A blue and white sign with white text

Description automatically generated

Capstone Project Phase A

**Using transfer learning for diagnosing LPR signs**

Fadi Srour ID: 322253758, Pier Mbariky ID: 207718826

Supervisor: Dr. Elena Kleiman

25-1-R-4

[1. Introduction 5](#_Toc189832855)

[**1.1** **Challenges in Diagnosis:** 5](#_Toc189832856)

[**1.2** **Focused Diagnostic Indicators for Machine Learning** 6](#_Toc189832857)

[**1.2.1** **Erythema** 6](#_Toc189832858)

[**1.2.2** **Ventricular Obliteration** 7](#_Toc189832859)

[**1.2.3** **Posterior Commissure Hypertrophy** 7](#_Toc189832860)

[**1.3** **Role of Machine Learning in Addressing Diagnostic Challenges** 8](#_Toc189832861)

[2. Background: 9](#_Toc189832862)

[**2.1** **Machine Learning and Deep Learning:** 9](#_Toc189832863)

[**2.1.1** **Deep Learning:** 9](#_Toc189832864)

[**2.1.2** **Artificial Neural Networks (ANNs):** 9](#_Toc189832865)

[**2.1.3** **Convolutional Neural Networks (CNN)** 10](#_Toc189832866)

[**2.1.4** **Forward Propagation in Neural Networks** 12](#_Toc189832867)

[**2.1.5** **Backpropagation in Neural Networks** 12](#_Toc189832868)

[**2.1.6** **Layer Freezing** 13](#_Toc189832869)

[**2.1.7** **Data Augmentation** 13](#_Toc189832871)

[**2.1.8** **Transfer Learning:** 15](#_Toc189832872)

[**2.2** **VGG19:** 18](#_Toc189832873)

[**2.2.1** **Introduction:** 18](#_Toc189832874)

[**2.2.2** **VGG-19 Architecture** 18](#_Toc189832875)

[**2.2.3** **Detailed Layer-by-Layer Architecture of VGG-Net 19** 19](#_Toc189832876)

[**2.2.4** **Use in Transfer Learning** 19](#_Toc189832877)

[**2.2.5** **Computational Complexity** 21](#_Toc189832878)

[**2.2.6** **Comparisons with ResNet and EfficientNet** 21](#_Toc189832879)

[**2.2.7** **Variants of VGG** 21](#_Toc189832880)

[**2.3** **EfficentNet-B2:** 22](#_Toc189832881)

[**2.3.1** **Introduction:** 22](#_Toc189832882)

[**2.3.2** **EfficientNet-B2 Architecture** 22](#_Toc189832883)

[**2.3.3** **Layer-by-Layer Breakdown of EfficientNet-B2** 22](#_Toc189832885)

[**2.3.4** **Use in Transfer Learning** 23](#_Toc189832887)

[**2.3.5** **Attention Mechanism in EfficientNet** 24](#_Toc189832890)

[**2.3.6** **How the SE Block Works** 24](#_Toc189832891)

[**2.3.7** **Computational Complexity** 24](#_Toc189832892)

[**2.3.8** **Comparisons with VGG-19 and ResNet:** 25](#_Toc189832893)

[**2.3.9** **Variants of EfficientNet** 25](#_Toc189832894)

[**2.4** **ResNet-101:** 26](#_Toc189832895)

[**2.4.1** **Introduction** 26](#_Toc189832896)

[**2.4.2** **ResNet-101 Architecture** 26](#_Toc189832897)

[**2.4.3** **Key Components:** 26](#_Toc189832898)

[**2.4.4** **Layer-by-Layer Breakdown of ResNet-101** 27](#_Toc189832899)

[**2.4.5** **Use in Transfer Learning** 27](#_Toc189832900)

[**2.4.6** **Computational Complexity** 28](#_Toc189832903)

[**2.4.7** **Comparisons with Other Architectures** 28](#_Toc189832904)

[**2.4.8** **Variants of ResNet** 28](#_Toc189832905)

[3. Proposed Method & Research Plan 29](#_Toc189832906)

[**3.1** **Preprocessing** 29](#_Toc189832907)

[4. Expected Challenges 30](#_Toc189832908)

[5. Diagrams & Preliminary Software Engineering Documents 31](#_Toc189832909)

[**5.1** **Flow Charts:** 31](#_Toc189832910)

[**5.1.1 Training model flow chart** 31](#_Toc189832911)

[**5.1.1** **Test model flow chart:** 31](#_Toc189832912)

[6. Test Plan & Experiments 32](#_Toc189832913)

[**6.1** **Test Plan** 32](#_Toc189832914)

[6.2 Planned Experiments 33](#_Toc189832915)

[7.Expected Results & Future Work 37](#_Toc189832916)

[**7.1** **Expected Results** 37](#_Toc189832917)

[**7.2** **Future Work** 37](#_Toc189832918)

[8. Bibliography 38](#_Toc189832919)

Github repository: <https://github.com/za3bor/Using-transfer-learning-for-diagnosing-LPR-signs.git>

(Our link is private because it includes sensitive data about real patient)

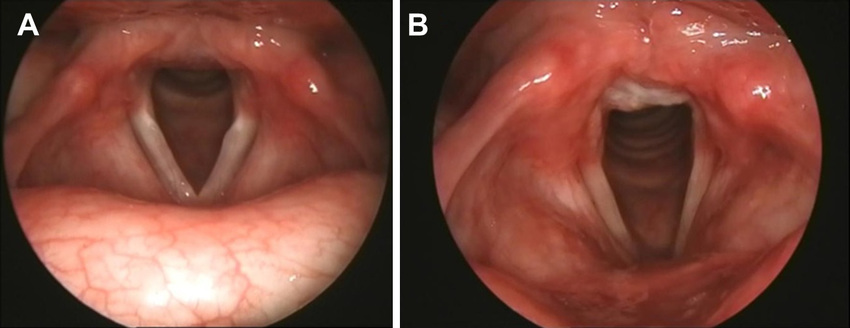
***Abstract***

*Laryngopharyngeal Reflux (LPR) is a medical condition resulting from the retrograde flow of gastric contents into the upper airway, causing irritation and inflammation. Unlike gastroesophageal reflux disease (GERD), LPR often lacks classic symptoms like heartburn, instead manifesting as hoarseness, chronic cough, throat clearing, globus sensation, and postnasal drip. These nonspecific symptoms, coupled with the subjective nature of current diagnostic methods such as symptom indices and laryngoscopy evaluations, pose significant challenges for accurate diagnosis and effective treatment​. This project aims to create a comprehensive computational system to analyse transnasal laryngoscopy images and assist in LPR diagnosis by detecting both visual and symptomatic indicators. The system utilizes state-of-the-art deep learning algorithms to identify and quantify anatomical changes linked to LPR, such as erythema, posterior commissure hypertrophy, and Ventricular Obliteration. Moreover, the approach incorporates automated symptom analysis by linking visible laryngoscopy features to commonly reported complaints like vocal fatigue or throat discomfort. Our methodology includes curating a diverse, annotated dataset of laryngoscopy images, employing advanced image preprocessing techniques, and optimizing machine learning architectures for feature detection and symptom correlation. Techniques like transfer learning, data augmentation, and explainable AI are integrated to enhance model performance and reliability. The tool also offers interpretability features to provide clinicians with clear, actionable insights, facilitating both diagnosis and patient education. The anticipated outcomes include a diagnostic system that reduces subjectivity, improves efficiency, and enhances clinical decision-making by enabling the automatic detection of symptomatic and anatomical markers of LPR. This innovation promises to streamline workflows for healthcare providers and improve patient outcomes through earlier and more accurate diagnosis.*

***Keywords:*** *Transfer Learning, Laryngopharyngeal Reflux, Laryngoscopy Video Analysis, Computer-Aided Diagnosis, Convolutional Neural Networks, VGG-19, EffiecentNet-B2, ResNet-101, medical imaging.*

## **Introduction**

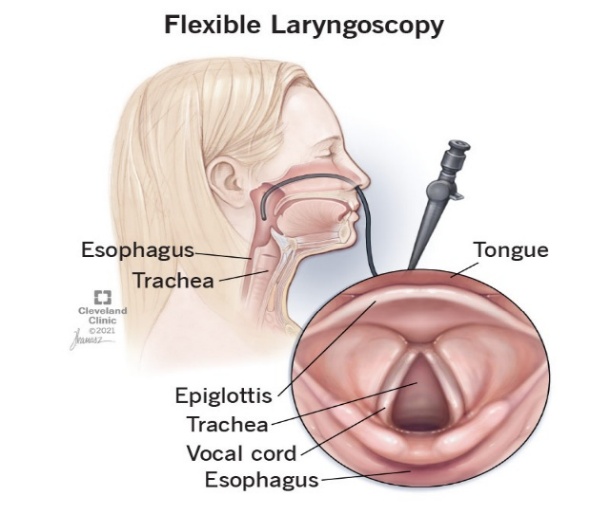
Laryngopharyngeal Reflux (LPR) is a condition characterized by the involuntary regurgitation of stomach contents, including acid, bile, and digestive enzymes, into the larynx and pharynx. Unlike Gastroesophageal Reflux Disease (GERD), which primarily affects the lower esophagus, LPR primarily impacts the upper airway. As a result, it manifests in symptoms such as hoarseness, throat clearing, chronic cough, and the sensation of a lump in the throat (Refer to Figure 1). These symptoms often overlap with those of other conditions, such as allergies or respiratory infections, making LPR particularly challenging to diagnose. Misdiagnosis is common, and the subtle nature of LPR symptoms contributes to its frequent underdiagnosis [1].



***Figure 1*:** Comparing the larynx without LPR (A) to one affected by LPR (B).

### **Challenges in Diagnosis:**

Laryngoscopy is the primary diagnostic tool for Laryngitis with Pharyngeal Secretions (LPR). During this procedure, a physician utilizes an endoscope to visually examine the larynx and pharynx for indications of inflammation or irritation (Figure 2). The video footage obtained is crucial for identifying diagnostic markers such as erythema, ventricular obliteration, and posterior commissure hypertrophy.



***Figure 2*: Laryngoscopy Procedure**

However, this diagnostic process faces several significant hurdles, which a machine learning system could potentially address:

1. **Subjectivity:** Doctors must rely on their experience to spot signs like redness or swelling, which can be tricky and different from doctor to doctor. This can lead to different diagnoses.
2. **Time-consuming:** Looking at long laryngoscopy videos to find the symptoms is a long and slow process. Doctors must go through lots of frames, which can be a problem, especially in busy clinics.
3. **Video quality issues**: Things like the patient moving around, the camera not being in the right place, or not enough light can make the footage bad, making it hard to see the signs of LPR and making the diagnosis harder.
4. **Cognitive load:** Doctors must think about what’s important and ignore the rest while looking at the important parts of the body. This can be tiring and make them more likely to miss something.
5. **Limited review:** Doctors often only review live video footage during procedures and immediately generate reports. They rarely revisit the footage afterward, potentially missing subtle or less apparent signs of LPR that might become more evident with a more comprehensive review.

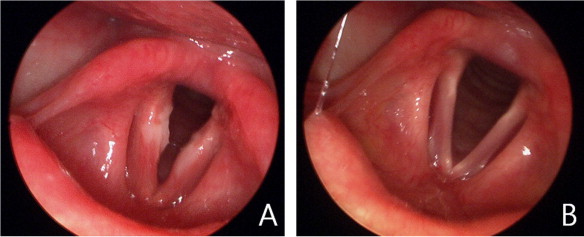
### **Focused Diagnostic Indicators for Machine Learning**

To streamline and enhance the diagnostic process of LPR, a machine learning system could focus on three primary indicators.

#### **Erythema (figure 3) [2]:**

Erythema, or redness and inflammation of the laryngeal tissues, is a distinctive characteristic of LPR, but it presents a challenge in detection due to the following reasons:

1. Variability in appearance, influenced by lighting, camera angles, and individual tissue color variations, significantly impacts its visibility.
2. Non-specificity: Erythema can also be caused by other conditions such as allergies or infections, making it challenging to attribute it solely to LPR.

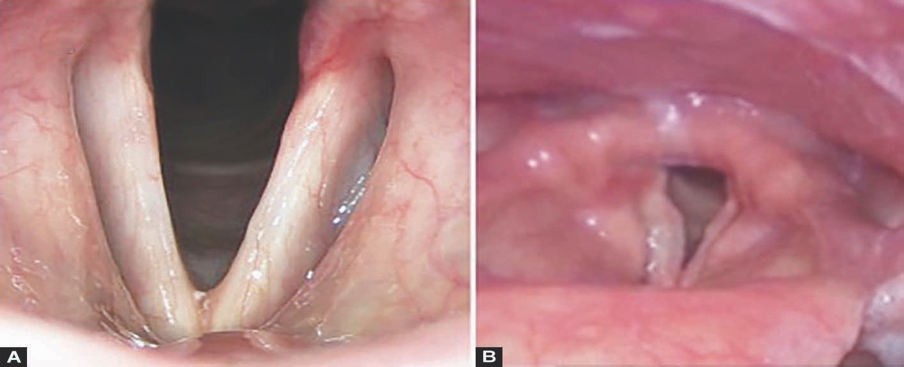


***Figure* 3: Laryngoscopic View of Erythema** – (A) Erythematous larynx in LPR; (B) Normal larynx.

#### **Ventricular Obliteration (figure 4) [2]:**

This refers to the obscuring of the normally visible ventricular space between the true and false vocal cords, which is frequently caused by swelling or edema. Detection is difficult because:

1. Subtle differences, such as mild swelling, can be challenging to distinguish from normal variations, especially without prior baseline images.
2. Subjectivity plays a crucial role in accurate identification. The physician’s interpretation of normal anatomy can vary significantly, which can impact the accuracy of the diagnosis.

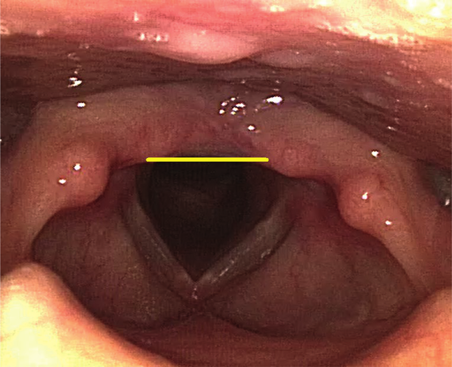


**Figure 4:** Laryngeal Comparison: Normal Vocal Cords (A) vs. Infected/Pathological Vocal Cords (B)

#### **Posterior Commissure Hypertrophy (figure 5) [2]:**

This condition, characterized by the thickening or enlargement of tissue at the back of the larynx, usually resulting from chronic irritation caused by reflux, presents several challenges in its detection.

1. Hypertrophy typically develops gradually, necessitating comparative assessments to identify any deviations.
2. Physicians may differ in their definitions of pathological hypertrophy, resulting in diagnostic variability due to inconsistent standards.



**Figure 5: Posterior Commissure Hypertrophy in the Larynx"**

### **Role of Machine Learning in Addressing Diagnostic Challenges**

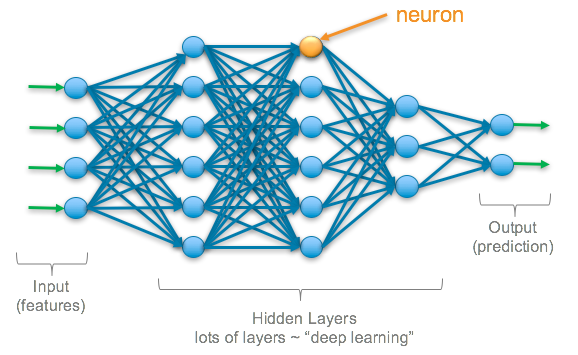
By integrating machine learning into the diagnostic workflow, these challenges can be effectively addressed. A machine learning model could potentially:

1. Standardize assessments by providing consistent, algorithm-driven evaluations of video footage to reduce subjectivity.
2. Automate the analyzis process to swiftly and precisely identify crucial indicators such as erythema, ventricular obliteration, and posterior commissure hypertrophy.
3. Enhance Video Quality Interpretation: Compensate for suboptimal footage by applying image enhancement techniques.
4. Assist Physicians by highlighting areas of concern in real-time or during post-procedure reviews to reduce cognitive load and ensure that subtle signs are not missed.

## **Background:**

### **Machine Learning and Deep Learning:**

#### **Deep Learning [3]:**

a specialized branch of machine learning that has become indispensable in solving complex problems across diverse fields. At its core, deep learning employs artificial neural networks with numerous layers, designed to mimic the way the human brain processes information. These networks can learn from vast amounts of unstructured data, such as images, text, and audio, by automatically identifying patterns and correlations that might not be discernible through conventional analytical methods. This unique capability makes deep learning highly effective for tasks like image classification, speech recognition, and natural language processing, among others. Its ability to enhance with more data and computational power has contributed to its widespread adoption across industries, ranging from healthcare, where it aids in medical image analyzis, to autonomous vehicles, where it powers self-driving systems. The ongoing advancements in deep learning continue to drive innovations that push the boundaries of artificial intelligence, positioning it as a pivotal technology in the development of intelligent systems that learn and adapt from experience (see Figure 6).

**Figure 6:** Overview of a Deep Learning Neural Network Architecture.

#### **Artificial Neural Networks (ANNs) [4]:**

are computational processing systems inspired by the operation of biological nervous systems, such as the human brain. ANNs comprise numerous interconnected computational nodes, known as neurons, that collaborate in a distributed manner to collectively learn from input data and optimize the final output. The fundamental structure of an ANN can be depicted as illustrated in Figure 7.

Within an ANN, input data, typically in the form of a multidimensional vector, is input into the input layer. Subsequently, this data is distributed to the hidden layers. Each hidden layer processes the information received from the preceding layer, adjusting and refining it through weighted connections. This adjustment process, referred to as learning, entails the hidden layers evaluating the impact of stochastic variations within the network on the final output, thereby improving or diminishing it.

For instance, in an image recognition task, the input layer might receive pixel values from an image. These values are subsequently transmitted through hidden layers, where the network acquires the ability to identify features such as edges, shapes, and textures. Ultimately, the network can classify the image, distinguishing between objects like cats and dogs with high accuracy. This learning process is iterative and involves adjusting weights based on errors in predictions, thereby facilitating the ANN’s continuous improvement over time.

**Figure 7:** Artificial Neural Network Structure

#### **Convolutional Neural Networks (CNN) [5]:**

A Convolutional Neural Network (CNN) is a type of neural network specifically designed to process and analyze multi-dimensional data, particularly images. It comprises an input layer, convolutional layers, pooling layers, and an output layer. These layers can be stacked and repeated to create a more sophisticated architecture tailored to specific tasks.

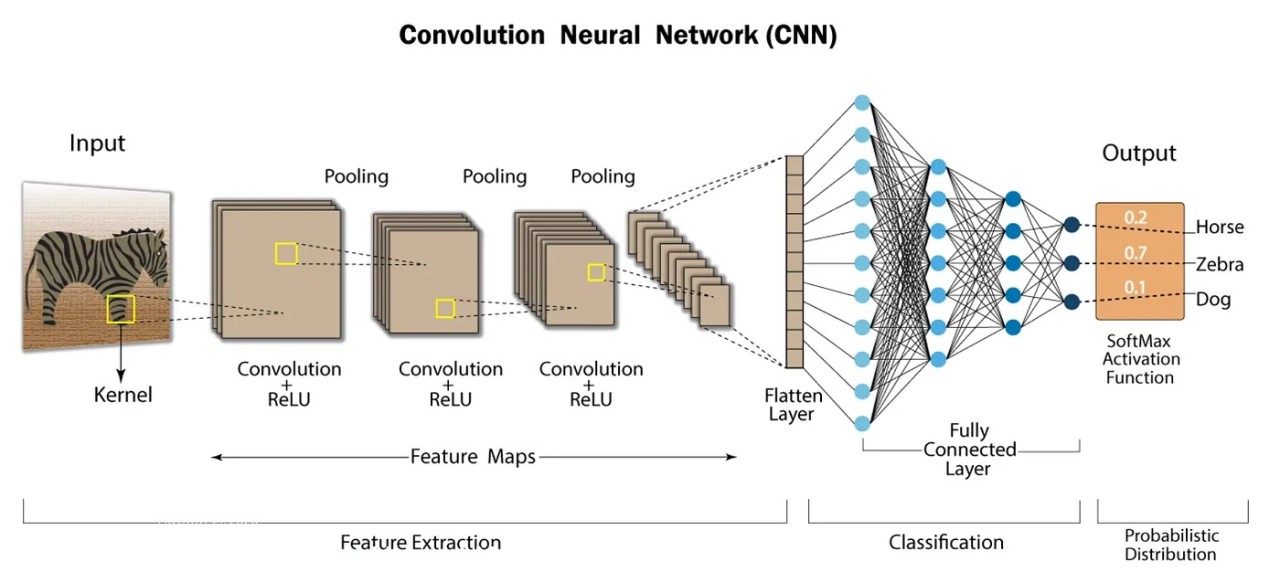
The input to the convolutional layer is usually a two-dimensional matrix that represents the image. Convolutional layers employ various filters to extract distinct features from the image. A filter is a smaller matrix that traverses the image, performing a convolution operation by multiplying each pixel value by the corresponding filter value and accumulating the results. This process generates a transformed image with new weights that emphasize specific features, such as edges or textures.

In the pooling layer, a new matrix is constructed by summarizing values from the filtered cells using techniques like max pooling or average pooling. Pooling reduces the dimensionality of the data, which helps minimize computational complexity and prevents overfitting while preserving essential features. The convolution and pooling steps are repeated multiple times to construct a deep network capable of identifying intricate patterns.

Similar to how our brains scan images from left to right or top to bottom, combining various local features to recognize an object, CNNs also operate hierarchically. For instance, in facial recognition, a CNN might initially identify edges, then shapes like eyes and mouths, and finally the overall structure of the face. This hierarchical approach enables CNNs to perform complex image classification tasks with high accuracy.

##### **CNN Architecture:**

Four fundamental types of layers are employed in constructing CNN architectures: the input layer, the convolutional layer, the pooling layer, and the fully connected layer. A comprehensive CNN architecture is typically assembled by sequentially stacking multiple instances of these layers. An illustrative example of a typical CNN architecture comprising two feature stages is presented in Figure 8.



**Figure 8:** General Conventional network Architecture.

##### **Layers of a Convolutional Neural Network**

1. **Input Layer:**

The input layer serves as the entry point for receiving input data and transmitting it to the subsequent convolutional layers. Preprocessing techniques, such as mean subtraction and feature scaling, are commonly employed within this layer to normalize the input data.

1. **Convolutional Layer**

The convolutional layer is a fundamental component of a convolutional neural network (CNN). It applies various filters to the input, performing convolutions across the spatial dimensions to generate 2D activation maps. These activation maps capture key features in the data.

1. **Pooling Layer**

Pooling layers are employed to reduce the spatial dimensions of the data, thereby mitigating computational overhead and controlling overfitting. Typically, a pooling layer is positioned immediately following each convolutional layer. The initial pooling layer predominantly utilizes max pooling with a 2x2 kernel size. receptive field and a stride of 2. Later pooling layers might use average pooling instead, depending on the design.

1. **Fully connected layer:**

Fully connected layers are indispensable in convolutional neural networks (CNNs), especially for tasks such as image recognition and classification in computer vision. Following convolution and pooling layers, which extract and analyze features from the image, the fully connected layer receives these features as inputs. It establishes connections with all preceding activation layers and plays a pivotal role in determining the final classification decision. This layer facilitates the integration of extracted features to yield the network’s output, which is typically utilized for classification tasks.

#### **Forward Propagation in Neural Networks [6]**

As the name suggests, forward propagation is the process of transmitting input data through a network in the forward direction. Each hidden layer receives input data, processes it using an activation function, and then transmits the output to the subsequent layer.

To generate an output, the input data must be propagated through the network in a unidirectional manner. This unidirectional flow is crucial because if the data were to flow in reverse, it would generate a cycle, hindering the network’s ability to produce a suitable output. (Figure 9)

#### **Backpropagation in Neural Networks [7]**

Backpropagation is a fundamental training algorithm for neural networks, enabling them to learn complex input-output relationships without explicitly revealing the underlying mathematical equations.

The learning rule in backpropagation employs the steepest descent method, which adjusts the weight and threshold values of the network to minimize the sum of squared errors.

Backpropagation is indispensable for neural network training, serving as a mechanism to refine the weights based on the error rate obtained in the preceding iteration. This process ensures lower error rates, thereby enhancing the model’s reliability and generalization capabilities. (Figure 9)

A diagram of a diagram

Description automatically generated

**Figure 9:** Forward and Backward Propagation in Neural Networks

#### **Layer Freezing [8]**

Gradual layer freezing prevents early modules from being retrained (“frozen”), thereby reducing the computational cost of backpropagation and subsequent updates to frozen modules. The rationale behind layer freezing is that early layers may acquire their features more rapidly than later layers, rendering them unnecessary for further updates during training.

#### **Data Augmentation [9]**

Data augmentation is the process of producing modified versions of preexisting data points in order to increase the size of a training dataset. This procedure creates new data samples by making minor adjustments or by employing sophisticated deep learning techniques. Several data augmentation methods for photos are examined in this study:

* **Rotation:** This technique involves rotating images in range -45 to 45. It helps the model become invariant to object orientation, enabling it to recognize objects from various viewpoints.

A collage of a cat

Description automatically generated

* **Zooming:** Zooming randomly enlarges or reduces the image, introducing scale variations. It helps the model learn to recognize objects at different sizes and distances.
* **Flipping:** Flipping refers to mirroring images either horizontally or vertically. This augmentation simulates different perspectives, helping the model generalize to varied orientations of objects.

A cartoon of a chair

Description automatically generated

* **Cropping:** Cropping involves cutting out random sections of an image. This technique simulates different focal points and helps the model become more resilient to partial occlusions.

A cartoon of a leg

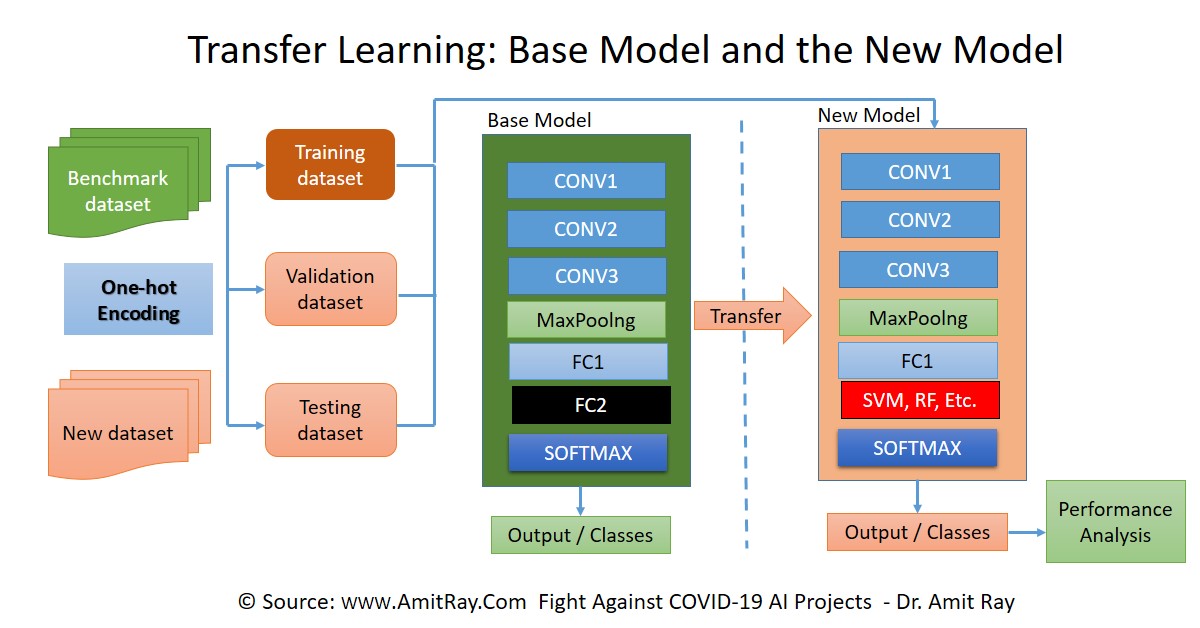
Description automatically generated with medium confidence

* **Stretching:** Stretching distorts the image by stretching it along one axis. This introduces variations in the shape of objects, improving the model’s ability to handle different aspect ratios and object distortions.

By exposing the model to a range of data variances during training, these augmentation strategies seek to increase the model's robustness. As a result, the model can more effectively generalize to new, unseen data in real-world situations.

#### **Transfer Learning [10]**

Transfer learning, a machine learning technique, enables the application of knowledge gained from one task to a related but different one. This approach accelerates model training, especially in scenarios where labeled data for the target task is scarce or absent. By leveraging pre-trained models, transfer learning achieves high performance with reduced computational resources (Figure 10).



***Figure 10*:** Overview of Transfer Learning Process

##### **Why Transfer Learning is Used: [10]**

* **Data Efficiency**: Transfer learning improves performance with limited labeled data by leveraging patterns learned from a larger dataset.
* **Computational Efficiency:** Pre-trained models save computational resources by reducing training time and power consumption.
* **Training Speed:** Transfer learning accelerates model training by starting with pre-learned weights, leading to faster convergence.

##### **When Transfer Learning is Used: [10]**

* **Small Datasets:** Beneficial when the target dataset is too small to train a robust model from scratch.
* **Similar Tasks:** Useful when a pre-trained model on a similar task can be adapted to the new task.
* **Time and Cost Constraints:** Useful in scenarios where training a model from scratch would be prohibitively expensive or time-consuming.

##### **How Transfer Learning Operates: [10]**

1. **Select a Pre-Trained Model**:  
   Choose a model that has been previously trained on a large dataset relevant to your task. For instance, models trained on ImageNet are commonly used for computer vision tasks.
2. **Configure the Model**:

* **Freeze Certain Layers**: Lock the weights of initial layers (e.g., convolutional layers in vision models) to preserve learned features.
* **Add New Layers**: Introduce new layers tailored to the target task, such as fully connected layers for classification.

1. **Fine-Tune the Model**:   
   Unfreeze some of the later layers and train the model on your specific dataset to adjust the weights according to the new task requirements.
2. **Adjust Loss Functions and Hyperparameters**:  
   Optimize the model by selecting appropriate loss functions and tuning hyperparameters like learning rate and batch size to achieve a balance between retaining learned features and adapting to new data.
3. **Train and Validate**:  
   Train the configured model on your target dataset and validate its performance to ensure it meets the desired accuracy and efficiency for your specific application.

In Natural Language Processing (NLP), transfer learning involves fine-tuning models like BERT on specific tasks by adding task-specific layers and adjusting the model to understand the nuances of the new data.

##### **Advantages:**

* **Computational Efficiency:** Reduces computational costs by leveraging pre-trained models.
* **Data Efficiency:** Minimizes the need for large labeled datasets for the target task.
* **Generalizability:** Improves the model’s ability to generalize to new, related tasks.

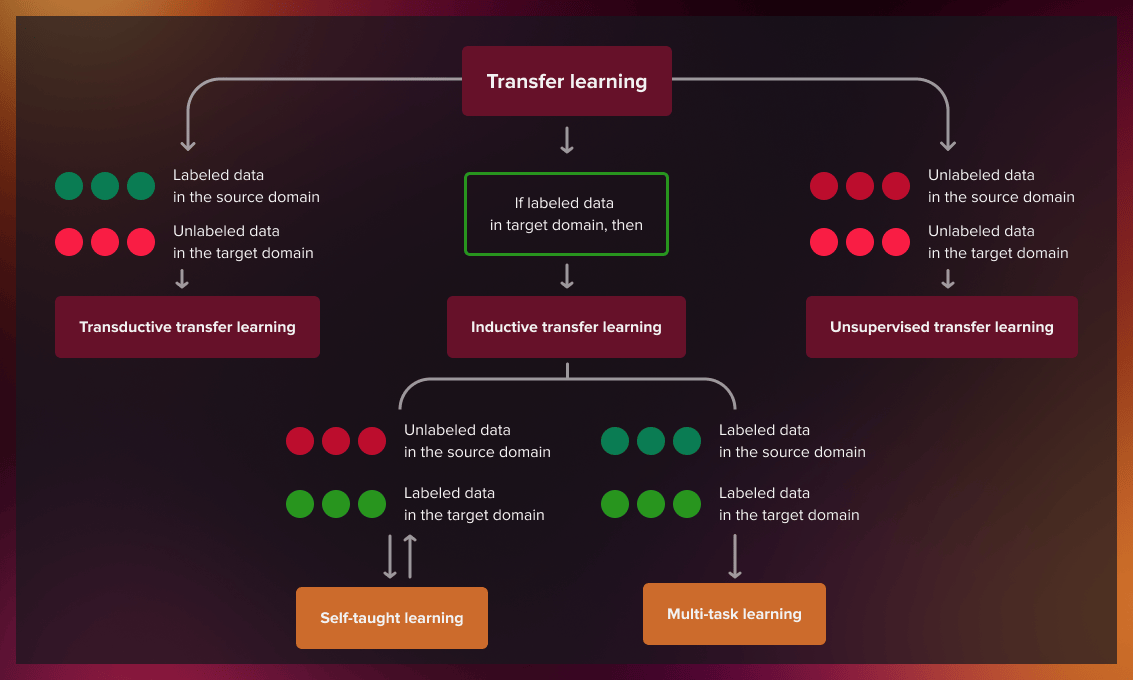
##### **Disadvantages:**

* **Negative Transfer**: Knowledge transfer can worsen performance when source and target tasks are too dissimilar.
* **Data Distribution Mismatch**: Performance suffers if the data distribution in the source task differs significantly from the target task.
* **Inappropriate Models:** Using an unsuitable pre-trained model can lead to poor knowledge transfer.

##### **2.1.8.6 Types of Transfer Learning (figure 11) [11]:**

* **Inductive Transfer:** Applies learned features from a source task to a different target task, often using labelled data.
* **Unsupervised Transfer:** Transfers knowledge without explicit labels, useful for tasks with unlabelled data.
* **Transductive Transfer:** Adapts a model to new data characteristics for the same target task.

Transfer learning is crucial in fields like NLP and computer vision, enabling models trained on large datasets like ImageNet or BERT to be fine-tuned for specific tasks.



***Figure 11*:** Overview of the Types of Transfer Learning

### **VGG19:**

#### **Introduction [12]:**

The**Visual Geometry Group (VGG) models**, particularly **VGG-16 and VGG-19**, have significantly influenced the field of computer vision since their inception. These models, introduced by the Visual Geometry Group from the University of Oxford, stood out in the 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) for their deep convolutional neural networks (CNNs) with a uniform architecture. VGG-19, the deeper variant of the VGG models, has garnered considerable attention due to its simplicity and effectiveness.

#### **VGG-19 Architecture [12]**

VGG-19 is a deep convolutional neural network with 19 weight layers, comprising 16 convolutional layers and 3 fully connected layers. The architecture follows a straightforward and repetitive pattern, making it easier to understand and implement.

The key components of the VGG-19 architecture are (figure 12):

1. **Convolutional Layers**: 3x3 filters with a stride of 1 and padding of 1 to preserve spatial resolution.
2. **Activation Function**: ReLU (Rectified Linear Unit) applied after each convolutional layer to introduce non-linearity.
3. **Pooling Layers**: Max pooling with a 2x2 filter and a stride of 2 to reduce the spatial dimensions.
4. **Fully Connected Layers**: Three fully connected layers at the end of the network for classification.
5. **Softmax Layer**: Final layer for outputting class probabilities.

A diagram of a graph

Description automatically generated with medium confidence

***Figure 12*:** VGG-19 Architecture

#### **Detailed Layer-by-Layer Architecture of VGG-Net 19 [12]**

The VGG-19 model consists of five blocks of convolutional layers, followed by three fully connected layers. Here is a detailed breakdown of each block:

**Block 1**

* Conv1\_1: 64 filters, 3x3 kernel, ReLU activation
* Conv1\_2: 64 filters, 3x3 kernel, ReLU activation
* Max Pooling: 2x2 filter, stride 2

**Block 2**

* Conv2\_1: 128 filters, 3x3 kernel, ReLU activation
* Conv2\_2: 128 filters, 3x3 kernel, ReLU activation
* Max Pooling: 2x2 filter, stride 2

**Block 3**

* Conv3\_1: 256 filters, 3x3 kernel, ReLU activation
* Conv3\_2: 256 filters, 3x3 kernel, ReLU activation
* Conv3\_3: 256 filters, 3x3 kernel, ReLU activation
* Conv3\_4: 256 filters, 3x3 kernel, ReLU activation
* Max Pooling: 2x2 filter, stride 2

**Block 4**

* Conv4\_1: 512 filters, 3x3 kernel, ReLU activation
* Conv4\_2: 512 filters, 3x3 kernel, ReLU activation
* Conv4\_3: 512 filters, 3x3 kernel, ReLU activation
* Conv4\_4: 512 filters, 3x3 kernel, ReLU activation
* Max Pooling: 2x2 filter, stride 2

**Block 5**

* Conv5\_1: 512 filters, 3x3 kernel, ReLU activation
* Conv5\_2: 512 filters, 3x3 kernel, ReLU activation
* Conv5\_3: 512 filters, 3x3 kernel, ReLU activation
* Conv5\_4: 512 filters, 3x3 kernel, ReLU activation
* Max Pooling: 2x2 filter, stride 2

**Fully Connected Layers**

* **FC1**: 4096 neurons, ReLU activation
* **FC2**: 4096 neurons, ReLU activation
* **FC3**: 1000 neurons, softmax activation (for 1000-class classification)

#### **Use in Transfer Learning**

##### **Overview of Transfer Learning: [12]**

Transfer learning with VGG-19 involves leveraging pre-trained weights (usually on ImageNet) to solve domain-specific problems. This approach reduces the need for extensive computational resources and large datasets, making it widely applicable for tasks such as medical imaging, object detection.

##### **Fine-Tuning VGG-19 [13]**

Fine-tuning VGG-19 involves selectively updating the weights of specific layers while freezing others. This approach allows the model to retain its ability to detect general features from the pre-trained layers while adapting to the new task-specific data.

* **Frozen Layers**

Lower Convolutional Layers (Blocks 1-4): These layers are generally frozen as they capture generic features like edges, corners, and textures. These features are transferable across tasks and do not require retraining. By freezing these layers, computational efficiency is improved, and overfitting is minimized, especially when working with smaller datasets.

Freezing is implemented by setting these layers to non-trainable during the fine-tuning process, ensuring their weights remain unchanged.

* **Fine-Tuned Layer**

Higher Convolutional Layer (Block 5): this layer is either partially fine-tuned or fully retrained depending on the complexity of the new task. This layer learns more specific features such as object shapes and patterns, which may need adjustment to align with the new domain.

* + - **Fully Connected Layers**

The original fully connected layers (FC1-3) are often replaced with task-specific ones.

For example, the final dense layer designed for 1000-class ImageNet classification is replaced with a layer that matches the number of output classes in the new dataset.

##### **Training Strategy [13]**

* Learning Rates: Fine-tuned layers are trained with a lower learning rate (e.g., 1e-4) to avoid drastic updates, while newly added layers are trained with higher rates (e.g., 1e-3).
* Dropout Regularization: Applied to the new fully connected layers to mitigate overfitting.

The fine-tuning process is particularly effective in domains with limited data, as it balances the reuse of robust pre-trained features with adaptability for new tasks. For example, VGG-19 has been successfully fine-tuned for applications like medical imaging (e.g., chest X-ray classification) and autonomous vehicle object detection. [13]

#### **Computational Complexity [12]**

##### **Memory Usage**

**GPU Memory:** VGG-19 has approximately 143.67 million parameters, requiring substantial GPU memory to store weights and intermediate activations. This often limits batch size during training.

**Storage Requirements:** The large model size demands significant storage space, posing challenges in resource-constrained environments.

##### **Overfitting**

**Risk of Overfitting:** Due to the high number of parameters, VGG-19 is prone to overfitting, especially on small datasets. Techniques like dropout, L2 regularization, and data augmentation are essential to mitigate this risk.

**Regularization Techniques:** Dropout layers are often added to fully connected layers to reduce overfitting.

##### **Training Time**

**Training Complexity:** Training VGG-19 from scratch is computationally intensive and time-consuming due to the large number of layers and parameters. Using transfer learning with pre-trained weights on ImageNet reduces both time and resources.

**Floating Point Operations (FLOPs):** VGG-19 requires approximately 19.6 billion FLOPs for a single forward pass, contributing to its high computational demand.

#### **Comparisons with ResNet and EfficientNet**

* + **VGG-19 vs. ResNet:** ResNet introduced residual connections that solve the vanishing gradient problem prevalent in deeper networks like VGG-19. ResNet-50 has significantly fewer parameters than VGG-19 (approximately 25.6 million vs. 143.67 million) while achieving superior accuracy. The use of identity mappings in ResNet allows deeper networks without performance degradation.
  + **VGG-19 vs. EfficientNet:** EfficientNet scales depth, width, and resolution systematically, optimizing performance with fewer parameters compared to VGG-19. For example, EfficientNet-B0 achieves comparable or better accuracy than VGG-19 with far fewer FLOPs and a significantly smaller model size, making it ideal for deployment in resource-constrained environments.

#### **Variants of VGG**

* **VGG-11, VGG-13, and VGG-16:** These are shallower variants of the VGG family. The primary difference among them is the number of convolutional layers, with VGG-19 having the deepest architecture.
* **VGG-C and VGG-D:** These models explore variations in convolutional filter arrangements to optimize performance.

### **EfficentNet-B2:**

#### **Introduction: [14]**

EfficientNet is a family of convolutional neural networks developed by Google AI, known for its scalability and efficiency. EfficientNet-B2 is a model in the EfficientNet family, designed to provide a good balance between accuracy and computational efficiency. It uses a compound scaling method to balance network depth, width, and resolution, resulting in a smaller and faster model compared to traditional CNNs like VGG-19 while maintaining high performance.

#### **EfficientNet-B2 Architecture (figure 13) [15]**

EfficientNet-B2 follows the principles of compound scaling and utilizes a combination of standard convolutional layers, depth wise separable convolutions, and the swish activation function. The architecture of EfficientNet-B2 is optimized for better performance with fewer parameters compared to previous CNN architectures

Key components of the EfficientNet-B2 architecture include:

* **Depthwise Separable Convolutions**: These are used to reduce the computational load by separating the filtering and channel mixing operation.
* **Swish Activation**: Swish is used instead of ReLU, which has been shown to improve performance in deeper networks.
* **MBConv Blocks**: EfficientNet-B2 uses mobile inverted bottleneck convolutions (MBConv), a lightweight convolution operation designed to improve efficiency.
* **Compound Scaling**: EfficientNet scales network dimensions (depth, width, and resolution) in a balanced way, which optimizes performance and computational requirements.

#### **Layer-by-Layer Breakdown of EfficientNet-B2 [15]**

The architecture of EfficientNet-B2 consists of several stages, each composed of MBConv blocks. Here is a detailed breakdown of the layers:

* **Input**: 260x260x3 (height, width, RGB channels)
* **Stem Layer**:
  + Conv1: 32 filters, 3x3 kernel, stride 2, Swish activation
  + Conv2: 16 filters, 3x3 kernel, stride 1, Swish activation
* **MBConv Blocks** (based on depthwise separable convolutions):
  + Stage 1: 1 MBConv block with 16 filters, 3x3 kernel, Swish activation, depthwise separable convolutions
  + Stage 2: 2 MBConv blocks with 24 filters, 3x3 kernel, Swish activation
  + Stage 3: 3 MBConv blocks with 40 filters, 3x3 kernel, Swish activation
  + Stage 4: 3 MBConv blocks with 80 filters, 3x3 kernel, Swish activation
  + Stage 5: 3 MBConv blocks with 112 filters, 3x3 kernel, Swish activation
  + Stage 6: 4 MBConv blocks with 192 filters, 3x3 kernel, Swish activation
  + Stage 7: 1 MBConv block with 320 filters, 3x3 kernel, Swish activation
* **Final Convolution Layer**:
  + Conv: 1280 filters, 1x1 kernel, Swish activation
* **Fully Connected Layer**:
  + A diagram of a flowchart

    Description automatically generatedDense: Output size is typically 1000 for ImageNet classification (modifiable for other tasks)

***Figure 13*:** EffientNet-B2 Architecture

#### **Use in Transfer Learning**

##### **Overview of Transfer Learning with EfficientNet-B2**

EfficientNet-B2, like other models in the EfficientNet family, excels in transfer learning by leveraging pre-trained weights (usually from ImageNet) to solve domain-specific tasks with limited data. Transfer learning reduces computational costs and improves performance, particularly in tasks like image classification, medical imaging, and object detection. The model’s efficient architecture, which scales depth, width, and resolution, enables it to transfer learned features effectively across various domain.

##### **Fine-Tuning EfficientNet-B2 [13]**

Fine-tuning involves updating the model for a new task while retaining the pre-trained features. The process typically includes replacing the final fully connected (FC) layer with a task-specific output layer and freezing the earlier convolutional layers.

**Frozen Layers:**

* **Lower Convolutional Layers**: These layers, capturing generic features like edges and textures, are frozen. They retain their ability to generalize across tasks, reducing the computational load and minimizing overfitting.

**Fine-Tuned Layers:**

* **Higher Convolutional Layers**: These layers, which capture more complex features, are typically fine-tuned to adjust the model for the specific task.
* **Fully Connected (FC) Layer**: Replaced with a new output layer corresponding to the number of classes in the new dataset (e.g., two classes for binary classification).

##### **Training Strategy [13]**

* **Learning Rate Adjustment**: Fine-tuned layers use a lower learning rate (e.g., 1e-4) to avoid large weight changes, while newly added layers use a higher rate (e.g., 1e-3) to adapt quickly.
* **Regularization**: Dropout is applied to the FC layers to reduce overfitting, especially with smaller datasets.
* **Batch Normalization**: Can be frozen or fine-tuned depending on the task.

#### **Attention Mechanism in EfficientNet [16]**

EfficientNet models, including EfficientNet-B2, integrate **Squeeze-and-Excitation (SE) blocks**, a lightweight attention mechanism designed to enhance the model’s ability to focus on important features. The SE block recalibrates channel-wise feature responses, helping the network emphasize relevant features while suppressing irrelevant ones, leading to improved performance in tasks like image classification and object detection.

#### **How the SE Block Works [16]**

The SE block operates in three steps:

1. **Squeeze**: It uses global average pooling to compress each feature map into a channel descriptor, capturing global dependencies across channels.
2. **Excite**: This descriptor is passed through a small fully connected network (two layers) to produce channel-wise attention weights that reflect the importance of each channel.
3. **Recalibrate**: The generated attention weights are then multiplied by the original feature map to scale the channels, enhancing important features and diminishing less relevant ones.

This mechanism allows the model to focus more on significant features without adding much computational overhead, thereby improving its representational power (Hu et al., 2018). EfficientNet-B2 benefits from this approach by recalibrating features at each processing stage, improving its ability to solve complex tasks efficiently.

#### **Computational Complexity**

EfficientNet-B2 is designed to be highly efficient compared to traditional CNNs like VGG-19, making it suitable for resource-constrained environments:

* **Memory Usage**: With 8.1 million parameters, EfficientNet-B2 is more compact than VGG-19, which has 143.67 million parameters.
* **Training Time**: The model’s efficiency means that it requires less time to train, especially when using transfer learning.
* **FLOPs**: EfficientNet-B2 requires approximately 1.4 billion floating-point operations (FLOPs) per forward pass, which is considerably fewer than VGG-19’s 19.6 billion FLOP.

#### **Comparisons with VGG-19 and ResNet:**

* **EfficientNet-B2 vs. VGG-19**: EfficientNet-B2 outperforms VGG-19 with far fewer parameters and operations, achieving similar or superior accuracy while requiring less computational resources. VGG-19’s architecture is much deeper, but the use of depthwise separable convolutions and compound scaling in EfficientNet-B2 results in higher efficiency.
* **EfficientNet-B2 vs. ResNet**: EfficientNet-B2 also compares favorably with ResNet models. While ResNet utilizes residual connections to enable very deep networks, EfficientNet-B2 balances depth, width, and resolution systematically, achieving superior performance with fewer parameters and FLOPs.

#### **Variants of EfficientNet [15]**

The EfficientNet family includes models that scale in terms of depth, width, and resolution. EfficientNet-B2 is part of this family and strikes a balance between computational efficiency and model accuracy. Other variants of EfficientNet include:

* **EfficientNet-B0 to B7**: These models scale the depth, width, and resolution of the network progressively, with EfficientNet-B0 being the smallest and EfficientNet-B7 the largest.
* **EfficientNet-Lite**: A smaller, more efficient version of EfficientNet designed for mobile and edge devices.

### **ResNet-101:**

#### **Introduction [17]**

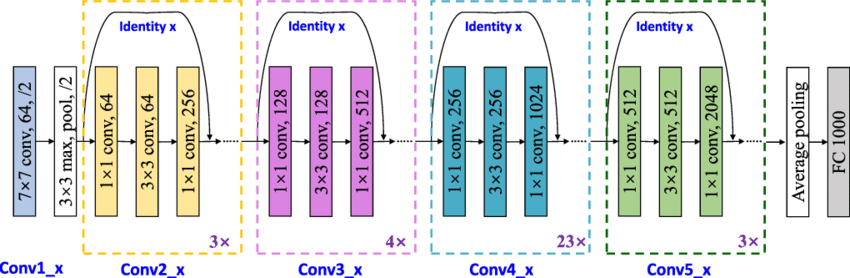
ResNet-101, short for Residual Network with 101 layers, is a deep convolutional neural network designed to address the vanishing gradient problem in very deep networks. Developed by Kaiming He et al., ResNet introduces residual connections, allowing models to train deeper architectures effectively while maintaining high accuracy. ResNet-101 is part of the ResNet family and is renowned for its robustness and performance in tasks like image classification, object detection, and segmentation.

#### **ResNet-101 Architecture (figure 14) [17]**

The ResNet-101 architecture is built using residual blocks, which consist of shortcut connections that bypass one or more layers. This design helps in efficiently training deep networks by enabling gradients to flow directly through the shortcut paths, mitigating the degradation problem.

#### **Key Components: [17]**

* **Residual Blocks:** Allow the model to learn residual functions instead of direct mappings, facilitating the training of deeper networks.
* **Batch Normalization (BN):** Applied after each convolutional layer to stabilize training and accelerate convergence.
* **ReLU Activation:** Used after each BN layer for non-linearity.
* **Bottleneck Design:** In deeper ResNet models like ResNet-101, bottleneck blocks are employed to reduce computational complexity by decreasing and then restoring feature dimensions within blocks.



***Figure 14*:** ResNet-101 Architecture

#### **Layer-by-Layer Breakdown of ResNet-101 (figure 14) [17]**

The ResNet-101 architecture consists of several stages, each comprising residual blocks:

* **Input:** 224x224 image resolution (default input size).
* **Initial Convolution Layer:**
  + Conv1: 64 filters, 7x7 kernel, stride 2, followed by BN, ReLU, and a 3x3 max pooling layer with stride 2.
* **Residual Blocks:**
  + **Stage 1:** 3 bottleneck blocks with 256 filters each (1x1, 3x3, 1x1 kernels).
  + **Stage 2:** 4 bottleneck blocks with 512 filters each.
  + **Stage 3:** 23 bottleneck blocks with 1024 filters each.
  + **Stage 4:** 3 bottleneck blocks with 2048 filters each.
* **Final Layers:**
  + Global Average Pooling (GAP): Reduces the spatial dimensions to 1x1.
  + Fully Connected (FC) Layer: Output size typically 1000 for ImageNet classification (modifiable for other tasks).

#### **Use in Transfer Learning**

##### **Overview of Transfer Learning with ResNet-101:**

ResNet-101 is widely used in transfer learning due to its deep architecture and pre-trained weights (usually from ImageNet). By leveraging pre-trained features, ResNet-101 can be adapted to various domain-specific tasks with limited labeled data. Applications include image classification, medical imaging, and semantic segmentation.

##### **Fine-Tuning ResNet-101: [18]**

**Fine-tuning involves adapting the model to a new task by:**

* Replacing the final FC layer with a task-specific output layer.
* Freezing earlier layers to retain pre-trained features while fine-tuning higher layers.

#### **Frozen Layers:**

* **Lower Layers:** Capture generic features like edges and textures. These are usually frozen to prevent overfitting and reduce computational costs.

#### **Fine-Tuned Layers:**

* **Higher Layers:** Capture task-specific features. Fine-tuning these layers helps the model adapt to the new dataset.
* **FC Layer:** Replaced with a new layer corresponding to the number of classes in the target dataset.

##### **Training Strategy [18]**

* **Learning Rate Adjustment:** Lower learning rate for fine-tuned layers (e.g., 1e-4) and higher learning rate for newly added layers (e.g., 1e-3).
* **Regularization:** Dropout is applied to reduce overfitting.
* **Data Augmentation:** Techniques like rotation, flipping, and scaling are used to increase dataset diversity.

#### **Computational Complexity**

ResNet-101 achieves a balance between depth and computational efficiency, making it suitable for resource-intensive tasks:

* **Parameters:** ResNet-101 has approximately 44.5 million parameters, significantly fewer than traditional CNNs like VGG-19 (143.67 million parameters) while achieving superior accuracy.
* **FLOPs:** The model requires around 7.6 billion FLOPs per forward pass, which is higher than EfficientNet-B2 but justified by its increased depth and performance.

#### **Comparisons with Other Architectures**

* **ResNet-101 vs. VGG-19:** ResNet-101 significantly outperforms VGG-19 in terms of accuracy while requiring fewer parameters and addressing the vanishing gradient problem through residual connections.
* **ResNet-101 vs. EfficientNet-B2:** ResNet-101 has more parameters and higher computational requirements than EfficientNet-B2 but offers greater flexibility for transfer learning in tasks requiring deep feature hierarchies.

#### **Variants of ResNet [17]**

The ResNet family includes various architectures, each tailored to specific use cases:

* **ResNet-18 and ResNet-34:** Shallow models suitable for low-resource environments.
* **ResNet-50, ResNet-101, and ResNet-152:** Deeper models designed for high-performance tasks.
* **ResNeXt:** An extension of ResNet incorporating grouped convolutions for enhanced performance.

## **Proposed Method & Research Plan**

### **Preprocessing**

An essential part of our research is the preprocessing of laryngoscopy videos. We have designed an efficient preprocessing pipeline to prepare the data for our models, ensuring the best performance in selecting frames. The preprocessing steps are as follows:

1. **Frame Extraction**:  
   We will utilize a reliable third-party software to break the input videos into individual frames. This step is vital as it converts continuous video data into distinct images, making them easier to process and analyze independently. Using established software ensures that the frame extraction process is both dependable and efficient.
2. **Content Focusing**:  
   Each extracted frame will undergo additional processing to crop it, focusing on the relevant content. This step eliminates unnecessary background information and highlights the key anatomical structures, streamlining the analyzis to the most important features.
3. **Data Augmentation**:  
   To improve the robustness and generalization of our model, we will apply various data augmentation techniques to the frames. These augmentations may include rotating, flipping, zooming, or other transformations suitable for laryngoscopy images. This helps the model better adapt to variations in the data.
4. **Database Storage**:  
   The final step in the preprocessing pipeline involves saving the processed frames into a database, along with important metadata and labels. Our database is structured to facilitate efficient data retrieval and support flexible experimentation:

* **Frame Identification**: Each frame will be assigned a unique identifier (ID), making it easy to distinguish it in the database.
* **Video Source**: Information about the source video for each frame will be stored, allowing us to trace each frame back to its original video.
* **Experimental Labels**: The database will support multiple labeling schemes, allowing us to track various experimental conditions or classification tasks. Each frame can have several labels depending on the experiment.
* **Dynamic Labeling**: The labeling process is integrated into the experimental workflow, meaning we can update or add labels to frames as needed. This allows the dataset to evolve and improve throughout our research.
* **Experiment Tracking**: Metadata for each experiment will be maintained, including the labeling criteria, so that we can filter and retrieve specific data subsets based on the experimental context.

This carefully structured preprocessing pipeline ensures that the input data is consistently prepared, focused on relevant content, and enhanced through augmentation. By following these steps, we aim to create a high-quality dataset that will enable our models to effectively select the most appropriate frames from laryngoscopy videos.

These preprocessing procedures will be thoroughly implemented and documented to ensure reproducibility and enable further refinement as our research progresses.

## **Expected Challenges**

We expect to encounter the following challenges during the research and implementation of our project:

1. **Working with Real Data**: Real laryngoscopy videos have variability in quality, lighting, and patient movement, requiring robust preprocessing to ensure accurate frame selection.
2. **Limited Computing Power**: Analysing high-resolution videos as well as training our models demand significant computational resources, requiring optimized processing pipelines for efficient frame analyzis.
3. **Challenges in Dataset Construction**: Constructing a comprehensive dataset involves managing relationships between frames, sequences, and metadata while ensuring scalability and flexibility.
4. **Manual Labelling of Frames**: labelling frames is time-consuming and requires domain expertise, with automation offering partial relief but still needing manual oversight for accuracy.

We believe that by using our models that has a very good reputation for object detection as well as experimenting with various labelling schemes and different parameters in the suggested algorithmic approach will allow us to successfully address these challenges.

## **Diagrams & Preliminary Software Engineering Documents**

### **Flow Charts:**

#### **Training model flow chart**

A diagram of a video

Description automatically generated

#### **Test model flow chart:**

**For each model:** A diagram of a process

Description automatically generated

## **Test Plan & Experiments**

We use a wide range of scenarios in our extensive test strategy to evaluate the models' performance, system resilience, and UI functionality. Furthermore, to optimize various features of the models, such as layer freezing, data augmentation, transfer learning, class grouping, and model size variations, we have created five important experiments. By contrasting alternative configurations, including frozen versus trainable layers, different data augmentation techniques, transfer learning versus training from scratch, different class grouping procedures, and model sizes, these tests will aid in improving our methodology. We will specifically use these tests on three models: EfficientNet-B2, ResNet-101, and VGG-19. This will enable us to use each of these models to determine the best configuration for our laryngoscopy frame selection task.

### **Test Plan**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Test Area | Objective | Steps to Test | Expected Outcome |
| TP-01 | Video Frame Extraction | Ensure video frames are extracted properly. | 1. Input a sample video. 2. Set frame rate (e.g., 5 fps). 3. Execute frame extraction tool. | Frames are extracted and saved correctly with the specified frame rate. |
| TP-02 | Frame Preprocessing | Remove black borders around frames. | 1. Input extracted frames with black borders. 2. Apply cropping logic. | Frames are cropped accurately, with no black borders remaining. |
| TP-03 | Data Augmentation | Verify rotation augmentation works (±15°). | 1. Input sample frames. 2. Apply rotation augmentation (±15°). | Frames are correctly rotated by the specified degree, preserving quality. |
| TP-04 | Data Augmentation | Verify stretching augmentation works. | 1. Input sample frames. 2. Apply stretching logic with predefined parameters. | Frames are stretched proportionally without distortion artifacts. |
| TP-05 | Data Augmentation | Verify noise addition augmentation works. | 1. Input sample frames. 2. Add Gaussian or salt-and-pepper noise using the augmentation pipeline. | Noise is applied consistently and does not degrade the frame beyond recognition. |
| TP-06 | Data Augmentation | Verify sharpening augmentation works. | 1. Input sample frames. 2. Apply sharpening using predefined kernel/filter. | Frames are sharpened appropriately without overshooting or introducing artifacts. |
| TP-07 | Data Augmentation | Verify saturation adjustment augmentation works. | 1. Input sample frames. 2. Apply saturation adjustments (increase and decrease saturation). | Saturation levels change as expected, maintaining visual integrity. |

## **6.2 Planned Experiments**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Experiment ID** | **Model** | **Experiment Area** | **Objective** | **Steps to Execute** | **Expected Outcome** |
| **EXP-01** | VGG-19 | Baseline Training | Train VGG-19 from scratch without transfer learning. | 1. Initialize VGG-19 with random weights. 2. Train the model on the dataset. 3. Record accuracy, loss, and other metrics. | Baseline performance metrics for VGG-19 provide a reference for comparison. |
| **EXP-02** | EfficientNet | Baseline Training | Train EfficientNet from scratch without transfer learning. | 1. Initialize EfficientNet with random weights. 2. Train the model on the dataset. 3. Record accuracy, loss, and other metrics. | Baseline performance metrics for EfficientNet provide a reference for comparison. |
| **EXP-03** | ResNet | Baseline Training | Train ResNet from scratch without transfer learning. | 1. Initialize ResNet with random weights. 2. Train the model on the dataset. 3. Record accuracy, loss, and other metrics. | Baseline performance metrics for ResNet provide a reference for comparison. |
| **EXP-04** | VGG-19 | Transfer Learning (Frozen) | Train VGG-19 using transfer learning with all pretrained layers frozen. | 1. Load a pretrained VGG-19 model. 2. Freeze all layers. 3. Add and train a classification head. | Metrics showcase VGG-19's performance with frozen transfer learning. |
| **EXP-05** | EfficientNet | Transfer Learning (Frozen) | Train EfficientNet using transfer learning with all pretrained layers frozen. | 1. Load a pretrained EfficientNet model. 2. Freeze all layers. 3. Add and train a classification head. | Metrics showcase EfficientNet's performance with frozen transfer learning. |
| **EXP-06** | ResNet | Transfer Learning (Frozen) | Train ResNet using transfer learning with all pretrained layers frozen. | 1. Load a pretrained ResNet model. 2. Freeze all layers. 3. Add and train a classification head. | Metrics showcase ResNet's performance with frozen transfer learning. |
| **EXP-07** | VGG-19 | Transfer Learning (Unfreeze) | Train VGG-19 using transfer learning and fine-tune the last few layers. | 1. Load a pretrained VGG-19 model. 2. Freeze initial layers. 3. Unfreeze the last few layers. 4. Train and validate the model. | Fine-tuned VGG-19 achieves improved performance compared to the frozen version. |
| **EXP-08** | EfficientNet | Transfer Learning (Unfreeze) | Train EfficientNet using transfer learning and fine-tune the last few layers. | 1. Load a pretrained EfficientNet model. 2. Freeze initial layers. 3. Unfreeze the last few layers. 4. Train and validate the model. | Fine-tuned EfficientNet achieves improved performance compared to the frozen version. |
| **EXP-09** | ResNet | Transfer Learning (Unfreeze) | Train ResNet using transfer learning and fine-tune the last few layers. | 1. Load a pretrained ResNet model. 2. Freeze initial layers. 3. Unfreeze the last few layers. 4. Train and validate the model. | Fine-tuned ResNet achieves improved performance compared to the frozen version. |
| **EXP-10** | General | Model Comparison | Compare all three models' performance after transfer learning. | 1. Train VGG-19, EfficientNet, and ResNet using transfer learning. 2. Evaluate on the same dataset. | Comparison reveals which model performs best for the task. |
| **EXP-11** | General | Generalization Testing | Evaluate the ability of all three models to generalize to unseen data. | 1. Train each model on the main dataset. 2. Test on an unseen dataset with similar symptoms. | Metrics indicate the generalization capability of each model. |
| **EXP-12** | VGG-19 | Layer-Freezing Strategies | Explore different layer-freezing strategies for VGG-19. | 1. Freeze/unfreeze specific layers (e.g., first 5 or last 5). 2. Train and validate. 3. Compare metrics. | Identifies the optimal layer-freezing strategy for VGG-19. |
| **EXP-13** | EfficientNet | Layer-Freezing Strategies | Explore different layer-freezing strategies for EfficientNet. | 1. Freeze/unfreeze specific layers (e.g., first 5 or last 5). 2. Train and validate. 3. Compare metrics. | Identifies the optimal layer-freezing strategy for EfficientNet. |
| **EXP-14** | ResNet | Layer-Freezing Strategies | Explore different layer-freezing strategies for ResNet. | 1. Freeze/unfreeze specific layers (e.g., first 5 or last 5). 2. Train and validate. 3. Compare metrics. | Identifies the optimal layer-freezing strategy for ResNet. |
| **EXP-15** | General | Final Evaluation (Accuracy) | Evaluate the final accuracy of all models on the test dataset. | 1. Train each model using the best configuration.  2. Test each model on the test dataset. | Provides the final accuracy metrics for VGG-19, EfficientNet, and ResNet. |
| **EXP-16**   |  | | --- | |  |   **EXP-16** | |  | | --- | | General |  |  | | --- | |  | | Final Evaluation (Inference Speed) | Compare the inference speed of all models on the test dataset. | 1. Test the inference time of each model on the test dataset using a consistent hardware setup. | Reveals the inference speed differences between VGG-19, EfficientNet, and ResNet. |

## **Expected Results & Future Work**

### **Expected Results**

As we advance in our research on developing a computer system capable of diagnosing Laryngopharyngeal Reflux signs with high accuracy, we anticipate several key outcomes:

1. **Efficient Processing**: Our system should efficiently process laryngoscopy videos by extracting relevant frames and discarding irrelevant ones, then running these frames through the computer system to output results in near real-time. This will substantially reduce the time required for manual review and diagnosis by doctors.
2. **Enhanced LPR Signs Diagnosis Accuracy**: While doctors currently diagnose LPR signs accurately in 80% of cases, our goal is to surpass this accuracy and achieve even better results in identifying Laryngopharyngeal Reflux signs.

### **Future Work**

Building upon our current research, we have identified three key directions for future work:

1. **Continuous Model Improvement**: Implement a system for ongoing model evaluation and refinement, allowing for continuous updates based on new patient data and feedback from medical professionals. This would ensure that the system adapts to emerging trends and improves its diagnostic accuracy over time.
2. **Real-time Diagnostic Assistance**: Explore the integration of real-time diagnostic tools within the laryngoscopy procedure. This would provide immediate feedback to doctors, aiding in quicker and more accurate decision-making during patient assessments.
3. **Expansion to Other Reflux Conditions**: Investigate the potential to adapt the system for diagnosing other types of reflux conditions, such as gastroesophageal reflux disease (GERD), which could broaden the clinical utility of the technology and enhance its overall impact.

These future directions aim to further improve the diagnostic capabilities of our system and extend its applicability in clinical settings, ultimately benefiting both healthcare providers and patients.

# **8. Bibliography**

**Resources:**

1. <https://www.mdpi.com/2075-4418/13/9/1643>
2. <https://www.researchgate.net/publication/343457894_Changes_of_Laryngeal_and_Extralaryngeal_Symptoms_and_Findings_in_Laryngopharyngeal_Reflux_Patients>
3. <https://www.geeksforgeeks.org/deep-learning-tutorial/>
4. <https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/>
5. <https://www.geeksforgeeks.org/introduction-convolution-neural-network/>
6. <https://www.geeksforgeeks.org/what-is-forward-propagation-in-neural-networks/>
7. <https://www.geeksforgeeks.org/backpropagation-in-neural-network/>
8. <https://docs.mosaicml.com/projects/composer/en/latest/method_cards/layer_freezing.html#:~:text=Layer%20Freezing%20gradually%20makes%20early,to%20and%20updating%20frozen%20modules>
9. <https://www.geeksforgeeks.org/what-is-data-augmentation-how-does-data-augmentation-work-for-images/>
10. <https://www.geeksforgeeks.org/ml-introduction-to-transfer-learning/>
11. <https://www.devopsschool.com/blog/what-is-transfer-learning/#Types_of_Transfer_Learning>
12. <https://www.geeksforgeeks.org/vgg-net-architecture-explained/>
13. <https://learnopencv.com/fine-tuning-pre-trained-models-tensorflow-keras/>
14. <https://www.geeksforgeeks.org/efficientnet-architecture/>
15. <https://medium.com/towards-data-science/complete-architectural-details-of-all-efficientnet-models-5fd5b736142>
16. <https://arxiv.org/pdf/1709.01507>
17. <https://www.researchgate.net/publication/372853599_Abnormal_event_detection_model_using_an_improved_ResNet101_in_context_aware_surveillance_system>
18. <https://pyimagesearch.com/2020/04/27/fine-tuning-resnet-with-keras-tensorflow-and-deep-learning/>