Brain Tumor Diagnosing through Neural Networks

Christopher Goff

The University of Tennessee, Knoxville, cgoff6@vols.utk.edu

Ewan McGarvey

The University of Tennessee, Knoxville, emcgarve@vols.utk.edu

Soe Thet Ko

The University of Tennessee, Knoxville, sko2@vols.utk.edu

Brain tumor diagnosis from Magnetic Resonance Imaging (MRI) is a critical task, as precise detection of brain tumors helps improve clinical outcomes and patients' quality of life. The objective of this project was to develop a deep neural network capable of classifying brain tumors from MRI scans with a target accuracy over 97%. For model training, we utilized the Brain Tumor MRI Dataset [1], which contains 7023 MRI images split across four classes: glioma tumor, meningioma tumor, pituitary tumor, and no tumor. We explored two variations of a custom convolutional neural network architecture, which leveraged depthwise separable convolution along with residual connections. Additionally, we fine-tuned six different models based on the pre-trained EfficientNet-B0 architecture. All models were evaluated using accuracy, precision, recall and f1-score metrics, and their performance was further assessed with the use of a confusion matrix. Our findings revealed that the best performing models, fine-tuned EfficientNet-B0 variants, achieved over 98% overall accuracy. These results demonstrate the effectiveness of fine-tuned deep learning models for high-accuracy medical image classification, surpassing our target performance and potentially contributing to faster and more reliable diagnostic tools in clinical settings.

CCS CONCEPTS • Computing methodologies ~ Neural networks • Applied computing ~ Health informatics • Applied computing ~ Medical imaging

Additional Keywords and Phrases: brain tumor diagnosis, magnetic resonance imaging, MRI, deep learning, convolutional neural networks, CNN, EfficientNet, transfer learning, medical image classification

1 INTRODUCTION

Brain tumors are abnormal growths of cells in and around the brain and potentially life-threatening. They require precise and timely diagnosis for the patients to get effective treatment, ultimately improving clinical outcomes and quality of life. It is difficult to correctly diagnose these aberrant conditions that are concealed within the intricate brain patterns. Magnetic Resonance Imaging (MRI) has become a vital tool in this process, providing fine-grained pictures of the structure of the brain. The sheer number of MRI scans and the minute differences that can differentiate between various tumor types and healthy tissues however make manual analysis a time-consuming and challenging task.

In the field of artificial intelligence, deep learning has demonstrated tremendous promise in automating and improving medical image analysis. Deep learning algorithms are able to classify images with remarkable accuracy by learning complex patterns from large amounts of data. By providing faster, more consistent and possibly more accurate insights to clinicians, deep learning models have the potential to completely transform the diagnosis of brain tumors.

Our goal in this study was to use deep learning to classify brain tumors from MRI images. First, we developed and refined our custom deep learning models by carefully adjusting their architectures to the specific characteristics of brain MRI data. Second, we explored the pre-trained models, neural networks that have already been trained on large image datasets, to tailor them to our particular task of classifying brain tumors. By assessing these two approaches on a collection of brain MRI pictures, we aimed to create a high-performing model that could reliably differentiate between various kinds of brain tumors and also detect their absence. Our objective was to surpass 97% classification accuracy demonstrating the potential of advanced deep learning techniques to significantly contribute to the field of medical diagnostics.

2 DATA

For model training, we utilized Brain Tumor MRI Dataset [1] from Kaggle, which contains a total of 7023 brain MRI images split across four classes: glioma tumor, meningioma tumor, pituitary tumor, and no tumor. The dataset features separate portions for training and testing with 5712 images and 1311 images respectively. Representative images of each tumor class are presented below.

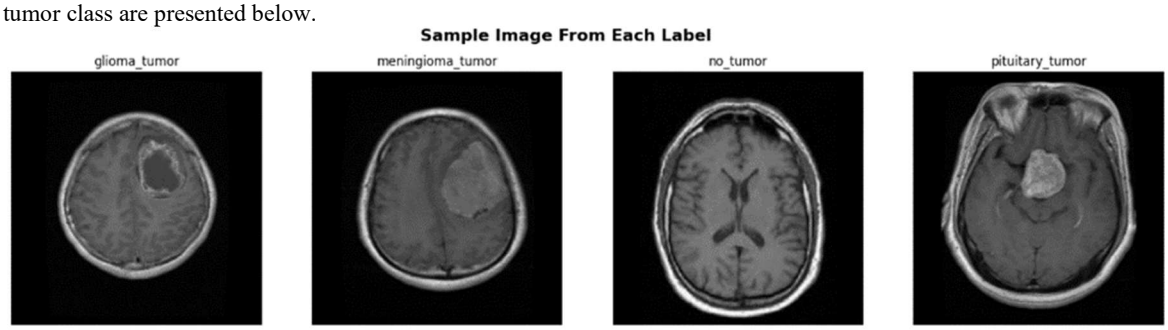


Figure 1: Sample Raw Images of Each Tumor Class

3 METHODS

In preparation for the training process, we split the data in the training portion (5712 images) with an 80/20 training-validation ratio, resulting in 4570 and 1142 images respectively. The testing portion with 1311 images was used for models' evaluation. We utilized the "image_dataset_from_directory" function from Keras to create batches of images from the directory. For the two models based on our custom architecture and a model based on EfficientNetB0, we set 64 images per batch with 180 x 180 image size. Regarding the other five models based on EfficientNetB0, we set 32 images per batch

with 224 x 224 image size. The target labels, tumor types, were encoded as integers by using the default parameter values (labels='inferred', label_mode='int'). Regarding normalization, we employed batch normalization layers in our custom architecture. The EfficientNet-B0 also utilizes batch normalization as part of its core architecture.

For our custom architecture models, we employed depthwise separable convolution layers which are more parameter-efficient than regular convolution layers. We also leveraged residual connections to facilitate the training and mitigate the vanishing gradient problem. Regarding the pre-trained model, we selected EfficientNet-B0 for its computational efficiency and accuracy.

3.1 Model 1 - Custom CNN

To improve model generalizability, we employed the following label-preserving transformations: random brightness, random contrast, gaussian noise and random translation. The input images were passed through these data augmentation layers and then rescaled to convert the pixels to [0, 1] range. After that, the model structure starts with a regular convolution layer with 32 filters. Next, the model comprises a series of 5 convolutional blocks with increasing feature depth (32, 62, 128, 256, 512). Each block consists of two batch-normalized depthwise separable convolution layers and a max pooling layer, with a residual connection around the block. After the 5 convolution blocks, we used a global average pooling layer, a dropout layer and the final dense layer with softmax activation.

For model training, we used "sparse categorical crossentropy" loss with "rmsprop" optimizer and monitored the loss and accuracy values. The model was trained for 75 epochs with "model checkpoint" callback to restore the best model. The accuracy and loss charts of training and validation data were observed. After training, the best performing model was evaluated on the test set using accuracy, precision, recall and f1-score metrics, and the performance was also assessed with the use of a confusion matrix.

The detailed architecture and parameter settings for this custom CNN model are provided in Appendix A1.

3.2 Model 2 - Custom CNN

For this model, we introduced two more transformations, random flip and random rotation. The main architecture still comprises 5 convolutional blocks like the previous model. However, we switched the depthwise separable convolution layers in the first and second blocks with regular convolution layers. Also, we changed the activation to "ReLU6" from "RELU". The model training process and evaluation procedure remained the same.

The detailed architecture and parameter settings for this custom CNN model are provided in Appendix A2.

3.3 Model 3 - EfficientNetB0

To improve model generalizability, we employed the following label-preserving transformations: random brightness, random contrast, gaussian noise and random translation. We utilized the convolutional base of EfficientNetB0 with "imagenet" weights and then attached a global average pooling layer, a dropout layer and the final dense layer with softmax activation. We tried to train the entire base without freezing any layer in this setup.

For model training, we used "sparse categorical crossentropy" loss with "rmsprop" optimizer and monitored the loss and accuracy values. The model was trained for 20 epochs with "model checkpoint" callback to restore the best model. The accuracy and loss charts of training and validation data were observed. After training, the best performing model was evaluated on the test set using accuracy, precision, recall and f1-score metrics, and the performance was also assessed with the use of a confusion matrix.

The detailed architecture and parameter settings for this EfficientNetB0 model are provided in Appendix A3.

3.4 Model 4 - EfficientNetB0

For this model, we introduced three more transformations: random flip, random rotation and random zoom. Unlike the previous model, we froze the first 100 layers of EfficientNetB0 convolutional base for stability. Also, we increased the dropout rate to 0.7 and added an additional dense layer with 256 neurons before the final dense layer. For model training, we tried to optimize the learning rate and also set up to reduce learning rate on plateau. The "early stopping" callback was also configured. The evaluation procedure remained the same.

The detailed architecture and parameter settings for this EfficientNetB0 model are provided in Appendix A4.

3.5 Model 5 - EfficientNetB0

The architecture of this model is the same as that of the previous model despite the dropout rate being reduced to 0.3. The model training process and evaluation procedure remained the same.

The detailed architecture and parameter settings for this EfficientNetB0 model are provided in Appendix A5.

3.6 Model 6 - EfficientNetB0

The architecture of this model is the same as that of the previous model despite the dropout rate being increased back to 0.7. Also, the model was trained longer to 100 epochs. The evaluation procedure remained the same.

The detailed architecture and parameter settings for this EfficientNetB0 model are provided in Appendix A6.

3.7 Model 7 - EfficientNetB0

In this model, we experimented with class weight computations to improve the performance. Also, we made some architectural changes after the convolutional base of EfficientNetB0. The base was attached to global average pooling, batch normalization, dropout (0.5 rate), dense (256 neurons), batch normalization, dropout (0.3 rate), and then the final dense layer with softmax activation. Learning rate scheduler was also introduced and the model was trained for 50 epochs. The evaluation procedure remained the same.

The detailed architecture and parameter settings for this EfficientNetB0 model are provided in Appendix A7.

3.8 Model 8 - EfficientNetB0

The architecture of this model is the same as that of the previous model. However, it was set up to unfreeze all layers in the convolution base at epoch 10. Also, we tried to experiment learning rate by using "AdamW" with a weight decay. The evaluation procedure remained the same.

The detailed architecture and parameter settings for this EfficientNetB0 model are provided in Appendix A8.

4 RESULTS

4.1 Model 1 - Custom CNN

Model 1 demonstrates strong overall performance, particularly in classifying Pituitary and No Tumor cases, with F1-scores of 97.49% and 95.79%, respectively. It also shows high precision on Glioma (96.67%) and No Tumor (98.27%) categories, indicating confidence in its predictions for those classes. However, the recall for Glioma and No Tumor is slightly lower (94.16% and 93.43%), suggesting the model occasionally fails to identify these tumors when present. The training and validation curves (Appendix A9) show consistent improvement and eventual convergence, with no signs of severe overfitting, supporting the model's generalization ability.

The model's key limitation lies in its classification of Meningioma tumors. Although the recall remains high at 94.29%, the precision drops to 86.27%, implying the model confuses other tumor types with Meningiomas more often than desired. This misclassification trend is clearly reflected in the confusion matrix (Appendix A10), which visualizes how predictions cluster and reveals that Meningioma is the most frequently confused class. These results suggest the model could benefit from class-specific enhancements, such as more representative training samples or attention mechanisms to better distinguish tumor subtypes with overlapping characteristics.

Table 1: Evaluation Metrics (Model 1 - Custom CNN)

Tumor Type	Precision	recall	f1-score
Glioma	96.67%	94.16%	95.39%
Meningioma	86.27%	94.29%	90.10%
None	98.27%	93.43%	95.79%
Pituitary	97.00%	97.98%	97.49%
Overall Accuracy: 94.81%			

4.2 Model 2 - Custom CNN

Model 2 expands upon the baseline by incorporating stronger regularization and a richer data augmentation pipeline, resulting in measurable performance improvements across tumor classes. With an overall test accuracy of 96.64%, it shows robust classification capabilities—most notably for the "No Tumor" and "Pituitary" categories, which achieve F1-scores of 98.17% and 97.86%, respectively. Glioma also benefits from the updated architecture with high precision (98.33%) and an F1-score of 96.56%. The training and validation accuracy/loss plots (Appendix A11) demonstrate stable learning and minimal overfitting, indicating the model was able to generalize well from the training set.

However, the model does exhibit a notable limitation in its handling of Meningioma cases. While the recall for Meningioma is very high at 98.19%, the lower precision of 88.89% reveals a tendency to overpredict this class—misclassifying some non-Meningioma images as Meningioma. This behavior is visually supported by the confusion matrix (Appendix A12), which highlights the misclassification pattern. Future work could address this imbalance through more class-specific augmentation, fine-grained tuning of the loss function, or calibration techniques to improve precision without sacrificing recall.

Table 2: Evaluation Metrics (Model 2 - Custom CNN)

Tumor Type	Precision	recall	f1-score
Glioma	98.33%	94.86%	96.56%
Meningioma	88.89%	98.19%	93.31%
None	99.51%	96.88%	98.17%

Pituitary	99.00%	96.74%	97.86%
Overall Accuracy: 96.64%			

4.3 Model 3 - EfficientNetB0

Model 3 achieved strong performance on the multi-class brain tumor classification task, reaching an overall test accuracy of 98.17%. Among the four classes, the model performed best in detecting the no tumor category, with a precision of 99.26%, recall of 99.75%, and F1-score of 99.50%. Similarly, it demonstrated high performance for pituitary tumors with all three metrics at 99.00%. Detection of glioma tumors was also strong, especially in terms of recall (99.31%), suggesting the model rarely misses this class.

However, the model showed relatively lower performance in identifying meningioma tumors, where recall dropped to 94.34% despite a high precision of 98.84%. This indicates the model occasionally misclassifies meningioma cases as other tumor types, which could have clinical implications. The confusion matrix (Appendix A14) supports this, showing greater overlap between meningioma and other classes compared to the rest.

Despite using dropout regularization, training and validation curves (Appendix A13) revealed signs of overfitting. This suggests that while the model generalizes well overall, it may still benefit from further regularization, more diverse data, or class-specific augmentation strategies—particularly to improve meningioma classification.

Table 3: Evaluation Metrics (Model 3 - EfficientNetB0)

Tumor Type	Precision	recall	f1-score
Glioma	96.00%	99.31%	97.63%
Meningioma	98.04%	94.34%	96.15%
None	99.26%	99.75%	99.50%
Pituitary	99.00%	99.00%	99.00%
Overall Accuracy: 98.17%			

4.4 Model 4 - EfficientNetB0

Model 4, which incorporated more aggressive data augmentation and regularization strategies, achieved an overall test accuracy of 97.19%. The model was especially strong in detecting glioma tumors, with a perfect recall of 100% and an F1-score of 98.65%, indicating it successfully captured nearly all true glioma cases. It also maintained excellent performance on the no tumor class (F1-score 99.26%) and the pituitary tumor class (F1-score 97.24%), though the slightly lower recall (94.92%) for pituitary tumors suggests a few were misclassified.

The main area of weakness remained meningioma tumor classification, with a precision of 94.12% and recall of 96.97%—an improvement over Model 3, but still comparatively lower than other classes. The confusion matrix (Appendix A16) reflected this, showing more frequent misclassifications involving meningioma. Despite enhancements like dropout and dynamic learning rate scheduling, the training/validation curves (Appendix A15) suggest mild overfitting persists. This implies further improvements may depend on addressing class imbalance or introducing domain-specific features.

Table 4: Evaluation Metrics (Model 4 - EfficientNetB0)

Tumor Type	Precision	recall	f1-score
Glioma	97.33%	100%	98.65%
Meningioma	94.12%	96.97%	95.52%
None	99.51%	99.02%	99.26%
Pituitary	99.67%	94.92%	97.24
Overall Accuracy: 97.19%			

4.5 Model 5 - EfficientNetB0

Model 5 maintained a high level of performance with an overall test accuracy of 97.10%, continuing to leverage transfer learning and extensive augmentation. It excelled at detecting no tumor cases, achieving an F1-score of 99.63%, and performed consistently well for pituitary tumors (F1-score 97.24%) with perfect precision (100%). Glioma tumors were also identified reliably, with a strong recall of 99.64%, though precision dropped slightly to 93.00%, indicating a few false positives.

The most notable challenge remained in classifying meningioma tumors, where the model recorded a precision of 95.42% and recall of 94.19%, resulting in the lowest F1-score (94.81%) among the four classes. Compared to previous models, performance gains were modest, and the slight reduction in dropout (from 0.7 to 0.3) may have increased sensitivity to noise in certain cases. The training/validation curves (Appendix A17) indicated stable convergence, but mild overfitting persisted. As visualized in the confusion matrix (Appendix A18), meningioma remained the class most prone to misclassification, underscoring the need for further refinement or additional training data to address class-specific weaknesses.

Table 5: Evaluation Metrics (Model 5 - EfficientNetB0)

Tumor Type	Precision	recall	f1-score
Glioma	93.00%	99.64%	96.21%
Meningioma	95.42%	94.19%	94.81%
None	99.51%	99.75%	99.63%
Pituitary	100%	94.64%	97.24%
Overall Accuracy: 97.10%			

4.6 Model 6 - EfficientNetB0

Model 6 achieved the highest overall performance in the study, reaching an accuracy of 98.17% on the test set. It demonstrated strong, balanced performance across all four classes. Glioma tumors were classified with perfect consistency (F1-score 98.33% for both precision and recall), while no tumor cases remained the most reliably detected (F1-score

99.38%). Pituitary tumors also showed high performance, with an F1-score of 98.35%, reflecting both strong precision (99.33%) and high recall (97.39%).

Meningioma tumors*, previously a recurring challenge, saw improved classification with an F1-score of 96.20%, indicating Model 6's enhanced ability to distinguish this class after longer training and increased regularization. The extended training regime (up to 100 epochs with increased early stopping patience) allowed the model to refine decision boundaries without overfitting, as evidenced by stable convergence in the training/validation curves (Appendix A19). The confusion matrix (Appendix A20) further confirmed this balanced performance, showing reduced misclassifications across all classes. However, while this model reduced prior weaknesses, its effectiveness on diverse external datasets remains untested, highlighting the ongoing limitation of generalizability in this domain.

Table 6: Evaluation Metrics (Model 6 - EfficientNetB0)

Tumor Type	Precision	recall	f1-score
Glioma	98.33%	98.33%	98.33%
Meningioma	95.10%	97.32%	96.20%
None	99.51%	99.26%	99.38%
Pituitary	99.33%	97.39%	98.35%
Overall Accuracy: 98.17%			

4.7 Model 7 - EfficientNetB0

Model 7 introduced a more sophisticated training strategy, combining fine-tuning, class imbalance correction, and improved data pipeline efficiency. It achieved an overall test accuracy of 97.70%, reflecting well-rounded performance across all four tumor classes. The model excelled in detecting no tumor cases with perfect precision and recall (99.51% F1-score), and pituitary tumors were also identified accurately (F1-score 97.70%), with minimal misclassifications.

Classification of glioma tumors remained strong, particularly in recall (99.65%), yielding an F1-score of 97.44%, while meningioma tumors showed improved balance between precision (96.08%) and recall (95.15%), resulting in an F1-score of 95.61%. The inclusion of class weighting addressed prior imbalances, contributing to more equitable treatment of underrepresented classes. Training and validation curves (Appendix A21) demonstrated stable convergence, aided by a warmup learning rate schedule and multiple callbacks. The confusion matrix (Appendix A22) confirmed reduced confusion across classes, though minor misclassification persisted, particularly with meningioma.

Table 7: Evaluation Metrics (Model 7 - EfficientNetB0)

Tumor Type	Precision	recall	f1-score
Glioma	95.33%	99.65%	97.44%
Meningioma	96.08%	95.15%	95.61%
None	99.51%	99.51%	99.51%

Pituitary	99.33%	96.13%	97.70%
Overall Accuracy: 97.70%			

4.8 Model 8 - EfficientNetB0

Model 8 delivered one of the most advanced performances of the study, achieving an overall test accuracy of 98.02%. Building upon the EfficientNetB0 architecture, this version introduced full fine-tuning with delayed unfreezing, SpatialDropout2D, AdamW optimization with weight decay, and a warmup-based learning rate scheduler. These enhancements collectively promoted deeper feature adaptation while preventing overfitting.

The model performed consistently across all tumor classes. Glioma tumors were identified with an F1-score of 98.16%, and pituitary tumors reached a precision of 99.67% with an F1-score of 98.03%. The no tumor class maintained exceptional performance (F1-score 99.26%), while meningioma tumors achieved notable improvement, reaching a recall of 97.97% and F1-score of 96.17%—indicating better handling of previously challenging cases.

Training curves (Appendix A23) showed stable convergence and no signs of early overfitting, aided by warmup scheduling and the UnfreezeCallback. Class weighting addressed dataset imbalance, and the confusion matrix confirmed strong predictive performance with minimal class confusion. These results highlight the benefit of combining aggressive regularization with staged full fine-tuning for high-fidelity medical image classification.

Table 8: Evaluation Metrics (Model 8 - EfficientNetB0)

Tumor Type	Precision	recall	f1-score
Glioma	98.00%	98.33%	98.16%
Meningioma	94.44%	97.97%	96.17%
None	99.51%	99.02%	99.26%
Pituitary	99.67%	96.45%	98.03%
Overall Accuracy: 98.02%			

5 LIMITATIONS

Although the fine-tuned EfficientNetB0 models demonstrated excellent accuracy on the Brain Tumor MRI Dataset, this study has certain limitations. Despite its substantial size, the dataset comprises only four tumor types and one image acquisition method. To evaluate the models' robustness and generalizability in real-world clinical settings, we would need larger and more diverse datasets. Also, we would need to assess aspects other than accuracy such as interpretability and computational efficiency for deployment.

6 FUTURE WORK

Future work could explore several promising directions. First, expanding the dataset with more diverse MRI scans—including multi-center, multi-modal data—would increase robustness and real-world applicability. Exploring ensemble methods that combine multiple high-performing models may further boost accuracy while providing uncertainty estimates.

With continued refinement, these models can form the backbone of reliable, AI-assisted diagnostic tools for brain tumor detection in medical imaging workflows.

7 CONCLUSION

This study demonstrated the progressive improvement of brain tumor classification performance through successive enhancements to the EfficientNetB0 architecture. Starting from a baseline with transfer learning and conventional data augmentation, each model iteration incorporated increasingly advanced techniques—including aggressive regularization, dynamic learning rate scheduling, class imbalance correction, and full fine-tuning. The best-performing models, particularly Models 6 and 8, achieved over 98% overall accuracy, with consistently high F1-scores across all four tumor classes. Notably, Model 8's fine-tuning strategy and use of the AdamW optimizer contributed to superior generalization and more stable convergence, effectively minimizing class confusion while preserving high sensitivity and precision for each tumor category.

REFERENCES

[1] Nickparvar, M. (2021, September). Brain Tumor MRI Dataset, Version 1. Retrieved April 30, 2025 from https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset.

[2] Mingxing Tan and Quoc V. Le. 2019. Rethinking Model Scaling for Convolutional Neural Networks. In Proceedings of the International Conference on Machine Learning (ICML). Long Beach, California, USA, June 09–15, 2019 (ICML'19), 10 pages. PMLR, Vol. 97.

A APPENDICES

A.1 Model 1 - Custom CNN

```
# Label-preserving transformations
data_augmentation = keras.Sequential([
   layers.RandomBrightness(0.1),
   layers.RandomContrast(0.1),
   layers.GaussianNoise(0.05),
   layers.RandomTranslation(0.1, 0.1),])

# Build model structure
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=5, use_bias=False)(x)

for size in [32, 64, 128, 256, 512]:
   residual = x

   x = layers.BatchNormalization()(x)
   x = layers.Activation("relu")(x)
   x = layers.SeparableConv2D(size, 3, padding="same", use_bias=False)(x)
```

```
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)
    x = layers.SeparableConv2D(size, 3, padding="same", use_bias=False)(x)

    x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

    residual = layers.Conv2D(size, 1, strides=2, padding="same", use_bias=False)(residual)

    x = layers.add([x, residual])

x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(4, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

# Configure loss function, optimizer and monitoring metric
model.compile(loss="sparse_categorical_crossentropy",
            optimizer="rmsprop",
            metrics=["accuracy"])

# Create checkpoints to save the best model
callbacks = [keras.callbacks.ModelCheckpoint(filepath="custom_model.keras",
                                    save_best_only=True,
                                    monitor="val_loss")]

# Train the model
history = model.fit(train_dataset,
                    epochs=75,
                    validation_data=validation_dataset,
                    callbacks=callbacks)
```

A.2 Model 2 - Custom CNN

```
# Label-preserving transformations (improved)
data_augmentation = keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.05),
    layers.RandomBrightness(0.1),
    layers.RandomContrast(0.1),
    layers.GaussianNoise(0.05),
    layers.RandomTranslation(0.1, 0.1),
])
```

```
# Build improved model structure
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=7, strides=2, padding="same", use_bias=False)(x)

for i, size in enumerate([32, 64, 128, 256, 512]):
    residual = x

    x = layers.BatchNormalization()(x)
    x = layers.ReLU(max_value=6)(x)  # ReLU6

    if i < 2:
        x = layers.Conv2D(size, 3, padding="same", use_bias=False,
                          kernel_regularizer=keras.regularizers.l2(1e-4))(x)
    else:
        x = layers.SeparableConv2D(size, 3, padding="same", use_bias=False,
                                   depthwise_regularizer=keras.regularizers.l2(1e-4),
                                   pointwise_regularizer=keras.regularizers.l2(1e-4))(x)

    x = layers.BatchNormalization()(x)
    x = layers.ReLU(max_value=6)(x)  # ReLU6 again

    if i < 2:
        x = layers.Conv2D(size, 3, padding="same", use_bias=False)(x)
    else:
        x = layers.SeparableConv2D(size, 3, padding="same", use_bias=False)(x)

    x = layers.MaxPooling2D(3, strides=2, padding="same")(x)
    residual = layers.Conv2D(size, 1, strides=2, padding="same", use_bias=False)(residual)
    x = layers.add([x, residual])

x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(4, activation="softmax")(x)

model = keras.Model(inputs=inputs, outputs=outputs)

# Configure loss function, optimizer and monitoring metric
model.compile(loss="sparse_categorical_crossentropy",
```

```
        optimizer="rmsprop",
        metrics=["accuracy"])

# Create checkpoints to save the best model
callbacks = [keras.callbacks.ModelCheckpoint(filepath="custom_model.keras",
                                     save_best_only=True,
                                     monitor="val_loss")]

# Train the model
history = model.fit(train_dataset,
                epochs=75,
                validation_data=validation_dataset,
                callbacks=callbacks)
```

A.3 Model 3 - EfficientNetB0

```
# Label-preserving transformations
data_augmentation = keras.Sequential([
   layers.RandomBrightness(0.1),
   layers.RandomContrast(0.1),
   layers.GaussianNoise(0.05),
   layers.RandomTranslation(0.1, 0.1),])

# Instantiate convolutional base of EfficientNetB0
conv_base = keras.applications.EfficientNetB0(weights="imagenet",
                                      include_top=False)

# Build model structure
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = conv_base(x)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(4, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

# Configure loss function, optimizer and monitoring metric
model.compile(loss="sparse_categorical_crossentropy",
        optimizer="rmsprop",
        metrics=["accuracy"])

# Create checkpoints to save the best model
```

```
callbacks = [keras.callbacks.ModelCheckpoint(filepath="efficientnetb0.keras",
                                             save_best_only=True,
                                             monitor="val_loss")]

# Train the model
history = model.fit(train_dataset,
                    epochs=20,
                    validation_data=validation_dataset,
                    callbacks=callbacks)
```

A.4 Model 4 - EfficientNetB0

```
# Label-preserving transformations
data_augmentation = keras.Sequential([
   layers.RandomBrightness(0.1),
   layers.RandomContrast(0.1),
   layers.GaussianNoise(0.05),
   layers.RandomTranslation(0.1, 0.1),
   layers.RandomFlip("horizontal_and_vertical"),
   layers.RandomRotation(0.2),
   layers.RandomZoom(0.2),
])

# Instantiate convolutional base of EfficientNetB0
conv_base = keras.applications.EfficientNetB0(
   weights="imagenet", include_top=False)

# Freeze first 100 layers for stability
for layer in conv_base.layers[:100]:
   layer.trainable = False

# Build model structure
model = keras.Sequential([
   layers.Input(shape=(224, 224, 3)),
   data_augmentation,
   conv_base,
   layers.GlobalAveragePooling2D(),
   layers.Dropout(0.7),
   layers.Dense(256, activation='relu'),
   layers.Dense(4, activation='softmax')
])
```

```
# Configure loss function, optimizer and monitoring metric
optimizer = keras.optimizers.RMSprop(learning_rate=1e-4)
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer=optimizer,
    metrics=["accuracy"]
)

# Configure checkpoint, early stopping, reduce learning rate callbacks
callbacks = [
    keras.callbacks.ModelCheckpoint(filepath="efficientnetb0.keras", save_best_only=True, monitor="val_loss"),
    keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True),
    keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3)
]

# Train the model
history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=20,
    callbacks=callbacks
)
```

A.5 Model 5 - EfficientNetB0

```
# Label-preserving transformations
data_augmentation = keras.Sequential([
    layers.RandomBrightness(0.1),
    layers.RandomContrast(0.1),
    layers.GaussianNoise(0.05),
    layers.RandomTranslation(0.1, 0.1),
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
    layers.RandomZoom(0.2),
])

# Instantiate convolutional base of EfficientNetB0
conv_base = keras.applications.EfficientNetB0(
    weights="imagenet", include_top=False)

# Freeze first 100 layers for stability
for layer in conv_base.layers[:100]:
```

```
    layer.trainable = False

# Build model structure
model = keras.Sequential([
    layers.Input(shape=(224, 224, 3)),
    data_augmentation,
    conv_base,
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.3),
    layers.Dense(256, activation='relu'),
    layers.Dense(4, activation='softmax')
])

# Configure loss function, optimizer and monitoring metric
optimizer = keras.optimizers.RMSprop(learning_rate=1e-4)
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer=optimizer,
    metrics=["accuracy"]
)

# Configure checkpoint, early stopping, reduce learning rate callbacks
callbacks = [
    keras.callbacks.ModelCheckpoint(filepath="efficientnetb0.keras", save_best_only=True, monitor="val_loss"),
    keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True),
    keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3)
]

# Train Model
history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=20,
    callbacks=callbacks
)
```

A.6 Model 6 - EfficientNetB0

```
# Label-preserving transformations
data_augmentation = keras.Sequential([
    layers.RandomBrightness(0.1),
    layers.RandomContrast(0.1),
```

```
    layers.GaussianNoise(0.05),
    layers.RandomTranslation(0.1, 0.1),
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
    layers.RandomZoom(0.2),
])

# Instantiate convolutional base of EfficientNetB0
conv_base = keras.applications.EfficientNetB0(
    weights="imagenet", include_top=False)

# Freeze first 100 layers for stability
for layer in conv_base.layers[:100]:
    layer.trainable = False

# Build model structure
model = keras.Sequential([
    layers.Input(shape=(224, 224, 3)),
    data_augmentation,
    conv_base,
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.7),
    layers.Dense(256, activation='relu'),
    layers.Dense(4, activation='softmax')
])

# Configure loss function, optimizer and monitoring metric
optimizer = keras.optimizers.RMSprop(learning_rate=1e-4)
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer=optimizer,
    metrics=["accuracy"]
)

# Configure checkpoint, early stopping, reduce learning rate callbacks
callbacks = [
    keras.callbacks.ModelCheckpoint(filepath="efficientnetb0.keras", save_best_only=True, monitor="val_loss"),
    keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3)
]
```

```
# Train Model
history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=100,
    callbacks=callbacks
)
```

A.7 Model 7 - EfficientNetB0

```
from sklearn.utils.class_weight import compute_class_weight
import tensorflow as tf

AUTOTUNE = tf.data.AUTOTUNE

# Prefetch for performance
train_dataset = train_dataset.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.cache().prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.cache().prefetch(buffer_size=AUTOTUNE)

# Estimate class weights
y_train = np.concatenate([y for _, y in train_dataset], axis=0)
class_weights = dict(enumerate(compute_class_weight(class_weight='balanced', classes=np.unique(y_train), y=y_train)))

# Data Augmentation
data_augmentation = keras.Sequential([
    layers.RandomBrightness(0.1),
    layers.RandomContrast(0.1),
    layers.GaussianNoise(0.05),
    layers.RandomTranslation(0.1, 0.1),
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
    layers.RandomZoom(0.2),
])

# Base Model: EfficientNetB0
base_model = keras.applications.EfficientNetB0(
    weights='imagenet',
    include_top=False,
    input_shape=(224, 224, 3)
)
base_model.trainable = True
```

```
for layer in base_model.layers[:100]:
    layer.trainable = False  # freeze first 100 layers

# Functional Model
inputs = keras.Input(shape=(224, 224, 3))
x = data_augmentation(inputs)
x = base_model(x, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.3)(x)
outputs = layers.Dense(4, activation='softmax')(x)
model = keras.Model(inputs, outputs)

# === Compile ===
optimizer = keras.optimizers.RMSprop(learning_rate=1e-4)
model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# Learning Rate Scheduler
def warmup_schedule(epoch, lr):
    if epoch < 3:
        return lr + 1e-5
    return lr

# Callbacks
callbacks = [
    keras.callbacks.ModelCheckpoint("efficientnetb0.keras", save_best_only=True, monitor="val_loss"),
    keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3),
    keras.callbacks.LearningRateScheduler(warmup_schedule)
]

# Training
history = model.fit(
    train_dataset,
```

```
    validation_data=validation_dataset,
    epochs=50,
    callbacks=callbacks,
    class_weight=class_weights
)
```

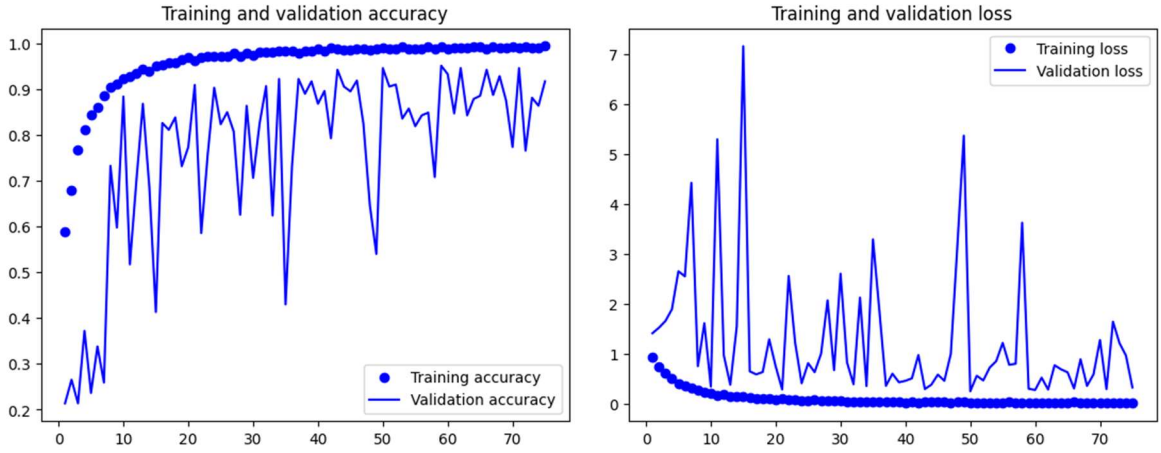
A.8 Model 8 - EfficientNetB0

```
from sklearn.utils.class_weight import compute_class_weight
import tensorflow as tf
from tensorflow.keras.applications.efficientnet import preprocess_input

AUTOTUNE = tf.data.AUTOTUNE

# Prefetch for performance
train_dataset = train_dataset.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.cache().prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.cache().prefetch(buffer_size=AUTOTUNE)

# Estimate class weights
y_train = np.concatenate([y for _, y in train_dataset], axis=0)
class_weights = dict(enumerate(compute_class_weight(class_weight='balanced', classes=np.unique(y_train), y=y_train)))

# Data Augmentation
data_augmentation = keras.Sequential([
    layers.RandomBrightness(0.1),
    layers.RandomContrast(0.1),
    layers.GaussianNoise(0.05),
    layers.RandomTranslation(0.1, 0.1),
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
    layers.RandomZoom(0.2),
], name="data_augmentation")

# Base Model
base_model = keras.applications.EfficientNetB0(
    weights='imagenet',
    include_top=False,
    input_shape=(224, 224, 3)
)
base_model.trainable = True
for layer in base_model.layers[:100]:
```

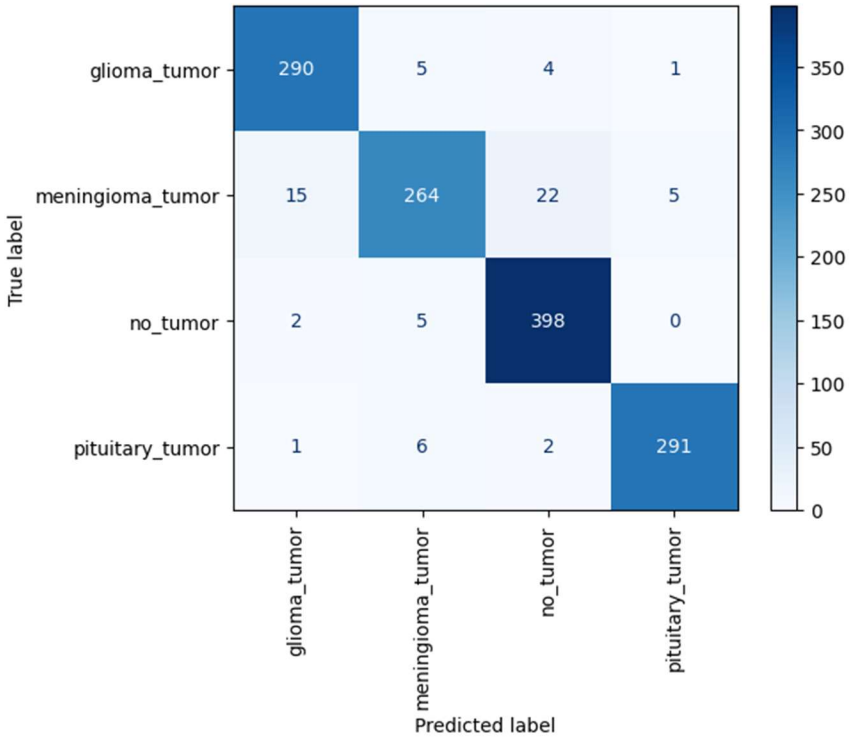
```
    layer.trainable = False  # Freeze early layers

# Functional Model
inputs = keras.Input(shape=(224, 224, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = layers.SpatialDropout2D(0.2)(x)
x = layers.GlobalAveragePooling2D()(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.3)(x)
outputs = layers.Dense(4, activation='softmax')(x)
model = keras.Model(inputs, outputs)

# Optimizer & Loss
optimizer = tf.keras.optimizers.AdamW(learning_rate=1e-4, weight_decay=1e-5)
loss_fn = keras.losses.SparseCategoricalCrossentropy()
model.compile(optimizer=optimizer, loss=loss_fn, metrics=['accuracy'])

# To schedule learning rate
def warmup_schedule(epoch, lr):
    if epoch < 3:
        return lr + 1e-5
    return lr

# To unfreeze entire base on epoch 10
class UnfreezeCallback(tf.keras.callbacks.Callback):
    def on_epoch_begin(self, epoch, logs=None):
        if epoch == 10:
            print("Unfreezing full EfficientNetB0")
            for layer in base_model.layers:
                layer.trainable = True

# Callbacks
callbacks = [
    keras.callbacks.ModelCheckpoint("efficientnetb0.keras", save_best_only=True, monitor="val_loss"),
    keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3),
```

```
    keras.callbacks.LearningRateScheduler(warmup_schedule),
    UnfreezeCallback()
]

# Training
history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=50,
    callbacks=callbacks,
    class_weight=class_weights
)
```

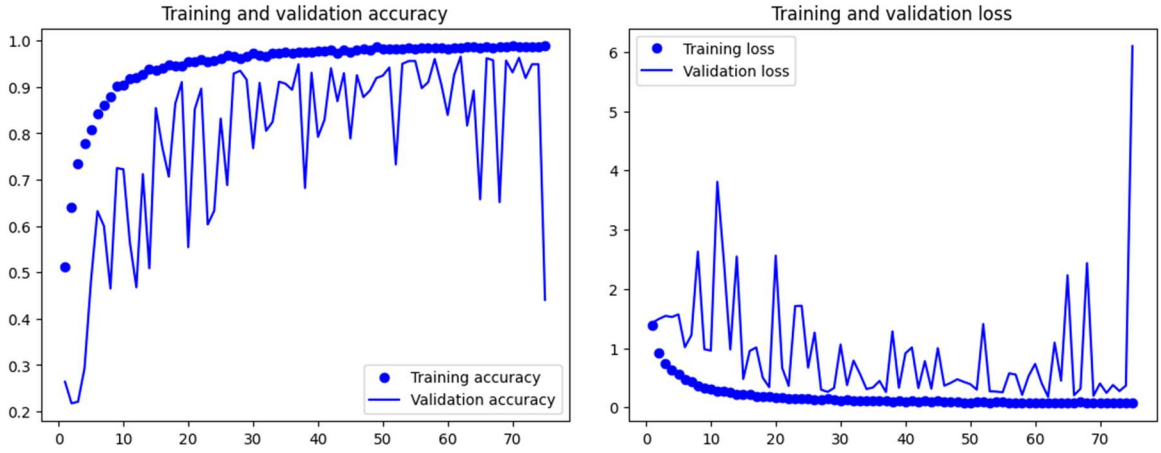
A.9 Training and Validation Accuracy and Loss Curves (Model 1 - Custom CNN)



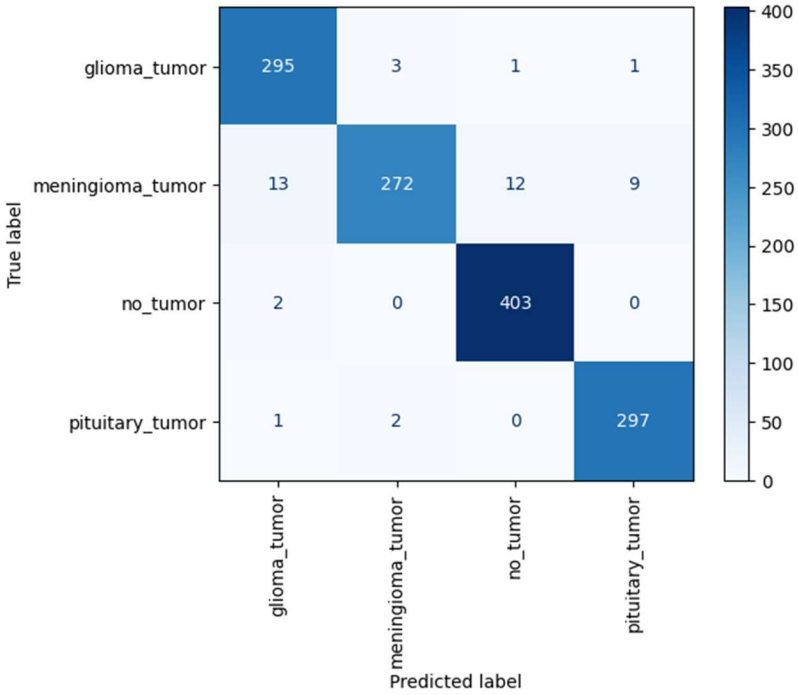
A.10 Confusion Matrix (Model 1 - Custom CNN)



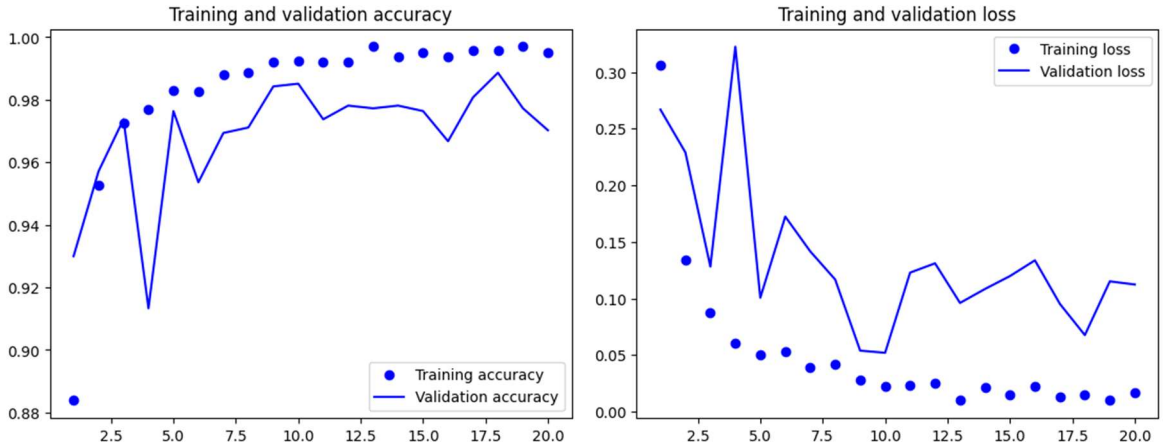
A.11 Training and Validation Accuracy and Loss Curves (Model 2 - Custom CNN)



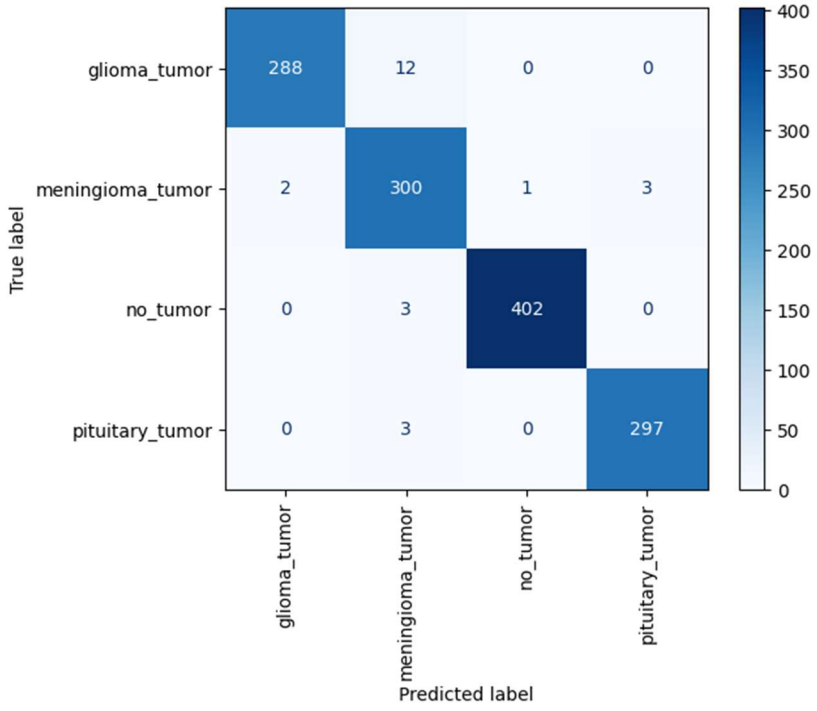
A.12 Confusion Matrix (Model 2 - Custom CNN)



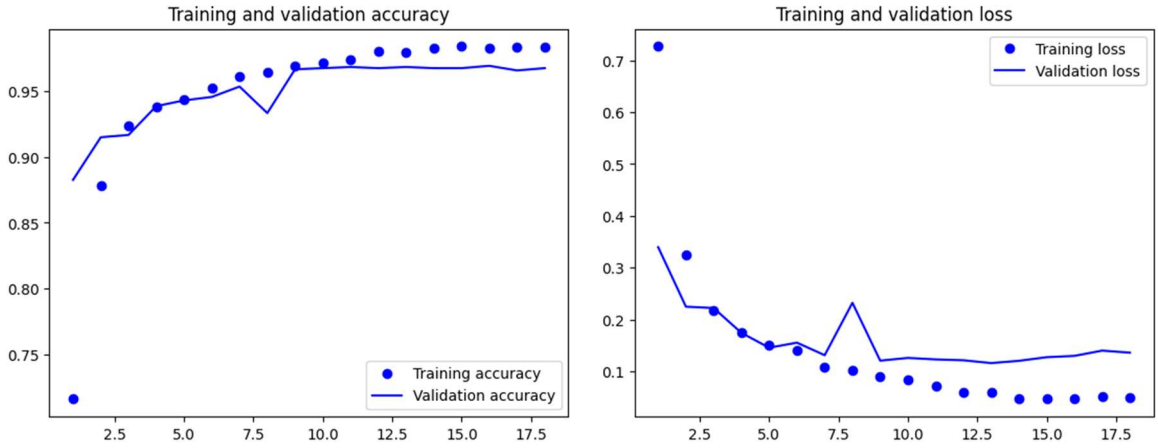
A.13 Training and Validation Accuracy and Loss Curves (Model 3 - EfficientNetB0)



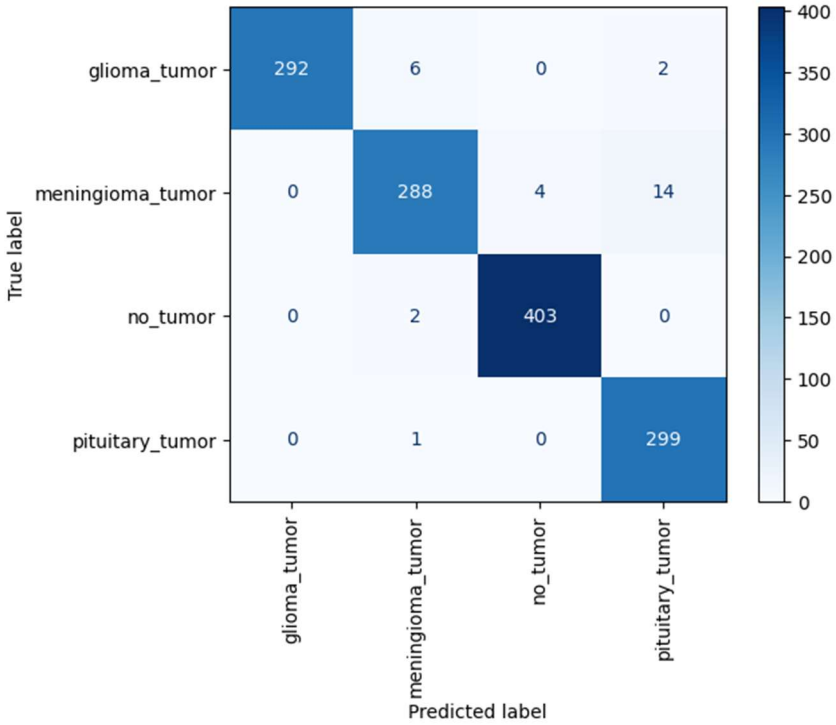
A.14 Confusion Matrix (Model 3 - EfficientNetB0)



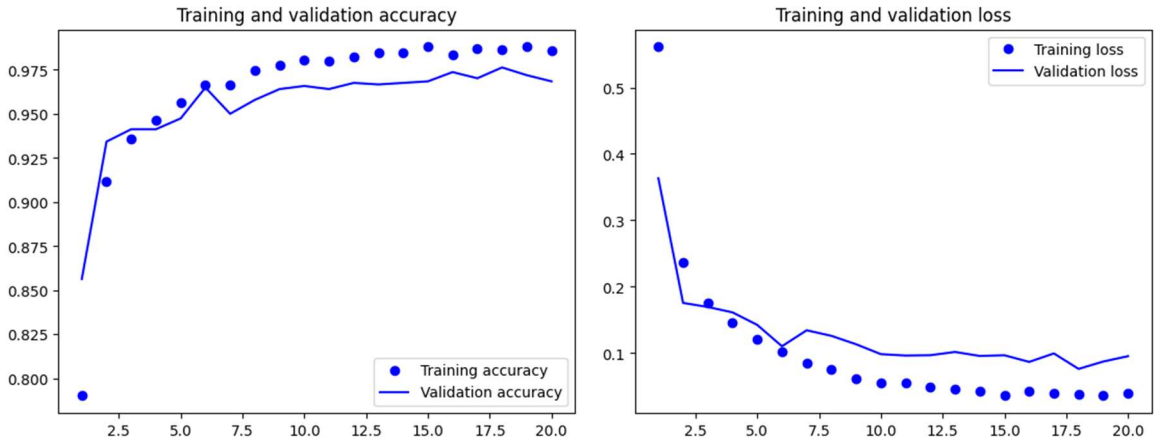
A.15 Training and Validation Accuracy and Loss Curves (Model 4 - EfficientNetB0)



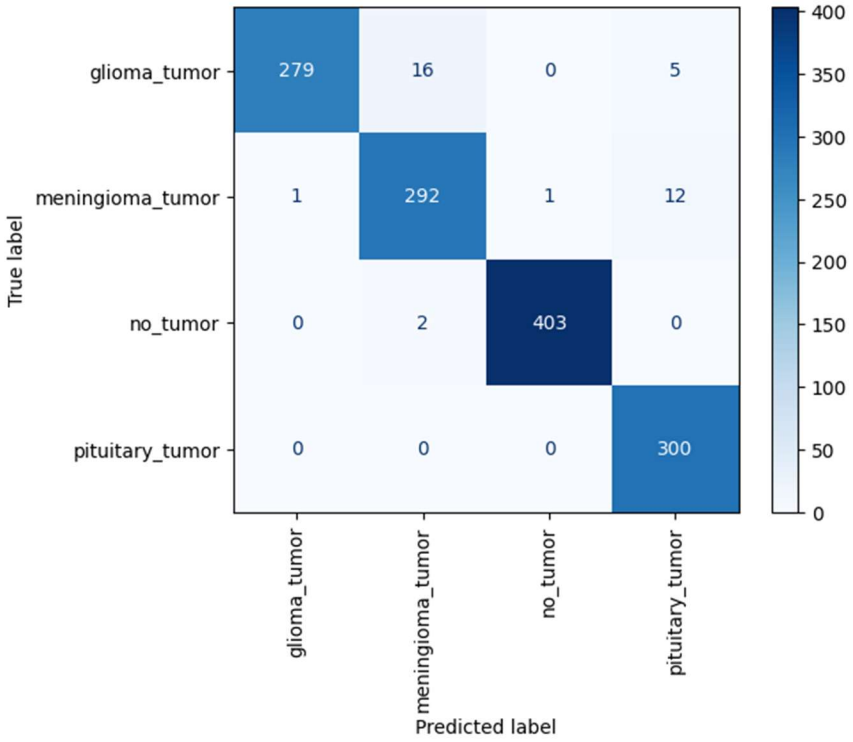
A.16 Confusion Matrix (Model 4 - EfficientNetB0)



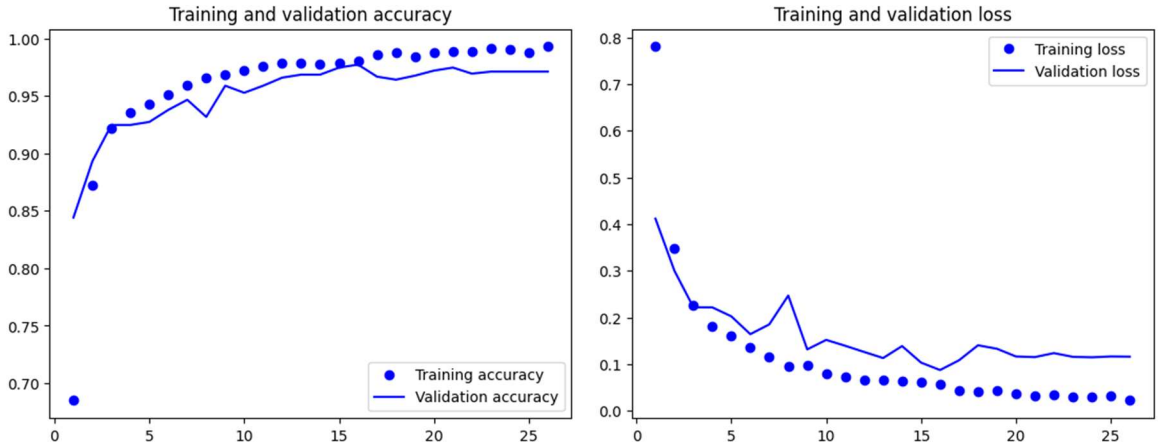
A.17 Training and Validation Accuracy and Loss Curves (Model 5 - EfficientNetB0)



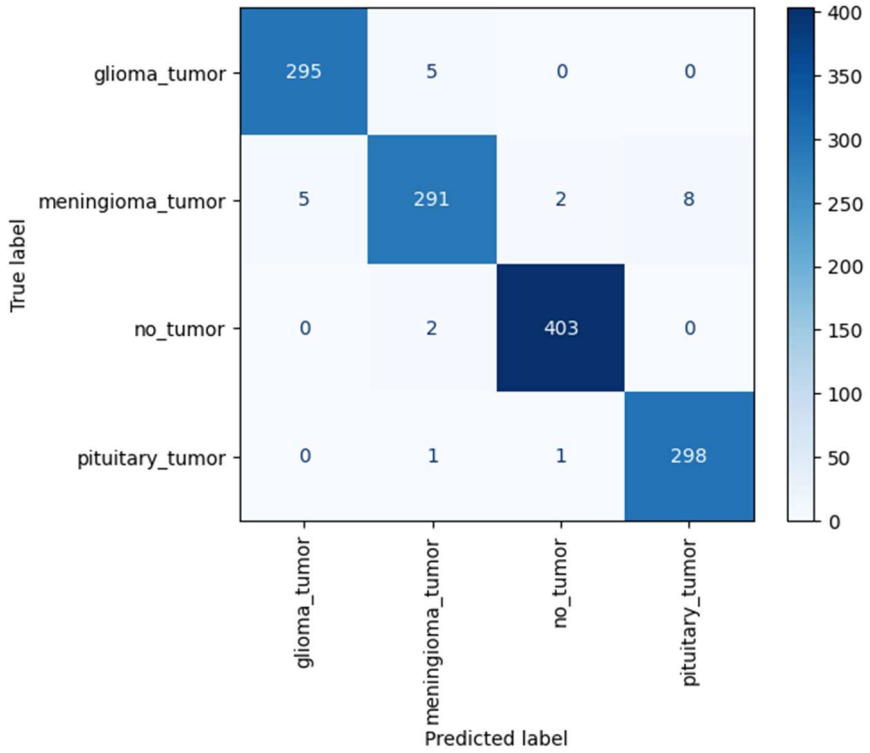
A.18 Confusion Matrix (Model 5 - EfficientNetB0)



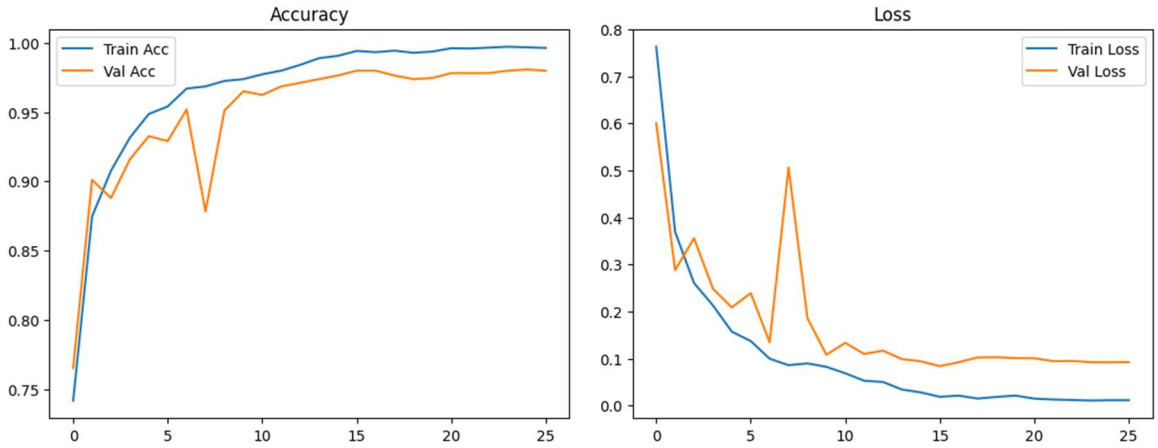
A.19 Training and Validation Accuracy and Loss Curves (Model 6 - EfficientNetB0)



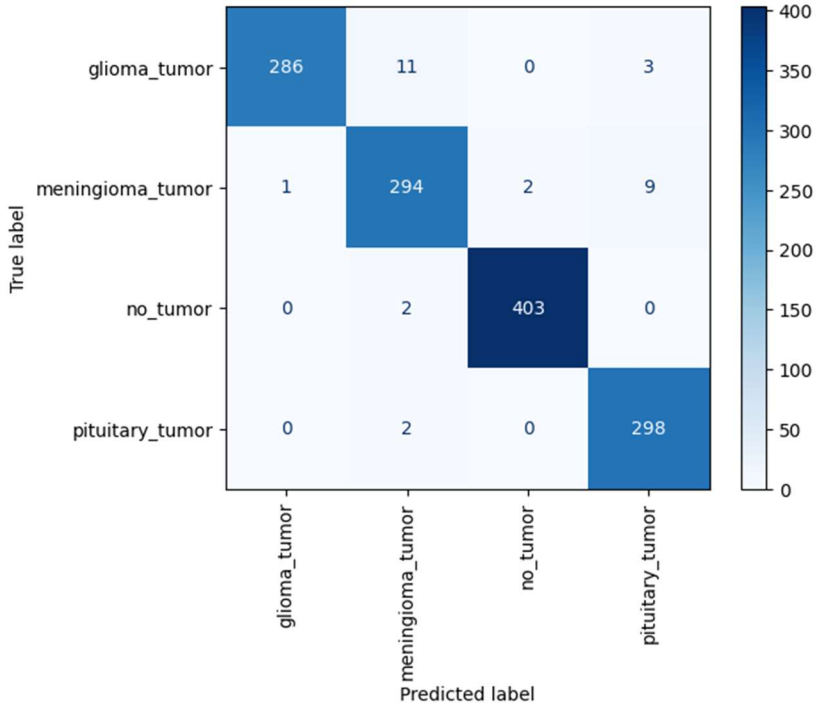
A.20 Confusion Matrix (Model 6 - EfficientNetB0)



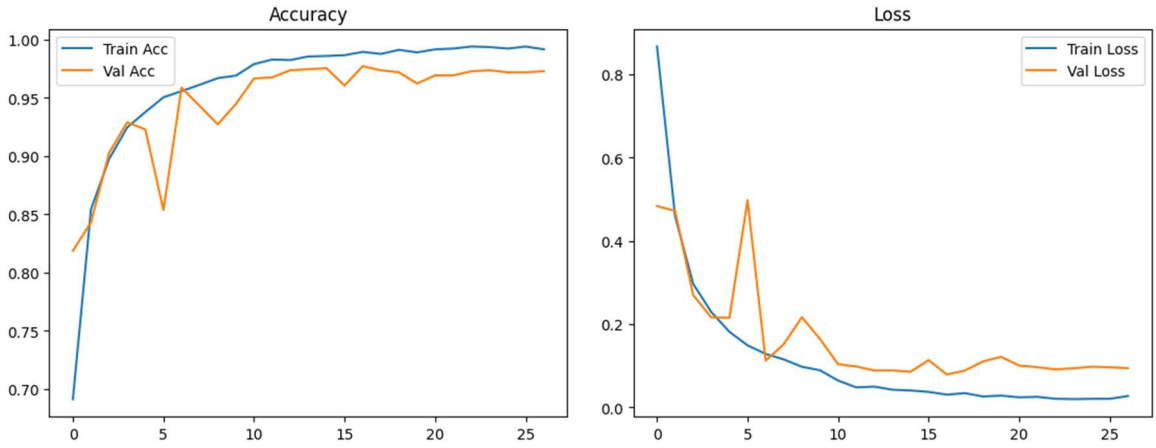
A.21 Training and Validation Accuracy and Loss Curves (Model 7 - EfficientNetB0)



A.22 Confusion Matrix (Model 7 - EfficientNetB0)



A.23 Training and Validation Accuracy and Loss Curves (Model 8 - EfficientNetB0)



A.24 Confusion Matrix (Model 8 - EfficientNetB0)