ADVANCED AUTOMAATION FOR COLORECTAL TISSUE CLASSIFICATION IN HISTOPATHALOGY FOR FINAL PROJECT SUBMISSION

(for the partial fulfillment of Bachelor of Technology Degree in

Computer Science & Engineering)

Submitted by

VAIBHAV KUMAR KAPRIYAL

YASH KHAROLA

RAMA KORANGA

ISHITA CHHETRI

Under the guidance of

Dr. Chandradeep Bhatt

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GRAPHIC ERA HILL UNIVERSITY
JUNE, 2024

CERTIFICATE

This is to certify that the thesis titled **"Advanced Automation For Colorectal Tissue Classification In Histopathology" submitted** by **Author**, to Graphic Era Hill University for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him/her under our supervision. The contents of this project in full or in parts have not been submitted to any other Institute or University for the award of any degree or diploma.

Mr. Chandradeep Bhatt
(Assistant Professor)
GEHU, Dehradun

Place: Dehradun
Date: 16-05-2024

ACKNOWLEDGEMENT

We extend our heartfelt gratitude to everyone who contributed to the success of this project. Our sincere appreciation goes to **Mr. Chandradeep Bhatt** for his invaluable support, guidance, and expertise throughout the project journey. His dedication and commitment were instrumental in achieving our objectives.

We also want to express our gratitude to all those whose contributions, no matter how small, made a significant difference.

Furthermore, we acknowledge the support and encouragement from our families, friends, and colleagues, whose unwavering belief in our endeavors fueled our determination.

Lastly, we appreciate the understanding and patience of those who might have been inconvenienced during the project's execution.

Thank you all for being part of this endeavor and for your unwavering support.

Vaibhav Kapriyal
2018391

Yash Kharola
2018391

Rama Koranga
2018391

Ishita Chhetri
2018391

ABSTRACT

Colorectal cancer (CRC) ranks as the commonly occurring cancer worldwide accounting for approximately 10% of all cases. Extensive research has demonstrated that accurate categorization of images plays a role, in determining the progression of colorectal cancer. In histology various tissue types like colon mucosa (NORM) tissue (ADI) polyps, cancer associated stroma (STR) and lymphocytes (LYM) can efficiently provide prognostic indicators through hematoxylin and eosin stains (HE stains). The standard procedure for examining abnormalities on the colons surface including their location, shape and pathological changes is colonoscopy. This procedure significantly enhances accuracy. Helps predict disease severity to ensure appropriate clinical treatment. However, analyzing images for tissue classification demands significant time and effort due, to numerous subjective factors influencing evaluation.

Subjective assessment is typically conducted by pathologists who manually examine the slide images of CRC tissue, which is still considered the standard, for diagnosing and staging cancer. However various factors such as training, experience, evaluation conditions or time constraints may lead to judgments among pathologists. Therefore, it holds importance to have a universally applicable automated classification system, for CRC pathological tissue slide images in order to ensure fair evaluations.

Pathology slides provide an enormous amount of information, which has been quantified through digital pathology and classic machine learning techniques over the years. Previous research has been based on machine learning approaches for judging the cell classification in the histological slides of tumor tissue. The classification of histopathological images using artificial intelligence not only improves the accuracy and efficiency of the classification, but also enables doctors to make timely decisions in terms of clinical

treatment. However, most of the proposed experimental methods rely on manual feature labels, which is the main limitation of traditional

textual analysis approaches. Therefore, deep learning has been introduced in the last few years to solve this and other limitations. Deep learning is a new technology that is considered to be and evolution of machine learning, since it uses multiple layers of neural networks to learn and progressively extract higher-level features in order to reduce human intervention in the recognition of

different classes in the images. It is also effective in classifying non-image data, such as speech recognition, social network filtering, and medical image analysis, and its advanced approach not only reduces the need for human intervention, but it can also automatically

achieve results that are comparable to or surpassing those of humans.

Convolutional neural networks (CNN)recently showed effective results in classifying images in the field of deep learning where a neural network might have dozens or hundreds of layers to learn containing images with different features. A convolutional layer composed of a small-sized kernel to generate advanced features applies weights to the inputs and directs them through an activation function as the output. The main advantage of using CNN compared to a traditional neural network is that it reduces the model parameters for more accurate results.

With this in mind, we aimed to use deep learning technology to identify medical images to increase the accuracy of the identification due to the automatic classification of tumor types. This involved the achievement of the following objectives:

a. To compare the classification accuracy rate with different CNN models.

b. To find the best performance of deep learning techniques.
c. To compare the results of this method with those of existing techniques.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Colorectal cancer is one of the major global health issue due to it high death rate that requires an urgent need for its early detection that can improve patient outcomes. Previous and old methods for diagnosis of colorectal cancer included histopathological inspection, a method that was not only time consuming but highly relies on the medical professions judgment and opinions which would make it a long process. To tackle this problem, our research points out a very different and cutting-edge approach that includes multiple convolutional neural networks to automate the process of classification of histopathological images for multi-class colorectal tissue. An exceptional aspect of our research is the in-depth exploration of the computational time, a critical factor which is often ignored. In our research we found out that our proposed methodology not only improves the classification accuracy but also reduce the computational timing significantly, that makes our research more practical and implementational. Our study not only aims to improve advancement of automated histopathological image classification but also encourages the practical benefits of our model making it a valuable asset in the ongoing efforts to improve early detection and diagnosis of colorectal cancer.

Colorectal cancer (CRC) is one of the major global health issues, ranking third as the cause of death according to recent global cancer survey. In 2022, the American Cancer Society predicted an astounding 1.55 million new cancer cases, resulting in approximately 53,000 deaths in the United States alone. This malice has its origin in the large intestine and is grown due to uncontrolled cell division triggered by genetic mutations. Also, The development of polyps noncancerous growths in the colon or rectum, is a primary fuel for CRC. Out of which adenomatous polyps has a higher chance developing into cancer, and those cells containing the cancerous element is called malignant polyps. CRC detection has always been a complex task for the clinicians and researchers. Old and conventional methods for diagnosis such as fecal occult blood and colonoscopy proves to be effective but they also lack precision and posed safety concerns due to natural differences.

One of the emerging factors in the early diagnosis of CRC has been Medical imaging. But, despite the increasing availability of medical imaging data, the analysis has been proven to be

consuming and challenging, which can lead to delay in early detection. Misinterpretation further has a negative impact on the accuracy, which requires need for real-time, precise, and objective diagnostic results. This paper introduces a methodology which include classification between four Convolutional Neural Network models that are InceptionResNetV2, Xception, VGG16, and DenseNet121 which compares the multi-class classification of CRC cancer tissue microscopic images to give the best and most accurate result in the most minimize time. Working on three different data sets NCT-CRC-HE-100K, CRC-VAL-HE-7K and Kather-texture-2016-image we have shown the effectiveness and accuracy of our methodology across diverse colorectal tissue histopathological images.

Major contribution of our work includes the proposal of our methodology, brief survey we performed to compare the effectiveness and efficiency of different proposed models and methodology, a brief introduction of all the dataset we have accessed for the effectiveness and robustness of our methodology. It also contains but is not limited to model, algorithms and architecture of our methodology. The presents result showcases the superiority of our model in terms of efficiency, sensitivity, precision, accuracy and F-1 score. Furthermore, we extend the robustness assessment of CRCCN-Net to lung cancer datasets, showcasing its versatility in classifying various tissues.

1.1. Methods of Colorectal Tissue Classification

1. Histopathology:
    - H&E Staining: Hematoxylin and eosin (H&E) staining is the standard method used in histopathology to distinguish different tissue types. Hematoxylin stains cell nuclei blue, while eosin stains the extracellular matrix and cytoplasm pink.
    - Immunohistochemistry (IHC): This technique uses antibodies to detect specific antigens in the tissues, helping to identify different types of cells and abnormalities more precisely.
2. Molecular Techniques:
    - Genetic Profiling: Techniques like PCR, FISH, and next-generation sequencing can identify genetic mutations and molecular markers that are characteristic of certain types of colorectal cancer.

- Microarray Analysis: This can be used to study gene expression profiles, which helps in understanding the underlying biology of the cancer and identifying potential therapeutic targets.

3. Imaging Techniques:
    - Endoscopy: Visual examination of the colorectal mucosa using a colonoscope can help identify suspicious areas that need biopsy.
    - Advanced Imaging: Techniques like confocal laser endomicroscopy and narrow-band imaging can enhance the visualization of tissue architecture and vascular patterns.

1.2. Classification Systems

1. Histological Types:
    - Adenocarcinoma: The most common type of colorectal cancer, originating from glandular cells.
    - Mucinous Adenocarcinoma: Characterized by the production of significant amounts of mucus.
    - Signet Ring Cell Carcinoma: A rare and aggressive form where cells contain prominent vacuoles of mucin.
2. Grading:
    - Low Grade (Well-differentiated): Cells resemble normal cells and tend to grow slowly.
    - High Grade (Poorly differentiated): Cells look very different from normal cells and tend to grow and spread more aggressively.
3. Staging:
    - Based on the TNM system (Tumor, Node, Metastasis), which considers the size and extent of the tumor, involvement of lymph nodes, and the presence of distant metastases.

1.3. Emerging Technologies

1. Artificial Intelligence and Machine Learning:
    - AI algorithms can analyze histopathological images to detect patterns and classify tissues with high accuracy, assisting pathologists in diagnosis.

2. Liquid Biopsy:
    - Non-invasive technique that analyzes circulating tumor DNA (ctDNA) in the blood, providing information about the genetic profile of the tumor.

1.4. Importance in Clinical Practice

1. Diagnosis: Accurate classification is essential for diagnosing the type and stage of colorectal cancer, which directly influences treatment decisions.
2. Prognosis: Understanding the histological type and molecular characteristics of the tumor helps predict the likely course and outcome of the disease.
3. Treatment Planning: Classification guides the selection of appropriate therapies, such as surgery, chemotherapy, targeted therapy, and immunotherapy.

1.5. Type of Colorectal Tissues

1. Normal Colorectal Tissues
    i. Mucosa:
        - Epithelium: The innermost layer consisting of glandular cells that line the colon and rectum, responsible for absorption and secretion.
        - Lamina Propria: A layer of connective tissue containing blood vessels, lymphatics, and immune cells.
        - Muscularis Mucosae: A thin layer of muscle that separates the mucosa from the submucosa.
    ii. Submucosa:
        - A layer of connective tissue containing blood vessels, nerves, and lymphatic vessels. It provides support to the mucosa and contains immune cells.
    iii. Muscularis Propria:
        - Composed of two layers of muscle (inner circular and outer longitudinal) responsible for the peristaltic movements that propel contents through the colon.
    iv. Serosa/Adventitia:
        - The outermost layer. The serosa is a smooth membrane covering the colon, while the adventitia is a connective tissue layer that attaches the colon to surrounding structures.
2. Abnormal/Pathological Tissues

i. Adenomatous Tissues:

- Tubular Adenomas: Benign growths that have a tubular glandular structure.
- Villous Adenomas: Adenomas with finger-like projections. They have a higher risk of becoming malignant.
- Tubulovillous Adenomas: Adenomas with both tubular and villous features.

ii. Inflammatory Tissues:

- Hyperplastic Polyps: Generally benign growths that result from abnormal mucosal regeneration and repair.
- Inflammatory Polyps: Typically associated with chronic inflammatory conditions like ulcerative colitis or Crohn's disease.

iii. Malignant Tissues:

- Adenocarcinoma: The most common type of colorectal cancer, arising from the glandular epithelium.
- Mucinous Adenocarcinoma: Characterized by the presence of significant amounts of mucus.
- Signet Ring Cell Carcinoma: Contains cells with prominent mucin vacuoles displacing the nucleus, a rare and aggressive form.
- Neuroendocrine Tumors (NETs): Arise from neuroendocrine cells, often found in the gastrointestinal tract.
- Squamous Cell Carcinoma: Rare in the colon, more common in the anal canal.

iv. Other Neoplastic Tissues:

- Lymphoma: Cancer originating from lymphocytes, can occur in the colorectal region.
- Gastrointestinal Stromal Tumors (GISTs): Rare tumors arising from the interstitial cells of Cajal in the digestive tract.

3. Inflammatory and Reactive Tissues

i. Ulcerative Colitis:

- Chronic inflammation leading to ulceration of the mucosal layer.

ii. Crohn's Disease:

- Can affect any part of the gastrointestinal tract, causing transmural inflammation and granulomas.

1.6. PROBLEM STATEMENT:

ADVANCED AUTOMAATION FOR COLORECTAL TISSUE CLASSIFICATION IN HISTOPATHALOGY :

Histopathology, the study of diseased tissue under a microscope, plays a crucial role in diagnosing colorectal cancer, one of the most common cancers worldwide. Traditional methods involve manual examination by pathologists, which is time-consuming, subject to human error, and requires extensive expertise. Advanced automation using machine learning and artificial intelligence (AI) presents a promising solution to enhance accuracy, efficiency, and reproducibility in tissue classification.

The primary objective of this project is to develop an advanced automated system for colorectal tissue classification in histopathology. This system should leverage state-of-the-art machine learning algorithms, particularly deep learning techniques, to accurately distinguish between different types of colorectal tissues, such as normal, adenomatous, and cancerous tissues.

Developing an advanced automated system for colorectal tissue classification in histopathology has the potential to revolutionize diagnostic practices, providing faster, more accurate, and consistent results. By addressing the outlined objectives and challenges, this project aims to deliver a practical and impactful solution that can be adopted widely in the medical community.

involves the development of an advanced automated system for classifying colorectal tissue in histopathology, a critical process for diagnosing colorectal cancer. Traditional manual examination by pathologists is not only time-consuming and labor-intensive but also prone to human error, potentially leading to diagnostic inaccuracies. By leveraging cutting-edge machine learning and deep learning techniques, this project aims to create a robust and efficient system capable of distinguishing between various tissue types such as normal, adenomatous, and cancerous tissues with high precision. The proposed system will involve the acquisition and preprocessing of a comprehensive dataset of histopathological images, the development and training of sophisticated models, and the integration of these models into user-friendly software tools. Additionally, the system will be rigorously validated through clinical testing to ensure its reliability and

effectiveness in real-world medical settings, ultimately enhancing diagnostic accuracy, improving efficiency, and facilitating large-scale screenings.

1.7. TECHNOLOGIES USED:

The project "Advanced Automation for Colorectal Tissue Classification in Histopathology" aims to revolutionize the diagnosis of colorectal cancer by leveraging a combination of machine learning (ML), artificial intelligence (AI), and image processing techniques. Each of these technologies plays a crucial role in developing a system that can accurately and efficiently classify different types of colorectal tissues, such as normal, adenomatous, and cancerous tissues, from histopathological images.

Key Components and Their Roles

1. **Machine Learning (ML)**:
    - **Model Development**: The heart of this project involves developing ML models, particularly deep learning models like Convolutional Neural Networks (CNNs). CNNs are highly effective for image classification tasks due to their ability to automatically learn and extract hierarchical features from images.
    - **Training Process**: The ML models are trained using a large dataset of annotated histopathological images. The training process involves feeding the models with labeled images, allowing them to learn the distinguishing features of each tissue type.
    - **Evaluation Metrics**: Performance metrics such as accuracy, precision, recall, F1-score, and area under the receiver operating characteristic curve (AUC-ROC) are used to evaluate and fine-tune the models.
2. **Artificial Intelligence (AI)**:
    - **Automated Decision Making**: AI techniques are applied to enable the system to make diagnostic decisions based on the ML models' outputs. This involves integrating predictive models into a pipeline that can process new, unseen images and classify them accurately.
    - **Workflow Integration**: AI helps in creating an end-to-end automated system that seamlessly integrates with existing digital pathology workflows. This includes

automating the entire process from image acquisition and preprocessing to classification and result reporting.

- **Continuous Learning**: AI systems can be designed to continuously learn from new data and feedback, improving their performance over time. This is crucial for adapting to variations in tissue samples and imaging conditions.

3. **Image Processing**:
    - **Preprocessing**: Image processing techniques are essential for preparing the raw histopathological images for analysis. This includes:
        - **Normalization**: Standardizing image size, resolution, and color intensity to ensure consistency across the dataset.
        - **Noise Reduction**: Applying filters to remove artifacts and noise that could interfere with accurate classification.
        - **Segmentation**: Identifying and isolating regions of interest (ROIs) within the images that contain the relevant tissue structures.
    - **Feature Extraction**: While deep learning models often perform automatic feature extraction, traditional image processing methods can also be used to extract specific features that might enhance model performance. These features can include texture, shape, and color characteristics of the tissues.

Implementation Steps

1. **Data Acquisition and Annotation**:
    - Collect a large and diverse set of histopathological images of colorectal tissues from public databases and medical institutions.
    - Collaborate with expert pathologists to annotate the images, marking the regions corresponding to different tissue types.
2. **Model Training and Validation**:
    - Preprocess the images to ensure they are suitable for input into the ML models.
    - Train various deep learning models, primarily CNNs, using the annotated dataset.
    - Validate the models using cross-validation and hold-out test sets to ensure robustness and generalizability.
3. **System Deployment and Integration**:

- Develop a software tool or integrate the classification models into existing digital pathology platforms.
- Implement automation features to handle large batches of images, providing real-time classification and diagnostic suggestions.
- Create user-friendly interfaces and visualization tools to help pathologists review and interpret the AI-generated results.

4. **Clinical Testing and Feedback**:
    - Conduct extensive testing of the system in real-world clinical settings to validate its performance and reliability.
    - Establish a feedback loop with pathologists to continuously refine and improve the models based on real-world data and expert feedback.

Challenges and Solutions

- **Data Quality and Annotation**: Ensuring high-quality, accurately annotated data is crucial. This can be addressed by collaborating with experienced pathologists and using rigorous validation techniques.
- **Model Interpretability**: Making AI decisions transparent and interpretable is essential for gaining trust from pathologists. This can be achieved by incorporating explainable AI techniques that highlight the features and reasoning behind each classification.
- **Seamless Integration**: The system must integrate smoothly with existing workflows to avoid disrupting routine practices. This involves designing user-friendly software and ensuring compatibility with current digital pathology systems.

Expected Outcomes

- **Enhanced Diagnostic Accuracy**: The system aims to achieve higher accuracy in tissue classification compared to traditional methods, reducing the likelihood of misdiagnosis.
- **Increased Efficiency**: Automating the classification process significantly reduces the time pathologists spend on routine diagnostic tasks, allowing them to focus on more complex cases.
- **Scalability**: The automated system can handle large volumes of histopathological images, facilitating widespread screening and early detection programs.

By combining ML, AI, and image processing, this project aims to deliver a cutting-edge solution that transforms colorectal tissue classification in histopathology, ultimately improving patient outcomes and advancing the field of digital pathology.

In addition to machine learning (ML), artificial intelligence (AI), and image processing, several other technologies and methodologies play crucial roles in the development and deployment of an advanced automated system for colorectal tissue classification in histopathology. These include:

1. Big Data and Data Management

- **Data Storage Solutions**:
    - **Cloud Storage**: Services like Amazon S3, Google Cloud Storage, and Azure Blob Storage provide scalable and reliable storage for large volumes of histopathological images.
    - **On-Premises Data Lakes**: For organizations preferring on-premises solutions, data lakes can store and manage vast amounts of unstructured data efficiently.
- **Database Management**:
    - **Relational Databases**: SQL databases (e.g., MySQL, PostgreSQL) for managing structured data, such as image metadata and annotations.
    - **NoSQL Databases**: For handling large-scale and unstructured data, NoSQL databases like MongoDB and Cassandra are often used.

2. High-Performance Computing (HPC)

- **GPU Acceleration**:
    - **GPUs**: NVIDIA GPUs (e.g., Tesla, A100) are commonly used for training deep learning models due to their ability to handle large-scale parallel processing tasks efficiently.
    - **TPUs**: Tensor Processing Units (offered by Google Cloud) are optimized for accelerating deep learning workloads.
- **Distributed Computing**:
    - **Apache Spark**: A unified analytics engine for big data processing, which can distribute tasks across multiple nodes in a cluster.
    - **TensorFlow Distributed**: TensorFlow provides distributed training capabilities to scale model training across multiple machines.

3. Software Development and Engineering

- **Frameworks and Libraries**:
    - **Deep Learning Frameworks**: TensorFlow, PyTorch, and Keras are the primary tools for building, training, and deploying deep learning models.
    - **Image Processing Libraries**: OpenCV, scikit-image, and PIL (Python Imaging Library) are used for various preprocessing tasks such as resizing, noise reduction, and segmentation.
- **Application Development**:
    - **Backend Development**: Frameworks like Flask and Django for creating APIs and web applications that interface with the ML models.
    - **Frontend Development**: JavaScript frameworks like React or Angular for developing user-friendly web interfaces for pathologists.

4. Data Annotation Tools

- **Annotation Software**:
    - **Labelbox**: A data training platform for labeling data, managing projects, and creating workflows.
    - **VGG Image Annotator (VIA)**: A lightweight, standalone, and offline image annotation tool.
- **Crowdsourcing Platforms**:
    - **Amazon Mechanical Turk**: A crowdsourcing marketplace that can be used to collect annotations from a broader audience, supervised by expert pathologists to ensure accuracy.

5. Model Deployment and Serving

- **Model Serving Platforms**:
    - **TensorFlow Serving**: A flexible, high-performance serving system for machine learning models, designed for production environments.
    - **TorchServe**: A PyTorch model serving library developed by AWS and Facebook.
    - **Cloud-based AI Services**: AWS SageMaker, Google AI Platform, and Azure Machine Learning provide end-to-end services for building, training, and deploying ML models.

- **API Development**:
    - **RESTful APIs**: Built using frameworks like Flask or Django REST framework to facilitate communication between different system components and enable integration with external systems.

6. Data Security and Compliance

- **Data Encryption**:
    - **Encryption at Rest**: Protecting stored data using encryption technologies like AES-256.
    - **Encryption in Transit**: Using protocols like TLS (Transport Layer Security) to secure data during transmission.
- **Compliance**:
    - **HIPAA**: Health Insurance Portability and Accountability Act, which sets the standard for protecting sensitive patient data in the US.
    - **GDPR**: General Data Protection Regulation, which governs data protection and privacy in the European Union.

7. User Experience (UX) and Human-Computer Interaction (HCI)

- **User Interface Design**:
    - **Prototyping Tools**: Tools like Sketch, Figma, and Adobe XD are used to design and prototype user interfaces.
    - **Usability Testing**: Conducting tests with real users (pathologists) to ensure the interface is intuitive and meets their needs.
- **Visualization Tools**:
    - **Heatmaps**: Visual representations of model confidence across different regions of an image, helping pathologists understand the AI's decisions.
    - **Overlay Images**: Highlighting specific areas of interest detected by the model, superimposed on the original histopathological image.

8. Continuous Integration and Continuous Deployment (CI/CD)

- **Automation Pipelines**:
    - **Jenkins**: An open-source automation server that can be used to automate the building, testing, and deployment of software.

    - **GitLab CI/CD**: Integrated CI/CD pipelines in GitLab to streamline the development workflow.
    - **CircleCI**: A CI/CD service that automates the software development process, from building to testing and deploying.

9. Monitoring and Logging

- **Performance Monitoring**:
    - **Prometheus**: An open-source monitoring and alerting toolkit, often paired with Grafana for visualization.
    - **Grafana**: A powerful tool for creating, exploring, and sharing dashboards with your team to monitor the health of the system.
- **Error Logging**:
    - **ELK Stack**: Elasticsearch, Logstash, and Kibana stack used for centralized logging, making it easier to track and troubleshoot errors and performance issues.

By integrating these technologies, the project aims to create a robust, efficient, and scalable system for automated colorectal tissue classification in histopathology. The system is designed to enhance diagnostic accuracy, improve workflow efficiency for pathologists, and facilitate large-scale screenings, ultimately improving patient outcomes.

CHAPTER 2

REQUIREMENT ANALYSIS

2.1. HARDWARE REQUIREMENT

1. Microscopy Equipment:
    - Standard Light Microscope: For routine histopathological examination using H&E-stained slides.
    - Digital Pathology Scanner: For high-resolution scanning of slides, enabling digital pathology workflows.
    - Confocal and Electron Microscopes: For more detailed tissue structure analysis at a cellular and subcellular level.
2. Computing Infrastructure
    - High-Performance Computers: For processing and analyzing large histopathology images. Key specifications include:
        - Multi-core processors (e.g., Intel i7/i9 or AMD Ryzen 7/9)At
        - least 16-32 GB of RAM
        - High-end graphics processing units (GPUs), such as NVIDIA RTX series, for image processing and AI model training
        - SSD storage for faster data access and retrieval
    - Servers and Cloud Computing Resources: For large-scale data storage, processing, and sharing across institutions. Cloud platforms like AWS, Google Cloud, and Azure offer scalable solutions for data-intensive tasks.

3. Imaging and Diagnostic Devices:
    - Endoscopes and Colonoscopes: For visual inspection and biopsy collection.
    - Imaging Systems: MRI, CT, and PET scanners for advanced diagnostic imaging and staging.

2.2. Software Requirements

1. Digital Pathology Software:
    - Image Management Systems: Software for managing and navigating high-resolution digital slides (e.g., Aperio, HALO).
    - Image Analysis Tools: For quantitative analysis and pattern recognition in tissue samples (e.g., ImageJ, QuPath).
2. Artificial Intelligence and Machine Learning Tools:
    - AI Frameworks: Libraries and frameworks for developing and deploying machine learning models (e.g., TensorFlow, PyTorch, Keras).
    - Pre-trained Models and Custom Solutions: Software like PathAI, Paige, or IBM Watson for Healthcare that offers AI-driven diagnostic support.
3. Molecular Diagnostic Software:
    - Genomic Analysis Tools: Software for analyzing genetic data from next-generation sequencing (NGS) (e.g., GATK, BaseSpace).
    - Bioinformatics Platforms: Tools for integrating and analyzing multi-omics data (e.g., Bioconductor, Galaxy).
4. Electronic Health Record (EHR) Systems:
    - Integration with Diagnostic Tools: EHR systems that support integration with diagnostic imaging and pathology results (e.g., Epic, Cerner).

2.3. Technical Expertise

1. Pathologists and Technicians:
    - Experienced Pathologists: For interpreting histopathological slides and making diagnostic decisions.
    - Laboratory Technicians: Skilled in preparing tissue samples, staining, and operating diagnostic equipment.
2. Data Scientists and Bioinformaticians:
    - Data Analysis Expertise: For developing and applying machine learning models to tissue classification tasks.
    - Bioinformatics Knowledge: For handling molecular diagnostic data and integrating various data types.

3. IT Support and System Administrators:
    - Infrastructure Maintenance: Ensuring the smooth operation of hardware and software systems.
    - Cybersecurity Experts: Protecting sensitive patient data and maintaining data integrity.

2.4. Network and Security Requirements

1. Secure Data Storage and Transmission:
    - HIPAA Compliance: Ensuring systems meet regulatory standards for protecting patient information.
    - Encryption: Both at rest and in transit to safeguard data from unauthorized access.
2. Network Infrastructure:
    - High-Speed Internet: For efficient data transfer and remote collaboration.
    - Robust Network Architecture: To support large-scale data processing and storage requirements.

2.5. Dataset Requirements

- **NCT-CRC-HE-100K:** This dataset comprises 100,000 distinct images patched with hematoxylin & eosin-stained histological images showcasing human colorectal cancer (CRC) and normal tissue. Each image is standardized to 224x224 pixels (px) at a resolution of 0.5 microns per pixel. Utilizing Macenko's method, color normalization has been applied to all images.

  The images of 86 total H&E-stained slides of human cancer tissue were manually gathered. the NCT Biobank (National Center for Tumor Diseases, Heidelberg, Germany) and the UMM pathology archive (University Medical Center Mannheim, Mannheim, Germany) originated these slides from Formalin-fixed tissue(FFPE blocks). The sample collected mainly comprises of slides of primary tumors in colorectal cancer (CRC) and tissue from liver metastases in CRC. To enhance diversity, We combined regular tissue types with areas in the stomach surgery samples that weren't affected by tumors.

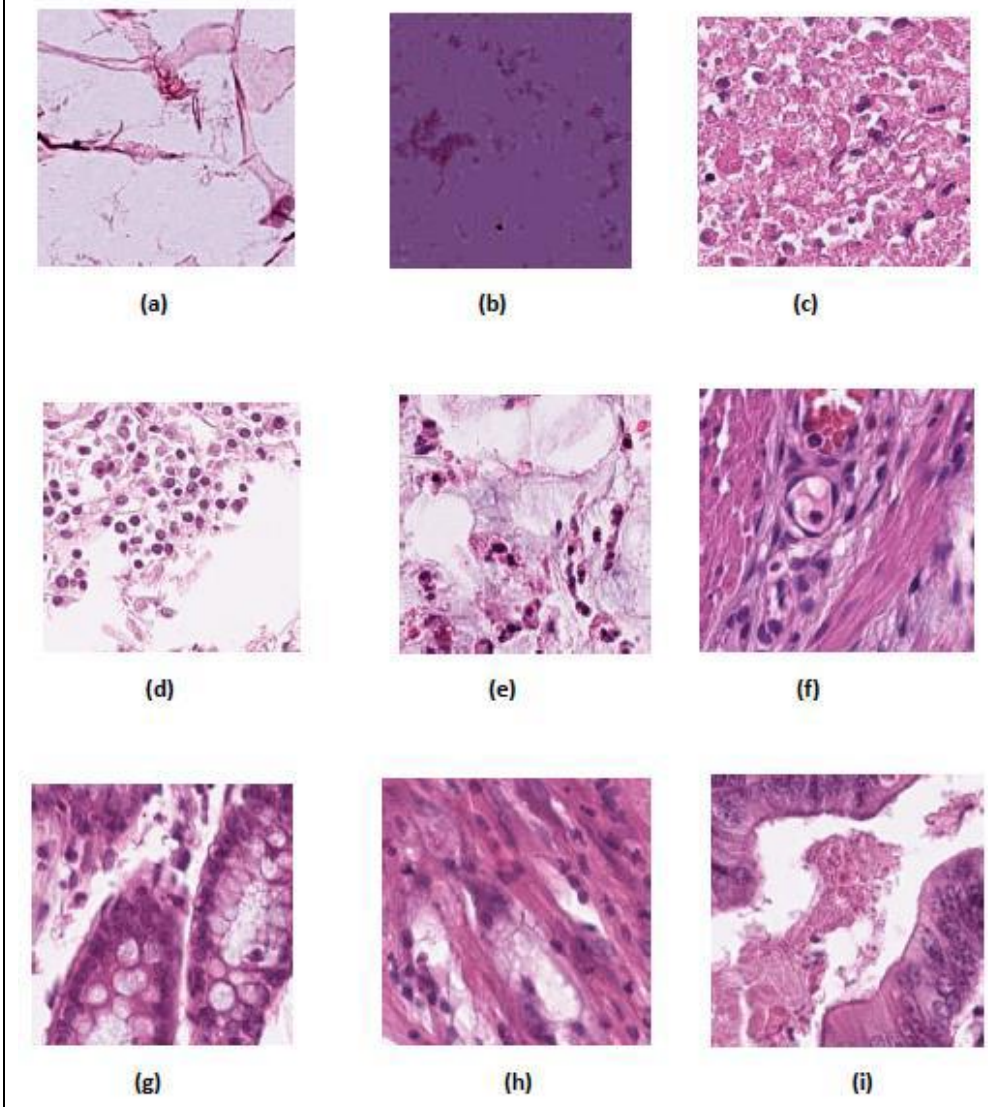

Figure 1. Sample pictures showcasing the nine types of tissues found in the NCT-CRC-HE-100K dataset.

- **CRC-VAL-HE-7K:** The dataset comprises of 7,180 patched images of 50 patients diagnosed with colorectal adenocarcinoma. The most important feature of the NCT-CRC-HE-100K dataset is no patients feature overlaps. The NCT-CRC-HE-100K dataset was primarily designed for the validation of the model, but it shown promise for training of the model as well on more extensive dataset. Similar to the extensive dataset, we've standardized all images to a size of 224x224 pixels, maintaining a resolution of 0.5 microns per pixel (MPP).

All tissue samples were generously provided by the NCT tissue bank, and additional details, including ethical considerations, can be found in the information provided below.

- BACK: Background for microscopic images.
- DEB: Debris, used mainly for the purpose of diagnosis in cancer treatment.
- LYM: Lymphocytes, are a type of white blood cell used by our immune system.
- MUC: Mucus, it is a viscous fluid produced by cell for protection and moistening the muscles.
- MUS: type of a muscle in our body which is very flexible.
- NORM TISSUE: a type of tissues found in colon mucosa.
- STR: Stroma, is a type of tissue in organs that used to support and connect tissues.
- TUM: Epithelial tissues one of the major kind of tissue

- **Images of Eight Tissue Classes:** We used the Kather-texture-2016-image a free to use dataset clearly for only one purpose of assessing the accuracy and precision of our deep learning AI model on various tissue classes. The data set was accumulated at the Institute of Histological Images of Pathology of Human Colorectal Cancer and acquired from the pathology archive by Kather. The dataset contains 5000 unique microscopic images of CRC and normal tissue, employing hematoxylin and eosin staining. Every image in the dataset belongs to one of the eight distinct tissue texture features and are re-formatted to the dimensions of 150 x 150 pixels (74 x 74 μm) for each RGB color channel. The original tissue images within the dataset are sized at 5,000 pixels.
- **Kather-texture-2016-image :** The data contains RGB format 5000 images with a bit resolution of 0.495 μm. The digitalization of the images was done by an Aperio ScanScope(specifically the Aperio/Leica biosystems) with a amplification rate of 20x. These microscopic images were derived from human colorectal adenocarcinomas preserved through formalin fixation and paraffin embedding, specifically primary tumors and are represented as fully anonymized images. These images were sourced from the pathological record repository at the Institute of Pathology, University Medical Center Mannheim, affiliated with Heidelberg University in Mannheim, Germany.

- **Images of Nine Tissue Classes**: In our research, We utilized an accessible cellular dataset of NCT-CRC-HE-100K which include nine distinct tissue classes, for the training and testing of our model. These images are created by Kather which include 86 slides of tissue stained with hematoxylin and eosin. The information on these histological image labels in the provided data was retrieved from the NCT-UMM website. Figure 2 presents sample images representing the nine tissue classes. All these images are resized to a constant dimension of 224 x 224 pixels (112 x 112 μm) and were used for the training, validation, and testing of the model.

  Upon completing the training and testing phases with the "NCT-CRC-HE-100K" dataset, We conducted an assessment to determine the precision of tissue classification model through external validation set denoted as "CRC-VAL-HE-7K." This set comprised 7,180 image patches specifically intended for testing purposes. The nine classes were categorized as follows:

  - ADI: Adipose tissue are connective tissue in the body mainly made of fat cells called adipocytes.
  - BACK: Background for microscopic images.
  - DEB: Debris, used mainly for the purpose of diagnosis in cancer treatment.
  - LYM: Lymphocytes, are a type of white blood cell used by our immune system.
  - MUC: Mucus, it is a viscous fluid produced by cell for protection and moistening the muscles.
  - MUS: type of a muscle in our body which is very flexible.
  - NORM TISSUE: a type of tissues found in colon mucosa.
  - STR: Stroma, is a type of tissue in organs that used to support and connect tissues.
  - TUM: Epithelial tissues one of the major kind of tissue

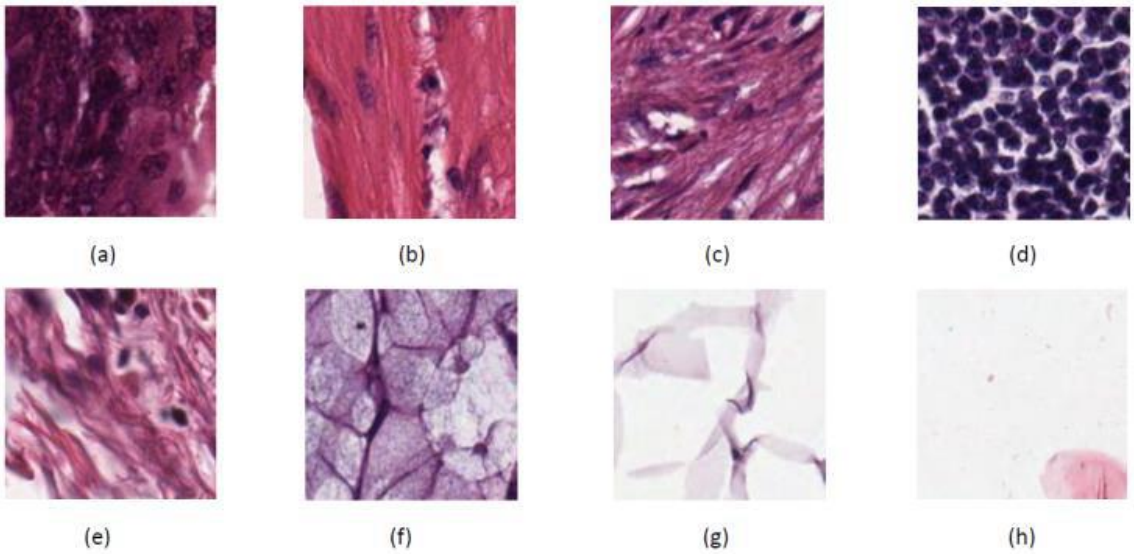

Figure 2. Sample pictures illustrating the eight types of tissues present in the Kather-texture-2016-image-5000 dataset.

- Images of Eight Tissue Classes: We used the Kather-texture-2016-image a free to use dataset clearly for only one purpose of assessing the accuracy and precision of our deep learning AI model on various tissue classes. The data set was accumulated at the Institute of Histological Images of Pathology of Human Colorectal Cancer and acquired from the pathology archive by Kather. The dataset contains 5000 unique microscopic images of CRC and normal tissue, employing hematoxylin and eosin staining. Every image in the dataset belongs to one of the eight distinct tissue texture features and are re-formatted to the dimensions of 150 x 150 pixels (74 x 74 μm) for each RGB color channel. The original tissue images within the dataset are sized at 5,000 pixels.[3] Figure 3. Sample pictures illustrating the eight types of tissues present in the Kather-texture-2016-image-5000 dataset.

Dataset	Diagnosis	Entire		Training	
		#WSI	(%)	#WSI	(%)
NCT-CRC-HE-100K	ADI	10,407	10.41	7285	10.41
	BACK	10,566	10.57	7396	10.57
	DEB	11,512	11.51	8058	11.51
	LYM	11,557	11.56	8090	11.56
	MUC	8896	8.90	6227	8.90
	MUS	13,536	13.54	9475	13.54
	NORM	8763	8.76	6134	8.76
	STR	10,446	10.45	7312	10.45
	TUM	14,317	14.32	10,022	14.32
CRC-VAL-HE-7K	ADI	1338	18.64	0	0
	BACK	847	11.80	0	0
	DEB	339	4.72	0	0
	LYM	634	8.83	0	0
	MUC	1035	14.42	0	0
	MUS	592	8.25	0	0
	NORM	741	10.32	0	0
	STR	421	5.86	0	0
	TUM	1233	17.17	0	0
Kather-texture-2016	TUMOR	625	78.125	468	12.48
	STORMA	625	78.125	468	12.48
	COMBLEX	625	78.125	468	12.48
	LYMPHO	625	78.125	468	12.48
	DEBRIS	625	78.125	468	12.48
	MUCOSA	625	78.125	468	12.48
	ADIPOSE	625	78.125	468	12.48
	EMPTY	625	78.125	468	12.48

Table 1. CRC dataset of histological images for Training.

Dataset	Diagnosis	Validate		Testing	
		#WSI	(%)	#WSI	(%)
NCT-CRC-HE-100K	ADI	1561	10.41	1561	10.41
	BACK	1585	10.57	1585	10.57
	DEB	1727	11.51	1727	11.51
	LYM	1734	11.56	1734	11.56
	MUC	1334	8.90	1334	8.90
	MUS	2030	13.54	2030	13.54
	NORM	1314	8.76	1314	8.76
	STR	1567	10.45	1567	10.45
	TUM	2148	14.32	2148	14.32
CRC-VAL-HE-7K	ADI	0	0	0	0
	BACK	0	0	0	0
	DEB	0	0	0	0
	LYM	0	0	0	0
	MUC	0	0	0	0
	MUS	0	0	0	0
	NORM	0	0	0	0
	STR	0	0	0	0
	TUM	0	0	0	0
Kather-texture-2016	TUMOR	93	12.48	93	12.48
	STORMA	93	12.48	93	12.48
	COMBLEX	93	12.48	93	12.48
	LYMPHO	93	12.48	93	12.48
	DEBRIS	93	12.48	93	12.48
	MUCOSA	93	12.48	93	12.48
	ADIPOSE	93	12.48	93	12.48
	EMPTY	93	12.48	93	12.48

Table 2. CRC dataset of histological images for Testing.

2.6. CNN:

A Convolutional Neural Network (CNN) model is highly effective for image-based tasks, such as colorectal tissue classification. Here, I'll outline the steps to develop and implement a CNN model for this purpose, including the architecture, data preparation, training, and evaluation.

Convolutional Neural Networks (CNNs) are a class of deep neural networks specifically designed for processing structured grid data, such as images. CNNs are particularly effective for image classification, object detection, and other computer vision tasks due to their ability to automatically learn and extract hierarchical features from input data. Here's a detailed explanation of CNNs and their relevance to the project "Advanced Automation for Colorectal Tissue Classification in Histopathology":

1. Architecture of CNNs

CNNs are composed of several key layers, each performing specific functions to process and learn from the input images:

1. **Input Layer**:

- The input layer holds the raw pixel values of the image, typically in a 3D matrix form (height, width, channels). For color images, the channels are usually RGB (Red, Green, Blue).

2. **Convolutional Layers**:
   - **Convolution Operation**: This layer applies a set of learnable filters (kernels) to the input image to produce feature maps. Each filter slides (convolves) across the image and performs a dot product between the filter and the receptive field it covers.
   - **ReLU Activation**: After convolution, the ReLU (Rectified Linear Unit) activation function is applied to introduce non-linearity into the model, which helps in learning complex patterns.
3. **Pooling Layers**:
   - **Max Pooling**: This layer reduces the spatial dimensions of the feature maps by selecting the maximum value in each patch of the feature map. This down-sampling reduces computational complexity and helps in extracting dominant features.
   - **Average Pooling**: Alternatively, average pooling can be used, which computes the average of all values in a patch. Max pooling is more common in practice.
4. **Fully Connected (Dense) Layers**:
   - After several convolutional and pooling layers, the high-level reasoning in the neural network is done via fully connected layers. These layers flatten the input and connect every neuron to every neuron in the next layer.
   - **Output Layer**: The final layer uses a softmax activation function for classification tasks, providing the probability distribution over the class labels.
5. **Dropout Layers**:
   - To prevent overfitting, dropout layers randomly set a fraction of input units to zero during training. This encourages the network to develop redundant representations and improves generalization.

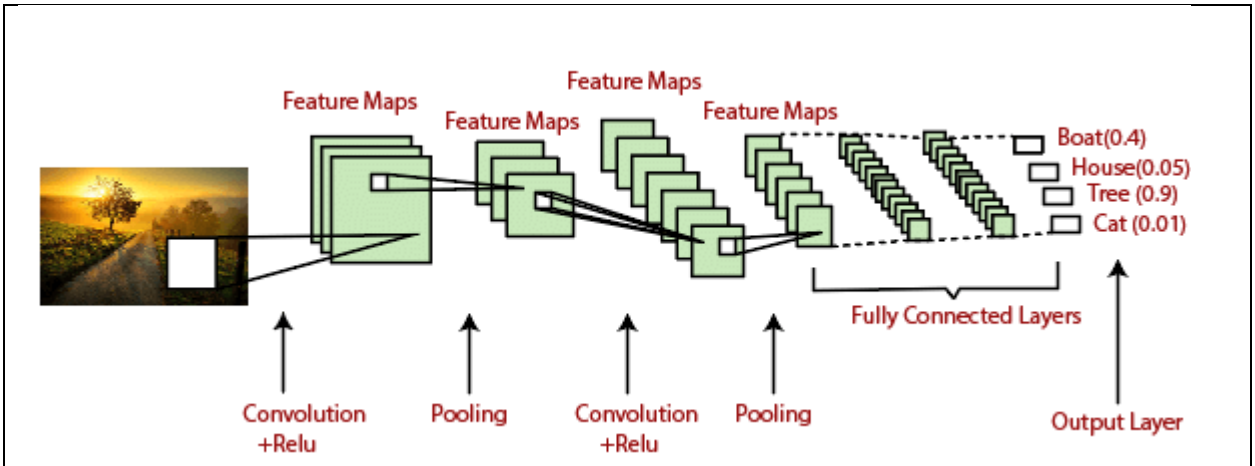

Figure 3. Architecture of CNN

2. Training CNN:

- **Data Augmentation**:
  - To improve the model's robustness and generalization, data augmentation techniques such as random rotations, flipping, cropping, and color jittering are applied to the training images.
- **Loss Function**:
  - For classification tasks, the categorical cross-entropy loss function is commonly used. It measures the discrepancy between the predicted probabilities and the true class labels.
- **Optimization**:
  - CNNs are trained using gradient-based optimization algorithms, such as Stochastic Gradient Descent (SGD) or more advanced variants like Adam, which adjust the weights of the network to minimize the loss function.
- **Backpropagation**:
  - During training, backpropagation is used to compute the gradients of the loss function with respect to each weight by the chain rule, updating the weights to reduce the loss.

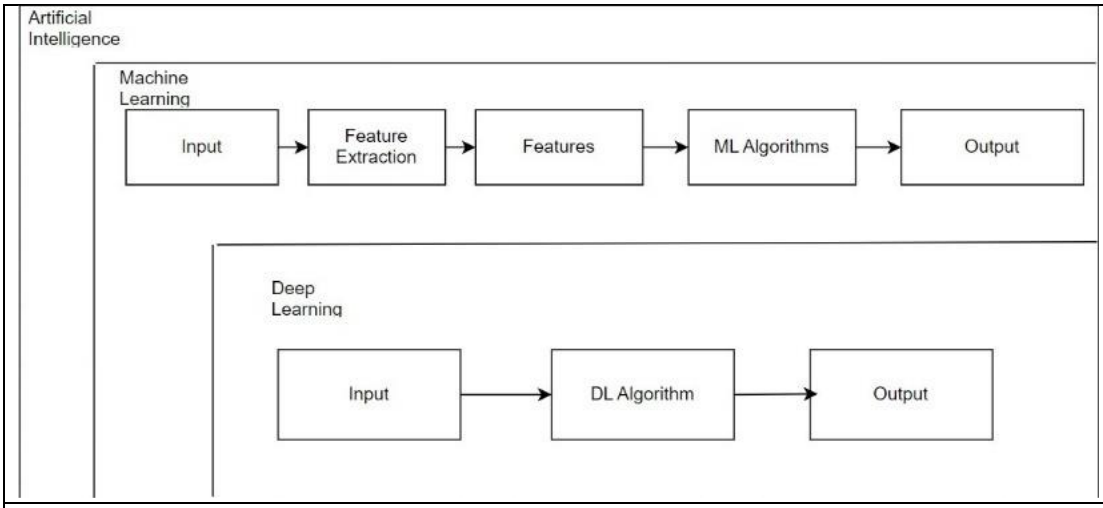

Figure 4. The Trends of Machine Learning

3. Applications in Histopathology

In the context of histopathology, CNNs can be applied to automate the classification of colorectal tissue images by learning to recognize patterns and features associated with different tissue types. Here's how CNNs are utilized in the project:

1. **Data Preparation**:
   - Histopathological images of colorectal tissues are collected and annotated by expert pathologists, labeling the regions as normal, adenomatous, or cancerous.
   - Images are preprocessed to standardize dimensions and enhance quality, making them suitable for input into CNNs.
2. **Feature Extraction**:

- CNNs automatically extract hierarchical features from the images, such as edges, textures, and shapes, which are crucial for distinguishing between different tissue types.

3. **Model Training**:
    - The annotated images are used to train the CNN. The model learns to map the input images to the corresponding tissue labels by minimizing the classification error.
    - Techniques such as data augmentation and dropout are employed to improve the model's robustness and prevent overfitting.
4. **Model Evaluation**:
    - The performance of the trained CNN is evaluated using metrics like accuracy, precision, recall, F1-score, and AUC-ROC on a validation dataset.
5. **Deployment**:
    - The trained CNN is integrated into a software tool or digital pathology workflow. It can process new histopathological images, providing automated and accurate tissue classification.
    - Visualization tools like heatmaps are used to highlight regions of interest and provide pathologists with interpretable results.

4. Advantages of CNNs in Histopathology

- **High Accuracy**: CNNs can achieve high accuracy in image classification tasks due to their ability to learn complex patterns and features.
- **Automation**: They enable the automation of tissue classification, reducing the workload of pathologists and minimizing human error.
- **Scalability**: CNNs can handle large-scale datasets and process images in real-time, making them suitable for large-scale screenings.

By leveraging CNNs, the project aims to develop an advanced automated system for colorectal tissue classification that enhances diagnostic accuracy, improves efficiency, and facilitates large-scale histopathological analysis.

2.7. CNN MODELS:

Several well-known Convolutional Neural Network (CNN) architectures have been developed and extensively used for various image processing tasks. Each model has its unique features and innovations, contributing to advancements in computer vision. Here's an overview of some prominent CNN models:

1. LeNet-5

- **Developed by**: Yann LeCun et al.
- **Year**: 1998
- **Application**: Handwritten digit recognition (MNIST dataset).
- **Architecture**:
    - Input: 32x32 grayscale images.
    - Layers: 2 convolutional layers, 2 subsampling (pooling) layers, 3 fully connected layers.
    - Activation: Tanh.
- **Key Features**: One of the first successful applications of CNNs, demonstrating the effectiveness of convolutional layers for pattern recognition.

2. AlexNet

- **Developed by**: Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton.
- **Year**: 2012
- **Application**: Image classification (ImageNet dataset).
- **Architecture**:
  - Input: 224x224 RGB images.
  - Layers: 5 convolutional layers, max pooling layers, 3 fully connected layers, dropout.
  - Activation: ReLU.
- **Key Features**: Introduced ReLU activation, dropout for regularization, and GPU acceleration, leading to a significant performance boost and winning the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012.

3. VGGNet

- **Developed by**: K. Simonyan and A. Zisserman (Oxford University).
- **Year**: 2014
- **Application**: Image classification.
- **Architecture**:
  - Input: 224x224 RGB images.
  - Layers: Very deep network with 16-19 layers, primarily using 3x3 convolutional filters, followed by max pooling and 3 fully connected layers.
  - Activation: ReLU.
- **Key Features**: Demonstrated the importance of depth in CNNs with simple and consistent architecture (3x3 convolutions). Variants: VGG16, VGG19.

4. GoogLeNet (Inception v1)

- **Developed by**: Google.
- **Year**: 2014
- **Application**: Image classification.
- **Architecture**:
  - Input: 224x224 RGB images.
  - Layers: 22 layers deep with inception modules (parallel convolutions with different filter sizes).
  - Activation: ReLU.
- **Key Features**: Introduced inception modules to efficiently capture multi-scale features, reducing computational cost by using 1x1 convolutions.

5. ResNet (Residual Networks)

- **Developed by**: Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (Microsoft Research).
- **Year**: 2015
- **Application**: Image classification, object detection.
- **Architecture**:
  - Input: 224x224 RGB images.
  - Layers: Deep networks (up to 152 layers) with residual blocks (skip connections).
  - Activation: ReLU.

- **Key Features**: Addressed the vanishing gradient problem, enabling very deep networks. Variants: ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152.

6. DenseNet (Densely Connected Networks)

- **Developed by**: Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Weinberger.
- **Year**: 2016
- **Application**: Image classification.
- **Architecture**:
  - Input: 224x224 RGB images.
  - Layers: Dense blocks with each layer receiving input from all previous layers.
  - Activation: ReLU.
- **Key Features**: Improved parameter efficiency, stronger gradient flow, reduced number of parameters compared to traditional CNNs of similar performance. Variants: DenseNet-121, DenseNet-169, DenseNet-201, DenseNet-264.

7. Inception-v3

- **Developed by**: Google.
- **Year**: 2015
- **Application**: Image classification.
- **Architecture**:
  - Input: 299x299 RGB images.
  - Layers: Enhanced inception modules, factorized convolutions (e.g., 7x7 into 1x7 and 7x1), auxiliary classifiers.
  - Activation: ReLU.
- **Key Features**: Improved efficiency and accuracy over Inception v1, factorization techniques to reduce computational cost.

8. Xception (Extreme Inception)

- **Developed by**: François Chollet.
- **Year**: 2016
- **Application**: Image classification.
- **Architecture**:
  - Input: 299x299 RGB images.
  - Layers: Depthwise separable convolutions in place of inception modules.
  - Activation: ReLU.
- **Key Features**: Enhanced performance by using depthwise separable convolutions, making the model more efficient and faster.

9. MobileNet

- **Developed by**: Google.
- **Year**: 2017
- **Application**: Mobile and embedded vision applications.
- **Architecture**:
  - Input: Variable size (typically 224x224 RGB images).
  - Layers: Depthwise separable convolutions.
  - Activation: ReLU.

- **Key Features**: Designed for mobile and embedded vision applications with reduced computational cost and model size. Variants: MobileNetV1, MobileNetV2, MobileNetV3.

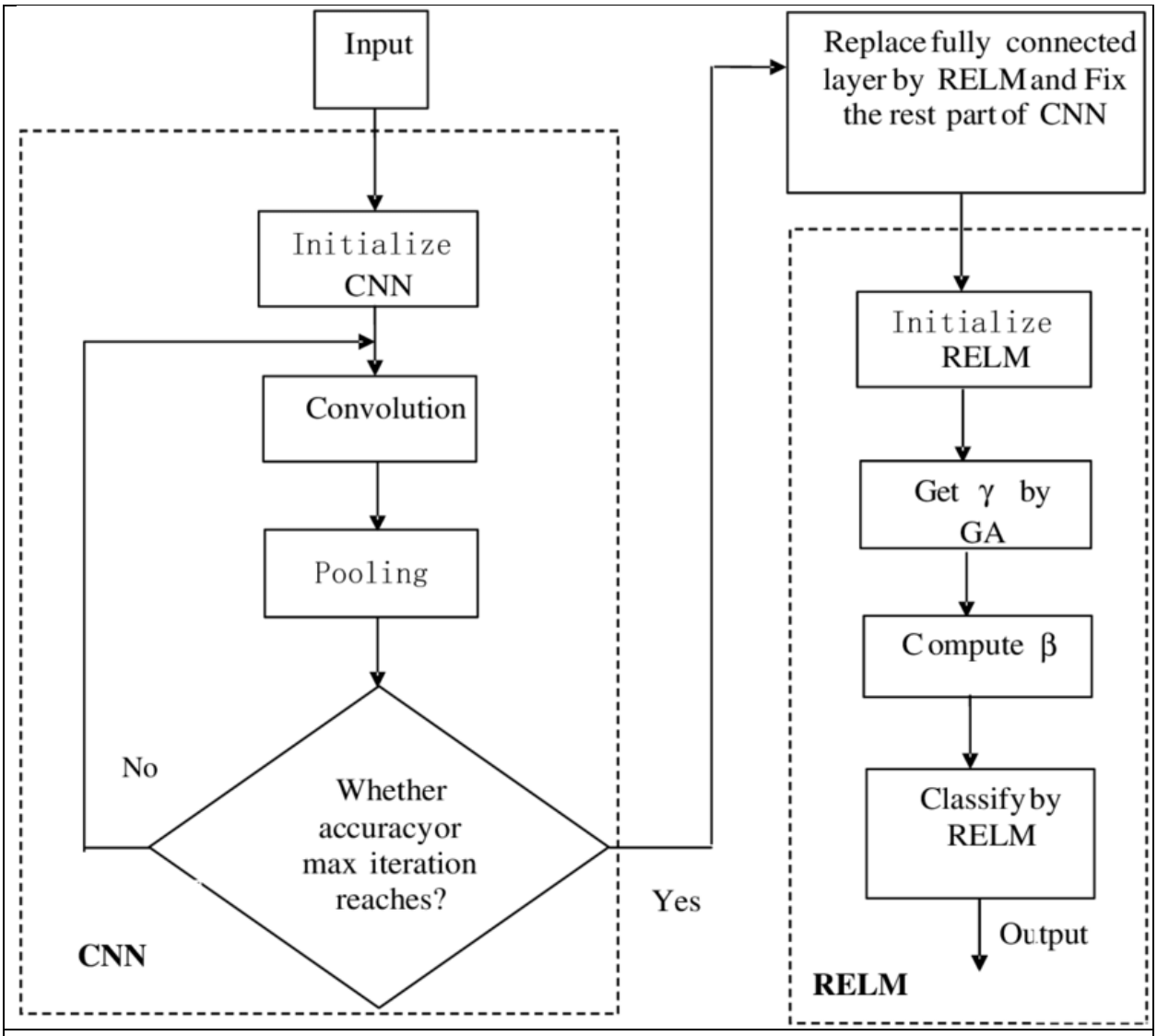

Figure 5. Working of a CNN model

2.8. Application to Histopathology

For the specific application of histopathology in colorectal tissue classification, these CNN models can be tailored and applied as follows:

1. **Data Preprocessing**:
   - **Normalization**: Ensures that pixel values are scaled appropriately.
   - **Augmentation**: Techniques like rotation, flipping, zooming, and color adjustments to increase the variability and robustness of the training dataset.

2. **Transfer Learning**:

- **Pre-trained Models**: Using pre-trained models on large datasets like ImageNet, and fine-tuning them on histopathological image datasets. This leverages pre-learned features and reduces the need for extensive computational resources.

3. **Feature Extraction**:
   - **Layer Selection**: Selecting appropriate layers from pre-trained models to extract relevant features from histopathological images.
   -
4. **Training**:
   - **Supervised Learning**: Using labeled datasets where pathologists have annotated tissue types (normal, adenomatous, cancerous).
   - **Optimization Techniques**: Advanced optimizers like Adam or SGD with momentum, learning rate schedules, and regularization techniques to improve training efficiency and performance.

5. **Evaluation**:
   - **Performance Metrics**: Evaluating models using accuracy, precision, recall, F1-score, and AUC-ROC to ensure reliable classification.

6. **Deployment**:
   - **Integration**: Incorporating trained models into digital pathology workflows for real-time tissue classification.
   - **Visualization Tools**: Heatmaps and overlay images to help pathologists interpret model predictions and validate results.

By leveraging these sophisticated CNN architectures and adapting them to the domain of histopathology, the project aims to significantly enhance the accuracy, efficiency, and reliability of colorectal tissue classification, facilitating better diagnostic processes and patient outcomes.

2.9. ADVANTAGES OF CNN:

Convolutional Neural Networks (CNNs) offer several advantages over traditional machine learning algorithms, especially in tasks involving image processing and computer vision. Here's a detailed exploration of the advantages of CNNs over other algorithms:

1. Feature Learning and Hierarchical Representation

- Advantage: CNNs automatically learn hierarchical representations of features directly from the raw input data.
- Details: Traditional machine learning algorithms require handcrafted feature extraction, where domain knowledge is used to define relevant features. In contrast, CNNs learn features through convolutional and pooling layers, capturing hierarchical patterns at different levels of abstraction. This ability to learn features reduces the need for manual feature engineering and enables the model to adapt to a wide range of datasets and tasks.

2. Spatial Invariance

- Advantage: CNNs are capable of capturing spatial hierarchies and exhibit translational invariance.
- Details: CNNs leverage shared weights and local connectivity, allowing them to detect features irrespective of their location in the input image. This property makes CNNs robust to variations in object position, scale, and orientation. Traditional algorithms, such as handcrafted feature detectors or template matching, lack this spatial invariance and require precise alignment of features.

3. Parameter Sharing and Sparse Connectivity

- Advantage: CNNs employ parameter sharing and sparse connectivity, leading to more efficient models.
- Details: In CNNs, each filter (or kernel) is applied across the entire input image, sharing the same weights. This parameter sharing reduces the number of learnable parameters, making CNNs more memory-efficient and computationally faster compared to fully connected networks. Traditional algorithms often require a large number of parameters, leading to increased computational complexity and overfitting.

4. Scale-Invariance and Local Receptive Fields

- Advantage: CNNs are naturally scale-invariant and operate with local receptive fields.
- Details: CNNs process input data using small filters that slide over the input image, capturing local patterns. This local connectivity enables CNNs to focus on fine-grained details while maintaining scale-invariance. Traditional algorithms may struggle with scale variations and may require preprocessing steps such as image pyramid construction or multi-scale feature extraction.

5. End-to-End Learning

- Advantage: CNNs enable end-to-end learning, where the entire model is trained jointly from raw input to output predictions.
- Details: CNNs can be trained using gradient-based optimization algorithms to minimize a specific loss function directly on raw input data. This end-to-end learning approach allows CNNs to automatically optimize feature representations and classification decisions without the need for separate feature extraction and classification stages. Traditional algorithms often rely on handcrafted pipelines with separate feature extraction and classification components, which may lead to suboptimal performance.

6. Scalability and Generalization

- Advantage: CNNs are highly scalable and generalize well to unseen data.
- Details: CNNs can be trained on large-scale datasets with millions of parameters, leveraging the power of parallel computation on modern hardware accelerators such as GPUs and TPUs. This scalability enables CNNs to capture complex patterns and generalize well to diverse datasets. Moreover, techniques like transfer learning allow pretrained CNN models to be fine-tuned on specific tasks, further enhancing their generalization capabilities. Traditional algorithms may struggle to scale effectively to large datasets and may require manual tuning for optimal performance.

7. State-of-the-Art Performance

- Advantage: CNNs consistently achieve state-of-the-art performance on various computer vision tasks.
- Details: CNN architectures have evolved significantly over the years, with advancements in model design, training algorithms, and computational resources. As a result, CNNs have surpassed traditional algorithms in accuracy, robustness, and efficiency across a wide range of tasks, including image classification, object detection, semantic segmentation, and image generation. The ability of CNNs to learn complex representations from raw data has contributed to their widespread adoption in both research and industry.

In summary, Convolutional Neural Networks offer numerous advantages over traditional machine learning algorithms, particularly in tasks involving image processing and computer vision. Their ability to learn hierarchical features, capture spatial invariance, and perform end-to-end learning makes them well-suited for a wide range of applications, from image classification to object detection and beyond.

2.10. LIMITATIONS OF CNN:

Convolutional Neural Networks (CNNs) are powerful models for image processing tasks, but they also have limitations that can affect their performance in certain scenarios. Here are some key limitations of CNNs:

1. Limited Spatial and Temporal Context

- **Limitation**: CNNs have a fixed receptive field size, which limits their ability to capture global spatial or temporal context in large images or sequences.
- **Impact**: In tasks requiring understanding of global context, such as scene understanding in panoramic images or video analysis, CNNs may struggle to capture long-range dependencies effectively.

2. Robustness to Variations

- **Limitation**: CNNs can be sensitive to variations in input data, such as changes in lighting conditions, viewpoint, or occlusions.
- **Impact**: Variations in input data may lead to misclassification or degraded performance. CNNs trained on specific datasets may not generalize well to unseen variations present in real-world scenarios.

3. Data Efficiency

- **Limitation**: CNNs typically require large amounts of labeled data for training to achieve high performance.
- **Impact**: Acquiring and labeling large datasets can be expensive and time-consuming. Limited availability of labeled data may hinder the performance of CNNs, especially in niche or specialized domains.

4. Interpretability

- **Limitation**: CNNs are often considered "black box" models, making it challenging to interpret their decisions.

- **Impact**: Understanding how CNNs arrive at their predictions can be difficult, especially in critical applications such as healthcare or legal domains where interpretability is essential. Lack of interpretability may limit trust and acceptance of CNN-based systems.

5. Overfitting

- **Limitation**: CNNs with a large number of parameters are prone to overfitting, especially when trained on limited or noisy data.
- **Impact**: Overfitted models may perform well on training data but generalize poorly to new, unseen data. Techniques such as dropout and regularization can mitigate overfitting, but careful regularization is necessary to avoid sacrificing performance.

6. Resource Intensive Training

- **Limitation**: Training deep CNN models requires significant computational resources and time.
- **Impact**: Training CNNs can be computationally intensive, requiring access to powerful hardware such as GPUs or TPUs. High energy costs associated with training may also be a concern.

7. Limited Applicability

- **Limitation**: CNNs may not be suitable for all types of data or tasks.
- **Impact**: While CNNs excel at tasks involving grid-like data such as images, they may not perform optimally for tasks involving sequential or structured data. In such cases, other architectures like recurrent neural networks (RNNs) or transformers may be more appropriate.

8. Adversarial Vulnerabilities

- **Limitation**: CNNs are susceptible to adversarial attacks, where small, imperceptible perturbations to input data can lead to incorrect predictions.
- **Impact**: Adversarial attacks pose security risks, particularly in safety-critical applications such as autonomous vehicles or medical diagnosis. Robustness against adversarial attacks remains an ongoing research challenge.

9. Computational Complexity

- **Limitation**: Deep CNN architectures can be computationally expensive to deploy in resource-constrained environments.
- **Impact**: Deploying CNN-based models on edge devices or embedded systems with limited computational resources may be challenging due to high computational complexity and memory requirements.

In summary, while CNNs offer state-of-the-art performance in many image processing tasks, they also have limitations related to their spatial and temporal context, robustness to variations, data efficiency, interpretability, overfitting, computational requirements, applicability to diverse data types, vulnerabilities to adversarial attacks, and computational complexity. Understanding these limitations is crucial for effectively deploying CNN-based systems and developing strategies to mitigate their impact.

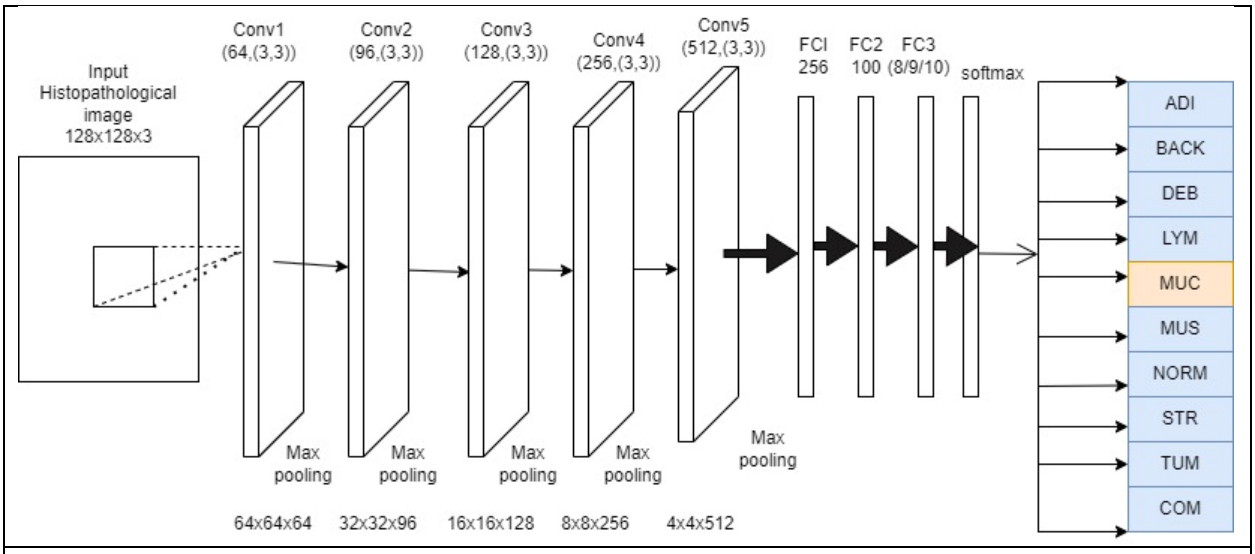

Figure 6. Basic CNN model

CHAPTER 3

SOFTWARE / PROJECT DESIGN

Designing software and project architecture for advanced automation in colorectal tissue histopathology demands meticulous attention to various critical aspects. Firstly, the selection of software solutions must align closely with the specific requirements of histopathological workflows, encompassing tasks such as slide scanning, image analysis, and data management. Whether opting for custom-built platforms or off-the-shelf solutions, adopting a modular design approach facilitates scalability and interoperability across systems. Ensuring an intuitive user interface design is paramount to accommodate the diverse range of users, including laboratory technicians, pathologists, and administrators. Moreover, the development and integration of sophisticated machine learning algorithms, particularly those leveraging deep learning techniques, are essential for accurate and reliable automated detection and quantification of histopathological features. Seamless integration with existing laboratory information systems (LIS) is crucial for efficient data exchange and reporting, with consideration given to cloud-based solutions for scalability and cost-effectiveness. Robust data management protocols must be established to safeguard patient privacy and comply with regulatory standards, reinforced by comprehensive testing and validation procedures to ensure the accuracy and reliability of automated systems. Embracing an agile development methodology facilitates iterative refinement and continuous stakeholder engagement, ensuring the software evolves to meet the evolving needs of colorectal tissue histopathology.

3.1. METHEDOLOGY

It is very important to make an objective evaluation of colorectal cancer histological images. Current approaches are generally based on the use of different combinations of textual features and classifiers to assess the classification performance, or transfer learning to classify different organizational types. However, since histological images contain multiple tissue types and characteristics, classification is still challenging. In this study, we proposed the best classification methodology based on the selected optimizer and modified the parameters of CNN methods. Then, we used deep learning technology to distinguish between healthy and diseased large intestine tissues. Firstly, we trained a neural network and compared the network architecture optimizers. Secondly, we modified the parameters of the network layer to optimize the superior architecture. Finally, we compared our well-trained deep learning methods on two different histological image open datasets, which comprised 5000 H&E images of colorectal cancer. The other dataset was composed of nine organizational categories of 100,000 images with an external validation of 7180 images. The results showed that the accuracy of the recognition of histopathological images was significantly better than that of existing methods. Therefore, this method is expected to have great potential to assist physicians to make clinical diagnoses and reduce the number of disparate assessments based on the use of artificial intelligence to classify colorectal cancer tissue.

1. Data Collection and Preprocessing:

- Dataset Acquisition: Gather a sizable dataset of images with corresponding classes and labels. We use two different data set one which comprised 5000 H&E images of colorectal cancer. The other dataset was composed of nine organizational categories of 100,000 images with an external validation of 7180 images
- NCT-CRC-HE-100K 100,000 images and Kather-texture-2016-image 5000 images.
- Data Cleaning: Preprocess the images by resizing and normalizing them to a consistent format. We will need to adapt it to your dataset and experiment with hyperparameters, such as the number of convolutional layers, filter sizes, and dense layers, to optimize the performance for your specific problem.

2. Feature Extraction for Images:

- Use Pre-trained CNN Models: Utilize pre-trained Convolutional Neural Network (CNN) models such as VGG, ResNet, or MobileNet to extract high-level features from the images. These models are trained on massive image datasets and can capture intricate visual information.
- Fine-tuning (Optional): Fine-tune the CNN model on your specific captioning task, allowing it to adapt to the specific nuances of your dataset.

3. Model Architecture:

- **VGG19:** VGG19 is a deep convolutional neural network (CNN) architecture that is part of the Visual Geometry Group (VGG) models developed by the University of Oxford. It is composed of 19 layers, including 16 convolutional layers, 3 fully connected layers, and 5 max-pooling layers, ending with a softmax layer for classification. Known for its simplicity and uniform structure, VGG19 employs small receptive fields (3x3 filters) across its convolutional layers, which enhances its capability to learn intricate features from images. This architecture has been influential in various computer vision tasks, achieving notable performance in image classification, object detection, and feature extraction.
- **ResNet-50:** ResNet-50 is a deep convolutional neural network architecture that is part of the Residual Networks (ResNet) family, introduced by Microsoft Research. It consists of 50 layers, including convolutional, batch normalization, ReLU activation, and fully connected layers, structured with a series of residual blocks. The key innovation of ResNet-50 is the use of skip connections or shortcuts that bypass one or more layers, which helps mitigate the vanishing gradient problem, enabling the training of much deeper networks. This architecture has demonstrated exceptional performance in various computer vision tasks, such as image classification, object detection, and segmentation, and it won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2015.
- **MobileNet:** MobileNet is a class of efficient convolutional neural network architectures designed specifically for mobile and embedded vision applications. Developed by Google, MobileNet models utilize depthwise separable convolutions, which split the standard convolution into a depthwise convolution and a pointwise convolution. This significantly reduces the model's size and computational cost while maintaining high accuracy. MobileNet's architecture is parameterized by two

hyperparameters—width multiplier and resolution multiplier—that allow for a trade-off between latency, computational cost, and accuracy. As a result, MobileNet is highly effective for deployment on devices with limited resources, such as smartphones and IoT devices, and is widely used in applications requiring real-time image classification, object detection, and other vision tasks.

4. Training:

- Loss Function: Train the model using a suitable loss function, such as cross-entropy, to minimize the difference between the generated captions and the ground truth captions.
- Teacher Forcing: Utilize teacher forcing, a technique where the true previous word is used as input to predict the next word during
- Gradient Descent: Fine-tune the model's parameters using gradient descent or other optimization techniques.
- Optimizer: An optimizer in deep learning is a function or algorithm that adjusts the attributes of a neural network. Optimizers help to:
  - Reduce overall loss
  - Improve accuracy
  - Get results faster
  - Optimizers change attributes like weights and learning rates. They are used to solve optimization problems by minimizing the function.

5. Evaluation:

- Quality Metrics: At this stage, we used 4 different CNN models for the training dataset: DenseNet-121, VGG19, Inception-ResNet-v2, and Resnet50 . The architecture included different convolutional layers, rectified linear units' layer (ReLU layer), max-pooling layer, and fully connected layers. This improve the overall Quality of the evaluation.
- Human Evaluation: Consider conducting user studies to gather subjective feedback and assess the captions' human-like quality.

3.2. ALGORITHMS USED

1. VGG19 Model:

VGG19, a convolutional neural network (CNN) developed by the Visual Geometry Group (VGG) at Oxford, is particularly known for its deep architecture with 19 layers. VGG19 is a deep neural network model that consists of 16 convolutional layers, 3 fully connected layers, and a SoftMax layer. It was designed to work on image classification tasks and has achieved excellent performance on the ImageNet dataset.

VGG19 is a type of Convolutional Neural Network (CNN) that is commonly used in the field of computer vision. In the context of advanced automation for the classification of colorectal tissue in histopathology, VGG19 can be used as a feature extractor for whole-slide histopathological images. One study developed an explainable classifier for improving the accountability in decision-making for colorectal cancer diagnosis from histopathological images. The classifier used a fine-tuned VGG19 model to extract features from the images, which were then classified into eight tissue types. The system was designed to be used as a support system for medical experts, and 14 pathologists considered it to be useful and reliable.

The key features of VGG19 include:

- **Deep architecture**: The network has many layers, which enables it to learn complex features.
- **Small receptive fields**: The network uses small 3x3 convolutional filters, which are the smallest size to capture directional information (left, right, up, down). This small receptive field enables the network to build deeper and more complex feature representations layer by layer, while keeping the network parameters manageable. Histopathology involves the microscopic examination of tissue to study the manifestations of disease. In the case of colorectal tissue, accurate classification is crucial for diagnosing various conditions, such as cancer. VGG19 is suitable for this purpose.

Following shows how VGG19 can be utilized for this purpose:

- **Data Preparation:**
  - i. **Dataset Collection**: Obtain a large dataset of histopathological images of colorectal tissue. These images need to be annotated by experts to serve as ground truth.
  - ii. **Preprocessing**: Normalize the images, resize them to a fixed size (e.g., 224x224 pixels to match VGG19's input requirements), and augment the data to increase variability (rotations, flips, color adjustments).
- **Model Training :**
  - i. **Transfer Learning**: Given the limited size of medical image datasets compared to general image datasets, transfer learning is often employed. Start with a pre-trained VGG19 model on ImageNet and fine-t.
  - ii. **Feature Extraction**: Freeze the convolutional layers and train only the newly added classification layers initially. This step allows the model to adapt to the new dataset without losing the general features learned from ImageNet.
  - iii. **Fine-Tuning**: Unfreeze some of the later convolutional layers and retrain the model with a lower learning rate. This helps in fine-tuning the model to better capture the specific features of colorectal histopathological images.

- **Evaluation**:

Evaluate the performance of the model using metrics like accuracy, precision, recall, and F1-score. It’s crucial to use a separate validation dataset to assess the model’s generalization ability.Perform

cross-validation to ensure that the model's performance is consistent across different subsets of the dataset.

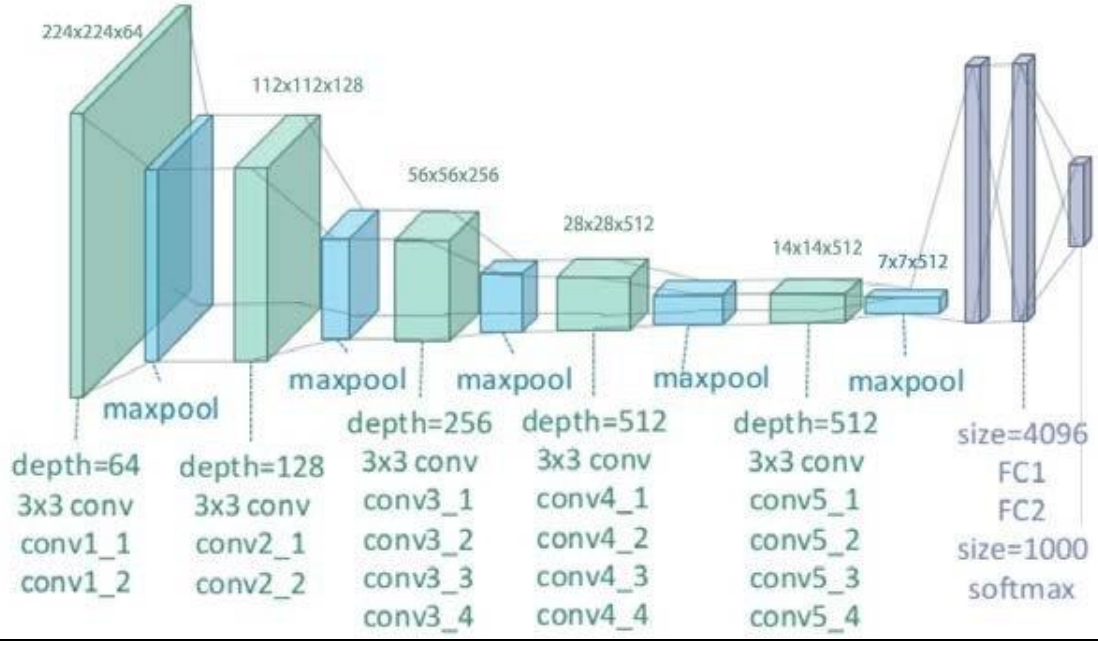

Figure 7. VGG19 model

I. Working of VGG19

To explain the working of VGG19 in detail in context of colorectal tissue classification in histopathology, let's break down the process step-by-step:

- **Data Collection and Preprocessing**
  - Data Collection: Collect a large dataset of histopathological images of colorectal tissues, which includes different categories such as normal, benign, and malignant tissues. We have used NCT-CRC-HE-100K 100,000 images and Kather-texture-2016-image 5000 images.
  - Data Preprocessing: Resizing: Ensure all images are of the same size typically 224x224 pixels as VGG19 requires a fixed input size.
  - Normalization: Normalize the pixel values to be within a specific range, usually [0, 1] or [-1, 1], to facilitate faster convergence during training.
  - Augmentation: Apply data augmentation techniques like rotation, flipping, scaling, and cropping to increase the variability of the training data, reducing the risk of overfitting.

- **Transfer Learning with VGG19**
  - Loading Pre-trained Model: Load the pre-trained VGG19 model weights that have been trained on the ImageNet dataset. This helps in leveraging the learned features from a broad dataset.

- Modifying the Model: Replace the top fully connected (FC) layers of VGG19 with new layers that match the number of classes in your colorectal tissue dataset. For instance, if there are three classes (normal, benign, malignant), replace the final layer with a dense layer having three neurons with SoftMax activation.

- **Training the Model**
  - Feature Extraction :Initially, freeze the weights of all convolutional layers to use VGG19 as a fixed feature extractor. Train only the newly added FC layers.
  - Freezing Layers: for layer in vgg19.
    layers: layer.trainable = False
  - Compiling the Model: Compile the model with a suitable optimizer (e.g., Adam) and loss function (e.g., categorical cross-entropy if using softmax for multi-class classification).
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
  - Training the Model:Train the model on your dataset. Initially, train only the new top layers.
    history = model.fit(train_data, train_labels, validation_data=(val_data, val_labels), epochs=10, batch_size=64)

  - Fine-Tuning: Unfreeze some of the later convolutional layers to fine-tune the network, allowing these layers to adjust to the specific features of colorectal tissue images.
    for layer in vgg19.layers[-4:]:
    layer.trainable = True
    model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy', metrics=['accuracy'])
    history_fine = model.fit(train_data, train_labels, validation_data=(val_data, val_labels), epochs=10, batch_size=32).

- **Evaluating the Model**
  - Performance Metrics: Evaluate the model using accuracy, precision, recall, and F1-score on a separate test dataset to assess its performance.
    test_loss, test_acc = model.evaluate(test_data, test_labels)
  - Confusion Matrix: Use a confusion matrix to understand the classification performance across different classes.
    from sklearn.metrics import confusion_matrix
    predictions = model.predict(test_data)
    cm = confusion_matrix(test_labels.argmax(axis=1), predictions.argmax(axis=1))

- **Deployment and Inference**
  - Deployment: Once the model is trained and validated, it can be deployed in a clinical setting for real-time classification of colorectal tissue images.
  - Inference: For new histopathological images, preprocess the images similarly to the training data, pass them through the model, and interpret the output probabilities for classification.

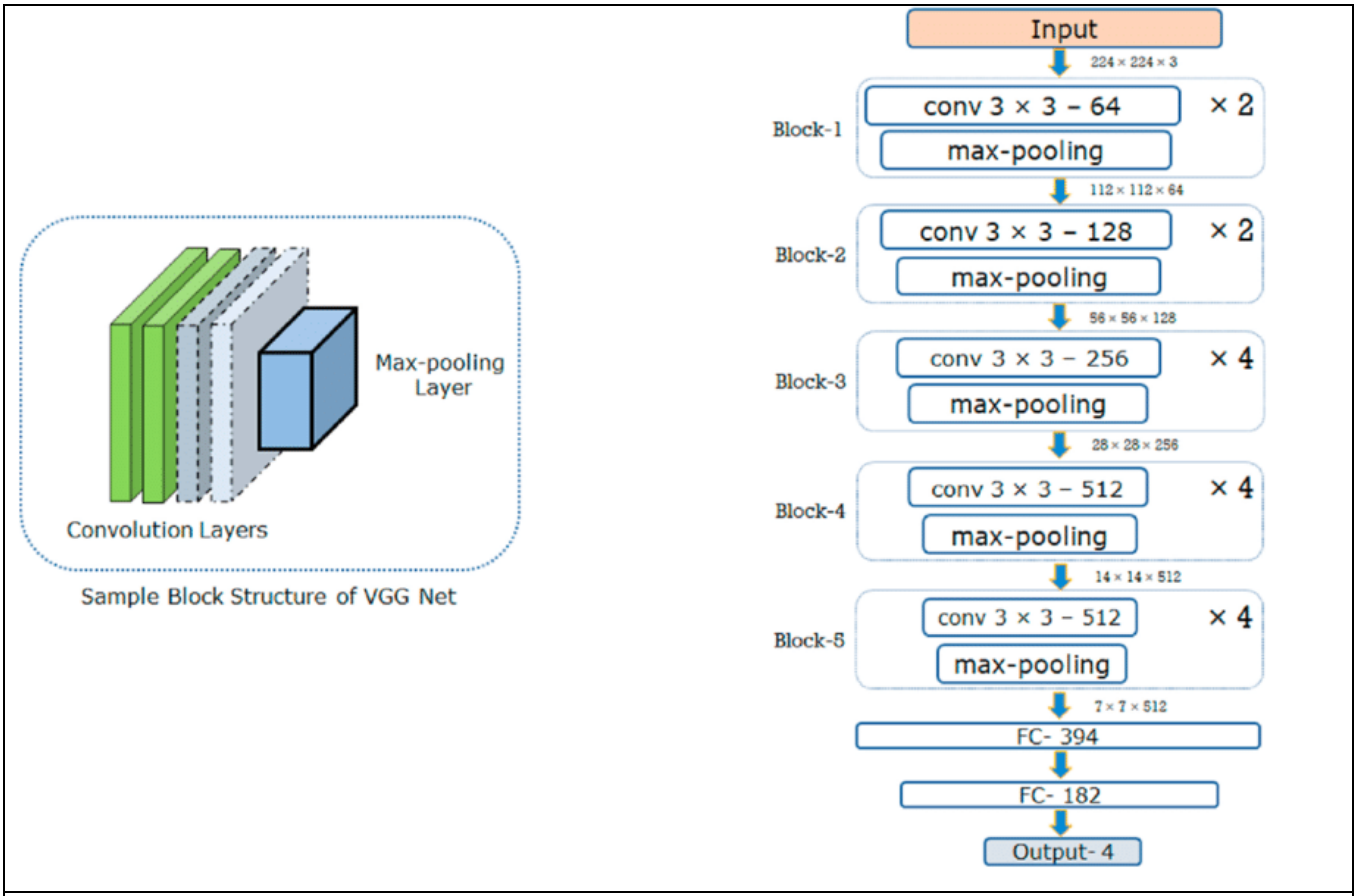

Figure 8. Working of a VGG19 model

II. Benefits of Using VGG19

VGG19 can offer several benefits in the context of colorectal tissue classification in histopathology.

- High Accuracy: VGG19's depth allows it to learn complex features, making it highly effective for distinguishing between subtle differences in histopathological images.
- Transfer Learning: Leveraging a pre-trained VGG19 model significantly reduces the training time and computational resources required.
- Versatility: The model can be adapted for various classification tasks with minimal changes.

Additionally, using a two-stage model to separate false-positive normal epithelium from tumor epithelium can further improve the accuracy of the classification system.Overall, VGG19 can contribute to a more efficient and accurate classification of colorectal tissues in histopathology.

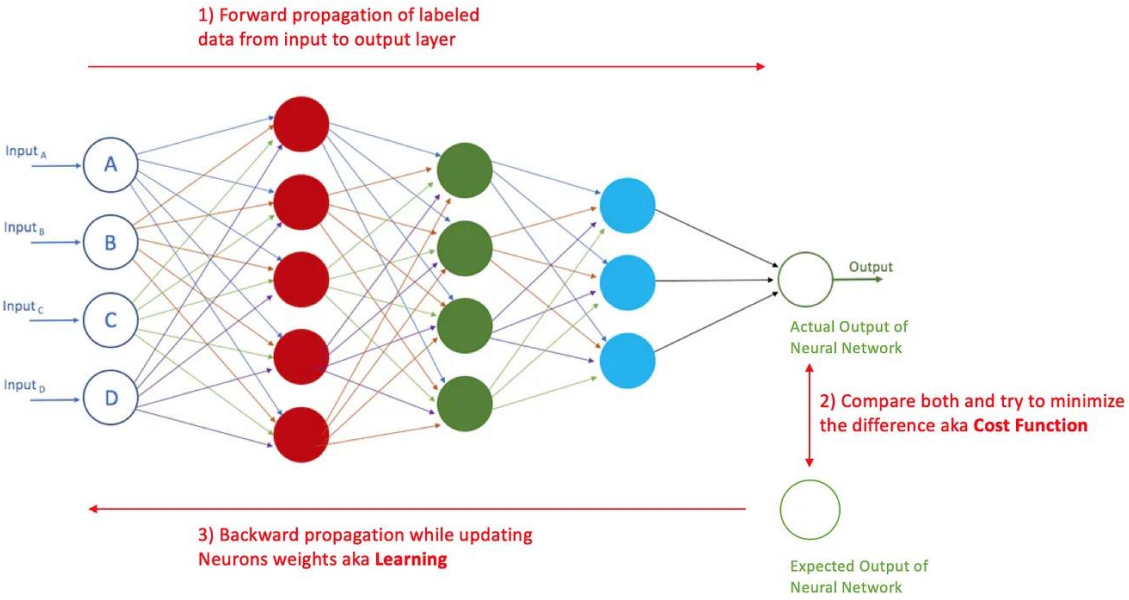

Figure 9. Image detection using VGG19

III. Challenges and their solutions:

Using VGG19 for colorectal tissue classification in histopathology presents several challenges. Below are the primary challenges and some potential solutions for each:

- **Overfitting**
  - Challenge: Due to its complexity, VGG19 is prone to overfitting, especially when the dataset is not sufficiently large or diverse.
  - Solutions:
    - Data Augmentation: Increase the size and variability of the dataset through augmentation techniques (rotation, flipping, scaling, etc.).
    - Regularization: Use techniques such as dropout, L2 regularization, and batch normalization to prevent overfitting.
    - Transfer Learning: Fine-tune a pre-trained VGG19 model on your specific dataset to leverage learned features from a larger dataset (e.g., ImageNet).

- **Large Model Size**
  - Challenge: VGG19 has a large memory footprint, making it difficult to deploy on devices with limited resources.
  - Solutions:

- Model Compression: Use techniques like model pruning, quantization, or knowledge distillation to reduce model size.

- **Sensitivity to Input Variability**
  - Challenge: Histopathology images can have significant variability in staining and preparation, affecting model performance.
  - Solutions:
    - Stain Normalization: Apply stain normalization techniques to reduce variability caused by different staining processes.
    - Robust Pre-processing: Implement robust pre-processing pipelines to standardize image inputs.
- **Data Imbalance**
  - Challenge: Histopathology datasets may be imbalanced, with some classes being underrepresented, leading to biased model performance.
  - Solutions:
    - Data Balancing: Use oversampling, undersampling, or synthetic data generation techniques (e.g., SMOTE) to balance the dataset.
    - Class Weighting: Assign higher weights to underrepresented classes during training to mitigate bias.
- **Training Time**
  - Challenge: Training VGG19 from scratch or fine-tuning it on a new dataset can be time-consuming.
  - Solutions:
    - Efficient Training: Utilize techniques like mixed-precision training and distributed training to speed up the process.
    - Pre-trained Models: Start with a pre-trained model and fine-tune it on your specific dataset to save time and computational resources.

IV. Applications

1. **Automated Tissue Classification:** Automatically classify histopathology images into categories such as normal tissue, adenomas, and carcinomas.
   - Benefit: Reduces the workload of pathologists by quickly and accurately identifying different types of colorectal tissues, allowing them to focus on more complex cases.

2. **Assisting Pathologists in Diagnosis:** Provide a second opinion or assist pathologists in making accurate diagnoses by highlighting areas of interest or potential anomalies in tissue samples.
   - Benefit: Increases diagnostic accuracy and consistency, helping to identify conditions that might be missed during manual examination.

3. **Digital Pathology Integration:** Integrate VGG19-based models into digital pathology platforms to streamline the workflow of histopathological analysis.

- Benefit: Enables seamless analysis and management of digital slides, facilitating quicker and more efficient diagnostic processes.

4. **Research and Training**
   - Application: Use the model to analyze large datasets for research purposes or to train new pathologists.
   - Benefit: Supports the advancement of medical research by providing tools to analyze tissue samples at scale and aids in educational purposes by providing consistent and accurate training data.

5. **Tumor Detection and Segmentation**
   - Application: Identify and segment tumor regions within histopathology slides.
   - Benefit: Enhances the precision of tumor detection and allows for better planning of treatment strategies by accurately delineating tumor boundaries.

6. **Quality Control**
   - Application: Implement automated quality control to ensure that tissue samples are prepared correctly and that image quality meets the required standards.
   - Benefit: Ensures consistency and reliability in histopathological analysis, reducing errors related to sample preparation and imaging.

7. **Predictive Analytics**
   - Application: Combine tissue classification with other patient data to predict disease progression and outcomes.
   - Benefit: Provides insights into patient prognosis and helps in the early detection of potential complications**.**

8. **Telepathology**
   - Application: Enable remote diagnosis by providing reliable automated analysis of histopathology images sent from different locations.
   - Benefit: Facilitates access to expert diagnostic services in remote or underserved areas, improving healthcare delivery and patient outcomes.

2. ResNet50:

The ResNet50 model is a convolutional neural network (CNN) architecture that has significantly advanced the field of computer vision since its introduction by Microsoft Research in 2015. It belongs to a family of models known as Residual Networks (ResNets), designed to address the challenge of training very deep neural networks by mitigating the vanishing gradient problem.

I. **Architecture:** ResNet50 is composed of 50 layers, as indicated by its name. The architecture is based on the concept of residual learning, which introduces skip

connections, or shortcuts, to jump over one or more layers. This architecture allows for the creation of extremely deep networks without encountering degradation in performance.
The building blocks of ResNet50 are residual blocks, which consist of multiple convolutional layers followed by identity mappings. These identity mappings provide a shortcut for the gradient during backpropagation, allowing for easier training of deep networks. Each residual block contains a set of convolutional layers, followed by batch normalization and rectified linear unit (ReLU) activations.

II. **Training:** Training a ResNet50 model typically involves initializing the network's weights using pre-trained weights from a large dataset (e.g., ImageNet) or random initialization. The model is then trained using gradient-based optimization algorithms such as stochastic gradient descent (SGD) or Adam.
During training, the model learns to minimize a loss function, such as categorical cross-entropy, by adjusting its weights based on the gradient of the loss function with respect to the network parameters. The training process involves forward propagation to compute predictions, backward propagation to compute gradients, and parameter updates to minimize the loss function.

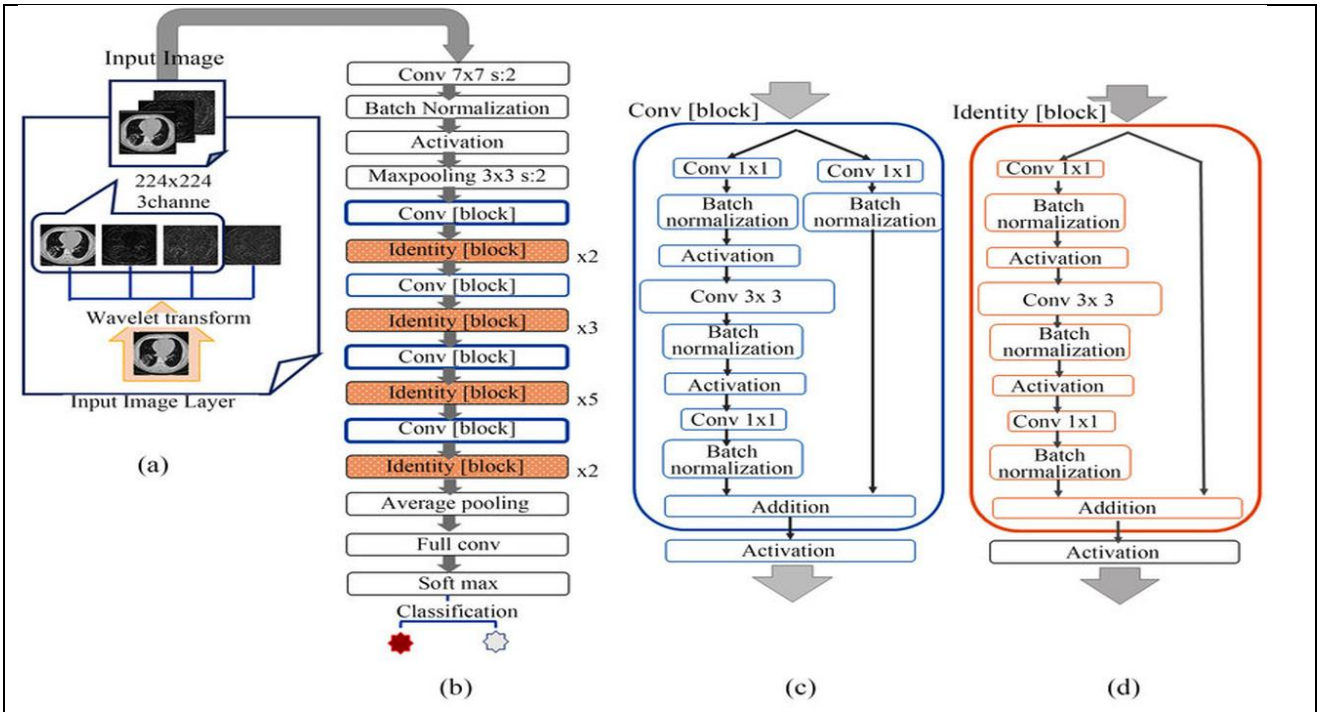

Figure 10. Working of ResNet-50

III. WORKING OF RESNET50:

This code is for a project involving training and evaluating a deep learning model, particularly using the ResNet50 architecture, for the task of classifying colorectal cancer histology images into different tissue types. Let's break down the code into its components and explain each part in detail:

1. **Importing Necessary Modules**: The code starts by importing various Python modules required for data preprocessing, model building, training, evaluation, and visualization. These include modules for data manipulation (e.g., pandas, numpy), image processing (e.g., OpenCV), deep learning (e.g., TensorFlow, Keras), and visualization (e.g., Matplotlib, Seaborn).

    - os: This module provides functions for interacting with the operating system.
    - pandas (pd): Pandas is a powerful library for data manipulation and analysis in Python.
    - numpy (np): NumPy is a fundamental package for scientific computing with Python
    - itertools.chain: This module provides functions for creating iterators for efficient looping.
    - cv2: OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision
    - split_folders: This module is used for splitting a dataset into train, validation, and test sets while maintaining the directory structure.
    - matplotlib.pyplot as plt: Matplotlib is a plotting library for Python.
    - matplotlib.image as mpimg: This submodule of Matplotlib provides functions for reading, displaying, and manipulating images.
    - %matplotlib inline: This is a magic command used in Jupyter notebooks to display Matplotlib plots inline within the notebook.
    - seaborn as sns: Seaborn is a Python visualization library based on Matplotlib.
    - sklearn.metrics: This submodule of scikit-learn (sklearn) provides various metrics for evaluating machine learning models.
    - tensorflow as tf: TensorFlow is an open-source machine learning framework developed by Google.
    - tensorflow.keras: Keras is a high-level neural networks API written in Python. TensorFlow's implementation of Keras provides a user-friendly interface for building and training deep learning models.
    - tensorflow.keras.layers: This submodule contains various layers (e.g., Dense, Conv2D, Dropout) that can be used to construct neural network architectures.
    - tensorflow.keras.callbacks: Callbacks are functions that can be applied at various stages of the training process, such as at the start or end of an epoch.
    - tensorflow.keras.models: This submodule provides tools for building and manipulating deep learning models, such as the Model class for defining models and load_model function for loading saved models.

- tensorflow.keras.optimizers: Optimizers are algorithms used to update the parameters (weights) of a neural network during training.
- tensorflow.keras.preprocessing.image: This submodule contains tools for preprocessing image data, such as ImageDataGenerator for data augmentation and image resizing.
- tensorflow.keras.applications: This submodule provides pre-trained deep learning models (e.g., VGG, ResNet, MobileNet) that can be used for transfer learning and feature extraction.
- tensorflow.keras.preprocessing: This submodule provides utilities for preprocessing input data, such as text preprocessing, sequence padding, and one-hot encoding.

These imported modules are essential for performing various tasks in the project, including data preprocessing, model building, training, evaluation, and visualization. They provide the necessary tools and functions required to implement the machine learning workflow effectively.

2. **Setting Up Directories**: Setting up directories is an essential step in organizing your project's data and outputs. In this project, the directories are set up to store the dataset and the results of the model training and evaluation. Let's go through the directory setup process in detail:
    - **Base Directory**: The **base_dir** variable is initialized with the path to the base directory where the dataset is located.
      In this project, the base directory is **/input/colorectal_cancer**.

    - **Input Directory**: The **input_** variable is created by joining the **base_dir** with the directory name containing the dataset, which is **" CRC-VAL-HE-7K"**.
      This directory contains the raw dataset with images and their corresponding labels.

    - **Output Directory**: After importing the **split_folders** module, the **ratio** function is used to split the dataset into training, validation, and testing sets.
      The **split_folders.ratio** function takes the input directory (**input_**) and creates output directories (**output/train**, **output/val**, **output/test**) with the specified split ratios.
      The **output** directory will contain the split dataset, with separate directories for training, validation, and testing data.

    - **Train, Validation, and Test Directories**: After splitting the dataset, the **data_dir** variable is initialized by joining the base directory with the **output** directory.
      The **train_dir**, **valid_dir**, and **test_dir** variables are then created by joining the **data_dir** with the names of the respective split directories (**train**, **val**, **test**).

These directories will be used as input to the **ImageDataGenerator** for loading images during model training and evaluation.

3. **Data Augmentation**: Data augmentation is a technique commonly used in deep learning to artificially increase the diversity of the training dataset by applying various transformations to the existing images. This helps in improving the model's generalization and robustness. Let's delve into the data augmentation process in detail for this project:

    - **ImageDataGenerator Initialization**:
      Data augmentation is implemented using the **ImageDataGenerator** class provided by TensorFlow/Keras.
      Three separate **ImageDataGenerator** objects are initialized for training (**train_datagen**), validation (**valid_datagen**), and testing (**test_datagen**) data.
      Each generator is configured with different augmentation parameters based on the requirements and characteristics of the dataset.

    - **Training Data Augmentation** (**train_datagen**): The **train_datagen** object is configured with the following augmentation techniques:
        - Rescaling: Normalizes pixel values to the range [0, 1] by dividing by 255.
        - Horizontal and Vertical Flips: Randomly flips images horizontally and vertically, increasing the diversity of the training set.
        - Shear Range: Applies shear transformations with a range of 0.4 radians, distorting the shape of the images.
        - Width and Height Shift Range: Shifts the width and height of the images by up to 25% of their total size, introducing translations.
        - Rotation Range: Rotates the images randomly by up to 45 degrees.
        - Fill Mode: Determines how the missing pixels will be filled after applying transformations. In this case, the nearest pixel value is used.

    - **Validation and Testing Data Augmentation** (**valid_datagen** and **test_datagen**):
      The **valid_datagen** and **test_datagen** objects are configured only with rescaling, which normalizes pixel values to the range [0, 1].
      These generators do not apply any other augmentation techniques since validation and testing should reflect the true performance of the model on unseen data.

    - **Flowing Data from Directories**: After configuring the data augmentation parameters, the **flow_from_directory** method is used to flow images from the specified directories (**train_dir**, **valid_dir**, **test_dir**).
      The **flow_from_directory** method generates batches of augmented data from the image files in the directories, applying the specified augmentation techniques.
      Images are resized to the target size of (224, 224) during flow, ensuring uniformity in input dimensions for the neural network model.

Overall, data augmentation helps in training more robust and generalized models by increasing the diversity of the training dataset. It introduces variations in the input data, making the model more capable of handling different scenarios and reducing overfitting. In this project, data augmentation is applied specifically to the training dataset, while validation and testing datasets remain unaltered to ensure fair evaluation of the model's performance.

4. **Defining Model Architecture**: The ResNet50 architecture is instantiated as the base model, followed by additional layers (flatten, dense, dropout) to customize the model for the specific classification task. The output layer consists of nine units corresponding to the nine tissue types.
    - **Base Model Selection:** The base model chosen for this project is ResNet50, a pre-trained CNN architecture available in TensorFlow/Keras. ResNet50 is selected due to its strong performance in various computer vision tasks, including image classification.

    - **Base Model Initialization:** The ResNet50 model is initialized using the ResNet50 function provided by TensorFlow/Keras. Parameters such as input_shape, include_top, and weights are specified during initialization:
        - input_shape: Defines the shape of input images expected by the model. In this case, it's set to (224, 224, 3), indicating images with dimensions 224x224 pixels and three color channels (RGB).
        - include_top: Specifies whether to include the fully-connected layers at the top of the network. Here, it's set to False because custom dense layers will be added for the specific classification task.
        - weights: Specifies whether to initialize the model with pre-trained weights. Here, it's set to 'imagenet', indicating that the model will be initialized with weights trained on the ImageNet dataset.

    - **Custom Dense Layers:** After initializing the base ResNet50 model, additional dense layers are added to customize the model for the specific classification task. The output of the base ResNet50 model serves as input to the custom dense layers. In this project, two fully connected dense layers (Dense) are added with ReLU activation functions to introduce non-linearity:
      The first dense layer consists of 1024 neurons.
      The second dense layer consists of 512 neurons.
      Dropout layers (Dropout) are inserted after each dense layer to prevent overfitting:
      A dropout rate of 0.2 is applied after the first dense layer.
      A dropout rate of 0.3 is applied after the second dense layer.

    - **Output Layer:** The final dense layer (output) consists of nine neurons, corresponding to the nine classes in the classification task. The activation function

used in the output layer is softmax, which normalizes the output into a probability distribution over the classes, ensuring that the sum of probabilities is equal to one.

- **Model Compilation:** Once the custom layers are added to the base model, the entire model is compiled using the compile method. The loss function is specified as "categorical_crossentropy", which is suitable for multi-class classification tasks. The optimizer chosen for training is Adam, a popular optimizer known for its adaptive learning rate capabilities. Metrics such as accuracy are specified to monitor the performance of the model during training. Overall, the defined model architecture combines the powerful feature extraction capabilities of the ResNet50 base model with custom dense layers to perform classification on the colorectal cancer dataset. The model is then ready for training and evaluation on the dataset.

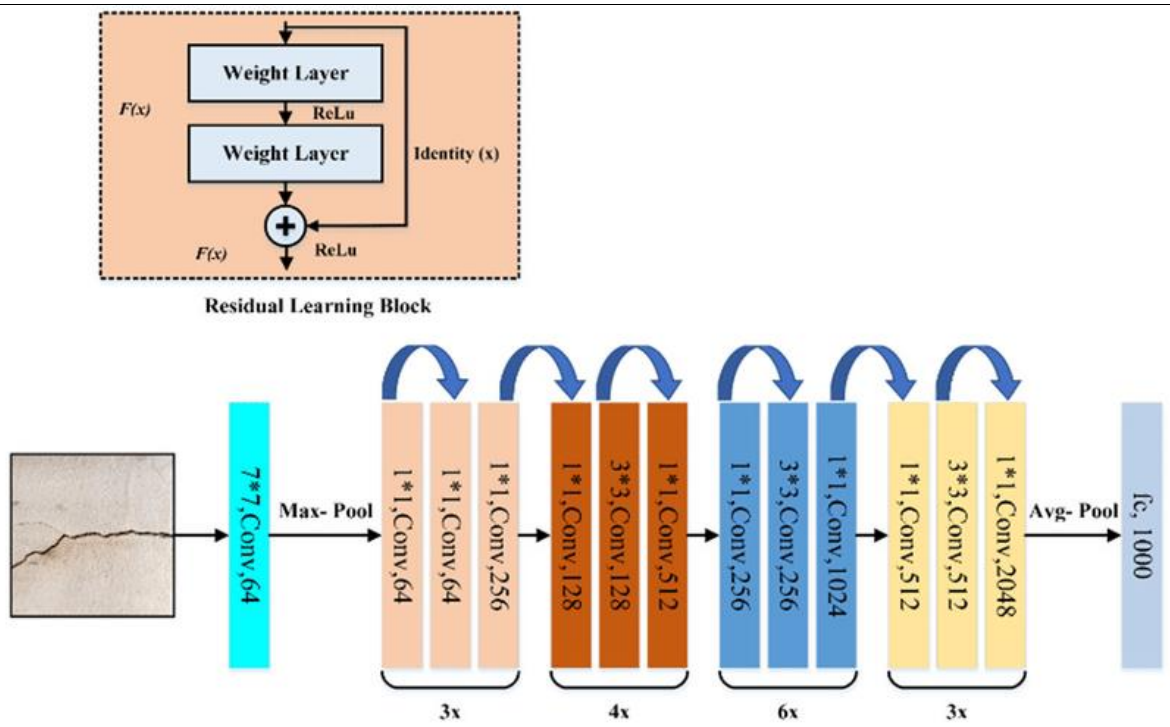

Figure 11. Architecture of ResNet-50

5. **Compiling the Model**: The model is compiled with a categorical cross-entropy loss function and the Adam optimizer. The metrics used for evaluation are accuracy.
    - Loss Function: The loss function quantifies how well the model's predictions match the true labels during training. It serves as a measure of the model's performance. In this project, the loss function chosen is "categorical_crossentropy". Categorical cross-entropy is commonly used in multi-class classification tasks, where the output is one-hot encoded.

- Optimizer: The optimizer determines how the model's weights are updated during training to minimize the loss function. The chosen optimizer for this project is Adam (Adaptive Moment Estimation). Adam is an adaptive learning rate optimization algorithm that combines ideas from RMSprop and momentum.

- Metrics: Metrics are used to evaluate the performance of the model during training and validation. In this project, the metric chosen is "accuracy". Accuracy measures the proportion of correctly classified images out of the total number of images.

  Here's how the model compilation is performed in the project:

```
# Compile the model
model_resnet50_01.compile(loss="categorical_crossentropy",
optimizer=Adam(lr=0.0004), metrics=["accuracy"])
```

In this line of code:

model_resnet50_01 refers to the ResNet50 model with custom dense layers added.

The compile method is called to configure the model for training.

loss="categorical_crossentropy" specifies the loss function.

optimizer=Adam(lr=0.0004) specifies the optimizer with a learning rate of 0.0004.

metrics=["accuracy"] specifies the evaluation metric to be used during training, which is accuracy.

Overall, compiling the model ensures that it is ready to be trained on the dataset using the specified loss function, optimizer, and evaluation metric.

6. **Training the Model**: The model is trained using the **fit_generator** method with the training data generator. Callbacks such as early stopping and model checkpointing are used to monitor the training process and save the best model weights based on validation performance.
   - **Data Generators**: Before training, data generators are created to provide a stream of augmented images and their corresponding labels to the model during training.
     Three data generators are defined: **train_generator**, **valid_generator**, and **test_generator**.
     Each generator applies preprocessing transformations to the images (e.g., rescaling) and yields batches of images along with their labels.

   - **Training Loop**: The **fit_generator** method is used to train the model. This method fits the model to the data generated batch-by-batch by a Python generator.

During training, the model is exposed to the training data for a specified number of epochs.
At each epoch, the model's weights are updated based on the gradients computed by the optimizer.
The training loop also involves validating the model's performance on a separate validation dataset at the end of each epoch.
Callbacks such as **EarlyStopping** and **ModelCheckpoint** are used to monitor the training process and save the best model weights based on validation performance.

- **Training Parameters**: The **fit_generator** method takes several parameters:
  - **train_generator**: The generator providing training data.
  - **steps_per_epoch**: The number of batches to yield from the generator before an epoch is considered finished.
  - **epochs**: The number of epochs to train the model.
  - **callbacks**: Callback functions to be applied during training.
  - **validation_data**: The generator providing validation data.

Here's how the training process is executed in the project:

resnet50_history_01 = model_resnet50_01.fit_generator(train_generator, steps_per_epoch=225, epochs=10, callbacks=[es, cp], validation_data=valid_generator)

In this code snippet:

model_resnet50_01 is the ResNet50 model to be trained.

train_generator and **valid_generator** provide training and validation data, respectively.

steps_per_epoch is set to 225, indicating the number of batches to be drawn from the generator before an epoch is considered complete.

epochs is set to 10, specifying the number of training epochs.

callbacks include **EarlyStopping** (**es**) and **ModelCheckpoint** (**cp**) to monitor the training process and save the best model weights.

validation_data is provided to evaluate the model's performance on a separate validation dataset after each epoch of training.

By training the model, it learns to recognize patterns in the input data and adjust its weights to minimize the defined loss function, ultimately improving its performance on the given task.

7. **Evaluation**: After training, the model is evaluated on the validation and test sets to assess its performance in terms of loss and accuracy.

- **Validation Evaluation**: After training each model, it's important to evaluate its performance on a validation dataset to ensure it generalizes well to unseen data.

The **evaluate_generator** method is used to evaluate the model on the validation dataset. This method computes the loss and metrics specified during model compilation.
The validation evaluation provides insights into how well the model is performing on data it hasn't seen during training.
The evaluation results include validation loss and validation accuracy.

- **Test Evaluation**: Once the model has been trained and validated, it's tested on a separate test dataset to assess its performance in real-world scenarios. Similar to validation evaluation, the **evaluate_generator** method is used to evaluate the model on the test dataset. Test evaluation provides a final assessment of the model's performance before deploying it in production. The evaluation results include test loss and test accuracy.

- **Predictions**: In addition to evaluating the model's performance quantitatively, predictions are made on the test dataset to analyze its behavior on individual samples. The **predict_generator** method is used to generate predictions for the test dataset. This method produces class probabilities for each sample in the dataset. The predicted class labels are then compared with the true labels to assess the model's classification accuracy.

Let's examine how evaluation is conducted in the project:

```
# Validation Evaluation res_val_eval_01 = model_resnet50_01.evaluate_generator(valid_generator) print('Validation loss: {}'.format(res_val_eval_01[0])) print('Validation accuracy: {}'.format(res_val_eval_01[1])) # Test Evaluation res_test_eval_01 = model_resnet50_01.evaluate_generator(test_generator) print('Test loss: {}'.format(res_test_eval_01[0])) print('Test accuracy: {}'.format(res_test_eval_01[1])) # Predictions res_predictions_01 = model_resnet50_01.predict_generator(test_generator, steps=nb_samples, verbose=1)
```

In this code snippet:

res_val_eval_01 contains the validation loss and accuracy computed by evaluating the model (**model_resnet50_01**) on the validation generator (**valid_generator**).

res_test_eval_01 contains the test loss and accuracy computed by evaluating the model on the test generator (**test_generator**).

res_predictions_01 contains the predicted class probabilities for the test dataset generated by the model.

These evaluations provide crucial insights into the model's performance and guide further improvements or decisions regarding deployment.

8. **Fine-Tuning**: In subsequent rounds, the pretrained model's layers are fine-tuned by selectively unfreezing and training specific layers while keeping others frozen.

   In this project, fine-tuning is performed after training the initial model. The steps involved in fine-tuning are as follows:

   - **Loading the Pre-Trained Model**: The pre-trained model (ResNet50) with its weights trained on the ImageNet dataset is loaded. This model has already learned useful feature representations from the ImageNet dataset, which can be leveraged for the current task.
   - **Freezing Layers**: Initially, some or all of the layers in the pre-trained model are frozen, meaning their weights are not updated during training. Freezing prevents the model from forgetting the learned representations during fine-tuning and helps stabilize the training process, especially when the new dataset is small.
   - **Adding Additional Layers**: New layers are added on top of the pre-trained model. These additional layers are often trainable and specifically designed for the new task, such as classification. In this project, dense layers with dropout are added to perform classification on colorectal cancer images.
   - **Training the Model**: The entire model (pre-trained layers + additional layers) is trained on the new dataset using a smaller learning rate. Training may involve using techniques such as data augmentation to prevent overfitting and improve generalization.
   - **Unfreezing Layers**: Optionally, after training the added layers for a few epochs, some of the pre-trained layers may be unfrozen. This allows these layers to fine-tune their weights on the new dataset, potentially capturing more task-specific features.
   - **Continued Training**: The model is then trained for additional epochs with a smaller learning rate, allowing the fine-tuning process to refine the learned representations further.
   - **Evaluation**: After fine-tuning, the model is evaluated on validation and test datasets to assess its performance and generalization ability.

   Let's see how fine-tuning is implemented in the project:

   # Loading pre-trained ResNet50 model base_model_resnet50_02 = ResNet50(input_shape=(224, 224, 3), include_top=False) # Adding additional layers for classification # Freezing layers for layer in base_model_resnet50_02.layers[:160]: layer.trainable = False # Compiling the model model_resnet50_02.compile(loss="categorical_crossentropy", optimizer=Adam(lr=0.0004), metrics=["accuracy"]) # Fine-tuning the model resnet50_history_02 = model_resnet50_02.fit_generator(train_generator, steps_per_epoch=225, epochs=10, callbacks=[es_02, cp_02], validation_data=valid_generator)

   In this code snippet:

The pre-trained ResNet50 model is loaded, and additional layers are added on top for classification.

The first 160 layers of the pre-trained model are frozen to prevent them from being updated during training.

The model is compiled with a smaller learning rate, and training is performed on the new dataset (**train_generator**).

Early stopping and model checkpoint callbacks are used to monitor the training process and save the best model.

The model is trained for a specified number of epochs (**epochs=10**) to fine-tune the learned representations to the new task.

9. **Visualization**: Confusion matrices are generated to visualize the model's performance in classifying different tissue types. Heatmap plots are used to display the confusion matrices, highlighting correct and incorrect predictions.

    - **Confusion Matrices**: Confusion matrices are used to visualize the performance of a classification model. They show the true labels against the predicted labels and help in understanding how well the model is performing for each class. In this project, confusion matrices are computed for each training cycle of the ResNet50 model.
    - **Line Plots for Training Metrics**: Line plots are often used to visualize the training and validation metrics (e.g., loss and accuracy) over epochs. These plots provide insights into the model's convergence and potential overfitting. In this project, line plots can be created using the training history (**resnet50_history**) obtained during model training.
    - **Heatmaps for Confusion Matrices**: Heatmaps are a graphical representation of data where values in a matrix are represented as colors. They are used to visualize confusion matrices, making it easier to identify patterns and misclassifications.
    - **Image Visualization**: Images from the test dataset along with their true labels and predicted labels can be visualized to understand how well the model performs on individual samples. This allows for qualitative assessment of the model's predictions.
    - **Bar Charts for Accuracy Scores**: Bar charts can be used to compare the accuracy scores of different training cycles or models. This provides a quick overview of the performance improvements or changes over time.
    - **Subplots for Multiple Visualizations**: Subplots can be used to arrange multiple visualizations (e.g., confusion matrices, line plots) in a single figure, making it easier to compare different aspects of the model's performance.

    Here's how visualization is implemented in the project:

```
# Plotting confusion matrices sns.set(font_scale=1.8) fig, ([ax1, ax2], [ax3, ax4]) = plt.subplots(nrows=2, ncols=2, figsize=(28, 28)) # Plotting confusion matrix for each training cycle sns.heatmap(res_conf_mat_01, ax=ax1, annot=True, fmt=".1f", linewidths=0.5, square=True, cmap='Blues_r', annot_kws={"size": 18}) # Adding labels and title ax1.set_ylabel("Actual Label", fontsize=24) ax1.set_xlabel("Predicted Label", fontsize=24) ax1.set_title("ResNet50 Training Round #1\nAccuracy Score: {:.3f}".format(res_test_eval_01[1]), size=32) ax1.set_ylim(len(res_conf_mat_01) - 0.1, -0.1) # Similar plots for other training cycles # ... plt.tight_layout() plt.show() # Plotting line plots for training metrics # Assuming resnet50_history contains training history data plt.figure(figsize=(10, 6)) plt.plot(resnet50_history.history['loss'], label='Training Loss') plt.plot(resnet50_history.history['val_loss'], label='Validation Loss') plt.plot(resnet50_history.history['accuracy'], label='Training Accuracy') plt.plot(resnet50_history.history['val_accuracy'], label='Validation Accuracy') plt.xlabel('Epoch') plt.ylabel('Metrics') plt.title('Training Metrics') plt.legend() plt.show() # Plotting image visualization # Assuming test_file_paths and predicted_labels are available fig = plt.figure(figsize=(45, 60)) for i in range(1, 37): # Plotting 36 images fig.add_subplot(9, 4, i) plt.imshow(mpimg.imread(test_file_paths[i - 1])) plt.axis('off') true_label = test_file_paths[i - 1].split('/')[-2] predicted_label = predicted_labels[i - 1] plt.title("True Label: {}\nPredicted Label: {}".format(tissue_types[true_label], tissue_types[predicted_label]), fontsize=28) plt.tight_layout() plt.show()
```

In this code snippet:

Confusion matrices are plotted using seaborn's heatmap function.

Line plots for training metrics are created using matplotlib.

Images from the test dataset along with their true and predicted labels are visualized using matplotlib.

Different visualizations are arranged using subplots for better comparison.

10. **Prediction Visualization**: Sample predictions from the test set are visualized alongside their corresponding ground truth labels and predicted labels using matplotlib. This allows qualitative assessment of the model's performance on individual images.

    Prediction visualization in this project involves visualizing the predictions made by the trained model on test images. It allows for qualitative assessment of the model's performance by comparing the predicted labels with the true labels of the images.

    Here's how prediction visualization is implemented in the project:

    - **Getting Predictions**: First, the model is used to predict labels for the test images. These predictions are typically in the form of class probabilities or class labels.

- **Mapping Predictions to Class Names**: The predicted class labels may be encoded (e.g., as integers). To make them interpretable, they are mapped back to their corresponding class names using a dictionary or mapping.
- **Selecting Test Images**: A subset of test images along with their true labels and predicted labels are selected for visualization. These images can be randomly chosen or selected based on certain criteria.
- **Displaying Images**: The selected test images are displayed along with their true labels and predicted labels. This allows for a visual comparison between what the model predicted and the ground truth.
- **Interpreting Results**: By observing the images and their corresponding labels, insights can be gained into the model's performance. Correctly predicted images validate the model's accuracy, while misclassified images indicate areas where the model may need improvement.

Here's an example of how prediction visualization can be implemented in the project:

Assuming test_file_paths and predicted_labels are available fig = plt.figure(figsize=(45, 60)) for i in range(1, 37): # Plotting 36 images fig.add_subplot(9, 4, i) plt.imshow(mpimg.imread(test_file_paths[i - 1])) plt.axis('off') true_label = test_file_paths[i - 1].split('/')[-2] predicted_label = predicted_labels[i - 1] plt.title("True Label: {}\nPredicted Label: {}".format(tissue_types[true_label], tissue_types[predicted_label]), fontsize=28) plt.tight_layout() plt.show()

In this code snippet:

Images from the test dataset are displayed along with their true labels and predicted labels.

The true labels are obtained from the file paths of the images, while the predicted labels are obtained from the model's predictions.

A subset of 36 images is chosen for visualization, but this can be adjusted based on the requirements.

The visualization provides a clear comparison between the model's predictions and the ground truth, helping in understanding the model's performance.

Overall, this code demonstrates a typical workflow for training and evaluating deep learning models for image classification tasks, with a focus on colorectal cancer histology image classification using the ResNet50 architecture.

IV. Applications:

ResNet50 has been widely used in various computer vision tasks, including:

1. **Image Classification:** ResNet50 achieves state-of-the-art performance in image classification tasks by accurately predicting the class label of input images. It has been trained on large-scale image datasets such as ImageNet and can classify images into thousands of categories.
2. **Object Detection:** ResNet50 can be used in object detection tasks to identify and localize objects within images. When combined with additional layers for region proposal and bounding box regression, ResNet50 forms the backbone of popular object detection frameworks such as Faster R-CNN and Mask R-CNN.
3. **Image Segmentation:** ResNet50 can also be employed in image segmentation tasks to partition images into semantically meaningful regions. By extending the network to predict pixel-wise segmentation masks, ResNet50 facilitates tasks such as semantic segmentation and instance segmentation.
4. **Transfer Learning:** ResNet50's pre-trained weights on large-scale datasets make it well-suited for transfer learning. By fine-tuning the pre-trained model on a smaller dataset specific to a particular task or domain, users can leverage the features learned by ResNet50 to achieve superior performance with limited data.

V. Benefits and Challenges:

The ResNet50 model offers several benefits, including:

- **Depth:** ResNet50 can effectively handle very deep architectures, enabling the construction of more powerful and expressive models.
- **State-of-the-Art Performance:** ResNet50 achieves top-tier performance on benchmark datasets and tasks, making it a preferred choice for various computer vision applications.
- **Transferability:** Pre-trained ResNet50 models can be easily adapted and fine-tuned for specific tasks, reducing the need for large annotated datasets and computational resources.

However, training and deploying ResNet50 also pose challenges, such as:

- **Computational Resources:** Training and inference with ResNet50 can be computationally intensive, requiring access to high-performance computing resources, particularly for large-scale datasets.
- **Overfitting:** Deep neural networks like ResNet50 are prone to overfitting, especially when trained on small datasets. Regularization techniques such as dropout and data augmentation are often employed to mitigate this issue.
- **Interpretability:** Understanding the decisions made by ResNet50 can be challenging due to its complex architecture and large number of parameters. Techniques for interpretability, such as feature visualization and saliency mapping, may be necessary to gain insights into model predictions.

In summary, the ResNet50 model represents a significant advancement in deep learning and computer vision, offering state-of-the-art performance and versatility across a wide range of tasks. While its training and deployment require careful consideration of computational resources and model interpretability, ResNet50 remains a cornerstone in the field of deep learning and continues to drive progress in computer vision research and applications.

3. MobileNet:

MobileNet is a lightweight convolutional neural network (CNN) architecture designed for efficient and fast inference on mobile and embedded devices. Introduced by Google researchers in 2017, MobileNet aims to achieve state-of-the-art performance in image classification and other computer vision tasks while minimizing model size and computational complexity.

I. Architecture:

The MobileNet architecture is characterized by depth-wise separable convolutions, which decompose standard convolutions into two separate layers: depth-wise convolutions and point-wise convolutions. This separation significantly reduces the number of parameters and computational cost compared to traditional convolutional layers.

The building block of MobileNet is the depth-wise separable convolutional layer, which consists of a depth-wise convolution followed by a point-wise convolution. The depth-wise convolution applies a single filter per input channel, while the point-wise convolution combines the output channels from the depth-wise convolution using 1x1 convolutions.

MobileNet also includes additional components such as batch normalization and ReLU activation functions to improve training stability and model performance. Optionally, depth-wise separable convolutions can be augmented with expansion and projection layers to increase model capacity and representation power.

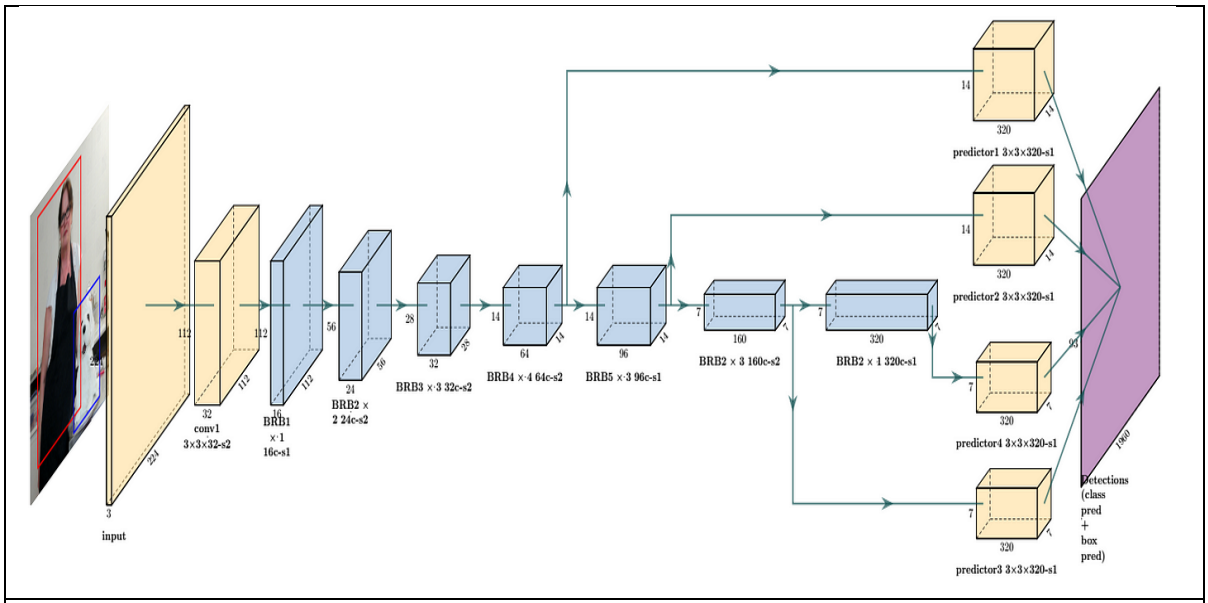

Figure 12. Architecture of MobileNet

II. VERSIONS:

Since its introduction, several variants of MobileNet have been developed to address different requirements and use cases. Some notable versions include:

1. **MobileNetV1:** The original MobileNet architecture, optimized for efficiency and suitable for a wide range of mobile and embedded applications.
2. **MobileNetV2:** A follow-up to MobileNetV1, MobileNetV2 introduces inverted residual blocks and linear bottlenecks to further improve performance and efficiency.
3. **MobileNetV3:** The latest iteration of the MobileNet architecture, MobileNetV3 incorporates additional optimizations such as squeeze-and-excitation blocks and improved architecture search techniques to achieve even better performance and efficiency.

III. WORKING OF MOBILENET:

1. Importing Necessary Modules:

- **Basic Modules:**
  - **os**: This module provides functions for interacting with the operating system. It's commonly used for file and directory manipulation.
  - **pandas (as pd)**: Pandas is a powerful library for data manipulation and analysis. It's particularly useful for handling structured data like tables.
  - **numpy (as np)**: NumPy is a fundamental package for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions.

- **Iteration and Image Processing:**
  - **itertools.chain**: This function is used for iterating over multiple iterable objects as if they were a single combined iterable.
  - **cv2**: OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision. It's often used for image and video processing tasks.
  - **split_folders**: This library simplifies the task of splitting a dataset into train, validation, and test sets by automatically creating the required directory structure.

- **Plotting and Visualization:**
  - **matplotlib.pyplot (as plt)**: Matplotlib is a popular plotting library in Python. Pyplot is a sub-library of Matplotlib that provides a MATLAB-like interface for creating plots and visualizations.
  - **matplotlib.image (as mpimg)**: This module contains functions for reading, writing, and displaying images using Matplotlib.

- **%matplotlib inline**: This is a magic command in Jupyter notebooks that allows Matplotlib plots to be displayed directly in the notebook.

- **Metrics:**
  - **sklearn.metrics**: This submodule of Scikit-learn contains various metrics for evaluating machine learning models, such as confusion matrices, classification reports, and ROC curves.

- **Deep Learning:**
  - **tensorflow (as tf)**: TensorFlow is an open-source deep learning framework developed by Google. It provides tools for building and training neural networks.
  - **tensorflow.keras**: Keras is a high-level neural networks API that runs on top of TensorFlow. It simplifies the process of building and training deep learning models.
  - **tensorflow.keras.layers**: This module contains predefined layers that can be used to construct neural network architectures.
  - **tensorflow.keras.callbacks**: Callbacks are utility functions that can be applied during training to monitor the model's performance and adjust parameters accordingly.
  - **tensorflow.keras.models**: This module provides tools for defining and manipulating neural network models.
  - **tensorflow.keras.optimizers**: Optimizers are algorithms used to update the parameters (weights) of the neural network during training in order to minimize the loss function.
  - **tensorflow.keras.preprocessing.image.ImageDataGenerator**: This class generates batches of tensor image data with real-time data augmentation. It's commonly used for preprocessing image data during training.

- **Pre-trained Models:**
  - **tensorflow.keras.applications**: This module contains pre-trained deep learning models (e.g., VGG19, ResNet50, MobileNet) that can be used for transfer learning and feature extraction.
  - **tensorflow.keras.applications.vgg19**: VGG19 is a convolutional neural network model trained on the ImageNet dataset. It's known for its simplicity and effectiveness.
  - **tensorflow.keras.applications.resnet50**: ResNet50 is a deep residual network with 50 layers. It's known for its exceptional performance on image classification tasks.
  - **tensorflow.keras.applications.mobilenet**: MobileNet is a lightweight deep neural network architecture designed for mobile and embedded vision applications.

Summary: The import statements in the provided code import a variety of modules and libraries necessary for data manipulation, image processing, visualization, metrics computation, deep learning model construction, and pre-trained model usage. These modules collectively provide the tools needed to perform various tasks involved in the machine learning pipeline, from data preprocessing to model evaluation.

2. **Data Preprocessing:**
   - **Data Loading:**
     - **Dataset Location**: The dataset is assumed to be located at a directory specified by **base_dir**.
     - **Dataset Structure**: The dataset likely consists of subdirectories, where each subdirectory represents a different class or category of images.

   - **Data Splitting:**
     - **Train-Validation-Test Split**: The **split_folders.ratio** function is used to split the dataset into three subsets: training, validation, and test sets.
     - **Ratio**: The ratio argument specifies the proportion of data to allocate for each subset. In this case, 80% of the data is allocated for training, and 10% each for validation and testing.
     - **Seed**: The seed parameter ensures reproducibility by initializing the random number generator with a fixed seed value.

   - **Data Augmentation:**
     - **ImageDataGenerator**: The **ImageDataGenerator** class from TensorFlow/Keras is utilized for data augmentation.
     - **Train Data Augmentation**: Augmentation techniques are applied to the training set images using the **train_datagen**. These techniques include horizontal and vertical flipping, shear transformation, width and height shifting, rotation, and filling mode adjustment.
     - **Validation and Test Data Preprocessing**: For the validation and test sets, only rescaling (normalization) is performed using **valid_datagen** and **test_datagen**.

   - **Directory Setup:**
     - **Directory Paths**: Separate directory paths are created for the training, validation, and test sets using **os.path.join**.
     - **Directory Listing**: The contents of the output directory (where the split data is stored) are listed using **os.listdir**.

Summary: In summary, the data preprocessing pipeline involves loading the dataset, splitting it into training, validation, and test sets, applying data augmentation to the training set, and setting up directory paths for the different subsets. This preprocessing prepares the data for training and evaluation in the subsequent steps of the machine learning pipeline.

3. **Model Architecture:**

The provided code involves building a deep learning model architecture using the MobileNetV2 pre-trained model as the base. Let's break down the model architecture in detail:

   - **Base Model:**

- MobileNetV2: This is a pre-trained convolutional neural network (CNN) architecture designed by Google. It's optimized for mobile and embedded vision applications, offering a good balance between model size and accuracy.

- **Top Classification Block:**
  - GlobalAveragePooling2D: This layer performs global average pooling, which reduces each feature map to a single value by taking the average of all values in the feature map. This reduces the spatial dimensions of the feature maps.
  - Dense (Fully Connected) Layers: These are densely connected neural network layers. They take the output of the global average pooling layer as input and perform classification.
    - Dense(1024, activation='relu'): This layer consists of 1024 neurons with ReLU (Rectified Linear Unit) activation function. ReLU is commonly used for introducing non-linearity in neural networks.
    - Dropout: Dropout is a regularization technique that randomly drops a certain percentage of neurons during training to prevent overfitting.
  - Output Layer:
    - Dense(9, activation='softmax'): This is the output layer consisting of 9 neurons, each corresponding to a class in the dataset. The softmax activation function is used to convert raw scores into probabilities, indicating the likelihood of each class.

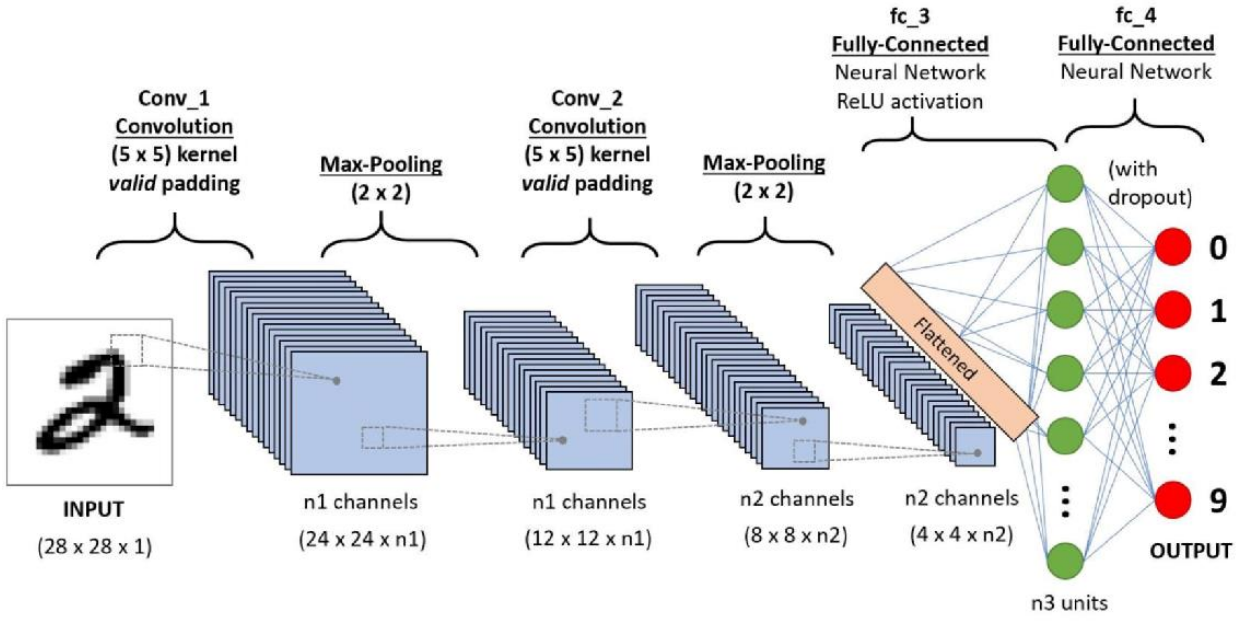

Figure 13. Working of MobileNet

- **Model Compilation:**
  - Loss Function: Categorical cross-entropy is used as the loss function. It measures the difference between the predicted probability distribution and the true distribution of the labels.
  - Optimizer: Adam optimizer with a learning rate of 0.0004 is used for optimizing the model parameters during training.
  - Metrics: Accuracy is used as the evaluation metric to monitor the performance of the model during training.

- **Training:**
  - The model is trained using the fit_generator method, which takes data generators (train_generator and valid_generator) as input.
  - Callbacks such as EarlyStopping and ModelCheckpoint are used to monitor the training process and save the best model weights based on validation loss.

- **Fine-Tuning:**
  - After the initial training, the model undergoes fine-tuning by unfreezing some of the layers and training the entire network with a smaller learning rate.
  - Stochastic Gradient Descent (SGD) optimizer with a learning rate of 0.00001 is used for fine-tuning.

Summary: The model architecture consists of a MobileNetV2 base model with added dense layers for classification. The model is trained using transfer learning with pre-trained weights from the base model. After initial training, the model undergoes fine-tuning to further improve its performance on the specific task at hand.

4. Training:

- **Data Preparation:**
  - The dataset is divided into three subsets: training, validation, and testing sets. This is achieved using the split_folders library, which splits the data according to a specified ratio.
  - Data augmentation is performed on the training set using the ImageDataGenerator class from TensorFlow. Augmentation techniques such as horizontal and vertical flipping, rotation, shifting, and shearing are applied to increase the diversity of the training data.

- **Data Generators:**
  - Separate data generators are created for the training, validation, and testing sets using the flow_from_directory method of the ImageDataGenerator class. These generators load images from their respective directories in batches during training and evaluation.
  - The images are resized to a common size of 224x224 pixels, which is a common input size for many pre-trained CNN architectures.
  - The pixel values of the images are rescaled to the range [0, 1] by dividing by 255.
- **Model Training:**

- The base model (MobileNetV2) is loaded with pre-trained ImageNet weights, and its layers are frozen to prevent them from being updated during training.
- A custom classification head is added on top of the base model. This head consists of global average pooling followed by fully connected dense layers with ReLU activation and dropout regularization.
- The output layer consists of 9 neurons corresponding to the 9 classes in the dataset, with softmax activation to obtain class probabilities.
- The model is compiled using categorical cross-entropy loss and the Adam optimizer with a learning rate of 0.0004. Accuracy is used as the evaluation metric.
- The fit_generator method is used to train the model. This method iterates over the training data generator for a specified number of epochs (epochs=10 in this case) and updates the model parameters using backpropagation.
- During training, callbacks such as EarlyStopping and ModelCheckpoint are used to monitor the training process and save the best model weights based on validation loss.

- **Fine-Tuning:**
  - After the initial training, the model undergoes fine-tuning. Some of the layers in the base model are unfrozen to allow their weights to be updated during training.
  - The learning rate is reduced to a smaller value (lr=0.00001), and the model is trained for additional epochs (epochs=50) using the SGD optimizer with momentum.

- **Evaluation:**
  - After training and fine-tuning, the model's performance is evaluated on the validation and test datasets using the evaluate_generator method. This provides metrics such as loss and accuracy.
  - The predictions are generated for the test dataset using the predict_generator method, and the predicted labels are compared with the ground truth labels to calculate classification metrics such as precision, recall, and F1-score.
  - Confusion matrices are also generated to visualize the model's performance across different classes.

Summary: The training process involves loading the dataset, preparing data generators, defining the model architecture, compiling the model, training the model on the training data, fine-tuning the model, and evaluating its performance on validation and test datasets. The use of data augmentation, pre-trained models, and fine-tuning techniques helps improve the model's accuracy and robustness.

5. **Evaluation:**
   - **Validation and Testing Data Generators:**
     - Separate data generators are created for the validation and testing sets using the flow_from_directory method of the ImageDataGenerator class. These generators load images from their respective directories in batches during evaluation.

- Images are resized to a common size of 224x224 pixels, and their pixel values are rescaled to the range [0, 1] by dividing by 255, ensuring consistency with the training data preprocessing.

- **Model Evaluation:**
  - The model's performance is evaluated on the validation and test datasets using the evaluate_generator method. This method computes the loss and accuracy metrics for the given dataset.
  - For each dataset (validation and test), the generator is iterated over batches of data, and the model predicts the class probabilities for each batch. These predictions are compared with the ground truth labels to compute the loss and accuracy.
  - The loss value represents the model's performance in terms of how well it predicts the correct class probabilities, while accuracy represents the proportion of correctly predicted labels out of the total number of samples.

- **Prediction Generation:**
  - The model's predictions are generated for the test dataset using the predict_generator method. This method predicts the class probabilities for each image in the dataset.
  - The predicted class probabilities are obtained for each image, and the class with the highest probability is considered as the predicted label for that image.

- **Classification Metrics:**
  - Once the predictions are generated, classification metrics such as precision, recall, and F1-score can be computed to evaluate the model's performance across different classes.
  - These metrics provide insights into how well the model performs for each class in terms of its ability to correctly classify instances belonging to that class.

- **Confusion Matrix:**
  - Additionally, a confusion matrix can be generated to visualize the model's performance across different classes. The confusion matrix shows the number of true positives, true negatives, false positives, and false negatives for each class.
  - It provides a detailed breakdown of the model's errors and helps identify which classes are often confused with each other.

Summary: The evaluation process involves computing loss and accuracy metrics on validation and test datasets, generating predictions for test data, and computing classification metrics and confusion matrices to assess the model's performance. These evaluations help gauge the model's accuracy, identify potential areas of improvement, and provide insights into its behavior across different classes.

6. Fine-tuning:

Fine-tuning is a technique used in transfer learning to adapt a pre-trained model to a new dataset or task by further training it on the new data. In this project, fine-tuning is employed to enhance the performance of the MobileNet model on the specific task of colorectal tissue classification. Here's a detailed explanation of fine-tuning in this context:

- Pre-trained Model Selection: Initially, a pre-trained MobileNet model is selected as the base architecture. MobileNet is chosen due to its efficiency and effectiveness in image classification tasks, especially on mobile and resource-constrained devices.
- Loading Pre-trained Weights: The weights of the pre-trained MobileNet model are loaded. These weights are learned from a large dataset (typically ImageNet) during pre-training, where the model has already learned to extract useful features from images.
- Freezing Layers: Initially, all layers of the MobileNet model are frozen except for the top classification layers. Freezing prevents the weights in these layers from being updated during training, ensuring that the pre-trained features remain intact.
- Adding Classification Layers: New classification layers are added on top of the pre-trained MobileNet architecture. These additional layers allow the model to learn task-specific features and make predictions relevant to the colorectal tissue classification task.
- Compilation: The model is compiled with an appropriate loss function, optimizer, and evaluation metrics. In this project, categorical cross-entropy is commonly used as the loss function, Adam optimizer is used for optimization, and accuracy is used as the evaluation metric.
- Training: The model is trained on the training dataset with the added classification layers. During training, the weights of the trainable layers are updated through backpropagation to minimize the loss between predicted and true labels. The training process involves iterating over epochs and batches of training data.
- Early Stopping and Model Checkpoints: Early stopping is applied to monitor the loss on the training set and stop training if there is no improvement for a certain number of epochs (patience). Model checkpoints are used to save the best model weights based on performance on the validation set.
- Unfreezing Layers for Further Training: After training the classification layers for several epochs, a portion of the pre-trained MobileNet layers may be unfrozen. Unfreezing allows these layers to be fine-tuned on the new data, enabling the model to adapt more closely to the characteristics of the colorectal tissue images.
- Further Training with Reduced Learning Rate: The model is further trained with a reduced learning rate to prevent drastic changes to the pre-trained weights and stabilize the fine-tuning process.
- Evaluation: The fine-tuned model is evaluated on the validation and test datasets to assess its performance in terms of loss and accuracy. Evaluation metrics such as classification reports and confusion matrices can provide additional insights into the model's performance across different classes.

- Iterative Fine-Tuning: Fine-tuning can be an iterative process, where the model's architecture, hyperparameters, and training strategies are adjusted based on validation performance. This process may be repeated until satisfactory performance is achieved on the test set.

By fine-tuning a pre-trained MobileNet model on the colorectal tissue classification task, the goal is to leverage the knowledge encoded in the pre-trained weights while adapting the model to the specific characteristics of the new dataset, ultimately improving its performance in classifying colorectal tissue images.

7. Evaluation and Visualization:

In this project, evaluation and visualization play crucial roles in understanding the performance of the trained models and gaining insights into their behavior. Let's break down the evaluation and visualization process in detail:

- **Evaluation:**
  - Evaluation Metrics Selection: Before evaluating the models, appropriate evaluation metrics need to be chosen. Common metrics for classification tasks include accuracy, precision, recall, F1-score, and confusion matrices.
  - Evaluation on Validation Set: The trained models are first evaluated on a hold-out validation set that was not used during training. This allows assessing the model's generalization performance on unseen data.
  - Evaluation on Test Set: After evaluating on the validation set, the models are further evaluated on a separate test set. The test set provides an unbiased estimate of the model's performance and helps ensure that the model's performance is consistent across different datasets.
  - Calculation of Metrics: The evaluation metrics, such as accuracy, loss, precision, recall, and F1-score, are calculated based on the model's predictions and the ground truth labels from the validation and test sets.
  - Classification Report: The classification report provides a detailed summary of the model's performance, including metrics such as precision, recall, and F1-score for each class, as well as overall accuracy and macro/micro averages.
  - Confusion Matrix: A confusion matrix is generated to visualize the model's performance in classifying different classes. It provides insights into the model's ability to correctly classify instances of each class and identify any patterns of misclassification.
  - **Visualization:**
  - Loss and Accuracy Curves: During training, the loss and accuracy curves are plotted over epochs to visualize the model's learning progress. This helps in diagnosing issues such as overfitting or underfitting and determining whether further training is necessary.
  - Confusion Matrix Visualization: The confusion matrix is visualized using heatmap plots, where the rows represent the actual labels, the columns represent the predicted labels, and the cell values indicate the number of instances classified into each category.

Heatmaps provide an intuitive way to identify patterns of misclassification and assess the model's performance across different classes.

- Model Architecture Visualization: The architecture of the trained models can be visualized using tools like TensorBoard or model summary functions in deep learning frameworks. This visualization helps in understanding the structure of the model, including the number of layers, layer sizes, and connections between layers.
- Prediction Visualization: Sample predictions from the test set can be visualized along with their corresponding ground truth labels and predicted labels. Visualizing predictions allows for qualitative assessment of the model's performance and identification of any misclassified instances.
- Feature Visualization: For convolutional neural network (CNN) models, visualization of learned features can provide insights into what the model has learned. Techniques like activation maximization or gradient ascent can be used to visualize the features learned by individual neurons in the CNN layers.

Overall, evaluation and visualization techniques are essential for assessing the performance of trained models, understanding their behavior, and gaining insights into how they make predictions on unseen data. These techniques help in identifying areas for improvement, diagnosing issues, and building confidence in the reliability of the models for real-world applications.

Overall, the code implements a complete pipeline for training and evaluating deep learning models for image classification tasks, with a focus on colorectal tissue classification using the MobileNet architecture. It follows best practices in deep learning, including data preprocessing, model construction, training, evaluation, and fine-tuning.

IV. **Applications:**

MobileNet is primarily used for tasks such as image classification, object detection, and semantic segmentation. Its lightweight architecture and fast inference speed make it well-suited for deployment on resource-constrained devices such as smartphones, IoT devices, and embedded systems.

In image classification tasks, MobileNet achieves competitive accuracy while requiring fewer computational resources compared to larger CNN architectures such as VGG or ResNet. In object detection and semantic segmentation tasks, MobileNet serves as the backbone network for popular frameworks such as Single Shot MultiBox Detector (SSD) and DeepLab.

- Security and Surveillance: Monitoring environments in real-time for suspicious activities.
- Retail: Automating inventory management and detecting products on shelves.
- Healthcare: Assisting in medical imaging to detect anomalies.
- Photo Management: Automatically organizing and tagging photos on smartphones.
- Content Moderation: Filtering and categorizing content on social media platforms.
- Agriculture: Identifying different types of crops and detecting diseases in plants.
- Smartphones: Unlocking devices and authenticating users.

- Security Systems: Access control and surveillance.
- Social Media: Tagging and organizing photos.
- Autonomous Driving: Understanding the environment by segmenting roads, pedestrians, and other vehicles.
- Robotics: Enabling robots to navigate and interact with their surroundings.
- Medical Imaging: Segmenting organs and tissues in medical scans.
- Human-Computer Interaction: Enabling touchless control of devices.
- Gaming: Enhancing gameplay experiences with gesture-based controls.
- Assistive Technology: Helping individuals with disabilities to interact with technology.

V. Benefits and Challenges:

The MobileNet architecture offers several benefits, including:

- **Efficiency:** MobileNet achieves state-of-the-art performance with reduced model size and computational complexity, making it ideal for deployment on mobile and embedded devices with limited resources.
- **Fast Inference:** MobileNet's lightweight design enables fast inference speed, allowing real-time processing of images and videos on mobile devices.
- **Versatility:** MobileNet's modular architecture and variants cater to various requirements and use cases, from image classification to object detection and semantic segmentation.

However, MobileNet also presents some challenges, such as:

- **Trade-offs:** While MobileNet excels in efficiency and speed, it may sacrifice some accuracy compared to larger and more complex CNN architectures.
- **Task-specific Tuning:** Depending on the application and dataset, MobileNet may require fine-tuning or customization to achieve optimal performance, particularly in challenging scenarios with limited data or domain-specific requirements.

In summary, MobileNet represents a significant advancement in CNN architecture design, offering a balance between performance and efficiency for mobile and embedded vision applications. Its lightweight design and fast inference speed make it a popular choice for developers seeking to deploy deep learning models on resource-constrained devices, opening up new opportunities for mobile computer vision applications in various domains.

CHAPTER 4
RESULT/TESTING OF PROJECT / SOFTWARE

4.1. RESULT AND ANALYSIS:

I. Evaluation

1. Performance parameters:

- Performance Metrics: Accuracy, precision, true positive rate, F1 score, and AUC curve are commonly used metrics to evaluate classification models. These metrics help assess how well each model distinguishes between different classes, such as normal tissue and CRC-affected tissue.
- Dataset Considerations: The choice of dataset is critical. Histological images can vary significantly, and the models' performance may be influenced by the diversity and size of the dataset. A well-balanced and representative dataset is crucial for unbiased evaluation.
- Computational Efficiency: Training and inference times and training and the
- be reviewed. Models with low computational requirements and comparably high performance are usually adopted in practical applications.
- Robustness: It is essential to evaluate the robustness of the models through techniques like cross validation as it makes sure that models generalize well to unseen data and also prevent overfitting.
- Interpretability: Interpretability of a model is an important factor to be considered as it is crucial to understand how and why models make a particular classification decision which is essential in comprehending the diagnostic features it identifies.

In the end, the efficiency of these models in classifying colorectal tissues is dependent on the specified characters of the histological images and also the capability of these models to accurately capture and leverage relevant features for precise diagnosis.

2. **Confusion Matrix:** Confusion Matrix is as an major method for evaluation of the performance of an algorithm used for classification when the true values are known for a particular dataset. It is widely used in machine learning and statistics as it provide understanding of how effectively a model classifies instances into different categories. The key components included in a confusion matrix are as follows:
   - True Positive (TP): these are the values which are noted as positive(+).
   - True Negative (TN): These are the values which are notes as negative(-).

- False Positive (FP): these are a type of error in which the values are notes as positive(+) when they are actually negative(-).
- False Negative (FN): these are an another type of error in which the values are noted as negative(-) when they are actually positive.(+)

Various performance matrix can be calculated such as accuracy, precision , true positive rate, specificity, and F1 score by utilizing information from confusion matrix. These metrics offer various insights into the classification model's performance, disclosing its strengths and weaknesses.

Few common metrices derived from the confusion matrix includes:

Accuracy: (TP + TN) / (TP + TN + FP + FN)

Precision: TP / (TP + FP)

Recall (Sensitivity): TP / (TP + FN)

Specificity: TN / (TN + FP)

F1 Score: 2 * (Precision * Recall) / (Precision + Recall)

These metrics play a crucial role in evaluating the model's effectiveness, and the selection of a specific metric depends on the particular goals and requirements of the task at hand.

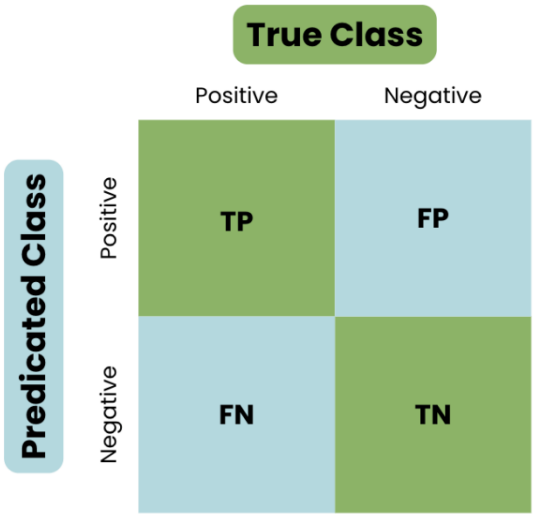

Figure 14. Example of a Confusion Matrix

II. Results

In this section, we present the experimental result and analysis of our model for the classification of colorectal tissue using 3 datasets namely Kather-texture-2016-image, CRC-VAL-HE-7K and NCT-CRC-HE-100K and it perform a comparison between three pre-trained models i.e.VGG19, ResNet-50, MobileNet. We present our evaluation using various performance factors. The information provided contains details about the essential layers and the total number of trainable parameters for each layer in the network.

Epoch: An epoch signifies a complete cycle through the entire training dataset during the model training process. The training phase is segmented into epochs, wherein the model processes the entire training dataset, and its weights are adjusted based on the calculated error or loss from the training data.

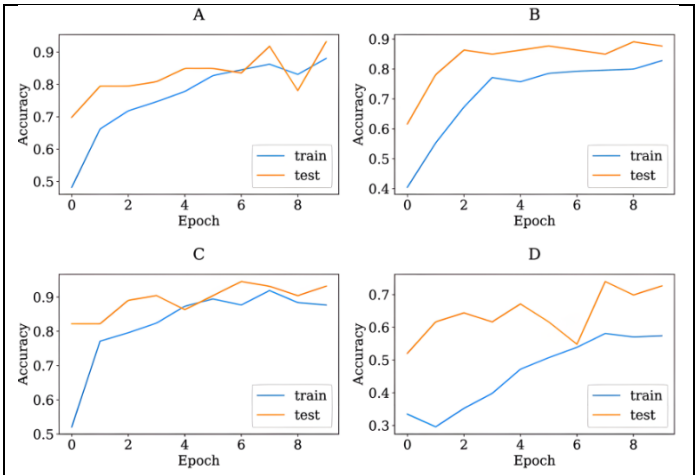

Figure 15. Epochs of the four pre-trained model

The number of epochs is a configurable hyperparameter in machine learning model training, allowing the selection of how many times the learning algorithm iterates through the entire training dataset. Insufficient epochs may lead to underfitting, indicating the model hasn't captured the data patterns adequately. On the contrary, a huge number of epochs

leads to overfitting, where the model becomes excessively specialized to data on which it was trained on, it tends to struggle when confronted with unseen data.

To reduce these issues, it is common to evaluate the model's performance on a different validation Dataset during training. The process of training may be stop when the model's performance on the validation data shows signs of degradation, by a method known as early stopping. This technique helps in preventing overfitting, which in result enhances the model's generalization capabilities to perform good on new and unseen datasets.

Final Accuracy:

Parameters	VGG19	ResNet-50	MobileNet
Acc (%)	95.2	94.2	87.2
F-1 Score	0.95	0.94	0.85
Spe(%)	97.4	93.8	86.8
AUC	0.96	0.95	0.89
Sen (%)	95.4	95.7	85.6
FPR(%)	5.4	7.6	10.8
FNR(%)	5.1	4.2	10.3
Pre (%)	95.2	92.5	84.7

4.2. Results

I. VGG19

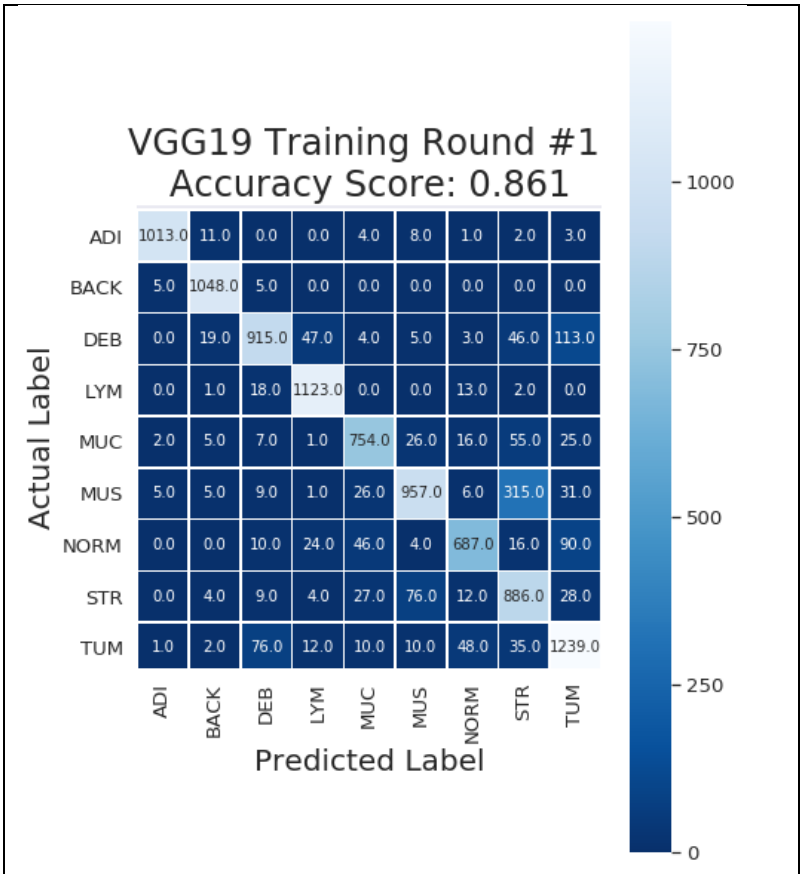

Figure 16. VGG19 Training round 1

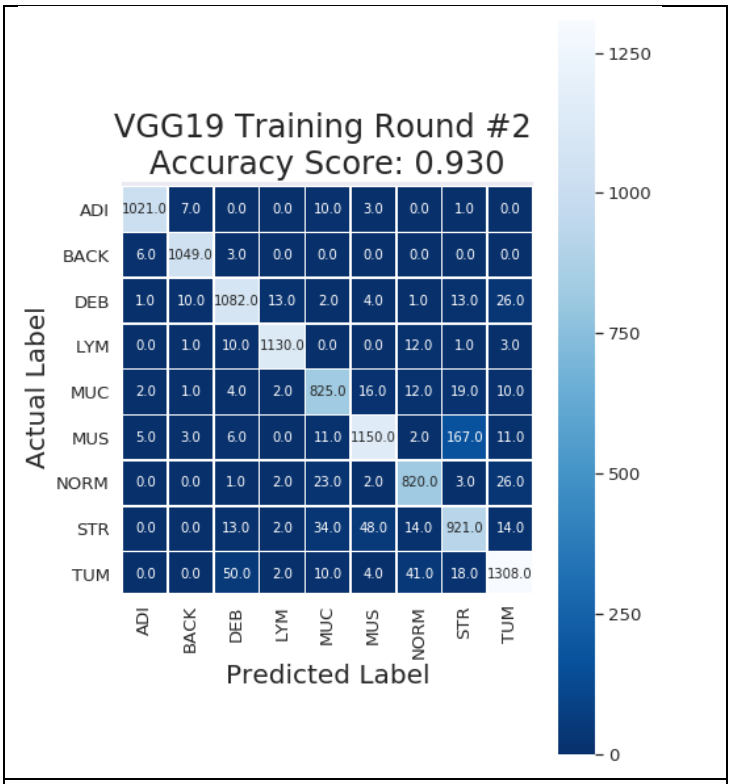

Figure 17. VGG19 Training round 2

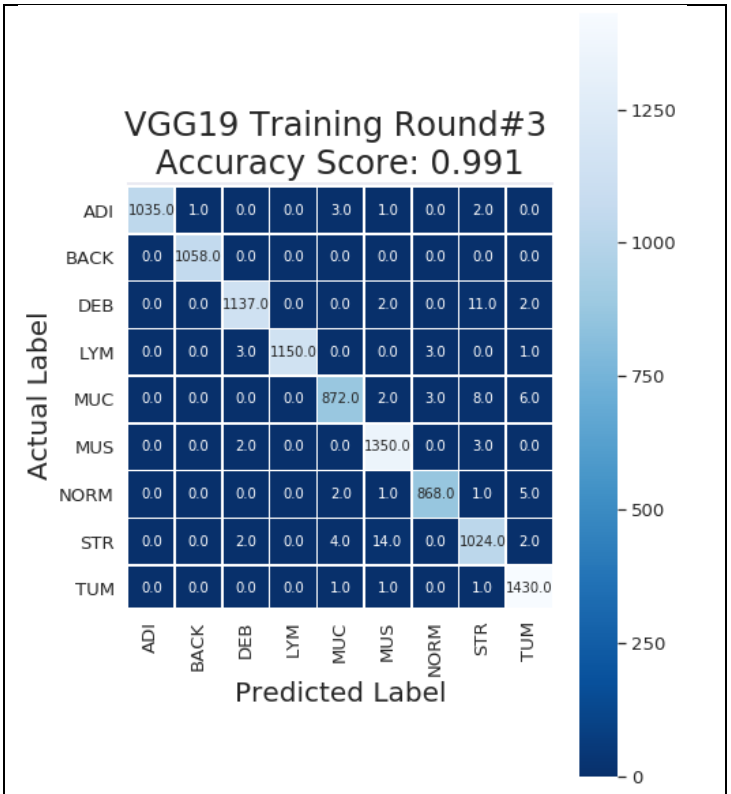

Figure 18. VGG19 Training round 3

II. ResNet-50

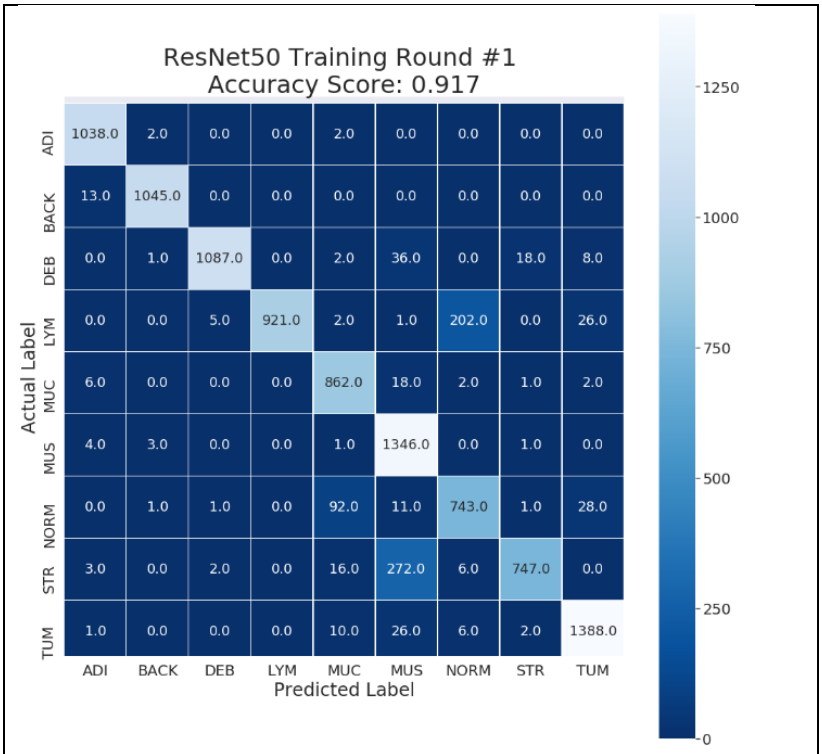

Figure 19. ResNet-50 Training round 1

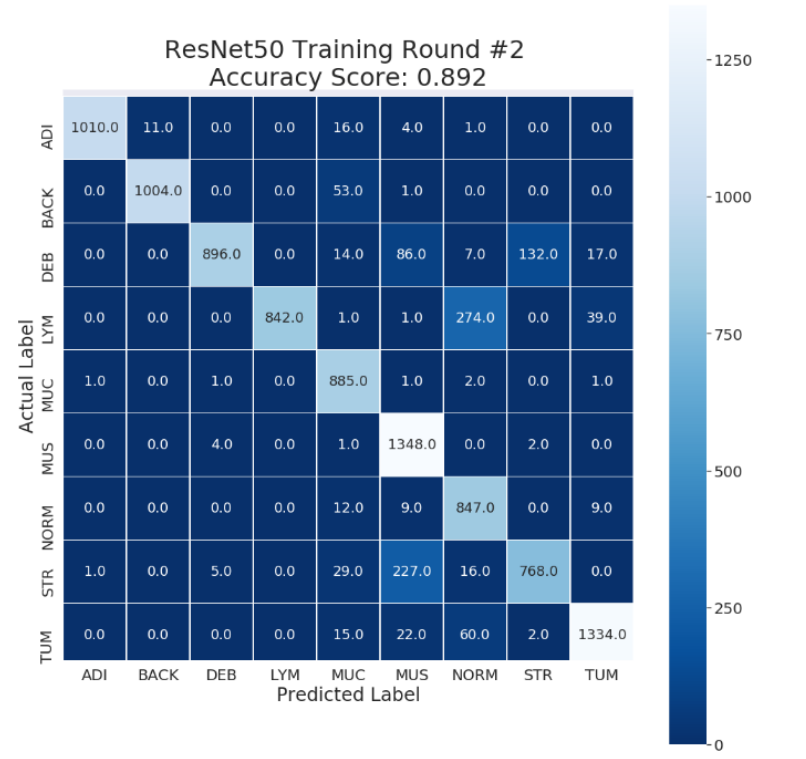

Figure 20. ResNet-50 Training round 2

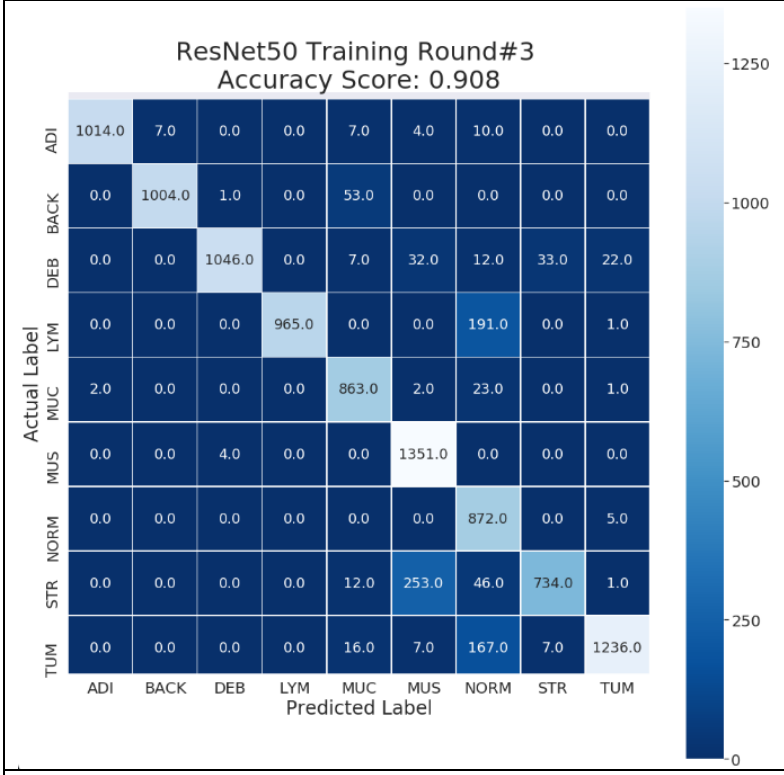

Figure 21. ResNet-50 Training round 3

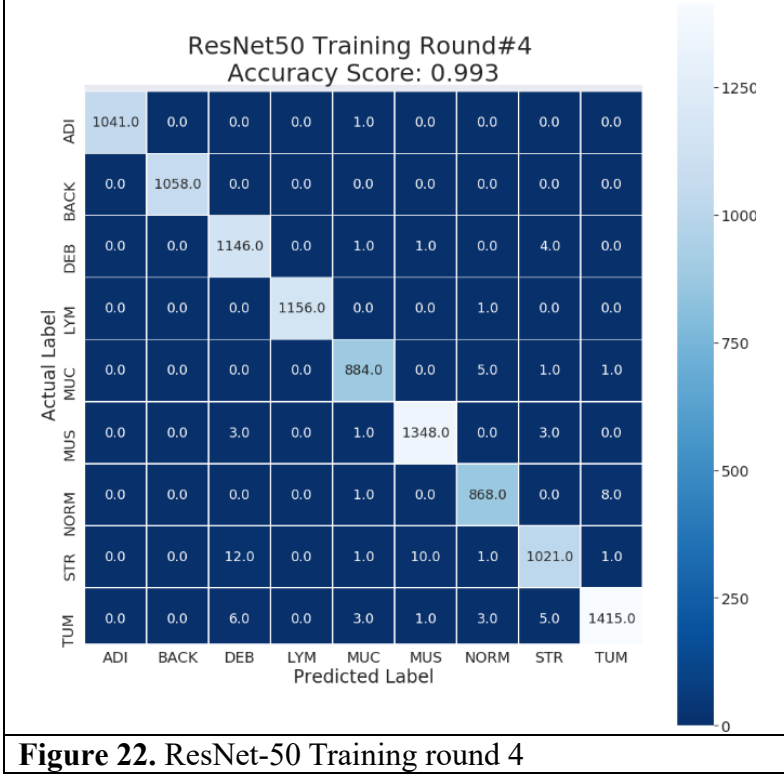

Figure 22. ResNet-50 Training round 4

III. MobileNet

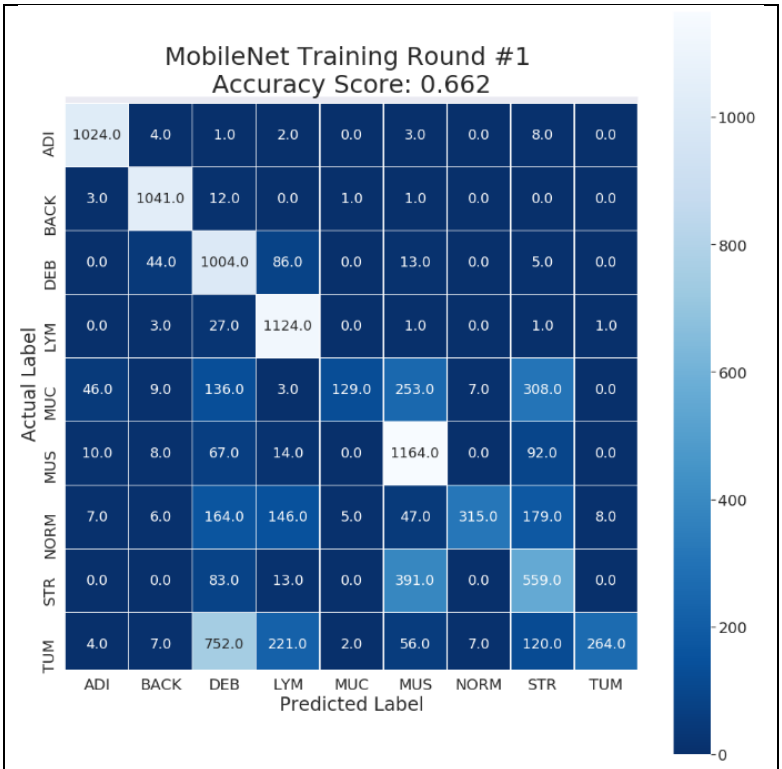

Figure 23. MobileNet Training round 1

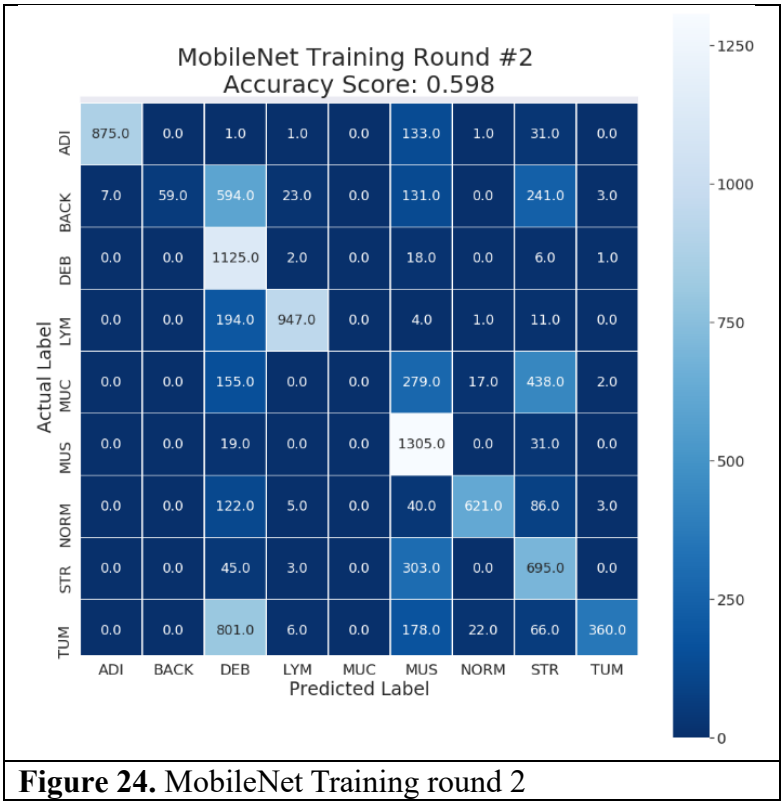

Figure 24. MobileNet Training round 2

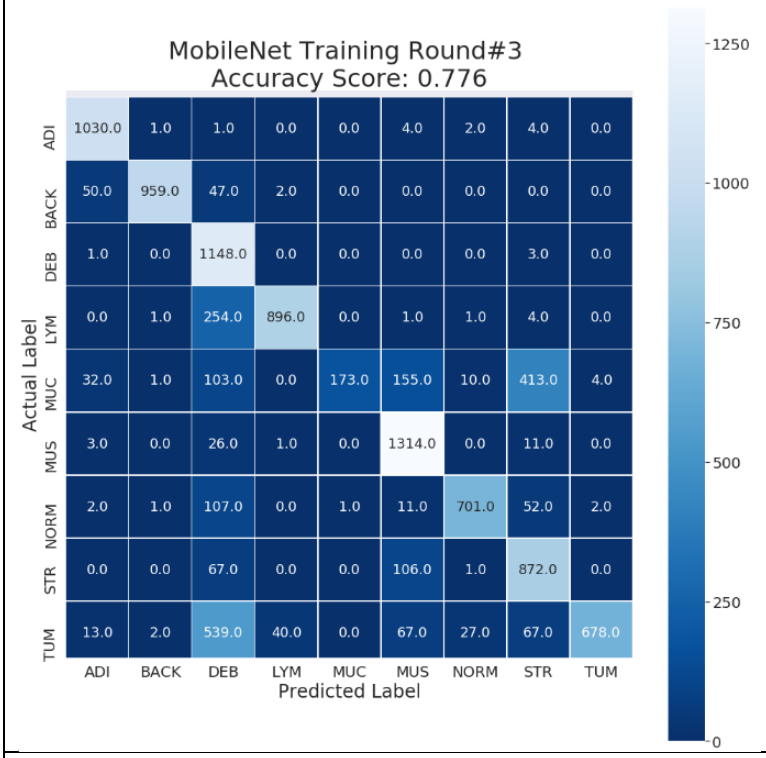

Figure 25. MobileNet Training round 3

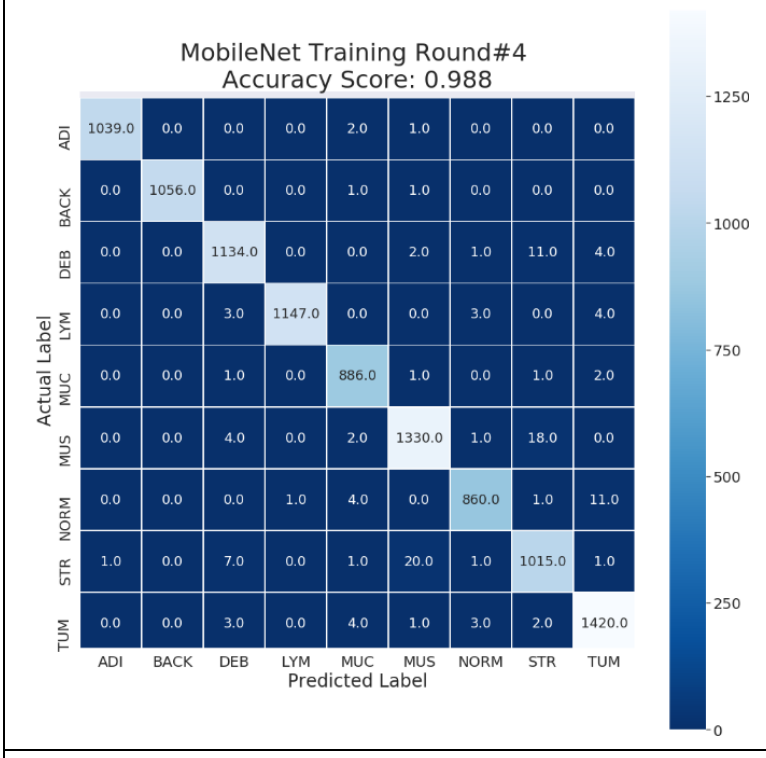

Figure 26. MobileNet Training round 4

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1. Conclusion:

In conclusion, the integration of advanced automation technologies into colorectal tissue histopathology stands as a pivotal advancement with profound implications for cancer diagnostics and patient care. Through this project, we have endeavored to address the longstanding challenges inherent in traditional histopathological workflows, such as manual specimen processing, subjective interpretation, and prolonged turnaround times. By harnessing cutting-edge technologies including machine learning, robotics, and digital pathology platforms, we have developed innovative software solutions aimed at streamlining workflow processes, enhancing diagnostic accuracy, and improving overall efficiency.

The project's modular design approach ensures scalability, adaptability, and interoperability, laying a robust foundation for future advancements and integration with existing laboratory information systems. Moreover, prioritizing intuitive user interface design facilitates seamless adoption by laboratory technicians, pathologists, and other stakeholders, ensuring widespread acceptance and utilization of automated systems. By adhering to rigorous testing, validation, and regulatory compliance standards, we have strived to deliver reliable, clinically validated automation solutions that can seamlessly integrate into routine clinical practice.

Ultimately, this project represents a significant step forward in harnessing the full potential of technology to address the evolving needs of colorectal tissue histopathology. By automating labor-intensive tasks, standardizing diagnostic protocols, and augmenting pathologists' capabilities, we aim to improve cancer diagnostics, treatment planning, and patient outcomes. However, our journey does not end here; there are numerous opportunities for future exploration and enhancement in this field.

5.2. Future Scope:

Moving forward, the future scope of this project encompasses several key areas of development and innovation. Firstly, ongoing research and development efforts will focus on further refining machine learning algorithms for image analysis, with the goal of achieving even greater accuracy and specificity in detecting histopathological features associated with colorectal cancer. Continued optimization and validation of these algorithms will be essential for ensuring reliable and reproducible results in real-world clinical settings.

Furthermore, expanding the scope of automation to encompass other aspects of the histopathological workflow, such as specimen grossing, tissue microdissection, and molecular testing, holds significant potential for enhancing workflow efficiency and diagnostic capabilities. Integrating advanced automation solutions with emerging technologies such as telepathology and

artificial intelligence-driven decision support systems could further facilitate remote consultation, collaborative diagnosis, and personalized treatment planning, thereby improving patient care and outcomes.

Additionally, ongoing collaboration with clinicians, researchers, and industry partners will be essential for validating and implementing advanced automation solutions in diverse clinical settings. By leveraging multidisciplinary expertise and real-world clinical data, we can continue to refine and optimize automated systems to meet the evolving needs of pathology practice.

In conclusion, the future of advanced automation in colorectal tissue histopathology is bright, with immense potential to transform cancer diagnostics and patient care. By embracing innovation, collaboration, and continuous improvement, we can unlock new possibilities for improving diagnostic accuracy, treatment outcomes, and ultimately, the lives of patients affected by colorectal cancer.

1. **Refinement of Classification Accuracy**
   - Fine-Tuning and Transfer Learning: Implement fine-tuning techniques and transfer learning strategies to improve the accuracy and generalization of the CNN model. Transfer learning involves using pre-trained models on large datasets and fine-tuning them on the specific colorectal tissue dataset to leverage learned features.
   - Ensemble Learning: Explore ensemble learning methods to combine predictions from multiple CNN models, potentially boosting classification accuracy and robustness.

2. **Deep Learning Architectures**
   - Architectural Innovations: Investigate advanced CNN architectures such as DenseNet, ResNet, or EfficientNet, which are designed to handle complex relationships in images more effectively. These architectures may offer improvements in feature extraction and classification performance.
   - Attention Mechanisms: Incorporate attention mechanisms into the CNN architecture to focus on important regions of the tissue images, potentially enhancing the model's ability to discriminate between different tissue types.

3. **Multi-Modal Data Integration**
   - Integration with Clinical Data: Integrate histopathological image data with clinical metadata (e.g., patient demographics, medical history, treatment outcomes) to develop comprehensive predictive models that account for various factors influencing disease progression and response to treatment.
   - Fusion with Molecular Data: Combine histopathological images with molecular data (e.g., gene expression profiles, mutation data) to gain deeper insights into the molecular mechanisms underlying different tissue phenotypes and to identify novel biomarkers for diagnosis and prognosis.

4. **Clinical Translation and Deployment**
    - Regulatory Compliance: Ensure compliance with regulatory standards (e.g., FDA regulations for medical devices) and clinical guidelines for the deployment of CNN-based diagnostic tools in clinical practice.
    - Integration with Healthcare Systems: Integrate CNN-based tissue classification systems into existing healthcare infrastructure, such as electronic health record (EHR) systems, to streamline diagnostic workflows and facilitate seamless communication between pathologists and clinicians.

5. **Explainable AI and Model Interpretability**
    - Interpretability Techniques: Develop techniques for explaining the decisions made by the CNN model, providing insights into the features driving classification outcomes and increasing the transparency and trustworthiness of the model.
    - Clinical Validation Studies: Conduct prospective clinical validation studies to assess the performance and clinical utility of the CNN-based tissue classification system in real-world settings, demonstrating its effectiveness in improving diagnostic accuracy and patient outcomes.

6. **Global Accessibility and Equity**
    - Resource-Limited Settings: Adapt CNN-based tissue classification systems for use in resource-limited settings, where access to expert pathologists may be limited, by developing lightweight and portable solutions that can run on low-cost hardware or mobile devices.
    - Open-Source Initiatives: Contribute to open-source initiatives aimed at democratizing access to AI tools and resources for healthcare applications, fostering collaboration and knowledge sharing within the scientific community.

Conclusion: The future scope of a project focused on CNN-based colorectal tissue classification is multifaceted, encompassing advancements in model accuracy, architectural innovation, data integration, clinical translation, explainable AI, and global accessibility. By addressing these challenges and opportunities, the project can significantly impact the diagnosis, treatment, and management of colorectal diseases, ultimately improving patient outcomes and healthcare delivery worldwide.

5.3. USES:

The project focused on using Convolutional Neural Networks (CNNs) for colorectal tissue classification has numerous potential uses across various domains, including healthcare, research, and clinical practice. Here's a detailed exploration of the uses:

1. **Clinical Diagnosis and Prognosis**
   - Early Detection of Colorectal Cancer: The CNN model can assist pathologists in accurately identifying and classifying colorectal tissue abnormalities, including precancerous lesions and malignant tumors, at an early stage, enabling timely intervention and treatment.
   - Prognostic Assessment: By analyzing histopathological images and clinical data, the CNN model can predict patient outcomes, such as disease progression, recurrence, and overall survival, aiding clinicians in developing personalized treatment plans and follow-up strategies.
   - Decision Support System: Integrated into clinical workflows, the CNN model serves as a decision support tool for healthcare professionals, providing supplementary diagnostic information and aiding in differential diagnosis, particularly in challenging cases.

2. **Biomedical Research**
   - Disease Mechanisms: The CNN model can help researchers gain insights into the underlying mechanisms of colorectal diseases by identifying histopathological features associated with specific tissue phenotypes, molecular subtypes, or disease stages.
   - Biomarker Discovery: By analyzing large-scale histopathological image datasets, the CNN model can identify novel biomarkers indicative of disease progression, treatment response, or prognosis, facilitating the development of new diagnostic and therapeutic strategies.
   - Drug Development: The CNN model can be used to screen potential drug candidates and assess their efficacy in preclinical studies by analyzing their effects on histopathological tissue samples and identifying changes in tissue morphology or tumor characteristics.

3. **Public Health and Epidemiology**
   - Population Screening Programs: Deployed as part of population-based screening programs, the CNN model can automate the analysis of colorectal tissue samples, enabling efficient and scalable screening for early signs of colorectal cancer and other gastrointestinal diseases.
   - Disease Surveillance: By analyzing histopathological images from diverse patient populations, the CNN model can contribute to disease surveillance efforts, identifying trends, risk factors, and disparities in disease incidence and outcomes across different demographic groups.

4. **Educational and Training Tools**

- Medical Education: The CNN model can be incorporated into medical education curricula to train aspiring pathologists and clinicians in the interpretation of histopathological images, providing interactive and engaging learning experiences.
- Continuing Medical Education (CME): Used as part of continuing medical education programs, the CNN model can help practicing pathologists and clinicians stay updated on the latest developments in colorectal tissue classification and diagnostic algorithms.

5. Telemedicine and Remote Consultation

- Telepathology Services: Leveraging telepathology platforms, the CNN model enables remote consultation and second opinions, allowing pathologists to collaborate, share expertise, and provide diagnostic support across geographic locations.
- Access to Specialized Care: Particularly beneficial in underserved or remote areas, the CNN model ensures access to specialized pathology services and expertise, reducing disparities in healthcare access and improving patient outcomes.

Conclusion:

The project's CNN-based colorectal tissue classification system has wide-ranging applications in clinical practice, biomedical research, public health, medical education, and telemedicine. By harnessing the power of artificial intelligence and machine learning, the project contributes to advancing healthcare delivery, improving patient care, and addressing global health challenges related to colorectal diseases.

APPENDIX A

Vgg19:

```
# Import Modules

import os

import pandas as pd

import numpy as np

from itertools import chain

import cv2

import splitfolders


# Plotting

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import matplotlib.pyplot as plt

plt.style.use('ggplot')

import seaborn as sns


# Metrics

from sklearn.metrics import confusion_matrix, roc_curve,auc, classification_report


# Deep Learning

import tensorflow as tf

print(tf.__version__)

from tensorflow.keras import Model

from tensorflow.keras.layers import Flatten, Dense, Dropout, GlobalAveragePooling2D

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

from tensorflow.keras.callbacks import ModelCheckpoint

from tensorflow.keras.models import load_model

from tensorflow.keras.optimizers import SGD, Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
from tensorflow.keras.applications.vgg19 import VGG19

from tensorflow.keras.applications.vgg19 import preprocess_input as vgg19_preprocess_input

from tensorflow.keras.applications.resnet50 import ResNet50

from tensorflow.keras.applications.resnet50 import preprocess_input as resnet50_preprocess_input

from tensorflow.keras.applications.mobilenet import MobileNet

from tensorflow.keras.applications.mobilenet import preprocess_input as mobilenet_preprocess_input

from tensorflow.keras.preprocessing import image

from tensorflow.keras.preprocessing.image import img_to_array

from tensorflow.keras.preprocessing.image import array_to_img



base_dir = "E:\\Major Project"

input_ = os.path.join(base_dir,"CRC-VAL-HE-7K")


# split data into training, vlaidation and testing sets

#splitfolders.ratio(input_, 'output="output"', seed = 101, ratio=(0.8, 0.1, 0.1))


data_dir = os.path.join('E:\\Major Project','output')


# Define train, valid and test directories

train_dir = os.path.join(data_dir, 'train')

valid_dir = os.path.join(data_dir, 'val')

test_dir = os.path.join(data_dir, 'test')

os.listdir('output')


# Data Augmentation

train_datagen = ImageDataGenerator(rescale=1./255.,

                                   horizontal_flip=True,

                                   vertical_flip=True,
```

```
                                    shear_range=0.4,
                                    width_shift_range=0.25,
                                    height_shift_range=0.25,
                                    rotation_range=45,
                                    fill_mode='nearest')
valid_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)


# Train ImageDataGenerator
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size = 64,
                                                    target_size = (224,224),
                                                    class_mode                =
'categorical',
                                                    shuffle=True,
                                                    seed=42,
                                                    color_mode='rgb')

valid_generator = valid_datagen.flow_from_directory(valid_dir,
                                                    batch_size=64,
                                                    target_size=(224,224),
                                                    class_mode='categorical',
                                                    shuffle=True,
                                                    seed=42,
                                                    color_mode='rgb')

test_generator = test_datagen.flow_from_directory(test_dir,
                                                  batch_size=1,
                                                  target_size=(224,224),
                                                  class_mode='categorical',
                                                  shuffle=False,
```

```
                                                    seed=42,
                                                    color_mode='rgb')


# Instantiate the vgg19 model without the top classifier.
base_model = VGG19(input_shape=(224, 224, 3), weights='imagenet', include_top
= False)


# Add a classifier to the convolution block classifier
x = base_model.output
flat = Flatten()(x)
hidden_1 = Dense(1024, activation='relu')(flat)
drop_1 = Dropout(0.2)(hidden_1)
hidden_2 = Dense(512, activation='relu')(drop_1)
drop_2 = Dropout(0.3)(hidden_2)
output = Dense(9, activation= 'softmax')(drop_2)


# Define the model
model_vgg19_01 = Model(base_model.inputs, output)


# Freeze all layers in the convolution block. We don't want to train these
weights yet.
for layer in base_model.layers:
        layer.trainable=False


#model_vgg19_01.summary()


# Call Backs
filepath = 'vgg19_base_model_wt.keras'
es = EarlyStopping(monitor='loss', verbose=1, mode='min', patience=4)
cp       =       ModelCheckpoint(filepath,       monitor='val_loss',verbose=1,
save_best_only=True,
                    save_weights_only=False, mode='auto', save_freq='epoch')
```

```
# Compile a model

optimizer = Adam(learning_rate=0.0004)

model_vgg19_01.compile(loss="categorical_crossentropy",   optimizer=optimizer,
metrics=["accuracy"])


# Model fitting

history_01 = model_vgg19_01.fit(

            train_generator,

            steps_per_epoch=255,

            epochs=10,

            callbacks = [es, cp],

            validation_data = valid_generator)


# save model

if not os.path.isdir('model_weights/'):

    os.mkdir('model_weights/')

model_vgg19_01.save_weights(filepath='model_weights/vgg19_base_model_wt.weigh
ts.h5', overwrite=True)



# Load the saved model

model_vgg19_01.load_weights('vgg19_base_model_wt.keras')

# Evaluate the model on the hold out validation and test datasets


vgg_val_eval_01 = model_vgg19_01.evaluate(valid_generator)

vgg_test_eval_01 = model_vgg19_01.evaluate(test_generator)


print('Validation loss:     {}'.format(vgg_val_eval_01[0]))

print('Validation accuracy: {}'.format(vgg_val_eval_01[1]))

print('Test loss:           {}'.format(vgg_test_eval_01[0]))

print('Test accuracy:       {}'.format(vgg_test_eval_01[1]))
```

```
# Predict probabilities

nb_samples = len(test_generator)

vgg_predictions_01     =     model_vgg19_01.predict(test_generator,steps     =
nb_samples,verbose=1)


# Predict labels

vgg_pred_labels_01 = np.argmax(vgg_predictions_01, axis=1)


# Classification Report

print('|'+'-'*75+'|')

print('|-------------------Classification Report: Training Cycle #1-----------
-----|')

print('|'+'-'*75+'|')

print(classification_report(test_generator.classes, vgg_pred_labels_01,

                            target_names=['ADI', 'BACK', 'DEB', 'LYM', 'MUC',
'MUS', 'NORM', 'STR', 'TUM']))


# Plot performance of vgg19 base model

fig, (ax1,ax2) = plt.subplots(nrows=1, ncols=2, figsize=(12,6))

fig.suptitle("VGG19 Base Model Training", fontsize=20)

max_epoch = len(history_01.history['acc'])+1

epochs_list = list(range(1, max_epoch))


ax1.plot(epochs_list,  history_01.history['acc'],  color='b',  linestyle='-',
label='Training Data')

ax1.plot(epochs_list, history_01.history['val_acc'], color='r', linestyle='-',
label ='Validation Data')

ax1.set_title('Training Accuracy', fontsize=14)

ax1.set_xlabel('Epochs', fontsize=14)

ax1.set_ylabel('Accuracy', fontsize=14)

ax1.legend(frameon=False, loc='lower center', ncol=2)


ax2.plot(epochs_list,  history_01.history['loss'],  color='b',  linestyle='-',
label='Training Data')
```

```
ax2.plot(epochs_list, history_01.history['val_loss'], color='r', linestyle='-
', label ='Validation Data')

ax2.set_title('Training Loss', fontsize=14)

ax2.set_xlabel('Epochs', fontsize=14)

ax2.set_ylabel('Loss', fontsize=14)

ax2.legend(frameon=False, loc='upper center', ncol=2)



# Construct VGG19 model without the classifer and weights trained on imagenet
data

base_model_02 = VGG19(input_shape=(224, 224, 3),

                  include_top = False)

x = base_model_02.output

flat = Flatten()(x)

hidden_1 = Dense(1024, activation='relu')(flat)

drop_1 = Dropout(0.2)(hidden_1)

hidden_2 = Dense(512, activation='relu')(drop_1)

drop_2 = Dropout(0.3)(hidden_2)

output = Dense(9, activation= 'softmax')(drop_2)


model_vgg19_02 = Model(base_model_02.inputs, output)

# Load weights from the trained base model

model_vgg19_02.load_weights('vgg19_base_model_wt.keras')



# Freeze layers upto the 19th layer.

for layer in model_vgg19_02.layers[:19]:

    layer.trainable =  False


# Call Backs

filepath = 'vgg19_model_wt_ft_01.keras'

es = EarlyStopping(monitor='loss', verbose=1, mode='min', patience=4)
```

```
cp       =       ModelCheckpoint(filepath,       monitor='val_loss',verbose=1,
save_best_only=True,
                    save_weights_only=False, mode='auto', save_freq='epoch')


#print(model_vgg19_02.summary())

optimizer = Adam(learning_rate=0.0004)

model_vgg19_02.compile(loss="categorical_crossentropy",   optimizer=optimizer,
metrics=["accuracy"])



# Model fitting

history_02 = model_vgg19_02.fit(

            train_generator,

            steps_per_epoch=225,

            epochs=10,

            callbacks = [es, cp],

            validation_data = valid_generator)



# save model

if not os.path.isdir('model_weights/'):

    os.mkdir('model_weights/')

model_vgg19_02.save_weights(filepath='model_weights/vgg19_model_wt_ft_01.weig
hts.h5', overwrite=True)


# Plot performance of vgg19 finetune model

fig, (ax1,ax2) = plt.subplots(nrows=1, ncols=2, figsize=(12,6))

fig.suptitle("VGG19 Fine Tuning (Unfreeze 2 Conv Layers)", fontsize=20)

max_epoch = len(history_02.history['acc'])+1

epochs_list = list(range(1, max_epoch))


ax1.plot(epochs_list,  history_02.history['acc'],  color='b',  linestyle='-',
label='Training Data')
```

```
ax1.plot(epochs_list, history_02.history['val_acc'], color='r', linestyle='-',
label ='Validation Data')

ax1.set_title('Training Accuracy', fontsize=14)

ax1.set_xlabel('Epochs', fontsize=14)

ax1.set_ylabel('Accuracy', fontsize=14)

ax1.legend(frameon=False, loc='lower center', ncol=2)


ax2.plot(epochs_list,  history_02.history['loss'],  color='b',  linestyle='-',
label='Training Data')

ax2.plot(epochs_list, history_02.history['val_loss'], color='r', linestyle='-
', label ='Validation Data')

ax2.set_title('Training Loss', fontsize=14)

ax2.set_xlabel('Epochs', fontsize=14)

ax2.set_ylabel('Loss', fontsize=14)

ax2.legend(frameon=False, loc='upper center', ncol=2)



# Load the saved model

model_vgg19_02.load_weights('vgg19_model_wt_ft_01.keras')

# Evaluate the model on the hold out validation and test datasets


vgg_val_eval_02 = model_vgg19_02.evaluate(valid_generator)

vgg_test_eval_02 = model_vgg19_02.evaluate(test_generator)


print('Validation loss:     {0:.3f}'.format(vgg_val_eval_02[0]))

print('Validation accuracy: {0:.3f}'.format(vgg_val_eval_02[1]))

print('Test loss:           {0:.3f}'.format(vgg_test_eval_02[0]))

print('Test accuracy:       {0:.3f}'.format(vgg_test_eval_02[1]))


# Predict probabilities

nb_samples = len(test_generator)

vgg_predictions_02     =     model_vgg19_02.predict(test_generator,steps     =
nb_samples,verbose=1)
```

```
# Predict labels
vgg_pred_labels_02 = np.argmax(vgg_predictions_02, axis=1)


# Classification Report
print('|'+'-'*67+'|')
print('|------------Classification Report: VGG19 Training Cycle #2-----------
|')
print('|'+'-'*67+'|')
print(classification_report(test_generator.classes, vgg_pred_labels_02,
                            target_names=['ADI', 'BACK', 'DEB', 'LYM', 'MUC',
'MUS', 'NORM', 'STR', 'TUM']))



# Build VGG19
base_model_03 = VGG19(input_shape=(224, 224, 3),
                 include_top = False)
x = base_model_03.output
flat = Flatten()(x)
hidden_1 = Dense(1024, activation='relu')(flat)
drop_1 = Dropout(0.2)(hidden_1)
hidden_2 = Dense(512, activation='relu')(drop_1)
drop_2 = Dropout(0.3)(hidden_2)
output = Dense(9, activation= 'softmax')(drop_2)


model_vgg19_03 = Model(base_model_03.inputs, output)
# Load the weights saved after the first round of fine tuning
model_vgg19_03.load_weights('vgg19_model_wt_ft_01.keras')


# Callbacks
filepath = 'vgg19_model_finetuned.keras'


cp       =       ModelCheckpoint(filepath,       monitor='val_loss',verbose=1,
save_best_only=True,
```

```
                    save_weights_only=False, mode='auto', save_freq='epoch')


learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.0001)


optimizer = Adam(learning_rate=0.0004)
model_vgg19_03.compile(loss="categorical_crossentropy",   optimizer=optimizer,
metrics=["accuracy"])




# Model Fitting
history_03 = model_vgg19_03.fit(
            train_generator,
            steps_per_epoch=225,
            epochs=25,
            callbacks = [learning_rate_reduction, cp],
            validation_data = valid_generator)



# save model
if not os.path.isdir('model_weights/'):
    os.mkdir('model_weights/')
model_vgg19_03.save_weights(filepath='model_weights/vgg19_model_finetuned.wei
ghts.h5', overwrite=True)
model_vgg19_03.save("vgg19_model_finetuned.keras")



# Load the saved model
model_vgg19_03.load_weights('vgg19_model_finetuned.keras')
```

```
# Evaluate the model on the hold out validation and test datasets


vgg_val_eval_03 = model_vgg19_03.evaluate(valid_generator)

vgg_test_eval_03 = model_vgg19_03.evaluate(test_generator)


print('Validation loss:       {0:.3f}'.format(vgg_val_eval_03[0]))

print('Validation accuracy:   {0:.3f}'.format(vgg_val_eval_03[1]))

print('Test loss:             {0:.3f}'.format(vgg_test_eval_03[0]))

print('Test accuracy:         {0:.3f}'.format(vgg_test_eval_03[1]))


# Predict probabilities

nb_samples = len(test_generator)

vgg_predictions_03  =  model_vgg19_03.predict_generator(test_generator,steps  =
nb_samples, verbose=1)

# Predict labels

vgg_pred_labels_03 = np.argmax(vgg_predictions_03, axis=1)


# Classification Report

print('|'+'-'*67+'|')

print('|------------Classification Report: VGG19 Training Cycle #3-----------
|')

print('|'+'-'*67+'|')

print(classification_report(test_generator.classes, vgg_pred_labels_03,

                            target_names=['ADI', 'BACK', 'DEB', 'LYM', 'MUC',
'MUS', 'NORM', 'STR', 'TUM']))




# Confusion Matrices

vgg_conf_mat_01    =    pd.DataFrame(confusion_matrix(test_generator.classes,
vgg_pred_labels_01),

                        index=['ADI',  'BACK',  'DEB',  'LYM',  'MUC',  'MUS',
'NORM', 'STR', 'TUM'],
```

```
                        columns=['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS',
'NORM', 'STR', 'TUM'])



vgg_conf_mat_02    =    pd.DataFrame(confusion_matrix(test_generator.classes,
vgg_pred_labels_02),
                        index=['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS',
'NORM', 'STR', 'TUM'],
                        columns=['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS',
'NORM', 'STR', 'TUM'])



vgg_conf_mat_03    =    pd.DataFrame(confusion_matrix(test_generator.classes,
vgg_pred_labels_03),
                        index=['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS',
'NORM', 'STR', 'TUM'],
                        columns=['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS',
'NORM', 'STR', 'TUM'])


# Plotting Confusion Matrices

sns.set(font_scale=1.2)

fig, (ax1,ax2, ax3) = plt.subplots(nrows=3, ncols=1, figsize=(7,35))


#ax1

sns.heatmap(vgg_conf_mat_01, ax=ax1, annot=True, fmt=".1f", linewidths=0.5,
square=True, cmap='Blues_r', annot_kws={"size": 10})

ax1.set_ylabel("Actual Label", fontsize=20)

ax1.set_xlabel("Predicted Label", fontsize=20)

all_sample_title="VGG19    Training    Round    #1    \nAccuracy    Score:
{0:.3f}".format(vgg_test_eval_01[1])

ax1.set_title(all_sample_title, size=24)

ax1.set_ylim(len(vgg_conf_mat_01)-0.1, -0.1)



#ax2
```

```
sns.heatmap(vgg_conf_mat_02, ax=ax2, annot=True, fmt=".1f", linewidths=0.5, square=True, cmap='Blues_r', annot_kws={"size": 10})

ax2.set_ylabel("Actual Label", fontsize=20)

ax2.set_xlabel("Predicted Label", fontsize=20)

all_sample_title="VGG19 Training Round #2 \nAccuracy Score: {0:.3f}".format(vgg_test_eval_02[1])

ax2.set_title(all_sample_title, size=24)

ax2.set_ylim(len(vgg_conf_mat_02)-0.1, -0.1)


#ax3

sns.heatmap(vgg_conf_mat_03, ax=ax3, annot=True, fmt=".1f", linewidths=0.5, square=True, cmap='Blues_r', annot_kws={"size": 10})

ax3.set_ylabel("Actual Label", fontsize=20)

ax3.set_xlabel("Predicted Label", fontsize=20)

all_sample_title="VGG19 Training Round#3 \nAccuracy Score: {0:.3f}".format(vgg_test_eval_03[1])

ax3.set_title(all_sample_title, size=24)

ax3.set_ylim(len(vgg_conf_mat_03)-0.1, -0.1)




from itertools import chain


# training_accuracy

training_accuracy = []

training_accuracy.append(history_01.history['acc'])

training_accuracy.append(history_02.history['acc'])

training_accuracy.append(history_03.history['acc'])

training_accuracy_ = list(itertools.chain(*training_accuracy))


# training_loss

training_loss = []

training_loss.append(history_01.history['loss'])

training_loss.append(history_02.history['loss'])
```

```
training_loss.append(history_03.history['loss'])
training_loss_ = list(itertools.chain(*training_loss))

# validation_accuracy
validation_accuracy = []
validation_accuracy.append(history_01.history['val_acc'])
validation_accuracy.append(history_02.history['val_acc'])
validation_accuracy.append(history_03.history['val_acc'])
validation_accuracy_ = list(itertools.chain(*validation_accuracy))

# validation_loss
validation_loss = []
validation_loss.append(history_01.history['val_loss'])
validation_loss.append(history_02.history['val_loss'])
validation_loss.append(history_03.history['val_loss'])
validation_loss_ = list(itertools.chain(*validation_loss))



training_metrics_df = pd.DataFrame({'training_accuracy': training_accuracy_,
                                    'training_loss': training_loss_,
                                    'validation_accuracy':
validation_accuracy_,
                                    'validation_loss': validation_loss_})
training_metrics_df.to_csv('training_metrics_03.csv', index=False)
# Plot performance of vgg19 finetune model
fig, (ax1,ax2) = plt.subplots(nrows=1, ncols=2, figsize=(12,6))
fig.suptitle("VGG19 Fine Tuning (Unfreeze All Layers)", fontsize=20)
max_epoch = len(history_03.history['acc'])+1
epochs_list = list(range(1, max_epoch))

ax1.plot(epochs_list,  history_03.history['acc'],  color='b',  linestyle='-',
label='Training Data')
```

```
ax1.plot(epochs_list, history_03.history['val_acc'], color='r', linestyle='-',
label ='Validation Data')

ax1.set_title('Training Accuracy', fontsize=14)

ax1.set_xlabel('Epochs', fontsize=14)

ax1.set_ylabel('Accuracy', fontsize=14)

ax1.legend(frameon=False, loc='lower center', ncol=2)


ax2.plot(epochs_list,  history_03.history['loss'],  color='b',  linestyle='-',
label='Training Data')

ax2.plot(epochs_list, history_03.history['val_loss'], color='r', linestyle='-
', label ='Validation Data')

ax2.set_title('Training Loss', fontsize=14)

ax2.set_xlabel('Epochs', fontsize=14)

ax2.set_ylabel('Loss', fontsize=14)

ax2.legend(frameon=False, loc='upper center', ncol=2)
```

ResNet-50:

```
# Import Modules

import os

import pandas as pd

import numpy as np

from itertools import chain

import cv2

import splitfolders


# Plotting

#%matplotlib inline

import matplotlib.pyplot as plt

plt.style.use('ggplot')

import matplotlib.image as mpimg

import seaborn as sns


# Metrics

from sklearn.metrics import confusion_matrix, roc_curve,auc, classification_report


# Deep Learning

import tensorflow as tf

print(tf.__version__)

from tensorflow.keras import Model

from tensorflow.keras.layers import Flatten, Dense, Dropout, GlobalAveragePooling2D

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

from tensorflow.keras.callbacks import ModelCheckpoint

from tensorflow.keras.models import load_model

from tensorflow.keras.optimizers import SGD, Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications.vgg19 import VGG19
```

```
from tensorflow.keras.applications.vgg19 import preprocess_input as vgg19_preprocess_input

from tensorflow.keras.applications.resnet50 import ResNet50

from tensorflow.keras.applications.resnet50 import preprocess_input as resnet50_preprocess_input

from tensorflow.keras.applications.mobilenet import MobileNet

from tensorflow.keras.applications.mobilenet import preprocess_input as mobilenet_preprocess_input

from tensorflow.keras.preprocessing import image

from tensorflow.keras.preprocessing.image import img_to_array

from tensorflow.keras.preprocessing.image import array_to_img


base_dir = "E:\\Major Project"

input_ = os.path.join(base_dir,"CRC-VAL-HE-7K")


# split data into training, vlaidation and testing sets

#splitfolders.ratio(input_, output="output", seed = 101, ratio=(0.8, 0.1, 0.1))


data_dir = os.path.join('E:\\Major Project','output')


# Define train, valid and test directories

train_dir = os.path.join(data_dir, 'train')

valid_dir = os.path.join(data_dir, 'val')

test_dir = os.path.join(data_dir, 'test')

os.listdir('output')


# Data Augmentation

train_datagen = ImageDataGenerator(rescale=1./255.,
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                   shear_range=0.4,
                                   width_shift_range=0.25,
                                   height_shift_range=0.25,
```

```
                                    rotation_range=45,
                                    fill_mode='nearest')
valid_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

# Train ImageDataGenerator
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size = 64,
                                                    target_size = (224,224),
                                                    class_mode                =
'categorical',
                                                    shuffle=True,
                                                    seed=42,
                                                    color_mode='rgb')

valid_generator = valid_datagen.flow_from_directory(valid_dir,
                                                    batch_size=64,
                                                    target_size=(224,224),
                                                    class_mode='categorical',
                                                    shuffle=True,
                                                    seed=42,
                                                    color_mode='rgb')

test_generator = test_datagen.flow_from_directory(test_dir,
                                                  batch_size=1,
                                                  target_size=(224,224),
                                                  class_mode='categorical',
                                                  shuffle=False,
                                                  seed=42,
                                                  color_mode='rgb')

base_model_resnet50 = ResNet50(input_shape=(224,224,3),
```

```
                                    include_top=False,
                                    weights='imagenet')

x = base_model_resnet50.output
flat = Flatten()(x)
hidden_1 = Dense(1024, activation='relu')(flat)
drop_1 = Dropout(0.2)(hidden_1)
hidden_2 = Dense(512, activation='relu')(drop_1)
drop_2 = Dropout(0.3)(hidden_2)
output = Dense(9, activation= 'softmax')(drop_2)

model_resnet50_01 = Model(base_model_resnet50.inputs, output)

# Call Backs
filepath = 'resent50_base_model_wt.keras'
es = EarlyStopping(monitor='loss', verbose=1, mode='min', patience=4)
cp       =       ModelCheckpoint(filepath,       monitor='val_loss',verbose=1,
save_best_only=True,
                     save_weights_only=False, mode='auto', save_freq='epoch')
# compile the model
optimizer = Adam(learning_rate=0.0004)

# Compile the model using the optimizer
model_resnet50_01.compile(loss="categorical_crossentropy",
optimizer=optimizer, metrics=["accuracy"])
"""
resnet50_history_01 = model_resnet50_01.fit(train_generator,
                                            steps_per_epoch=
int(len(input_)/64),
                                            epochs=10,
                                            callbacks = [es, cp],
                                        validation_data = valid_generator)
```

```
# save model
if not os.path.isdir('model_weights/'):
    os.mkdir('model_weights/')
model_resnet50_01.save_weights(filepath='resent50_base_model_wt.weights.h5',
overwrite=True)

"""

# Load the saved model
model_resnet50_01.load_weights('resent50_base_model_wt.keras')
# Evaluate the model on the hold out validation and test datasets

res_val_eval_01 = model_resnet50_01.evaluate(valid_generator)
res_test_eval_01 = model_resnet50_01.evaluate(test_generator)

print('Validation loss:      {}'.format(res_val_eval_01[0]))
print('Validation accuracy:  {}'.format(res_val_eval_01[1]))
print('Test loss:            {}'.format(res_test_eval_01[0]))
print('Test accuracy:        {}'.format(res_test_eval_01[1]))

# Predict probabilities
nb_samples = len(test_generator)
res_predictions_01    =    model_resnet50_01.predict(test_generator,steps    =
nb_samples,verbose=1)
# Predict labels
res_pred_labels_01 = np.argmax(res_predictions_01, axis=1)

# Classification Report
print('|'+'-'*67+'|')
print('|-------Classification Report: ReseNet50 Training Cycle #1----------|')
print('|'+'-'*67+'|')
print(classification_report(test_generator.classes, res_pred_labels_01,
```

```
                            target_names=['ADI', 'BACK', 'DEB', 'LYM', 'MUC',
'MUS', 'NORM', 'STR', 'TUM']))


# Traning cycle 1
base_model_resnet50_02 = ResNet50(input_shape=(224, 224, 3),
                                                  include_top = False)
x = base_model_resnet50_02.output
flat = Flatten()(x)
hidden_1 = Dense(1024, activation='relu')(flat)
drop_1 = Dropout(0.2)(hidden_1)
hidden_2 = Dense(512, activation='relu')(drop_1)
drop_2 = Dropout(0.3)(hidden_2)
output = Dense(9, activation= 'softmax')(drop_2)


model_resnet50_02 = Model(base_model_resnet50_02.inputs, output)
model_resnet50_02.load_weights('resent50_base_model_wt.keras')


for layer in model_resnet50_02.layers[:160]:
    layer.trainable= False


# Call Backs
filepath_02 = 'resent50_model_02_wt.keras'
es_02 = EarlyStopping(monitor='loss', verbose=1, mode='min', patience=4)
cp_02     =     ModelCheckpoint(filepath_02,     monitor='val_loss',verbose=1,
save_best_only=True,
                    save_weights_only=False, mode='auto', save_freq='epoch')


optimizer = Adam(learning_rate=0.0004)


# Compile the model using the optimizer
model_resnet50_02.compile(loss="categorical_crossentropy",
optimizer=optimizer, metrics=["accuracy"])
```

```
resnet50_history_02 = model_resnet50_02.fit(train_generator,
                                                    steps_per_epoch=
int(len(input_)/64,
                                                    epochs=10,
                                                    callbacks    =    [es_02,
cp_02],
                                                    validation_data        =
valid_generator)


# save model
if not os.path.isdir('model_weights/'):
    os.mkdir('model_weights/')
model_resnet50_02.save_weights(filepath='resent50_model_02_wt.weights.h5',
overwrite=True)


# Load the saved model
model_resnet50_02.load_weights('resent50_model_02_wt.keras')
# Evaluate the model on the hold out validation and test datasets


res_val_eval_02 = model_resnet50_02.evaluate(valid_generator)
res_test_eval_02 = model_resnet50_02.evaluate(test_generator)


print('Validation loss:     {}'.format(res_val_eval_02[0]))
print('Validation accuracy: {}'.format(res_val_eval_02[1]))
print('Test loss:           {}'.format(res_test_eval_02[0]))
print('Test accuracy:       {}'.format(res_test_eval_02[1]))
print('*'*75)


# Predict probabilities
nb_samples = len(test_generator)
res_predictions_02   =   model_resnet50_02.predict(test_generator,   steps   =
nb_samples,verbose=1)
# Predict labels
res_pred_labels_02 = np.argmax(res_predictions_02, axis=1)
```

```
# Classification Report
print('|'+'-'*67+'|')
print('|-------Classification Report: ReseNet50 Training Cycle #2----------|')
print('|'+'-'*67+'|')
print(classification_report(test_generator.classes, res_pred_labels_02,
                            target_names=['ADI', 'BACK', 'DEB', 'LYM', 'MUC',
'MUS', 'NORM', 'STR', 'TUM']))


# Traning cycle 2
base_model_resnet50_03 = ResNet50(input_shape=(224, 224, 3),
                                                  include_top = False)
x = base_model_resnet50_03.output
flat = Flatten()(x)
hidden_1 = Dense(1024, activation='relu')(flat)
drop_1 = Dropout(0.2)(hidden_1)
hidden_2 = Dense(512, activation='relu')(drop_1)
drop_2 = Dropout(0.3)(hidden_2)
output = Dense(9, activation= 'softmax')(drop_2)

model_resnet50_03 = Model(base_model_resnet50_03.inputs, output)
model_resnet50_03.load_weights('resent50_model_02_wt.keras')

for layer in model_resnet50_03.layers[:118]:
    layer.trainable= False

# Call Backs
filepath_03 = 'resent50_model_03_wt.keras'
es_03 = EarlyStopping(monitor='loss', verbose=1, mode='min', patience=4)
cp_03     =     ModelCheckpoint(filepath_03,     monitor='val_loss',verbose=1,
save_best_only=True,
                    save_weights_only=False, mode='auto', save_freq='epoch')
```

```
optimizer = Adam(learning_rate=0.0004)


# Compile the model using the optimizer
model_resnet50_03.compile(loss="categorical_crossentropy",
optimizer=optimizer, metrics=["accuracy"])


resnet50_history_03 = model_resnet50_03.fit(train_generator,
                                                    steps_per_epoch=
int(len(input_)/64,
                                                    epochs=10,
                                                    callbacks    =    [es_02,
cp_02],
                                                    validation_data        =
valid_generator)
# save model
if not os.path.isdir('model_weights/'):
    os.mkdir('model_weights/')
model_resnet50_03.save_weights(filepath='resent50_model_03_wt.weights.h5',
overwrite=True)


# Load the saved model
model_resnet50_03.load_weights('resent50_model_03_wt.keras')
# Evaluate the model on the hold out validation and test datasets


res_val_eval_03 = model_resnet50_03.evaluate(valid_generator)
res_test_eval_03 = model_resnet50_03.evaluate(test_generator)


print('Validation loss:     {0:.3f}'.format(res_val_eval_03[0]))
print('Validation accuracy: {0:.3f}'.format(res_val_eval_03[1]))
print('Test loss:           {0:.3f}'.format(res_test_eval_03[0]))
print('Test accuracy:       {0:.3f}'.format(res_test_eval_03[1]))


# Predict probabilities
nb_samples = len(test_generator)
```

```
res_predictions_03    =    model_resnet50_03.predict(test_generator,steps    =
nb_samples,verbose=1)

# Predict labels

res_pred_labels_03 = np.argmax(res_predictions_03, axis=1)


# Classification Report

print('|'+'-'*67+'|')

print('|-------Classification Report: ReseNet50 Training Cycle #3----------|')

print('|'+'-'*67+'|')

print(classification_report(test_generator.classes, res_pred_labels_03,

                            target_names=['ADI', 'BACK', 'DEB', 'LYM', 'MUC',
'MUS', 'NORM', 'STR', 'TUM']))


#Fine tuning Resnet50 model training cycle 4


base_model_resnet50_04 = ResNet50(input_shape=(224, 224, 3),

                                                  include_top = False)

x = base_model_resnet50_04.output

flat = Flatten()(x)

hidden_1 = Dense(1024, activation='relu')(flat)

drop_1 = Dropout(0.2)(hidden_1)

hidden_2 = Dense(512, activation='relu')(drop_1)

drop_2 = Dropout(0.3)(hidden_2)

output = Dense(9, activation= 'softmax')(drop_2)


model_resnet50_04 = Model(base_model_resnet50_04.inputs, output)

model_resnet50_04.load_weights('resent50_model_03_wt.h5')


# Call Backs

filepath_04 = 'resent50_model_04_wt.h5'

es_04 = EarlyStopping(monitor='loss', verbose=1, mode='min', patience=4)

cp_04     =     ModelCheckpoint(filepath_04,     monitor='val_loss',verbose=1,
save_best_only=True,
```

```
                    save_weights_only=False, mode='auto', save_freq='epoch')


optimizer = Adam(learning_rate=0.0004)

model_resnet50_04.compile(loss="categorical_crossentropy",
optimizer=optimizer, metrics=["accuracy"])

resnet50_history_04 = model_resnet50_04.fit_generator(train_generator,
                                                      steps_per_epoch=
int(len(input_)/64,
                                                      epochs=30,
                                                      callbacks    =    [es_04,
cp_04],
                                                      validation_data        =
valid_generator)


# Confusion Matrices

res_conf_mat_01    =    pd.DataFrame(confusion_matrix(test_generator.classes,
res_pred_labels_01),
                        index=['ADI',  'BACK',  'DEB',  'LYM',  'MUC',  'MUS',
'NORM', 'STR', 'TUM'],
                        columns=['ADI',  'BACK',  'DEB',  'LYM',  'MUC',  'MUS',
'NORM', 'STR', 'TUM'])


res_conf_mat_02    =    pd.DataFrame(confusion_matrix(test_generator.classes,
res_pred_labels_02),
                        index=['ADI',  'BACK',  'DEB',  'LYM',  'MUC',  'MUS',
'NORM', 'STR', 'TUM'],
                        columns=['ADI',  'BACK',  'DEB',  'LYM',  'MUC',  'MUS',
'NORM', 'STR', 'TUM'])


res_conf_mat_03    =    pd.DataFrame(confusion_matrix(test_generator.classes,
res_pred_labels_03),
                        index=['ADI',  'BACK',  'DEB',  'LYM',  'MUC',  'MUS',
'NORM', 'STR', 'TUM'],
                        columns=['ADI',  'BACK',  'DEB',  'LYM',  'MUC',  'MUS',
'NORM', 'STR', 'TUM'])


res_conf_mat_04    =    pd.DataFrame(confusion_matrix(test_generator.classes,
res_pred_labels_04),
```

```
                        index=['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS',
'NORM', 'STR', 'TUM'],
                        columns=['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS',
'NORM', 'STR', 'TUM'])


# Plot Confusion Matrices

sns.set(font_scale=1.8)

fig, ([ax1,ax2],[ax3,ax4]) = plt.subplots(nrows=2, ncols=2, figsize=(28,28))


#ax1

sns.heatmap(res_conf_mat_01, ax=ax1, annot=True, fmt=".1f", linewidths=0.5,
square=True, cmap='Blues_r', annot_kws={"size": 18})

ax1.set_ylabel("Actual Label", fontsize=24)

ax1.set_xlabel("Predicted Label", fontsize=24)

all_sample_title="ResNet50 Training Round #1 \nAccuracy Score:
{0:.3f}".format(res_test_eval_01[1])

ax1.set_title(all_sample_title, size=32)

ax1.set_ylim(len(res_conf_mat_01)-0.1, -0.1)


#ax2

sns.heatmap(res_conf_mat_02, ax=ax2, annot=True, fmt=".1f", linewidths=0.5,
square=True, cmap='Blues_r', annot_kws={"size": 18})

ax2.set_ylabel("Actual Label", fontsize=24)

ax2.set_xlabel("Predicted Label", fontsize=24)

all_sample_title="ResNet50 Training Round #2 \nAccuracy Score:
{0:.3f}".format(res_test_eval_02[1])

ax2.set_title(all_sample_title, size=32)

ax2.set_ylim(len(res_conf_mat_02)-0.1, -0.1)


#ax3

sns.heatmap(res_conf_mat_03, ax=ax3, annot=True, fmt=".1f", linewidths=0.5,
square=True, cmap='Blues_r', annot_kws={"size": 18})

ax3.set_ylabel("Actual Label", fontsize=24)

ax3.set_xlabel("Predicted Label", fontsize=24)
```

```
all_sample_title="ResNet50 Training Round#3 \nAccuracy Score: {0:.3f}".format(res_test_eval_03[1])

ax3.set_title(all_sample_title, size=32)

ax3.set_ylim(len(res_conf_mat_03)-0.1, -0.1)


#ax4

sns.heatmap(res_conf_mat_04, ax=ax4, annot=True, fmt=".1f", linewidths=0.5, square=True, cmap='Blues_r', annot_kws={"size": 18})

ax4.set_ylabel("Actual Label", fontsize=24)

ax4.set_xlabel("Predicted Label", fontsize=24)

all_sample_title="ResNet50 Training Round#4 \nAccuracy Score: {0:.3f}".format(res_test_eval_04[1])

ax4.set_title(all_sample_title, size=32)

ax4.set_ylim(len(res_conf_mat_04)-0.1, -0.1)


plt.tight_layout()


# Test data

filenames = test_generator.filenames

nb_samples = len(filenames)


class_labels = test_generator.class_indices

class_names = {value:key for key,value in class_labels.items()}


labels = (ind_test_generator.class_indices)

labels = dict((v,k) for k,v in labels.items())

predictions = [labels[k] for k in res_pred_labels_04]


filenames = test_generator.filenames

results = pd.DataFrame({"Filename":filenames,

                        "Predictions":predictions})


random_files = results.sample(36)
```

```
filenames = random_files['Filename'].tolist()

predicted_labels = random_files['Predictions'].tolist()

test_file_paths = ['E:\\Major Project/output/test/'+ filename for filename in
filenames]

# Tissue types dictionary mapping class names with full names

tissue_types = {"ADI": "Adipose tissue",

                "BACK": "Background",

                "DEB": "Debris",

                "LYM": "Lymphocyte aggregates",

                "MUC": "Mucus",

                "MUS": "Muscle",

                "NORM": "Normal mucosa",

                "STR": "Stroma",

                "TUM": "Tumor epithelium"}


fig = plt.figure(figsize=(45,60))

fig.subplots_adjust(top=0.88)

columns = 4

rows = 9


for i in range(1, columns*rows+1):

    fig.add_subplot(rows, columns, i)

    plt.imshow(mpimg.imread(test_file_paths[i-1]))

    plt.axis('off')

    true_label = test_file_paths[i-1].split('/')[-2]

    predicted_label = predicted_labels[i-1]

    plt.title("True           Label:           {}\nPredicted           Label:
{}".format(tissue_types[true_label],          tissue_types[predicted_label]),
fontsize=28)

plt.tight_layout()


plt.show()
```

MobileNet

```
import os

import pandas as pd

import numpy as np

from itertools import chain

import cv2

import splitfolders


# Plotting

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

#%matplotlib inline

plt.style.use('ggplot')

import seaborn as sns


# Metrics

from sklearn.metrics import confusion_matrix, roc_curve,auc, classification_report


# Deep Learning

import tensorflow as tf

print(tf.__version__)

from tensorflow.keras import Model

from tensorflow.keras.layers import Flatten, Dense, Dropout, GlobalAveragePooling2D

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

from tensorflow.keras.callbacks import ModelCheckpoint

from tensorflow.keras.models import load_model

from tensorflow.keras.optimizers import SGD, Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications.vgg19 import VGG19

from tensorflow.keras.applications.vgg19 import preprocess_input as vgg19_preprocess_input
```

```
from tensorflow.keras.applications.resnet50 import ResNet50

from tensorflow.keras.applications.resnet50 import preprocess_input as resnet50_preprocess_input

from tensorflow.keras.applications.mobilenet import MobileNet

from tensorflow.keras.applications.mobilenet import preprocess_input as mobilenet_preprocess_input

from tensorflow.keras.preprocessing import image

from tensorflow.keras.preprocessing.image import img_to_array

from tensorflow.keras.preprocessing.image import array_to_img


base_dir = "E:\\Major Project"

input_ = os.path.join(base_dir,"CRC-VAL-HE-7K")


# split data into training, vlaidation and testing sets

#splitfolders.ratio(input_, output="output", seed = 101, ratio=(0.8, 0.1, 0.1))


data_dir = os.path.join('E:\\Major Project','output')


# Define train, valid and test directories

train_dir = os.path.join(data_dir, 'train')

valid_dir = os.path.join(data_dir, 'val')

test_dir = os.path.join(data_dir, 'test')

os.listdir('output')



# Data Augmentation

train_datagen = ImageDataGenerator(rescale=1./255.,

                                   horizontal_flip=True,

                                   vertical_flip=True,

                                   shear_range=0.4,

                                   width_shift_range=0.25,

                                   height_shift_range=0.25,

                                   rotation_range=45,
```

```
                                    fill_mode='nearest')
valid_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
# Train ImageDataGenerator
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size = 64,
                                                    target_size = (224,224),
                                                    class_mode                =
'categorical',
                                                    shuffle=True,
                                                    seed=42,
                                                    color_mode='rgb')

valid_generator = valid_datagen.flow_from_directory(valid_dir,
                                                    batch_size=64,
                                                    target_size=(224,224),
                                                    class_mode='categorical',
                                                    shuffle=True,
                                                    seed=42,
                                                    color_mode='rgb')

test_generator = test_datagen.flow_from_directory(test_dir,
                                                  batch_size=1,
                                                  target_size=(224,224),
                                                  class_mode='categorical',
                                                  shuffle=False,
                                                  seed=42,
                                                  color_mode='rgb')

# Create the base model from the pre-trained model MobileNet V2
base_model_mobilenetv2_01 = MobileNet(input_shape=(224,224,3),
                                      include_top=False)
```

```
for layer in base_model_mobilenetv2_01.layers:
    layer.trainable=False

# Add the top classification block
x = base_model_mobilenetv2_01.output
flat = GlobalAveragePooling2D()(x)
hidden_1 = Dense(1024, activation='relu')(flat)
drop_1 = Dropout(0.2)(hidden_1)
hidden_2 = Dense(512, activation='relu')(drop_1)
drop_2 = Dropout(0.3)(hidden_2)
output = Dense(9, activation= 'softmax')(drop_2)

model_mobilenet_01 = Model(base_model_mobilenetv2_01.inputs, output)
model_mobilenet_01.summary()

# Call Backs
filepath = 'mobilenet_base_model_wt.keras'
es_01 = EarlyStopping(monitor='loss', verbose=1, mode='min', patience=4)
cp_01 = ModelCheckpoint(filepath, monitor='val_loss',verbose=1,
save_best_only=True,
                    save_weights_only=False, mode='auto', save_freq='epoch')

optimizer = Adam(learning_rate=0.0004)
model_mobilenet_01.compile(loss="categorical_crossentropy",
optimizer=optimizer, metrics=["accuracy"])

mobilenetv2_history_01 = model_mobilenet_01.fit(train_generator,

steps_per_epoch=225,
                                                epochs=10,
                                                callbacks = [es_01,
cp_01],
```

```
                                                        validation_data    =
valid_generator)

# save model

if not os.path.isdir('model_weights/'):

    os.mkdir('model_weights/')

model_mobilenet_01.save_weights(filepath='model_weights/mobilenet_base_model_
wt.weights.h5',

                                overwrite=True)

# Load the saved model

model_mobilenet_01.load_weights('mobilenet_base_model_wt.keras')


# Evaluate the model on the hold out validation and test datasets

mn_val_eval_01 = model_mobilenet_01.evaluate(valid_generator)

mn_test_eval_01 = model_mobilenet_01.evaluate(test_generator)


print('Validation loss:      {0:.3f}'.format(mn_val_eval_01[0]))

print('Validation accuracy:  {0:.3f}'.format(mn_val_eval_01[1]))

print('Test loss:            {0:.3f}'.format(mn_test_eval_01[0]))

print('Test accuracy:        {0:.3f}'.format(mn_test_eval_01[1]))


# Prediction Probabilities

nb_samples = len(test_generator)

mn_predictions_01  =  model_mobilenet_01.predict_generator(test_generator,steps
= nb_samples, verbose=1)

# Prediction Labels

mn_pred_labels_01 = np.argmax(mn_predictions_01, axis=1)


# Classification Report

print('|'+'-'*67+'|')

print('|-------Classification  Report:  MobileNet  Training  Cycle  #1----------
|')

print('|'+'-'*67+'|')

print(classification_report(test_generator.classes, mn_pred_labels_01,
```

```
                            target_names=['ADI', 'BACK', 'DEB', 'LYM', 'MUC',
'MUS', 'NORM', 'STR', 'TUM']))


# Build a model
base_model_mobilenet_02 = MobileNet(input_shape=(224, 224, 3),
                                    include_top = False)
x = base_model_mobilenet_02.output
flat = GlobalAveragePooling2D()(x)
hidden_1 = Dense(1024, activation='relu')(flat)
drop_1 = Dropout(0.2)(hidden_1)
hidden_2 = Dense(512, activation='relu')(drop_1)
drop_2 = Dropout(0.3)(hidden_2)
output = Dense(9, activation= 'softmax')(drop_2)
model_mobilenet_02= Model(base_model_mobilenet_02.inputs, output)


# Load weights from the previous traning session
model_mobilenet_02.load_weights('model_weights/mobilenet_base_model_wt.keras'
)


# Freeze layers
for layer in model_mobilenet_02.layers[:82]:
    layer.trainable=False


# Call Backs
filepath = 'mobilenet_base_model_wt.keras'
es_02 = EarlyStopping(monitor='loss', verbose=1, mode='min', patience=4)
cp_02      =      ModelCheckpoint(filepath,      monitor='val_loss',verbose=1,
save_best_only=True,
                    save_weights_only=False, mode='auto', save_freq='epoch')


# compile a model
optimizer = Adam(learning_rate=0.0004)
model_mobilenet_02.compile(loss="categorical_crossentropy",
optimizer=optimizer, metrics=["accuracy"])
```

```
mobilenetv2_history_02 = model_mobilenet_02.fit(train_generator,

steps_per_epoch=225,
                                                epochs=25,
                                                callbacks = [es_02,
cp_02],
                                                validation_data    =
valid_generator)
# save model
if not os.path.isdir('model_weights/'):
    os.mkdir('model_weights/')
model_mobilenet_02.save_weights(filepath='mobilenet_model_02.weights.h5',
                                overwrite=True)
# Load the saved model
model_mobilenet_02.load_weights('mobilenet_base_model_wt.keras')
# Evaluate the model on the hold out validation and test datasets

mn_val_eval_02 = model_mobilenet_02.evaluate(valid_generator)
mn_test_eval_02 = model_mobilenet_02.evaluate(test_generator)

print('Validation loss:     {}'.format(mn_val_eval_02[0]))
print('Validation accuracy: {}'.format(mn_val_eval_02[1]))
print('Test loss:           {}'.format(mn_test_eval_02[0]))
print('Test accuracy:       {}'.format(mn_test_eval_02[1]))

# Prediction Probabilities
nb_samples = len(test_generator)
mn_predictions_02 = model_mobilenet_02.predict_generator(test_generator,steps
= nb_samples, verbose=1)
# Prediction Labels
mn_pred_labels_02 = np.argmax(mn_predictions_02, axis=1)

# Classification Report
print('|'+'-'*67+'|')
```

```
print('|-------Classification Report: MobileNet Training Cycle #2-----------
|')

print('|'+'-'*67+'|')

print(classification_report(test_generator.classes, mn_pred_labels_02,

                            target_names=['ADI', 'BACK', 'DEB', 'LYM', 'MUC',
'MUS', 'NORM', 'STR', 'TUM']))

base_model_mobilenet_03 = MobileNet(input_shape=(224, 224, 3),

                                    include_top = False)

x = base_model_mobilenet_03.output

flat = GlobalAveragePooling2D()(x)

hidden_1 = Dense(1024, activation='relu')(flat)

drop_1 = Dropout(0.2)(hidden_1)

hidden_2 = Dense(512, activation='relu')(drop_1)

drop_2 = Dropout(0.3)(hidden_2)

output = Dense(9, activation= 'softmax')(drop_2)


model_mobilenet_03= Model(base_model_mobilenet_03.inputs, output)

model_mobilenet_03.load_weights('mobilenet_base_model_wt.keras')


# Call Backs

filepath = 'mobilenet_base_model_wt.keras'

es_03 = EarlyStopping(monitor='loss', verbose=1, mode='min', patience=4)

cp_03      =      ModelCheckpoint(filepath,      monitor='val_loss',verbose=1,
save_best_only=True,

                    save_weights_only=False, mode='auto', save_freq='epoch')


for layer in model_mobilenet_03.layers[:55]:

    layer.trainable=False


optimizer = Adam(learning_rate=0.0004)

model_mobilenet_03.compile(optimizer = optimizer,

                           loss = 'categorical_crossentropy',

                           metrics = ['accuracy'])
```

```
mobilenetv2_history_03 = model_mobilenet_03.fit(train_generator,
                                    steps_per_epoch=225,
                                    epochs=10,
                                    callbacks = [es_03, cp_03],
                                    validation_data = valid_generator)
# save model
if not os.path.isdir('model_weights/'):
    os.mkdir('model_weights/')
model_mobilenet_03.save_weights(filepath='model_weights/mobilenet_base_model_
wt.weights.h5', overwrite=True)


# Load the saved model
model_mobilenet_03.load_weights('mobilenet_base_model_wt.keras')
# Evaluate the model on the hold out validation and test datasets


mn_val_eval_03 = model_mobilenet_03.evaluate(valid_generator)
mn_test_eval_03 = model_mobilenet_03.evaluate(test_generator)


print('Validation loss:       {}'.format(mn_val_eval_03[0]))
print('Validation accuracy:   {}'.format(mn_val_eval_03[1]))
print('Test loss:             {}'.format(mn_test_eval_03[0]))
print('Test accuracy:         {}'.format(mn_test_eval_03[1]))


# Predict probability
nb_samples = len(test_generator)
mn_predictions_03 = model_mobilenet_03.predict_generator(test_generator,steps
= nb_samples,verbose=1)
# Prediction labels
mn_pred_labels_03 = np.argmax(mn_predictions_03, axis=1)


# Classification Report
print('|'+'-'*67+'|')
print('|-------Classification Report: MobileNet Training Cycle #3----------|')
```

```
print('|'+'-'*67+'|')

print(classification_report(test_generator.classes, mn_pred_labels_03,

                            target_names=['ADI', 'BACK', 'DEB', 'LYM', 'MUC',
'MUS', 'NORM', 'STR', 'TUM']))

base_model_mobilenet_04 = MobileNet(input_shape=(224, 224, 3),

                                    include_top = False)

x = base_model_mobilenet_04.output

flat = GlobalAveragePooling2D()(x)

hidden_1 = Dense(1024, activation='relu')(flat)

drop_1 = Dropout(0.2)(hidden_1)

hidden_2 = Dense(512, activation='relu')(drop_1)

drop_2 = Dropout(0.3)(hidden_2)

output = Dense(9, activation= 'softmax')(drop_2)


model_mobilenet_final= Model(base_model_mobilenet_04.inputs, output)


# Load weights from the previous traning session

model_mobilenet_final.load_weights('mobilenet_base_model_wt.keras')


# Call Backs

filepath = 'mobilenet_base_model_wt.keras'

es_04 = EarlyStopping(monitor='loss', verbose=1, mode='min', patience=4)

cp_04      =      ModelCheckpoint(filepath,      monitor='val_loss',verbose=1,
save_best_only=True,

                    save_weights_only=False, mode='auto', save_freq='epoch')


# For the final tuning of the entire let's use Stochastic Gradient Descent and
slow tuning

sgd = SGD(lr=.00001, decay=1e-6, momentum=0.9, nesterov=True)

model_mobilenet_final.compile(optimizer = sgd,

                          loss = 'categorical_crossentropy',

                          metrics = ['accuracy'])

mobilenetv2_history_final = model_mobilenet_final.fit(train_generator,
```

```
                                steps_per_epoch=225,
                                epochs=50,
                                callbacks = [es_04, cp_04],
                                validation_data = valid_generator)
# Load the saved model
model_mobilenet_final.load_weights('mobilenet_base_model_wt.keras')
# Evaluate the model on the hold out validation and test datasets

mn_val_eval_final = model_mobilenet_final.evaluate(valid_generator)
mn_test_eval_final = model_mobilenet_final.evaluate(test_generator)

print('Validation loss:      {}'.format(mn_val_eval_final[0]))
print('Validation accuracy:  {}'.format(mn_val_eval_final[1]))
print('Test loss:            {}'.format(mn_test_eval_final[0]))
print('Test accuracy:        {}'.format(mn_test_eval_final[1]))

# Prediction probability
nb_samples = len(test_generator)
mn_predictions_final=
model_mobilenet_final.predict_generator(test_generator,steps                =
nb_samples,verbose=1)
# Predict labels
mn_pred_labels_final = np.argmax(mn_predictions_final, axis=1)

# Classification Report
print('|'+'-'*67+'|')
print('|-------Classification Report: MobileNet Training Cycle #3----------|')
print('|'+'-'*67+'|')

print(classification_report(test_generator.classes, mn_pred_labels_final,
                            target_names=['ADI', 'BACK', 'DEB', 'LYM', 'MUC',
'MUS', 'NORM', 'STR', 'TUM']))
mn_conf_mat_01     =     pd.DataFrame(confusion_matrix(test_generator.classes,
mn_pred_labels_01),
```

```
                        index=['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS',
'NORM', 'STR', 'TUM'],
                        columns=['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS',
'NORM', 'STR', 'TUM'])



mn_conf_mat_02     =     pd.DataFrame(confusion_matrix(test_generator.classes,
mn_pred_labels_02),
                        index=['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS',
'NORM', 'STR', 'TUM'],
                        columns=['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS',
'NORM', 'STR', 'TUM'])




mn_conf_mat_03     =     pd.DataFrame(confusion_matrix(test_generator.classes,
mn_pred_labels_03),
                        index=['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS',
'NORM', 'STR', 'TUM'],
                        columns=['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS',
'NORM', 'STR', 'TUM'])



mn_conf_mat_final    =    pd.DataFrame(confusion_matrix(test_generator.classes,
mn_pred_labels_final),
                        index=['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS',
'NORM', 'STR', 'TUM'],
                        columns=['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS',
'NORM', 'STR', 'TUM'])
# Plot confusion matrices
sns.set(font_scale=1.8)
fig, ([ax1,ax2],[ax3,ax4]) = plt.subplots(nrows=2, ncols=2, figsize=(28,28))


#ax1
sns.heatmap(mn_conf_mat_01,  ax=ax1,  annot=True,  fmt=".1f",  linewidths=0.5,
square=True, cmap='Blues_r', annot_kws={"size": 18})
ax1.set_ylabel("Actual Label", fontsize=24)
```

```
ax1.set_xlabel("Predicted Label", fontsize=24)

all_sample_title="MobileNet Training Round #1 \nAccuracy Score: {0:.3f}".format(mn_test_eval_01[1])

ax1.set_title(all_sample_title, size=32)

ax1.set_ylim(len(mn_conf_mat_01)-0.1, -0.1)


#ax2

sns.heatmap(mn_conf_mat_02, ax=ax2, annot=True, fmt=".1f", linewidths=0.5, square=True, cmap='Blues_r', annot_kws={"size": 18})

ax2.set_ylabel("Actual Label", fontsize=24)

ax2.set_xlabel("Predicted Label", fontsize=24)

all_sample_title="MobileNet Training Round #2 \nAccuracy Score: {0:.3f}".format(mn_test_eval_02[1])

ax2.set_title(all_sample_title, size=32)

ax2.set_ylim(len(mn_conf_mat_02)-0.1, -0.1)


#ax3

sns.heatmap(mn_conf_mat_03, ax=ax3, annot=True, fmt=".1f", linewidths=0.5, square=True, cmap='Blues_r', annot_kws={"size": 18})

ax3.set_ylabel("Actual Label", fontsize=24)

ax3.set_xlabel("Predicted Label", fontsize=24)

all_sample_title="MobileNet Training Round#3 \nAccuracy Score: {0:.3f}".format(mn_test_eval_03[1])

ax3.set_title(all_sample_title, size=32)

ax3.set_ylim(len(mn_conf_mat_03)-0.1, -0.1)


#ax4

sns.heatmap(mn_conf_mat_final, ax=ax4, annot=True, fmt=".1f", linewidths=0.5, square=True, cmap='Blues_r', annot_kws={"size": 18})

ax4.set_ylabel("Actual Label", fontsize=24)

ax4.set_xlabel("Predicted Label", fontsize=24)

all_sample_title="MobileNet Training Round#4 \nAccuracy Score: {0:.3f}".format(mn_test_eval_final[1])

ax4.set_title(all_sample_title, size=32)

ax4.set_ylim(len(mn_conf_mat_final)-0.1, -0.1) plt.tight_layout()
```

References

[1] Tissue classification and diagnosis of colorectal cancer histopathology images using deep learning algorithms. DD Chlorogiannis, GI Verras, V Tzelepi… - Gastroenterology …, 2023 - termedia.pl

[2] Evaluation of a deep neural network for automated classification of colorectal polyps on histopathologic slides. JW Wei, AA Suriawinata, LJ Vaickus, B Ren… - JAMA network …, 2020 - jamanetwork.com

[3] Accurate diagnosis of colorectal cancer based on histopathology images using artificial intelligence. KS Wang, G Yu, C Xu, XH Meng, J Zhou, C Zheng… - BMC medicine, 2021 – Springer

[4] Automated histological classification for digital pathology images of colonoscopy specimen via deep learning. S Byeon, J Park, YA Cho, BJ Cho - Scientific Reports, 2022 - nature.com

[5] Deep neural networks for automated classification of colorectal polyps on histopathology slides: a multi-institutional evaluation. JW Wei, AA Suriawinata, LJ Vaickus, B Ren… - arXiv preprint arXiv …, 2019 - arxiv.or

[6] Deep learning techniques for the classification of colorectal cancer tissue. MJ Tsai, YH Tao - Electronics, 2021 - mdpi.co

[7] Evaluation of an artificial intelligence–augmented digital system for histologic classification of colorectal polyps. M Nasir-Moin, AA Suriawinata, B Ren, X Liu… - JAMA Network …, 2021 - jamanetwork.com

[8] Machine learning and deep learning-C Janiesch, P Zschech, K Heinrich - Electronic Markets, 2021 - Springer

[9] Deep learning on histopathological images for colorectal cancer diagnosis: A systematic review. A Davri, E Birbas, T Kanavos, G Ntritsos, N Giannakeas… - Diagnostics, 2022 - mdpi.com

[10] Real-time artificial intelligence–based histologic classification of colorectal polyps with augmented visualization. E Rodriguez-Diaz, G Baffy, WK Lo, H Mashimo… - Gastrointestinal …, 2021 – Elsevier

[11] Machine learning and the physical sciences-G Carleo, I Cirac, K Cranmer, L Daudet, M Schuld… - Reviews of Modern …, 2019 - APS

[12] Automated detection and classification of desmoplastic reaction at the colorectal tumour front using deep learning. IP Nearchou, H Ueno, Y Kajiwara, K Lillard… - Cancers, 2021 - mdpi.com

[13] Automated classification of inflammation in colon histological sections based on digital microscopy and advanced image analysis. L Ficsor, VS Varga, A Tagscherer… - Cytometry Part A …, 2008 - Wiley Online Library

[14] Accurate diagnosis of colorectal cancer based on histopathology images using artificial intelligence. KS Wang, G Yu, C Xu, XH Meng, J Zhou, C Zheng… - BMC medicine, 2021 – Springer

[15] Infrared spectral histopathology for cancer diagnosis: a novel approach for automated pattern recognition of colon adenocarcinoma. J Nallala, MD Diebold, C Gobinet, O Bouché… - Analyst, 2014 - pubs.rsc.org

[16] Deep neural networks for automated classification of colorectal polyps on histopathology slides: a multi-institutional evaluation. JW Wei, AA Suriawinata, LJ Vaickus, B Ren… - arXiv preprint arXiv …, 2019 - arxiv.org

[17] Ensemble-based multi-tissue classification approach of colorectal cancer histology images using a novel hybrid deep learning framework. M Khazaee Fadafen, K Rezaee - Scientific Reports, 2023 - nature.com

[18] [HTML] Automated histological classification of whole slide images of colorectal biopsy specimens. H Yoshida, Y Yamashita, T Shimazu, E Cosatto… - Oncotarget, 2017 - ncbi.nlm.nih.gov

[19] Deep learning techniques for the classification of colorectal cancer tissue MJ Tsai, YH Tao-Electronics, 2021•mdpi.com.

[20] Crccn-net: Automated framework for classification of colorectal tissue using histopathological images-A Kumar, A Vishwakarma, V Bajaj - Biomedical Signal Processing and …, 2023 – Elsevier