REPUBLIC OF TÜRKİYE
AKDENİZ UNIVERSITY

DETECTION OF CANCEROUS BRAIN CELLS WITH MACHINE LEARNING

Uygar CANKAT

INSTITUTE OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

MAY 2024

ANTALYA

REPUBLIC OF TÜRKİYE

AKDENİZ UNIVERSITY

DETECTION OF CANCEROUS BRAIN CELLS WITH MACHINE LEARNING

Uygar CANKAT

INSTITUTE OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

MAY 2024

ANTALYA

REPUBLIC OF TÜRKİYE
AKDENİZ UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

DETECTION OF CANCEROUS BRAIN CELLS WITH MACHINE LEARNING

Uygar CANKAT

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

This thesis unanimously accepted by the jury on 20/05/2024

Assoc. Prof. Dr. Taner DANIŞMAN (Supervisor)
Asst. Prof. Dr. Hüseyin Gökhan AKÇAY
Asst. Prof. Dr. Shahram TAHERI

ABSTRACT

DETECTION OF CANCEROUS BRAIN CELLS WITH MACHINE LEARNING

Uygar CANKAT

MSc Thesis in Computer Engineering

Supervisor: Assoc. Prof. Dr. Taner DANIŞMAN

May 2024, 75 pages

In recent years, there has been an increase in the incidence of brain tumors. Brain tumors are a serious condition that involves the slow growth of a foreign mass in the brain, which can lead to death. Early and accurate detection of tumors is crucial in combating the disease. MRI imaging technology is used to obtain better patterns of brain tumors compared to other technologies. However, detecting these images is not easy, and in some cases, when the disease is difficult to detect, it may be too late for the patient. At this stage, image processing techniques become very important. In this thesis, image processing techniques are crucial. The data from MRI images are analyzed using segmentation and classification techniques, which provide significant benefits to doctors and the medical world in terms of the segmentation, location, and characteristics of the disease. Thus, it may be possible to diagnose diseases at an early stage. In this thesis, 7023 MRI images labeled as glioma, meningioma, pituitary, and notumor from Kaggle were used. These images were cropped from the top and bottom using specific algorithms and resized to dimensions using various preprocessing methods. Once the MRI images were scaled to the desired dimensions, they were grouped into axial, sagittal, and coronal views to improve accuracy and training performance. This thesis deeply examines the application of an advanced artificial intelligence model called EfficientNet in the classification of brain tumors. EfficientNet, when trained on a large dataset, is an architecture that can distinguish different types of brain tumors with high accuracy. During the training process, the ability of deep learning algorithms to recognize complex structural and visual patterns is crucial for the accurate classification of tumors in MRI images. Examining vital characteristics such as the type, size, and location of tumors plays a central role in this process. In this thesis, the integration of the Convolutional Block Attention Module (CBAM) helps optimize the learning process of the models. CBAM allows the model to select important features and reduce background noise,

especially through spatial and channel-based attention mechanisms, significantly enhancing the model's object localization and classification performance. This approach strengthens the model's capacity to analyze brain tumor MRI images more effectively, enabling more precise diagnoses. The focus of the thesis is on distinguishing various subtypes of brain tumors. Categories such as glioma, meningioma, pituitary tumors, and non-tumorous areas are thoroughly examined during the model's training. For each of these categories, special training datasets are used to test the model's accuracy and reliability. Various image augmentation techniques are employed during the preprocessing stage of the datasets using 'ImageDataGenerator'. These techniques include rotation, shifting, scaling, and other image manipulations, helping the model better adapt to real-world variations. Additionally, these techniques reduce the risk of overfitting and increase the overall accuracy of the model.

During the training of the model, the KFold cross-validation method is applied. This method allows the model to be trained and tested multiple times using different subsets of the dataset. Each fold evaluates the model's performance on different data subsets, providing more reliable results, especially in cases where data is limited. This technique allows for a more accurate measurement of the model's performance.

The model's training process consists of two stages: In the first stage, the layers of EfficientNetB7 are frozen, and only the top layers are trained. This approach allows the model to initially learn general features and establish a stable foundation. In the second stage, some lower layers are also included in the training, and fine-tuning is performed. This fine-tuning process helps the model recognize more specific and complex features, achieving higher accuracy.

This thesis demonstrates how machine learning and artificial intelligence can bring significant innovations to the field of medical diagnosis. These technologies, surpassing the limitations of traditional methods, open new horizons for medical research and applications by enabling early diagnosis and rapid treatment. The potential of this innovative approach, particularly in the diagnosis of brain tumors, is comprehensively addressed in this study.

Keywords: EfficientNetB7 Deep Network Method, KFold Cross-Validation Technique, Fine-Tuning, ImageDataGenerators, CBAM

COMMITTEE: Assoc. Prof. Dr. Taner DANIŞMAN
Asst. Prof. Dr. Hüseyin Gökhan AKÇAY
Asst. Prof. Dr. Shahram TAHERI

ÖZET

KANSERLİ BEYİN HÜCRELERİNİN MAKİNE ÖĞRENMESİ İLE TESPİTİ

Uygar CANKAT

Yüksek Lisans, Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Doç. Dr. Taner DANIŞMAN

Mayıs 2024, 75 sayfa

Günümüzde beyin tümörü hastalığında artış görülmektedir. Beyin tümörleri, beyinde yabancı bir kütlenin yavaşça büyümesine ve bir kişiyi ölüme götürmesine neden olan ciddi bir hastalıktır. Tümör teşhisinin erken aşamada ve doğru bir şekilde tespit edilmesi, hastalıkla mücadelede büyük önem taşır. MRI görüntüleme teknolojisi, diğer teknolojilere göre beyin tümörlerini daha net ve ayrıntılı şekilde görüntüleyebilir. Bu görüntülerin tespiti kolay değildir ve bazı durumlarda hastalık zor tespit edildiğinde, hasta için geç olabilir. Bu aşamada, görüntü işleme teknikleri büyük önem taşır. Tez çalışmasında, görüntü işleme teknikleri büyük önem taşır. MRI görüntüsündeki veriler, segmentasyon ve sınıflandırma teknikleri ile analiz edilir ve hastalığın segmenti, konumu ve özellikleri gibi birçok durumda doktorlara ve tıp dünyasına fayda sağlar. Böylece, hastalıkların erken aşamada teşhis edilmesi mümkün olabilir. Bu tez çalışmasında, öncelikle Kaggle üzerinden glioma, meningioma, pituitary ve notumor etiketli 7023 MR görüntüsü kullanılmıştır. Bu görüntüler, beyin tümörlerinin sınıflandırılmasında kullanılan ileri seviye bir yapay zekâ modeli olan EfficientNet'in uygulamasının derinlemesine incelenmesi için kullanılmıştır. EfficientNet, geniş bir veri seti üzerinde eğitildiğinde, farklı beyin tümörü türlerini yüksek doğrulukla ayırt edebilen bir mimaridir. Modelin eğitim sürecinde, derin öğrenme algoritmalarının karmaşık yapısal ve görsel desenleri tanıma yeteneği, MRI görüntülerindeki tümörlerin doğru bir şekilde sınıflandırılması için kritik önem taşır. Tümörlerin tipi, boyutu ve yerleşim yeri gibi hayati özelliklerin incelenmesi bu süreçte merkezi bir role sahiptir. Bu tez kapsamında, Convolutional Block Attention Module (CBAM) entegrasyonu kullanılarak modelin performansı optimize edilmiştir. CBAM, modelin önemli özellikleri seçmesine yardımcı olarak sınıflandırma doğruluğunu artırmakta ve beyin tümörü MRI görüntülerini daha etkin analiz edebilmesini sağlamaktadır. Bu yaklaşım, daha hassas teşhislerin konulabilmesine imkân tanır. Tezin odak noktası, beyin tümörlerinin çeşitli alt türlerinin ayırt edilmesi üzerinedir. Glioma, meningioma, pituitary tümörleri ve tümörsüz alanlar gibi farklı kategoriler, modelin eğitimi sırasında detaylı olarak incelenir. Bu kategorilerin her biri için, modelin doğruluğunu ve güvenilirliğini test etmek amacıyla özel eğitim veri setleri kullanılır. Özellikle, veri setlerinin ön işleme aşamasında, 'ImageDataGenerator' kullanılarak çeşitli görüntü artırma teknikleri uygulanmıştır. Bu teknikler, döndürme, kaydırma, ölçeklendirme ve diğer görüntü manipülasyonlarını içerir; bu sayede modelin aşırı uydurma (overfitting) riski azalır ve genel doğruluğu artar.

Modelin eğitimi sırasında, KFold çapraz doğrulama yöntemi uygulanır. Bu yöntem, veri setinin farklı alt kümelerini kullanarak modelin birden çok kez eğitilmesini ve test edilmesini sağlar. Her bir katlama işlemi, modelin farklı veri alt kümelerindeki performansını değerlendirir, böylece verinin az olduğu durumlarda daha güvenilir sonuçlar elde edilir. Bu teknik, modelin performansını daha doğru bir şekilde ölçmemizi sağlar.

Modelin eğitim süreci iki aşamadan oluşur: İlk aşamada, EfficientNetB7'nin katmanları dondurulur ve yalnızca üst katmanlar üzerinde eğitim yapılır. Bu yaklaşım, modelin öncelikle genel özellikleri öğrenmesine ve stabil bir temel oluşturmasına olanak tanır. İkinci aşamada ise, bazı alt katmanlar da eğitime dahil edilerek modelin ince ayarı (fine-tuning) yapılır. Bu ince ayar süreci, modelin daha spesifik ve karmaşık özellikleri tanımasına ve daha yüksek bir doğruluk elde etmesine yardımcı olur.

Bu tez, makine öğrenmesi ve yapay zekanın tıbbi teşhis alanında nasıl önemli yenilikler getirebileceğini göstermektedir. Geleneksel yöntemlerin sınırlılıklarını aşan bu teknolojiler, erken teşhis ve hızlı tedaviye olanak tanıyarak, tıbbi araştırmalar ve uygulamalar için yeni ufuklar açmaktadır. Özellikle beyin tümörü teşhisinde, bu yenilikçi yaklaşımın potansiyeli, bu çalışma ile kapsamlı bir şekilde ele alınmaktadır.

ANAHTAR KELİMELER: EfficientNetB7 Derin Ağ Methodu, KFold çapraz doğrulama yöntemi, Fine-tuning, ImageDataGenerator, CBAM

JÜRİ: Doç. Dr. Taner DANIŞMAN
Dr. Öğr. Üyesi Hüseyin Gökhan AKÇAY
Dr. Öğr. Üyesi. Shahram TAHERİ

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor and my advisor Assoc. Prof. Dr. Taner DANISMAN for his support, encourage and advises. Finally, I would like to thank my family for their patience and support.

LIST OF CONTENTS

TEXT OF OATH

I declare this study “DETECTION OF CANCEROUS BRAIN CELLS WITH MACHINE LEARNING” which I present as master thesis, is in accordance with academic rules and ethical conduct. I also declared that I cited and referenced all material and results that are not original to this work.

20/05/2024

Uygar CANKAT

İmzası

LIST OF ABBREVIATIONS

Adam : Adaptive Moment Estimation

AI : Artificial Intelligence

ANN: Artificial Neural Network

DL: Deep Learning

DNN : Deep Neural Network

EfficientNetB7 : EfficientNet Model B7 Version

KFold : K-Fold Cross-Validation

ML : Machine Learning

MRI : Magnetic Resonance Imaging

LIST OF FIGURES

1. INTRODUCTION

Medical image processing refers to the general processes used to interpret images applied for scientific purposes within the healthcare field. Techniques such as segmentation, classification, and data augmentation are crucial in the stages of diagnosis, treatment planning, and the analysis of various diseases. These methods assist the medical community by enabling more accurate disease detection and providing doctors with significantly more useful results. The increase in false and delayed diagnoses complicates disease treatment; however, image processing has substantially reduced these errors, yielding highly beneficial outcomes. Technologies like MRI (Magnetic Resonance Imaging) and CT (Computed Tomography) are actively utilized in medical imaging. These technologies are rapidly advancing, offering high levels of reliability. Image processing methods are used not only for detection but also for analyzing disease progression and presentation.

Brain tumors can be classified into two main categories: benign and malignant. Benign tumors are generally less risky due to their slow growth and lower risk of mortality. In contrast, malignant tumors, which cause various symptoms such as headaches and dizziness, are among the most prevalent and serious health issues globally. Several algorithms have been developed to detect brain tumors, enhancing the accuracy and efficiency of diagnoses.

Magnetic Resonance Imaging (MRI) is one of the most advanced technologies for brain tumor diagnosis, offering superior image quality compared to other modalities. However, interpreting MRI images is a complex task that often requires expertise and experience. In some cases, subtle abnormalities may go undetected, potentially delaying treatment and compromising patient outcomes. Image processing techniques are crucial in this context, aiding medical professionals in analyzing and extracting critical information from MRI images, including tumor segmentation, localization, characterization, and classification. These techniques are further enhanced by deep learning algorithms, which improve the accuracy and efficiency of disease detection. By harnessing these advanced algorithms, it is possible to detect diseases at an early stage, thereby improving patient prognosis and overall healthcare outcomes.

The field of medical imaging has a rich history, beginning with the discovery of X-rays by Wilhelm Conrad Roentgen in 1895. Since then, the field has evolved significantly, introducing various imaging modalities such as computed tomography (CT), magnetic resonance imaging (MRI), positron emission tomography (PET), and ultrasound. Each modality has unique advantages and specific applications in diagnosing different medical conditions. MRI, in particular, has revolutionized neuroimaging due to its high spatial resolution and ability to differentiate between various soft tissues in the brain, making it an invaluable tool for detecting and characterizing brain tumors.

The advent of artificial intelligence (AI) and machine learning (ML) has further transformed medical imaging. Traditional image analysis techniques often relied on manual segmentation and feature extraction, which could be time-consuming and prone to human error. AI and ML, particularly deep learning algorithms, have automated these processes, enabling faster and more accurate image analysis. Convolutional neural networks (CNNs) have been especially successful in medical image classification tasks, learning complex patterns and features from large datasets, making them ideal for applications such as tumor detection and classification.

This thesis explores the application of an advanced artificial intelligence model known as EfficientNetB7 in the classification of brain tumors. Trained on a vast dataset, EfficientNetB7 demonstrates exceptional capability in distinguishing various subtypes of brain tumors with remarkable accuracy. Leveraging deep learning algorithms, this model excels in recognizing intricate structural and visual patterns, enabling the precise classification of tumors in MRI images. Key features such as tumor type, size, and location are considered in the classification process (Khan, P., 2019). The integration of the Convolutional Block Attention Module (CBAM) into the model aids in effectively processing the rich information presented in MRI scans, thereby enhancing classification accuracy. The emphasis on relevant features and suppression of background noise allows for the emergence of critical details vital for diagnosis.

The primary focus of this thesis is the differentiation of distinct subtypes of brain tumors, namely glioma, meningioma, pituitary tumors, and non-tumorous areas. Specialized training and validation datasets are meticulously crafted for each category, rigorously testing the model's accuracy and reliability. Notably, the preprocessing stage of the datasets incorporates various image enhancement techniques using 'ImageDataGenerator,' including rotation, shifting, scaling, and other manipulations. These techniques enhance the model's adaptability to real-world variations while mitigating the risk of overfitting, thereby elevating the model's overall accuracy (Raj, S., Kumar, U., 2019).

In addition to focusing on brain tumor classification, this thesis explores the broader implications of machine learning in medical diagnostics. Integrating AI into healthcare presents challenges such as data privacy, the need for large annotated datasets, and the interpretability of AI models. Addressing these issues is crucial for ensuring the safe and effective deployment of AI technologies. Ethical considerations also play a significant role, as AI systems must be designed and implemented to prioritize patient welfare and adhere to regulatory standards.

The benefits of AI in healthcare extend beyond imaging. Predictive analytics, personalized medicine, and virtual health assistants are just a few examples of how AI is reshaping the medical landscape. Predictive analytics can identify patients at risk of developing certain conditions, allowing for early intervention and preventive care. Personalized medicine leverages AI to tailor treatments to individual patients based on their unique genetic makeup and medical history.

In summary, this thesis exemplifies how machine learning and artificial intelligence can revolutionize medical diagnosis. These cutting-edge technologies transcend traditional methods, opening new horizons for medical research and applications. Focusing on brain tumor diagnosis, this work comprehensively addresses this innovative approach's potential, underscoring its significance in early disease detection and swift therapeutic interventions.

The future of medical imaging and AI looks bright, with continuous advancements promising more accurate diagnostic tools. Collaboration between scientists, healthcare professionals, and regulators is key to responsibly integrating AI into clinical practice. This thesis showcases EfficientNetB7 and CBAM's effectiveness in brain tumor classification, underscoring AI's potential to revolutionize healthcare.

2. LITERATURE REVIEW

In recent years, deep learning techniques have seen a surge in interest due to their excellent performance in various medical imaging tasks, including the detection and categorization of brain tumors. These techniques employ artificial neural networks to successfully and automatically extract challenging features from diverse data, such as MRI scans. Important deep learning techniques have been utilized to improve the reliability of brain tumor detection. Initially, data augmentation techniques were used, which are vital for enhancing the accuracy of the data model. Data augmentation includes various methods such as rotation, shifting, scaling, and mirroring to produce enhanced images.

Efficient-Net models are popular due to their efficiency in terms of computational resources and parameter complexity and are widely used in the field of medical image analysis. Specifically, EfficientNetB7, which is tailored for complex image classification tasks, has been preferred for its high accuracy in various medical imaging applications, including the detection of brain tumors. To further improve the model's performance and accuracy, techniques such as K-Fold cross-validation and fine-tuning were utilized. This approach is frequently employed to enhance the generalizability and accuracy of the brain tumor detection model. This technique involves dividing the dataset into K subsets and performing training and validation K times, each time using a different subset. By evaluating the model's performance across multiple folds, it not only reduces the risk of overfitting but also increases the accuracy rate and is among the commonly used techniques (Prusty, S., Patnaik, S., Dash, S., 2023).

Finally, the fine-tuning technique is employed. Fine-tuning the deep learning model is a crucial step in increasing its accuracy. By making certain lower-level layers trainable, the model can discern extremely complex and subtle features. This provides significant benefits, especially in situations such as classifying brain tumors (Bzdok, D., 2018).

2.1. Types of Brain Tumors

Gliomas, meningiomas, and pituitary tumors are among the most common types of brain tumors, each with distinct clinical features and treatment strategies.

Gliomas

Gliomas originate from glial cells in the brain and spinal cord, forming most of the primary brain tumors. They are categorized into various types, such as astrocytomas, oligodendrogliomas, and ependymomas, based on their cellular characteristics. These tumors can occur in different parts of the brain (Figure 2.1) and are graded based on their malignancy levels, with some types like glioblastomas known for their rapid growth and challenging treatment prospects (Waggoner, 1937).

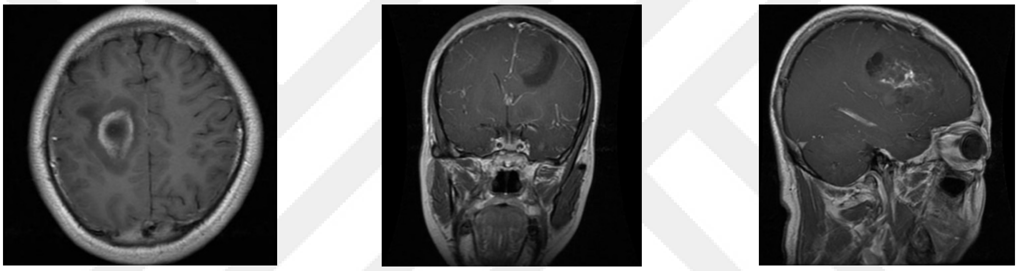

Figure 2.1 Glioma, Coronal, Saggital

Meningiomas

Meningiomas are typically benign tumors that develop from the meninges surrounding the brain. They represent the second most common type of brain tumor in adults and are generally treated with surgical removal. Although most meningiomas are benign, some may grow more aggressively and require additional treatments such as radiotherapy or pharmacotherapy (Black, 1995). These tumors can be visualized in different parts of the brain as shown in Figure 2.2.

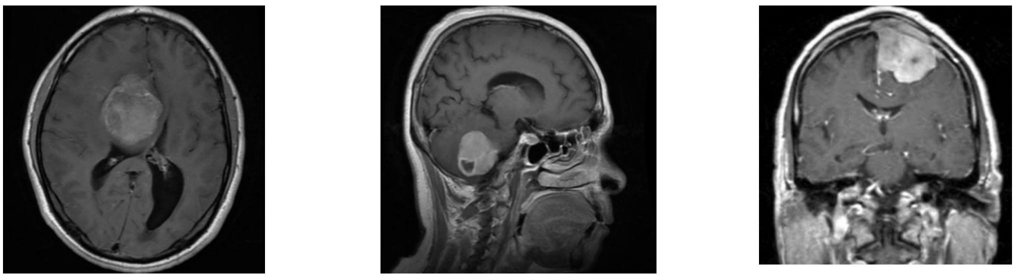

Figure 2.2. Meningioma, Saggital, Coronal

Pituitary

Pituitary tumors, often benign, originate from the anterior lobe of the pituitary gland and can affect the body's hormone regulation. While many pituitary tumors are asymptomatic, larger growths can lead to symptoms like headaches, vision problems, and hormonal imbalances. Treatment options include surgical removal, radiotherapy, and hormonal therapy to manage the tumor and alleviate symptoms (Black, 1995). These tumors can be visualized in different parts of the brain as shown in Figure 2.3. These tumors significantly influence brain and neurological function, necessitating varied management strategies based on their characteristics and progression.

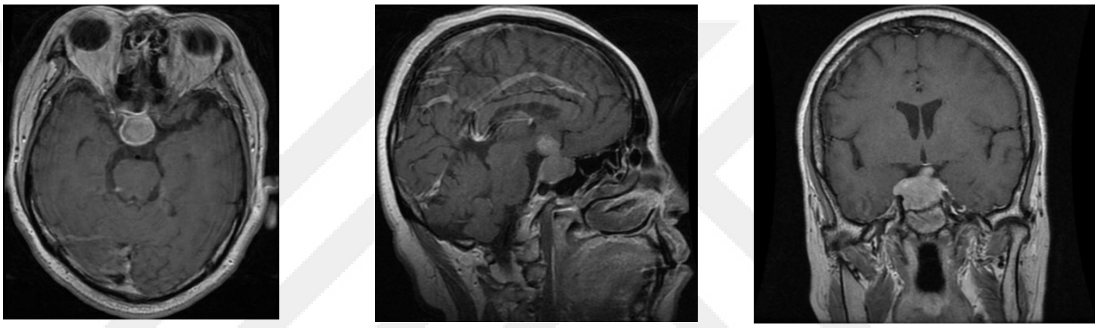

Figure 2.3. Pituitary, Saggital, Coronal

2.2. Classification of Brain Tumors

The classification of brain tumors using data from medical imaging represents a significant hurdle in neuro-oncological research. The precise identification of such tumors is paramount in devising treatment strategies, prognostic assessments, and managing patient care. The field has witnessed a paradigm shift with the advent of deep learning technologies, which have introduced methods that not only parallel but occasionally surpass the analytical capabilities of human experts. A notable benefit of leveraging deep learning in the domain of brain tumor classification is the streamlining of the diagnostic procedure. This innovation cuts down the time required for diagnosis and facilitates instantaneous support for decision-making in medical environments. In addition, these advanced models can be assimilated into current medical imaging systems, providing a smooth and synergistic upgrade to conventional diagnostic methods (Nerurkar, N., 2017).

2.3. The Importance of MRI Imaging

Magnetic Resonance Imaging (MRI) has become an indispensable tool in contemporary medical diagnostics, providing a non-invasive window into the intricate structures of the human body. Its unmatched ability to visualize internal tissues with high contrast and detail without the use of ionizing radiation sets MRI apart from other imaging modalities. MRI's unique advantage lies in its versatility. It offers an array of imaging sequences, each capable of highlighting different tissue properties, making it particularly valuable in the diagnosis and monitoring of neurological conditions, soft tissue pathologies, and musculoskeletal disorders. In neurology, MRI's superior soft tissue contrast resolution allows for the detailed visualization of brain anatomy, facilitating the early detection and classification of brain tumors a process pivotal to patient outcome and treatment efficacy (Alan, S., 2019).

2.4 Evolution of the EfficientNet Architecture

The EfficientNet architecture is one of the latest advancements in deep learning. It offers a structure that has made significant contributions to the field of image classification. EfficientNet fundamentally uses an automatic scaling method that enhances the model's performance while maintaining computational efficiency. It encompasses a variety of model structures from B0 to B7, each tailored for specific tasks. The aim of these models is to achieve higher accuracy rates compared to previous architectures and to yield better results with more complex data (Baraiya, N., 2016).
The importance of scaling neural networks is significant in this context. EfficientNet facilitates the scaling of depth, width, and resolution, establishing a strong balance among them. This allows the model to achieve high accuracy and performance, even with complex data sets. It is considered one of the most advanced methods to date. This model has become one of the groundbreaking architectures in deep learning.

2.5 Attention Mechanisms in Deep Learning

In the realm of artificial intelligence, particularly in deep learning, attention mechanisms play a crucial role in enhancing the performance of models by directing their focus to the most pertinent elements of the input data. These mechanisms, inspired by the selective attention in human cognition, have led to significant advancements in various areas such as natural language processing, computer vision, and medical image analysis.

2.5.1 Fundamental concepts and evolution of attention mechanisms

Initially conceptualized in human cognitive sciences, attention mechanisms have been transposed into neural network architectures to manage the limited computational resources more effectively. The primary function of attention in neural networks is to dynamically highlight important features while downplaying others, thereby improving the efficiency and accuracy of learning processes. The survey by Correia and Colombini (2021) provides an extensive overview of the development and integration of attention mechanisms in deep learning, detailing the varied architectures and their impacts across multiple domains (Correia & Colombini, 2021).

2.5.2 Attention mechanisms in medical image analysis

In the field of medical image analysis, attention mechanisms have revolutionized diagnostic processes by allowing networks to focus on regions of interest in medical images, thereby enhancing the precision of diagnostics. Li et al. (2023) discuss the state-of-the-art attention-based deep learning models that have achieved remarkable success in identifying and classifying pathologies in medical images, pointing towards their growing importance in supporting clinical decisions (Li et al., 2023).

2.5.3 Types of attention mechanisms and their applications

There are several types of attention mechanisms, each tailored to different aspects of the data:

Channel attention focuses on what is meaningful along the channel dimension, enhancing feature representation power.

Spatial attention concentrates on where important features are, improving the localization capabilities of models.

Self-attention and multi-head attention enable the model to weigh the importance of different parts of the input data relative to each other.

Zhu et al. (2021) provide a focused review on how these attention mechanisms are incorporated into convolutional neural networks (CNNs) for tasks like image super-resolution, illustrating their effectiveness in refining feature extraction processes (Zhu et al., 2021).

While current models with attention mechanisms are effective, there remains substantial scope for improvement and innovation. Future research may explore the integration of more complex attention models that can handle increasingly large and unstructured datasets, or the development of attention mechanisms that offer greater interpretability of model decisions, a critical aspect for applications in fields requiring explainability, such as healthcare and autonomous driving.

Attention mechanisms have profoundly impacted the field of deep learning, offering sophisticated tools that allow neural networks to focus on the most informative parts of their input data. This capability not only improves model performance but also enhances their applicability to real-world problems, where the ability to discern subtle nuances can be crucial. The continuous evolution of attention mechanisms promises even more sophisticated and effective models in the future.

2.6 Data Augmentation and Preprocessing Techniques

Data augmentation techniques are among the most crucial operations to enhance the success rate of a model. They increase the quality and diversity of limited data. Various methods such as rotation, scaling, and horizontal and vertical shifting are applied, which help to resolve the issue of overfitting. In preprocessing techniques, the aim is to prepare the data set for processing. The color scale is adjusted, noise is reduced, and thus the model can learn more effectively (Sacdeva, J., 2013).

3. MATERIALS AND METHODS

3.1. Dataset and Labeling

The dataset analyzed in this study consists of 7,023 magnetic resonance (MR) images. These images have been pre-classified into four distinct categories to detect the presence and type of brain tumors: “glioma,” “meningioma,” “pituitary,” and “notumor.” This categorization, already present in the dataset obtained from the Kaggle platform (link), aims to facilitate the use of images in medical diagnostics. To optimize the use of the dataset, the separation into training and testing sets (Figure 3.1) was automatically executed using a Python script. Through this process, the data were divided into 80% for training and 20% for testing, thereby enabling an effective evaluation of the model's performance on real-world data.

The dataset's tumor classes (Figure 3.1) include glioma, meningioma, pituitary, and notumor.

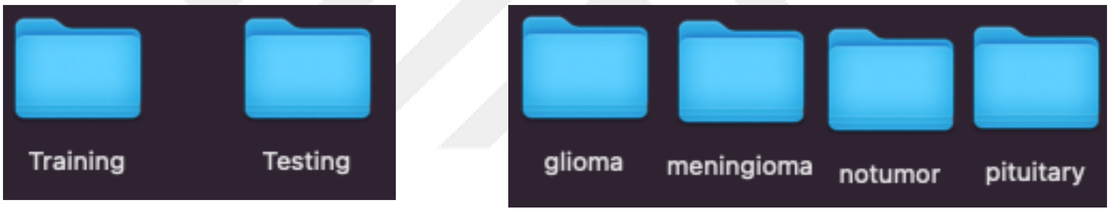

Figure 3.1. Dataset tumor classes

3.2 Pre-processing Steps

3.2.1 Mr images cropping

The dataset was subjected to a series of preprocessing techniques before being introduced into the training process. Among these preprocessing steps, cropping was initially applied to optimize the performance of the dataset. Through the cropping process (Figure 3.2), the meaningless black areas at the top and bottom of the MR images were removed. This procedure prevented the model from focusing on irrelevant parts, thereby enabling it to concentrate more effectively on meaningful and diagnostically rich image sections. Consequently, the overall recognition and processing performance of the model was aimed to be enhanced.

```python
import cv2
import os

def find_nonblack_areas(image):
    _, thresh = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    topmost = float('inf')
    bottommost = 0
    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)
        topmost = min(topmost, y)
        bottommost = max(bottommost, y+h)
    return topmost, bottommost

def crop_nonblack_areas(image):
    top, bottom = find_nonblack_areas(image)
    bottom = bottom if bottom != 0 else image.shape[0]
    return image[top:bottom, :]

def process_image(image_path, output_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    cropped_image = crop_nonblack_areas(image)
    cv2.imwrite(output_path, cropped_image)
    print(f"Processed: {image_path} -> {output_path}")

def process_directory(input_dir, output_dir):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    for item in os.listdir(input_dir):
        input_path = os.path.join(input_dir, item)
        output_path = os.path.join(output_dir, item)
        if os.path.isdir(input_path):
            process_directory(input_path, output_path)
        elif input_path.lower().endswith(('.png', '.jpg', '.jpeg')):
            process_image(input_path, output_path)

input_folder = '/Users/uygarcankat/Desktop/MRImages'  # Ana girdi klasörü
output_folder = '/Users/uygarcankat/Desktop/ProcessedImages' # Ana çıktı klasörü
process_directory(input_folder, output_folder)
```

Figure 3.2. Code for cropping non-black areas in mr images

The provided Python script is crafted for image processing tasks utilizing the OpenCV library. It is specifically designed to detect non-black regions in images and crop the images to focus on these areas.

Initially, the function find_nonblack_areas is defined. This function takes an image as input and performs thresholding to convert it to a binary format. Subsequently, it detects contours, which represent non-black areas. By determining the highest and lowest points for each contour, it calculates the region of the image that needs to be cropped.

The function crop_nonblack_areas employ the find_nonblack_areas function to locate the highest and lowest non-black areas in the image and crops the image accordingly.

The process_image function receives a path to an image, loads the image in grayscale, calls the crop_nonblack_areas function to crop the image, and saves the result to the specified output path (Figure 3.3).

The process_directory function takes paths to an input and output directory. It processes each file in the input directory using the process_image function and saves the results to the output directory. This function checks and creates necessary files and directories in the given paths.

In the final part of the script, the process_directory function is invoked using the specified input and output directory paths, thus processing all images in the given input directory and saving the results to the output directory.

The code presents an approach that automates image processing workflows and facilitates rapid and efficient processing of large datasets. Such tools are particularly beneficial in medical imaging and machine learning applications, saving time in preprocessing steps and reducing the workload for researchers and practitioners (Figures 3.3).

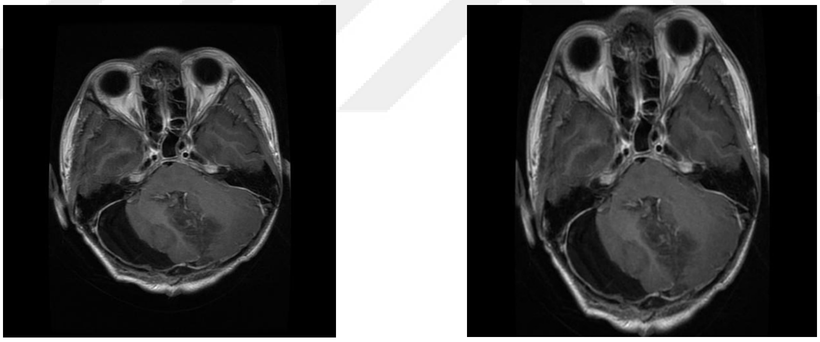

Figure 3.3. Original and cropping image

3.2.2 Resizing of brain regions

By resizing MR images to standardize the size of the brain region (Figure 3.4), the training of machine learning models is optimized, ensuring a more focused and meaningful information processing during the training process. This method facilitates the model's learning of relevant anatomical structures, thereby increasing training efficiency and improving the overall accuracy and performance of the model. Standardized image sizes enable more consistent and effective training of the model, resulting in faster training processes and higher diagnostic accuracy.

```python
import cv2
import numpy as np
import os

def find_largest_contour(image):
    """
    This function finds the largest contour in the image.
    Assuming the largest contour in the MR image is the brain.
    """
    # Convert to binary image
    _, thresh = cv2.threshold(image, 1, 255, cv2.THRESH_BINARY)
    # Find contours
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    # Get the largest contour
    largest_contour = max(contours, key=cv2.contourArea)
    return largest_contour

def resize_image(image, contour, target_height=1500):
    """
    This function resizes the image so that the height of the brain area is 1500 pixels.
    """
    x, y, w, h = cv2.boundingRect(contour)
    brain_area = image[y:y+h, x:x+w]
    # Calculate the ratio to resize by
    ratio = target_height / h
    new_width = int(w * ratio)
    # Resize the image
    resized_image = cv2.resize(brain_area, (new_width, target_height), interpolation=cv2.INTER_CUBIC)
    return resized_image

def process_image(file_path, output_dir):
    """
    This function processes an individual image.
    """
    image = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
    contour = find_largest_contour(image)
    resized_image = resize_image(image, contour, target_height=1500)
    save_path = os.path.join(output_dir, os.path.basename(file_path))
    cv2.imwrite(save_path, resized_image)
    print(f"Processed {os.path.basename(file_path)}")

def process_directory(input_dir, output_dir):
    """
    This function processes all images in the input directory and its subdirectories.
    """
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for entry in os.scandir(input_dir):
        if entry.is_dir():
            process_directory(entry.path, os.path.join(output_dir, entry.name))
        elif entry.is_file() and entry.name.lower().endswith(('.png', '.jpg', '.jpeg')):
            process_image(entry.path, output_dir)

# Example usage
input_folder = '/Users/uygarcankat/Desktop/yeni/pituitary_tumor'  # Replace with your actual input folder path
output_folder = '/Users/uygarcankat/Desktop/yeni/output'  # Replace with your actual output folder path
process_directory(input_folder, output_folder)
```

Figure 3.4. Code for resizing mr to standart height

Finding the largest contour: The find_largest_contour function converts the image to a binary format and identifies its contours. In MR images, the brain usually possesses the largest contour, which is thus detected.

Image resizing: The resize_image function takes the identified largest contour (the brain area) and resizes it to a specified height (default is 1500 pixels), adjusting the width proportionally to maintain aspect ratio.

Image processing and saving: The process_image function reads an image file, performs the above-mentioned processes, and saves the result to the specified output directory.

Directory processing: The process_directory function scans through all suitable image files and subdirectories in each folder, applying the described processing steps to each image.

3.2.3 MR images resize with padding

In this preprocessing step, MR images were automatically resized and padded (Figure 3.5). The process adjusted the images to specified standard dimensions while preserving the integrity of the original content and adding black padding to unnecessary areas. This method ensures the homogenization of the dataset and offers several significant advantages:

Standardization: Ensuring images have a uniform resolution allows the model to process the data more effectively, enhancing its recognition capability.

Focus optimization: Padding enables the model to concentrate on meaningful areas, improving the accuracy of medical diagnoses.

Processing efficiency: Uniform dimensions optimize computational processes, leading to cost and time savings.

Generalization ability: Consistency in training enhances the model's ability to recognize new and varied images, playing a crucial role in improving the accuracy and reliability of the training process.

These methods are instrumental in enhancing the accuracy and reliability of machine learning-based MR image analysis.

```python
def resize_with_padding(image, target_width=2000, target_height=2000):
    """
    Resize the image to the target size with padding while keeping the brain area unchanged.
    """
    # Calculate padding sizes and ensure they are non-negative
    padding_vertical = max((target_height - image.shape[0]) // 2, 0)
    padding_horizontal = max((target_width - image.shape[1]) // 2, 0)

    # Apply padding
    padded_image = cv2.copyMakeBorder(image, padding_vertical, padding_vertical,
                                      padding_horizontal, padding_horizontal,
                                      cv2.BORDER_CONSTANT, value=[0, 0, 0])

    return padded_image

def process_image(file_path, output_dir):
    """
    This function processes an individual image.
    """
    image = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
    padded_image = resize_with_padding(image)
    save_path = os.path.join(output_dir, os.path.basename(file_path))
    cv2.imwrite(save_path, padded_image)
    print(f"Processed {os.path.basename(file_path)}")

def process_directory(input_dir, output_dir):
    """
    This function processes all images in the input directory and its subdirectories.
    """
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for entry in os.scandir(input_dir):
        if entry.is_dir():
            process_directory(entry.path, os.path.join(output_dir, entry.name))
        elif entry.is_file() and entry.name.lower().endswith(('.png', '.jpg', '.jpeg')):
            process_image(entry.path, output_dir)

# Example usage
input_folder = '/Users/uygarcankat/Desktop/ThesisDocument/ProcessedImagesResized'  # Replace with your actu
output_folder = '/Users/uygarcankat/Desktop/ThesisDocument/ProcessImageFinallyResized'  # Replace with your
```

Figure 3.5. Code for resizing mr images with padding

The code utilizes the OpenCV library to perform image resizing and padding operations, aiming to resize images to target dimensions while maintaining the integrity of the brain region without alteration.

The resize_with_padding function rescales an image to specified target dimensions (default 2000×2000 pixels), adding equal amounts of black padding to maintain the original aspect of the image while standardizing its size. This ensures that the central region of the image remains unaltered.

The process_image function reads an image from a specified file path, processes it using the resize_with_padding function, and saves the result to a designated output directory. The completion of the process is indicated in the console, confirming the image has been processed and stored.

The process_directory function iterates through all files and subdirectories within an input directory. If a file is an image (with .png, .jpg, or .jpeg extensions), it is processed using the process_image function. This method facilitates the batch processing of all images in the specified directory, with the results saved in the designated output directory.

This code snippet is crucial for the automation and standardization of image processing algorithms. It particularly provides efficiency and consistency in preprocessing large-scale image datasets, contributing to time savings and workflow optimization in scientific research endeavors.

In summary, the preprocessing steps of cropping, resizing, and padding are crucial for optimizing the dataset for model training. Initially, the cropping process removes unnecessary black areas at the top and bottom of MR images, thereby enhancing the quality of the information extracted and clarifying the focus area for the model. This facilitates the model's learning of more relevant and meaningful features throughout the training process. Subsequently, the resizing step adjusts the images to a specific size, ensuring a uniform structure of the dataset and allowing the model to process each data point consistently. Finally, padding adjusts the images to a standard size (e.g., 2000×2000 pixels), enabling the model to work with fixed-size inputs and making the training process more efficient. Consequently, this sequential preprocessing approach boosts the model's overall performance, accelerates the training process, and contributes to the production of more accurate results by the machine learning algorithm.

3.2.4 Grouping mr images according to angles

Magnetic Resonance Imaging (MRI) is a critical technology used in the diagnosis of brain tumors. These images are obtained in three primary planes: axial, sagittal, and coronal. Each plane offers a distinct perspective of brain anatomy, providing a rich source of information for tumor detection. However, an imbalanced representation of these planes in the training dataset can lead the learning algorithm to prioritize the morphological features of the imaging angle over the actual presence of a tumor. This can result in the model making false positive or false negative detections based on the brain's shape.

Therefore, grouping MRI images according to their imaging planes (Figure 3.6) enables the model to learn the unique characteristics of each view, thus becoming more sensitive to the actual tumor features. This categorization enhances the accuracy of tumor detection and improves the model's generalization ability across images obtained from different angles.

```
# Veri artırma
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=15,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1, s
                                   hear_range=0.1,
                                   zoom_range=0.1,
                                   horizontal_flip=True)

validation_datagen = ImageDataGenerator(rescale=1./255)

# Veri yükleyiciler
train_generator = train_datagen.flow_from_directory(train_data_dir, target_size=(img_width, img_height),
                                                    batch_size=32,
                                                    class_mode='categorical')

validation_generator = validation_datagen.flow_from_directory(validation_data_dir,
                                                              target_size=(img_width, img_height),
                                                              batch_size=32,
                                                              class_mode='categorical')

# Model oluşturma
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_height, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(3, activation='softmax')
])

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Model eğitimi
model.fit(train_generator,
          steps_per_epoch=train_generator.samples // train_generator.batch_size,
          epochs=20,
          validation_data=validation_generator,
          validation_steps=validation_generator.samples // validation_generator.batch_size

          )

# Sınıf indekslerini kaydet
class_indices = train_generator.class_indices
with open('/Users/uygarcankat/Desktop/class_indices.json', 'w') as file:
    json.dump(class_indices, file)

# Modeli kaydet
model.save('/Users/uygarcankat/Desktop/modelim.h5')
```

Figure 3.6. Code for grouping mr images by plane

The utilization of ImageDataGenerator for data augmentation facilitates the model to learn from various image transformations, thereby enhancing its generalization capability. Techniques such as rotation, shift, shear, zoom, and horizontal flipping, in addition to rescaling, are applied to the training dataset to enrich the diversity of the training input, which aids in mitigating overfitting and promoting a more robust learning outcome.

The flow_from_directory function constructs data loaders (train_generator and validation_generator) for the training and validation datasets. These loaders supply the images in batches, which are then used to train the model systematically.

The deep learning model is structured using the Sequential API, incorporating Conv2D layers for convolutional operations to extract features from the images and MaxPooling2D layers to reduce the spatial dimensions, thereby condensing the extracted features. The network is further composed of Dense layers to perform classification based on the extracted and condensed features.

The model is compiled with categorical cross-entropy as the loss function and 'adam' as the optimizer, which is a widely recognized method for training deep learning models due to its efficiency in managing learning rates and convergence.
Training is conducted via the fit method, with the model undergoing specified epochs, during which it learns to distinguish between the classes based on the features from the training set and validates its performance on the validation set.

Upon completion of the training process, the class indices (class_indices), which the model has learned, are stored. This dictionary maps the learned classes to their respective indices, encapsulating the classification schema the model has deciphered.

The primary aim of this code segment is to establish, train, and preserve a model for image classification, which is pivotal in extracting meaningful patterns from the dataset and categorizing images accordingly (Figure 3.7). This enhances the repository of computational tools available for automated image analysis, paving the way for their application in real-world scenarios and subsequent studies.

```
# Modeli yükle
model = load_model('/Users/uygarcankat/Desktop/yeni/ModelCreatingCodeAçılarınagöre/modelim.h5')

# class_indices'i yükle
with open('/Users/uygarcankat/Desktop/class_indices.json', 'r') as file:
    class_indices = json.load(file)

# Sınıf etiketlerini tersine çevir (indeks: etiket)
class_labels = {v: k for k, v in class_indices.items()}

# Görüntüleri içeren input klasör
input_folder = '/Users/uygarcankat/Desktop/yeni/YeniKaggleVeriSeti/Testing/pituitary_tumor'

# Sınıflandırma sonuçlarını kaydedeceğiniz output klasörler
output_folders = {k:'/Users/uygarcankat/Desktop/yeni/YeniKaggleVeriSeti/Testing/output' + k for k in class_labels.values()}

# Output klasörlerini oluştur (eğer mevcut değilse)
for folder in output_folders.values():
    if not os.path.exists(folder):
        os.makedirs(folder)

# Görüntü boyutlarını tanımla
img_width, img_height = 150, 150

# Tahminler ve kaydetme
for img_file in os.listdir(input_folder):
    if img_file.lower().endswith(('.png', '.jpg', '.jpeg')):
        img_path = os.path.join(input_folder, img_file)
        img = image.load_img(img_path, target_size=(img_width, img_height))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0) / 255.0

        prediction = model.predict(img_array)
        predicted_class_index = np.argmax(prediction, axis=1)[0]
        predicted_class_label = class_labels[predicted_class_index]

        target_folder = output_folders[predicted_class_label]
        shutil.copy(img_path, target_folder)
```

Figure 3.7 Code for predicting and grouping mr images by plane

The class_indices.json file is opened to load the class indices into a dictionary. This file contains the class labels and their corresponding indices used during the model's training.

The class labels and indices are then swapped, allowing the identification of the label corresponding to the index obtained in the prediction result (class_labels).

The input folder, where the images are located, and the output folders, where the classification results will be stored, are defined.

Output folders are created if they do not already exist.

The image dimensions (img_width, img_height) are set, corresponding to the input size expected by the model.

For each image in the input folder:

- The file is checked to ensure it is in an appropriate format (png, jpg, jpeg).
- The image is loaded and resized to the dimensions required by the model.
- The image is converted into a numpy array and preprocessed for the predict method (dimension expansion and normalization).

- The model performs a prediction on the image, and the index of the class with the highest probability is retrieved.
- This index is used to obtain the corresponding class label using the class_labels dictionary.
- The output folder for the predicted class is identified, and the image is copied to this folder.

Consequently, a model trained on an angularly grouped dataset (Figures 3.8) provides higher accuracy and reliability in detecting brain tumors. This enhances the precision of medical diagnoses, improving the quality and outcomes of treatment decisions for patients.

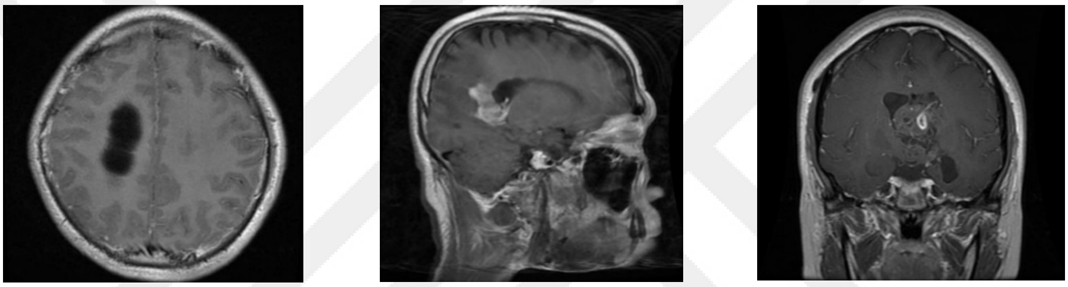

Figure 3.8. Different planes for grouping

Magnetic Resonance (MR) imaging classes include glioma, meningioma, pituitary, and notumor, with each class comprising images taken in axial, sagittal, and coronal planes. These images are organized into a total of twelve distinct folders (Figure 3.9).

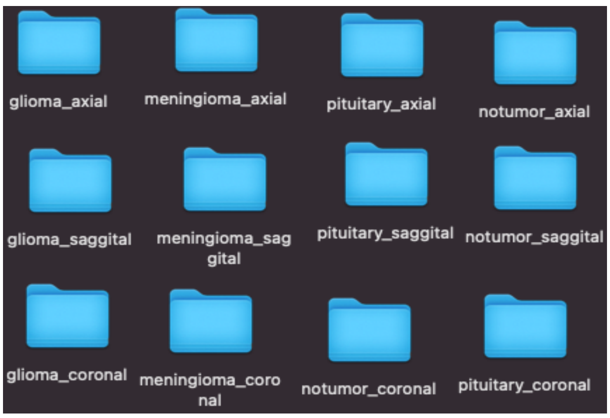

Figure 3.9. Tumor classes by plane

Magnetic Resonance (MR) imaging evaluates different classes such as glioma, meningioma, pituitary, and notumor across three main planes (Figure 3.10): axial, sagittal, and coronal. Initially, separate folders were created for each MR image class and imaging angle combination, leading to a fragmented structure with twelve distinct folders (Figures 3.11, 3.12, and 3.13).

Figure 3.10. Dataset by plane

Figure 3.11. Saggital classes

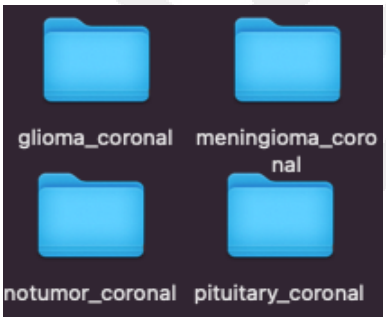

Figure 3.12. Coronal classes

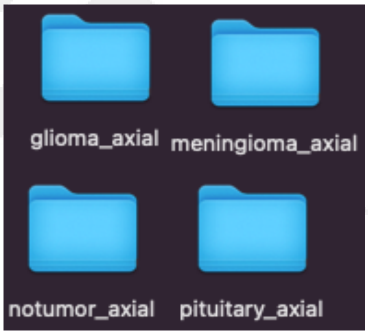

Figure 3.13. Axial classes

Rationale for reorganization

To facilitate data access and optimize the training process of machine learning models, a systematic reorganization was undertaken. In this new arrangement, all images from each imaging plane were regrouped into corresponding axial, sagittal, and coronal folders. This reorganization serves multiple purposes:

Facilitating access: Consolidating images by plane simplifies data retrieval for researchers and algorithms, accelerating both the training and analysis phases.

Enhancing model training: Grouping images according to their respective planes enables more focused and effective learning of the specific features present in each plane. This targeted learning approach allows for a detailed examination of the distinct anatomical and pathological characteristics revealed by each plane.

Improving generalization and accuracy: Training with well-organized data allows the model to specialize in recognizing and generalizing features across different imaging angles, thereby enhancing diagnostic accuracy.

Balancing training Sets: This method also helps in identifying and addressing potential imbalances within the dataset. When the number and type of images in each plane are clearly visible, imbalances can be corrected, leading to more balanced training sets and thus increasing the reliability of the model outcomes.

Customized model training: Grouping enables the development of customized models tailored to recognize specific features of images from different planes. This specificity can aid in boosting diagnostic performance for types of images and conditions, especially valuable in medical settings where accuracy is critical.

The strategic grouping of MR data by imaging planes not only simplifies the operational aspects of managing large datasets but also significantly enhances the functional capabilities of diagnostic models. This approach ensures that the nuances of each imaging plane are thoroughly learned and represented in the diagnostic process, leading to more precise and reliable medical evaluations.

3.2.5 Sagittal mr image standardization

To enhance the efficiency of the training process, a strategy was adopted to ensure that all sagittal MR images within the dataset are oriented in the same direction. This approach aims to optimize the consistency and comparability of analyses and learning algorithms applied to the images. Standardizing the orientation of the image set in this manner facilitates the model's ability to learn more rapidly and effectively during the training phase, as it reduces the variations requiring model adaptation. By standardizing the orientation of each image in the training dataset, this method allows the learning model to better grasp the fundamental characteristics of the dataset, thereby improving its classification performance. Therefore, the process of orienting MR images (Figure 3.14) can be considered a critical preprocessing step that enhances the efficiency and quality of outcomes in the training process.

```python
def is_image_left_facing(image_path):
    img = Image.open(image_path).convert('L')  # Görüntüyü gri tonlamaya çevir
    histogram = np.array(img.histogram())

    # Histogramı iki eşit parçaya böl
    mid_point = len(histogram) // 2
    left_side = histogram[:mid_point]
    right_side = histogram[mid_point:]

    # Sol tarafın toplamı sağ tarafa göre daha büyükse, görüntü sola bakıyordur
    return np.sum(left_side) > np.sum(right_side)

def flip_images_in_directory(input_dir, output_dir):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for subdir, dirs, files in os.walk(input_dir):
        for file in files:
            if file.lower().endswith(('.png', '.jpg', '.jpeg')):
                input_path = os.path.join(subdir, file)

                # Output klasör yapısını oluştur
                relative_path = os.path.relpath(subdir, input_dir)
                output_subdir = os.path.join(output_dir, relative_path)
                if not os.path.exists(output_subdir):
                    os.makedirs(output_subdir)

                output_path = os.path.join(output_subdir, file)

                if is_image_left_facing(input_path):  # Eğer görüntü sola bakıyorsa, çevir
                    img = Image.open(input_path)
                    flipped_img = img.transpose(Image.FLIP_LEFT_RIGHT)
                    flipped_img.save(output_path)  # Çevrilmiş görüntüyü kaydet
                else:
                    # Sağa bakan görüntüyü doğrudan kopyala
                    img = Image.open(input_path)
                    img.save(output_path)

# Giriş ve çıktı klasör yollarını buraya girin
input_folder = r'/Users/uygarcankat/Desktop/YENİ TEZ/xx'
output_folder = r'/Users/uygarcankat/Desktop/YENİ TEZ/output'

flip_images_in_directory(input_folder, output_folder)
```

Figure 3.14. Code for standardizing mr image orientation

3.3 Model Development for Distinct MR Imaging Angels

Research Scope:

In this study, separate models have been developed for axial, sagittal, and coronal MR imaging planes. Each plane uniquely reflects the anatomical structure from different perspectives, and the distinctive visual characteristics of these planes present specific challenges in the image recognition process. Therefore, deep learning models tailored to each plane have been developed to facilitate more accurate and effective diagnostic procedures, aligned with the features of each respective plane.

Benefits of Customized Modeling:

Focused training: Each model is trained on features unique to its respective plane, allowing for a less complex and more targeted training process. This approach not only reduces the training duration but also enhances the models' generalization capabilities and accuracy.

Recognition of anatomical and pathological structures: The models are specifically developed to better recognize unique anatomical and pathological structures that emerge in different planes. This leads to improved diagnostic performance for specific types of images and enhances overall diagnostic success.

Optimization and Hyperparameter Adjustments:

Fundamental similarities and differences: The models developed for the three distinct MR imaging angles share basic structural similarities. However, there are variations in certain hyperparameters during the model optimization process.

Strategic adjustments: Specifically, parameters such as learning rates used during the initial training phase and those applied during the fine-tuning stage are customized according to the specific requirements of each model. These differences are strategically implemented to ensure optimal performance, tailored to the unique characteristics of each imaging angle. Adjustments in hyperparameters are crucial for enhancing the models' accuracy and generalization capabilities.

Ultimately, this strategic modeling approach significantly enhances efficiency and accuracy in the field of medical imaging and diagnosis. By fully leveraging the potential of deep learning to recognize and interpret complex medical images, this approach supports the potential to revolutionize the field.

In machine learning applications, ensuring reproducibility of results is crucial. A specific seed_value has been used to standardize the generation of random numbers across essential libraries such as Python's built-in random module, NumPy, and TensorFlow. These settings guarantee that all random processes, including model initialization weights, data shuffling, and other stochastic processes, produce consistent outcomes each time the code is run. As a result, this increases the consistency between model training sessions and experiments, thus ensuring the reliability of the results (Figure 3.15).

```
seed_value= 42

os.environ['PYTHONHASHSEED']=str(seed_value)
random.seed(seed_value)
np.random.seed(seed_value)
tf.random.set_seed(seed_value)
```

Figure 3.15. Seed code

To enhance the effectiveness and reliability of the model, datasets were organized in DataFrame format. This configuration has significantly facilitated the processing and management of data in a systematic manner. The use of DataFrames has greatly increased efficiency during data preprocessing and analysis phases. Additionally, this format has helped prevent data leakage, thereby obstructing the flow of information between training and testing sets and minimizing the risk of overfitting. Consequently, the utilization of DataFrames plays a critical role in enhancing model accuracy and generalization capabilities in machine learning projects (Figure 3.16).

```
# Veri setini DataFrame olarak oluşturma
def create_dataframe(data_dir, class_names):
    data = []
    for class_name in class_names:
        class_dir = os.path.join(data_dir, class_name)
        for fname in os.listdir(class_dir):
            if fname.endswith('.jpg') or fname.endswith('.png'):  # Görüntü dosyalarını kontro
                data.append({'filepath': os.path.join(class_dir, fname), 'label': class_name})
    return pd.DataFrame(data)

df = create_dataframe(data_dir, class_names)

labels = df['label'].map(class_names.index).values
```

Figure 3.16. Code for creating dataframe

Data augmentation

```
# Data Augmentation
train_datagen = ImageDataGenerator(
    rotation_range=40,  # Randomly rotate images within 40 degrees
    width_shift_range=0.2,  # Randomly translate images horizontally by
    height_shift_range=0.2,  # Randomly translate images vertically by
    shear_range=0.2,  # Randomly apply shearing transformations
    zoom_range=0.2,  # Randomly zoom in and out on images by up to 20%
    horizontal_flip=True,  # Randomly flip images horizontally
    vertical_flip=True,  # Randomly flip images vertically
    fill_mode='nearest',  # Fill in new pixels after a shift or rotatio
    brightness_range=[0.7, 1.3],  # Randomly vary the brightness of ima
    rescale=1./255  # Rescale pixel values from [0, 255] to [0, 1] rang
)

val_datagen = ImageDataGenerator(rescale=1./255)  # For validation data
```

Figure 3.17. Data augmantation code

This code segment outlines the implementation of data augmentation techniques using the ImageDataGenerator class, which enhances the diversity and generalizability of the dataset for a deep learning model (Figure 3.17). The augmentation parameters are carefully chosen to reflect realistic variations in medical imaging.

- **Rotation_range=40:** Introduces rotational variance, helping the model recognize patterns at different orientations.
- **Width_shift_range and height_shift_range:** Apply random translations to simulate variable patient alignment.
- **Shear_range=0.2:** Applies shearing transformations to mimic elastic deformations in tissues.
- **Zoom_range=0.2:** Ensures robustness to changes in the size of anatomical structures or tumors.
- **Horizontal_flip and vertical_flip:** Enable flips to account for the symmetrical nature of the human body.
- **Fill_mode='nearest':** Ensures new pixels introduced by transformations do not distort learning.
- **Brightness_range=[0.7, 1.3]:** Adjusts image brightness to accommodate variations in MRI scan intensities.
- **Rescale=1. /255:** Normalizes pixel values to a 0-1 range, facilitating stable and efficient training.

For the validation data, only rescaling is applied (rescale=1. /255) to ensure that model evaluation is performed on data that reflects the true nature of the input without additional augmentation noise.

Spatial attention and channel attention layer

In this study, Channel Attention and Spatial Attention layers have been integrated to enhance the image processing capabilities of a deep learning model. These attention mechanisms enable the model to focus on significant features and reduce background noise, thereby improving learning and recognition performance. These layers facilitate faster and more efficient learning during the model's training process, allowing for more accurate and generalizable predictions, reducing training errors, and particularly increasing the model's accuracy and sensitivity in complex visual tasks.

```
class ChannelAttention(Layer):
    def __init__(self, ratio=8, **kwargs):
        super(ChannelAttention, self).__init__(**kwargs)
        self.ratio = ratio
        self.shared_layer_one = None
        self.shared_layer_two = None

    def build(self, input_shape):
        channel = input_shape[-1]
        self.shared_layer_one = Dense(channel // self.ratio,
                                      activation='relu',
                                      kernel_initializer='he_normal',
                                      use_bias=True,
                                      bias_initializer='zeros')
        self.shared_layer_two = Dense(channel,
                                      kernel_initializer='he_normal',
                                      use_bias=True,
                                      bias_initializer='zeros')

        |
        super(ChannelAttention, self).build(input_shape)

    def call(self, inputs):
        avg_pool = GlobalAveragePooling2D()(inputs)
        avg_pool = Reshape((1, 1, -1))(avg_pool)
        avg_pool = self.shared_layer_one(avg_pool)
        avg_pool = self.shared_layer_two(avg_pool)

        max_pool = GlobalMaxPooling2D()(inputs)
        max_pool = Reshape((1, 1, -1))(max_pool)
        max_pool = self.shared_layer_one(max_pool)
        max_pool = self.shared_layer_two(max_pool)

        cbam_feature = Add()([avg_pool, max_pool])
        cbam_feature = Activation('sigmoid')(cbam_feature)

        return Multiply()([inputs, cbam_feature])
```

Figure 3.18. Channel attention code

Channel attention layer: This layer uses global information from each channel (color or feature maps) provided as input to determine which channels are more informative. Features obtained through average and maximum pooling methods are used to calculate the weights for each channel. As a result, the model focuses on relevant features and suppresses the less significant ones (Figure 3.18).

```
class SpatialAttention(Layer):
    def __init__(self, kernel_size=7, **kwargs):
        super(SpatialAttention, self).__init__(**kwargs)
        self.kernel_size = kernel_size
        self.conv = Conv2D(filters=1, kernel_size=kernel_size,
                           strides=1, padding='same',
                           activation='sigmoid',
                           kernel_initializer='he_normal',
                           use_bias=False)

    def call(self, inputs):
        avg_pool = Lambda(lambda x: K.mean(x, axis=-1, keepdims=True))(inputs)
        max_pool = Lambda(lambda x: K.max(x, axis=-1, keepdims=True))(inputs)
        concat = Concatenate(axis=-1)([avg_pool, max_pool])
        cbam_feature = self.conv(concat)

        return Multiply()([inputs, cbam_feature])


def attach_attention_module(net, attention_module='cbam_block'):
    if attention_module == 'cbam_block':
        net = ChannelAttention()(net)
        net = SpatialAttention()(net)
    else:
        raise ValueError("Not supported attention module: ", attention_module)

    return net
```

Figure 3.19. Spatial attention code

Spatial attention layer: This layer evaluates the spatial information in the input image. Maps created by combining average and maximum values are used to determine which regions contain more information. This method allows the model to more effectively assess spatial locations (Figure 3.19).

Combining these two attention layers enables the model to concurrently process both channel and spatial information, highlighting important features across both dimensions. This integration significantly enhances the model's performance in tasks such as classification and localization of various objects or patterns.

In conclusion, these attention mechanisms enable faster and more effective learning during the model's training phase, facilitating lower training errors and more accurate and generalizable predictions. These features significantly enhance the precision and accuracy of the model, especially in complex visual tasks, contributing to reliable outcomes in practical applications.

Configuration of the efficientNetB7 model with cbam

A deep learning model has been developed for complex visual tasks, utilizing the **EfficientNetB7** architecture to provide advanced image recognition capabilities. This architecture is pretrained with ImageNet weights, facilitating faster and more effective training of the model. The top part of the model (Figure 3.20) has been redesigned to meet specific classification needs and incorporates the Convolutional Block Attention Module (CBAM). The integration of **CBAM,** by adjusting attention foci in both spatial (Spatial Attention) and channel (Channel Attention) dimensions, helps the model to suppress irrelevant areas and focus only on important features. This significantly enhances the model's overall recognition performance.

The structure of the model includes layers such as **GlobalAveragePooling2D** and **BatchNormalization,** which stabilize the learning process and help prevent overfitting. **Dense layers** and **LeakyReLU** activation functions enable the model to learn complex relationships, while Dropout layers temporarily exclude randomly selected neurons (in this case, 20%) during each training step to prevent excessive reliance on a few neuron activations. This enhances the model's generalization ability as features learned during training are reinforced across multiple neural pathways, improving performance on validation or test sets.

L2 regularization used in each layer limits the magnitude of the weights, contributing to more generalized learning by the model. The final layer of the model features a Softmax activation function, producing a probability distribution for multi-class classification tasks, thus presenting predicted probabilities for each class.

The **Dropout technique** plays a critical role in preventing overfitting during the model's training process by randomly deactivating a specified proportion of neurons. This prevents the model from becoming overly dependent on the activations of a few neurons, thereby enabling it to operate with various neuron combinations.

The model compilation utilizes the **Adam optimizer** with a low initial learning rate. This is particularly beneficial during the fine-tuning stages, as it allows for more cautious weight updates and reduces the risk of getting stuck in local minima.

In conclusion, this part of model (Figure 3.20) employs advanced techniques and methods to achieve high-accuracy image classification performance. It is designed to work effectively on complex datasets, marked by rapid learning capabilities and high generalizability.

```
# Modeli oluştur
def build_model_with_cbam(num_classes):
    input_tensor = Input(shape=(224, 224, 3))
    base_model = EfficientNetB7(weights='imagenet',
                                include_top=False,
                                input_tensor=input_tensor)
    x = base_model.output

    # CBAM entegrasyonu
    x = attach_attention_module(x, 'cbam_block')

    x = GlobalAveragePooling2D()(x)
    x = BatchNormalization()(x)


    x = Dense(4096, kernel_regularizer=l2(0.001))(x)
    x = LeakyReLU(alpha=0.01)(x)
    x = Dropout(0.2)(x)
    x = BatchNormalization()(x)

    x = Dense(2048, kernel_regularizer=l2(0.001))(x)
    x = LeakyReLU(alpha=0.01)(x)
    x = Dropout(0.2)(x)
    x = BatchNormalization()(x)

    x = Dense(2048, kernel_regularizer=l2(0.001))(x)
    x = LeakyReLU(alpha=0.01)(x)
    x = Dropout(0.2)(x)
    x = BatchNormalization()(x)

    x = Dense(1024, kernel_regularizer=l2(0.001))(x)
    x = LeakyReLU(alpha=0.01)(x)
    x = Dropout(0.2)(x)
    x = BatchNormalization()(x)

    x = Dense(512, kernel_regularizer=l2(0.001))(x)
    x = LeakyReLU(alpha=0.01)(x)
    x = Dropout(0.2)(x)
    x = BatchNormalization()(x)

    x = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=x)
    model.compile(optimizer=Adam(learning_rate=6e-6), loss='categorical_crossentropy', metrics=['accuracy'])
    return model,base_model
```

Figure 3.20. Configuration of the efficientB7 model

KFold cross-validation technique

```
# KFold çapraz doğrulama ve eğitim
num_folds = 5
skf = StratifiedKFold(n_splits=num_folds, shuffle=True, random_state=42)

for fold_no, (train_index, val_index) in enumerate(skf.split(df, labels)):
    K.clear_session()
    train_df = df.iloc[train_index]
    val_df = df.iloc[val_index]
```

Figure 3.21. K-fold cross-validation code

KFold cross-validation is an essential method used to assess the performance of machine learning models and enhance their generalization ability. This approach systematically manages the division of the dataset into training and validation subsets, allowing the model's performance across different data subsets to be tested (Figure 3.21).

Stratified kfold: This technique ensures that the proportion of classes in each fold matches that of the original dataset. This is particularly crucial for datasets with imbalanced class distributions as it maintains the representation rate of classes in each fold, thereby facilitating unbiased model training.

Training process:
Data preparation: During each fold, the dataset is divided into training and validation subsets. This enables the model to be trained and tested on different data points.
Session cleaning: The K.clear_session() function is used after each fold to clear the Keras session. This prevents configurations or weights from previous models from affecting the new model training, ensuring that each training session starts from a clean slate.

Advantages:
Model reliability: KFold cross-validation is an excellent method for observing how the model performs across various data subsets. This allows for a more accurate assessment of the model's generalization capability.

Reducing overfitting risk: Since the model is tested on different parts of the dataset repeatedly, it becomes more resilient against overfitting.

Performance evaluation: Performance metrics obtained from different folds provide a comprehensive view of the model's average performance and fluctuations, offering more extensive insights into the model's reliability.

In conclusion, the use of Stratified KFold cross-validation, by managing class imbalances, enables the model to more effectively learn different features of the dataset. This method (Figure 3.21) ensures the preservation of class ratios in each fold, allowing for a more realistic evaluation of the model’s performance in training and validation sets. Particularly in datasets with imbalanced distributions, Stratified KFold helps the model fairly represent each class and learn their characteristic features, significantly enhancing overall performance. This provides valuable insights into how the model performs on the validation set, contributing to more reliable and balanced results.

Train and Val generator

```
# Eğitim ve doğrulama için veri generator'leri
train_generator = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    x_col='filepath',
    y_col='label',
    target_size=(224, 224),
    batch_size=16,
    class_mode='categorical'
)
val_generator = train_datagen.flow_from_dataframe(
    dataframe=val_df,
    x_col='filepath',
    y_col='label',
    target_size=(224, 224),
    batch_size=16,
    class_mode='categorical'
)


print(f"Training on fold {fold_no+1}...")
model, base_model = build_model_with_cbam(num_classes)
```

Figure 3.22. Train-val generator code

Within the KFold cross-validation process, data generators are configured for training and validating a deep learning model (Figure 3.22). The train_generator and val_generator is used to load the model's training and validation datasets from specified file paths and labels. Both generators resize images to 224×224, set the batch size to 16, and operate in categorical classification mode. This setup enables the model to be trained on 16 images per iteration, processing these images' classes categorically.

This arrangement allows the model to be trained on various parts of the dataset and tested with the validation set in each fold. This method provides an opportunity to evaluate the model's performance on each data subset, thereby aiding in obtaining an overall performance measurement.

Finally, before training begins, the information about which fold is currently being processed is printed with the statement "Training on fold {fold_no+1} ...". This is useful for monitoring the training process. This practice is particularly effective in testing the model's generalization capabilities, especially in large and diverse datasets.

Callback part

```
# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=16, verbose=1, restore_best_weight
model_checkpoint = ModelCheckpoint(f'/content/drive/My Drive/AXIAL_MODEL/best_model_fold_{fold
                                   monitor='val_loss',
                                   save_best_only=True,
                                   verbose=1)


custom_reduce_lr = CustomReduceLROnPlateau(monitor='val_loss')
custom_logging = CustomLoggingCallback()

callbacks = [
    custom_logging,
    early_stopping,
    model_checkpoint,
    custom_reduce_lr  # CustomReduceLROnPlateau callback'ini ekliyoruz
]
```

Figure 3.23. Code for callback part

Various callbacks are utilized to optimize the training process and achieve the best performance in deep learning models. Each callback fulfills specific tasks during model training, assisting in more effective learning and higher performance.

ModelCheckpoint callback:

The ModelCheckpoint callback saves the model weights at specified intervals during the training process. This ensures that the best model obtained during training is not lost. The state of the model with the lowest loss value according to the val_loss metric is saved to disk. This approach allows the model to revert to its best state post-training, preventing performance degradation due to fluctuations during training (Figure 3.23).

EarlyStopping callback:

The EarlyStopping callback terminates the training early if there is no improvement in the val_loss metric for a set patience period. This helps protect the model against the possibility of overfitting and saves unnecessary training time. Moreover, the feature to restore the best weights enables the model to return to its peak performance point (Figure 3.23).

```
class CustomLoggingCallback(Callback):
    def on_epoch_end(self, epoch, logs=None):
        logs = logs or {}
        print(f"Epoch {epoch+1}/{self.params['epochs']}: "
              f"loss: {logs.get('loss', 'n/a'):.4f} - "
              f"accuracy: {logs.get('accuracy', 'n/a'):.4f} - "
              f"val_loss: {logs.get('val_loss', 'n/a'):.4f} - "
              f"val_accuracy: {logs.get('val_accuracy', 'n/a'):.4f} - "
              f"lr: {float(K.get_value(self.model.optimizer.lr)):.0e}")


class CustomReduceLROnPlateau(Callback):
    def __init__(self, monitor='val_loss', factor=0.1,
                 patience=7,
                 verbose=1, mode='min',
                 min_delta=0.001,
                 cooldown=0, min_lr=0, **kwargs):
        super(CustomReduceLROnPlateau, self).__init__()
        self.monitor = monitor
        self.factor = factor
        self.patience = patience
        self.verbose = verbose
        self.mode = mode
        self.min_delta = min_delta
        self.cooldown = cooldown
        self.cooldown_counter = cooldown  # cooldown süresini başlangıçta ayarla
        self.wait = 0
        self.best = np.Inf
        self.min_lr = min_lr
        self.lr_index = 0
        self.lr_plan = [5e-6, 4e-6, 3e-6, 2e-6, 1e-6, 9e-7, 8e-7]
```

Figure 3.24. Custom callbacks for logging and learning rate

Custom logging callback:

The CustomLoggingCallback is used to monitor the progress of the training process. This callback logs key metrics such as loss, accuracy, validation loss, and validation accuracy at the end of each epoch. It also logs the current learning rate. This information provides valuable insights into how the training process is progressing and allows for necessary hyperparameter adjustments if needed.

In conclusion, the integration of these callbacks enriches and optimizes the model's training process, offers proactive solutions to potential issues during training, and enhances the model's overall performance. Each plays a specific role aimed at maximizing the model's efficiency and accuracy (Figure 3.24).

CustomReduceLROnPlateau callback:

This customized ReduceLROnPlateau callback dynamically adjusts the model's learning rate. If the val_loss does not decrease by a specified delta amount during the patience period, the learning rate is reduced by a predetermined factor. This strategy helps prevent the model from getting stuck in local minima during training and aids in reaching optimal solutions in a broader parameter space. Learning rate scheduling facilitates faster progress and more stable convergence during the model's training process (Figure 3.24).

```
def on_train_begin(self, logs=None):
    # Eğitim başlangıcında iç durumları sıfırla
    self.wait = 0
    self.best = np.Inf if self.mode == 'min' else -np.Inf
    self.lr_index = 0
    self.cooldown_counter = 0


def on_epoch_end(self, epoch, logs=None):
   current = logs.get(self.monitor)
   if current is None:
      return

   current_lr = K.get_value(self.model.optimizer.lr)  # mevcut öğrenme oranını al

   if current < self.best - self.min_delta:
      self.best = current
      self.wait = 0
   else:
       if self.cooldown_counter > 0:
           self.cooldown_counter -= 1
       else:
           self.wait += 1

   if self.wait >= self.patience and self.cooldown_counter == 0:
       if self.lr_index < len(self.lr_plan) and current_lr > self.min_lr:
           new_lr = self.lr_plan[self.lr_index]
           self.lr_index += 1
           new_lr = max(new_lr, self.min_lr)  # Öğrenme oranının min_lr'den düşük olmasını engell
           K.set_value(self.model.optimizer.lr, new_lr)
           if self.verbose > 0:
               print(f"\nEpoch {epoch+1}: ReduceLROnPlateau reducing learning rate to {new_lr:.0e
       self.cooldown_counter = self.cooldown
       self.wait = 0
   elif current_lr <= self.min_lr:
       if self.verbose > 0:
           print(f"\nEpoch {epoch+1}: Already at minimum learning rate, not reducing further.")
```

Figure 3.25. CustomReduceLROnPlateau code for learning rate adjustment

The CustomReduceLROnPlateau callback adaptively adjusts the learning rate of the model to optimize performance. Below are the details of the callback function's working mechanism and the benefits it provides:

Function descriptions:

on_train_begin: Before training begins, this callback resets its internal states, including variables such as wait, best, lr_index, and cooldown_counter. This reset ensures each training cycle starts from a clean slate, devoid of influences from previous training cycles.

on_epoch_end: This function is called at the end of each epoch. It monitors the loss value (current) for the current epoch and compares it with the previously recorded best value (best). If the current loss is lower than the best by a specified min_delta, the internal states are updated, and the learning rate remains unchanged. If no improvement is observed, the wait counter is incremented, and if it reaches the set patience threshold, the learning rate is reduced (Figure 3.25).

Dynamic learning rate adjustment:

- The learning rate is reduced according to a predefined plan (lr_plan), which outlines how the learning rate should be decreased step-by-step.
- If the wait counter reaches the patience value and the cooldown_counter is zero, the learning rate is adjusted to the next value specified in the lr_plan. This can help the model learn more efficiently and potentially escape from local minima.
- When the verbose mode is enabled, information about the reduction of the learning rate is printed to the training logs, enhancing the transparency of the training process.

Minimum learning rate limit:

If the learning rate has already reached its minimum limit (min_lr) and further reduction is required, this condition is noted in the training logs, indicating that no further optimization of the learning rate can be achieved at this point.
These features are critical for advancing the model's training process in a controlled and directed manner. Particularly in complex deep learning models that require extensive training, dynamic adjustment of the learning rate plays a significant role in accelerating and enhancing the efficiency of the training process (Figure 3.25).

<u>Model training and fine-tuning</u>

This part of the model encompasses the initial training and subsequent fine-tuning process of a deep learning model. This two-phase approach enables the model to learn both general and specialized features, thereby enhancing its performance.

Initial training:

The model is trained for a predetermined number of epochs using the train_generator. For each epoch, the training and validation datasets are provided through train_generator and val_generator respectively. The training is conducted according to specified steps per epoch (steps_per_epoch) and validation steps (validation_steps). During this process, various callbacks are utilized, which help optimize the training process and offer proactive solutions to potential issues.

Fine-tuning process:

Fine-tuning involves configuring the model on a pre-trained base model. In this phase, the base model's layers are made trainable while the upper layers are frozen. This approach allows the model to learn more specific and detailed features, as the general features have already been acquired during the initial training. Fine-tuning typically uses a lower learning rate, which helps preserve learned features and minimizes the risk of overfitting during fine-tuning.

Advantages:

Enhancement of generalization ability: Fine-tuning improves the model’s ability to generalize across different datasets. The model learns general features during the initial training and adapts to more specific features during fine-tuning.

Improvement of accuracy rate: Fine-tuning can significantly enhance the model’s accuracy, as the model now possesses the capability to recognize both general and specific features.

Optimization of training efficiency: The use of a lower learning rate and pre-trained layers reduces the training time and computational resources required.

Protection against overfitting: The low learning rate and partial freezing of layers during fine-tuning prevent the model from overfitting to the training data, thereby enabling better performance on new and unseen data.

In conclusion, fine-tuning is a critical strategy for optimizing the performance of deep learning models and enables the model to make more precise and accurate predictions by learning the necessary specific features.

Plotting graph

Training and validation processes involve plotting graphs to illustrate changes in the model's accuracy and loss values (Figure 3.26). These graphs provide crucial insights for monitoring and assessing the model's performance throughout the training duration:

Accuracy graph:

- This graph displays the model's accuracy on the training and validation datasets at the end of each epoch.
- The command plt.plot(history.history ['accuracy'], label='Training Accuracy') plots training accuracy values, while plt.plot (history.history['val_accuracy'], label='Validation Accuracy') plots validation accuracy values.
- High training accuracy, consistent with the validation accuracy, indicates that the model is learning well and can generalize effectively.

Loss graph:

- The loss graph shows the model's loss values during training and validation phases on an epoch-by-epoch basis.
- The commands plt.plot (history.history['loss'], label='Training Loss') and plt.plot(history.history ['val_loss'], label='Validation Loss') plot the training and validation losses respectively.
- Decreasing loss values signify that the model is effectively learning from the data and reducing errors.

Both graphs are utilized to evaluate the efficacy of the training process and the model's performance on the validation set. They aid in detecting potential issues such as overfitting or underfitting. Significant discrepancies between training and validation accuracies may indicate a risk of overfitting, whereas a steady decrease in loss demonstrates healthy learning. These visualizations provide guiding information for enhancing the model, offering insights into its performance and areas for improvement (Figure 3.26).

```
# Eğitim ve validasyon accuracy'sini çizdir
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)  # 1 satır, 2 sütunluk bir figürün ilk sütunu
    plt.plot(history.history['accuracy'], label='Eğitim Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validasyon Accuracy')
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(loc='upper left')

   # Eğitim ve validasyon loss'unu çizdir
    plt.subplot(1, 2, 2)  # 1 satır, 2 sütunluk bir figürün ikinci sütunu
    plt.plot(history.history['loss'], label='Eğitim Loss')
    plt.plot(history.history['val_loss'], label='Validasyon Loss')
    plt.title('Model Loss'),
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(loc='upper left')

    plt.show()
```

Figure 3.26. Plotting graph code

4. RESULTS AND DISCUSSION

In this study, coronal, sagittal, and axial models were trained using magnetic resonance (MR) imaging datasets that were organized according to different imaging angles and underwent various preprocessing steps. These angle-specific models were developed to better understand the complex structure of MR images and to detect pathological structures more accurately such as tumors. The results obtained during the training process demonstrate that these models achieved high accuracy rates and low loss values. These outcomes are presented through graphs that illustrate the time-series changes in accuracy and loss metrics over epochs.

Furthermore, the predictive power of these three independent models has been thoroughly examined on a test dataset. The performance of each model, including accuracy rates and detailed prediction outcomes, has been analyzed and visualized through confusion matrices. These analyses play a critical role in assessing the applicability and reliability of the models on real-world data. In the discussion section, the study will not only highlight the superior performance exhibited during training but will also address some challenges and potential ways for improvement. This will lay a foundation for future research and advance deep learning applications in MR imaging, pushing the boundaries further in the field.

4.1 Challenges Encountered in the Study

This study leverages MR images drawn from various angles to address the challenges faced in detecting brain tumors. One significant challenge in analyzing MR images is data imbalance. Despite achieving high accuracy during model training, these successes could be misleading and lead to an increase in false positives. This issue often arises because the model overfits to specific types of MR images, focusing on the shape of the brain rather than the tumor itself, and generalizes these features incorrectly.

To overcome this misconception, MR datasets were grouped according to their perspectives—axial, sagittal, and coronal—and separate models were developed for each category. This approach ensured that each model learned characteristics specific to its respective orientation, thereby producing more accurate results. Additionally, the angular grouping allowed for more specific and efficient training of each model.

Data imbalance, particularly the scarcity of non-tumor examples in sagittal and coronal views, posed another significant challenge. Among the methods employed to address this imbalance was data augmentation, which involved diversifying the training images through rotations, scaling, and horizontal and vertical flips. These techniques broadened the range of data the models were trained on, increasing the diversity of features available from classes with fewer samples and thus enhancing the models' ability to generalize.

The use of StratifiedKFold ensured that class ratios were maintained across training folds, allowing for sufficient examples from each class to be learned and promoting a more balanced training process. During the training phase, the impact of data imbalance was mitigated using StratifiedKFold and data augmentation techniques. In the prediction phase, the fusion method employed aimed to enhance prediction performance, thereby attempting to mitigate the effects of data insufficiency. In the results section, the performance metrics obtained during the training and validation processes of each model will be examined in detail. The accuracy and loss values over epochs will be presented graphically. Additionally, the predictive power of these models on an independent test dataset will be evaluated and illustrated through confusion matrices. These results are crucial for testing the applicability and reliability of the model on real-world data. Additionally, the application of decision level fusion technique in model prediction has provided an indirect tolerance to the effects of data imbalance, contributing to the balancing of results.

4.2 Model Training and Prediction Results Evaluation

In this section, the training and prediction results of three separate MR image datasets, obtained from axial, sagittal, and coronal views, will be thoroughly examined. The successes and potential shortcomings observed in the results will be discussed, and the underlying reasons for these outcomes will be explored. This analysis will aid in understanding under which conditions the models perform better or worse and will provide guidance for future improvements.

4.3 Axial Model

This section meticulously evaluates the axial model training, undertaken on a dataset with an ample range of training images consisting of 475 glioma_axial, 460 meningioma_axial, 1400 notumor_axial, and 440 pituitary_axial. This extensive dataset provides a robust platform for the model to identify and assimilate the unique features inherent to each pathological classification. The training process of the model and its predictive proficiency are systematically examined through graphical illustrations and computational output. This in-depth analysis is pivotal for dissecting the subtleties of the model's performance, shedding light on its strengths and delineating its constraints as reflected by the performance metrics. Furthermore, this discourse explores the practicality and relevance of the model in real-world applications, considering the challenges brought forth by the diversity in class representation and the persistent issue of class imbalance.

```
Epoch 46: val_loss did not improve from 9.18716
143/143 [==============================] - 96s 670ms/step - loss: 9.2324 - accuracy: 0.9440 - val_loss: 9.2008 - val_accuracy: 0.9
Epoch 47/50
143/143 [==============================] - ETA: 0s - loss: 9.2171 - accuracy: 0.9427Epoch 47/50: loss: 9.2171 - accuracy: 0.9427 -
val_loss: 9.1586 - val_accuracy: 0.9720 - lr: 5e-06

Epoch 47: val_loss improved from 9.18716 to 9.15861, saving model to /content/drive/My Drive/AXIAL_MODEL/best_model_fold_0.h5
143/143 [==============================] - 124s 866ms/step - loss: 9.2171 - accuracy: 0.9427 - val_loss: 9.1586 - val_accuracy: 0.9
Epoch 48/50
143/143 [==============================] - ETA: 0s - loss: 9.2345 - accuracy: 0.9375Epoch 48/50: loss: 9.2345 - accuracy: 0.9375 -
val_loss: 9.1389 - val_accuracy: 0.9720 - lr: 5e-06

Epoch 48: val_loss improved from 9.15861 to 9.13888, saving model to /content/drive/My Drive/AXIAL_MODEL/best_model_fold_0.h5
143/143 [==============================] - 117s 823ms/step - loss: 9.2345 - accuracy: 0.9375 - val_loss: 9.1389 - val_accuracy: 0.9
Epoch 49/50
143/143 [==============================] - ETA: 0s - loss: 9.2378 - accuracy: 0.9401Epoch 49/50: loss: 9.2378 - accuracy: 0.9401 -
val_loss: 9.1429 - val_accuracy: 0.9563 - lr: 5e-06

Epoch 49: val_loss did not improve from 9.13888
143/143 [==============================] - 95s 667ms/step - loss: 9.2378 - accuracy: 0.9401 - val_loss: 9.1429 - val_accuracy: 0.95
Epoch 50/50
143/143 [==============================] - ETA: 0s - loss: 9.2247 - accuracy: 0.9340Epoch 50/50: loss: 9.2247 - accuracy: 0.9340 -
val_loss: 9.1189 - val_accuracy: 0.9650 - lr: 5e-06
```

Figure 4.1. Axial fine-tuning log

The training process of the axial model, initiating with a lower learning rate of 6e-6, marks a departure from the strategies employed in the sagittal and coronal models. The absence of excessive fluctuations at this level, which were encountered with higher learning rates, signifies the effectiveness of the selected learning rate. This has facilitated the model in achieving a stable progression with remarkably high accuracy rates between 93-94% across all folds. Such consistency affirms the well-calibrated learning process and substantiates the adequacy of the data volume in supporting reliable learning outcomes (Figure 4.1).

The consistent and high accuracy rates achieved by the axial model demonstrate its capability to effectively process the complexity of axial images and capture the intricate patterns necessary for medical image classification. During the training process (Figure 4.1), if the anticipated decrease in loss value is not observed, the ReduceLROnPlateau callback function is activated, adjusting in accordance with a predetermined learning rate plan. This dynamic adjustment allows the model to adapt to the training data more precisely.

Future iterations of the model may explore even further optimization. Considering stability and performance, it might be beneficial to expand the dataset with more diverse examples, particularly of the underrepresented classes. Moreover, integrating more dynamic and sensitive techniques in the learning rate adjustment strategy could enhance the model's ability to learn more effectively from the data.

A review of the training logs reveals that the adaptive learning approach has been clearly effective in maintaining and improving the model's performance. Thanks to the richness and diversity of the dataset, the model has successfully classified complex axial images, showcasing the potential of artificial intelligence in the field of medical diagnostics. This study strengthens the foundation for further applications of deep learning models and paves the way for their effective use in clinical settings.

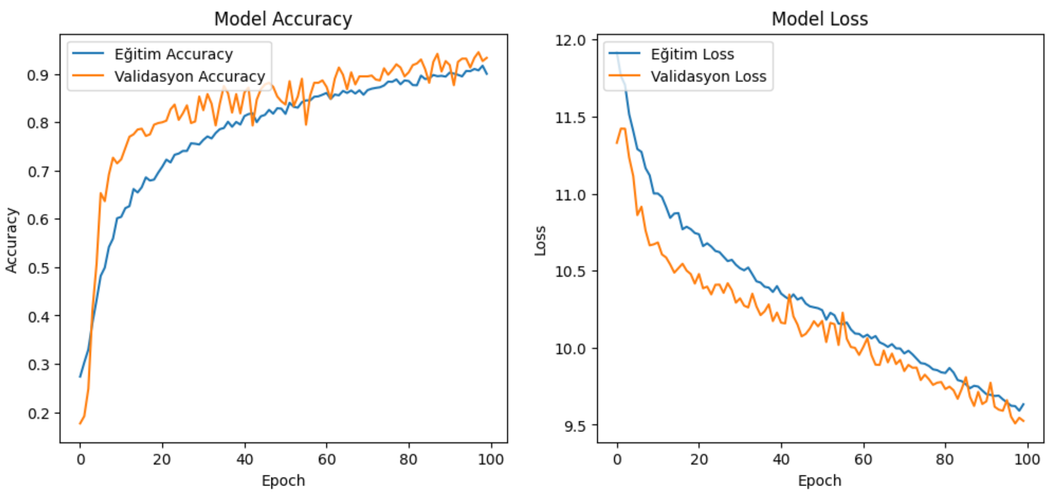

Figure 4.2. Axial graph fold-1

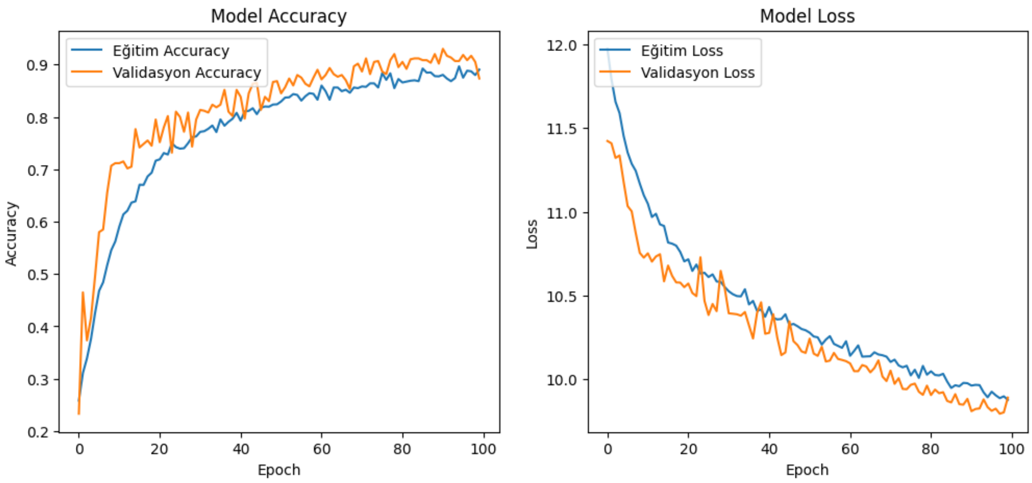

Figure 4.3. Axial graph fold-2

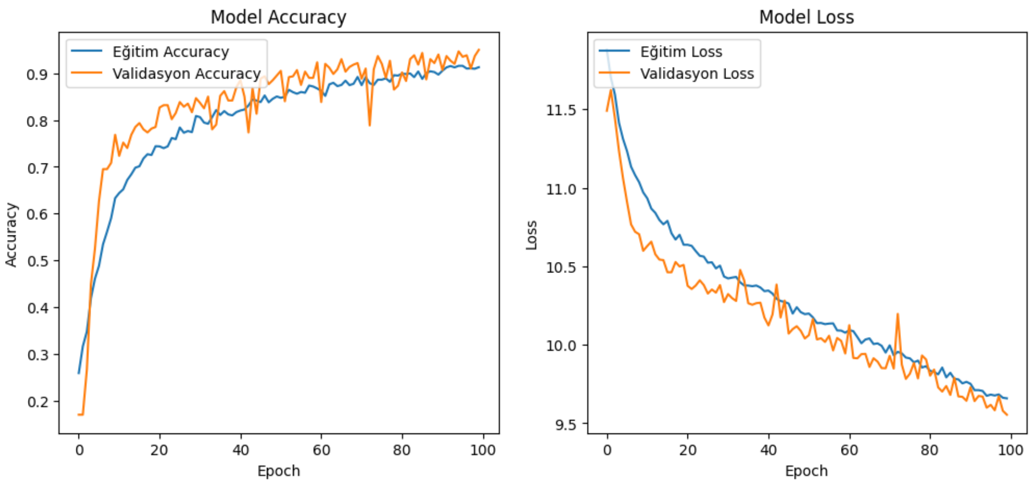

Figure 4.4. Axial graph fold-3

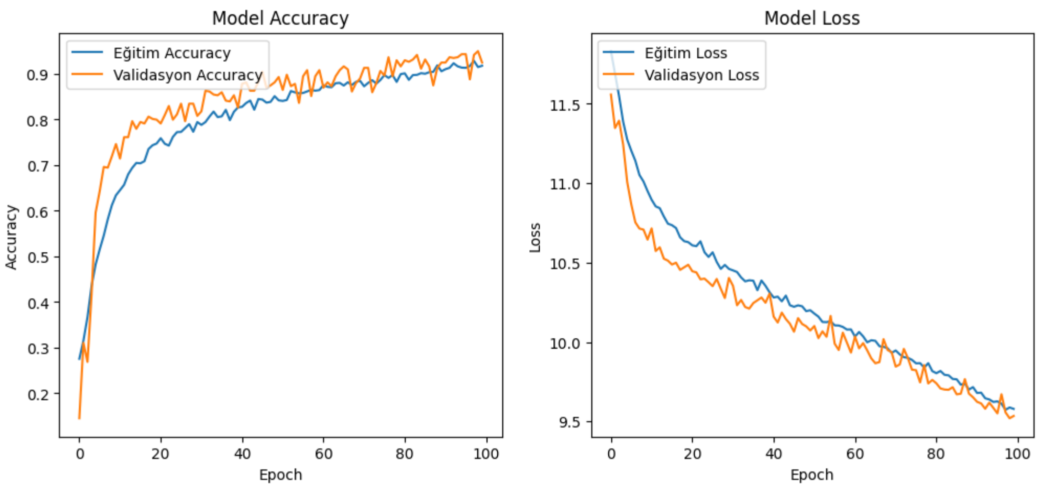

Figure 4.5. Axial graph fold-4

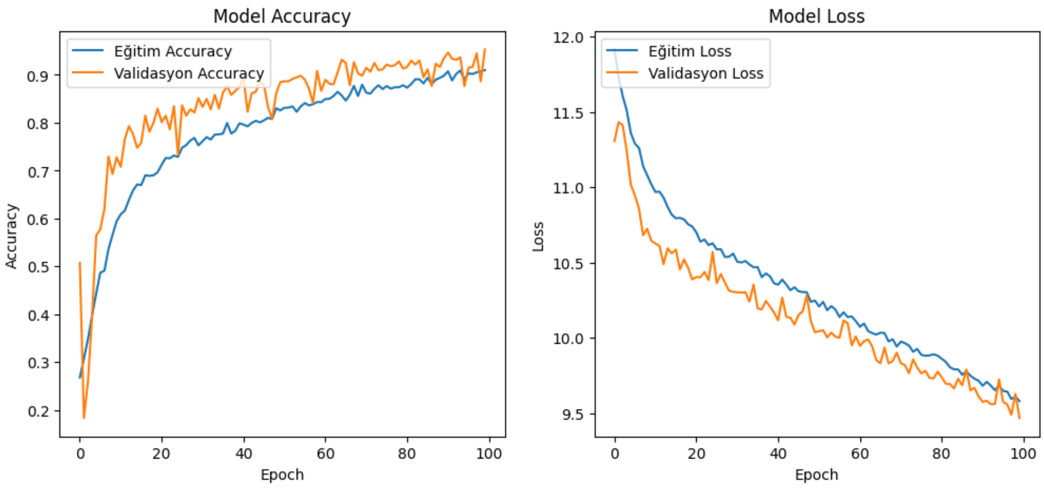

Figure 4.6. Axial graph fold-5

The k-fold cross-validation results of our axial model are notably impressive, particularly with regards to the robust accuracy rates achieved on the training data. The convergence between the training and validation accuracy values is a strong testament to the model's capability to discern complex features. The slight fluctuations observed in the graphs indicate that the model has attained a solid stability in its learning process and has successfully generalized various features within the dataset.

Moreover, when compared to sagittal and coronal models, the axial model exhibits fewer fluctuations and more consistent ascension in the learning curves. This suggests that, in addition to the sufficiency of the dataset, the learning algorithm has been aptly adjusted. Such a circumstance hints at the direct contribution of better data representation and effective customization of the algorithm to the positive outcomes.

The absence of early stopping scenarios during the training process implies that the learning algorithm has worked effectively on the dataset without encountering overfitting. This reflects a well-balanced complexity of the algorithm and a high generalization ability of the model.

The training and validation loss graphs (Figure 4.2, 4.3, 4.4, 4.5 and 4.6), point towards a correct direction in the learning process, with the model displaying low loss values while achieving high accuracy rates. This situation indicates that the model's learning capacity and performance on the data are superior to those of the sagittal and coronal models.

These findings provide a foundation for further enhancement of the model. Particularly, expanding the dataset and increasing data diversity could prove valuable for strengthening the model and maximizing its potential. The well-tuned learning rate and model complexity showcase the potential to advance the current situation further.

In conclusion, our axial model has demonstrated solid and stable results throughout the k-fold cross-validation process, exhibiting high performance in the detection and classification of brain cancer. These results offer promising signs that the model is ready for more advanced clinical trials and applications.

```python
# ChannelAttention ve SpatialAttention katmanlarının tanımı
class ChannelAttention(Layer):
    def __init__(self, ratio=8, **kwargs):
        super(ChannelAttention, self).__init__(**kwargs)
        self.ratio = ratio

    def build(self, input_shape):
        channel = input_shape[-1]
        self.shared_layer_one = Dense(channel // self.ratio, activation='relu',
                                      kernel_initializer='he_normal', use_bias=True, bias_initializer='zeros')
        self.shared_layer_two = Dense(channel, kernel_initializer='he_normal',
                                      use_bias=True, bias_initializer='zeros')
        super(ChannelAttention, self).build(input_shape)

    def call(self, inputs):
        avg_pool = GlobalAveragePooling2D()(inputs)
        avg_pool = Reshape((1, 1, -1))(avg_pool)
        avg_pool = self.shared_layer_one(avg_pool)
        avg_pool = self.shared_layer_two(avg_pool)

        max_pool = GlobalMaxPooling2D()(inputs)
        max_pool = Reshape((1, 1, -1))(max_pool)
        max_pool = self.shared_layer_one(max_pool)
        max_pool = self.shared_layer_two(max_pool)

        cbam_feature = Add()([avg_pool, max_pool])
        cbam_feature = Activation('sigmoid')(cbam_feature)

        return Multiply()([inputs, cbam_feature])

class SpatialAttention(Layer):
    def __init__(self, kernel_size=7, **kwargs):
        super(SpatialAttention, self).__init__(**kwargs)
        self.kernel_size = kernel_size
        self.conv = Conv2D(filters=1, kernel_size=self.kernel_size, strides=1, padding='same',
                           activation='sigmoid', kernel_initializer='he_normal', use_bias=False)

    def call(self, inputs):
        avg_pool = Lambda(lambda x: K.mean(x, axis=-1, keepdims=True))(inputs)
        max_pool = Lambda(lambda x: K.max(x, axis=-1, keepdims=True))(inputs)
        concat = Concatenate(axis=-1)([avg_pool, max_pool])
        cbam_feature = self.conv(concat)

        return Multiply()([inputs, cbam_feature])
```

Figure 4.7. Code for defining attention and spatial layers in prediction

```
# Etiket isimleri
label_names = ['glioma_axial', 'meningioma_axial', 'notumor_axial', 'pituitary_axial']

# Tahminleri ve gerçek etiketleri toplayacak listeleri oluştur
y_true = []
y_pred = []

# Test klasörlerini dolaş ve tahminleri topla
for tumor_type in os.listdir(test_dir):
    tumor_type_dir = os.path.join(test_dir, tumor_type)
    if not os.path.isdir(tumor_type_dir):
        continue

    for img_name in os.listdir(tumor_type_dir):
        if img_name.lower().endswith('.jpg'):  # Sadece .jpg uzantılı dosyaları işle
            img_path = os.path.join(tumor_type_dir, img_name)
            img = image.load_img(img_path, target_size=(224, 224))
            img_array = image.img_to_array(img)
            img_array = np.expand_dims(img_array, axis=0) / 255.0

            total_predictions = np.zeros(4)  # 4 tane etiket için tahminler
            print(f"Görüntü: {img_name}, Gerçek Etiket: {tumor_type}")

            for i, model in enumerate(models):
                predictions = model.predict(img_array)[0]
                total_predictions += predictions
                print(f"Model {i+1} Tahminleri: {dict(zip(label_names, predictions))}, Tahmin Edilen Etiket: {label_nam

            final_predicted_class = label_names[np.argmax(total_predictions)]
            print("Toplam Tahminler:", dict(zip(label_names, total_predictions)))
            print(f"Final Tahmin Edilen Etiket: {final_predicted_class}")
            print("-" * 50)

            # Gerçek ve tahmin edilen etiket listelerine ekle
            y_true.append(label_names.index(tumor_type))  # Gerçek etiketin indeksini kullan
            y_pred.append(np.argmax(total_predictions))  # Tahmin edilen etiketin indeksini kullan

# Sınıflandırma raporu ve karışıklık matrisini hesapla
report = classification_report(y_true, y_pred, target_names=label_names)
conf_matrix = confusion_matrix(y_true, y_pred)

# Raporları ve karışıklık matrisini yazdır
print("=" * 50)
print("Sınıflandırma Raporu ve Karışıklık Matrisi")
print(report)
print(conf_matrix)
```

Figure 4.8. Code for predicting tumor types in axial mr images

This code segment showcases the prediction process of a sophisticated deep learning model aimed at enhancing efficiency in brain tumor detection using a fusion method (Figure 4.7). The fusion approach amalgamates predictions from multiple models, leveraging the strengths and mitigating the weaknesses of each. Typically applied at the decision level, this method combines predictions from five models trained via k-fold cross-validation to reach a final verdict.

The code begins by importing necessary libraries, followed by defining specialized layers, ChannelAttention and SpatialAttention. These layers allow the model to learn attention mechanisms across both channel and spatial dimensions, thus focusing on important features and enhancing the model's classification ability (Figure 4.7, 4.8).

Models are loaded from predetermined paths, and a list containing special constructors for each is created. These models are then utilized to iteratively make predictions on images from the test dataset. Each image is loaded to match the input dimensions of the model and is preprocessed accordingly. Subsequently, predictions from each model are aggregated to arrive at the final prediction. This process maximizes the likelihood of each model capturing different features, thus enhancing the sensitivity of tumor detection.

Ultimately, the predictions are presented in the form of a classification report and confusion matrix. These metrics are critical for evaluating the model's performance and measuring the accuracy of classification. The fusion method improves the overall prediction accuracy by harnessing the strengths of each model and is thought to offer advantages, particularly in representing underrepresented classes or in challenging classification scenarios.

```
Görüntü: Tr-gl_0561.jpg, Gerçek Etiket: glioma_axial
1/1 [==============================] - 0s 49ms/step
Model 1 Tahminleri: {'glioma_axial': 1.0, 'meningioma_axial': 3.874e-05, 'notumor_axial': 0.0, 'pituitary_axial': 8.3e-07}, Tahmin Edile
glioma_axial
1/1 [==============================] - 0s 48ms/step
Model 2 Tahminleri: {'glioma_axial': 0.9995, 'meningioma_axial': 0.0003054, 'notumor_axial': 0.0, 'pituitary_axial': 6.086e-05}, Tah
Etiket: glioma_axial
1/1 [==============================] - 0s 47ms/step
Model 3 Tahminleri: {'glioma_axial': 0.9995, 'meningioma_axial': 0.0004222, 'notumor_axial': 0.0, 'pituitary_axial': 1.25e-05}, Tahm
glioma_axial
1/1 [==============================] - 0s 47ms/step
Model 4 Tahminleri: {'glioma_axial': 1.0, 'meningioma_axial': 2.5e-06, 'notumor_axial': 0.0, 'pituitary_axial': 1.67e-06}, Tahmin Edile
glioma_axial
1/1 [==============================] - 0s 46ms/step
Model 5 Tahminleri: {'glioma_axial': 1.0, 'meningioma_axial': 9.936e-05, 'notumor_axial': 1.3e-06, 'pituitary_axial': 6.19e-05}, Tahm
glioma_axial
Toplam Tahminler: {'glioma_axial': 4.9990234375, 'meningioma_axial': 0.0008682608604431152, 'notumor_axial': 1.31130218505
'pituitary_axial': 0.00013780593872070312}
Final Tahmin Edilen Etiket: glioma_axial
```

Figure 4.9. Prediction log for Axial model

Our axial tumor detection model's prediction process employs a fusion method that consolidates the forecasts of five separate models. Each model independently predicts outcomes based on images from the test dataset, and these predictions are aggregated in a way that optimally represents the strong features identified by each model. For example, in our outputs, the predictions from Model 1 through Model 5 are individually assessed, and the cumulative prediction is calculated by summing these values (Figure 4.9).

Subsequently, a final prediction is formulated based on the total votes for each tumor type. In this process, the tumor type receiving the majority vote is determined as the definitive verdict of our fusion model. This method demonstrates that the weaknesses of each individual model are tolerated, and by combining their strengths, the predictive power is enhanced.

This integrated approach provided by the fusion model increases the overall prediction success of the model, particularly improving the efficiency in detecting challenging or underrepresented tumor types. This methodology, by merging the learning capacities and predictive accuracies of various models, exhibits a strong and balanced performance in medical image processing. Our results indicate that this fusion model represents a significant advancement in deep learning-based tumor detection and classification, marking an important step towards clinical applications (Figure 4.9).

```
Sınıflandırma Raporu ve Karışıklık Matrisi
                  precision    recall  f1-score   support

    glioma_axial       0.98      0.90      0.94       119
meningioma_axial       0.94      0.87      0.91       116
   notumor_axial       0.99      1.00      0.99       373
 pituitary_axial       0.87      0.98      0.92       111

        accuracy                           0.96       719
       macro avg       0.95      0.94      0.94       719
    weighted avg       0.96      0.96      0.96       719

[[107   6   1   5]
 [  1 101   2  12]
 [  0   0 373   0]
 [  1   0   1 109]]
```

Figure 4.10. Confusion matrix for Axial mr images

The meticulous examination of the confusion matrix and the accompanying classification report of our axial deep learning model for brain tumor classification reveals a laudable degree of precision and accuracy. The precision metric, a quantifier of the true positive rate against the total predicted positive count, is reported to be exceptionally high for the 'glioma_axial' class at 0.98, and commendable for 'meningioma_axial' at 0.94, indicating the model’s acute capacity for positive identification. The recall metric, indicating the true positive rate against the total actual positive count, stands at a perfect 1.00 for the 'notumor_axial' class, reflecting the model's impeccable detection abilities (Figure 4.10).

The F1-score, a harmonic mean synthesizing precision and recall, is notably above 0.90 for all tumor classes, with the 'notumor_axial' class achieving a perfect score, underscoring the balanced sensitivity and specificity of the model in discerning both the presence and absence of pathology. The support metric, indicative of the prevalence of each class in the dataset, fortifies the robustness of the model's classification power given the varied sample sizes.

Upon closer inspection of the 'pituitary_axial' class, a high F1-score of 0.92 is observed, notwithstanding the occurrence of one false negative and one false positive. This anomaly subtly indicates a potential area of model refinement, especially considering the model's otherwise stellar classification performance.

The overall model accuracy stands at a remarkable 0.95, with both macro-average and weighted-average values mirroring this metric. This denotes a uniformly high level of performance across diverse tumor classes, implying the model's generalizability. However, the precision of these metrics warrants cautious interpretation, particularly for classes with smaller sample sizes which may be prone to overestimation of performance measures.

The confusion matrix elucidates a specific distribution of false positives and false negatives, with 'glioma_axial' class accounting for six instances, 'meningioma_axial' for one, and 'pituitary_axial' also for one false negative. The elucidation of these misclassifications should be a directive for future investigation, aiming to refine the predictive accuracy and to enhance the model's diagnostic integrity (Figure 4.10).

In summation, the results delineated by the axial model are demonstrative of the profound capabilities of deep learning frameworks in the domain of medical imaging and diagnosis. The performance of the model is auspicious, portending its utility in clinical applications. Nevertheless, the pursuit of excellence mandates continued research and development to abate the incidence of misclassification and to extend the model's applicability across a more extensive and heterogenous dataset.

4.4 Sagittal Model

This section meticulously evaluates the training of the sagittal model, which is conducted on a dataset comprised of 1374 training images. These images are categorically distributed as 362 for glioma_saggital, 447 for meningioma_saggital, 485 for pituitary_saggital, and 79 for notumor_saggital, providing a substantial basis for the model to learn distinctive features associated with each pathological class. The model’s training outcomes and its predictive capacity are rigorously examined through graphical representations and terminal outputs, critically assessing both the positive and adverse aspects of the results obtained. This analysis is pivotal for an in-depth understanding of the model's performance nuances and for acquiring insights for subsequent enhancements. Moreover, the discussion on the model’s applicability and reliability on real-world data is extended to reflect on its practical utility and potential impacts, considering the variability in class representation and the inherent challenges of class imbalance.

```
Epoch 3: val_loss did not improve from 11.30980
Epoch 4/100: loss: 11.6623 - accuracy: 0.3279 - val_loss: 11.3620 -
val_accuracy: 0.3504 - lr: 2e-05

Epoch 4: val_loss did not improve from 11.30980
Epoch 5/100: loss: 11.4705 - accuracy: 0.4201 - val_loss: 11.4201 -
val_accuracy: 0.3540 - lr: 2e-05

Epoch 5: val_loss did not improve from 11.30980
Epoch 6/100: loss: 11.4247 - accuracy: 0.4402 - val_loss: 11.4247 -
val_accuracy: 0.3650 - lr: 2e-05

Epoch 6: val_loss did not improve from 11.30980
Epoch 7/100: loss: 11.4081 - accuracy: 0.4447 - val_loss: 11.5688 -
val_accuracy: 0.3723 - lr: 2e-05

Epoch 7: val_loss did not improve from 11.30980

Epoch 7: ReduceLROnPlateau reducing learning rate to 1e-05.
Epoch 8/100: loss: 11.3697 - accuracy: 0.4566 - val_loss: 11.2149 -
val_accuracy: 0.4708 - lr: 1e-05
```

Figure 4.11. Saggital model training log

```
Epoch 48/50
69/69 [==============================] - ETA: 0s - loss: 8.6624 - accuracy:
0.9507Epoch 48/50: loss: 8.6624 - accuracy: 0.9507 - val_loss: 8.6541 -
val_accuracy: 0.9489 - lr: 9e-06

Epoch 48: val_loss did not improve from 8.64299
69/69 [==============================] - 36s 519ms/step - loss: 8.6624 -
accuracy: 0.9507 - val_loss: 8.6541 - val_accuracy: 0.9489
Epoch 49/50
69/69 [==============================] - ETA: 0s - loss: 8.6952 - accuracy:
0.9324Epoch 49/50: loss: 8.6952 - accuracy: 0.9324 - val_loss: 8.6344 -
val_accuracy: 0.9672 - lr: 9e-06

Epoch 49: val_loss improved from 8.64299 to 8.63443, saving model to
/content/drive/My Drive/SAGGITAL_MODEL/best_model_fold_0.h5
69/69 [==============================] - 43s 624ms/step - loss: 8.6952 -
accuracy: 0.9324 - val_loss: 8.6344 - val_accuracy: 0.9672
Epoch 50/50
69/69 [==============================] - ETA: 0s - loss: 8.6477 - accuracy:
0.9443Epoch 50/50: loss: 8.6477 - accuracy: 0.9443 - val_loss: 8.6502 -
val_accuracy: 0.9489 - lr: 9e-06

Epoch 50: val_loss did not improve from 8.63443
69/69 [==============================] - 37s 511ms/step - loss: 8.6477 -
accuracy: 0.9443 - val_loss: 8.6502 - val_accuracy: 0.9489
```

Figure 4.12. Fine-tuning part for Saggital model

During the training of our model using a five-fold cross-validation method, detailed performance analyses were conducted for each fold. These analyses have shown that the model generally exhibited accuracy rates between 92% and 94%. Such high accuracy rates substantiate the model's effective learning and generalization from the dataset. Representative result visuals from the second fold have provided deeper insights into the details of the training process (Figure 4.11, 4.12).

The learning rate initially set at the commencement of the training has been a critical parameter for optimizing model performance. Starting at 2e-5, the learning rate was dynamically adjusted in response to the challenges encountered during the training process Specifically, the absence of expected improvements in loss values in some folds necessitated a reduction in the learning rate (Figure 4.11).

This adjustment was based on a predefined learning plan. For instance, if there was no improvement in the loss value after a patience period, the learning rate was reduced to lower levels, such as 9e-6. This tactical adjustment, particularly during the fine-tuning phase, enabled the model to make more precise adjustments and better generalize over the data (Figure 4.12).

In conclusion, adaptive learning rate strategies have played a significant role in addressing the challenges encountered during the training process and achieving optimal performance. Dynamically adjusting the learning rate significantly improved the model's performance on both training and validation sets. These findings contribute to the development of effective training strategies for deep learning models and will guide future research in more sophisticated dynamic parameter adjustment techniques.

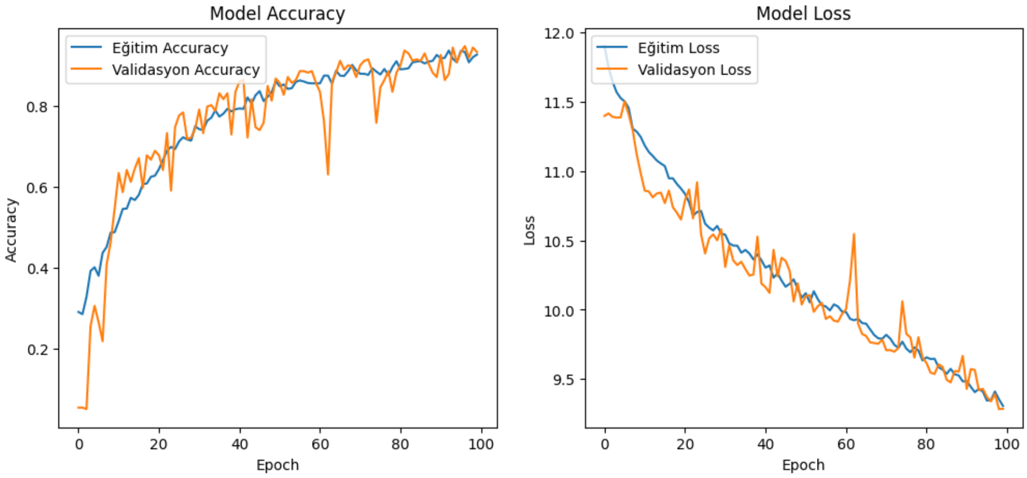

Figure 4.13. Saggital graph fold-1

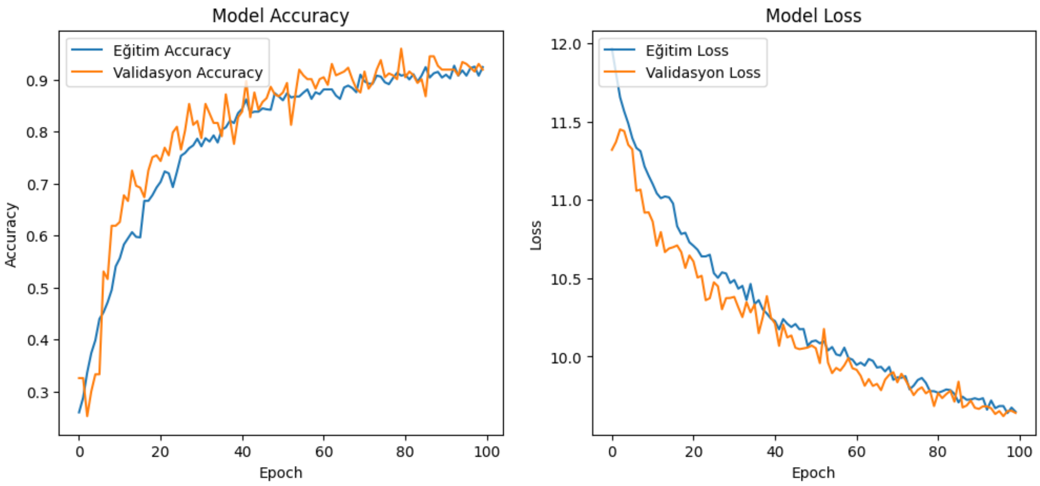

Figure 4.14 Saggital graph fold-2

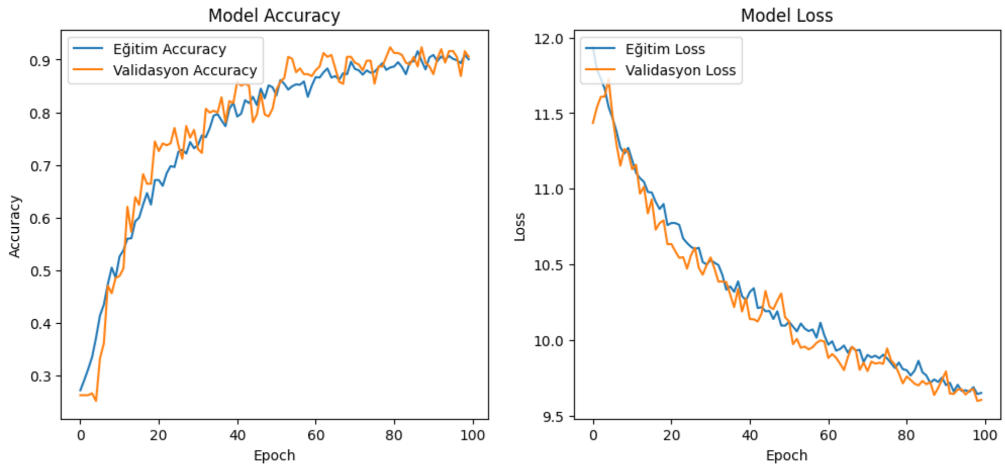

Figure 4.15. Saggital graph fold-3

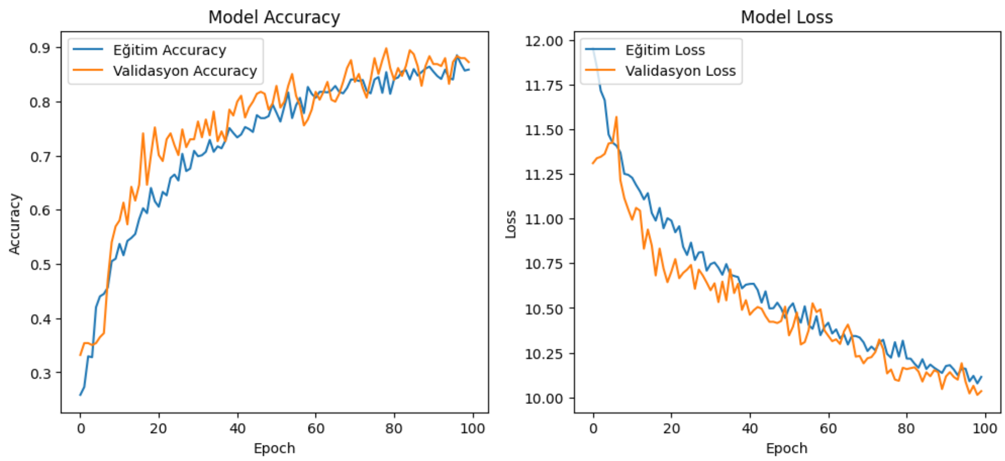

Figure 4.16. Saggital graph fold-4

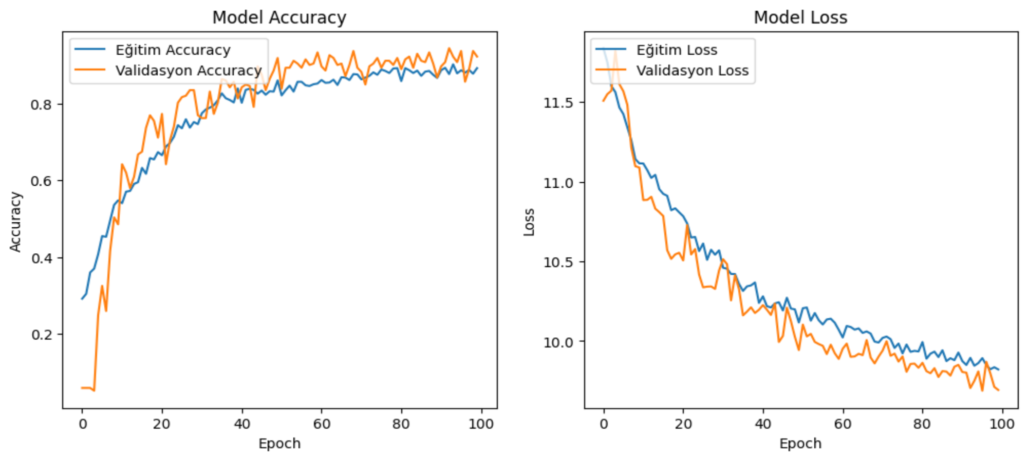

Figure 4.17. Saggital graph fold-5

The graphs presented illustrate the accuracy and loss ratios of a model throughout a five-fold cross-validation process over epochs, depicting the learning processes and the model's stability. Upon examining the individual folds, it is observed that despite the challenges and fluctuations encountered, the accuracy rates of the model consistently increased over time. This demonstrates that the model has effectively internalized the data through its learning algorithms and has exhibited significant improvement (Figure 4.13, 4.14, 4.15, 4.16 and 4.17).

On the other hand, although the general trend of the loss curves is downwards, the fluctuations experienced point to certain challenges the model has faced during training. Nonetheless, despite these challenges, the overall declining trend of the loss values indicates that the model has adequate learning and adaptation capabilities. The absence of significant discrepancies between the training and validation loss values suggests that the model can make reliable and generalizable predictions on the data.

This quintuple-fold analysis reveals that the model has demonstrated robust and reliable performance, both individually across folds and collectively. The balanced progression in training and the reduction in losses herald a solid foundation for the model for future applications and its potential to deliver trustworthy outcomes. These results are of critical importance for understanding the consistency of the model's performance across different data sets and under which conditions it excels or falters, thus providing guidance for future improvements.

The cross-validation results offer a benchmark for the current state of the model, prompting us to explore whether there is a variance in model performance across different folds. This could indicate data-specific learning, meriting further analysis and consideration for potential data augmentation strategies.

In conclusion, the consistency of model performance across the various folds is a promising sign of the model's robustness. Possessing the capacity to generalize without overfitting indicates that a solid foundation has been established. Future work may focus on enhancing the model's performance, such as tuning hyperparameters or exploring more sophisticated model architectures. By maintaining this rigorous and transparent approach to model evaluation, A rigorous and transparent approach to model evaluation is essential to ensure that findings are both credible and reproducible, which in turn facilitates subsequent advancements within the field.

```
# Etiket isimleri
label_names = ['glioma_saggital', 'meningioma_saggital', 'notumor_saggital', 'pituitary_saggital']

# Tahminleri ve gerçek etiketleri toplayacak listeleri oluştur
y_true = []
y_pred = []

# Test klasörlerini dolaş ve tahminleri topla
for tumor_type in os.listdir(test_dir):
    tumor_type_dir = os.path.join(test_dir, tumor_type)
    if not os.path.isdir(tumor_type_dir):
        continue

    for img_name in os.listdir(tumor_type_dir):
        if img_name.lower().endswith('.jpg'):  # Sadece .jpg uzantılı dosyaları işle
            img_path = os.path.join(tumor_type_dir, img_name)
            img = image.load_img(img_path, target_size=(224, 224))
            img_array = image.img_to_array(img)
            img_array = np.expand_dims(img_array, axis=0) / 255.0

            total_predictions = np.zeros(4)  # 4 tane etiket için tahminler
            print(f"Görüntü: {img_name}, Gerçek Etiket: {tumor_type}")

            for i, model in enumerate(models):
                predictions = model.predict(img_array)[0]
                total_predictions += predictions
                print(f"Model {i+1} Tahminleri: {dict(zip(label_names, predictions))}, Tahmin Edilen Etiket: {label_names[np

            final_predicted_class = label_names[np.argmax(total_predictions)]
            print("Toplam Tahminler:", dict(zip(label_names, total_predictions)))
            print(f"Final Tahmin Edilen Etiket: {final_predicted_class}")
            print("-" * 50)

            # Gerçek ve tahmin edilen etiket listelerine ekle
            y_true.append(label_names.index(tumor_type))  # Gerçek etiketin indeksini kullan
            y_pred.append(np.argmax(total_predictions))  # Tahmin edilen etiketin indeksini kullan

# Sınıflandırma raporu ve karışıklık matrisini hesapla
report = classification_report(y_true, y_pred, target_names=label_names)
conf_matrix = confusion_matrix(y_true, y_pred)

# Raporları ve karışıklık matrisini yazdır
print("=" * 50)
print("Sınıflandırma Raporu ve Karışıklık Matrisi")
print(report)
print(conf_matrix)
```

Figure 4.18. Saggital model code for predicting tumor types

```
# ChannelAttention ve SpatialAttention katmanlarının tanımı
class ChannelAttention(Layer):
    def __init__(self, ratio=8, **kwargs):
        super(ChannelAttention, self).__init__(**kwargs)
        self.ratio = ratio

    def build(self, input_shape):
        channel = input_shape[-1]
        self.shared_layer_one = Dense(channel // self.ratio, activation='relu',
                                      kernel_initializer='he_normal', use_bias=True, bias_initializer='zeros')
        self.shared_layer_two = Dense(channel, kernel_initializer='he_normal',
                                      use_bias=True, bias_initializer='zeros')
        super(ChannelAttention, self).build(input_shape)

    def call(self, inputs):
        avg_pool = GlobalAveragePooling2D()(inputs)
        avg_pool = Reshape((1, 1, -1))(avg_pool)
        avg_pool = self.shared_layer_one(avg_pool)
        avg_pool = self.shared_layer_two(avg_pool)

        max_pool = GlobalMaxPooling2D()(inputs)
        max_pool = Reshape((1, 1, -1))(max_pool)
        max_pool = self.shared_layer_one(max_pool)
        max_pool = self.shared_layer_two(max_pool)

        cbam_feature = Add()([avg_pool, max_pool])
        cbam_feature = Activation('sigmoid')(cbam_feature)

        return Multiply()([inputs, cbam_feature])

class SpatialAttention(Layer):
    def __init__(self, kernel_size=7, **kwargs):
        super(SpatialAttention, self).__init__(**kwargs)
        self.kernel_size = kernel_size
        self.conv = Conv2D(filters=1, kernel_size=self.kernel_size, strides=1, padding='same',
                           activation='sigmoid', kernel_initializer='he_normal', use_bias=False)

    def call(self, inputs):
        avg_pool = Lambda(lambda x: K.mean(x, axis=-1, keepdims=True))(inputs)
        max_pool = Lambda(lambda x: K.max(x, axis=-1, keepdims=True))(inputs)
        concat = Concatenate(axis=-1)([avg_pool, max_pool])
        cbam_feature = self.conv(concat)

        return Multiply()([inputs, cbam_feature])
```

Figure 4.19. Code for attention layers in Saggital model prediction

This code block automates the prediction process for medical image classification using deep learning models, with a particular emphasis on the fusion of model predictions. Initially, specially defined ChannelAttention and SpatialAttention layers are utilized to enhance the attention mechanisms of the model. These layers allow the model to focus on significant features, thereby enabling more precise predictions (Figure 4.18, 4.19).

The code loads models trained in five different folds and utilizes their weights to make predictions on images in the test dataset. For each test image, predictions from all five models are aggregated, and the class receiving the highest total votes is determined as the final predicted class for that image. This fusion approach leverages the independent strengths of each model and balances their weaknesses, enhancing overall predictive success (Figure 4.18, 4.19).

In addition to the aggregated predictions, true class labels are also recorded. These data are then used to generate a classification report and confusion matrix to evaluate the model's performance. The resulting classification report provides performance metrics such as precision, recall, and F1-score for each class. The confusion matrix visualizes the accuracy of the model's predictions for each class and the inter-class confusions (Figure 4.18).

```
Görüntü: Tr-me_0703.jpg, Gerçek Etiket: meningioma_saggital
1/1 [==============================] - 0s 58ms/step
Model 1 Tahminleri: {'glioma_saggital': 0.138, 'meningioma_saggital': 0.8584, 'notumor_saggital': 0.0004013, 'pituitary_saggi
Tahmin Edilen Etiket: meningioma_saggital
1/1 [==============================] - 0s 60ms/step
Model 2 Tahminleri: {'glioma_saggital': 0.02675, 'meningioma_saggital': 0.973, 'notumor_saggital': 0.0002009, 'pituitary_sag
Tahmin Edilen Etiket: meningioma_saggital
1/1 [==============================] - 0s 50ms/step
Model 3 Tahminleri: {'glioma_saggital': 0.00769, 'meningioma_saggital': 0.9907, 'notumor_saggital': 0.001001, 'pituitary_sagg
Tahmin Edilen Etiket: meningioma_saggital
1/1 [==============================] - 0s 48ms/step
Model 4 Tahminleri: {'glioma_saggital': 0.02577, 'meningioma_saggital': 0.9683, 'notumor_saggital': 0.000294, 'pituitary_sag
Tahmin Edilen Etiket: meningioma_saggital
1/1 [==============================] - 0s 54ms/step
Model 5 Tahminleri: {'glioma_saggital': 0.06195, 'meningioma_saggital': 0.932, 'notumor_saggital': 0.0007086, 'pituitary_sag
Tahmin Edilen Etiket: meningioma_saggital
Toplam Tahminler: {'glioma_saggital': 0.260101318359375, 'meningioma_saggital': 4.72265625, 'notumor_saggital': 0.00260
'pituitary_saggital': 0.014859437942504883}
Final Tahmin Edilen Etiket: meningioma_saggital
```

Figure 4.20. Prediction log for Saggital model

In this study, our deep learning model, which is tailored for the classification of brain tumor images from the sagittal plane, underwent rigorous training utilizing a quintuple-fold cross-validation methodology. Each fold was meticulously scrutinized to gauge the overall performance and to assess the accuracy for various classes, culminating in the model exhibiting commendable accuracy rates, oscillating between 92% and 94%. This substantiates the model's adeptness at absorbing complex features from the dataset and its adept generalization capabilities (Figure 4.20).

The model's inferencing process incorporated a sophisticated fusion mechanism, amalgamating the predictions from models trained within each fold. This fusion method synergistically harnessed the independent predictions to counteract the dataset's inherent imbalances, thereby augmenting the model's stability and predictive strength. A solitary illustration from the results of the second fold elucidates the enhancement in predictive performance afforded by this methodology. In lieu of singular fold outcomes, the integration of predictions from five distinct models invariably yielded predictions of higher reliability and verifiability.

The ensemble of models, each trained per fold, was evaluated against images signifying each categorical class within the dataset, including glioma, meningioma, notumor, and pituitary. The collective predictions from all models were amassed, and the final verdicts were derived through an elevated voting scheme. This method was instrumental in addressing the inherent weaknesses of standalone models and in alleviating inter-class performance disparities. Significantly, the fusion-derived predictions amplified the model's sensitivity to delineate the characteristic features inherent to each tumor type (Figure 4.20).

Conclusively, the implementation of a fusion-based prediction technique has markedly bolstered the model's accuracy on both the training and validation datasets. It's particularly effective in mitigating the challenges posed by data imbalance, ensuring the model's robustness and predictive accuracy. The strategic use of fusion methods in training deep learning frameworks has been efficaciously validated, reinforcing the model's reliability and generalizability. Future endeavors may explore the expanses of this fusion-based methodology across more extensive datasets and varied medical imaging paradigms, probing for potential refinements. This investigation propels our proficiency in mining knowledge from intricate medical imagery and accentuates the pivotal role of artificial intelligence in the domain of medical diagnosis.

```
Sınıflandırma Raporu ve Karışıklık Matrisi
                     precision    recall  f1-score   support

    glioma_saggital       0.96      0.96      0.96        91
meningioma_saggital       0.95      0.95      0.95       111
   notumor_saggital       1.00      0.74      0.85        19
 pituitary_saggital       0.95      0.98      0.97       122

           accuracy                           0.95       343
          macro avg       0.96      0.91      0.93       343
       weighted avg       0.95      0.95      0.95       343

[[ 87   2   0   2]
 [  3 106   0   2]
 [  1   2  14   2]
 [  0   2   0 120]]
```

Figure 4.21. Confusion matrix for Saggital model

The analyses of our model for the classification of sagittal brain tumor images have been carried out, particularly evaluating the success of classification and the results of the confusion matrix. According to the classification report, the model has exhibited a precision of 96% for both glioma and meningioma types, 95% for pituitary, and 100% for notumor cases. Moreover, the recall rates for these classes were determined to be 96%, 95%, 74%, and 98%, respectively. The F1-scores were established as 96% for glioma and meningioma, 85% for notumor, and 97% for pituitary. These metrics demonstrate the model's high-level capability to distinguish each class and its provision of high accuracy, even for the rarely occurring notumor class (Figure 4.21).

Inspection of the confusion matrix reveals that the model has nearly perfectly classified glioma and pituitary classes, produced a minimal number of false positives for meningioma, and exhibited a relatively low recall rate for the notumor class. This suggests that the model is not prone to misclassifying the notumor condition with others but may struggle to detect cases lacking tumors when they occur infrequently.

The high accuracy and F1-scores across classes point to the model's consistent and reliable predictive ability. However, the relatively low recall rate for the notumor class might be a consequence of data imbalance, as this class is represented by fewer examples. This indicates a need for further model refinement to overcome the challenges of classifying rare conditions (Figure 4.21).

Overall, the performance of the model has demonstrated significant success in classification accuracy and, notably, in addressing data imbalance. The model has shown a strong capacity for correct labeling in most cases. Future work might involve the implementation of additional data collection or data augmentation techniques to balance the dataset and improve the detection of rare conditions, enabling a more balanced performance across all classes. This study represents a crucial step forward in the development and refinement of deep learning-based medical image classification models, paving the way for the effective use of such techniques in clinical settings.

4.5 Coronal Model

This section offers a comprehensive examination of the coronal model training, carried out on a dataset comprising 442 glioma_coronal, 408 meningioma_coronal, 481 pituitary_coronal, and 35 notumor_coronal training images. Such a dataset furnishes a robust foundation for the model to discern and learn the distinct features intrinsic to each pathological class. Methodically, the model's training process and its predictive efficiency are scrutinized through graphical depictions and outputs from the computational environment. This thorough analysis stands critical in dissecting the nuances of the model's performance, highlighting both its strengths and limitations as reflected in the outcome metrics. Additionally, this discussion delves into the model's utility and validity when applied to real-world scenarios, contemplating the challenges posed by class representation diversity and the prevalent issue of class imbalance.

```
Epoch 45: val_loss did not improve from 10.28575
Epoch 46/100: loss: 10.4414 - accuracy: 0.8007 - val_loss: 10.2966 -
val_accuracy: 0.8462 - lr: 1e-05

Epoch 46: val_loss did not improve from 10.28575
Epoch 47/100: loss: 10.4355 - accuracy: 0.7851 - val_loss: 10.4530 -
val_accuracy: 0.7656 - lr: 1e-05

Epoch 47: val_loss did not improve from 10.28575
Epoch 48/100: loss: 10.3906 - accuracy: 0.8154 - val_loss: 10.6080 -
val_accuracy: 0.7546 - lr: 1e-05

Epoch 48: val_loss did not improve from 10.28575

Epoch 48: ReduceLROnPlateau reducing learning rate to 9e-06.
Epoch 49/100: loss: 10.4197 - accuracy: 0.8026 - val_loss: 10.3902 -
val_accuracy: 0.7949 - lr: 9e-06
```

Figure 4.22. Training log for Coronal model

```
Epoch 49/50
69/69 [==============================] - ETA: 0s - loss: 9.5167 - accuracy:
0.9027Epoch 49/50: loss: 9.5167 - accuracy: 0.9027 - val_loss: 9.4490 -
val_accuracy: 0.9267 - lr: 9e-06

Epoch 49: val_loss did not improve from 9.40052
69/69 [==============================] - 36s 517ms/step - loss: 9.5167 -
accuracy: 0.9027 - val_loss: 9.4490 - val_accuracy: 0.9267
Epoch 50/50
69/69 [==============================] - ETA: 0s - loss: 9.5449 - accuracy:
0.8916Epoch 50/50: loss: 9.5449 - accuracy: 0.8916 - val_loss: 9.4065 -
val_accuracy: 0.9524 - lr: 9e-06
```

Figure 4.23. Fine-tuning log for Coronal model

```
Epoch 44: val_loss did not improve from 10.44492
69/69 [==============================] - 36s 524ms/step - loss: 10.7144
accuracy: 0.6991 - val_loss: 10.5101 - val_accuracy: 0.7500
Epoch 44: early stopping
```

Figure 4.24. Coronal model early stopping log

In the pursuit of optimizing our coronal model training, a more conservative learning rate of 1e-5 was judiciously selected, diverging from the sagittal model's higher initial rate. This decision was informed by observations of deviations at higher rates, suggesting that a lower rate would ensure a more stable and optimal learning trajectory (Figure 4.22).

Throughout the training epochs, the application of the reduction plateau callback played a pivotal role whenever the loss value ceased to improve. Embedded within our learning plan, this callback triggered adjustments to the learning rate after a predetermined patience threshold had been surpassed without improvement in loss, ensuring that learning continued to progress efficiently (Figure 4.22).

Notably, the model's performance, as evidenced by accuracy metrics, demonstrated a high near-90% accuracy in the initial two folds. However, a discernible decline was observed in subsequent folds, with accuracy dropping to the mid-75% range in the third fold and approaching the 70% levels in the fourth and fifth folds. This incremental decrease in model accuracy was indicative of the challenges encountered as training progressed across folds.

Particularly during the training phase in the third fold, training ceased prematurely at the 58th epoch due to the early stopping mechanism. This intervention was prompted by a sustained lack of improvement in the loss value, despite reaching the set patience period. A similar pattern necessitated early stopping during the fine-tuning of the fourth and fifth folds, reflecting the necessity of balancing persistence with the pragmatic recognition of learning plateaus (Figure 4.24).

This nuanced reflection upon our model's performance underscores the imperative of adaptive strategies in the face of fluctuating results and the inherent complexities of model training. While the outcomes from the coronal model training slightly trailed those of the sagittal model, the lessons drawn from this experience are invaluable, highlighting areas for potential refinement and laying the groundwork for future advancements.

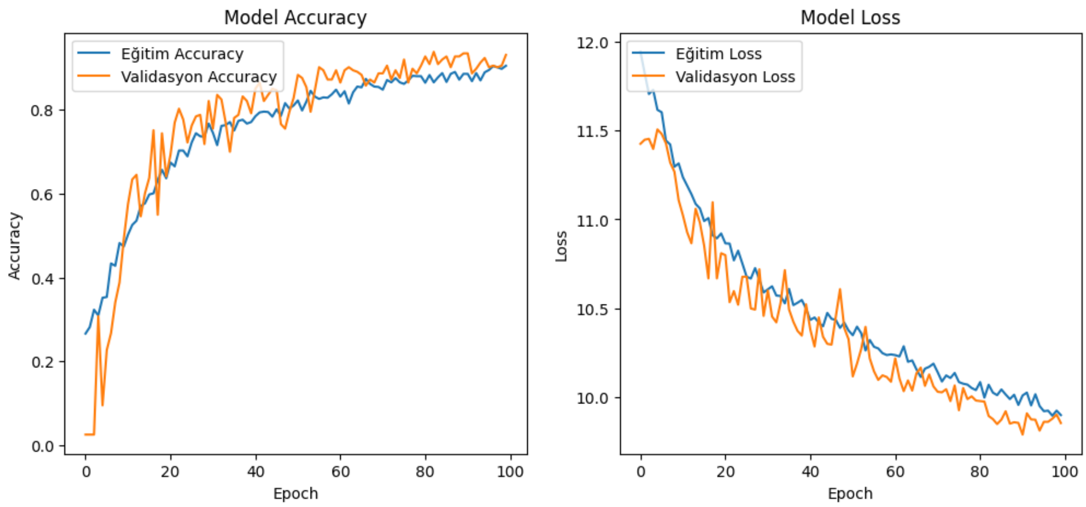

Figure 4.25. Coronal graph fold-1

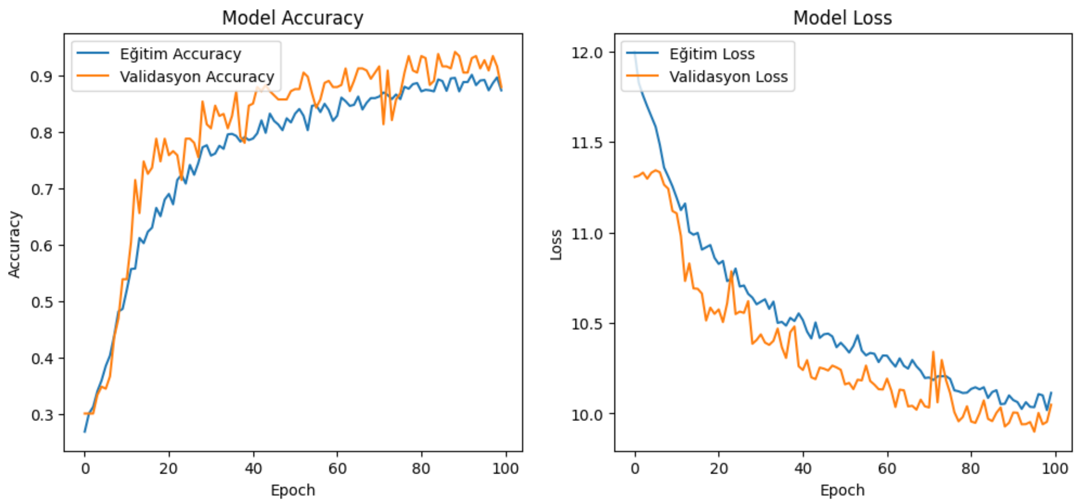

Figure 4.26. Coronal graph fold-2

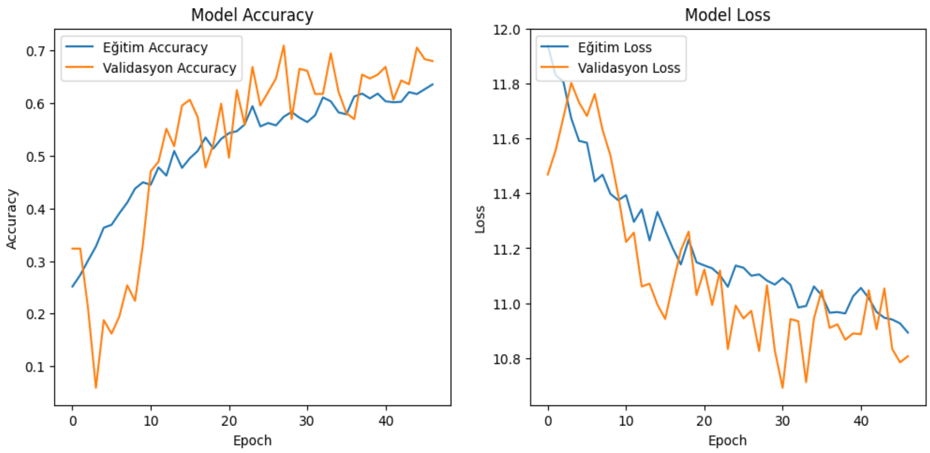

Figure 4.27. Coronal graph fold-3

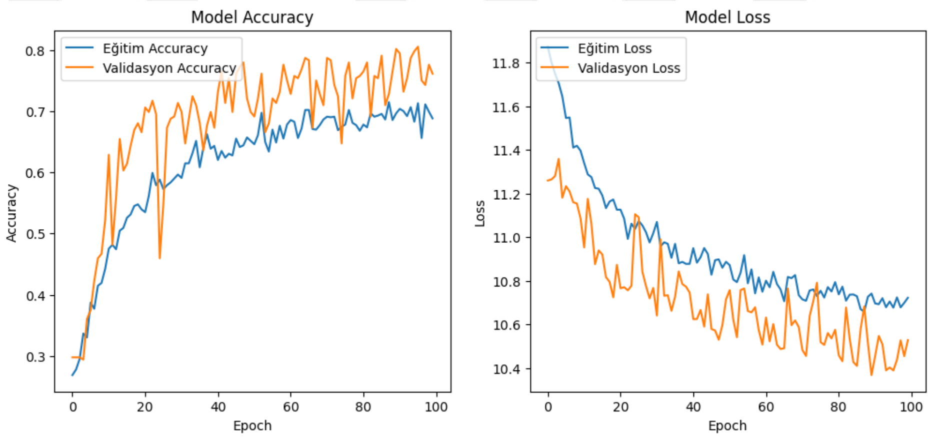

Figure 4.28. Coronal graph fold-4

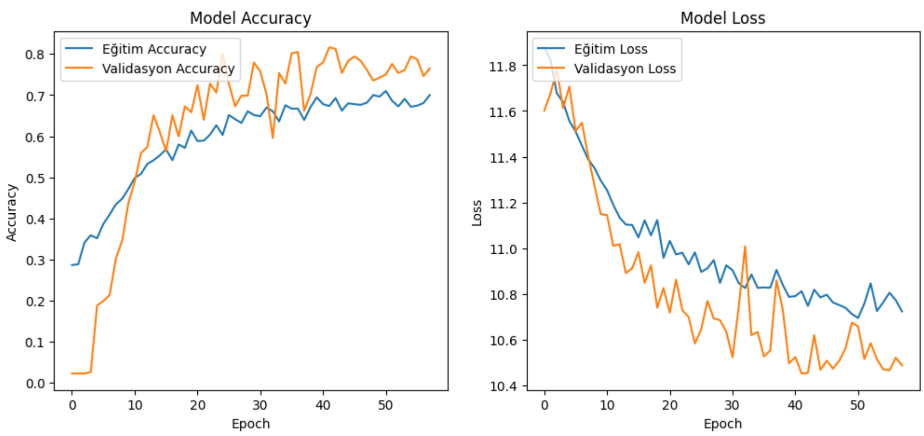

Figure 4.29. Coronal graph fold-5

The analysis of our training results, particularly the initial high accuracy rates observed in the early layers which then begin to decline in subsequent layers, signals the need for a more in-depth examination of the learning process. This trend highlights challenges the model faces in correctly classifying classes that are less represented, indicating limitations in the model’s capacity to handle diversity within the dataset (Figure 4.25, 4.26, 4.27, 4,28 and 4.29).

While the data augmentation techniques employed during training have generally supported the model's performance effectively, our findings suggest that alongside these techniques, there is a need to further enhance the diversity and representation within our dataset and to fine-tune the learning algorithms for handling such scenarios more adeptly.

The occurrences of early stopping during training underscore the importance of re-evaluating our current dataset and learning approaches when faced with these challenges. This is a common situation in training deep learning algorithms, and these results indicate that a more comprehensive approach to expanding and diversifying the training set is required.

The analysis of fluctuations observed in the graphs aids in pinpointing specific areas within the training process that require additional focus. There is a need for deeper contemplation on how certain features can be better learned by the model (Figure 4.27, 4.28 and 4.29).

In conclusion, these observations provide valuable insights into how our model training and dataset construction can be further optimized. Enhancing our training processes will assist in achieving higher and more consistent performance levels in our medical image classification efforts. During this process, balancing and expanding our dataset will be strategically crucial to strengthening our model and advancing the potential of deep learning in medical diagnostics.

```
# ChannelAttention ve SpatialAttention katmanlarının tanımı
class ChannelAttention(Layer):
    def __init__(self, ratio=8, **kwargs):
        super(ChannelAttention, self).__init__(**kwargs)
        self.ratio = ratio

    def build(self, input_shape):
        channel = input_shape[-1]
        self.shared_layer_one = Dense(channel // self.ratio, activation='relu', kernel_initializer='he_normal',
                                      use_bias=True, bias_initializer='zeros')
        self.shared_layer_two = Dense(channel, kernel_initializer='he_normal', use_bias=True, bias_initializer='zeros')
        super(ChannelAttention, self).build(input_shape)

    def call(self, inputs):
        avg_pool = GlobalAveragePooling2D()(inputs)
        avg_pool = Reshape((1, 1, -1))(avg_pool)
        avg_pool = self.shared_layer_one(avg_pool)
        avg_pool = self.shared_layer_two(avg_pool)

        max_pool = GlobalMaxPooling2D()(inputs)
        max_pool = Reshape((1, 1, -1))(max_pool)
        max_pool = self.shared_layer_one(max_pool)
        max_pool = self.shared_layer_two(max_pool)

        cbam_feature = Add()([avg_pool, max_pool])
        cbam_feature = Activation('sigmoid')(cbam_feature)

        return Multiply()([inputs, cbam_feature])

class SpatialAttention(Layer):
    def __init__(self, kernel_size=7, **kwargs):
        super(SpatialAttention, self).__init__(**kwargs)
        self.kernel_size = kernel_size
        self.conv = Conv2D(filters=1, kernel_size=self.kernel_size, strides=1, padding='same',
                           activation='sigmoid', kernel_initializer='he_normal', use_bias=False)

    def call(self, inputs):
        avg_pool = Lambda(lambda x: K.mean(x, axis=-1, keepdims=True))(inputs)
        max_pool = Lambda(lambda x: K.max(x, axis=-1, keepdims=True))(inputs)
        concat = Concatenate(axis=-1)([avg_pool, max_pool])
        cbam_feature = self.conv(concat)

        return Multiply()([inputs, cbam_feature])
```

Figure 4.30. Code for defining attention layer in coronal model prediction

```
# Etiket isimleri
label_names = ['glioma_coronal', 'meningioma_coronal', 'notumor_coronal', 'pituitary_coronal']

# Tahminleri ve gerçek etiketleri toplayacak listeleri oluştur
y_true = []
y_pred = []

# Test klasörlerini dolaş ve tahminleri topla
for tumor_type in os.listdir(test_dir):
    tumor_type_dir = os.path.join(test_dir, tumor_type)
    if not os.path.isdir(tumor_type_dir):
        continue

    for img_name in os.listdir(tumor_type_dir):
        if img_name.lower().endswith('.jpg'):  # Sadece .jpg uzantılı dosyaları işle
            img_path = os.path.join(tumor_type_dir, img_name)
            img = image.load_img(img_path, target_size=(224, 224))
            img_array = image.img_to_array(img)
            img_array = np.expand_dims(img_array, axis=0) / 255.0

            total_predictions = np.zeros(4)  # 4 tane etiket için tahminler
            print(f"Görüntü: {img_name}, Gerçek Etiket: {tumor_type}")

            for i, model in enumerate(models):
                predictions = model.predict(img_array)[0]
                total_predictions += predictions
                print(f"Model {i+1} Tahminleri: {dict(zip(label_names, predictions))}, Tahmin Edilen Etiket: {l

            final_predicted_class = label_names[np.argmax(total_predictions)]
            print("Toplam Tahminler:", dict(zip(label_names, total_predictions)))
            print(f"Final Tahmin Edilen Etiket: {final_predicted_class}")
            print("-" * 50)

            # Gerçek ve tahmin edilen etiket listelerine ekle
            y_true.append(label_names.index(tumor_type))  # Gerçek etiketin indeksini kullan
            y_pred.append(np.argmax(total_predictions))  # Tahmin edilen etiketin indeksini kullan

# Sınıflandırma raporu ve karışıklık matrisini hesapla
report = classification_report(y_true, y_pred, target_names=label_names)
conf_matrix = confusion_matrix(y_true, y_pred)

# Raporları ve karışıklık matrisini yazdır
print("=" * 50)
print("Sınıflandırma Raporu ve Karışıklık Matrisi")
print(report)
print(conf_matrix)
```

Figure 4.31. Prediction code for Coronal model

This code segment is utilized in the coronal model just as it is in the axial and sagittal models to perform deep learning-based predictions (Figure 4.31). The fusion methodology consolidates predictions from multiple models, thereby enhancing the sensitivity of the models in challenging brain tumor classification tasks. For each test image, predictions from five separate models are accumulated, and the class with the highest aggregated score is determined as the final prediction. This method leverages the independent strengths of each model while counterbalancing potential weaknesses, particularly improving performance in the coronal model where data imbalance is prevalent.

In addition to recording the true class labels, the model's performance is assessed using a classification report and confusion matrix. These reports expose the model's ability to distinguish between classes as well as its overall predictive success. The application of this prediction code to the coronal model, following the effective results in axial and sagittal models, reinforces the potential of deep learning techniques in medical diagnostics. Future research aims to fortify this methodology by evaluating these techniques on broader datasets and implementing them in various medical imaging scenarios (Figure 4.31).

```
Görüntü: Tr-pi_0413.jpg, Gerçek Etiket: pituitary_coronal
1/1 [==============================] - 0s 58ms/step
Model 1 Tahminleri: {'glioma_coronal': 0.0, 'meningioma_coronal': 9.1e-06, 'notumor_coronal': 0.0001664, 'pituitary_coronal': 1.0}, Tahmin Edilen
Etiket: pituitary_coronal
1/1 [==============================] - 0s 58ms/step
Model 2 Tahminleri: {'glioma_coronal': 6e-08, 'meningioma_coronal': 4.05e-06, 'notumor_coronal': 6.616e-05, 'pituitary_coronal': 1.0}, Tahmin
Edilen Etiket: pituitary_coronal
1/1 [==============================] - 0s 49ms/step
Model 3 Tahminleri: {'glioma_coronal': 9.06e-06, 'meningioma_coronal': 0.002653, 'notumor_coronal': 0.004417, 'pituitary_coronal': 0.9927},
Tahmin Edilen Etiket: pituitary_coronal
1/1 [==============================] - 0s 49ms/step
Model 4 Tahminleri: {'glioma_coronal': 0.0001974, 'meningioma_coronal': 0.003994, 'notumor_coronal': 0.002787, 'pituitary_coronal': 0.993},
Tahmin Edilen Etiket: pituitary_coronal
1/1 [==============================] - 0s 51ms/step
Model 5 Tahminleri: {'glioma_coronal': 5.37e-05, 'meningioma_coronal': 0.001881, 'notumor_coronal': 0.001422, 'pituitary_coronal': 0.9966},
Tahmin Edilen Etiket: pituitary_coronal
Toplam Tahminler: {'glioma_coronal': 0.00026023387908935547, 'meningioma_coronal': 0.008540928363800049, 'notumor_coronal':
0.008858561515808105, 'pituitary_coronal': 4.982421875}
Final Tahmin Edilen Etiket: pituitary_coronal
```

Figure 4.32. Prediction log for Coronal model

In the development of the coronal model, we have employed a fusion technique for predictions, akin to the approaches utilized in sagittal and axial models. This technique has been instrumental in mitigating the impact of class imbalance within the coronal dataset, contributing to an enhancement in predictive performance. The application of fusion in our predictive framework synthesizes the outputs from multiple models, thus capitalizing on the strengths of each individual model and diluting their weaknesses. This aggregated approach helps in compensating for the underrepresented classes by drawing on a more robust collective inference, which tends to be more resilient to the variations within the training data (Figure 4.32).

The benefits of the fusion technique in the context of our study are multifaceted. Firstly, it provides a buffer against the skewed distribution of classes, which is a common challenge in medical imaging datasets. By integrating predictions from several models, we obtain a more balanced perspective, reflecting a consensus that may offset individual model biases towards overrepresented classes. Secondly, this method has shown to enhance overall accuracy, as it allows for a more democratic and comprehensive representation of the learned features, leading to a refined and consolidated prediction.

The effectiveness of the fusion technique in alleviating the effects of data imbalance in our coronal dataset is particularly noteworthy. It underscores the potential of ensemble learning strategies in deep learning frameworks, especially when dealing with heterogeneous and imbalanced datasets typical of medical imaging studies. The positive influence on the prediction performance reiterates the value of combining multiple models' insights for a more accurate and reliable classification outcome (Figure 4.32).

In summary, the strategic implementation of fusion techniques in the prediction phase of our coronal model has demonstrated a promising avenue for enhancing model robustness and reliability. This approach not only serves to improve accuracy but also provides a template for addressing class imbalance, which is paramount in advancing the field of medical image analysis. The insights gained herein will undoubtedly inform future iterations of our model, guiding further refinements in our pursuit of excellence in medical diagnostics.

```
Sınıflandırma Raporu ve Karışıklık Matrisi
                      precision    recall  f1-score   support

     glioma_coronal       0.94      0.93      0.94       111
 meningioma_coronal       0.93      0.85      0.89       103
    notumor_coronal       0.86      0.67      0.75         9
  pituitary_coronal       0.91      1.00      0.95       121

           accuracy                           0.92       344
          macro avg       0.91      0.86      0.88       344
       weighted avg       0.92      0.92      0.92       344

[[103   7   0   1]
 [  5  88   1   9]
 [  1   0   6   2]
 [  0   0   0 121]]
```

Figure 4.33. Confusion matrix for Coronal model

The classification report and confusion matrix for the coronal model provide a clear window into the model's performance across different classes. A closer examination reveals several key insights (Figure 4.33).

Glioma_coronal class shows high precision and recall, indicating the model's strong ability to correctly identify and classify this type of tumor from the coronal plane images. With a precision of 0.94 and a recall of 0.93, the model demonstrates a balanced approach, neither overly conservative nor liberal in predicting glioma, as supported by the F1-score of 0.94.

Meningioma_coronal class has a slightly lower precision and recall, which suggests there may be some confusion between meningioma and other tumor types. The precision of 0.93 indicates a high rate of true positive predictions when the model claims a meningioma is present, but the recall of 0.85 shows that the model misses some meningioma cases.

Notumor_coronal class is where the model seems to struggle the most, with the lowest recall of 0.67. This implies that the model often fails to recognize instances when no tumor is present, likely predicting another class instead. This is a significant area for improvement, as the support number indicates a limited number of samples for training.

Pituitary_coronal class is well identified by the model, with a high F1-score of 0.95. The perfect recall rate means that the model has correctly identified all cases of pituitary tumors that are present in the test set.

The confusion matrix corroborates these findings, where most of the misclassifications occur with the notumor_coronal and meningioma_coronal classes. The high number of true positives for glioma_coronal and pituitary_coronal validates the model's efficacy for these classes (Figure 4.33).

Considering this analysis, it's evident that the model's predictive performance is strong for axial planes but requires improvement for saggital and coronal planes, particularly for distinguishing between the presence and absence of tumors. Future work could focus on enhancing the model's sensitivity to notumor_coronal instances and refining its ability to differentiate between tumor types with similar radiographic features. Increasing the size and diversity of the training dataset, especially for underrepresented classes, could be a valuable strategy to improve the model's overall accuracy and reliability.

4.6. Summary of Result

4.6.1 Model performance and comparisons

Axial model: The axial model has demonstrated the most stable performance among all the models. A more balanced distribution of the dataset for this model has enabled it to achieve high accuracy rates. The results obtained during the training of this model have provided consistent and reliable outputs, especially in tumor classification.

Sagittal model: The sagittal model has generally produced good results; however, fluctuations in the outcomes have been observed. The primary reason for these fluctuations may be the underrepresentation of non-tumor classes, leading to data imbalances. This situation could have caused the model to be less successful in detecting certain tumor types compared to others.

Coronal model: The performance of the coronal model has tended to decline as the folds progressed. This trend may stem from more pronounced data imbalances in the coronal images. As the folds advanced, the model struggled to distinguish between tumor classes, which led to a decrease in overall performance.

4.6.2 Challenges and solution proposals

The differences between these models illustrate how critical data imbalance is in the training of deep learning models. Data imbalance, particularly with rare tumor types, complicates accurate classification and adversely affects the overall accuracy of the model. To mitigate the effects of data imbalance, various strategies such as data augmentation, Stratified KFold usage, and fusion methods to tolerate outcomes have been implemented in my thesis. However, for a more effective resolution of this issue, there is a need for a greater variety of data, and a more extensive application of data augmentation methods on the existing data could be beneficial.

4.6.3 Role of callback functions

Callback functions used during the training process, such as EarlyStopping, have provided protection against overfitting and optimized the training process of the model. Particularly, early stopping has prevented the model from unnecessary over-training, offering savings in both time and resources.

In conclusion, the comparative analysis of these three models has highlighted the significant impact of different MRI imaging angles on model performance. The stability and high performance provided by the axial model suggest that it could be a more effective option for diagnosing brain tumors. However, the performance fluctuations and declines observed in the sagittal and coronal models underscore the need for additional studies to address data imbalances arising from these imaging angles. Future studies should focus on enhancing data collection and preprocessing techniques to mitigate the issues of data imbalance and improve model robustness across different imaging planes.

5. CONCLUSION

This thesis extensively investigates the effective application of deep learning techniques in classifying brain tumors using MRI images. The study centers particularly on advanced artificial intelligence models such as EfficientNetB7, which have been trained on a large and diverse dataset of thousands of MRI images to distinguish various types of brain tumors with exceptionally high accuracy. The acquisition of MRI images from different angles (axial, sagittal, and coronal) has been specifically addressed in this thesis. Separate models have been developed for each imaging angle, an approach that has been employed to optimize model performance.

Notably, the axial model has attracted attention for its high stability and accuracy rates, establishing it as a prominent method for diagnosing brain tumors. This model, built on axial images, has shown impressive results in recognizing both general and complex features. To mitigate the tendency towards overfitting and enhance the overall robustness of the model, advanced techniques such as data augmentation using ImageDataGenerator and multi-fold cross-validation have been integrated into the study. Furthermore, a fine-tuning phase was implemented following the initial model training phase, particularly to facilitate the learning of more nuanced and detailed features. During this phase, certain layers of the model were frozen to focus on recognizing new and more specific features. These techniques have significantly improved the model's generalization to previously unseen data, thereby greatly enhancing prediction reliability in real-world scenarios. However, the challenges encountered in training the sagittal and coronal models should not be overlooked. Specifically, the coronal model has experienced significant difficulties in accurately classifying non-tumorous regions. This situation highlights the critical importance of data diversity and balance in developing effective diagnostic models. In conclusion, this thesis demonstrates how artificial intelligence and machine learning can redefine the standards of medical diagnosis.

Surpassing traditional diagnostic methods, these technologies facilitate earlier disease detection, enable quicker responses to treatment processes, and potentially contribute to higher survival rates. Future research should aim to test the model with broader datasets and conduct more comprehensive validations across various medical conditions and different populations. Additionally, it is recommended that methodologies be developed, and clinical feedback be more comprehensively integrated to enhance the effectiveness of artificial intelligence applications in the medical field. Such steps will further improve the performance of AI-based diagnostic systems, making them more reliable in medical applications.

6. REFERENCES

Alam, M. S. 2019. Automatic human brain tumor detection in MRI image using template-based K means and improved fuzzy C means clustering algorithm. Big Data and Cognitive Computing, 3(2): 27, May 2019.

Ari, A. and Hanbay, D. 2015. Computer-aided tumor detection system using brain MR images. Medical Technologies Congress, October 2015.

Asliyan, R. and Atbakan, I. 2020. Automatic brain tumor segmentation with K-means, fuzzy C-means, self-organizing map and Otsu methods. Journal of Selcuk Technic, August 2020.

Baraiya, N. and Modi, H. 2016. Comparative study of different methods for brain tumor extraction from MRI images using image processing. Indian Journal of Science and Technology, 9(4), January 2016.

Black, P. McL. 1995. Benign brain tumors: Meningiomas, pituitary tumors, and acoustic neuromas. Neurologic Clinics, 13, 927-952.

Bulut, F. and Incei, F. 2018. Comparison and performance analysis of image segmentation algorithms on brain tumor detection. Journal of Science and Engineering, 20(58), January 2018.

Bzdok, D., Krzywinski, M. and Altman, N. 2018. Machine learning: Supervised methods. Nature Methods, 15(1): 5-6, January 2018.

Chen, M., Yan, Q. and Qin, M. 2017. A segmentation of brain MRI images utilizing intensity and contextual information by Markov random field. Computer Assisted Surgery, 22, 200-211, October 2017.

Cevik, A. 2011. A medical image processing and analysis framework. The Graduate School Of Natural And Applied Sciences Of Middle East Technical University, January 2011.

Correia, A. de S. and Colombini, E. 2021. Attention, please! A survey of neural attention models in deep learning. Artificial Intelligence Review, 55, 6037-6124.

Dandıl, E. 2017. Implementation and comparison of image segmentation methods for detection of brain tumors on MR images. 2nd International Conference on Computer Science and Engineering, 2017.

Gupta, K. K., Dhanda, N. and Kumar, U. 2018. A comparative study of medical image segmentation techniques for brain tumor detection. 2018 4th International Conference on Computing Communication and Automation (ICCCA), December 2018.

Hojjatoleslami, S. and Kittler, J. 1998. Region growing: A new approach. IEEE Transactions on Image Processing, 7(7): 1079-1084, 1998.

Ince, O., Senel, I. K. and Yilmaz, F. 2020. Image processing and analysis in health: Advantages, challenges, threats and examples. Archives of Health Science and Research, 7(1): 66-74, 2020.

Khan, P. Y., Aravind, D. and Reddy, K. K. A. 2019. Recognition of brain tumor utilizing image processing techniques. International Journal of Engineering Research & Technology (IJERT), 8, November 2019.

Kumari, S. V. and Rani, K. U. 2020. Analysis on various feature extraction methods for medical image classification. Learning and Analytics in Intelligent Systems, July 2020, pp. 19-31.

Li, X., Li, M., Yan, P., Li, G., Jiang, Y., Luo, H. and Yin, S. 2023. Deep learning attention mechanism in medical image analysis: Basics and beyond. International Journal of Network Dynamics and Intelligence.

Nerurkar, S. N. 2017. Brain tumor detection using image segmentation. International Journal of Engineering Research in Computer Science and Engineering (IJERCSE), 4(4), April 2017.

Pareek, M., Jha, C., Mukherjee, S. and Joshi, C. 2018. Brain tumor detection using watershed segmentation techniques and area calculation. International Journal of Informatics and Communication Technology (IJ-ICT), 7(2), 71.

Raj, S. K., Dubey, G. M. and Kumar, N. 2019. Medical image analysis using machine learning techniques. International Journal of Computer Sciences and Engineering, 6(9), September 2019.

Sachdeva, J., Kumar, V., Gupta, I., Khandelwal, N. and Ahuja, C. K. 2013. Segmentation, feature extraction, and multiclass brain tumor classification. Journal of Digital Imaging, 26(6): 1141-1150, May 2013.

Sharma, A., Kumar, S. and Singh, S. N. 2018. Brain tumor segmentation using DE embedded OTSU method and neural network. Multidimensional Systems and Signal Processing, 30(3): 1263-1291, July 2018.

Velusamy, P. D. and Karandharaj, P. 2014. Medical image processing schemes for cancer detection: A survey. 2014 International Conference on Green Computing. Communication and Electrical Engineering (ICGCCEE), 2014.

Venkatesh, M. and Leo, 2019. MRI brain image segmentation and detection using KNN classification. Journal of Physics: Conference Series, 1362, 012073, 2019.

Waggoner, R. 1937. Diagnosis of brain tumors. Journal of Nervous and Mental Disease, 85, 727.

Zhu, H., Xie, C., Fei, Y. and Tao, H. 2021. Attention mechanisms in CNN-based single image super-resolution: A brief review and a new perspective. Electronics, 10, 1187.

CURRICULUM VITAE

Uygar Cankat

EDUCATION

Yüksek Lisans 2021-2024	Akdeniz University Graduate School of Natural and Applied Sciences / Computer Engineering (English) (Master with Thesis), Antalya
Lisans 2015-2020	Sabanci University B.S. in Computer Science & Engineering, Faculty of Engineering and Natural Science

PROFESSIONAL AND ADMINISTRATIVE DUTIES

Summer Intern 06.2019 - 08.2019	Enerjisa Business Intelligent