# **GlioBLAST:** Establishing Prognosis and Targeted Therapy for Glioblastoma by Applying Convolutional Neural Networks to Detect Histological Features, Molecular Subtypes, MGMT Methylation, and EGFR Amplification from Brain-Biopsy Whole-Slide Images

Bhushan Mohanraj

The Lawrenceville School, Lawrenceville, New Jersey

# Table of Contents

**Abstract**

Glioblastoma is the most aggressive and deadly malignant brain tumor, having a five-year survival rate of only 7% and accounting for 48% of all primary malignant brain tumors. The poor prognosis associated with most glioblastoma diagnoses results from heterogeneity between and within tumors. However, early detection and classification can significantly improve patient prognosis. Diagnosis and prognosis extensively rely on identifying tumor features (like pseudopalisading necrosis) and determining genetic biomarkers (like MGMT gene methylation and EGFR gene amplification) from biopsy imagery. Yet, manual image assessment requires days for diagnosis and encounters variability between pathologists. Artificial intelligence innovations in healthcare, like deep learning and computer vision, can address these constraints with accurate medical image analysis. Artificial intelligence has been applied to modalities like CT scans and MRI images, but few studies explore histopathological image analysis, even though histopathological diagnosis is the gold-standard classification method. I developed GlioBLAST, a novel neural-network system to identify tumor features and genetic biomarkers from histopathological whole-slide images (WSIs) of brain-biopsy tissue. With UNet and convolutional neural networks, applying transfer learning and fine-tuning, my models achieved 98% accuracy in classifying molecular subtypes, 94% accuracy in detecting MGMT methylation, and 97% accuracy in detecting EGFR amplification. I also developed a web application where medical professionals can upload WSIs and obtain the corresponding heatmaps of predictions for tumor features, molecular subtypes, and genetic factors. The application can perform tumor segmentation of a WSI in under 20 minutes and create molecular subtypes, MGMT gene methylation, and EGFR gene amplification heatmaps in under 10 minutes. The automated GlioBLAST system provides reliable and accurate glioblastoma detection to predict prognosis and potentially save lives.

# 1 Background

Glioblastoma is the most common brain cancer occurring in adults. The disease derives from astrocytoma, a cancer that begins in astrocytes, cells that support neurons in the central nervous system (brain and spinal cord) [1]. Glioblastoma accounts for almost half of all malignant brain tumors, and the disease is extremely aggressive with very poor prognosis and few survivors: the median survival is under two years and almost no patients survive beyond five years [2].

Patients with glioblastoma often report to the hospital with symptoms like headaches, weakness, and numbness [3]. Medical professionals then typically perform physical and neurological exams to diagnose the condition and conduct scans to determine the potential location and size of the tumor [3]. However, while exams and scans can indicate and locate brain tumors, biopsies (removal of small tumor samples) and subsequent pathologist examinations are critical to identify and classify glioblastoma with certainty [4].

Typical modern glioblastoma care begins with a resection surgery, which removes the primary cancerous tissue [5]. Surgical removal of the primary tumor is followed by local radiotherapy (radiation to kill cancerous cells) and chemotherapy (drugs to diminish or kill cancerous cells), which prevent metastatic spread of glioblastoma to adjacent brain tissue [5]. However, glioblastoma does not typically spread beyond the brain and spinal cord [5].

Recent advances in cancer care have focused heavily on therapy targeting specific genetic characteristics of disease, and this approach could also apply to glioblastoma [6]. Glioblastoma often exhibits certain genetic mutations and alterations that medical professionals could utilize to improve specific care [6].

## 1.1 Segmentation of Tumor Features

Pathologists utilize histological-tumor-feature analysis for comprehensively diagnosing and classifying glioblastoma and planning patient therapy. The major tumor features for glioblastoma include the leading edge (the outer boundary of the tumor with few tumor cells), the infiltrating tumor (a denser area with 10 to 20 tumor cells for every 100 normal cells), the cellular tumor (the tumor core with almost no normal cells), and necrosis regions (areas within the tumor core with dying or dead cells that have fragmented nuclei) [7].

These histological tumor features are determined by viewing tissue samples from tumor biopsies under microscopes [8]. Pathologists then search for characteristic histological tumor features like microvascular proliferation to diagnose glioblastoma rather than a lower brain cancer [9]. Biopsy tissue examination for determining histological tumor features also indicates tumor aggressiveness, aiding medical professionals when determining prognosis and therapy [10].

## 1.2 Classification of Molecular Subtypes

Glioblastoma is commonly classified according to molecular subtype. Molecular subtyping classifies cancers into clusters with similar characteristics according to biomarkers like informative genes [11]. The Cancer Genome Atlas (TCGA) network in 2010 identified four

molecular subtypes of glioblastoma with genomic analysis (classical, mesenchymal, proneural and neural) [12].[1] Tumors from these subtypes each exhibit certain genetic characteristics [12].

- The classical subtype primarily exhibits EGFR amplification, which encourages tumor growth by stimulating cell proliferation, and chromosome 10 loss [12].
- The mesenchymal subtype notably affects and lowers expression of the NF1 gene on chromosome 17 [12], which encodes the neurofibromin protein that suppresses tumor growth by preventing rapid cell division [15].
- The proneural subtype uniquely alters and amplifies the PDGFRA gene [12], which encodes a receptor protein that stimulates cell growth (like the EGFR gene) [16].

Molecular subtype determination for glioblastoma holds great prognostic value, since subtype clusters exhibit different outcomes and survival rates. For example, the proneural subtype, which occurs more commonly for younger patients, generally corresponds to better prognosis and survival [12]. Subtype determination also holds therapeutic value, since treatment effectiveness varies between subtypes. For example, aggressive treatment (chemotherapy and radiotherapy together) after resection surgery greatly improves survival for the classical and mesenchymal subtypes and potentially improves survival for the neural subtype [12]. However, aggressive treatment shows no statistically significant improvement for survival for the proneural subtype.

## 1.3 Classification of MGMT Methylation

Cancerous glioblastoma tissue often exhibits MGMT (O6-methylguanine DNA methyltransferase) gene promoter methylation, which reduces gene expression [6]. The MGMT gene encodes a DNA repair enzyme that protects tumor cells from damage by alkylating agents, which prevent tumor growth by attacking tumor cell DNA [17]. MGMT methylation reduces MGMT transcription, decreasing the production of MGMT proteins which protect tumor cells from chemotherapy.

The alkylating agent temozolomide has shown particular chemotherapy effectiveness in patients with MGMT methylation, resulting in improved prognosis and survival [6]. Older patients, with less tolerance for chemotherapy and radiotherapy together, could especially benefit from temozolomide treatment [6]. Treatment trials have indicated that patients without MGMT methylation could effectively be treated with radiotherapy alone while patients with MGMT methylation could be treated with temozolomide alone [6].

Since MGMT methylation occurs in 35% of glioblastoma cases [6], the biomarker has predictive value for determining prognosis for many patients. Detecting MGMT methylation could also encourage treatment with temozolomide, reducing unnecessary costs and preventing side effects of aggressive therapy in older patients.

[1]More recent studies note that the neural subtype originates from marginal tumor regions with far greater normal cell proportions than core tumor areas, arguing that the molecular subtype classification for glioblastoma should only include the classical, mesenchymal, and proneural subtypes [13], [14].

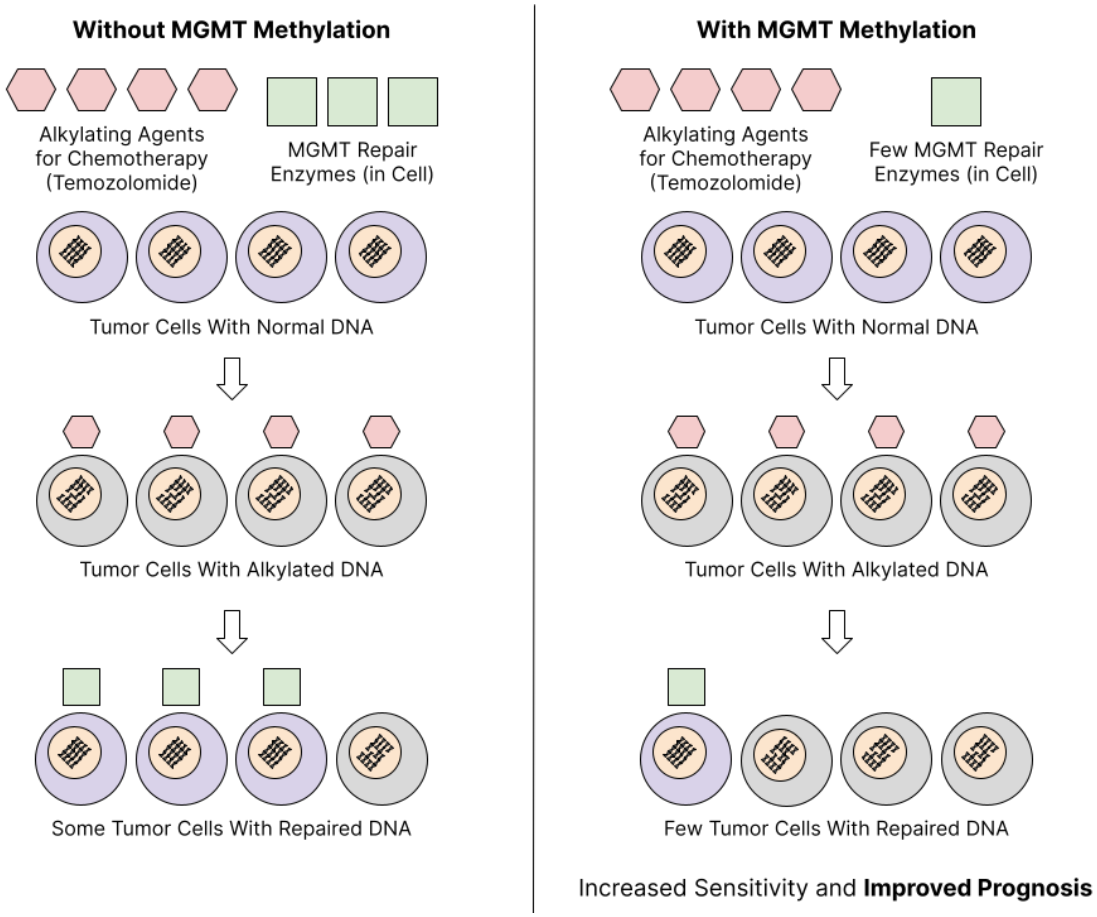



**Figure:** A depiction of tumor response to alkylating agents without and with MGMT methylation. MGMT methylation improves prognosis by preventing MGMT repair enzymes from aiding damaged tumor cells, increasing chemotherapy sensitivity.

## 1.4 Classification of EGFR Amplification

Glioblastoma tumor tissue also often exhibits EGFR (Epidermal Growth Factor Receptor) gene amplification, which increases gene expression [18]. The EGFR gene encodes a transmembrane (spanning the cell membrane) receptor which binds to EGF (Epidermal Growth Factor) proteins [19]. The EGF proteins, after binding, stimulate cell growth and division [19]. EGFR amplification in tumors increases EGFR transcription, producing more receptors that can stimulate tumor growth.

By increasing cell proliferation, EGFR amplification increases tumor resistance to radiotherapy and chemotherapy, the common treatments for glioblastoma after resection surgery [18]. However, since EGFR amplification occurs homogeneously in tumor tissue but almost nowhere in normal tissue, the EGFR gene could be an effective therapy target [18]. Many studies have also found that EGFR amplification unfavorably affects overall survival (although some have found no association or even positive association) [20].

Since EGFR amplification occurs in 40% of glioblastoma cases [18], the biomarker could significantly aid patient prognosis from its potential correlation with survival. Detecting EGFR amplification could also encourage applying future treatments that target the gene.

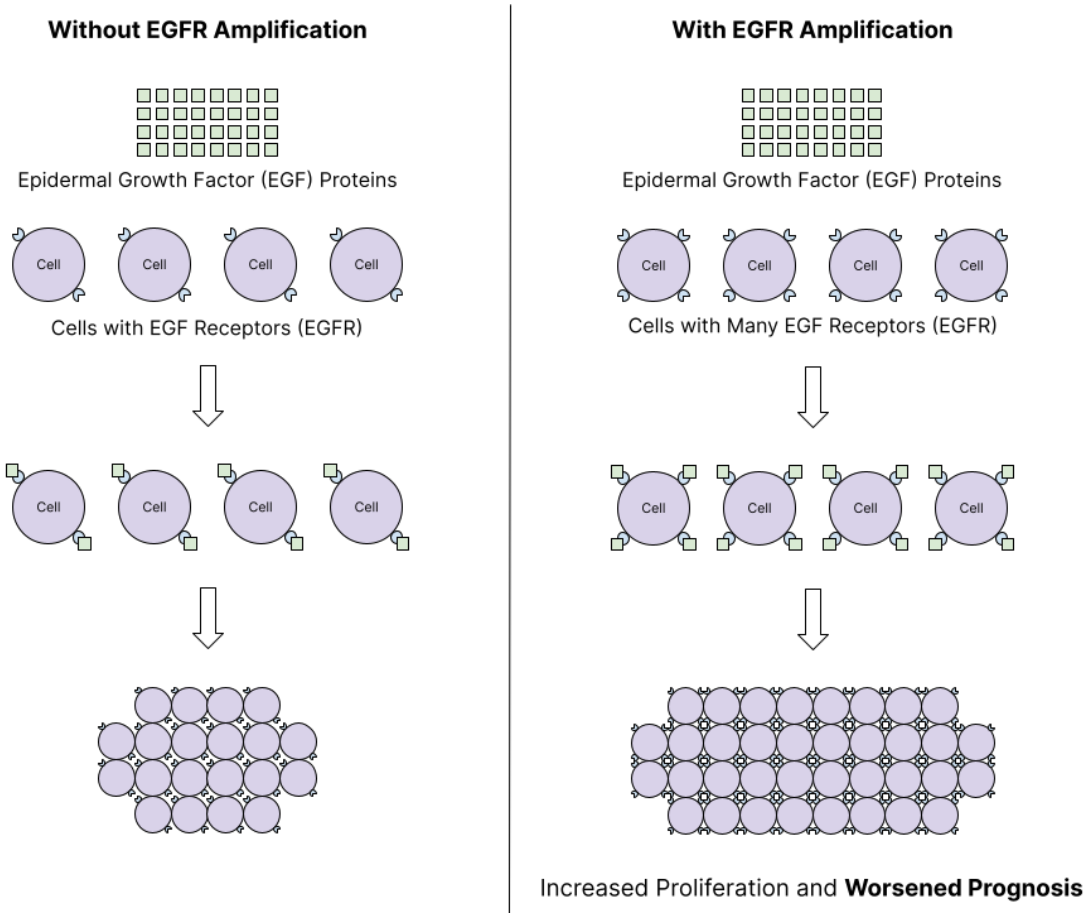



**Figure:** A depiction of tumor growth without and with EGFR amplification. EGFR amplification worsens prognosis by encouraging cell proliferation in tumors.

## 1.5 Limitations

Although manual pathologist examination can determine tumor features and genetic factors, the process requires days to produce a final report [21]. Results from biopsy analysis can also vary according to pathologist expertise and other factors [21]. An accurate, quick, and consistent segmentation tool is therefore necessary to locate tumor features and distinguish genetic factors from biopsy whole slide images. This tool could work alongside pathologists with their domain expertise to produce consistent prognoses and timely diagnoses for patients.

## 1.6 GlioBLAST

This paper presents GlioBLAST, a novel deep learning approach to segmenting tumor features and detecting molecular subtypes, MGMT methylation, and EGFR amplification in whole slide images from brain biopsies. Since whole slide images are extremely large (often over 50

megabytes in file size), they are less used in research due to processing difficulties, including memory limits and time constraints.

Tiling was used to split large whole slide images into more manageable sections for analysis and prediction. The UNet model architecture was employed to construct a segmentation model that locates histological tumor features in whole slide image tiles. The VGG16 model architecture was extended to construct three classification models that detect molecular subtype, MGMT methylation, and EGFR amplification with high accuracy.

These models can predict on individual tiles within less than a second. When running on one baseline GPU, these models can analyze an entire whole slide image for these features within 20 minutes, significantly faster than conventional pathology requiring up to 10 days [17]. This time could greatly decrease with resources like distributed prediction and cloud GPU clusters. A web application was also created, using which medical professionals from regions with limited pathologist access can upload whole slide images, execute these models, and obtain the corresponding heatmaps for tumor features, molecular subtypes, MGMT methylation, and EGFR amplification.

# 2 Research Questions

- Can a deep-learning system be constructed to predict glioblastoma prognosis by detecting molecular subtypes, MGMT methylation, and EGFR amplification and by segmenting tumor features in biopsy whole-slide images?
- Can the deep-learning software be hosted publicly with a web application to become globally accessible, especially in areas lacking pathologists with the specific expertise needed for image analysis and disease diagnoses?

# 3 Data

The segmentation and classification models developed were trained using data from the public Ivy Glioblastoma Atlas Project (Ivy GAP), curated and maintained by the Allen Institute for Brain Science. The Ivy GAP dataset includes 11,250 whole slide images of glioblastoma tissue from a cohort of 41 glioblastoma patients who donated 42 tumors [7]. After biopsy, the tissue slides were scanned with a microscope at a 20X objective (0.5 microns per pixel) [7]. All the tissue slides were stained with hematoxylin and eosin [7], a broadly used histology stain that colors cell nuclei blue and cytoplasm and other material pink in images [22].

The whole slide images were stored as JPG files, typically 15,000 pixels wide and 18,000 pixels high (the corresponding tumor feature masks had similar dimensions) [23]. Using the Allen Brain Map API, metadata within the dataset was downloaded for the 11,250 images: the image ID (within the Allen Brain Map database), molecular subtype, MGMT methylation status, and EGFR amplification status (for the classification models).

| ? Tumor | Molecular Subtype | Extent of Resection | Surgery | MGMT Methylation | Survival Days | EGFR Amplification | Initial KPS |
|---|---|---|---|---|---|---|---|
| W1-1-2 | Classical | Complete | primary | No | 105 | Yes | 100 |
| W2-1-1 | Classical, Neural | Complete | primary | Yes | 1096 | Yes | 90 |
| W3-1-1 | Classical, Mesenchymal | Complete | primary | No | 982 | Yes | 100 |
| W4-1-1 | Mesenchymal, Neural | Complete | primary | No | 540 | No | 90 |
| W5-1-1 | Classical, Neural | Complete | primary | No | | Yes | 90 |
| W6-1-1 | Mesenchymal | Complete | primary | No | 633 | No | 90 |
| W7-1-1 | Mesenchymal | Sub-total | primary | Yes | 437 | No | 100 |
| W8-1-1 | Classical, Mesenchymal | Complete | primary | No | 442 | No | 70 |
| W9-1-1 | Proneural | Complete | primary | No | 145 | No | 90 |
| W10-1-1 | Proneural | Complete | primary | Yes | | No | 100 |
| W11-1-1 | Classical, Mesenchymal | Complete | primary | Yes | 1076 | No | 100 |
| W12-1-1 | Classical | Sub-total | primary | No | 80 | Yes | 70 |

**Figure:** A sample of the metadata from the Ivy GAP dataset for different tumors [23].

The metadata for all the images was stored within a CSV file. From this metadata, a random sample was selected containing 1,000 images (with the Python random module). Using the Allen Brain Map image download service, each image and its tumor feature mask was downloaded. A sample of 1,000 images was chosen due to large file sizes: the images ranged from 25 to 75 megabytes (depending on white space), while the masks ranged from 5 to 10 megabytes. The 2,000 files totalled 55 gigabytes and required five hours to download.

```
@dataclass
class ImageMetadata:
    """
    The metadata for a single image from the dataset, along with preprocessing.
    """

    # The ID of the image in the online dataset.
    id: int

    # Biological data about the image.
    molecular_subtype: list = None
    mgmt_methylation: bool = None
    egfr_amplification: bool = None
```

**Figure:** The Python class used to manage the metadata for each image during processing.

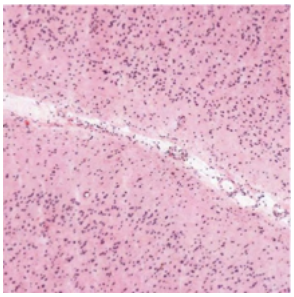
## 3.1 Tumor Features

The Ivy GAP dataset includes tumor feature annotations for all eleven thousand whole slide images, which indicate nine different tumor features with different colors [7]. These tumor annotations were constructed with a semiautomated application, whose results were trained and reviewed for accuracy by consulting with neuropathologists [7].

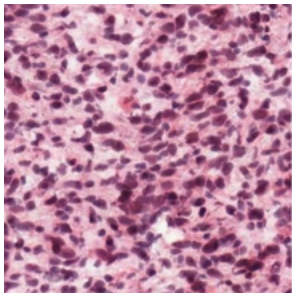
| | |
|---|---|
| | **Leading Edge:** The leading edge represents the outer boundary of the tumor and has a few tumor cells for every 100 normal cells. |
| | **Infiltrating Tumor:** The infiltrating tumor resides after the leading edge but before the dense tumor area and has 10 to 20 tumor cells for every 100 normal cells. |

| | |
|---|---|
| | **Cellular Tumor:** The cellular tumor represents the dense area of the core tumor and has almost no normal cells. |
| | **Perinecrotic Zone:** The perinecrotic zone resides within the cellular tumor and has cells that border the necrotic zone (without pseudopalisading cells detected). |
| | **Pseudopalisading Cells but No Visible Necrosis:** The pseudopalisading cell zone resides within the cellular tumor and has a high density of tumor cells. |
| | **Pseudopalisading Cells Around Necrosis:** The pseudopalisading cell zone resides within the cellular tumor and has a high density of tumor cells. |
| | **Hyperplastic Blood Vessels:** Hyperplastic blood vessels reside within the cellular tumor and have thickened walls from cell proliferation. |
| | **Microvascular Proliferation:** Microvascular proliferation occurs in the cellular tumor and has blood vessels sharing walls that contain endothelial and muscle cells. |
| | **Necrosis:** The necrotic zone resides within the cellular tumor and primarily has dead or dying cells with fragmented nuclei. |

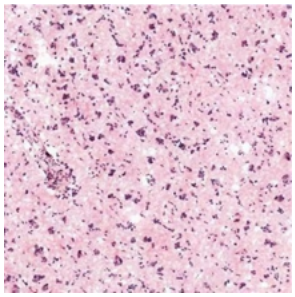
Example tiles displaying tumor features from the dataset white paper are shown below [7].



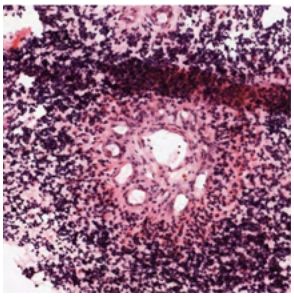
Leading Edge



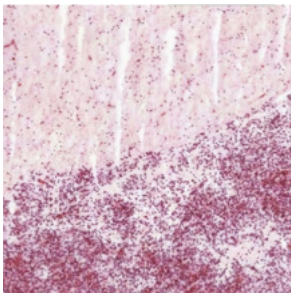
Cellular Tumor



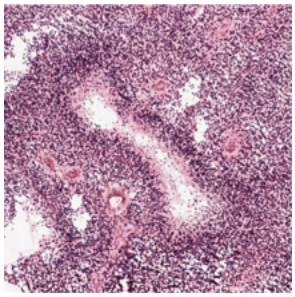
Infiltrating Tumor



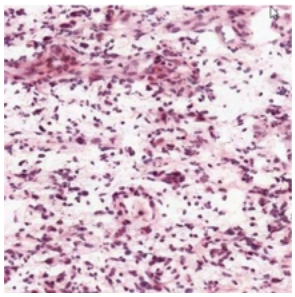
Microvascular Proliferation



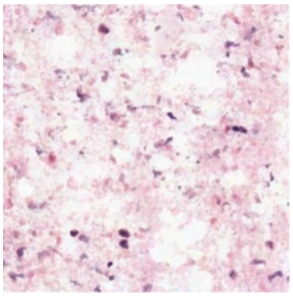
Perinecrotic Zone



Pseudopalisading Cells

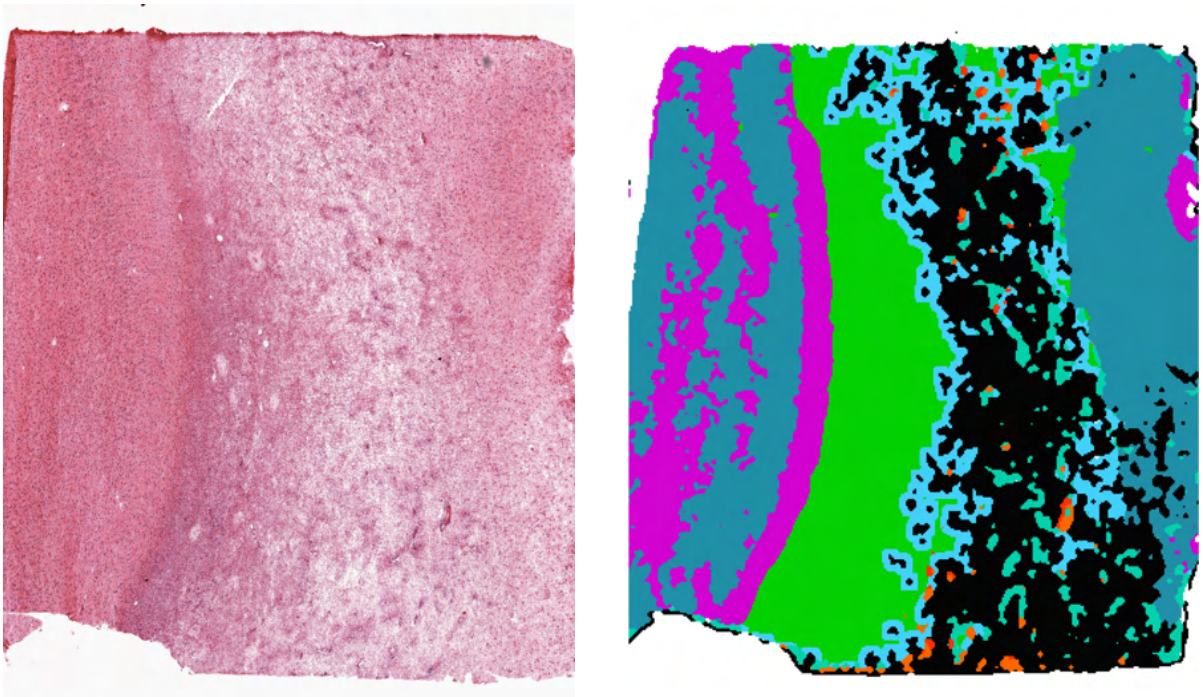


Hyperplastic Blood Vessels



Necrosis

From these tumor features and the corresponding colors (above), an example segmentation mask of tumor features from the dataset is shown below [23].



## 3.2 Molecular Subtypes

The Ivy GAP dataset provides molecular subtype values for all 42 glioblastoma tumors from the 41 patients. For each tumor, cellular-tumor samples (from the core tumor region) were collected from multiple tumor blocks and the corresponding RNA-sequencing data from the dataset was examined. By matching this genetic data with the genetic characteristics of each subtype, the subtype for each tumor was determined, and the tumors were classified as mixtures if different cellular-tumor samples displayed different subtypes.

The molecular-subtype values were stored within the metadata CSV file using the given class name, such as "Classical" or "Proneural."

## 3.3 MGMT Methylation

The Ivy GAP dataset provides MGMT-methylation values for all 42 glioblastoma tumors. These values were determined from standard diagnostic tests to determine whether the MGMT gene was methylated in tumor samples.

The MGMT-methylation values were stored within the metadata CSV file as binary values, with "No" for MGMT-methylation negative and "Yes" for MGMT-methylation positive.

## 3.4 EGFR Amplification

The Ivy GAP dataset provides EGFR-amplification values for all 42 glioblastoma tumors. These values were determined from standard diagnostic tests to determine whether the EGFR gene was copied multiple times within the tumor-cell DNA.

The EGFR-amplification values were stored within the metadata CSV file as binary values, with "No" for EGFR amplification negative and "Yes" for EGFR amplification positive.

# 4 Technology

The Python programming language was used throughout the project for building both the deep learning models and the web application. Python was chosen for its simplicity and expressive code, and the language finds broad use in data science and web development.

Using Python, a virtual environment was created using the standard library venv module. Virtual environments allow developers to install specific versions of libraries (such as TensorFlow 2.7 or Flask 2.0) for use in different projects. The venv module is part of the Python standard library (available with every installation of Python).

All scripts for the project were initially developed with Python 3.9 and run on a laptop running Windows 10. A NVIDIA GeForce GTX 1650 GPU within the laptop was used for training image data. While experimenting with different model architectures and sizes, some models were also trained on an Amazon Web Services (AWS) virtual server (EC2) running Ubuntu 18.04.

## 4.1 Models

The GlioBLAST models were developed with deep learning. Deep learning, a subfield of machine learning and artificial intelligence, uses neural networks to create models that produce output predictions from input data. Artificial intelligence involves computers learning to make decisions, while machine learning focuses on computers learning from *example data*. Deep learning extends machine learning with neural networks, which are mathematical models inspired by the human brain.

Neural networks contain layers of neurons, which are inspired by the homonymous biological structures. Each neuron connects to some (and sometimes all) of the neurons in the previous layer. The neuron has a particular value, which depends on the neurons it connects to and the *weights* of the connections. The layers each transform input (from the previous layer) in some way, extracting progressively-more-abstract features while moving toward the output layer.

Training neural networks first involves randomly initializing the weights of the neural network. For each data point of a dataset, the network output is observed and compared to the correct output. With *backpropagation*, the network weights are improved *slightly* at each step, according to a certain *loss function*. (Training decreases the loss-function value over time, using an *optimizer* that specifies how to change the weights after each step.)

Different classes (or architectures) of neural networks specify structures of network neurons and the connections between them. With experimentation, different neural-network architectures are developed for specific types of input data. Convolutional neural networks (CNNs) are popularly applied to image data for their superior performance and independent learning.

The GlioBLAST models, after being trained with deep learning, accept input data (tiles from whole slide images) and output certain predictions. The segmentation model classifies each pixel of the input tile with a certain class, while the classification models output a single probability or probability distribution for the entire tile. The segmentation model is based on the UNet model architecture, and the classification models are based on the VGG16 model architecture.

### 4.1.1 Technology Stack

The models were primarily developed with the Python libraries TensorFlow (2.7), Pillow (8.4), and Matplotlib (3.5).

TensorFlow is a popular platform for machine learning and deep learning. While developed primarily in Python and C++, TensorFlow can be accessed from many languages including Java and JavaScript. TensorFlow relies on the Keras API to allow developers to construct and train neural networks. Using the CUDA toolkit and cuDNN library developed by NVIDIA, TensorFlow also provides thorough integration with GPUs (Graphical Processing Units) for accelerating neural network training.

Pillow is an image processing library derived from the original PIL (Python Imaging Library), whose development was discontinued in 2011. Pillow allows developers to open images from multiple file formats (such as JPG and PNG), convert files to and from NumPy arrays, and manipulate images with common transformations (such as resizing). Using Pillow, images can be opened from the file system and then converted into NumPy arrays and then TensorFlow tensors, which can be used as input to neural network models.

Matplotlib is a visualization and plotting library for Python that, like TensorFlow, depends on the NumPy library. Matplotlib can create common statistical graphs (like line plots, bar charts, and scatter plots) from data passed as Python lists or NumPy arrays. The library also supports more advanced features, such as producing 3D plots and contour plots and creating heatmaps. With respect to deep learning, Matplotlib can be used to quickly generate plots of training metrics like accuracy and loss over time.

### 4.1.2 Custom TensorFlow Callbacks

In TensorFlow, model callbacks allow developers to build custom behavior that runs at particular points during training (such as after completing a single batch or a single epoch). Various custom callbacks were built (extending the Callback class from the TensorFlow API) and used throughout model training for visualizing performance to better optimize hyperparameters like the learning rate during model experimentation.

- A **ModelCheckpoint** callback was built extending the default ModelCheckpoint callback (from the TensorFlow API) to save checkpoints with different filenames after each epoch. (Model checkpoint files contain the model weights and allow developers to restart training or choose a specific model for use according to training metrics like accuracy.)
- A custom **MetricPlot** callback was built to plot metric values over time with Matplotlib. The callback accepts a Keras metric name (such as "accuracy") and, after each epoch,

stores the training and validation values of the metric within a Python list. Then, using Pyplot (from the Matplotlib API), the callback plots the graphs of both the training metric and the validation metric over time and saves the plot at a desired location.

- A custom **ConfusionMatrixPlot** callback was built to plot confusion matrixes over time with Matplotlib. (A confusion matrix depicts model performance on categorical data with a table. The table contains as many rows and columns as classes, where the rows depict actual classes and the columns depict predicted classes. For every actual class and predicted class, the corresponding table element represents the number of occurrences where the actual class was predicted as the predicted class.) During each epoch validation period, the callback stores both the target predictions and the model predictions after each batch and combines these to produce a confusion matrix after each epoch. The callback finally saves the plot for each epoch within a desired folder.

## 4.2 Web Application

Through the GlioBLAST web application, medical professionals can upload tissue WSIs or tiles from tissue WSIs. For single tiles, the professionals can run the three classification models (for molecular subtype, MGMT methylation, and EGFR amplification) and will receive the corresponding classification predictions. For WSIs, the professionals can generate heat maps depicting the predictions for all four models (the segmentation model and the three classification models). These predictions and heat maps are stored for later use, and different medical professionals can log in and access their previous records.

### 4.2.1 Technology Stack

The web application was primarily developed with the Python libraries Flask (2.0), SQLAlchemy (1.4), WTForms (2.3), and Huey (2.4).

Flask is a web microframework based on the WSGI (Web Server Gateway Interface) specification for Python web frameworks and servers. (WSGI provides a standardized way for frameworks and servers to interact with each other, allowing developers to choose the framework and server they use at will.) Flask specifically controls the routing behind the web application, running different Python functions whenever users access specific routes. Flask also controls session management like storing the current user ID.

SQLAlchemy is a database library that interacts with SQL databases. The SQLAlchemy library itself includes two primary components: SQLAlchemy Core and SQLAlchemy ORM. The Core abstracts raw SQL commands with Python functions, while the ORM builds on the Core and allows developers to create SQL database tables from Python objects. The ORM component was used for interacting with an SQLite database. (SQLite is a lightweight SQL database format where data is stored within a file).

WTForms is a form library for Python. Like the SQLAlchemy ORM, WTForms development centers around Python classes. Each Python class represents a particular HTML form, where different attributes of the class represent individual fields within the form (such as a text field for

entering a patient name). WTForms allows developers to dynamically create HTML forms from data and also focuses on data validation, allowing developers to check that entered form data matches particular specifications.

Huey is a task queue written in Python. A task queue is a computer process that runs particular tasks asynchronously when prompted by another process, such as a web application. The web application uses the task queue to run long processes such as generating heatmaps from large whole slide images (these processes can require almost 20 minutes). The Huey task queue binds Python functions to tasks and manages their statuses and processing. Huey can also utilize multiple workers for running tasks.

### 4.2.2 Backend and Frontend Architecture

The backend follows the model, view, controller (MVC) architecture, a common design pattern for web servers. The models are classes that represent objects stored within a database, the views are templates that define how data is displayed to users, and the controllers are functions that execute certain business logic before rendering a view.

In the GlioBLAST application, the models use the SQLAlchemy ORM, which defines the database tables and their attributes. The controllers use Flask routes and WTForms forms, which execute certain tasks (such as reading or updating database objects) before rendering the view. The views use Jinja templates: Jinja extends HTML and enables dynamic rendering of data that the controller passes.

The web application frontend was designed entirely with the Bootstrap framework. Bootstrap is a popular, open source framework that provides a set of HTML components that define page formatting and user interaction. These HTML components are defined by the languages CSS (Cascading Style Sheets), which determines page formatting, and JavaScript, which determines user interaction. Developers can use these HTML components by labeling different HTML page elements with specific classes.

# 5 Segmentation with the UNet Architecture

Image segmentation involves assigning each image pixel to a class. The segmentation model was built for locating histological tumor features within the whole-slide images. For each tile, the tumor-feature model returns a heatmap indicating the most likely segmentation class for every pixel. The model was built using the UNet architecture, a CNN architecture designed specifically for "biomedical image segmentation" [24]. The architecture consists of two components: the encoder layers, which convert the image into feature representations, and the decoder layers, which convert the feature representations into a segmentation map.

The encoder layers and decoder layers are divided into distinct blocks. Each encoder block consists of two convolution layers (which determine features that can then be detected across the image) and one maximum pooling layer (which determines the most present feature over small image patches) [24]. Each decoder block consists of a transposed convolution layer (which

projects the feature representations to a higher resolution space), a concatenation layer (which concatenates part of the encoder block with the current decoder block), and then two convolution layers [24].

The strength of the UNet architecture for segmentation lies in the concatenation and convolution layers used within the decoder component. By combining high resolution features from the encoder block with upsampled features from the decoder block, the later convolution layers can "assemble a more precise output" [24]. This occurs because they have information about both the image layout (from the high resolution features) and the predicted features (from the upsampled features) [24].

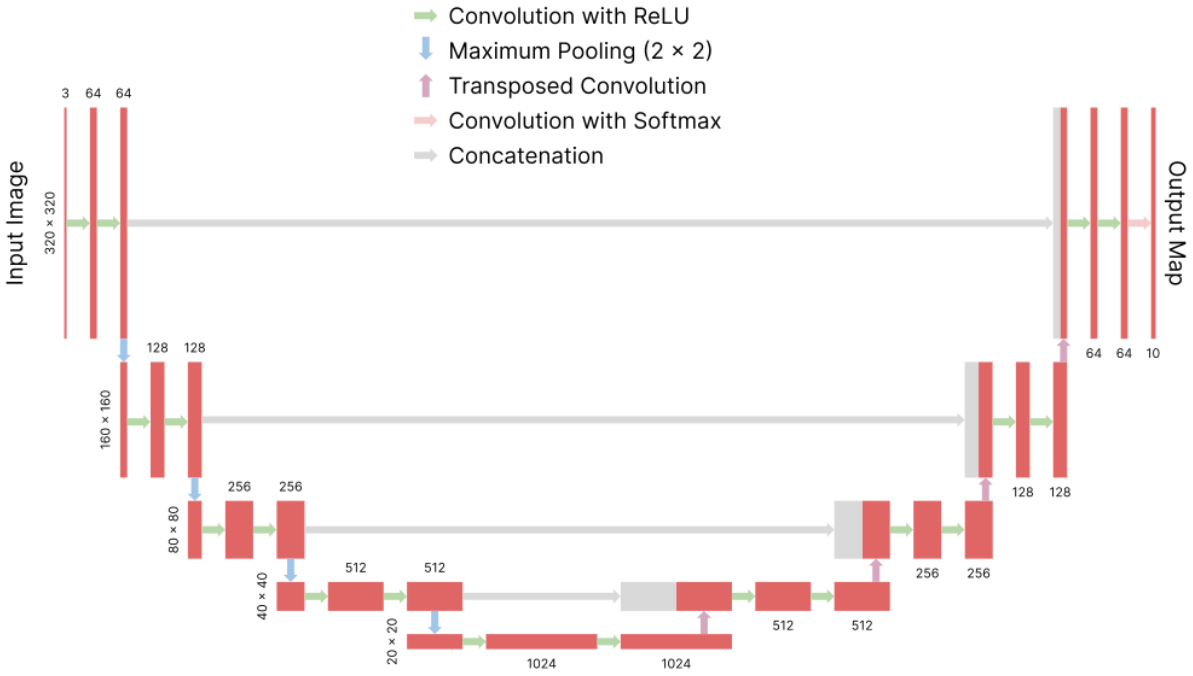



**Figure:** The segmentation UNet architecture. Inspired by the original UNet diagram [24].

The segmentation model using the UNet architecture was trained with the standard categorical-cross-entropy loss function,

$$L(T, O) = -\frac{1}{IRC}\sum_{i=1}^{I}\sum_{r=1}^{R}\sum_{c=1}^{C}\sum_{n=1}^{N} T_{i,r,c,n}\log\left(O_{i,r,c,n}\right),$$

where $T$ and $O$ are the target tensor (from the dataset) and output tensor (from the model), respectively. The tensors have shapes $(I, R, C, N)$, corresponding to $I$ batch images, $R$ rows and $C$ columns, and $N$ classes. The element for image $i$ at row $r$ and column $c$ represents the probability distribution of the classes for the corresponding image pixel.

```
inputs = layers.Input(
    shape=(
        SEGMENTATION_TILE_ROWS,
```

```
        SEGMENTATION_TILE_COLUMNS,
        COLOR_CHANNELS,
    )
)

x = inputs

# The downsampling block outputs, for concatenation with upsampling layers.
downsampling_block_outputs = []

# The downsampling layers.
for filters in [64, 128, 256, 512]:
    x = conv2d_layer(filters)(x)
    x = conv2d_layer(filters)(x)

    downsampling_block_outputs.append(x)

    x = layers.MaxPool2D()(x)
    # x = layers.Dropout(0.1)(x)

# The middle convolutional layers.
x = conv2d_layer(1024)(x)
x = conv2d_layer(1024)(x)

# A dropout layer.
x = layers.Dropout(0.5)(x)

# The upsampling layers.
for filters in [512, 256, 128, 64]:
    x = conv2d_transpose_layer(filters)(x)

    x = layers.Concatenate()([downsampling_block_outputs.pop(), x])
    # x = layers.Dropout(0.1)(x)

    x = conv2d_layer(filters)(x)
    x = conv2d_layer(filters)(x)

# The output layer.
outputs = layers.Conv2D(
    filters=len(SEGMENTATION_MASK_CLASSES),
    kernel_size=1,
    padding="same",
    activation=activations.softmax,
)(x)

model = Model(inputs, outputs)

model.compile(
    optimizer=optimizers.Adam(learning_rate=1e-5),
    loss=losses.categorical_crossentropy,
    metrics=[
        metrics.categorical_accuracy,
```

```
        dice_coefficient,
    ],
)
```

**Figure:** The Python function used to create the UNet segmentation model.

## 5.1 Data Preprocessing

For segmentation, the whole slide images were divided into 320 by 320 pixel tiles. Each image was loaded into a Pillow image and then a TensorFlow tensor, and the number of tiles in the vertical and horizontal directions was calculated from the dimensions. For each tile, the coordinates of the top left corner were calculated, and the original image tensor was sliced to produce the tensor containing the tile pixels. This TensorFlow tensor was converted to a Pillow image and then saved as a JPEG file, producing 2,500 segmentation tiles per whole slide image.

After dividing the images into tiles, the corresponding segmentation masks were similarly divided. The masks (containing colors) were first converted into class arrays with classes corresponding to the different tumor features. However, some mask pixels did not exactly match an expected class color (the RGB tuples differed slightly from the expected ones) because of inaccuracies from lossy JPEG compression. These inaccuracies prevented determining the pixel classes by directly searching for the pixel color among the ten expected ones.

From the semiautomated application that produced the masks, the masks consisted of 45 pixel squares, each corresponding to one segmentation class. The inaccurate pixels mostly occurred at the borders of these squares (due to smoothing at the boundaries of different colors). To produce the class array, the center pixel was extracted from each 45 pixel square (these center pixels rarely had inaccurate RGB values). For each pixel, the corresponding pixel class was found from the pixel color and then extrapolated to the entire 45 pixel square.

This process quickly and accurately produced a class array for each mask. These class arrays, stored as TensorFlow tensors, were divided into tiles using the same algorithm as performed on the image arrays. The class numbers were then converted into one-hot arrays (for using the tiles during segmentation training) and the tile arrays were finally stored as NumPy array files.

## 5.2 Training

For training the segmentation model (based on the UNet architecture), 100 images were divided into tiles as described above. These 100 images were randomly selected (with the Python random model) from the 1000 downloaded images. For each image, 500 image and mask tile pairs were randomly selected from the 2,500 image and mask tile pairs produced, leading to 50,000 tile pairs selected for training. These pairs were split into 80% training and 20% validation sets, leading to 40,000 tile pairs available for training.

The segmentation UNet model was trained for 20 epochs. During model experimentation, various hyperparameters were selected and modified between training runs, using feedback from the training metrics including categorical accuracy and dice coefficient.

- The batch size was selected as 1, also used by the UNet authors for their segmentation model. Higher batch sizes (2, 4, and 8) were also tried but resulted in poorer performance or exhausted GPU memory (with more data loaded during each batch).
- The numbers of filters in the encoder and decoder convolutions were selected as 64, 128, 256, 512, and 1024. Different combinations that created smaller models (which used less GPU memory and trained more quickly) were also tried, such as filters from 64 to 512 (three encoder and decoder blocks) and from 32 to 512 (four blocks). However, the original and larger combination proved superior on metrics like categorical accuracy.
- The preliminary UNet implementation used an UpSampling2D layer (from the Keras API) when moving between the decoder blocks. This layer was later switched with a Conv2DTranspose layer (from the Keras API). The transposed convolution layer has parameters (unlike the upsampling layer) and performs an inverse convolution that is sometimes used instead of upsampling between decoder blocks.
- The preliminary UNet implementation had a single Dropout layer (from the Keras API) with dropout probability 0.5. Other dropout layouts were tried, including no dropout layers and dropout layers after each encoder and decoder block. However, the original configuration with a single Dropout layer was most effective.
- The model was trained with the Adam optimizer and learning rate of 0.00001. Other learning rates were also tried, mostly ranging between 0.000001 and 0.0001. The model was trained with the categorical cross entropy loss (from the Keras API). Other loss functions were also tried, including custom weighted categorical cross entropy, Dice coefficient, and generalized Dice coefficient loss functions, although none provided better performance. (The dice coefficient was still used as a training metric.)

Model checkpoints were produced and saved after each epoch using the ModelCheckpoint callback. The training metric history for the categorical accuracy, validation categorical accuracy, Dice coefficient, and validation Dice coefficient were visualized with the MetricPlot training callback. Confusion matrixes, which proved especially useful for visualizing performance, were produced and saved after each epoch using the ConfusionMatrixPlot callback.

# 6 Classification with the VGG16 Architecture

Image classification involves assigning an entire image to a particular class. For the classification models, whole slide image tiles were classified according to molecular subtypes, MGMT methylation, and EGFR amplification, resulting in *three* classification models.

- For each whole slide image tile, the molecular subtype model returns a value list indicating the predicted probabilities for the four classes. For example, the probabilities could be 0.1, 0.8, 0.05, 0.05, signifying the second as the most likely molecular subtype.
- For each tile, the MGMT methylation model returns a single value indicating the predicted probability that the tile exhibits MGMT methylation.
- For each tile, the EGFR amplification model similarly returns a single value indicating the predicted probability that the tile exhibits EGFR amplification.

The classification models were built upon the VGG16 architecture, which originates from the famous VGG16 model trained for the 2014 ILSVRC (ImageNet Large Scale Visual Recognition Challenge). In the ILSVRC, teams compete to classify images according to 1000 general classes by training models on ImageNet, a database of over 14 million annotated images.

The original VGG16 architecture consists of convolution layers followed by three dense (or fully connected) layers. The first dense layers contain 4096 units, while the final one contains 1000 units. To highlight the most prominent features after different convolutions, the model also consists of maximum pooling layers within the convolution layers. The convolution layers are followed by ReLU (Rectified Linear Unit) activations, while the final dense layer is followed by a softmax activation to produce a probability distribution.

For classification, the convolution layers (the first 18 layers of the network) were used with their weights from pretraining on ImageNet. These layers were frozen so that they were not altered during backpropagation (to increase training speed). These layers were followed by a flattening layer (which flattens input into a single list), a single dense layer with 512 units, and a dropout layer with dropout probability 0.5.

For the categorical model (molecular subtype), these layers were followed by a dense layer with 4 units (the number of molecular subtype classes) and a softmax activation (to output a probability distribution over the classes). For the binary models (MGMT methylation and EGFR amplification), these layers were followed by a dense layer with 1 unit and a sigmoid activation (to output a single probability).

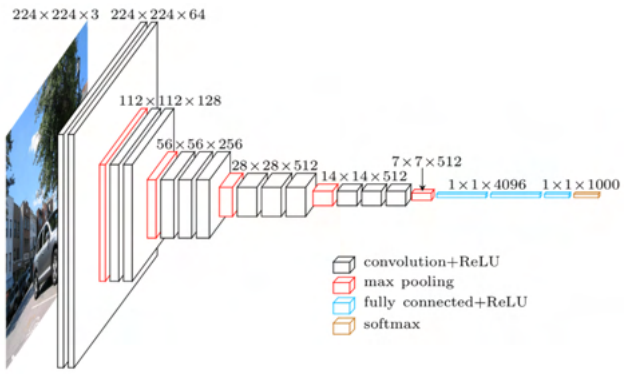



**Figure:** The classification VGG16 architecture [25].

The categorical classification model using the VGG16 architecture was trained with the standard categorical cross-entropy loss function,

$$L(T, O) = -\frac{1}{I}\sum_{i=1}^{I}\sum_{n=1}^{N} T_{i,n}\log\left(O_{i,n}\right),$$

where $T$ and $O$ are the target tensor (from the dataset) and output tensor (from the model), respectively. The tensors have shapes $(I, N)$, corresponding to $I$ batch images and $N$ classes. The element for image $i$ represents the probability distribution of the classes for the image.

The binary classification models using the VGG16 architecture were trained with the standard binary cross-entropy loss function,

$$L(T, O) = -\frac{1}{I}\sum_{i=1}^{I} T_i \log\left(O_i\right) + \left(1 - T_i\right) \log\left(1 - O_i\right),$$

where $T$ and $O$ are the target tensor (from the dataset) and output tensor (from the model), respectively. The tensors have shapes $(I)$, corresponding to $I$ batch images. The element for image $i$ is the probability that the image exhibits the property for the model.

```
base_model = VGG16(
    include_top=False,
    input_shape=(TILE_ROWS, TILE_COLUMNS, COLOR_CHANNELS),
)

# Freeze the base model layers to only train the top layers.
for layer in base_model.layers:
    layer.trainable = False

x = base_model.output

x = layers.Flatten()(x)

x = layers.Dense(512, activation=activations.relu)(x)
x = layers.Dropout(0.5)(x)

if binary:
    outputs = layers.Dense(1, activation=activations.sigmoid)(x)

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

    model.compile(
        optimizer=optimizers.Adam(learning_rate=1e-4),
        loss=losses.binary_crossentropy,
        metrics=["accuracy"],
    )

else:
    outputs = layers.Dense(output_classes, activation=activations.softmax)(x)

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

    model.compile(
        optimizer=optimizers.Adam(learning_rate=1e-4),
        loss=losses.categorical_crossentropy,
        metrics=[metrics.categorical_accuracy],
    )
```

**Figure:** The Python function used to create the VGG16 classification model.

## 6.1 Data Preprocessing

For classification, the whole slide images were divided into 160 by 160 pixel tiles. However, only tiles contained in the cellular tumor region were used for classification, since the cellular tumor region contains the tumor core with the greatest proportion of tumor cells (which exhibit the molecular subtype, MGMT methylation, and EGFR amplification properties).

The whole slide image was tiled for classification similarly as for segmentation, by loading the image into a TensorFlow tensor and slicing the result to produce the tile tensor. To determine whether the tile was contained within the cellular tumor, the segmentation mask *class array* was also loaded (see the segmentation data processing section). The coordinates of the tile center were calculated and these coordinates were then located in the mask class array. If the corresponding segmentation class was 3 (the class for the cellular tumor feature), then the tile tensor was converted into a Pillow image and saved as a JPEG file.

## 6.2 Training

For training the classification model, 100 whole slide images were divided into tiles as described above. These 100 images were randomly selected (with the Python random model) from the 1000 downloaded images. However, when training the three classification models, not all 100 images could be used, since the dataset did not include values for certain sample tumors (from the 42 total) and the corresponding images. In particular, 5 tumor samples did not have a molecular subtype value, 2 tumor samples did not have an MGMT methylation value, and 11 tumor samples did not have a EGFR amplification value.

When preparing data for training, the images without values for a particular classification property (such as molecular subtype) were ignored. From the remaining images (at least 70 images were available for each model), 100 tiles were randomly selected from all tiles produced by the tiling procedure. For each image, the selected tiles were paired with the corresponding image label (for example, if an image had a molecular subtype of Classical, the selected tiles were all paired with this label). Then, these pairs (at least 7,000 for each model) were split into 80% training and 20% validation sets, leading to at least 5,600 tile pairs available for training.

The three classification models were trained for different numbers of epochs: the molecular subtype model was trained for 10 epochs, the MGMT methylation model was trained for 20 epochs, and the EGFR amplification model was trained for 30 epochs (depending on the time before the models began significantly overfitting). The three models were trained with similar hyperparameters, which were chosen through experimentation and modification between training runs, using feedback from the training metrics such as accuracy and loss.

- The batch size was selected as 32. Other batch sizes (4, 8, 16, and 64) were also tried but resulted in poorer performance, so the original batch size of 32 was kept for training.
- The number of filters in the final dense layer (see the VGG16 section) was selected to be 512. Other filter numbers were tried for the single dense layer (128, 256, and 1024), and multiple consecutive dense layers were also tried (two dense layers with 512 filters each).

However, these other values either resulted in poorer performance or much longer training without performance improvements.

- The preliminary implementation had a single Dropout layer (from the Keras API) with dropout probability 0.5. Other layouts were tried, including no dropout layers and different probabilities (0.1 and 0.3), but the original configuration was most effective.
- The model was trained with the Adam optimizer and learning rate of 0.0001. Other learning rates were also tried, mostly ranging between 0.00001 and 0.001. The models with a single output were trained with the binary categorical cross entropy loss, while the model with multiple outputs was trained with the categorical cross entropy loss.

Model checkpoints were produced and saved after each epoch using the ModelCheckpoint callback. The training metric history for the accuracy and validation accuracy or categorical accuracy and validation categorical accuracy were visualized with the MetricPlot training callback. Confusion matrixes, which proved especially useful for visualizing performance, were produced and saved after each epoch using the ConfusionMatrixPlot callback.

# 7 Web Application

The GlioBLAST web applications makes the segmentation and classification models accessible to medical professionals worldwide. The application hosts the models using an Amazon Web Services (AWS) server, which provides a user interface where users can interact with the models. Medical professionals can create patient records and then upload patient images (either whole slide images or tiles from whole slide images) to receive the corresponding predictions or heatmaps after the server runs the models on the provided images. These records, predictions, and heatmaps are stored within a database, allowing medical professionals to log in later and review or download the results generated earlier.

## 7.1 Tile Predictions

Medical professionals can upload singular tiles (160 by 160 pixels) to review specific data about a tumor region. After tiles are uploaded, the files are stored within the server file system and recorded within the server database. Medical professionals can then access the tile record and run any classification model to determine the molecular subtype, MGMT methylation status, or EGFR amplification status exhibited by the tile. After the server generates these results, they are also stored within the server database for later review if desired.

## 7.2 Whole Slide Image Heatmaps

Medical professionals can also upload whole slide images (with dimensions of thousands of pixels) to review heatmaps depicting model predictions across the whole slide image. As with tiles, the whole slide images are stored within the server file system and recorded within the server database. For the segmentation model (histological tumor features) and the three classification models (molecular subtype, MGMT methylation, and EGFR amplification), medical professionals can then access the record and generate a prediction heatmap.

When a medical professional requests a prediction heatmap, the server launches a separate worker to run the prediction. During heatmap generation, the whole slide images are loaded into TensorFlow tensors and divided into tiles (320 by 320 pixel tiles for segmentation and 160 by 160 pixel tiles for classification). The server runs the model on these tiles and produces heatmap images which depict the results (with different colors representing different classes). These images are stored within the server file system and recorded within the server database, after which medical professionals can view the heatmap alongside the image.

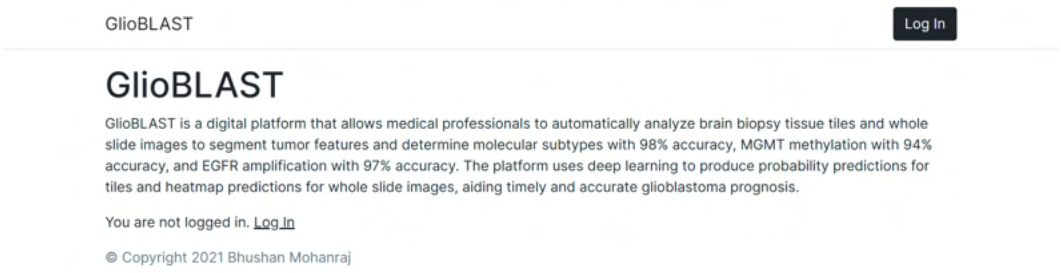
## 7.3 API and Publishing

While the web application provides a custom interface allowing medical professionals to interact with the models, the prediction functionality could also be extracted into an API allowing other programs or servers to interact with the models. Developers for medical schools or hospital networks could then develop applications which meet their organizational needs but can also access the powerful and accurate models presented in this paper.

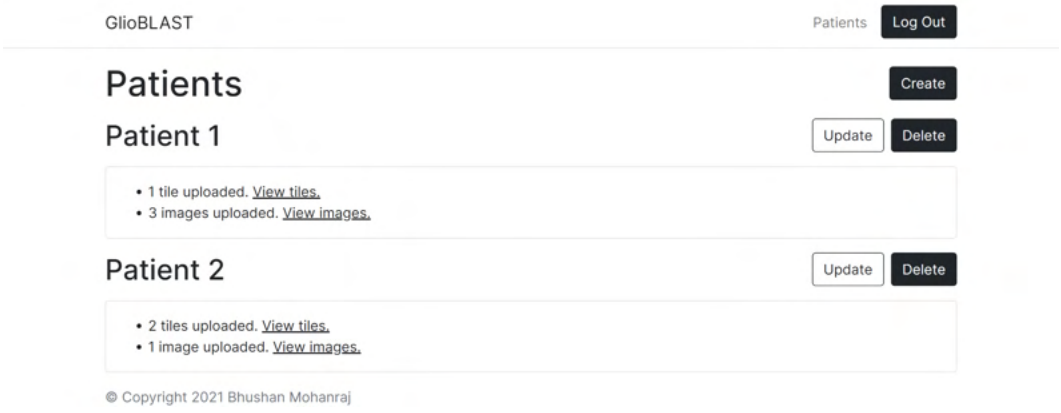
The model could also be published to networks like TensorFlow Hub, where other developers could download the model weights and reuse or repurpose the model as needed.

## 7.4 Screenshots from the Web Application

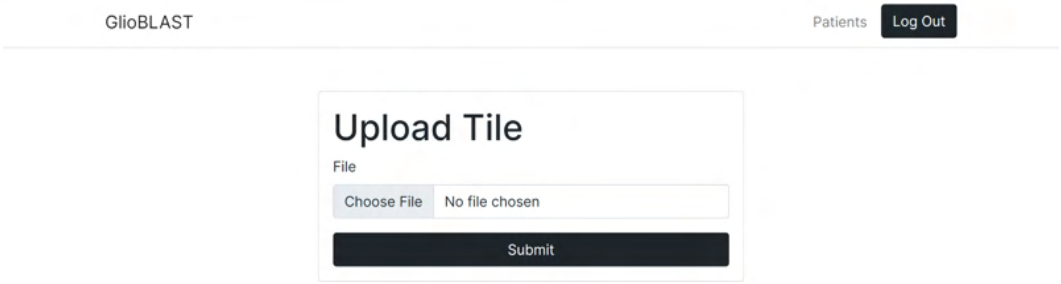
### 7.4.1 Home



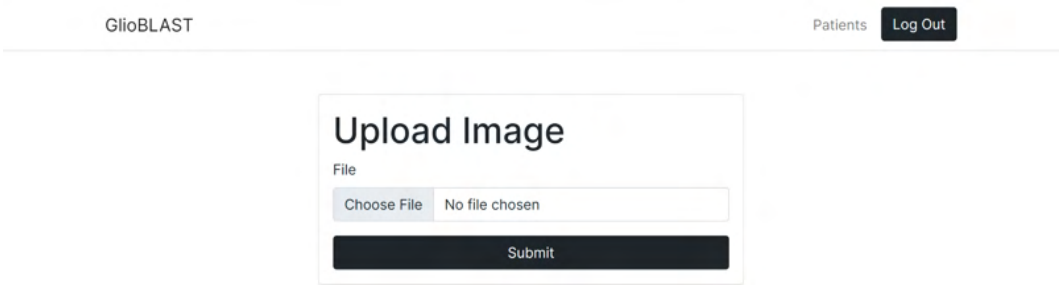

### 7.4.2 Patients



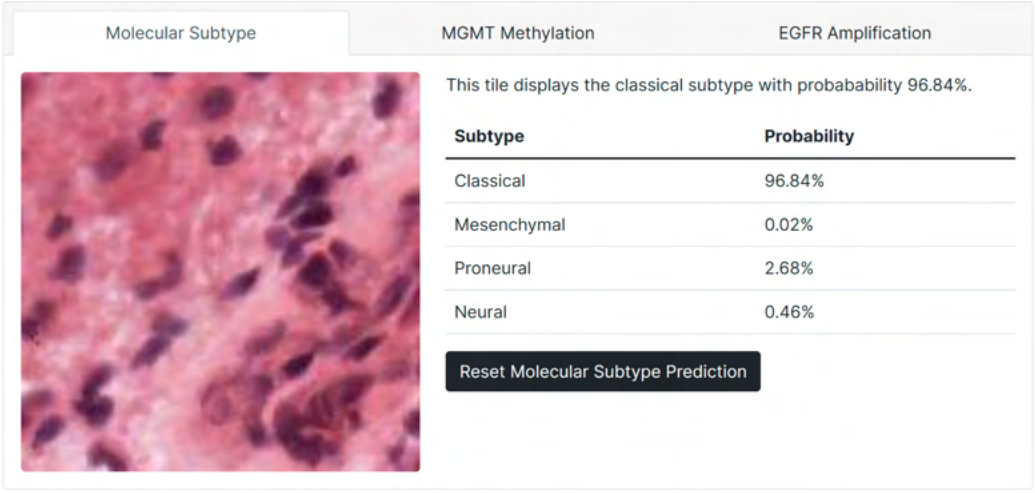

### 7.4.3 Tile Upload



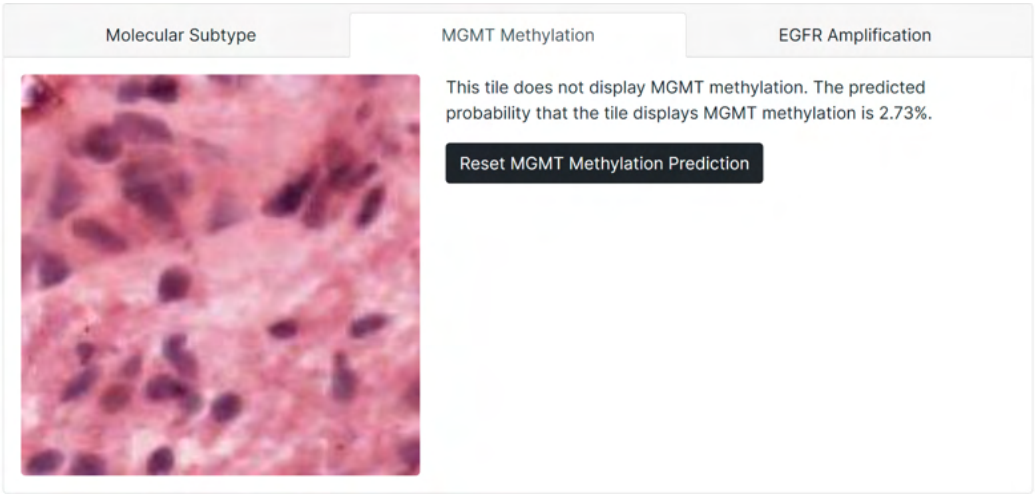

### 7.4.4 Image Upload



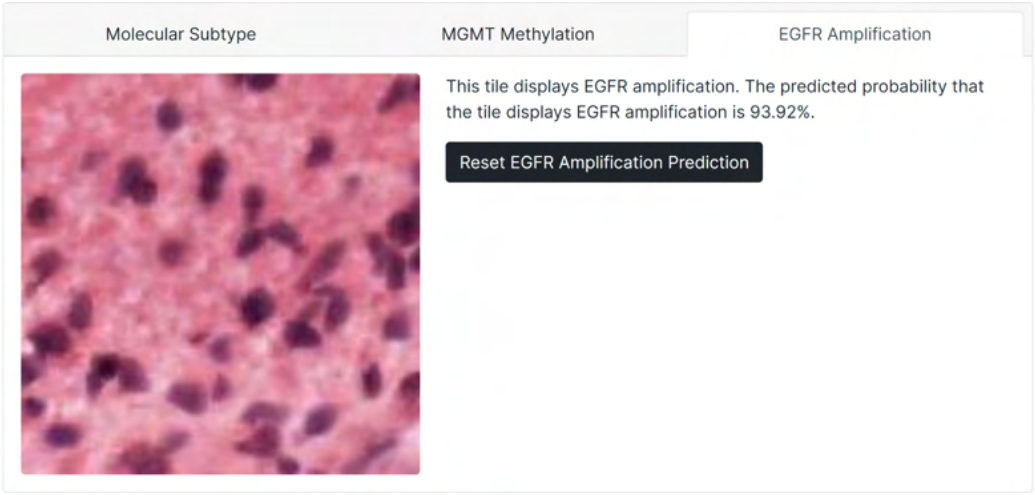
### 7.4.5 Tile Molecular Subtype Prediction



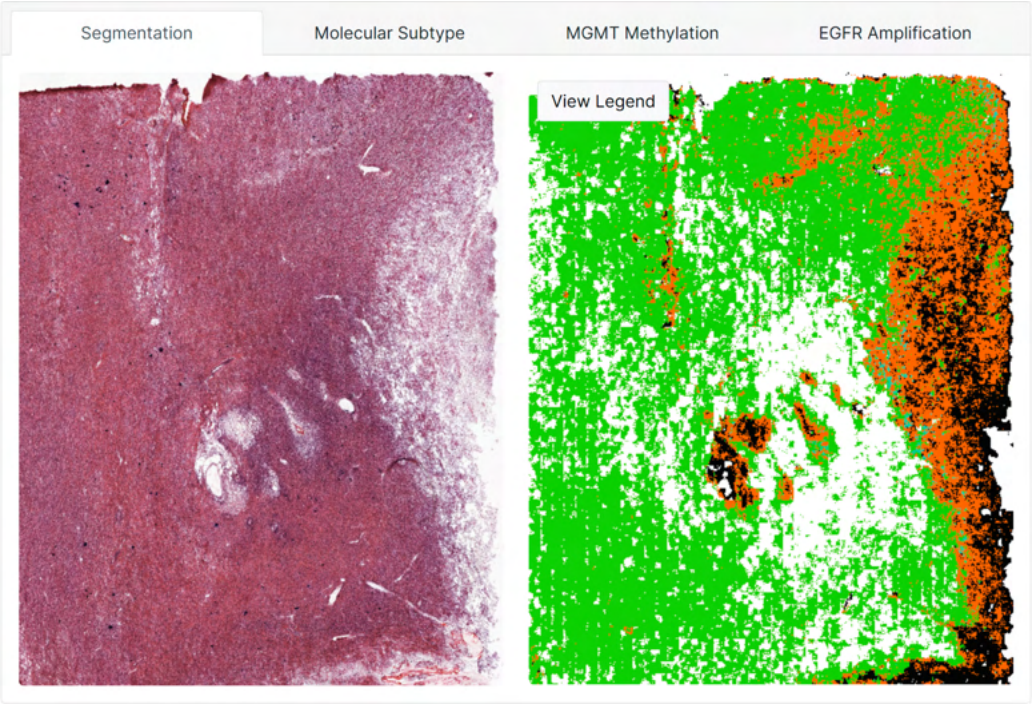

### 7.4.6 Tile MGMT Methylation Prediction



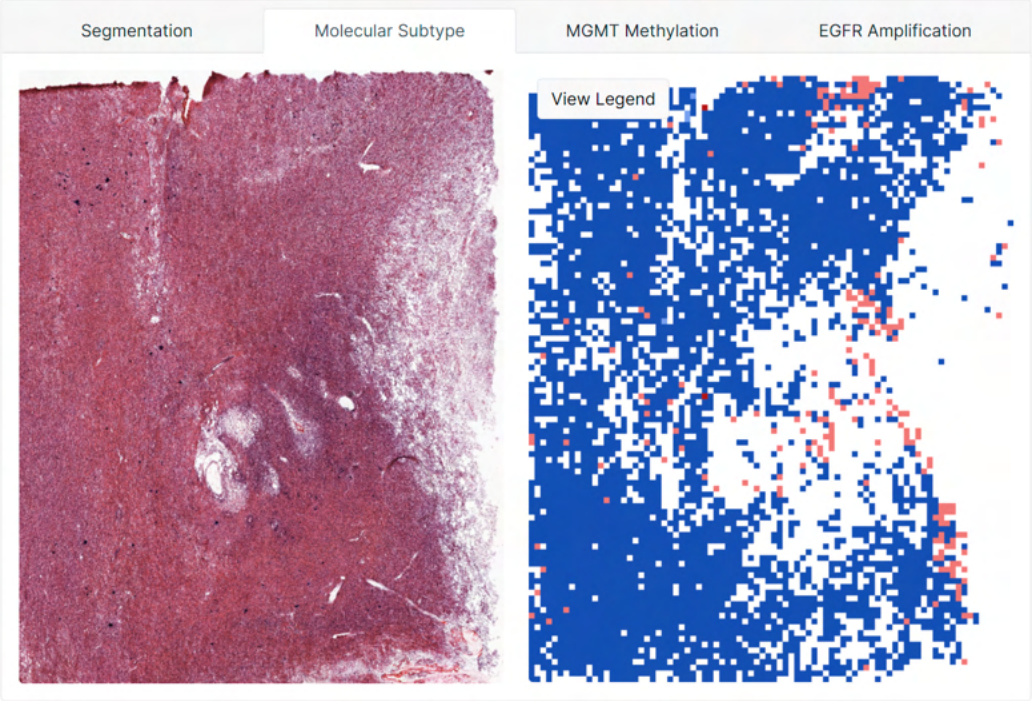

### 7.4.7 Tile EGFR Amplification Prediction



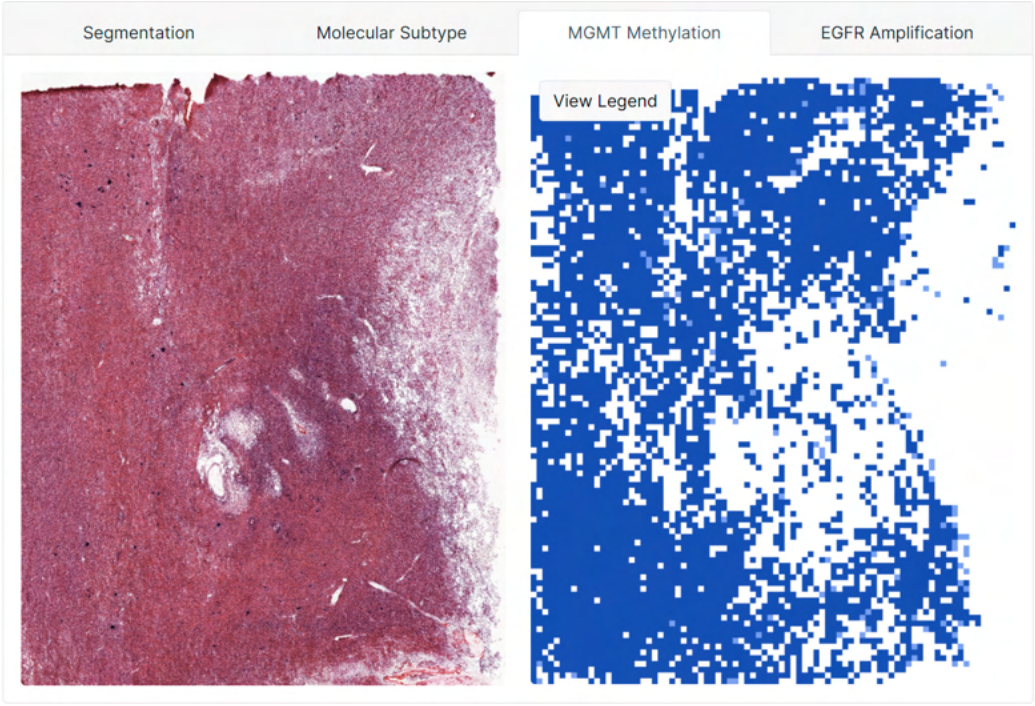
### 7.4.8 WSI Segmentation Heatmap



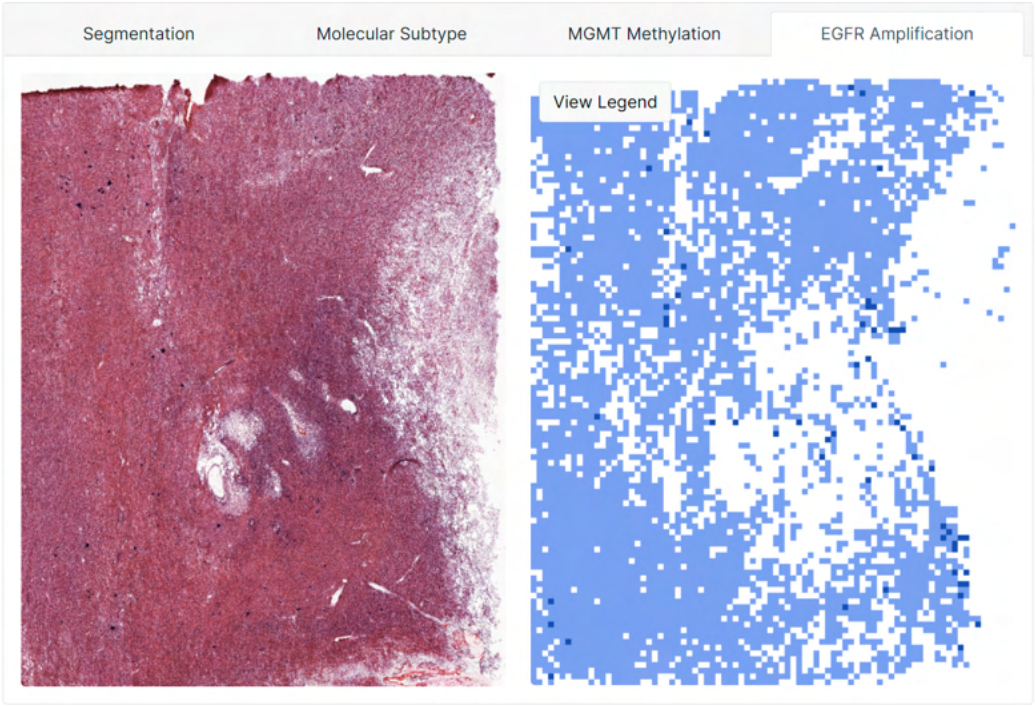

### 7.4.9 WSI Molecular Subtype Heatmap



### 7.4.10 WSI MGMT Methylation Heatmap




### 7.4.11 WSI EGFR Amplification Heatmap



# 8 Results

## 8.1 Tumor-Feature Model

The tumor feature model achieved its greatest categorical accuracy before experiencing significant overfitting after the 13th epoch of training, with a validation accuracy of 75%.

The corresponding confusion matrix exhibits strong performance on major tumor features, including the leading edge, infiltrating tumor, cellular tumor, and necrosis regions (along with the empty class for white space). Yet, the model did not achieve strong performance on the less prevalent classes, which were consistently predicted as cellular tumor regions. However, these predictions are *correct* since these regions are actually within the cellular tumor [7]. Accounting for these classes, their prevalence within the dataset, and the performance per class from the confusion matrix, the categorical accuracy is 82% (other studies have even combined these classes with the cellular tumor class for segmentation).

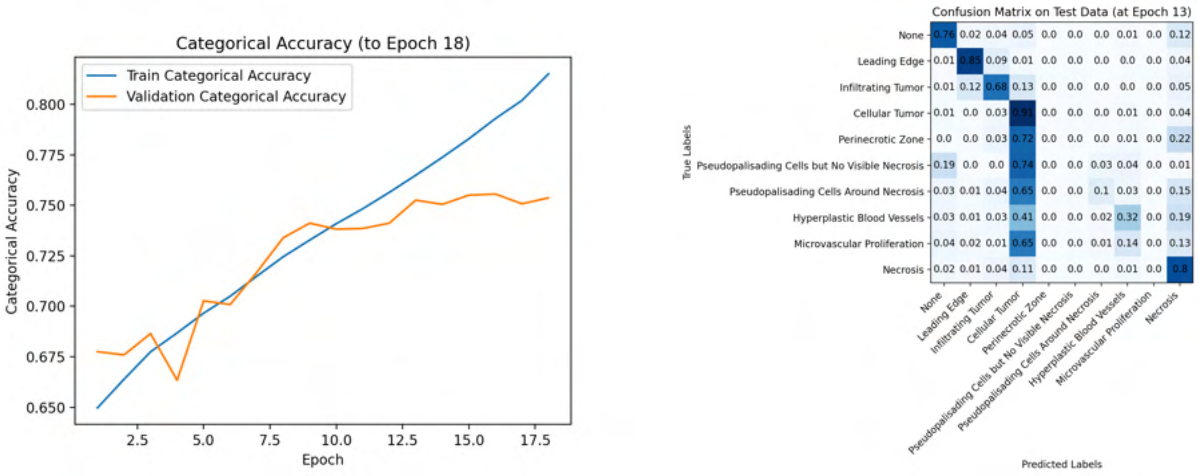



**Figure:** The categorical accuracy plot and confusion matrix for training the histological tumor feature model for 20 epochs.

Future model improvements could focus on improving performance for the less-prevalent classes. The model may have had more difficulty distinguishing these classes because the corresponding regions likely resemble the surrounding cellular tumor areas. These five classes are also significantly underrepresented, consisting only one tenth of the dataset. These issues could be addressed in the future with exposure to more data and with improved computational environments: significantly more data is available than was used due to memory limits.

## 8.2 Molecular-Subtype Model

The molecular subtype model achieved its greatest validation accuracy and lowest validation loss after the 9th epoch of training, with an accuracy of 98%. The loss (measured with categorical cross entropy from the Keras API) decreased from 0.26 to 0.08 from epoch 1 to epoch 9.

The corresponding confusion matrix exhibits strong performance across all classes.

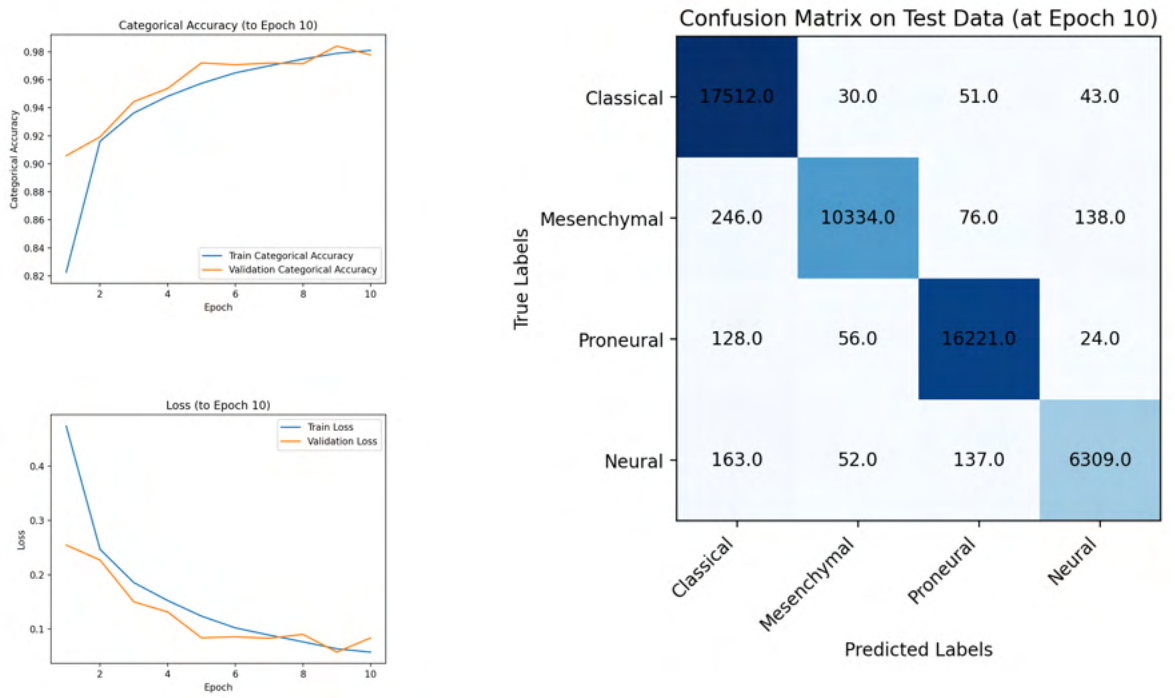



**Figure:** The categorical accuracy plot, loss function plot, and confusion matrix for training the molecular subtype model for 10 epochs.

## 8.3 MGMT-Methylation Model

The MGMT methylation model achieved its greatest validation accuracy and lowest validation loss after the 20th epoch of training, with an accuracy of 94%. The loss (measured with binary cross entropy from the Keras API) decreased from 0.39 to 0.16 from epoch 1 to epoch 20.

The corresponding confusion matrix exhibits strong performance across both classes, with relatively few false positives and false negatives.

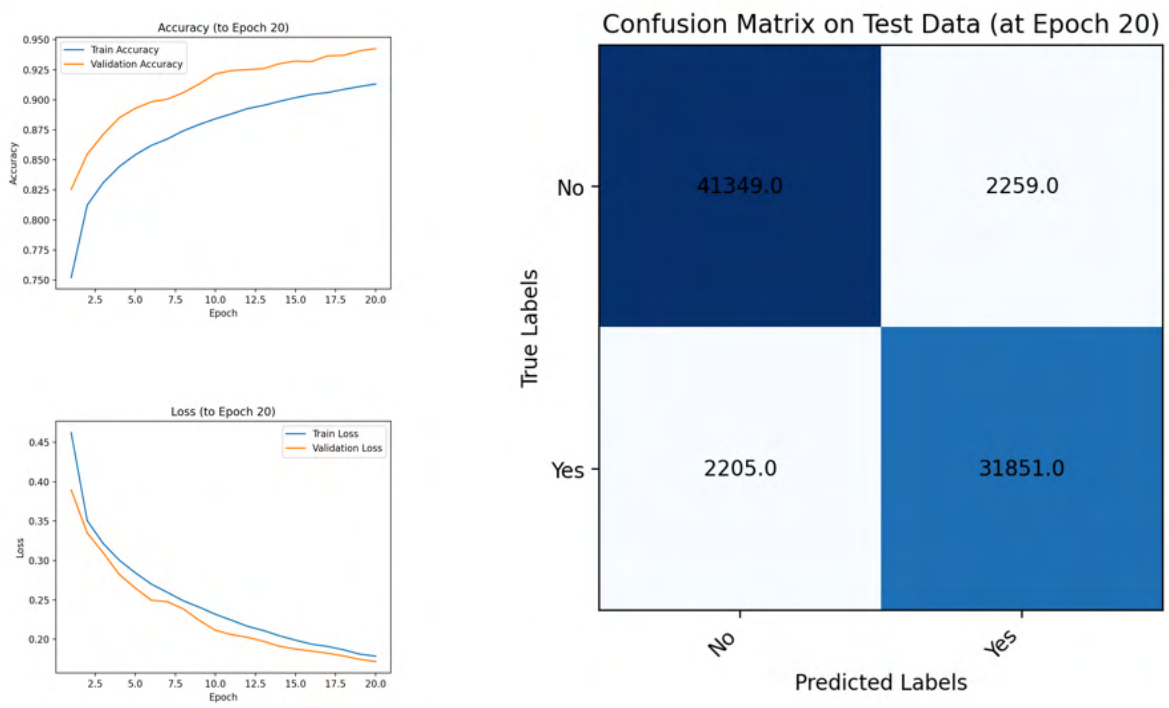



**Figure:** The accuracy plot, loss function plot, and confusion matrix for training the MGMT methylation model for 20 epochs.

## 8.4 EGFR-Amplification Model

The EGFR amplification model achieved its greatest validation accuracy and lowest validation loss after the 30th epoch of training, with an accuracy of 97%. The loss (measured with binary cross entropy from the Keras API) decreased from 0.30 to 0.13 from epoch 1 to epoch 30.

The corresponding confusion matrix exhibits strong performance across both classes, with relatively few false positives and false negatives.

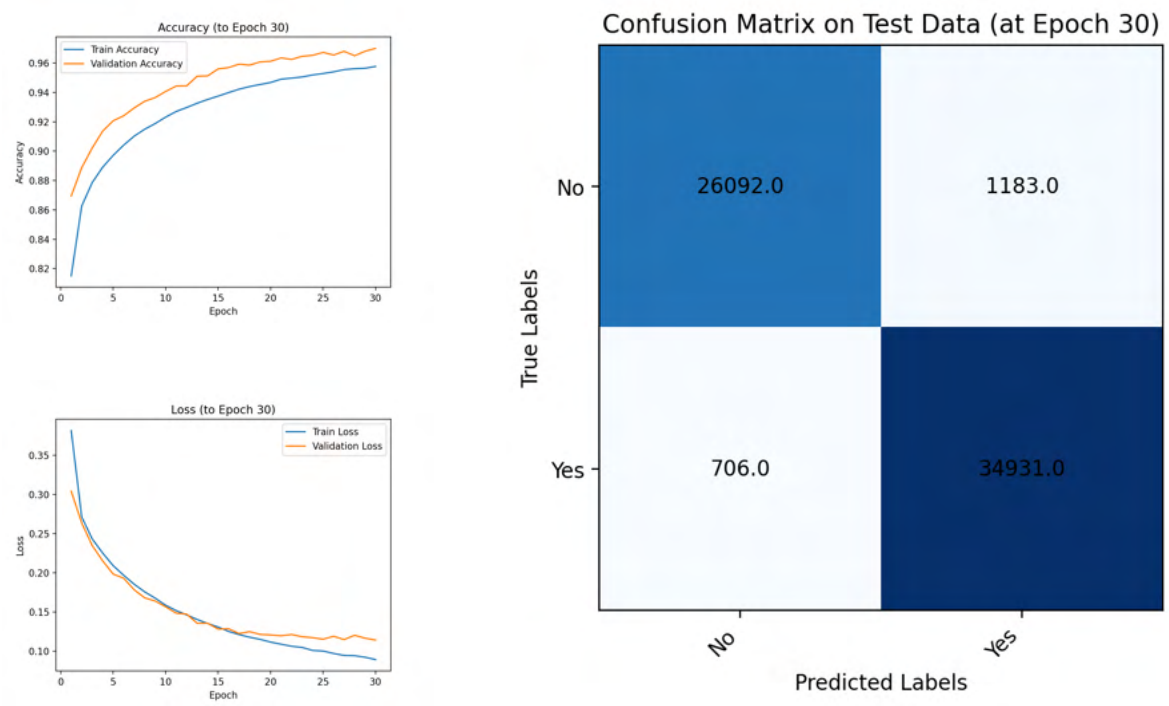



**Figure:** The accuracy plot, loss function plot, and confusion matrix for training the EGFR amplification model for 30 epochs.

# 9 Discussions

## 9.1 Segmentation

After searching for research involving similar deep-learning analysis, the tumor-feature model was compared with existing models according to validation categorical accuracy.

| **Model** | **Categorical Accuracy** |
|---|---|
| FC DenseNet (Tiramisu) on Resized WSIs [26] | 0.70 |
| **My UNet Convolutional Neural Network on Tiled WSIs** | **0.82 (0.75 Unadjusted)** |

The lack of similar research indicates that segmentation for glioblastoma whole-slide images has not been explored thoroughly. This paper addresses the potential benefits of segmentation with deep learning for glioblastoma prognosis. With improved training resources, my methods could achieve improved accuracy, enabling robust glioblastoma tumor-feature segmentation to improve the time and accuracy of glioblastoma diagnosis and prognosis.

## 9.2 Classification

After searching for research involving similar deep-learning analysis, the classification models were compared with existing models according to validation categorical accuracy. No similar research was found using deep-learning analysis to predict molecular subtypes, MGMT methylation, or EGFR amplification from either entire whole-slide image or tiles. However, similar research was found applying deep learning to similar tasks for other gliomas or with data sources like MRI scans. My results display that applying deep learning to whole-slide images has high effectiveness and accuracy.

### 9.2.1 Molecular-Subtype Model

| **Model** | **Categorical Accuracy** |
|---|---|
| DeepRA on MRIs [27] | 0.82 |
| **My VGG16 Convolutional Neural Network on Tiled WSIs** | **0.98** |

### 9.2.2 MGMT-Methylation Model

| **Model** | **Categorical Accuracy** |
|---|---|
| Convolutional Recurrent Neural Network on MRIs [28] | 0.67 |
| L1-Regularized Neural Network on MRIs [29] | 0.88 |
| **My VGG16 Convolutional Neural Network on Tiled WSIs** | **0.94** |

### 9.2.3 EGFR-Amplification Model

| **Model** | **Categorical Accuracy** |
|---|---|
| Logistic Regression on MRIs for Lower-Grade Gliomas [30] | 0.90 |
| **My VGG16 Convolutional Neural Network on Tiled WSIs** | **0.97** |

# 10 Conclusions

This paper presents GlioBLAST, a novel deep learning system that identifies histological tumor features, molecular subtypes, MGMT methylation, and EGFR amplification in glioblastoma tumors. Deep learning has spearheaded advancements in medicine, and this work illustrates that deep learning unlocks improved diagnosis and prognosis for glioblastoma patients. The proposed models, utilizing transfer learning to extend modern image analysis architectures, outperform (in speed and accuracy) other deep learning applications and traditional analysis by pathologists. This allows timely and precise glioblastoma diagnosis and prognosis, reducing unnecessary costs and the side effects of aggressive therapy by enabling targeted treatment. This research also presents the GlioBLAST web application, which complements and enhances traditional glioblastoma treatment. Through this neural-network application, physicians can upload biopsy whole-side images and receive predictions. The publicly-accessible application addresses the shortage of pathologist expertise in many regions through precise diagnosis and prognosis without trained experts or costly systems. With global availability and reliable automation, the GlioBLAST system and web application represent an advancement in applying artificial intelligence to healthcare.

# 11 References


[1] "Astrocytoma - Overview - Mayo Clinic." https://www.mayoclinic.org/diseases-conditions/astrocytoma/cdc-20350132 (accessed Jan. 02, 2022).

[2] "Glioblastoma Multiforme." https://www.aans.org/en/Patients/Neurosurgical-Conditions-and-Treatments/Glioblastoma-Multiforme (accessed Jan. 02, 2022).

[3] "Gliomas." https://www.hopkinsmedicine.org/health/conditions-and-diseases/gliomas (accessed Jan. 02, 2022).

[4] "Glioblastoma - Overview - Mayo Clinic." https://www.mayoclinic.org/diseases-conditions/glioblastoma/cdc-20350148 (accessed Jan. 02, 2022).

[5] "Glioblastoma—Unraveling the Threads to Make Progress," Aug. 03, 2017. https://www.cancer.gov/news-events/cancer-currents-blog/2017/glioblastoma-research-making-progress (accessed Jan. 02, 2022).

[6] E. Calabrese, J. E. Villanueva-Meyer, and S. Cha, "A fully automated artificial intelligence method for non-invasive, imaging-based identification of genetic alterations in glioblastomas," *Scientific Reports*, vol. 10, no. 1. 2020. doi: 10.1038/s41598-020-68857-8.

[7] Allen Institute for Brain Science, "Ivy GAP Technical White Paper," Allen Institute for Brain Science, May 2015.

[8] "Glioma." https://www.mayoclinic.org/diseases-conditions/glioma/diagnosis-treatment/drc-20350255 (accessed Jan. 02, 2022).

[9] D. J. Pisapia, R. Magge, and R. Ramakrishna, "Improved Pathologic Diagnosis—Forecasting the Future in Glioblastoma," *Front. Neurol.*, vol. 8, p. 707, 2017.

[10] "Glioblastoma - Overview - mayo clinic." https://www.mayoclinic.org/diseases-conditions/glioblastoma/cdc-20350148 (accessed Jan. 02, 2022).

[11] L. Zhao, V. H. F. Lee, M. K. Ng, H. Yan, and M. F. Bijlsma, "Molecular subtyping of cancer: current status and moving toward clinical applications," *Brief. Bioinform.*, vol. 20, no. 2, pp. 572–584, Mar. 2019.

[12] R. G. W. Verhaak *et al.*, "Integrated genomic analysis identifies clinically relevant subtypes of glioblastoma characterized by abnormalities in PDGFRA, IDH1, EGFR, and NF1," *Cancer Cell*, vol. 17, no. 1, pp. 98–110, Jan. 2010.

[13] P. Sidaway, "CNS cancer: Glioblastoma subtypes revisited," *Nat. Rev. Clin. Oncol.*, vol. 14, no. 10, p. 587, Oct. 2017.

[14] B. J. Gill *et al.*, "MRI-localized biopsies reveal subtype-specific differences in molecular and cellular composition at the margins of glioblastoma," *Proceedings of the National Academy of Sciences*, vol. 111, no. 34. pp. 12550–12555, 2014. doi: 10.1073/pnas.1405839111.

[15] "NF1 gene." https://medlineplus.gov/genetics/gene/nf1/ (accessed Jan. 02, 2022).

[16] "PDGFRA gene." https://medlineplus.gov/genetics/gene/pdgfra/ (accessed Jan. 02, 2022).

[17] "MGMT Promoter Methylation." https://www.mayocliniclabs.com/test-catalog/overview/36733#Clinical-and-Interpretive (accessed Jan. 02, 2022).

[18] F. S. Saadeh, R. Mahfouz, and H. I. Assi, "EGFR as a clinical marker in glioblastomas and

other gliomas," *Int. J. Biol. Markers*, vol. 33, no. 1, pp. 22–32, Jan. 2018.
[19] "EGFR Gene." https://medlineplus.gov/genetics/gene/egfr/ (accessed Jan. 02, 2022).
[20] J. Hobbs *et al.*, "Paradoxical relationship between the degree of EGFR amplification and outcome in glioblastomas," *Am. J. Surg. Pathol.*, vol. 36, no. 8, pp. 1186–1193, Aug. 2012.
[21] M. Perkuhn *et al.*, "Clinical Evaluation of a Multiparametric Deep Learning Model for Glioblastoma Segmentation Using Heterogeneous Magnetic Resonance Imaging Data From Clinical Routine," *Investigative Radiology*, vol. 53, no. 11. pp. 647–654, 2018. doi: 10.1097/rli.0000000000000484.
[22] A. H. Fischer, K. A. Jacobson, J. Rose, and R. Zeller, "Hematoxylin and eosin staining of tissue and cell sections," *CSH Protoc.*, vol. 2008, p. db.prot4986, May 2008.
[23] Allen Institute for Brain Science, "Ivy Glioblastoma Atlas Project." May 2015. [Online]. Available: https://glioblastoma.alleninstitute.org/
[24] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, 2015, pp. 234–241.
[25] M. Loukadakis, J. Cano, and M. O'Boyle, "Accelerating Deep Neural Networks on Low Power Heterogeneous Architectures," presented at the Eleventh International Workshop on Programmability and Architectures for Heterogeneous Multicores (MULTIPROG-2018), Manchester, UK, Jan. 2018. Accessed: Dec. 31, 2021. [Online]. Available: https://eprints.gla.ac.uk/183819/
[26] A. Z. Shirazi *et al.*, "A deep convolutional neural network for segmentation of whole-slide pathology images identifies novel tumour cell-perivascular niche interactions that are associated with poor survival in glioblastoma," *British Journal of Cancer*, vol. 125, no. 3. pp. 337–350, 2021. doi: 10.1038/s41416-021-01394-x.
[27] R. Su, X. Liu, Q. Jin, X. Liu, and L. Wei, "Identification of glioblastoma molecular subtype and prognosis based on deep MRI features," *Knowledge-Based Systems*, vol. 232, p. 107490, Nov. 2021.
[28] L. Han and M. R. Kamdar, "MRI to MGMT: predicting methylation status in glioblastoma patients using convolutional recurrent neural networks," in *Biocomputing 2018*, WORLD SCIENTIFIC, 2017, pp. 331–342.
[29] I. Levner, S. Drabycz, G. Roldan, P. De Robles, J. G. Cairncross, and R. Mitchell, "Predicting MGMT Methylation Status of Glioblastomas from MRI Texture," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2009*, 2009, pp. 522–530.
[30] Y. Li *et al.*, "MRI features can predict EGFR expression in lower grade gliomas: A voxel-based radiomic analysis," *Eur. Radiol.*, vol. 28, no. 1, pp. 356–362, Jan. 2018.

All figures were created by the author, Bhushan Mohanraj, unless otherwise specified and cited.