implementar un sistema basado en inteligencia artificial para la clasificación precisa y explicable de tumores cerebrales a partir de imágenes de resonancia magnética (MRI), que combine la clasificación de tumores cerebrales a través del uso de redes neuronales convolucionales, mecanismos de explicabilidad (Grad-CAM) y la generación de reportes en lenguaje natural mediante modelos de lenguaje de gran tamaño (LLMs).

##### *Objetivos específicos*

1. Diseñar, entrenar y evaluar el desempeño de distintas arquitecturas de redes neuronales convolucionales (CNN) para la clasificación multiclase de tumores cerebrales (*glioma, meningiona, no tumor, pituitary*) utilizando *Brain Tumor MIR Dataset* alojado en Kaggle como fuente de datos.
2. Integrar el algoritmo de *Gradient-weighted Class Activation Mapping* (Grad-CAM) para identificar las regiones que más influyeron en la decisión de clasificación del modelo.
3. Desarrollar un módulo de interpretación basado en un modelo de lenguaje de gran tamaño que a partir de los resultados de la predicción de la CNN y Grad-CAM genere un reporte descriptivo en lenguaje natural.

## Capítulo II

### 2. Marco teórico

#### 2.1 Tumores cerebrales y diagnóstico por imágenes

Los tumores cerebrales son masa de células anormales que proliferan de manera incontrolada dentro del encéfalo. Su diagnóstico temprano y preciso es crucial para la determinación del tratamiento adecuado.

La MRI es la técnica de preferencia para la evaluación de tumores cerebrales debido a la visualización clara del contraste de los tejidos blandos y la capacidad de proporcionar imágenes en múltiples planos y modos (secuenciales, T1, T2, T1 con contraste, FLAIR, difusión) cada secuencia representa diferentes características del tejido, lo que permite la evaluación del tamaño, ubicación y lesiones. [2]

#### 2.2 Redes convolucionales y su aplicación en medicina

La inteligencia artificial (AI), particularmente el campo de aprendizaje profundo ha demostrado un rendimiento sobresaliente en la automatización de tareas de reconocimiento de patrones en imágenes médicas.

Las redes neuronales convolucionales (CNN) son el estándar en el campo para el análisis de imágenes. Su diseño basado en el sistema visual biológico utiliza capas convolucionales y de *pooling* para detectar automáticamente características como bordes, texturas, formas y patrones específicos, que son esenciales para tareas como la clasificación o segmentación de tumores. [3]

#### 2.3 Explicabilidad en IA (XAI)

Uno de los mayores obstáculos para la adopción clínica de la IA es la falta de transparencia en como los modelos llegan a una decisión. Grad-CAM es una técnica de visualización que genera mapas de calor que resaltan las regiones de la imagen que fueron más influyentes para la decisión de clasificación de una CNN. Este método utiliza los gradientes de la última capa convolucional para producir localidades de alta resolución, mostrando en qué área “mira” el modelo. [4]

## 2.4 Modelos de lenguaje de gran tamaño en medicina

Los LLMs como GPT-5, Gemini o DeepSeek, son modelos de IA entrenados con vastos corpus de texto que pueden generar, resumir y traducir lenguaje humano de forma coherente y contextualmente relevante.

En el contexto médico, los LLMs pueden ser utilizados para transformar datos estructurados como la etiqueta de un tumor, la probabilidad, las coordenadas de un mapa de calor en informes narrativos comprensibles. Mediante el uso de ingeniería de *prompts* y el *retrieval* de información de fuentes confiables, se pueden generar resúmenes descriptivos, explicaciones y recomendaciones de próximos pasos, actuando como puente entre el output técnico del modelo y el lenguaje técnico.

## 2.5 Integración clínica y consideraciones éticas

El objetivo final de herramientas como la propuesta es actuar como un sistema de apoyo a la decisión clínica CDSS de “segunda opinión”, aumentando las capacidades del radiólogo al reducir la fatiga, la variabilidad del observador y los tiempos de diagnóstico, no reemplazando el criterio experto. [5]

La implementación de estas tecnologías conlleva consideraciones críticas sobre la privacidad de los datos, la mitigación de sesgos en los conjuntos de datos de entrenamiento y la necesidad de una validación clínica rigorosa mediante ensayos prospectivos para demostrar su eficiencia, seguridad antes de un despliegue y uso masivo.

## Capítulo III

### 3. Metodología

#### 3.1 Exploración de imágenes MRI

El conjunto de datos utilizado es *Brain Tumor MRI Dataset* publicado por Masoud Nickparvar bajo una licencia CC0 de domino público a través del enlace <https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset/data>. El conjunto de datos es diseñado específicamente para la tarea de clasificación de imágenes de resonancia magnética de tumores cerebrales.

El conjunto de datos contiene 4 clases distintas, lo que define el problema de clasificación como multiclase:

1. Glioma: Tipo de tumor que se origina en las células gliales del cerebro. Suele ser el tipo más agresivo y común entre los tumores cerebrales malignos.
2. Meningioma: Tumor que surge de las meninges (las membranas que envuelven el cerebro y la medula espinal). Generalmente es benigno, pero su localización puede causar complicaciones.
3. Pituitary: Tumor en la glándula pituitaria. Suelen ser benignos (adenomas) pero pueden afectar significativamente el equilibrio hormonal del cuerpo.
4. No tumor: Imágenes que muestran un cerebro sano sin presencia de tumores.

#### 3.2 Procesamiento de imágenes MRI

La transformación de las imágenes se llevó a cabo mediante el uso de `ImageDataGenerator` de Keras. Se aplicó un escalado de los píxeles al rango [0, 1] y se utilizaron diversas técnicas de aumento de datos (rotación, desplazamiento, cizallamiento, *zoom* y volteo horizontal) en el conjunto de entrenamiento para mejorar la robustez del modelo (véase anexo experimentos notebook – *Data Preparation*)

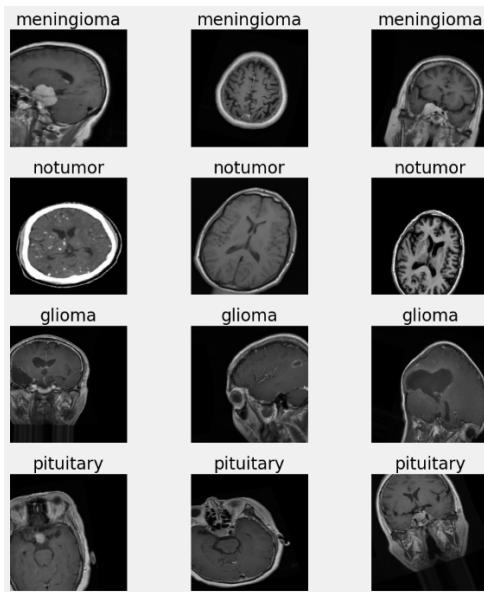


Ilustración 1 Clases del conjunto de datos Brain Tumor MRI Dataset.

El conjunto de datos se dividió en tres subconjuntos para entrenamiento, validación y pruebas. En la siguiente ilustración se muestra la distribución de clases en cada subconjunto utilizado durante la experimentación mostrando una distribución balanceada entre las clases. (véase anexo experimentos notebook– Data balance)

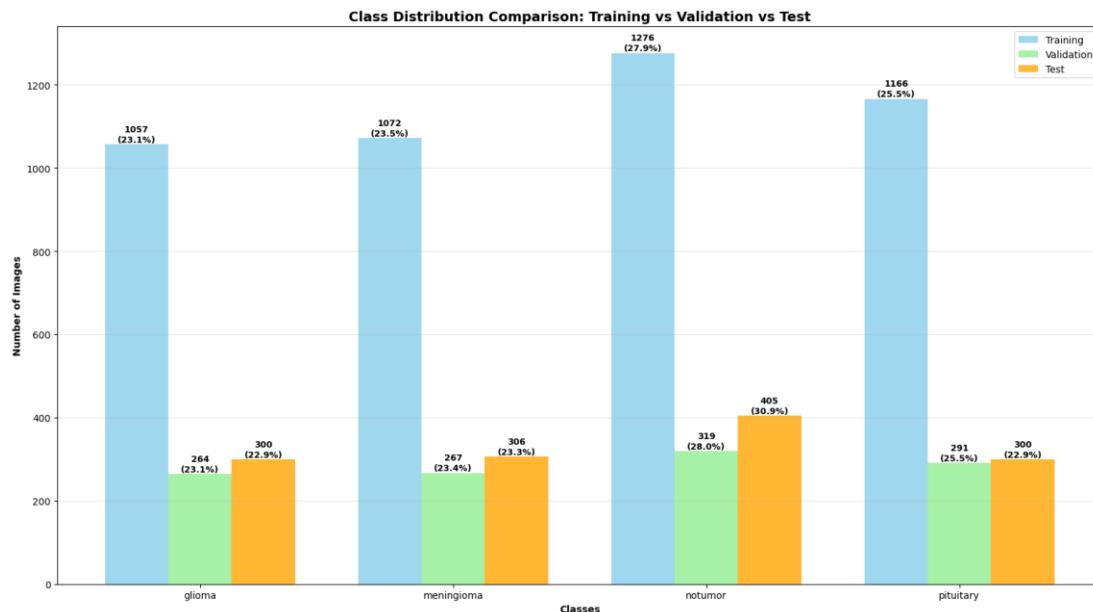


Ilustración 2 Distribución de datos entre los subconjuntos entrenamiento, validación y pruebas.

### 3.3 Experimentación redes neuronales convolucionales CNN

#### Modelo 1 – Transfer learning EfficientNetV2B0

EfficientNetV2B0 es una opción adecuada para *transfer learning* en tareas de clasificación, ya que combina la eficiencia computacional con un alto poder de representación. Preentrenado en el conjunto de datos a gran escala ImageNet, proporciona un amplio conjunto de representaciones de características que pueden adaptarse eficazmente a los ámbitos de las imágenes médicas, en los que los datos anotados suelen ser limitados (véase anexo experimentos notebook - Transfer Learning EfficientNetV2B0).

En la siguiente ilustración se muestra las capas que componen el modelo utilizado para el primer experimento.

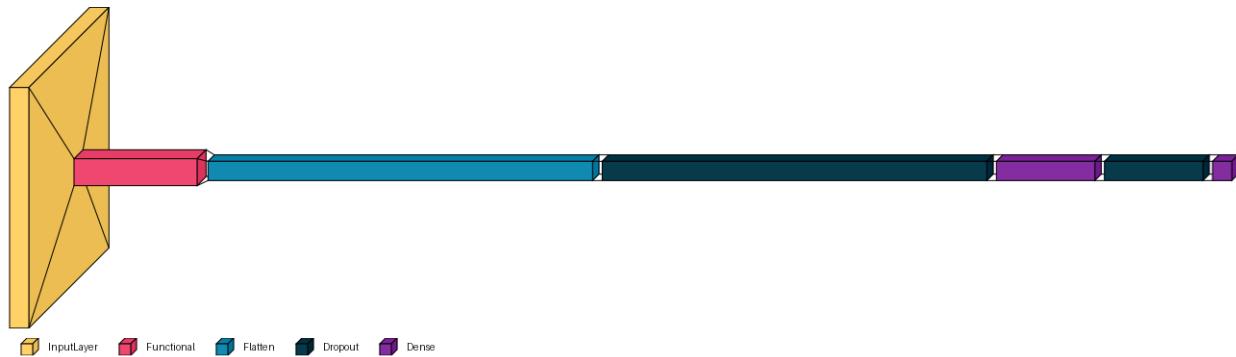


Ilustración 3 Capas del modelo 1 transfer learning EfficientNetV2B0.

Se entrenó el modelo durante 16 épocas debido a la aplicación de *EarlyStopping* y evitar el sobreajuste del modelo.

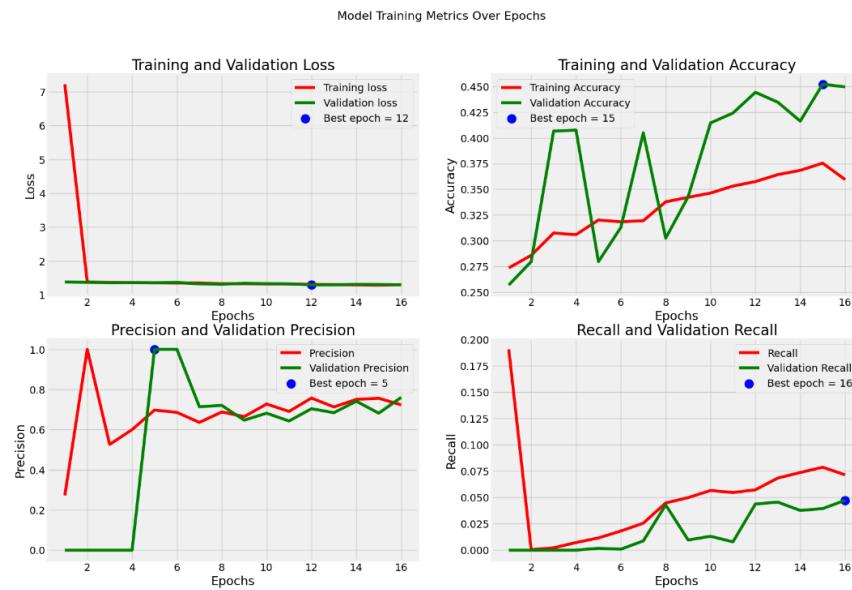


Ilustración 4 Modelo 1 EfficientNetV2B0 métricas durante entrenamiento.

Se evalúa el modelo obteniendo un *accuracy* del 43.43% con el subconjunto de entrenamiento y un 39.89% con el subconjunto de pruebas. Estas métricas indican que el modelo tiene un rendimiento bajo, tanto en entrenamiento como en la evaluación, y que hay un leve sobreajuste ya que la pérdida de evaluación es ligeramente mayor que la de entrenamiento.

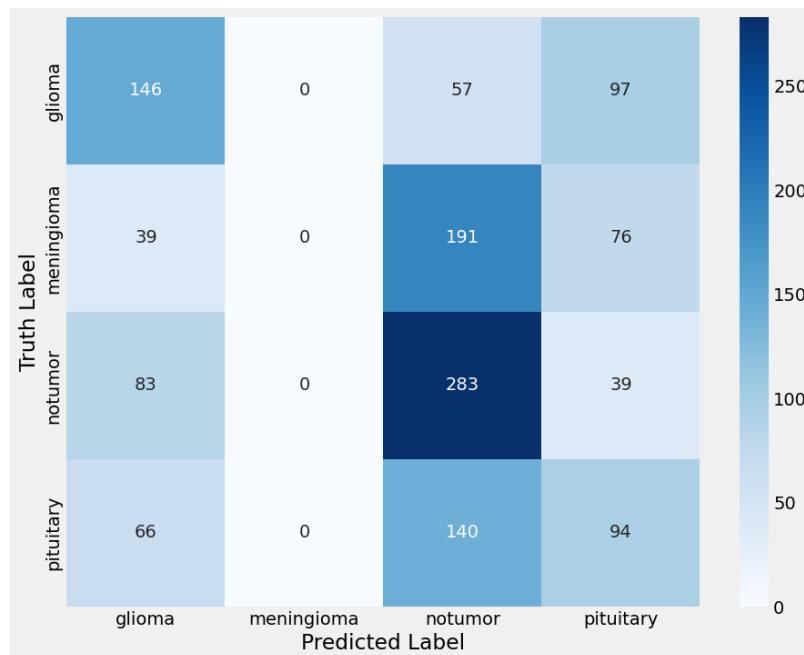


Ilustración 5 Modelo 1 EfficientNetV2B0 matriz de confusión.

- Clase “meningioma” siempre se predice como otra clase (0 correctas): Esto indica que el modelo no está aprendiendo a reconocer meningiomas y los confunde principalmente con “notumor” y “pituitary”.
- Clase “notumor” relativamente bien predicha: Con 283 predicciones correctas, esta clase es la mejor identificada, probablemente porque las imágenes sin tumor tienen características más consistentes.
- Clases “glioma” y “pituitary” parcialmente confundidas: Hay muchas predicciones incorrectas mezclando estas clases con “notumor” y entre sí. Esto sugiere que el modelo no está capturando bien los patrones específicos de cada tipo de tumor.

### **Modelo 2 – Transfer learning Xception**

La arquitectura Xception destaca por sus convoluciones separables en profundidad, que permiten un aprendizaje eficiente de las características espaciales y de canal. Preentrenada en ImageNet, Xception proporciona representaciones de características Enriquecidas que pueden reutilizarse para tareas de imagen médica, especialmente cuando los datos son limitados. (véase anexo experimentos notebook – Transfer learning Xception)

En la siguiente ilustración se muestra las capas que conforman el segundo modelo con *Xception* como modelo base para *transfer learning*.

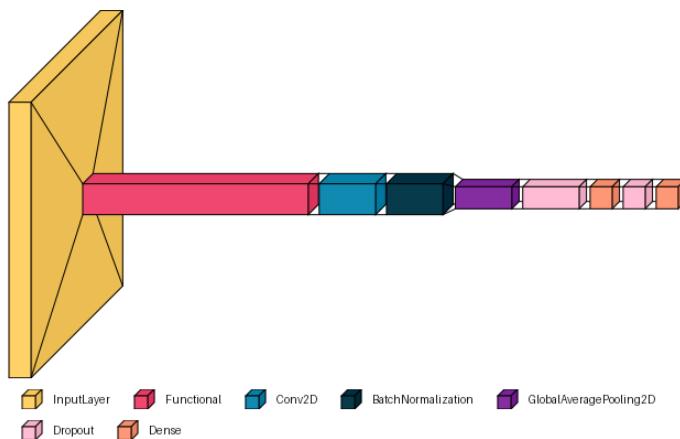


Ilustración 6 Capas del modelo 2 transfer learning Xception.

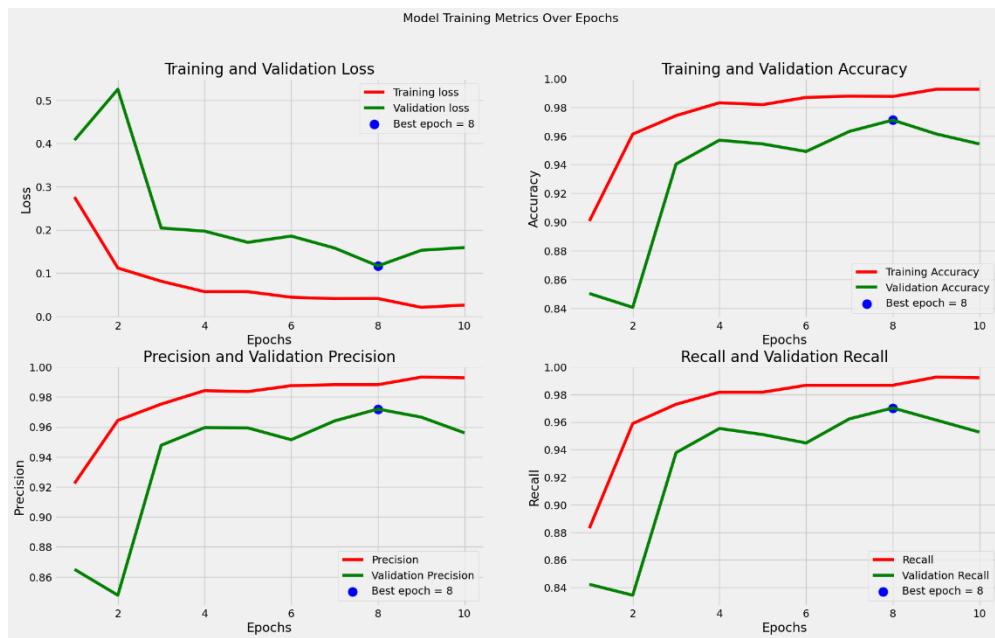


Ilustración 7 Modelo 2 Xception métricas durante entrenamiento.

Tanto la pérdida en entrenamiento como la pérdida en validación disminuyen consistentemente hasta la época 8, lo cual indica un aprendizaje de forma efectiva

Se evalúa el modelo con un *accuracy* de 99.58% con el subconjunto de entrenamiento y un 98.40% con el subconjunto de pruebas. Muestra un modelo con un alto desempeño y balanceado según los resultados de la matriz de confusión.

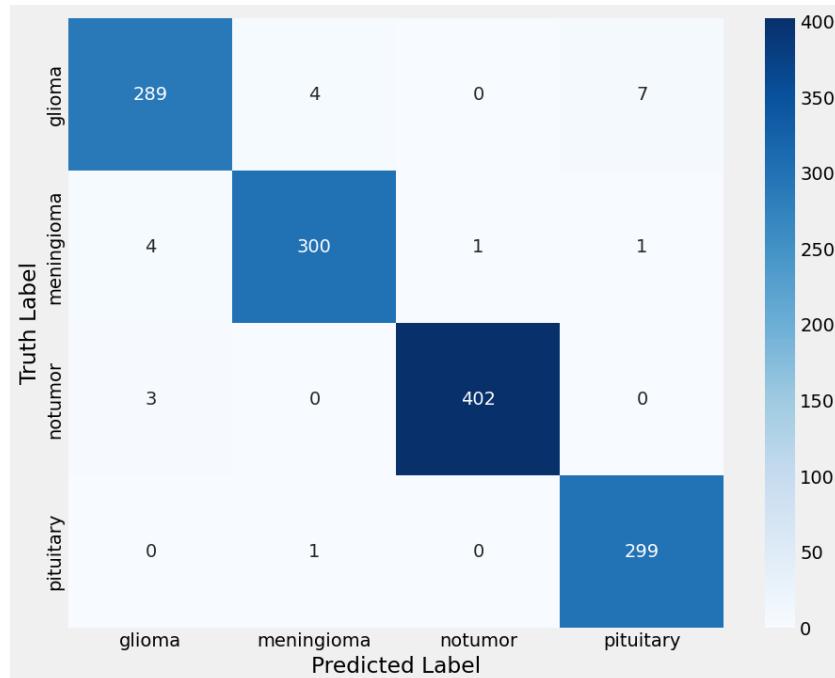


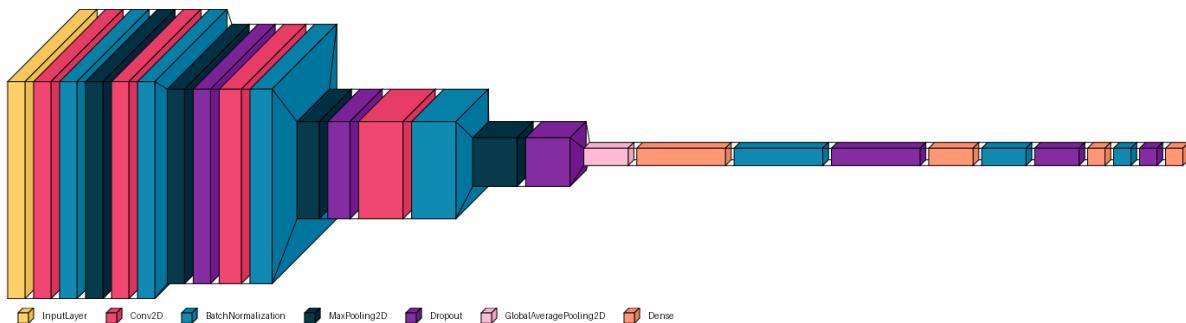
Ilustración 8 Modelo 2 Xception matriz de confusión.

- Glioma: 289 de 300 tumores fueron correctamente clasificados, 4 tumores de meningioma y 3 de no-tumor fueron incorrectamente clasificados como glioma.
- Meningioma: 300 de 306 tumores fueron correctamente clasificados, 4 tumores de glioma y 1 tumor de pituitary fueron incorrectamente clasificados como meningioma.
- Notumor: 402 de 405 imágenes fueron correctamente clasificadas como "no tumor". Solo 1 tumor de meningioma fue clasificado erróneamente como "no tumor".
- Pituitary: 299 de 300 tumores fueron correctamente clasificados, 7 tumores de glioma y 1 tumor de meningioma fueron incorrectamente clasificados como pituitary.

### **Modelo 3 – CNN desde cero**

Una CNN personalizada creada desde cero permite un control total sobre el proceso de extracción de características específicamente adaptado a las imágenes de resonancia magnética. El uso de múltiples bloques convolucionales con tamaños de filtro cada vez mayores permite el aprendizaje jerárquico de características, capturando tanto patrones de bajo nivel (bordes, texturas) como estructuras de alto nivel (formas y regiones tumorales). En la siguiente porción de código se muestra la construcción de las capas convolucionales del experimento. (véase anexo experimentos notebook – Build CNN from scratch)

En la siguiente ilustración se muestran todas las capas que componen el tercer modelo con una arquitectura construida desde cero.



*Ilustración 9 Capas del modelo 3 construido desde cero.*

Se entreno el modelo durante 5 épocas debido al *EarlyStopping* para evitar el sobre ajuste del modelo, los resultados dentro del entrenamiento se presentan en la siguiente ilustración.

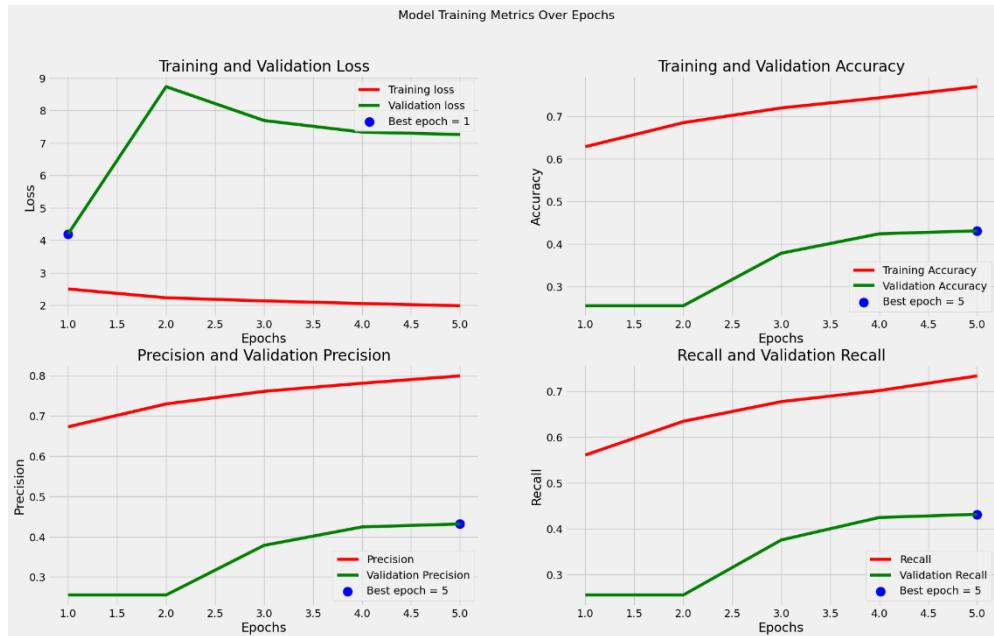


Ilustración 10 Modelo 3 CNN desde cero métricas durante entrenamiento.

La pérdida en entrenamiento disminuye gradualmente, mientras que la pérdida de validación experimenta un aumento drástico en la segunda época, lo cual indica que el modelo no está aprendiendo a generalizar debidamente.

El *accuracy* en entrenamiento como en la validación es extremadamente baja, no logrando superar el 50%. Esto indica que el modelo no es capaz de clasificar correctamente las imágenes ni en el conjunto de entrenamiento ni en el de validación. La precisión de entrenamiento es consistentemente más alta que la de validación, lo que también sugiere un sobreajuste, aunque el desempeño general es muy pobre.

El desempeño del modelo se ve claramente en la matriz de confusión. La cual revela la causa principal del bajo rendimiento del modelo.

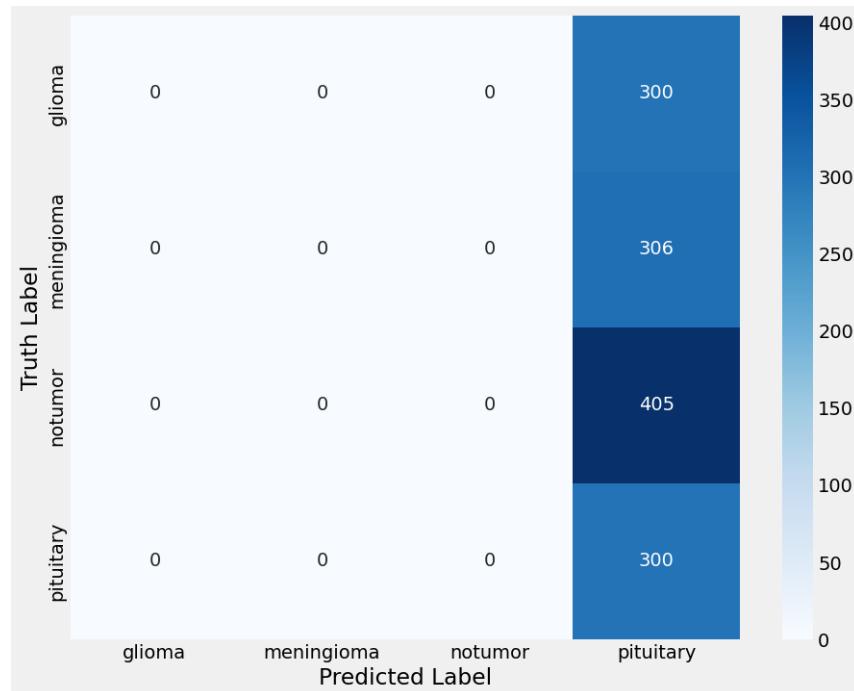


Ilustración 11 Modelo 3 CNN desde cero matriz de confusión.

La matriz de confusión muestra que el modelo no clasificó correctamente ninguna imagen de las categorías glioma, meningioma y notumor. En cambio, el modelo predijo que todas las imágenes pertenecían a la clase pituitary. Los valores de la matriz fuera de la diagonal principal son cero, excepto en la columna de la clase pituitary, donde se agrupan todas las predicciones del modelo.

### 3.4 Selección del mejor modelo

Las métricas utilizadas para evaluar el rendimiento de los modelos fueron:

- *Accuracy* (Exactitud): Proporción de predicciones correctas.
- *Precision* (Precisión): Capacidad del modelo para no predecir falsos positivos.
- *Recall* (Exhaustividad): Capacidad del modelo para encontrar todas las muestras positivas.
- *Loss* (Pérdida): Medida del error del modelo durante el entrenamiento.
- Matriz de confusión: Para visualizar el rendimiento del modelo en cada clase e identificar posibles errores de clasificación.

La selección del mejor modelo se basó en la comparación de los resultados de las métricas de evaluación (*accuracy*, *precision*, *recall*) en el subconjunto de pruebas para cada una de las arquitecturas experimentadas. Se visualizó esta comparación mediante un gráfico de barras agrupadas.

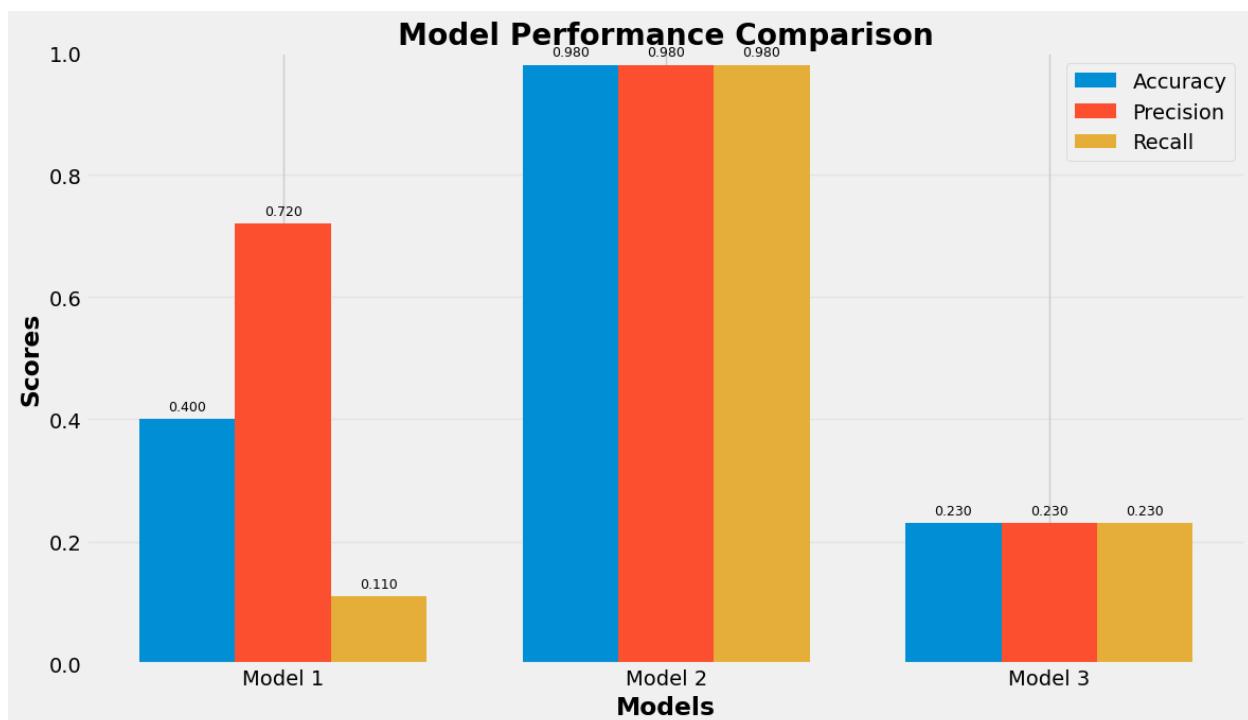


Ilustración 12 Comparación de métricas entre experimentos.

El modelo con mejor rendimiento general se consideró el mejor modelo, en este caso el modelo con la arquitectura de *transfer learning* utilizando *Xception* como modelo base es claramente el ganador, mostrando un rendimiento excepcional en todas las métricas.

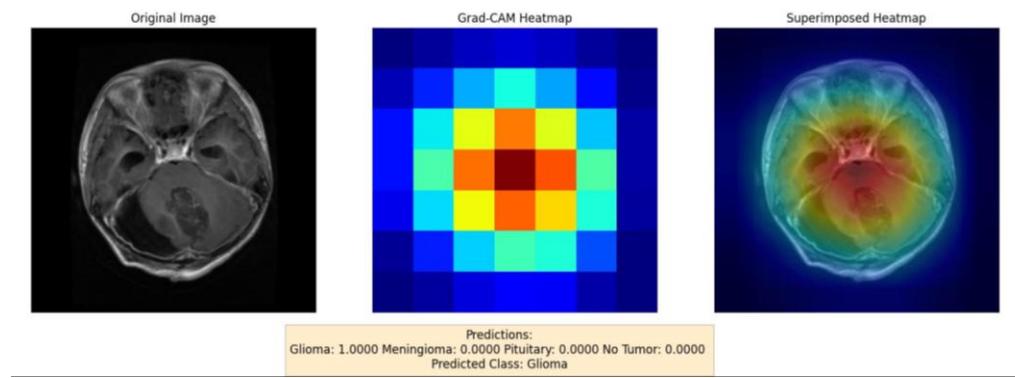
## Capítulo IV

### 4. Productivización

#### 4.1 Inferencia y Grad-CAM

Se crea la clase GradCAMVisualizer la cual contiene los métodos para la realización de inferencias con el modelo de mejor desempeño durante la etapa de experimentación, además existen los métodos para la implementación de Grad-CAM (véase anexo inferencia y grad-cam)

La inferencia del modelo muestra el resultado de aplicar Grad-CAM la cual funciona creando un mapa de calor que resalta las regiones de la imagen más importantes para la predicción de la CNN.



*Ilustración 13 Resultado de la Inferencia **Glioma** aplicado mapa de calor generado por Grad-CAM.*

1. Imagen Original: Se muestra una resonancia magnética cerebral con la presencia de un tumor en la parte central.
2. Mapa de Calor (Grad-CAM): Las áreas en colores cálidos (rojo y naranja) indican las regiones de la imagen que el modelo consideró más importantes para su predicción. El mapa de calor se centra directamente en la ubicación del tumor.
3. Mapa de Calor Superpuesto: La superposición del mapa de calor sobre la imagen original confirma que el modelo basó su decisión en el área del tumor.
4. Resultados de la Predicción: El modelo predijo que la imagen corresponde a un tumor de tipo glioma con una confianza del 100%. La confianza para las otras clases (meningioma, pituitary y no tumor) es del 0%.

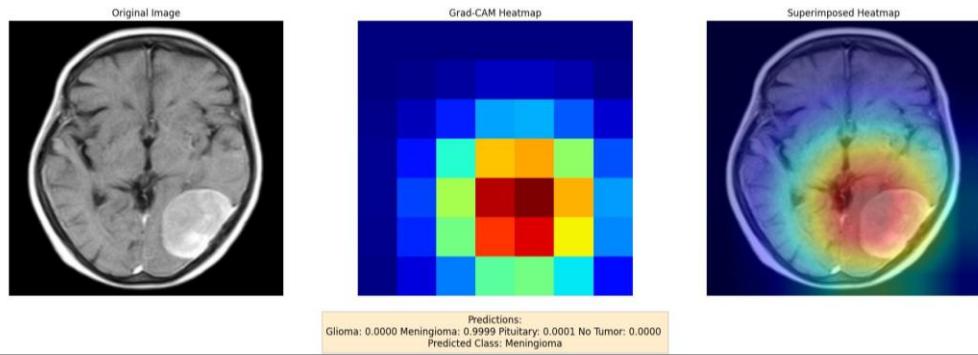


Ilustración 14 Resultado de la Inferencia **Meningioma** aplicado mapa de calor generado por Grad-CAM.

1. La imagen original es una resonancia magnética que muestra un tumor en la parte inferior derecha del cerebro.
2. El mapa de calor (Grad-CAM) resalta, con colores cálidos (rojo y naranja), las áreas de la imagen que fueron más importantes para la decisión del modelo. La región de mayor activación se superpone directamente con la ubicación del tumor.
3. La predicción del modelo es meningioma, con una confianza del 99.99%. Las otras clases (glioma, pituitary y no tumor) tienen una confianza del 0%.

## 4.2 Modelo de lenguaje de gran tamaño para interpretación de resultados

Se implementa la interpretación de los resultados a través de un modelo de lenguaje de gran tamaño por su capacidad única de traducir resultados técnicos en informes clínicamente accionables. Mientras la CNN ofrece una predicción precisa y Grad-CAM permite visualizar las zonas críticas de la imagen, estas salidas siguen siendo aun abstractas para un médico ocupado (véase anexo Explicabilidad LLM).

A través del *prompt* se contextualiza y sintetiza las instrucciones para transformar un LLM en un asistente especializado, seguro y clínicamente relevante en el flujo de trabajo de diagnóstico por imágenes (véase anexo prompt). Sus características clave son:

1. Especificidad de Rol y Contexto: Define con precisión la identidad del modelo y el escenario exacto de uso.
2. Estructura de Tareas Secuencial y Lógica: Guía al LLM a través de un flujo de trabajo clínico simulado:
  - Análisis: Comienza con la observación objetiva de los datos (imágenes y scores).
  - Ética/Seguridad: Incluye de manera obligatoria un descargo de responsabilidad, crucial para el uso médico responsable.
3. Prevención de Alucinaciones y Garantía de Veracidad: La instrucción Do not include speculative or unverified content y la restricción a fuentes de alto prestigio (NIH, NCI, WHO, FDA, PubMed) son críticas para mitigar el mayor riesgo de los LLMs en medicina: generar información plausible pero incorrecta. Esto convierte al LLM en un canal de información fiable.

### 4.3 Control de versiones

El control de versiones del código de este proyecto se gestiona mediante Git y está alojado en un repositorio público de GitHub. Esto permite un seguimiento completo del historial de cambios, facilita la colaboración entre desarrolladores y asegura la integridad del código. Se puede acceder al repositorio a través del siguiente enlace:

<https://github.com/Cristopxer/TFMBrainTumorClassifier.git>

### 4.4 Interfaz Grafica

Se construye la interfaz gráfica de usuario dividida en dos principales secciones, la entrada de los datos y el análisis de los resultados. La siguiente ilustración muestra la pantalla inicial de la interfaz en la cual se crea un nombre para el caso de estudio se habilita un control para la carga la MRI y una vista previa antes de generar el análisis. (véase anexo interfaz gráfica)

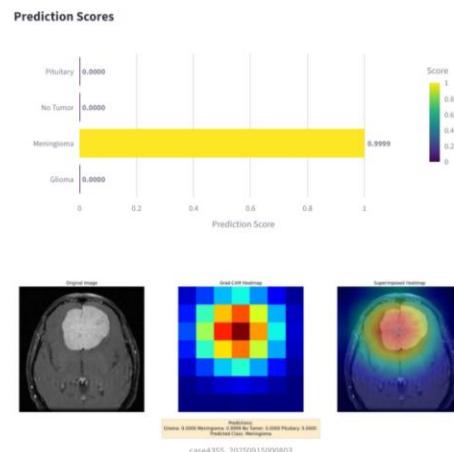
### Brain Tumor Detector Tool



*Ilustración 15 GUI - Generación de caso de análisis*

Al presionar el botón rojo “*Proceed with analysis*” se inicia con el proceso de inferencia del modelo. Luego de procesar la imagen muestra los resultados del análisis que incluye un grafico de barras con las puntuaciones de la inferencia complementado con una imagen con el MRI original, el mapa de calor generado por Grad-CAM y la superposición del mapa de calor sobre la imagen original que confirma visualmente en que región el modelo se enfoca para determinar la clase.

Predicted Class: Meningioma



*Ilustración 16 GUI - Inferencia CNN*

En la siguiente sección de la pantalla se muestran los resultados de la interpretación del *LLM (Gemini-2.5Flash)* proveyendo una advertencia de uso, el análisis de la inferencia y recursos complementarios obtenidos de fuentes médicas confiables.

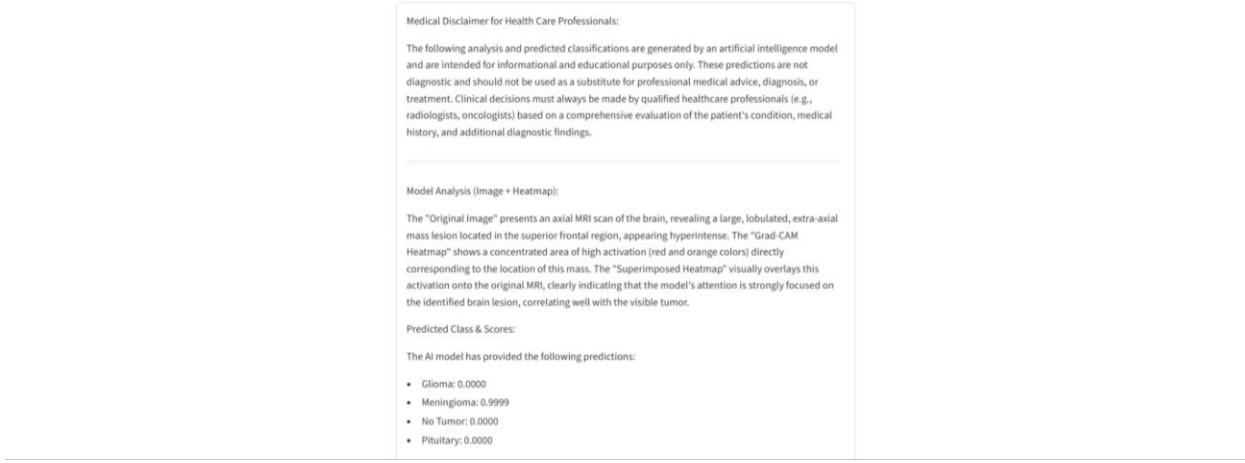


Ilustración 17 GUI - Interpretación de resultados por LLM

## Capítulo V

### 5. Conclusiones y trabajo futuro

#### 5.1 Conclusiones

La red neuronal convolucional diseñada con la técnica de *transfer learning* utilizando como *Xception* como modelo base demostró un rendimiento excepcional en la clasificación de multiclasificación de tumores cerebrales en imágenes de resonancia magnética MRI alcanzando métricas de *accuracy*, *precision* y *recall* de 98% de efectividad.

La integración de Grad-CAM es fundamental para transformar el modelo de una “caja negra” en una herramienta transparente. Los mapas de calor generados confirmaron que el modelo basa sus decisiones en regiones biomédicamente relevantes, lo que construye confianza y permite la verificación por parte de un especialista.

El módulo de interpretación basado en un modelo de lenguaje de gran tamaño cumplió efectivamente con la función de generar automáticamente reportes descriptivos, junto a información relevante de fuentes confiables (NCCN, WHO) agregando valor significativo al transformar una inferencia simple en un informe contextualizado y útil para la toma de decisiones.

#### 5.2 Trabajo a futuro

A partir de los resultados y limitaciones de este proyecto, se proponen las siguientes líneas de trabajo futuro:

1. Validación clínica: Esto implica colaborar con instituciones médicas para evaluar su desempeño contra diagnósticos histopatológicos y medir su impacto real en la precisión del diagnóstico.
2. Incorporación de datos clínicos: Mejorar el modelo integrando múltiples secuencias de MRI de forma simultánea, sino también datos clínicos del paciente para que las predicciones sean más holísticas y personalizadas.
3. Optimización del LLM para Seguridad Clínica: Investigar y desarrollar técnicas de RAG más robustas para el LLM, asegurando que toda la información médica generada esté fundamentada exclusivamente en bases de datos de literatura médica actualizada y guías clínicas.

## 6. Referencias

- [1] J. A. H. MD, «Burnout of Radiologists: Frequency, Risk Factors, and Remedies: A Report of the ACR Commission on Human Resources,» *Journal of the American College of Radiology*, pp. 411-416, 2016.
- [2] «RadiologyInfo,» 20 March 2024. [En línea]. Available: <https://www.radiologyinfo.org/en/info/mri-brain>. [Último acceso: 09 09 2025].
- [3] Y. B. & G. H. Yann LeCun, «Deep learning,» *nature*, 27 May 2015. [En línea]. Available: <https://doi.org/10.1038/nature14539>.
- [4] M. C. A. D. R. V. D. P. D. B. Ramprasaath R. Selvaraju, «Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization,» arxiv, 3 Dec 2019. [En línea]. Available: <https://arxiv.org/abs/1610.02391>. [Último acceso: 09 2025].
- [5] National Cancer Institute, «Artificial Intelligence (AI) and Cancer,» 30 May 2024. [En línea]. Available: <https://www.cancer.gov/research/infrastructure/artificial-intelligence>.

## 7. Anexos

### 7.1 Experimentos Notebook

#### Brain Tumor Classifier MRI

```
In [1]: !pip install visualkeras

Collecting visualkeras
  Downloading visualkeras-0.1.4-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.12/dist-packages (from visualkeras) (11.3.0)
Requirement already satisfied: numpy>=1.18.1 in /usr/local/lib/python3.12/dist-packages (from visualkeras) (2.0.2)
Collecting aggdraw>=1.3.11 (from visualkeras)
  Downloading aggdraw-1.3.19-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (655 bytes)
  Downloading visualkeras-0.1.4-py3-none-any.whl (17 kB)
  Downloading aggdraw-1.3.19-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.0 MB)
                                                               1.0/1.0 MB 25.7 MB/s eta 0:00:00
Installing collected packages: aggdraw, visualkeras
Successfully installed aggdraw-1.3.19 visualkeras-0.1.4
```

### Libraries and setup

```
In [18]: import kagglehub
import tensorflow as tf
from tensorflow.keras import mixed_precision
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

from tensorflow.keras.applications import EfficientNetV2B0, Xception
from tensorflow.keras.layers import (Input, Dense, GlobalAveragePooling2D, Dropout,
                                      Flatten, Conv2D, MaxPooling2D, BatchNormalization)
from tensorflow.keras.models import Model
from tensorflow.keras.metrics import Precision, Recall

from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import numpy as np

from visualkeras import layered_view
```

```
In [3]: # Check if GPU is available
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
print("GPU Devices: ", tf.config.list_physical_devices('GPU'))
```

```
Num GPUs Available: 1
GPU Devices: [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

### Download dataset

```
In [4]: # Download Latest version
path = kagglehub.dataset_download("masoudnickparvar/brain-tumor-mri-dataset")
```

Using Colab cache for faster access to the 'brain-tumor-mri-dataset' dataset.

```
In [5]: print("Path to dataset files:", path)
```

Path to dataset files: /kaggle/input/brain-tumor-mri-dataset

## Data Preparation

The Brain Tumor MRI Dataset contains a collection of MRI images designed for the development and evaluation of machine learning models in medical imaging. The dataset includes T1-weighted contrast-enhanced MRI scans categorized into four classes:

Glioma tumor

Meningioma tumor

Pituitary tumor

No tumor

```
In [6]: # Data directories
train_dir = '/kaggle/input/brain-tumor-mri-dataset/Training'
test_dir = '/kaggle/input/brain-tumor-mri-dataset/Testing'

# Image parameters
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
```

```
In [7]: # Data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2
)

test_datagen = ImageDataGenerator(rescale=1./255)
```

```
In [8]: # Create data generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training'
```

```
)
validation_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False
)
```

Found 4571 images belonging to 4 classes.  
 Found 1141 images belonging to 4 classes.  
 Found 1311 images belonging to 4 classes.

In [53]: # Function to show images

```
def show_batch(image_batch, label_batch, class_names, num_examples=3):
    """
    Display a batch of images with their corresponding labels.

    Args:
        image_batch (numpy.ndarray): Batch of images.
        label_batch (numpy.ndarray): Batch of labels.
        class_names (list): List of class names.
        num_examples (int): Number of examples to display per class.

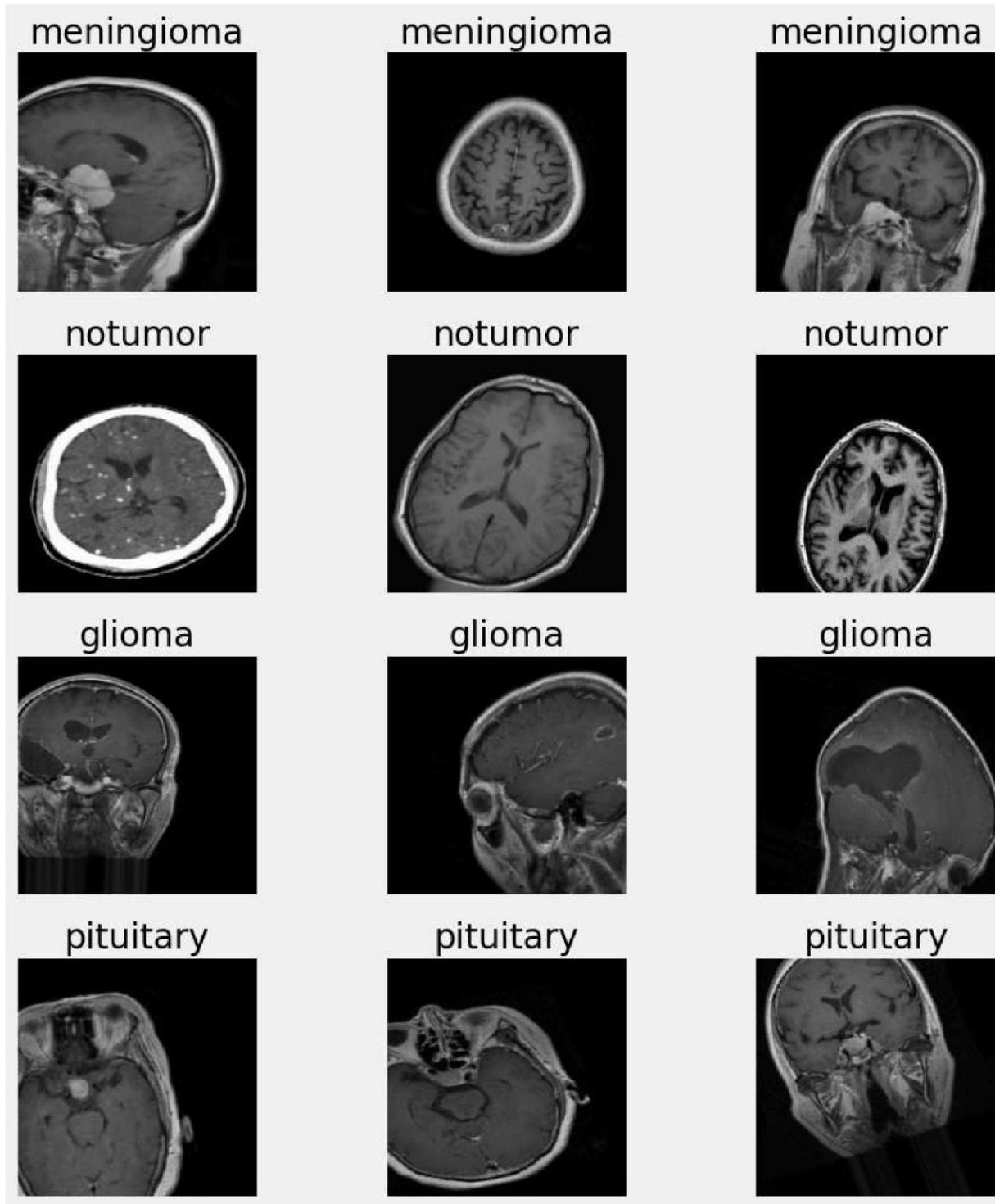
    """
    plt.figure(figsize=(10,10))
    # Find indices for each class
    class_indices = {}
    for i, label in enumerate(label_batch):
        class_idx = np.argmax(label)
        class_name = class_names[class_idx]
        if class_name not in class_indices:
            class_indices[class_name] = []
        if len(class_indices[class_name]) < num_examples:
            class_indices[class_name].append(i)

    # Display images
    plot_idx = 1
    for class_name, indices in class_indices.items():
        for idx in indices:
            plt.subplot(len(class_names), num_examples, plot_idx)
            plt.imshow(image_batch[idx])
            plt.title(class_name)
            plt.axis("off")
            plot_idx += 1
    plt.tight_layout()
    plt.show()
```

```
# Get a batch of images and Labels from the training generator
image_batch, label_batch = next(train_generator)

# Get class names from the generator
class_names = list(train_generator.class_indices.keys())

# Show examples
show_batch(image_batch, label_batch, class_names, num_examples=3)
```



Each image is stored in JPEG format and has been preprocessed for consistency. The dataset provides a balanced number of samples across classes, enabling both classification research

## Data balance

```
In [9]: # Check for class balance
# Get class information
num_classes = len(train_generator.class_indices)
class_names = list(train_generator.class_indices.keys())
train_class_counts = np.bincount(train_generator.classes)
val_class_counts = np.bincount(validation_generator.classes)
test_class_counts = np.bincount(test_generator.classes)

# More detailed analysis with percentages
plt.figure(figsize=(16, 9))

x = np.arange(len(class_names))
width = 0.25

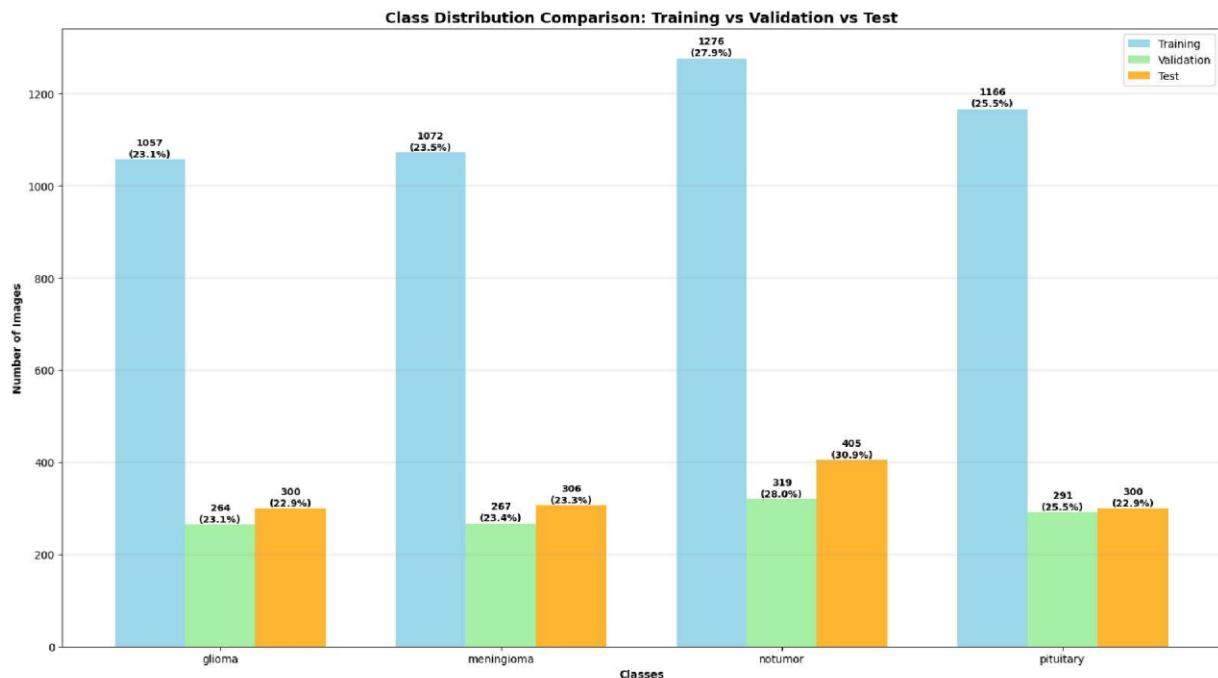
# Calculate percentages
train_percentages = (train_class_counts / train_class_counts.sum()) * 100
val_percentages = (val_class_counts / val_class_counts.sum()) * 100
test_percentages = (test_class_counts / test_class_counts.sum()) * 100

# Create bars
bars1 = plt.bar(x - width, train_class_counts, width, label='Training', color='skyblue')
bars2 = plt.bar(x, val_class_counts, width, label='Validation', color='lightgreen')
bars3 = plt.bar(x + width, test_class_counts, width, label='Test', color='orange')

# Add labels and title
plt.xlabel('Classes', fontweight='bold')
plt.ylabel('Number of Images', fontweight='bold')
plt.title('Class Distribution Comparison: Training vs Validation vs Test', fontweight='bold')
plt.xticks(x, class_names)
plt.legend()

# Add value labels with percentages
for i, (bar1, bar2, bar3) in enumerate(zip(bars1, bars2, bars3)):
    # Training Labels
    plt.text(bar1.get_x() + bar1.get_width()/2, bar1.get_height() + 0.5,
             f'{train_class_counts[i]}\n({train_percentages[i]:.1f}%)',
             ha='center', va='bottom', fontsize=9, fontweight='bold')
    # Validation Labels
    plt.text(bar2.get_x() + bar2.get_width()/2, bar2.get_height() + 0.5,
             f'{val_class_counts[i]}\n({val_percentages[i]:.1f}%)',
             ha='center', va='bottom', fontsize=9, fontweight='bold')
    # Test Labels
    plt.text(bar3.get_x() + bar3.get_width()/2, bar3.get_height() + 0.5,
             f'{test_class_counts[i]}\n({test_percentages[i]:.1f}%)',
             ha='center', va='bottom', fontsize=9, fontweight='bold')

plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()
```



## Transfer Learning (EfficientNetV2B0)

EfficientNetV2B0 is a suitable choice for transfer learning in brain tumor classification tasks because it combines computational efficiency with high representational power. Pretrained on the large-scale ImageNet dataset, it provides a rich set of feature representations that can be effectively adapted to medical imaging domains where annotated data is often limited.

```
In [10]: ## Model definition
def create_model1(num_classes):
    """
    Build a transfer learning model using EfficientNetV2B0 as the feature extractor

    This function constructs a deep learning model for image classification by leveraging
    EfficientNetV2B0 pretrained on ImageNet as the base model. The base is frozen to act as a
    fixed feature extractor, and a custom classification head is added on top.

    Architecture:
    - Input layer matching the global `IMG_SIZE` (height, width, 3).
    - EfficientNetV2B0 backbone (pretrained, frozen).
    - Flatten layer to convert feature maps into a 1D vector.
    - Dense (1024 units, ReLU) with Dropout(0.25) for regularization.
    - Dense output layer with softmax activation for multi-class classification.

    Args:
        num_classes (int): Number of target classes for classification.

    Returns:
        tensorflow.keras.Model: Compiled Keras model ready for training or fine-tuning.

    Notes:
        - The base model layers are frozen by default (not trainable).
        To enable fine-tuning, set `base_model.trainable = True` after creating the model.
    """

    base_model = EfficientNetV2B0(weights='imagenet', include_top=False, input_shape=(IMG_SIZE, IMG_SIZE, 3))
    base_model.trainable = False

    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.25)(x)
    x = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=x)

    return model
```

```
"""
# Load base model with pretrained weights
base_model = EfficientNetV2B0(
    include_top=False,
    weights='imagenet',
    input_shape=(*IMG_SIZE, 3))

# Freeze base model Layers
base_model.trainable = False

# Create new model on top
inputs = tf.keras.Input(shape=(*IMG_SIZE, 3))
x = base_model(inputs, training=False)
x = Flatten()(x)
x = Dropout(0.25)(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.25)(x)

outputs = Dense(num_classes, activation='softmax', dtype='float32')(x)

model = Model(inputs, outputs)

return model
```

In [11]:

```
# Create model
model1 = create_model1(num_classes)

for layer in model1.layers:
    print(f"{layer.name}: {layer.dtype_policy}")
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/efficientnet\_v2/efficientnetv2-b0\_notop.h5  
24274472/24274472 \_\_\_\_\_ 0s 0us/step  
input\_layer\_1: <DTypePolicy "float32">  
efficientnetv2-b0: <DTypePolicy "float32">  
flatten: <DTypePolicy "float32">  
dropout: <DTypePolicy "float32">  
dense: <DTypePolicy "float32">  
dropout\_1: <DTypePolicy "float32">  
dense\_1: <DTypePolicy "float32">

In [12]:

```
model1.summary()
```

**Model: "functional"**

Layer (type)	Output Shape	Param #
input_layer_1 ( <a href="#">InputLayer</a> )	( <a href="#">None</a> , 224, 224, 3)	0
efficientnetv2-b0 ( <a href="#">Functional</a> )	( <a href="#">None</a> , 7, 7, 1280)	5,919,312
flatten ( <a href="#">Flatten</a> )	( <a href="#">None</a> , 62720)	0
dropout ( <a href="#">Dropout</a> )	( <a href="#">None</a> , 62720)	0
dense ( <a href="#">Dense</a> )	( <a href="#">None</a> , 1024)	64,226,304
dropout_1 ( <a href="#">Dropout</a> )	( <a href="#">None</a> , 1024)	0
dense_1 ( <a href="#">Dense</a> )	( <a href="#">None</a> , 4)	4,100

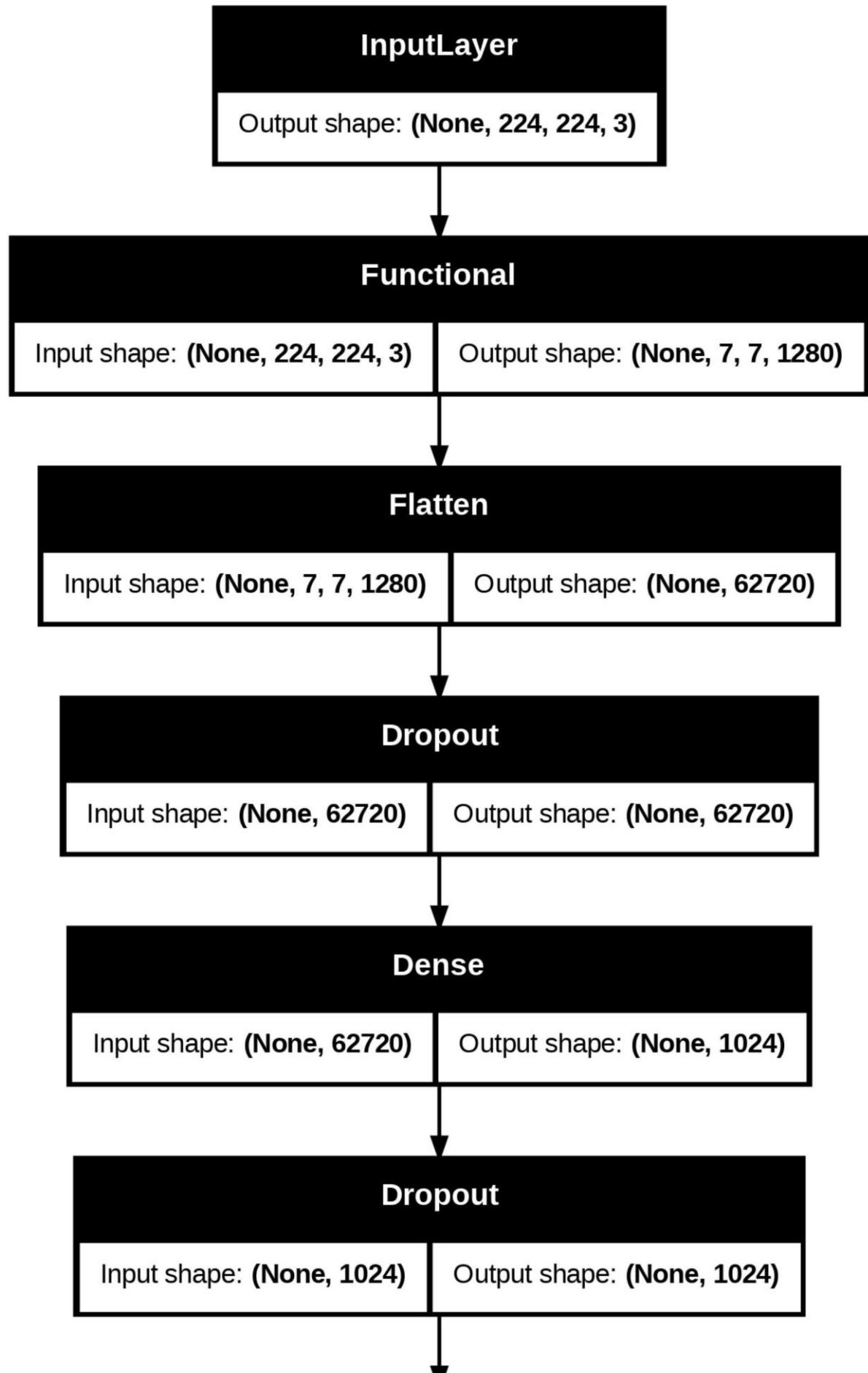
Total params: 70,149,716 (267.60 MB)

Trainable params: 64,230,404 (245.02 MB)

Non-trainable params: 5,919,312 (22.58 MB)

In [13]: `tf.keras.utils.plot_model(model1, show_shapes=True)`

Out[13]:



## Dense

Input shape: <b>(None, 1024)</b>	Output shape: <b>(None, 4)</b>
----------------------------------	--------------------------------

## Model compilation

```
In [14]: # Custom Learning rate
initial_learning_rate = 0.001

# Compile the model with accuracy, precision and recall as evaluation metrics
model1.compile(
    optimizer=tf.keras.optimizers.Adamax(learning_rate=initial_learning_rate),
    loss='categorical_crossentropy',
    metrics=['accuracy', Precision(name='precision'), Recall(name='recall')])
)
```

## Model Training

```
In [15]: # Callbacks
"""
List of Keras callbacks used to optimize and monitor the training process:

1. ModelCheckpoint:
    - Saves the model to "best_model1.keras".
    - Monitors training accuracy (`monitor="accuracy"`).
    - Keeps only the best model according to maximum accuracy (`mode="max"`).

2. EarlyStopping:
    - Stops training early if no improvement is seen.
    - Uses `patience=4` (waits 4 epochs before stopping).
    - Restores the weights of the best-performing epoch (`restore_best_weights=True`).

3. ReduceLROnPlateau:
    - Reduces the learning rate when a metric has stopped improving.
    - Reduces by a factor of 0.1 after `patience=3` epochs without improvement.
"""

callbacks = [
    tf.keras.callbacks.ModelCheckpoint("best_model1.keras", monitor="accuracy", save_best_only=True),
    tf.keras.callbacks.EarlyStopping(patience=4, restore_best_weights=True),
    tf.keras.callbacks.ReduceLROnPlateau(factor=0.1, patience=3)
]

# Training
history1 = model1.fit(
    train_generator,
    epochs=30,
    validation_data=validation_generator,
    callbacks=callbacks
)
```

Epoch 1/30

**143/143** 174s 998ms/step - accuracy: 0.2575 - loss: 16.5933 - precision: 0.2515 - recall: 0.2255 - val\_accuracy: 0.2568 - val\_loss: 1.3826 - val\_precision: 0.0000e+00 - val\_recall: 0.0000e+00 - learning\_rate: 0.0010

Epoch 2/30

**143/143** 85s 594ms/step - accuracy: 0.2864 - loss: 1.3857 - precision: 0.9514 - recall: 8.6621e-04 - val\_accuracy: 0.2796 - val\_loss: 1.3696 - val\_precision: 0.0000e+00 - val\_recall: 0.0000e+00 - learning\_rate: 0.0010

Epoch 3/30

**143/143** 82s 573ms/step - accuracy: 0.2969 - loss: 1.3679 - precision: 0.1624 - recall: 5.4035e-04 - val\_accuracy: 0.4067 - val\_loss: 1.3552 - val\_precision: 0.0000e+00 - val\_recall: 0.0000e+00 - learning\_rate: 0.0010

Epoch 4/30

**143/143** 72s 505ms/step - accuracy: 0.3090 - loss: 1.3554 - precision: 0.6069 - recall: 0.0094 - val\_accuracy: 0.4075 - val\_loss: 1.3567 - val\_precision: 0.0000e+00 - val\_recall: 0.0000e+00 - learning\_rate: 0.0010

Epoch 5/30

**143/143** 86s 605ms/step - accuracy: 0.3257 - loss: 1.3525 - precision: 0.5524 - recall: 0.0087 - val\_accuracy: 0.2796 - val\_loss: 1.3545 - val\_precision: 1.0000 - val\_recall: 0.0018 - learning\_rate: 0.0010

Epoch 6/30

**143/143** 72s 505ms/step - accuracy: 0.3094 - loss: 1.3560 - precision: 0.5993 - recall: 0.0144 - val\_accuracy: 0.3129 - val\_loss: 1.3663 - val\_precision: 1.0000 - val\_recall: 8.7642e-04 - learning\_rate: 0.0010

Epoch 7/30

**143/143** 80s 560ms/step - accuracy: 0.3295 - loss: 1.3476 - precision: 0.6812 - recall: 0.0278 - val\_accuracy: 0.4049 - val\_loss: 1.3251 - val\_precision: 0.7143 - val\_recall: 0.0088 - learning\_rate: 0.0010

Epoch 8/30

**143/143** 89s 610ms/step - accuracy: 0.3264 - loss: 1.3307 - precision: 0.7356 - recall: 0.0447 - val\_accuracy: 0.3024 - val\_loss: 1.3100 - val\_precision: 0.7206 - val\_recall: 0.0429 - learning\_rate: 0.0010

Epoch 9/30

**143/143** 124s 873ms/step - accuracy: 0.3395 - loss: 1.3276 - precision: 0.7012 - recall: 0.0520 - val\_accuracy: 0.3427 - val\_loss: 1.3439 - val\_precision: 0.6471 - val\_recall: 0.0096 - learning\_rate: 0.0010

Epoch 10/30

**143/143** 100s 703ms/step - accuracy: 0.3493 - loss: 1.3205 - precision: 0.7183 - recall: 0.0566 - val\_accuracy: 0.4145 - val\_loss: 1.3311 - val\_precision: 0.6818 - val\_recall: 0.0131 - learning\_rate: 0.0010

Epoch 11/30

**143/143** 121s 848ms/step - accuracy: 0.3558 - loss: 1.3181 - precision: 0.7010 - recall: 0.0581 - val\_accuracy: 0.4242 - val\_loss: 1.3227 - val\_precision: 0.6429 - val\_recall: 0.0079 - learning\_rate: 0.0010

Epoch 12/30

**143/143** 91s 637ms/step - accuracy: 0.3505 - loss: 1.3213 - precision: 0.7537 - recall: 0.0403 - val\_accuracy: 0.4443 - val\_loss: 1.2965 - val\_precision: 0.7042 - val\_recall: 0.0438 - learning\_rate: 1.0000e-04

Epoch 13/30

**143/143** 101s 711ms/step - accuracy: 0.3614 - loss: 1.3081 - precision: 0.7357 - recall: 0.0697 - val\_accuracy: 0.4347 - val\_loss: 1.3004 - val\_precision: 0.6842 - val\_recall: 0.0456 - learning\_rate: 1.0000e-04

Epoch 14/30

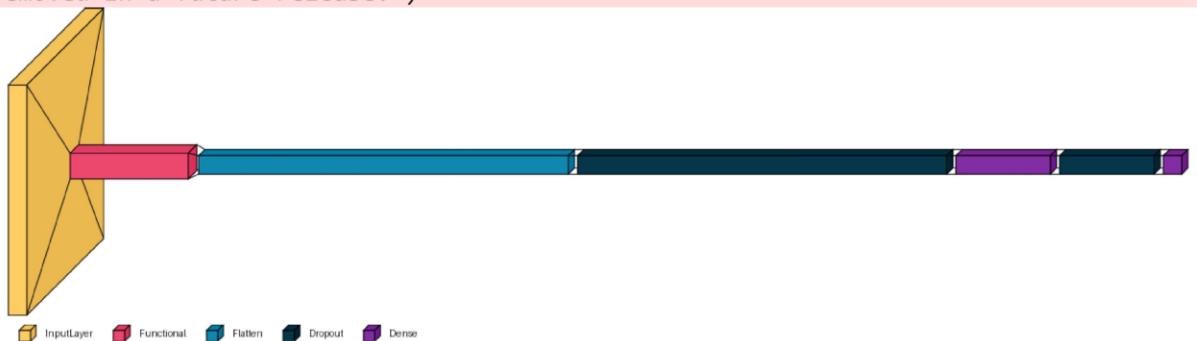
**143/143** 94s 660ms/step - accuracy: 0.3570 - loss: 1.3090 - precision: 0.7153 - recall: 0.0660 - val\_accuracy: 0.4163 - val\_loss: 1.3096 - val\_precision: 0.7414 - val\_recall: 0.0377 - learning\_rate: 1.0000e-04

```
Epoch 15/30
143/143 ————— 89s 622ms/step - accuracy: 0.3702 - loss: 1.3022 - precision: 0.7372 - recall: 0.0735 - val_accuracy: 0.4522 - val_loss: 1.3067 - val_precision: 0.6818 - val_recall: 0.0394 - learning_rate: 1.0000e-04
Epoch 16/30
143/143 ————— 73s 511ms/step - accuracy: 0.3499 - loss: 1.3067 - precision: 0.7006 - recall: 0.0678 - val_accuracy: 0.4496 - val_loss: 1.2997 - val_precision: 0.7606 - val_recall: 0.0473 - learning_rate: 1.0000e-05
```

```
In [19]: # Visualize Layer model
layered_view(model1, legend=True, max_xy=250)
```

```
/usr/local/lib/python3.12/dist-packages/visualkeras/layered.py:86: UserWarning: The legend_text_spacing_offset parameter is deprecated and will be removed in a future release.
warnings.warn("The legend_text_spacing_offset parameter is deprecated and will be removed in a future release.")
```

Out[19]:



## Model Evaluation

```
In [20]: def model_evaluation(model, train_gen, test_gen):
    """
    Evaluate a trained Keras model on training and testing datasets.

    This function computes evaluation metrics (loss, accuracy, precision, recall)
    for both training and testing sets, prints the results in a readable format,
    and returns the key metrics for the test set.

    Args:
        model (tf.keras.Model): Trained Keras model to be evaluated.
        train_gen (tf.keras.utils.Sequence or tf.data.Dataset):
            Data generator or dataset for training data.
        test_gen (tf.keras.utils.Sequence or tf.data.Dataset):
            Data generator or dataset for testing data.

    Returns:
        dict: A dictionary containing rounded evaluation metrics for the test set:
            - 'accuracy' (float): Model accuracy on the test set.
            - 'precision' (float): Model precision on the test set.
            - 'recall' (float): Model recall on the test set.

    Notes:
        - Assumes that `model.compile()` was called with metrics including
          accuracy, precision, and recall (in that order).
        - Prints both training and testing loss/accuracy to the console.
    """
    # Implementation of the function follows here, using the provided arguments and
    # returning the specified dictionary of metrics.
```

```
"""
train_score = model.evaluate(train_gen, verbose=1)
test_score = model.evaluate(test_gen, verbose=1)

print(f"Train Loss: {train_score[0]:.4f}")
print(f"Train Accuracy: {train_score[1]*100:.2f}%")
print('-' * 20)
print(f"Test Loss: {test_score[0]:.4f}")
print(f"Test Accuracy: {test_score[1]*100:.2f}%")
return {'accuracy':round(test_score[1],2),'precision':round(test_score[2],2),'rec
```

In [21]: `test1_metrics = model_evaluation(model1, train_generator, test_generator)`

```
143/143 ━━━━━━━━━━ 61s 428ms/step - accuracy: 0.4439 - loss: 1.2587 - precision: 0.9677 - recall: 0.0710
41/41 ━━━━━━━━━━ 18s 453ms/step - accuracy: 0.4108 - loss: 1.3279 - precision: 0.3407 - recall: 0.0580
Train Loss: 1.2619
Train Accuracy: 43.43%
-----
Test Loss: 1.2786
Test Accuracy: 39.89%
```

In [22]: `test1_metrics`

Out[22]: `{'accuracy': 0.4, 'precision': 0.72, 'recall': 0.11}`

In [23]: `def plot_history(hist):`  
`"""`  
`Plot training and validation metrics from a Keras History object.`  
`This function visualizes the progression of loss, accuracy, precision, and recall across training epochs for both training and validation sets. It also highlights the best epoch (based on validation performance) for each metric.`

Args:

```
hist (tf.keras.callbacks.History):
    History object returned by `model.fit()`, containing training and validation metrics across epochs.
```

Plots:

- Training vs Validation Loss (with best epoch marked).
- Training vs Validation Accuracy (with best epoch marked).
- Training vs Validation Precision (with best epoch marked).
- Training vs Validation Recall (with best epoch marked).

"""

```
# Get evaluation metrics during training
tr_acc = hist.history['accuracy']
tr_loss = hist.history['loss']
tr_per = hist.history['precision']
tr_recall = hist.history['recall']
val_acc = hist.history['val_accuracy']
val_loss = hist.history['val_loss']
val_per = hist.history['val_precision']
val_recall = hist.history['val_recall']
```

```

# Get best results
index_loss = np.argmin(val_loss)
val_lowest = val_loss[index_loss]
index_acc = np.argmax(val_acc)
acc_highest = val_acc[index_acc]
index_precision = np.argmax(val_per)
per_highest = val_per[index_precision]
index_recall = np.argmax(val_recall)
recall_highest = val_recall[index_recall]

# Get the epoch with the best score
Epochs = [i + 1 for i in range(len(tr_acc))]
loss_label = f'Best epoch = {str(index_loss + 1)}'
acc_label = f'Best epoch = {str(index_acc + 1)}'
per_label = f'Best epoch = {str(index_precision + 1)}'
recall_label = f'Best epoch = {str(index_recall + 1)}'

# Set main canvas
plt.figure(figsize=(20, 12))
plt.style.use('fivethirtyeight')

# Plot Loss
plt.subplot(2, 2, 1)
plt.plot(Epochs, tr_loss, 'r', label='Training loss')
plt.plot(Epochs, val_loss, 'g', label='Validation loss')
plt.scatter(index_loss + 1, val_lowest, s=150, c='blue', label=loss_label)
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

# Plot Accuracy
plt.subplot(2, 2, 2)
plt.plot(Epochs, tr_acc, 'r', label='Training Accuracy')
plt.plot(Epochs, val_acc, 'g', label='Validation Accuracy')
plt.scatter(index_acc + 1, acc_highest, s=150, c='blue', label=acc_label)
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

# Plot Precision
plt.subplot(2, 2, 3)
plt.plot(Epochs, tr_per, 'r', label='Precision')
plt.plot(Epochs, val_per, 'g', label='Validation Precision')
plt.scatter(index_precision + 1, per_highest, s=150, c='blue', label=per_label)
plt.title('Precision and Validation Precision')
plt.xlabel('Epochs')
plt.ylabel('Precision')
plt.legend()
plt.grid(True)

# Plot Recall

```

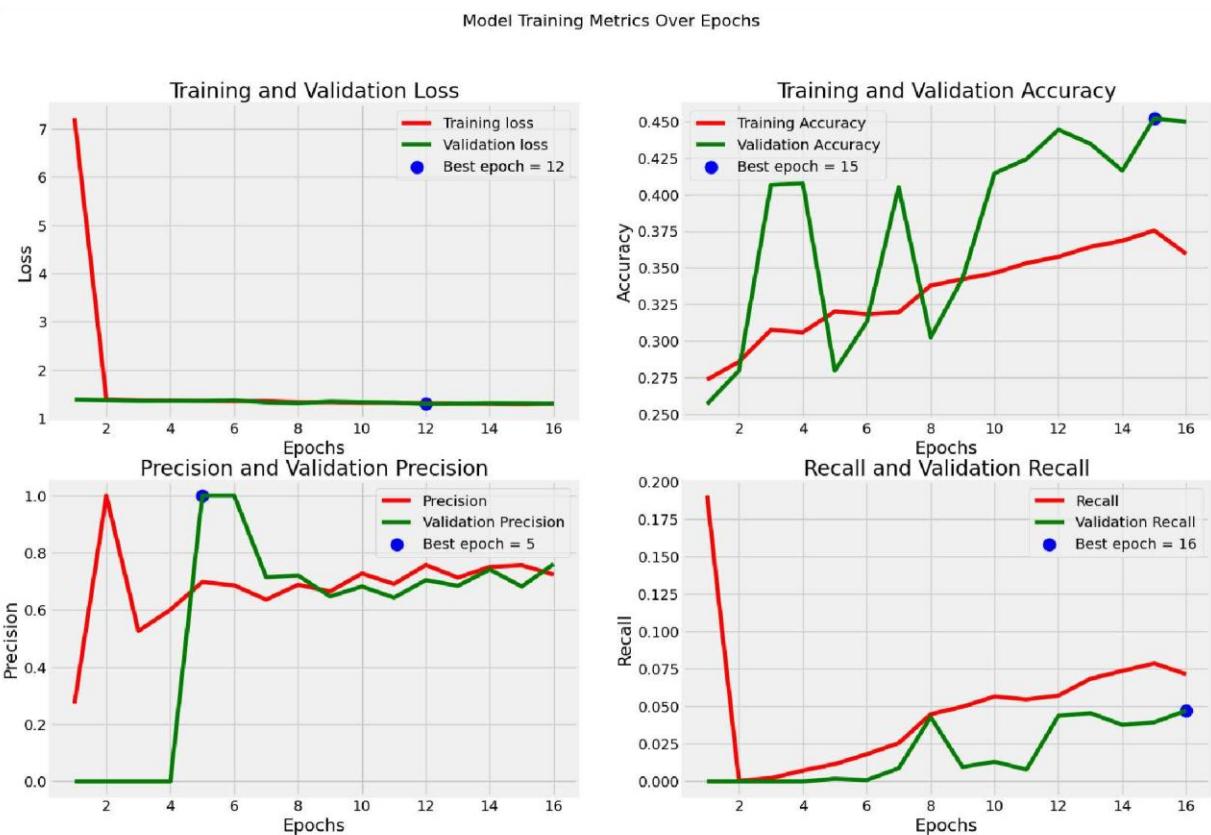
```

plt.subplot(2, 2, 4)
plt.plot(Epochs, tr_recall, 'r', label='Recall')
plt.plot(Epochs, val_recall, 'g', label='Validation Recall')
plt.scatter(index_recall + 1, recall_highest, s=150, c='blue', label=recall_label)
plt.title('Recall and Validation Recall')
plt.xlabel('Epochs')
plt.ylabel('Recall')
plt.legend()
plt.grid(True)

plt.suptitle('Model Training Metrics Over Epochs', fontsize=16)
plt.show()

```

In [24]: `plot_history(history1)`



In [25]: `def plot_confusion_matrix(tst_gen, y_pred):`  
 `"""`

`Plot a confusion matrix for classification results.`

`This function generates a heatmap visualization of the confusion matrix comparing true labels from a test generator with predicted labels.`

`Args:`

```

tst_gen (tf.keras.utils.Sequence or DirectoryIterator):
    Test data generator containing ground truth labels (`classes`)
    and class indices (`class_indices`).
y_pred (array-like of shape (n_samples,)):
    Predicted class labels for the test set.

```

`Plots:`

- Confusion matrix as a seaborn heatmap with:
  - \* Rows = True labels.
  - \* Columns = Predicted labels.
  - \* Annotated cell counts.
  - \* Class names as tick labels.

**Notes:**

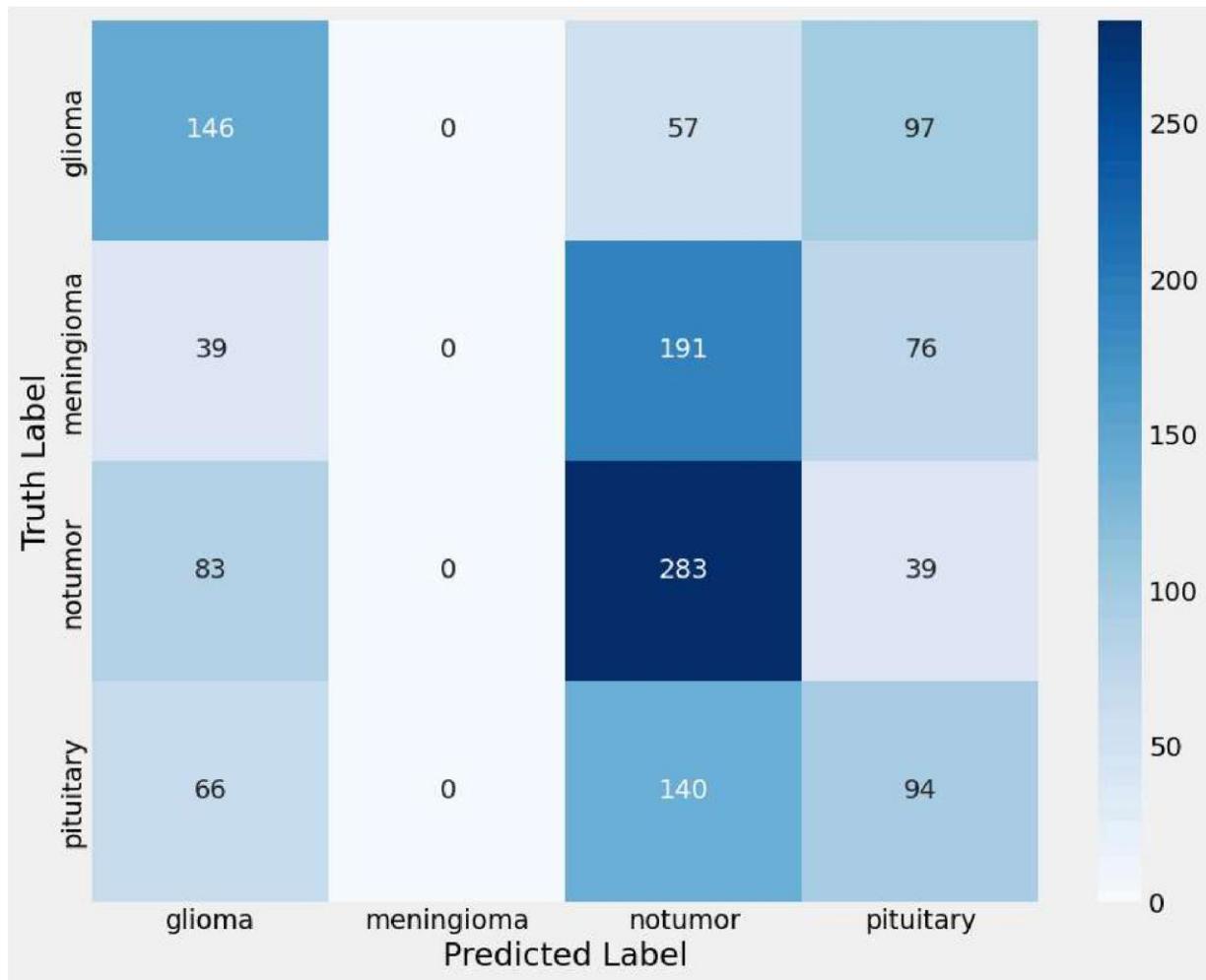
- Assumes `y\_pred` contains class indices (not probabilities).
- Uses matplotlib and seaborn for visualization.

....

```
# Get confusion matrix
cm = confusion_matrix(tst_gen.classes, y_pred)
# Plot confusion matrix
class_dict = tst_gen.class_indices
labels = list(class_dict.keys())
plt.figure(figsize=(10,8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Label')
plt.ylabel('Truth Label')
plt.show()
```

In [26]: # Test model with new data and plot confusion matrix  
preds = model1.predict(test\_generator)  
y\_pred = np.argmax(preds, axis=1)  
plot\_confusion\_matrix(test\_generator, y\_pred)

41/41 ————— 15s 236ms/step



## Transfer learning (Xception)

The Xception architecture excels due to its depthwise separable convolutions, which enable efficient learning of spatial and channel-wise features. Pretrained on ImageNet, Xception provides rich feature representations that can be repurposed for medical imaging tasks, particularly where data is limited.

```
In [27]: def create_model2(num_classes, IMG_SIZE=(224, 224)):
    """
    Build a transfer learning model using Xception as the feature extractor
    with an additional convolutional layer for Grad-CAM interpretability.

    Architecture:
    - Input layer matching the specified IMG_SIZE (height, width, 3).
    - Xception backbone (pretrained, frozen, without top layers).
    - Convolutional layer (3x3, 512 filters, ReLU, same padding) for Grad-CAM.
    - BatchNormalization for training stability.
    - GlobalAveragePooling2D to reduce feature maps to a vector.
    - Dense layer with 128 units and ReLU activation, with Dropout for regularization.
    - Dense output layer with `num_classes` units and softmax activation, explicit
    """

    model = Sequential()
    model.add(Xception(weights='imagenet', include_top=False, input_shape=IMG_SIZE))
    model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))
    model.add(BatchNormalization())
    model.add(GlobalAveragePooling2D())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
```

```
    IMG_SIZE (tuple, optional): Input image size as (height, width). Defaults to (299, 299).
```

**Returns:**

```
    tf.keras.Model: Compiled Keras model ready for training or fine-tuning.
```

**Notes:**

- The base Xception layers are frozen by default (`trainable=False`), but can be unfrozen for fine-tuning.
- The added convolutional layer is designed to support Grad-CAM visualizations by highlighting regions important for model predictions.
- The output layer is set to float32 to ensure numerical stability during training.

```
"""
```

```
# define input
```

```
inputs = Input(shape=(*IMG_SIZE, 3))
```

```
# build the backbone directly
```

```
x = Xception(
    include_top=False,
    weights='imagenet',
    input_tensor=inputs,
    pooling=None
)(inputs)
```

```
# Freeze the Xception base model layers
```

```
x.trainable = False
```

```
# Convolutional Layer after the base model for Grad-CAM
```

```
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='gradcam_conv')(x)
x = BatchNormalization()(x)
```

```
# Dense Layers for classification
```

```
x = GlobalAveragePooling2D()(x)
x = Dropout(0.3)(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.2)(x)
outputs = Dense(num_classes, activation='softmax', dtype='float32')(x)
```

```
model = Model(inputs, outputs)
```

```
return model
```

In [28]: # Create model

```
model2 = create_model2(num_classes)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83683744/83683744 ━━━━━━━━━━━━ 0s 0us/step
```

In [29]: model2.summary()

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
input_layer_2 ( <a href="#">InputLayer</a> )	(None, 224, 224, 3)	0
xception ( <a href="#">Functional</a> )	(None, 7, 7, 2048)	20,861,480
gradcam_conv ( <a href="#">Conv2D</a> )	(None, 7, 7, 512)	9,437,696
batch_normalization_4 ( <a href="#">BatchNormalization</a> )	(None, 7, 7, 512)	2,048
global_average_pooling2d ( <a href="#">GlobalAveragePooling2D</a> )	(None, 512)	0
dropout_2 ( <a href="#">Dropout</a> )	(None, 512)	0
dense_2 ( <a href="#">Dense</a> )	(None, 128)	65,664
dropout_3 ( <a href="#">Dropout</a> )	(None, 128)	0
dense_3 ( <a href="#">Dense</a> )	(None, 4)	516

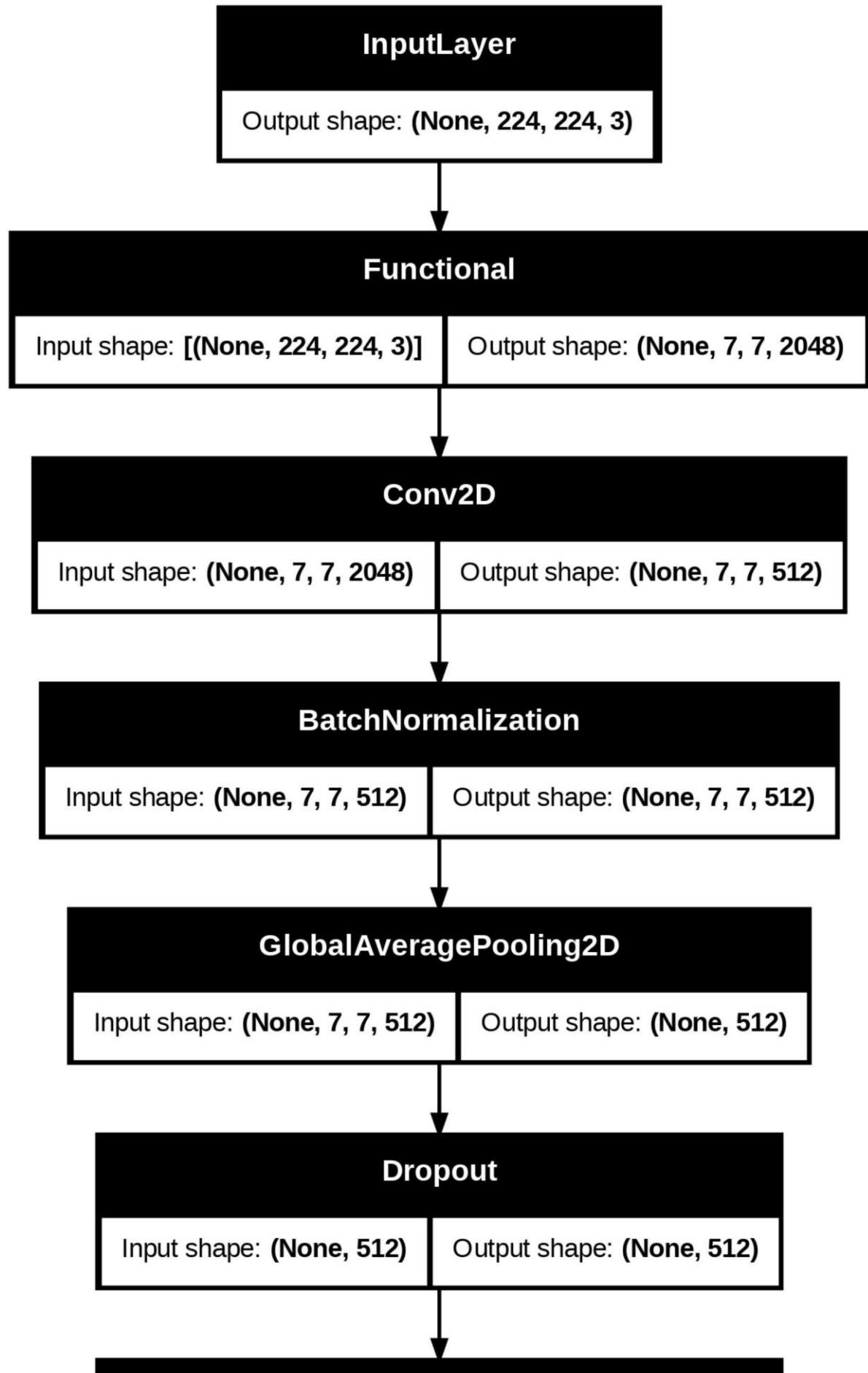
**Total params:** 30,367,404 (115.84 MB)

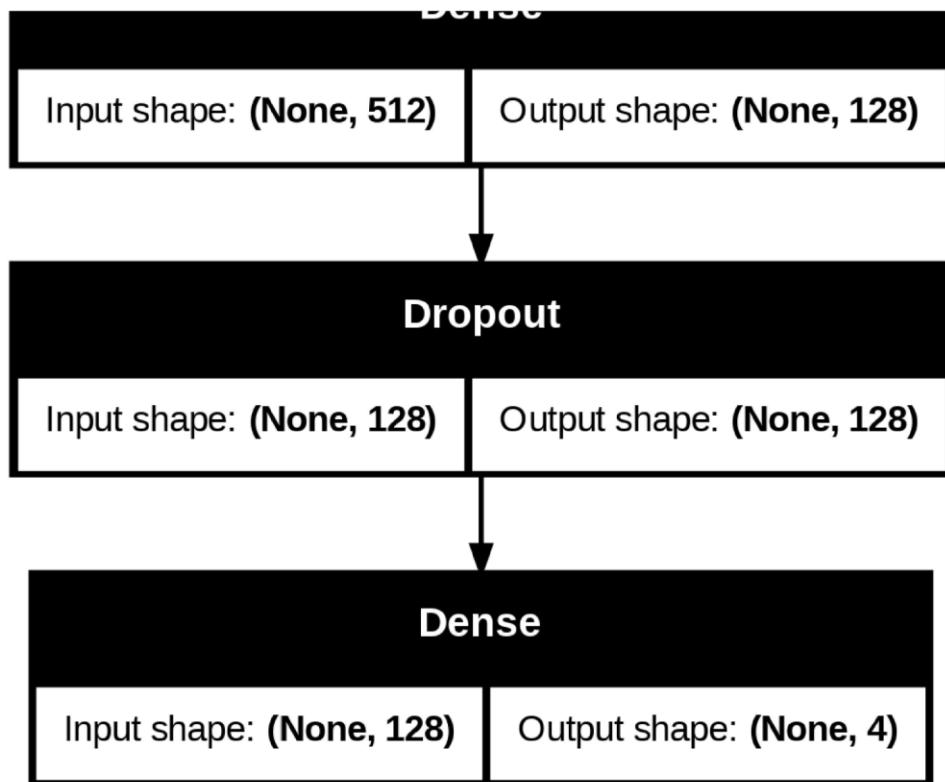
**Trainable params:** 30,311,852 (115.63 MB)

**Non-trainable params:** 55,552 (217.00 KB)

In [30]: `tf.keras.utils.plot_model(model2, show_shapes=True)`

Out[30]:





## Model Compilation

```
In [31]: # Custom Learning rate (often helpful with mixed precision)
initial_learning_rate = 0.001

model2.compile(
    optimizer=tf.keras.optimizers.Adamax(learning_rate=initial_learning_rate),
    loss='categorical_crossentropy',
    metrics=['accuracy', Precision(name='precision'), Recall(name='recall')])
)
```

## Model Training

```
In [32]: # Callbacks
callbacks = [
    tf.keras.callbacks.ModelCheckpoint("best_model2.keras", monitor="accuracy", save_best_only=True),
    tf.keras.callbacks.EarlyStopping(patience=3, restore_best_weights=True),
    tf.keras.callbacks.ReduceLROnPlateau(factor=0.1, patience=3)
]

# Training
history2 = model2.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
    callbacks=callbacks
)
```

```

Epoch 1/10
143/143 277s 1s/step - accuracy: 0.8294 - loss: 0.4510 - precision: 0.8663 - recall: 0.7894 - val_accuracy: 0.8501 - val_loss: 0.4073 - val_precision: 0.8650 - val_recall: 0.8422 - learning_rate: 0.0010
Epoch 2/10
143/143 110s 767ms/step - accuracy: 0.9622 - loss: 0.1078 - precision: 0.9651 - recall: 0.9604 - val_accuracy: 0.8405 - val_loss: 0.5250 - val_precision: 0.8477 - val_recall: 0.8344 - learning_rate: 0.0010
Epoch 3/10
143/143 108s 755ms/step - accuracy: 0.9736 - loss: 0.0916 - precision: 0.9745 - recall: 0.9723 - val_accuracy: 0.9404 - val_loss: 0.2041 - val_precision: 0.9477 - val_recall: 0.9378 - learning_rate: 0.0010
Epoch 4/10
143/143 111s 775ms/step - accuracy: 0.9824 - loss: 0.0577 - precision: 0.9833 - recall: 0.9798 - val_accuracy: 0.9571 - val_loss: 0.1969 - val_precision: 0.9595 - val_recall: 0.9553 - learning_rate: 0.0010
Epoch 5/10
143/143 99s 688ms/step - accuracy: 0.9816 - loss: 0.0507 - precision: 0.9835 - recall: 0.9814 - val_accuracy: 0.9544 - val_loss: 0.1710 - val_precision: 0.9593 - val_recall: 0.9509 - learning_rate: 0.0010
Epoch 6/10
143/143 135s 943ms/step - accuracy: 0.9887 - loss: 0.0398 - precision: 0.9896 - recall: 0.9879 - val_accuracy: 0.9492 - val_loss: 0.1855 - val_precision: 0.9515 - val_recall: 0.9448 - learning_rate: 0.0010
Epoch 7/10
143/143 111s 773ms/step - accuracy: 0.9899 - loss: 0.0314 - precision: 0.9909 - recall: 0.9891 - val_accuracy: 0.9632 - val_loss: 0.1578 - val_precision: 0.9640 - val_recall: 0.9623 - learning_rate: 0.0010
Epoch 8/10
143/143 98s 683ms/step - accuracy: 0.9867 - loss: 0.0431 - precision: 0.9870 - recall: 0.9856 - val_accuracy: 0.9711 - val_loss: 0.1165 - val_precision: 0.9719 - val_recall: 0.9702 - learning_rate: 0.0010
Epoch 9/10
143/143 113s 790ms/step - accuracy: 0.9925 - loss: 0.0181 - precision: 0.9936 - recall: 0.9925 - val_accuracy: 0.9614 - val_loss: 0.1526 - val_precision: 0.9665 - val_recall: 0.9614 - learning_rate: 0.0010
Epoch 10/10
143/143 98s 687ms/step - accuracy: 0.9951 - loss: 0.0189 - precision: 0.9953 - recall: 0.9946 - val_accuracy: 0.9544 - val_loss: 0.1591 - val_precision: 0.9560 - val_recall: 0.9527 - learning_rate: 0.0010

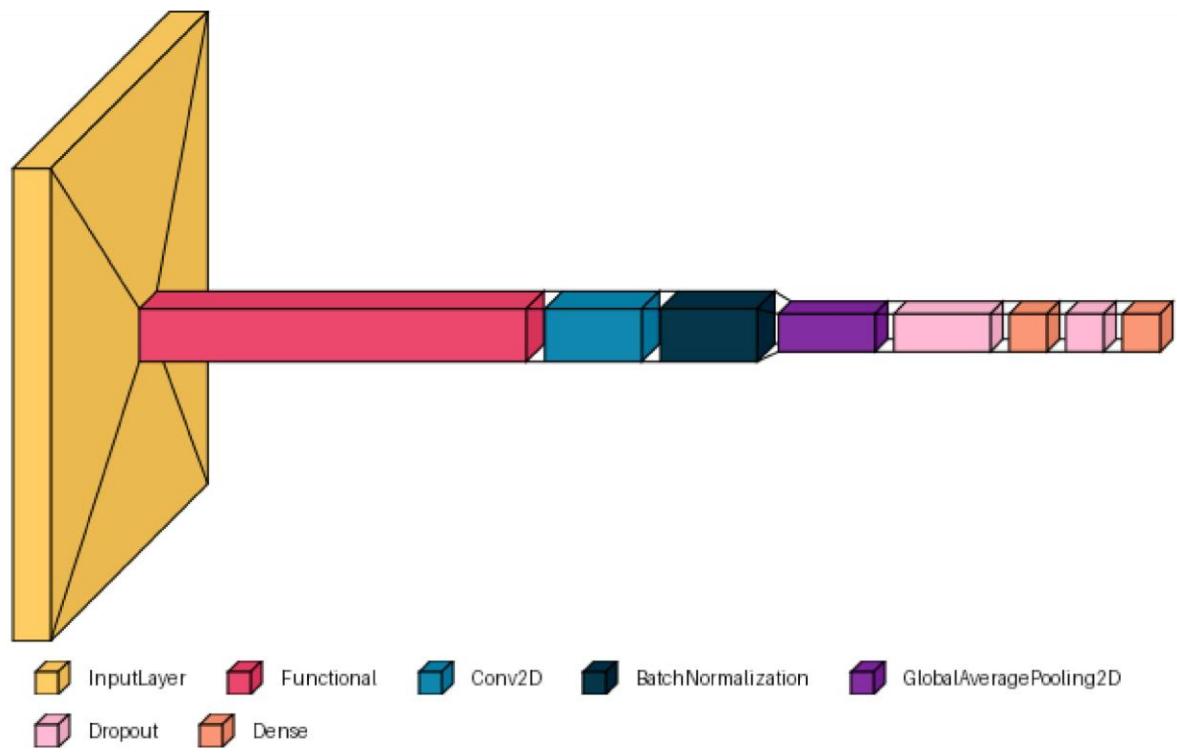
```

In [33]: # Visualize layer model

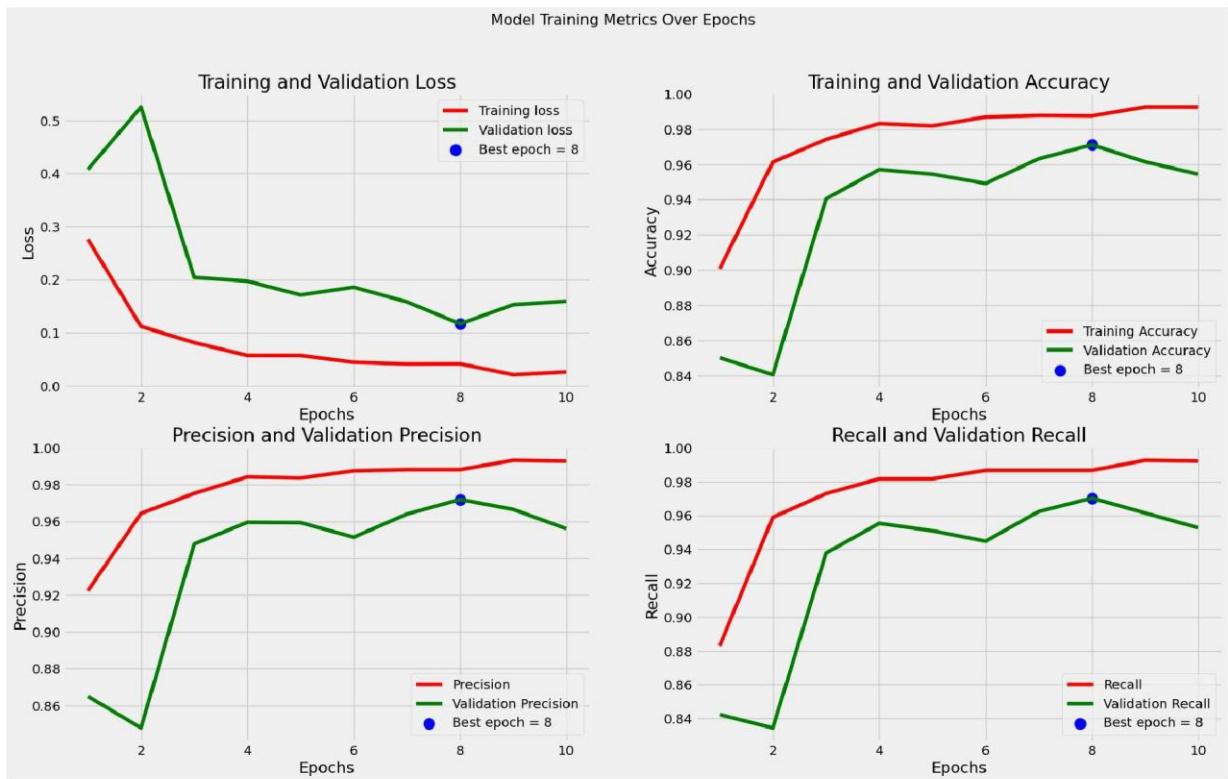
```
layered_view(model2, legend=True, max_xy=250)
```

```
/usr/local/lib/python3.12/dist-packages/visualkeras/layered.py:86: UserWarning: The legend_text_spacing_offset parameter is deprecated and will be removed in a future release.
    warnings.warn("The legend_text_spacing_offset parameter is deprecated and will be removed in a future release.")
```

Out[33]:



## Model Evaluation

In [34]: `plot_history(history2)`


Training metrics shows that the best epoch is the 8th

In [35]: `test2_metrics = model_evaluation(model2, train_generator, test_generator)`

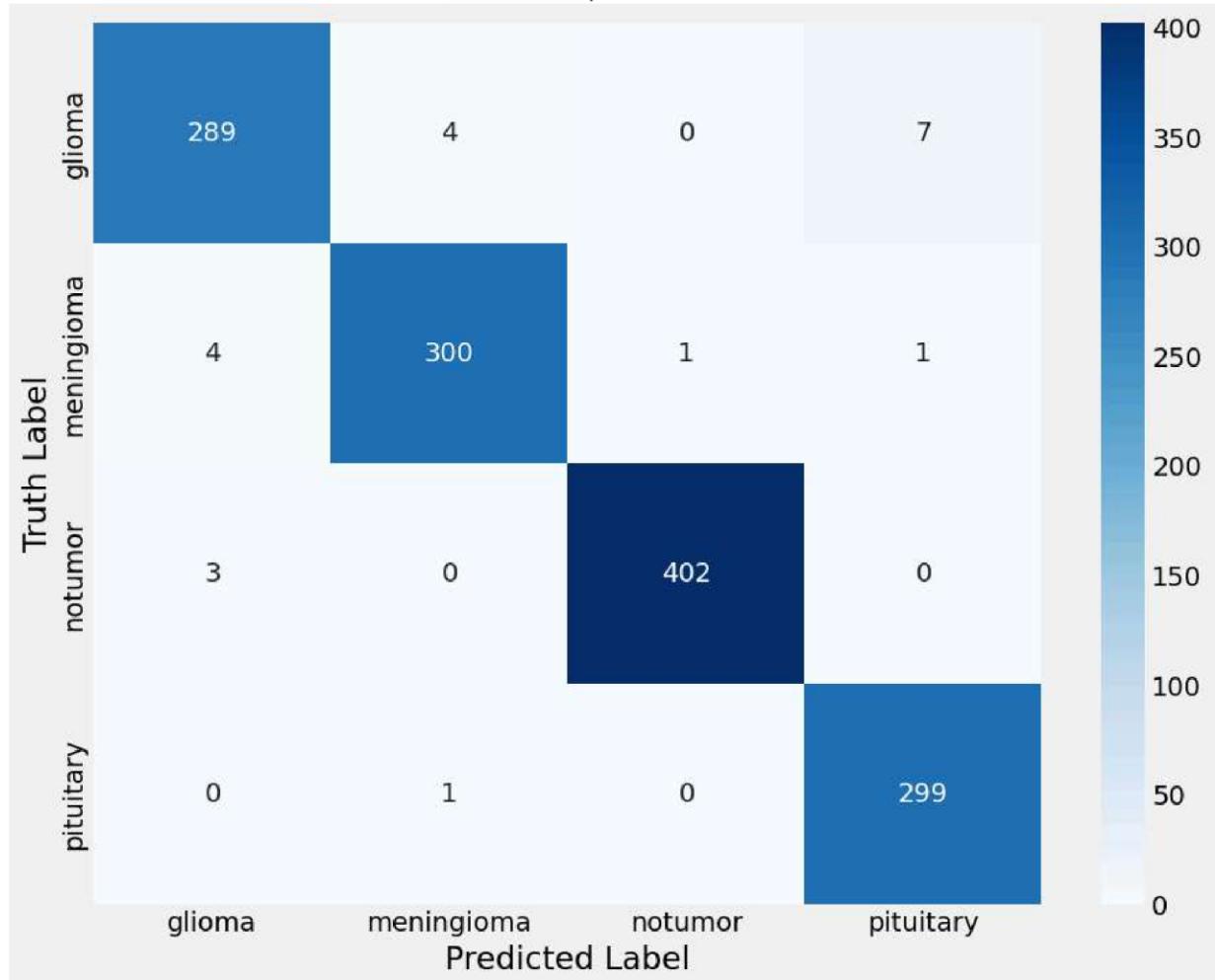
```
143/143 ━━━━━━━━━━ 67s 465ms/step - accuracy: 0.9952 - loss: 0.0101 - precision: 0.9952 - recall: 0.9952
41/41 ━━━━━━━━━━ 23s 560ms/step - accuracy: 0.9728 - loss: 0.1021 - precision: 0.9728 - recall: 0.9728
Train Loss: 0.0097
Train Accuracy: 99.58%
-----
Test Loss: 0.0604
Test Accuracy: 98.40%
```

In [36]: `test2_metrics`

Out[36]: `{'accuracy': 0.98, 'precision': 0.98, 'recall': 0.98}`

In [37]: `preds = model2.predict(test_generator)
y_pred = np.argmax(preds, axis=1)
plot_confusion_matrix(test_generator, y_pred)`

41/41 ━━━━━━━━━━ 14s 239ms/step



## Build CNN from Scratch

A custom CNN built from scratch, allows full control over the feature extraction process specifically tailored to MRI images. The use of multiple convolutional blocks with increasing

filter sizes enables hierarchical feature learning, capturing both low-level patterns (edges, textures) and high-level structures (tumor shapes and regions).

```
In [38]: ## Model definition
def create_model3(num_classes):
    """
        Build a convolutional neural network (CNN) model from scratch for image classification.

        Architecture:
        - Input layer matching the global IMG_SIZE (height, width, 3).
        - Initial Conv2D layer (64 filters, 7x7 kernel, ReLU) with BatchNormalization and Dropout(0.25).
        - Three convolutional blocks with increasing filters (128, 256, 512), each containing:
            * Conv2D layer (3x3, ReLU, same padding)
            * BatchNormalization
            * MaxPooling2D
            * Dropout(0.25)
        - GlobalAveragePooling2D to reduce feature maps to a vector.
        - Fully connected layers with decreasing units (1024 → 512 → 128), each with:
            * ReLU activation
            * L2 kernel regularization
            * BatchNormalization
            * Dropout (0.5 → 0.3 → 0.2)
        - Dense output layer with softmax activation (float32) for classification.

        Args:
            num_classes (int): Number of target classes for classification.

        Returns:
            tf.keras.Model: Keras model instance ready for compilation and training.
    """

    # Create new model from scratch
    inputs = tf.keras.Input(shape=(*IMG_SIZE, 3))
    x = Conv2D(64, (7,7), activation='relu', padding='same')(inputs)
    x = BatchNormalization()(x)
    x = MaxPooling2D((2,2))(x)

    # Multiple convolutional blocks with increasing filters
    filters = [128, 256, 512]
    for f in filters:
        x = Conv2D(f, (3,3), activation='relu', padding='same')(x)
        x = BatchNormalization()(x)
        x = MaxPooling2D((2,2))(x)
        x = Dropout(0.25)(x)

    # Global average pooling instead of flatten
    x = GlobalAveragePooling2D()(x)

    # Fully connected layers
    x = Dense(1024, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001))(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)

    x = Dense(512, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001))(x)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)

    x = Dense(128, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001))(x)
    x = BatchNormalization()(x)
    x = Dropout(0.2)(x)

    # Output layer
    outputs = Dense(num_classes, activation='softmax')(x)

    # Create the model
    model = tf.keras.Model(inputs=inputs, outputs=outputs)

    return model
```

```

x = Dense(128, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))(x)
x = BatchNormalization()(x)
x = Dropout(0.2)(x)

# Ensure output layer is float32 for stability
outputs = Dense(num_classes, activation='softmax', dtype='float32')(x)

model = Model(inputs, outputs)

return model

```

In [39]:

```

# Create model
model3 = create_model3(num_classes)

for layer in model3.layers:
    print(f"{layer.name}: {layer.dtype_policy}")

```

```

input_layer_3: <DTypePolicy "float32">
conv2d_4: <DTypePolicy "float32">
batch_normalization_5: <DTypePolicy "float32">
max_pooling2d: <DTypePolicy "float32">
conv2d_5: <DTypePolicy "float32">
batch_normalization_6: <DTypePolicy "float32">
max_pooling2d_1: <DTypePolicy "float32">
dropout_4: <DTypePolicy "float32">
conv2d_6: <DTypePolicy "float32">
batch_normalization_7: <DTypePolicy "float32">
max_pooling2d_2: <DTypePolicy "float32">
dropout_5: <DTypePolicy "float32">
conv2d_7: <DTypePolicy "float32">
batch_normalization_8: <DTypePolicy "float32">
max_pooling2d_3: <DTypePolicy "float32">
dropout_6: <DTypePolicy "float32">
global_average_pooling2d_1: <DTypePolicy "float32">
dense_4: <DTypePolicy "float32">
batch_normalization_9: <DTypePolicy "float32">
dropout_7: <DTypePolicy "float32">
dense_5: <DTypePolicy "float32">
batch_normalization_10: <DTypePolicy "float32">
dropout_8: <DTypePolicy "float32">
dense_6: <DTypePolicy "float32">
batch_normalization_11: <DTypePolicy "float32">
dropout_9: <DTypePolicy "float32">
dense_7: <DTypePolicy "float32">

```

In [40]:

```
model3.summary()
```

**Model: "functional\_2"**

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 224, 224, 3)	0
conv2d_4 (Conv2D)	(None, 224, 224, 64)	9,472
batch_normalization_5 (BatchNormalization)	(None, 224, 224, 64)	256
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_5 (Conv2D)	(None, 112, 112, 128)	73,856
batch_normalization_6 (BatchNormalization)	(None, 112, 112, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 128)	0
dropout_4 (Dropout)	(None, 56, 56, 128)	0
conv2d_6 (Conv2D)	(None, 56, 56, 256)	295,168
batch_normalization_7 (BatchNormalization)	(None, 56, 56, 256)	1,024
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 256)	0
dropout_5 (Dropout)	(None, 28, 28, 256)	0
conv2d_7 (Conv2D)	(None, 28, 28, 512)	1,180,160
batch_normalization_8 (BatchNormalization)	(None, 28, 28, 512)	2,048
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 512)	0
dropout_6 (Dropout)	(None, 14, 14, 512)	0
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 512)	0
dense_4 (Dense)	(None, 1024)	525,312
batch_normalization_9 (BatchNormalization)	(None, 1024)	4,096
dropout_7 (Dropout)	(None, 1024)	0
dense_5 (Dense)	(None, 512)	524,800
batch_normalization_10 (BatchNormalization)	(None, 512)	2,048
dropout_8 (Dropout)	(None, 512)	0

dense_6 (Dense)	(None, 128)	65,664
batch_normalization_11 (BatchNormalization)	(None, 128)	512
dropout_9 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 4)	516

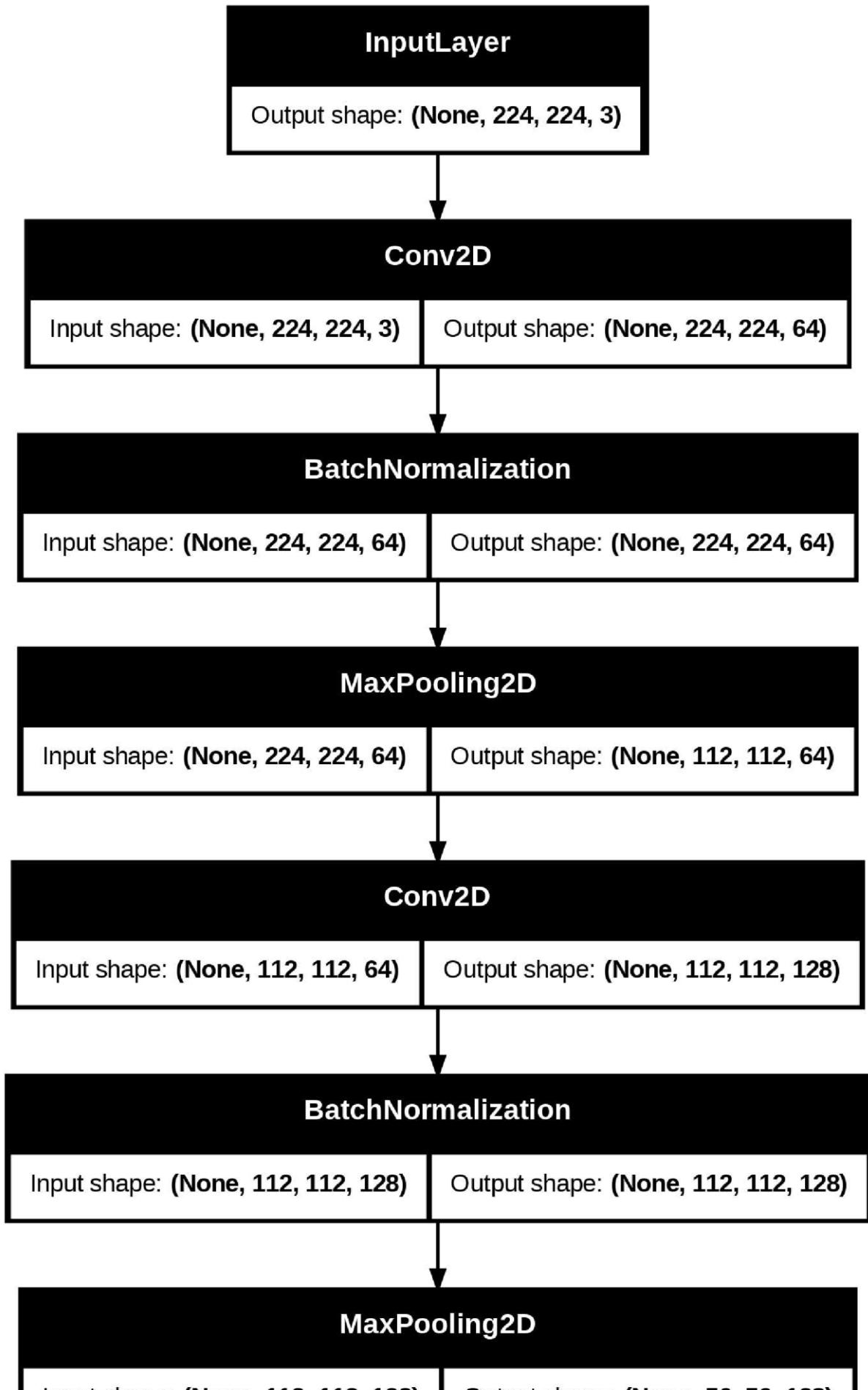
**Total params:** 2,685,444 (10.24 MB)

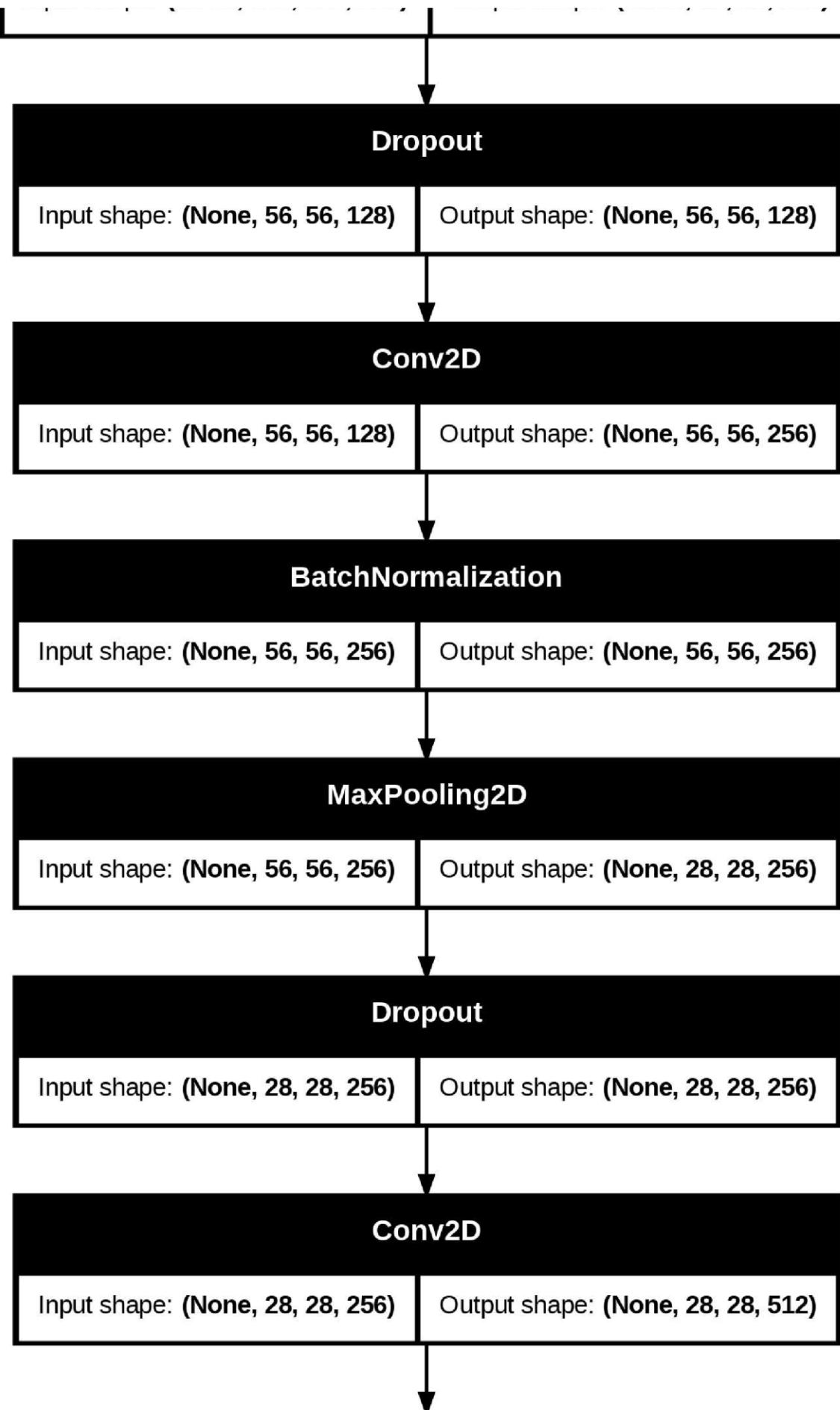
**Trainable params:** 2,680,196 (10.22 MB)

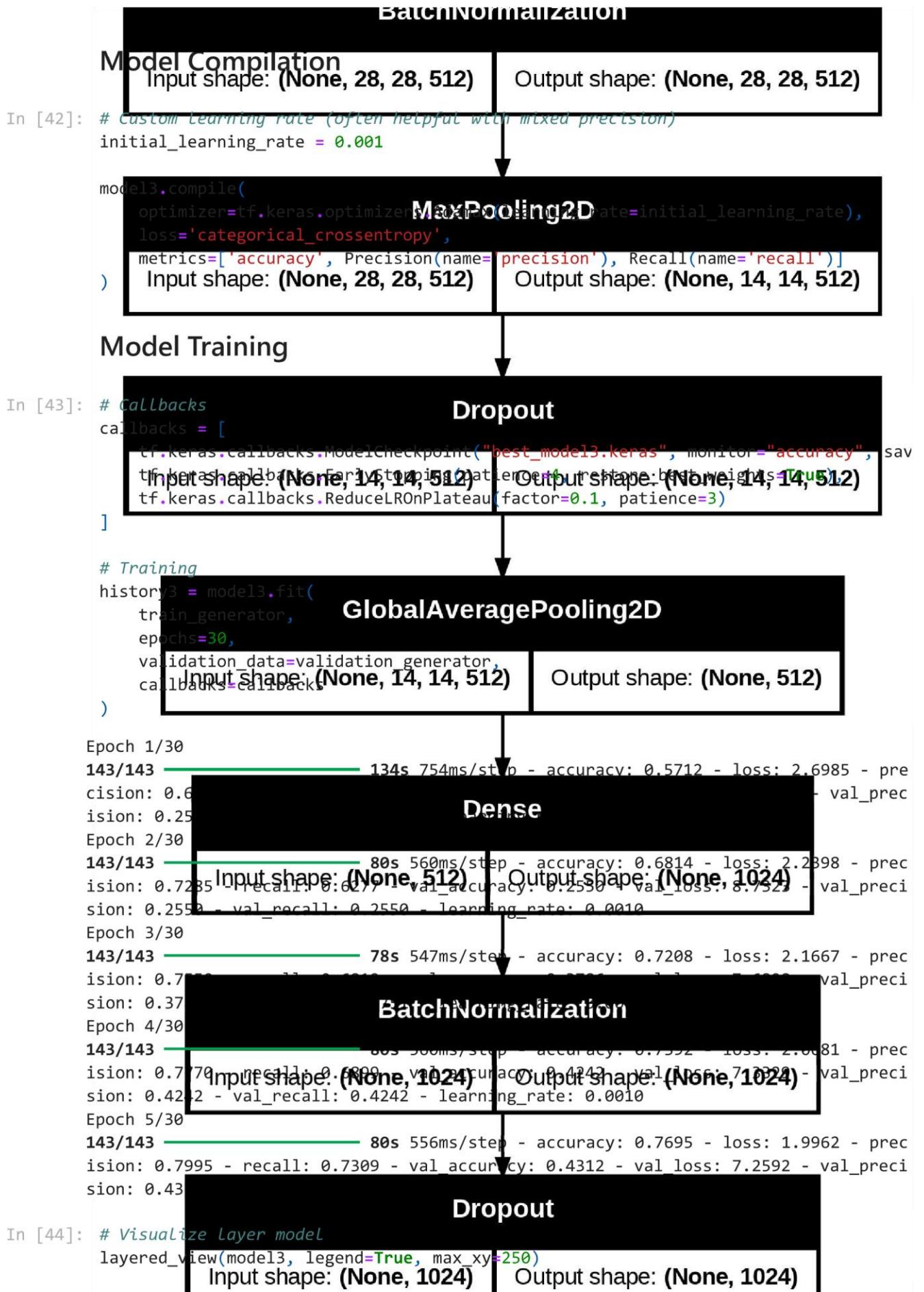
**Non-trainable params:** 5,248 (20.50 KB)

```
In [41]: tf.keras.utils.plot_model(model3, show_shapes=True)
```

Out[41]:

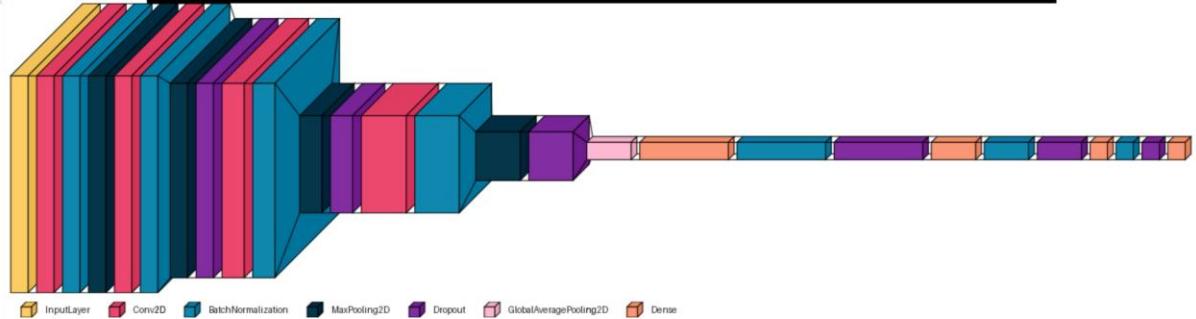






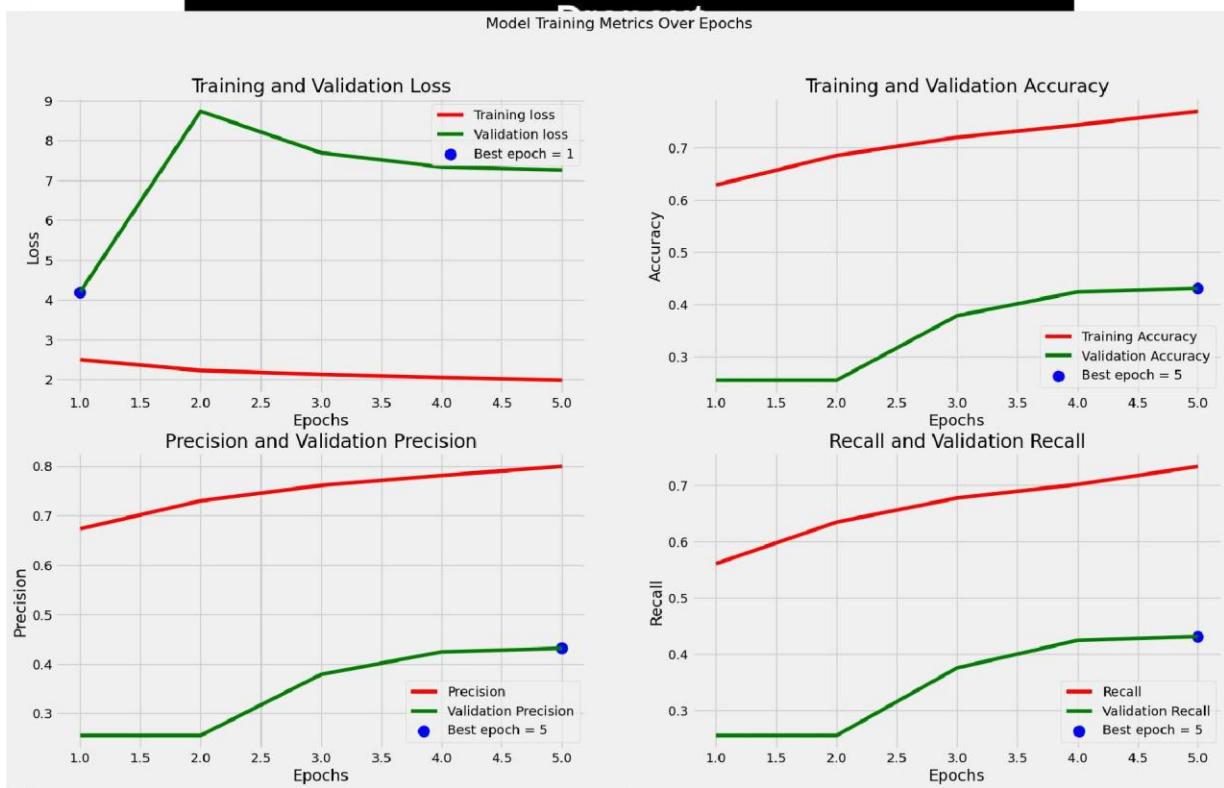
```
/usr/local/lib/python3.12/dist-packages/visualkeras/layered.py:86: UserWarning: The
legend_text_spacing offset parameter is deprecated and will be removed in a future r
elease.
  warnings.warn("The legend_text_spacing offset parameter is depre
d will be
removed in
```

Out[44]:



## Model Evaluation

In [45]: `plot_history(history3)`



The history metrics plot shows that the 5th epoch was the one with best performance, but the model does not seem to have been trained well.

## Dropout

In [46]: `test3_metrics = model_evaluation(model3, train_generator, test_generator)`

Input shape: (None, 128)

Output shape: (None, 128)

## Dense

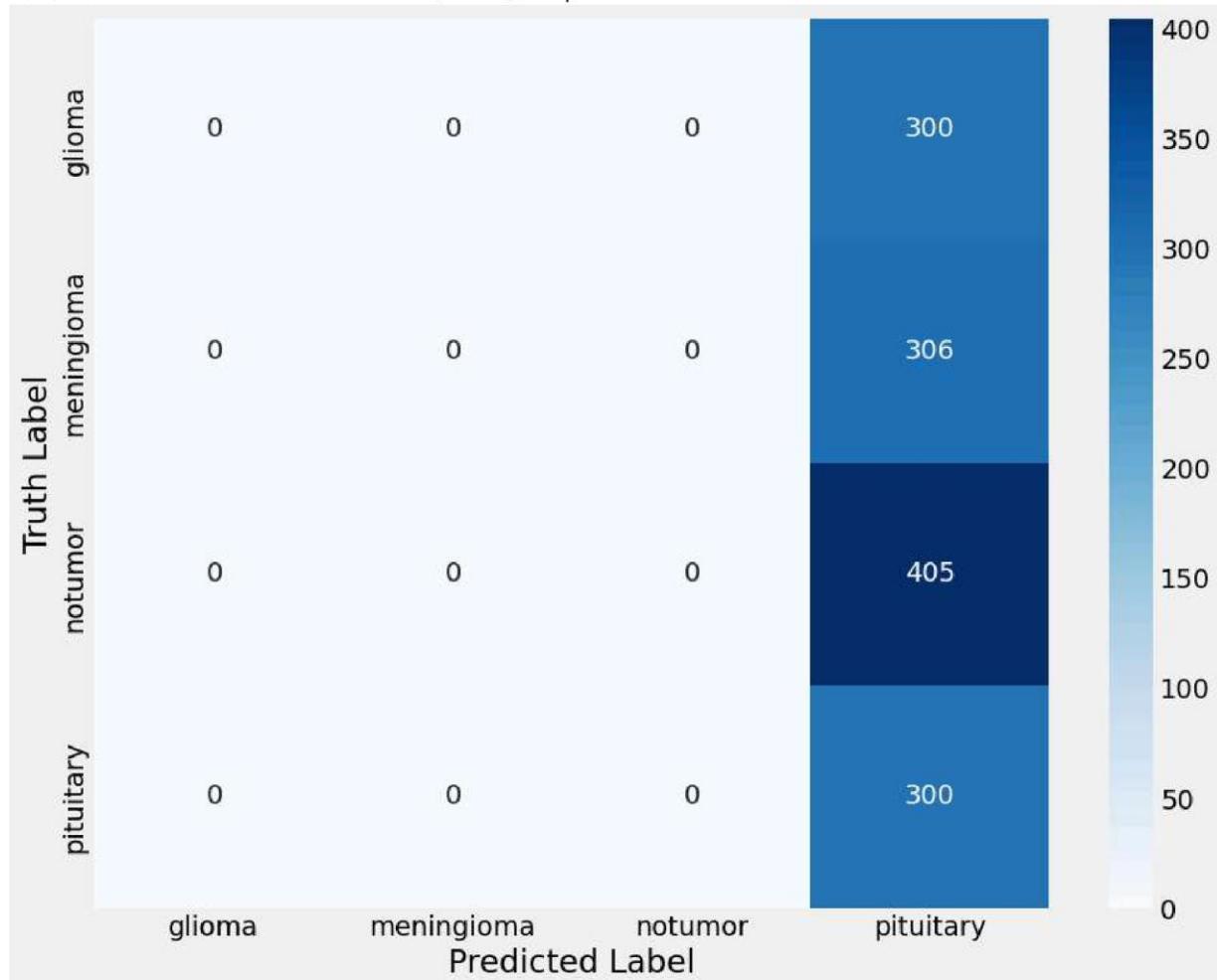
```
143/143 ━━━━━━━━ 63s 442ms/step - accuracy: 0.2500 - loss: 4.2202 - prec
ision: 0.2500 - Input shape: (None, 128) Output shape: (None, 4)
41/41 ━━━━━━━━ 8s 209ms/step - accuracy: 0.0361 - loss: 4.7450 - precisi
on: 0.0361 - recall: 0.0361
Train Loss: 4.1912
Train Accuracy: 25.51%
-----
Test Loss: 4.2850
Test Accuracy: 22.88%
```

In [47]: test3\_metrics

Out[47]: {'accuracy': 0.23, 'precision': 0.23, 'recall': 0.23}

In [48]: preds = model3.predict(test\_generator)
y\_pred = np.argmax(preds, axis=1)
plot\_confusion\_matrix(test\_generator, y\_pred)

41/41 ━━━━━━━━ 4s 84ms/step



## Best model

In [49]: # Show a model metrics comparation
metrics\_data = [test1\_metrics, test2\_metrics, test3\_metrics]
model\_names = ['Model 1', 'Model 2', 'Model 3']

```

metric_names = 'accuracy', 'precision', 'recall'

# Create grouped bar plot
plt.figure(figsize=(12, 7))

x = np.arange(len(model_names))
width = 0.25

for i, metric in enumerate(metric_names):
    values = [data[metric] for data in metrics_data]
    plt.bar(x + i * width, values, width, label=metric.capitalize())
    t

plt.xlabel('Models', fontweight='bold')
plt.ylabel('Scores', fontweight='bold')
plt.title('Model Performance Comparison', fontweight='bold')
plt.xticks(x + width, model_names)
plt.legend()
plt.ylim(0, 1)

# Add value Labels
for i, model_data in enumerate(metrics_data):
    for j, metric in enumerate(metric_names):
        plt.text(i + j * width, model_data[metric] + 0.01,
                 f'{model_data[metric]:.3f}', ha='center', va='bottom', fontsize=9)

plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()

```



From the 3 experiments the one with the best performance is the Xception architecture.

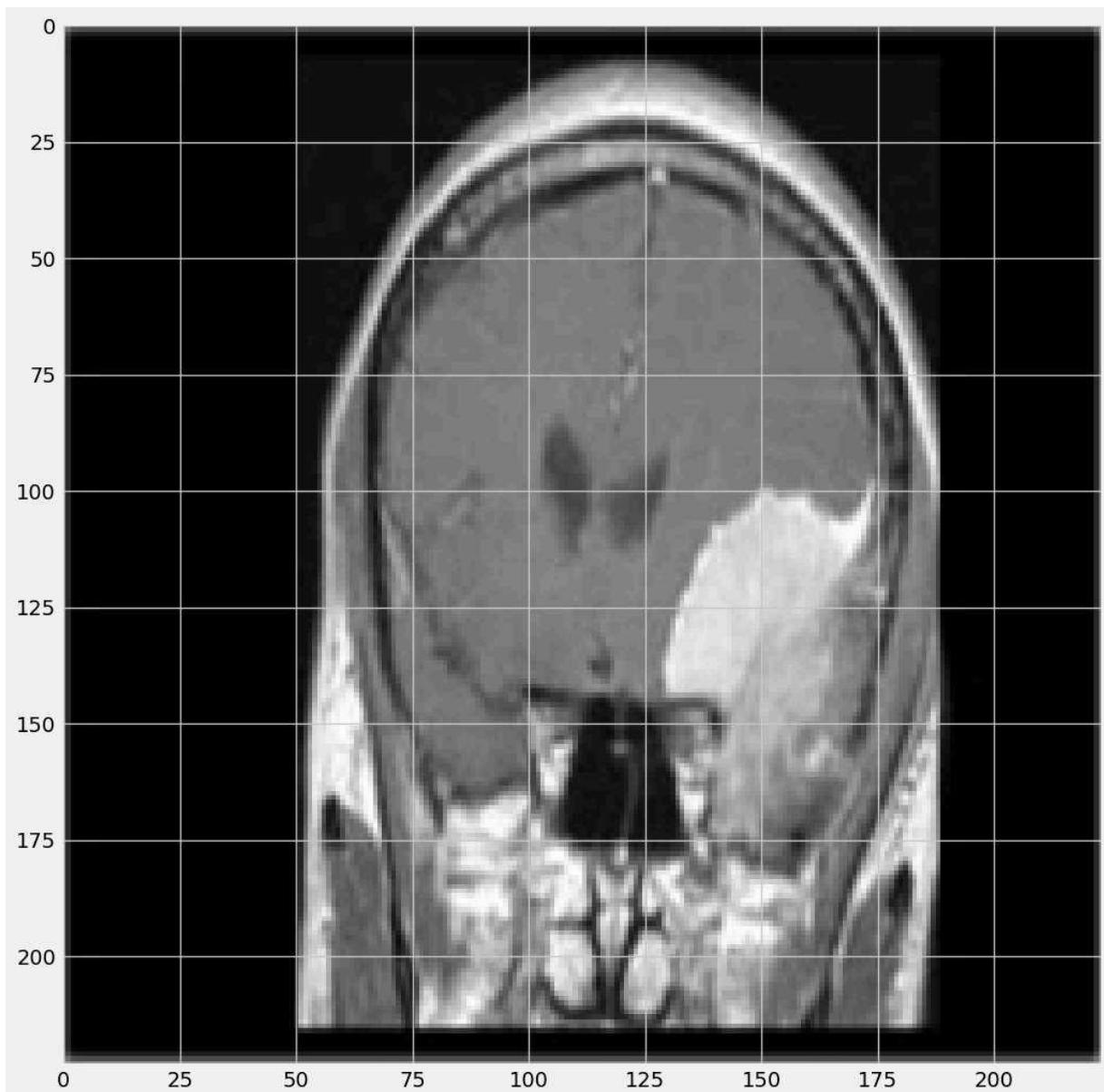
## Inference with best model

```
In [54]: model = tf.keras.models.load_model("best_model2.keras")
class_dict = {'Glioma':0, 'Meningioma':1, 'Pituitary':2, 'No Tumor':3}

def predict(img_path):
    import numpy as np
    import matplotlib.pyplot as plt
    from PIL import Image
    label = list(class_dict.keys())
    plt.figure(figsize=(12, 12))
    img = Image.open(img_path)
    resized_img = img.resize((224, 224))
    img = np.asarray(resized_img)
    img = np.expand_dims(img, axis=0)
    img = img / 255
    predictions = model.predict(img)
    probs = list(predictions[0])
    labels = label
    plt.imshow(resized_img)
    plt.show()
    for i in range(len(labels)):
        print(f"{labels[i]}: {probs[i]*100:.2f}%")

predict('/kaggle/input/brain-tumor-mri-dataset/Testing/meningioma/Te-meTr_0000.jpg')

1/1 ━━━━━━━━━━ 5s 5s/step
[[3.0375380e-05 9.9931097e-01 4.5102736e-04 2.0762616e-04]]
```



Glioma: 0.00%

Meningioma: 99.93%

Pituitary: 0.05%

No Tumor: 0.02%

## 7.2 Inferencia y Grad-CAM

```
def generate_heatmap(self, img_array: np.ndarray, top_pred_index: int):
    """
    Generate Grad-CAM heatmap for a given image.

    Args:
        img_array: Preprocessed image array
        top_pred_index: Index of the top predicted class

    Returns:
        Grad-CAM heatmap or None if generation fails
    """
    try:
        grad_model_conv, classifier_model = self._build_grad_model()
        with tf.GradientTape() as tape:
            last_conv_layer_output = grad_model_conv(img_array)
            tape.watch(last_conv_layer_output)
            preds = classifier_model(last_conv_layer_output)
            # Handle different model output formats
            if isinstance(preds, list):
                predictions_tensor = preds[1]
            else:
                predictions_tensor = preds
            top_class_channel = predictions_tensor[:, top_pred_index]
            # Compute gradients
            grads = tape.gradient(top_class_channel, last_conv_layer_output)
            pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
            # Generate heatmap
            last_conv_layer_output = last_conv_layer_output[0]
            heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
            heatmap = tf.squeeze(heatmap)
            # Apply ReLU and normalize
            heatmap = tf.maximum(heatmap, 0)
            max_heatmap = tf.reduce_max(heatmap)
            if max_heatmap != 0:
                heatmap /= max_heatmap
        return heatmap.numpy()
    
```

```
def superimpose_heatmap(self, img_path: str, heatmap: np.ndarray) -> np.ndarray:
    """
    Superimpose heatmap on the original image.

    Args:
        img_path: Path to the original image
        heatmap: Grad-CAM heatmap

    Returns:
        Image with superimposed heatmap
    """
    # Load and resize original image
    img = Image.open(img_path).convert('RGB')
    img = img.resize(self.config.img_size)
    img_array = np.asarray(img).astype(np.uint8)

    # Resize heatmap to match image dimensions
    heatmap = cv2.resize(heatmap, (img_array.shape[1], img_array.shape[0]))

    # Apply colormap to heatmap
    heatmap_colored = np.uint8(255 * heatmap)
    heatmap_colored = cv2.applyColorMap(heatmap_colored, cv2.COLORMAP_JET)
    heatmap_colored = cv2.cvtColor(heatmap_colored, cv2.COLOR_BGR2RGB)

    # Blend images
    superimposed_img = cv2.addWeighted(
        img_array.astype(np.uint8), 0.6,
        heatmap_colored.astype(np.uint8), 0.4, 0)
    return superimposed_img
```

### 7.3 Prompt

You are a medical AI assistant specialized in radiology and oncology. You are given an image that contains three panels:

- Left: Original brain MRI/X-ray.
- Center: Grad-CAM heatmap highlighting regions the CNN focused on.
- Right: Superimposed heatmap over the MRI showing tumor localization.
- Below the images, inference scores for multiple tumor classes are provided (e.g., Glioma, Meningioma, Pituitary, No Tumor).

Tasks:

1. Analyze the image: Describe what the model focused on and summarize predicted class and scores.
2. Interpretation disclaimer: State that AI predictions are not diagnostic and require confirmation from a radiologist or oncologist.
3. Complementary medical resources: Search in the web for 3 to 5 authoritative references with URLs on:
  - Diagnostic imaging protocols for brain tumors.
  - Recommended next steps to confirm the type of tumor predicted diagnosis.
  - Clinical guidelines or treatment pathways from reliable institutions.
  - Patient-oriented resources from recognized cancer centers or associations.
4. Do not include speculative or unverified content. Only return references from NIH, NCI, WHO, FDA, PubMed, or equivalent.

Output format:

- Medical Disclaimer for Health Care professionals
- Model Analysis (Image + Heatmap)
- Predicted Class & Scores
- Complementary Resources (with URLs)

## 7.4 Explicabilidad LLM

```
def chat_completion(prompt:str, img_path:str):
    """
    Generate a grounded response from Gemini using a text prompt and an image file.
    This function sets up a Gemini client, uploads an image, and performs a content
    generation request with Google Search grounding enabled. It returns the raw
    response object from the API, which can later be processed with `add_citations()`.

    Args:
        prompt (str):
            The user query or instruction to send to the model.
        img_path (str):
            Path to the image file to include in the request.

    Returns:
        google.genai.types.GenerateContentResponse:
            The raw response object containing generated text, candidates,
            and grounding metadata.

    """
    # Configure the client
    client = genai.Client()
    # Define the grounding tool
    grounding_tool = types.Tool(google_search=types.GoogleSearch())
    # Configure generation settings
    config = types.GenerateContentConfig(tools=[grounding_tool])
    my_file = client.files.upload(file=f"{img_path}")
    # Make the request
    response = client.models.generate_content(
        model="gemini-2.5-flash",
        contents=[my_file, prompt],
        config=config,
```

## 7.5 Interfaz gráfica

```
# App title
st.title("Brain Tumor Detector Tool")

# Input field for case name
title = st.text_input(
    "Case name:",
    key='input_text',
    help="Write your case name and press enter"
)
if len(title) > 0:
    # Build unique case identifier with timestamp
    case_name = f"{title}_{date}"
    case_name = case_name.replace(" ", "").replace("-", "").replace(":", "").lower()
    # File uploader for MRI brain image
    uploaded_file = st.file_uploader(
        "Upload MRI Brain Image",
        type=["jpg", "jpeg", "png"]
    )
    if uploaded_file is not None:
        # Read uploaded image into bytes
        image_bytes = uploaded_file.getvalue()
        # Open image with Pillow
        image = Image.open(io.BytesIO(image_bytes))
        # Display uploaded image
        st.image(image, caption=f"Study case: {case_name}", width='stretch')
        # Action button to start inference
        make_inference = st.button("Proceed with analysis", width='stretch',
        type='primary')
        if make_inference:
            # Save uploaded image locally
            os.makedirs(f"cases/{case_name}")
            img_path = f"cases/{case_name}/{uploaded_file.name}"
            img_path_inf = f"cases/{case_name}/inference_{uploaded_file.name}"
            with open(img_path, "wb") as f:
                f.write(image_bytes)
```

```
# Run model inference and GradCAM visualization
with st.spinner("Waiting the MRI image is being analyzed...", show_time=True):
    results = model.visualize(img_path, img_path_inf)
    # Show predicted class
    st.header(f"Predicted Class: {results['predicted_class_name']}")  

    # Show prediction scores
    scores_plot = plot_scores(results, model.config.class_names)
    st.plotly_chart(scores_plot, use_container_width=True)
    # Show GradCAM visualization
    st.image(img_path_inf, case_name)

# Run complementary analysis with LLM + citations
with st.spinner("Finding complementary information...", show_time=True):
    prompt = build_prompt()
    response = chat_completion(prompt, img_path_inf)
    response_formatted = add_citations(response)
    if len(response_formatted) > 0:
        with st.container(border=True, height=800):
            st.markdown(response_formatted)

# Button to reset for a new case study
new_case = st.button(
    'New case study',
    width='stretch',
    type='tertiary',
    on_click=reset_controls
)
```