

**Lung Cancer Detection**

**50.039 Deep Learning Project Report**

GitHub: <https://github.com/zhangjianyu1006156/50.039-DeepLearning/tree/main>

**Group 13 Team members:**

Mehta Yash Piyush, 1006516

Zhang JianYu, 1006156

Zachary Seow, 1005979

**Table of Contents**

[1. Introduction 3](#_Toc163999842)

[2.Problem Definition and Algorithm 4](#_Toc163999843)

[2.1 Task Definition 4](#_Toc163999844)

[2.2 Dataset 5](#_Toc163999845)

[2.2.1 About Dataset 5](#_Toc163999846)

[2.2.2 Challenges and Considerations in Dataset Utilization 6](#_Toc163999847)

[2.3 Models 7](#_Toc163999848)

[2.3.1 Loss Function 8](#_Toc163999849)

[2.3.2 Optimizer 9](#_Toc163999850)

[2.3.3 Learning Rate Scheduler 12](#_Toc163999851)

[2.4 Training 12](#_Toc163999852)

[3. Experimental Evaluation – Custom Models 13](#_Toc163999853)

[3.1 Methodology 13](#_Toc163999854)

[3.2 Multi-layer Perceptron (MLP) 14](#_Toc163999855)

[3.2.1 Model Definition 14](#_Toc163999856)

[3.2.2 Experiment Results 15](#_Toc163999857)

[3.2.3 Experimental Findings 16](#_Toc163999858)

[3.3 Convoluted Neural Network (CNN) 17](#_Toc163999859)

[3.3.1 Model Definition 17](#_Toc163999860)

[3.3.2 Experiment Results 18](#_Toc163999861)

[3.3.3 Experimental Findings 20](#_Toc163999862)

[3.4 CNN with LSTM 21](#_Toc163999863)

[3.4.1 Model Definition 21](#_Toc163999864)

[3.4.2 Experiment Results 22](#_Toc163999865)

[3.4.3 Experiment Findings 22](#_Toc163999866)

[4. Experimental Evaluation – Custom Models 22](#_Toc163999867)

[5. Future Work 22](#_Toc163999868)

[5.1 Enhancing the Dataset 22](#_Toc163999869)

[5.2 Extending Model Capabilities 23](#_Toc163999870)

[5.3 Application Development 23](#_Toc163999871)

[6. Conclusion 24](#_Toc163999872)

[7. Acknowledgements 24](#_Toc163999873)

[8. Reference 24](#_Toc163999874)

[9. Appendix 25](#_Toc163999875)

[Other experiments for CNN that was conducted were as follows: 25](#_Toc163999876)

# 1. Introduction

Lung cancer is the leading cause of cancer-related deaths worldwide, accounting for almost one-fifth of all cancer deaths. Late-stage diagnoses often result in limited treatment options and poor prognosis. Early detection and intervention are crucial in mitigating its impact on public health. Therefore, it is imperative to prioritize early detection and intervention.

In recent years, lung cancer research has experienced a significant transformation due to advancements in medical imaging and computational methodologies. Specifically, the emergence of deep learning techniques has renewed hope for more precise and timely diagnostic methods.

Several studies have highlighted the potential of artificial intelligence in transforming lung cancer screening and management. For example, Ardila et al. (2019) demonstrated the effectiveness of a deep learning model in analyzing chest radiographs to detect pulmonary nodules, achieving performance comparable to that of radiologists. Likewise, Hu et al. (2020) showed the usefulness of convolutional neural networks (CNNs) in identifying lung cancer subtypes and predicting patient survival based on histopathological images.

Research efforts have been focused on using large-scale datasets to train robust deep learning models for lung cancer detection.

Our study aims to contribute to the field of AI-driven lung cancer diagnosis by utilizing the Iraq-Oncology Teaching Hospital/National Center for Cancer Diseases (IQ-OTH/NCCD) lung cancer dataset. Our goal is to explore the potential of AI in distinguishing between benign and malignant pulmonary lesions using state-of-the-art deep learning architectures. This will help expedite diagnosis and enable personalized treatment strategies.

# 2.Problem Definition and Algorithm

## 2.1 Task Definition

Formally, let our LUNG-CANCER dataset be , where is an input chest CT image and is the label indicating the tumor type (benign, malignant, or normal), and is the total number of samples in . Without preprocessing, is a high-resolution image and is a categorical string.

Preprocessing Step:

In our preprocessing step, we transform by applying a series of image processing techniques to prepare the data for effective analysis by a convolutional neural network. The transformation steps include:

1. Resizing: All images are resized to a uniform dimension of pixels to ensure consistency in input size for the CNN.

2. Normalization: Images are normalized using mean values of and standard deviation values of . This normalization is based on the commonly used ImageNet values and helps in stabilizing the learning process by adjusting pixel intensity distributions.

3. Data Augmentation:

- Horizontal Flip: Each image has a 100% probability of being flipped horizontally. This augmentation increases the dataset's variability and helps the model generalize better by simulating different viewing angles.

- Random Rotation: Images are randomly rotated within a range of degrees. This introduces a variety of angles, mimicking real-world scenarios where CT scans might not always be perfectly aligned.

4. Conversion to Tensor: The images are converted to tensor format which is a suitable input type for PyTorch models.

We also encode the labels [](https://www.codecogs.com/eqnedit.php?latex=%5C%20y_i%20%5C#0) with integer values ranging from 0 to , where is the total number of label categories (three in this case: benign, malignant, normal). This encoding transforms categorical labels into a format that can be processed by the neural network during training.

After preprocessing, suppose our input and output spaces are and respectively. Thus, our machine learning task can be described as finding a mapping (hypothesis) , where is a family of parameterized functions represented by a convolutional neural network.

In this deep learning project, we focus on utilizing CNN-based methods due to their proven efficacy in feature extraction and pattern recognition in image data. The goal is to develop custom models that accurately classifies the type of tumor in a non-invasive manner, leveraging the inherent capabilities of deep learning to enhance early detection and diagnosis of lung cancer. This will then be compared with state-of-the-art models. Our approach seamlessly integrates theoretical principles with practical experimentation, as we explore various hyperparameter combinations informed by theoretical insights.

## 2.2 Dataset

### 2.2.1 About Dataset

The Iraq-Oncology Teaching Hospital/National Center for Cancer Diseases (IQ-OTH/NCCD) lung cancer dataset is a meticulously curated resource fundamental to advancing the landscape of lung cancer diagnostics. Acquired over a rigorously controlled three-month period in fall 2019 from specialized medical centers, this dataset comprises a comprehensive collection of CT scan images obtained from patients diagnosed with varying stages of lung cancer, alongside individuals exhibiting healthy pulmonary profiles.

This dataset encompasses 1,190 CT scan slices, meticulously categorized into 110 distinct cases, thereby providing a rich repository for academic inquiry and algorithmic exploration. Each case within the dataset comprises an extensive array of CT scan slices, ranging from 80 to 200 slices per case.

These cases are meticulously stratified into three distinct classes, delineating the multifaceted spectrum of pulmonary pathology encountered in clinical practice:

1. **Benign Cases (Class 0):** These cases encapsulate non-cancerous anomalies or growths within the lung tissue, including benign tumors, cysts, or other non-malignant lesions. Such cases serve as crucial benchmarks for differentiating between benign and malignant conditions, thereby informing clinical decision-making processes.
2. **Malignant Cases (Class 1):** Constituting the focal point of pathological interest, malignant cases signify the presence of cancerous growths or tumors within the lung tissue. These lesions exhibit varying degrees of aggressiveness and metastatic potential, which may necessitate prompt and accurate diagnostic interventions to optimize patient outcomes.
3. **Normal Cases (Class 2):** Serving as a comparative reference, normal cases denote instances where the lung tissue appears devoid of any discernible abnormalities or lesions.

Furthermore, the dataset embodies considerable demographic diversity, reflecting disparities in gender, age, educational attainment, and geographical origin among its constituent cases.

### 2.2.2 Challenges and Considerations in Dataset Utilization

In harnessing the Iraq-Oncology Teaching Hospital/National Center for Cancer Diseases (IQ-OTH/NCCD) lung cancer dataset for deep learning-based lung cancer detection, several inherent challenges necessitate careful consideration to ensure robust model training and evaluation. Herein, we outline key challenges encountered during dataset utilization, proposed solution approaches, and analogical explanations elucidating these concepts.

**Challenges:**

1. **Interdependence of Slices within a Case:** The dataset exhibits interdependence among slices within the same case, rendering them non-independent observations. This interdependence arises from the spatial continuity of CT scan slices within a single patient's case, leading to high correlation among slices.
2. **Variability in Slices per Case:** Another salient challenge arises from the significant variability in the number of slices per case. This variability introduces complexities in stratifying the dataset by class label at the slice level, potentially compromising the representativeness of case distribution across data splits.
3. **Not Enough Representation of a particular class/Lack of sufficient data:** As we will see later, most of the cases are either Malignant or Normal. There are very few examples of the in-between Benign case. This prevents our various models from obtaining higher accuracy as it has very little information to learn from and is unable to distinguish Benign (Class 0) from Malignant (Class 1).

**Considerations for Code Implementation:**

1. **Stratification Assumptions:** The stratification approach we have utilized in our train, test and split functions assumes independence between slices, necessitating caution in its application given the interdependence inherent within cases. This is an assumption we had to make when training and testing the dataset.
2. **Data Leakage:** Unfortunately, due to lack of additional metadata information, slices from the same case could possibly inadvertently infiltrate both training and test/validation sets, potentially inflating performance metrics. This could provide a false sense of the model's true predicting power.

**Potential workaround:**

A possible workaround could be further refining the dataset, for example, by integrating metadata linking slices to their respective cases. This would lead the way to more advanced stratification or case-level splitting methodologies. An advanced stratification methodology that would aim to balance class proportions and the number of slices contributed by each case across data splits. Moreover, a case-level splitting strategy involving partitioning cases into training, validation, and testing sets prior to considering individual slices would prevent data leakage and preserve the integrity of the dataset's distribution.

## 2.3 Models

For this project, our focus has centered on the development and exploration of two straightforward yet meticulously crafted custom models. Our intention was to cultivate a deep understanding of the underlying mechanisms while judiciously fine-tuning hyperparameters for optimal performance. These models comprise a Convolutional Neural Network (CNN) and a Multi-Layer Perceptron.

These custom models have been compared against a spectrum of sophisticated, state-of-the-art architectures, including ResNet-152, VGG-19, DenseNet-161, MobileNetV3-Large, Wide ResNet-101-2, AlexNet, and GoogleNet. The selection criteria for these models were grounded in their exceptional performance metrics, such as Acc@1 and Acc@5 scores on the renowned ImageNet database, as well as their widespread adoption and extensive research in tackling multi-label image classification challenges.

|  |  |  |
| --- | --- | --- |
| Sr | Model Name | Model Type |
| 1 | Convolutional Neural Network (CNN) | Custom |
| 2 | Multi-Layer Perceptron (MLP) | Custom |
| 3 | Convolutional Neural Network (CNN) + LSTM (Long-Short Term Memory) | Custom |
| 4 | ResNet-152 | State-of-the-art |
| 5 | VGG-19 | State-of-the-art |
| 6 | DenseNet-161 | State-of-the-art |
| 7 | MobileNetV3-Large | State-of-the-art |
| 8 | Wide ResNet-101-2 | State-of-the-art |
| 9 | GoogleNet | State-of-the-art |

### 2.3.1 Loss Function

In tackling the multi-class classification problem inherent in lung cancer detection, careful consideration of the loss function and optimizer is pivotal to achieving optimal model performance. Herein, we discuss our rationale behind the selection of the cross-entropy loss function and the Adam optimizer, elucidating their efficacy in facilitating model convergence and enhancing predictive accuracy.

**Loss Function:**

Given the categorical nature of our classification task, wherein cases are categorized into one of three classes (normal, benign, malignant), the **cross-entropy loss function** emerges as a natural choice. Cross-entropy loss, also known as log loss, quantifies the disparity between predicted class probabilities and ground truth labels. By penalizing deviations from the true class distribution, cross-entropy loss incentivizes the model to assign higher probabilities to the correct class, thereby optimizing classification performance (Goodfellow et al., 2016). Cross-entropy loss increases as the predicted probability diverges from the actual label, making it an effective measure for our three-class classification (benign, malignant, normal).

The formulation of cross-entropy loss for multi-class classification is expressed as:

Where:

Two other loss functions have been explored as well in our LossHelper.py file. The first is the **weighted cross-entropy loss**.

The weighted cross-entropy loss is an extension of the standard cross-entropy loss, where different weights are assigned to different classes or samples. This is useful when there is class imbalance in the dataset, and we want to assign higher importance to certain classes or samples.

Where:

By assigning higher weights to these classes, we ensure that the model pays more attention to them during training.

Another loss function we have explored is the **multi-margin loss**. This equation aims to maximize the margin between the correct class and the other classes.

Where:

is the margin parameter

### 2.3.2 Optimizer

To facilitate efficient model training and convergence, we have mainly employed the **Adam optimizer** due to its effectiveness in handling sparse gradients and its adaptability to different data scales, which is crucial for image-based tasks. Adam combines the advantages of two other extensions of stochastic gradient descent, namely Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp), by maintaining separate adaptive learning rates for each parameter and incorporating momentum to accelerate convergence (Kingma & Ba, 2014).

The adaptive nature of the Adam optimizer enables it to adjust learning rates dynamically based on the gradients of individual parameters, thereby mitigating issues associated with fixed learning rates and enabling faster convergence in the presence of sparse gradients or non-stationary objectives. Furthermore, the incorporation of momentum enhances optimization by accelerating parameter updates in the direction of the gradient, thereby facilitating smoother convergence and reducing the likelihood of getting stuck in local minima (Kingma & Ba, 2014).

Where:

: First moment estimate, which is the exponentially decaying average of past gradients. It is similar to the momentum term in other optimization algorithms.

: The second moment estimate, which is the exponentially decaying average of past squared gradients. It helps to adjust the learning rates for different parameters adaptively.

: Bias-corrected first moment estimate. It compensates for the fact that is biased towards zero, especially during the initial time steps.

: Bias-corrected second moment estimate. Similar to it corrects the bias in due to initialization at zero.

: The parameters (weights) being optimized at iteration .

: The gradient of the loss function with respect to the parameters at iteration t.

: The learning rate, which controls the step size in parameter updates.

and : Exponential decay rates for the first and second moment estimates, respectively. These values typically range between 0 and 1, with common choices being close to 1 (e.g., 0.9 and 0.999).

: A small constant added to the denominator for numerical stability. It prevents division by zero.

Our implementation of Adam includes the following parameters:

1. **Learning Rate (lr):** This is a tunable hyperparameter used to control the rate at which our model updates its weights in the network. We will determine the best learning rate through experimentation.
2. **Weight Decay (weight\_decay):** Set at , this parameter helps in regularizing and preventing the coefficients from growing too large.
3. **Betas (betas):** The beta parameters control the exponential decay rates of the moving averages of past gradients and squared gradients respectively.
4. **Epsilon (eps):** The epsilon value of is a very small number to prevent any division by zero in the implementation.
5. **Amsgrad:** We enable the amsgrad variant of Adam which provides an alternative computation of the adaptive learning rate, which can lead to better convergence in practice.

Apart from using Adam optimizer, our OptimizerHelper.py also includes other optimizers like AdamW, Stochastic Gradient Descent, Adagrad and RMSProp.

AdamW is an extension of Adam that incorporates weight decay directly into the optimization process. Weight decay is a regularization technique that penalizes large parameter values to prevent overfitting. In AdamW, the weight decay term is added to the parameter update step, effectively decoupling weight decay from the learning rate schedule which is as shown below:

We have used our weight\_decay as 0.01 for AdamW. This version of the Adam optimizer will work better with large neural networks and for efficiently extracting features from complex datasets.

Stochastic gradient descent is an iterative optimization algorithm used to minimize a loss function by adjusting the parameters of a model. It operates by updating the parameters in the direction of the negative gradient of the loss function with respect to the parameters. It is highly efficient with large datasets, having a faster convergence and is known to escape local minima. Sometimes, it might have a high variance and be highly sensitive to certain hyperparameters which would lead to Adam and its variants or RMSProp. For our implementation of Stochastic Gradient Descent, we have utilized a momentum of 0.9 and a weight decay of 0.0001.

### 2.3.3 Learning Rate Scheduler

Learning Rate Schedulers are techniques used to adjust the learning rate during the training of machine learning models.

Our SchedulerHelper class implements many schedulers as follows:

1. **Step Learning Rate Scheduler** - The learning rate is reduced by a factor (gamma) after a fixed number of epochs (step\_size). We have utilized a factor (gamma) of 0.1 and a step\_size of 5.
2. **Exponential Learning Rate Scheduler** -The learning rate is multiplied by a fixed gamma factor after each epoch. We have utilized a gamma factor of 0.95. This automatically adjusts the learning rate and should also provide a smooth decrease in the learning rate over time.
3. **Cosine Annealing Learning Rate Scheduler –** In Cosine Annealing LR, the learning rate follows a cosine annealing schedule, gradually decreasing from an initial value to a minimum value over a specified number of epochs (T\_max). It not only provides a smooth and gradual decrease in the learning rate, potentially leading to better convergence and generalization. But also, the cosine annealing schedule can help the optimization process navigate complex loss landscapes more effectively, potentially improving the model's ability to find optimal solutions.
4. **Reduce Learning Rate on Plateau Scheduler -** This Learning Rate Scheduler dynamically adjusts the learning rate based on the model's performance, potentially leading to better convergence. This is the main one used throughout all models especially because it automatically adapts to changes in the training dynamics. However, the downside is that it can be sensitive to the choice of monitoring metric and patience.

## 2.4 Training

The provided train\_and\_evaluate function outlines a comprehensive approach to training and evaluating a deep learning model, with several key components tailored to handle a variety of scenarios.

The key components are as follows:

The train\_and\_evaluate function stands as a foundational component within our training framework for deep learning-based lung cancer detection. Its design embodies a comprehensive suite of features essential for orchestrating the training, validation, and evaluation processes of neural network models in a rigorous and methodical manner:

**Dynamic Training and Validation:** This function serves as the engine driving the iterative optimization process of our neural network models. Through meticulous management of training epochs, it facilitates the continual refinement of model parameters, fostering convergence towards an optimal solution and maximizing predictive performance.

**Loss Computation and Optimization:** By employing the cross-entropy loss function, the function quantifies the disparity between predicted and actual class distributions, guiding the optimization process towards minimizing classification errors. Furthermore, the integration of the Adam optimizer ensures the efficient adaptation of learning rates and the incorporation of momentum optimization, enabling swift convergence and robust parameter updates.

**Early Stopping Mechanism:** To safeguard against overfitting and promote model generalization, the function incorporates an early stopping mechanism. This mechanism vigilantly monitors validation performance and halts training when improvements cease, thereby preventing the undue optimization of training data and enhancing model robustness.

**Learning Rate Scheduler:** The function is equipped with a learning rate scheduler, a critical component for stabilizing the training process and facilitating convergence. By dynamically adjusting learning rates based on validation performance, it ensures adaptive optimization and promotes the exploration of diverse model configurations, enhancing overall training efficacy.

**Model Persistence:** Ensuring reproducibility and facilitating model deployment, the function enables the persistent storage of the best-performing model state. This feature not only promotes transparency and reproducibility in research but also facilitates knowledge transfer and collaboration within the scientific community.

# 3. Experimental Evaluation – Custom Models

## 3.1 Methodology

To ensure a comprehensive assessment of the neural network developed for lung cancer detection, we systematically experimented with a variety of hyperparameters and configurations. The objective was to determine the optimal settings that maximize the model's performance in terms of accuracy, efficiency, and generalizability. The methodology adopted for our experimental evaluations is detailed below:

**Optimizer**: Due to time constraints, we simply used Adam as our optimizer to analyse its impact on the convergence rate and final performance of the model.

**Learning Rate**: The learning rate was varied across a predefined range to identify the optimal value that balances the speed of convergence with the stability of the training process. A lower learning rate ensures more stable convergence, while a higher rate can accelerate the training but might lead to overshooting the minimum.

**Dropout Rate**: To mitigate the risk of overfitting, dropout layers were incorporated into the NN architecture. The dropout rate was varied to find the best rate that prevents overfitting while maintaining the model’s ability to generalize from training data to unseen data.

**Early Stopping**: This technique was employed to prevent overfitting and to halt the training process once the validation loss ceases to decrease for a predefined number of epochs.

**Number of Epochs**: The number of epochs for which the model was trained was adjusted based on the observations of training and validation performance. This helped in determining enough training cycles required before the model fully learns from the training data without overfitting.

**Experiment Findings**: The results from these experiments are reported in subsequent subsections, where the impact of each hyperparameter on the model’s performance is analysed.

This structured approach to experimentation not only ensures that our model is robust and reliable but also provides insights into the behaviour of different hyperparameters and their influence on the model's effectiveness. Such thorough testing is crucial for developing an AI tool that can reliably assist in the early detection of lung cancer.

## 3.2 Multi-layer Perceptron (MLP)

### 3.2.1 Model Definition

In the initial phase of our model exploration, we opted to implement a straightforward yet effective neural network architecture, the Multi-Layer Perceptron (MLP), using the PyTorch framework. This choice was driven by the need to establish a baseline for performance against more complex architectures that would be explored later in the study.

**Structure of MLP**: The MLP is a type of feedforward neural network that consists of multiple layers of neurons, each connected fully to all neurons in the previous and subsequent layers. The general mathematical model for each layer in an MLP is given by:

where:

- represents the activation function that introduces non-linearity into the model, enabling it to learn more complex patterns in the data.

- is the weight matrix associated with the layer.

- is the input vector to the layer, which may be the raw input data or the output from the previous layer.

- is the bias vector, enhancing the flexibility of the model to fit the data.

**Configuration Details**: For our experiments, the MLP was configured with 1-3 hidden layers. This structure was chosen to provide a balance between model complexity and computational efficiency. Each layer is fully connected, meaning that each neuron in a layer is connected to all neurons in the preceding and succeeding layers, thus allowing the network to potentially capture intricate patterns in the dataset.

**Training**: The model was trained using a backpropagation algorithm, which is standard for neural networks. This method involves adjusting the weights and biases of the neurons in a direction that minimally reduces the overall error of the model, measured by a loss function (in this case, cross-entropy loss).

The MLP serves as our baseline model for this project. Its performance, measured in terms of accuracy, loss, and computational time, provides valuable insights into how more sophisticated models might improve or build upon these initial results. The findings and specifics of these measurements are detailed in the subsequent results section.

### 3.2.2 Experiment Results

For Multi-Layer Perceptron, the experiments and their results are shown below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Experiment** | **Optimizer** | **Learning Rate** | **Dropout rate** | **Hidden units** | **Early Stopping** | **Scheduler** | **Loss function** | **Val. Acc.** | **Test**  **Acc.** |
| Baseline Configuration | Adam | 0.001 | 0.25 | [512, 256] | No | Reduce LR on Plateau | Cross Entropy Loss | 92.07 | 90.96 |
| High Learning Rate | Adam | 0.01 | 0.25 | [512, 256] | No | Reduce LR on Plateau | Cross Entropy Loss | 51.21 | 51.20 |
| Low Learning Rate | Adam | 0.0001 | 0.25 | [512, 256] | No | Reduce LR on Plateau | Cross Entropy Loss | 96.34 | 96.99 |
| Increased Dropout | Adam | 0.001 | 0.5 | [512, 256] | No | Reduce LR on Plateau | Cross Entropy Loss | 77.44 | 81.33 |
| Additional Hidden Layer | Adam | 0.001 | 0.25 | [512, 256, 128] | No | Reduce LR on Plateau | Cross Entropy Loss | 86.59 | 87.35 |
| Early Stopping Applied | Adam | 0.001 | 0.25 | [512, 256] | Yes | Reduce LR on Plateau | Cross Entropy Loss | 71.34 | 78.92 |

### 3.2.3 Experimental Findings

**A graph of a graph

Description automatically generated with medium confidence**

A graph with lines and numbers

Description automatically generated

Figure 1: Plots of Low Learning Rate Experiment

**Baseline Configuration:** With Adam optimizer, a learning rate of 0.001, and dropout rate of 0.25, the model achieved respectable validation and test accuracies of 92.07% and 90.96%, respectively. The balanced dropout and moderate learning rate helped in avoiding overfitting while ensuring efficient learning, evident from both the good performance metrics and the lowest testing losses among all setups​​.

**High Learning Rate:** Increasing the learning rate to 0.01 resulted in a drastic drop in performance (51.21% validation accuracy and 51.20% test accuracy). This suggests that the higher learning rate led to unstable training dynamics, possibly overshooting optimal weights during updates, as evidenced by high initial training losses and low precision and recall across classes​​.

**Low Learning Rate:** Reducing the learning rate to 0.0001 significantly improved the model’s accuracy to 96.34% on validation and 96.99% on testing, which was the highest among all experiments. This configuration allowed for finer adjustments in weight updates, leading to better convergence as shown by the steadily decreasing training and testing losses​​.

**Increased Dropout:** Doubling the dropout rate to 0.5 resulted in lower accuracies (77.44% validation and 81.33% test), suggesting that too much regularization might be preventing the model from learning sufficient patterns from the training data. The training and testing losses indicate more instability in learning compared to configurations with lower dropout rates​​.

**Additional Hidden Layer:** Adding an extra hidden layer (512, 256, 128) slightly reduced performance to 86.59% validation and 87.35% test accuracies. The additional complexity might have required more training data or epochs to fully leverage the deeper architecture, as higher layers generally capture more abstract patterns​​.

**Early Stopping Applied:** Implementing early stopping resulted in 71.34% validation and 78.92% test accuracies. This was likely due to not allowing the model sufficient time to reach optimal performance, as seen from the relatively higher training and validation losses at early epochs compared to other configurations where training was more prolonged​​.

## 3.3 Convoluted Neural Network (CNN)

### 3.3.1 Model Definition

Leveraging the capabilities of the PyTorch framework, our group has architected a 2D convolutional neural network (CNN) specifically tailored for lung cancer detection from CT images. The network architecture is strategically designed with a series of four convolutional layers followed by two fully connected linear layers, incorporating nonlinear activation functions and max pooling operations to effectively learn hierarchical features.

Convolutional Layers:

Each convolutional layer computes a 2D cross-correlation operation, defined as:

where signifies the 2D cross-correlation operation, is the batch size, and represents the number of channels. The output of these layers undergoes batch normalization:

where is the input, and are learnable parameters, and and are the mean and variance computed over the mini-batch. These normalized outputs are passed through a ReLU activation function to introduce non-linearity:

Post-activation, the data is processed by a 2D max pooling operation:

where and represent the kernel height and width, respectively.

Fully Connected Layers:

The output from the convolutional stacks is flattened and fed into two linear layers. These layers are designed to perform high-level reasoning from the features extracted by the convolutional layers. The first fully connected layer reduces dimensionality to a specified number of fully connected units, while the second maps these to the number of output classes.

Dropout Regularization:

To prevent overfitting, dropout regularization is strategically applied after each convolutional and fully connected layer at a rate determined by the hyperparameter .

PyTorch Implementation:

Our PyTorch implementation includes the following layer specifications:

* The first convolutional layer uses 32 filters, the second uses 64 filters, both with a kernel size of 3 and padding of 1 to preserve spatial dimensions after convolution.
* Batch normalization is applied following each convolutional layer.
* ReLU activation functions follow each batch normalization layer.
* Max pooling with a kernel size of 2 and stride of 2 reduces spatial dimensions by half after each convolutional layer stack.
* Dropout is applied after max pooling and each fully connected layer.
* The fully connected layers are sized according to and the final output classes.

The forward pass of the network defines the data flow through these layers and operations, resulting in a feature-rich representation of the input data suitable for classifying lung cancer from CT images.

### 3.3.2 Experiment Results

For Convolutional Neural Network, the experiments and their results are shown below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Experiment** | **Optimizer** | **Learning Rate** | **Dropout rate** | **Hidden units** | **Early Stopping** | **Scheduler** | **Loss function** | **Val. Acc.** | **Test**  **Acc.** |
| Baseline Model | Adam | 0.001 | 0.5 | 64 | No | Step LR | Cross Entropy Loss | 69.51 | 72.89 |
| High Learning Rate | Adam | 0.01 | 0.5 | 64 | No | Step LR | Cross Entropy Loss | 54.27 | 53.61 |
| Low Learning Rate | Adam | 0.0001 | 0.5 | 64 | No | Step LR | Cross Entropy Loss | 55.49 | 53.01 |
| Increased Dropout | Adam | 0.001 | 0.7 | 64 | No | Step LR | Cross Entropy Loss | 67.68 | 65.66 |
| Decreased Dropout | Adam | 0.001 | 0.3 | 64 | No | Step LR | Cross Entropy Loss | 78.05 | 85.54 |
| Increased FC Units | Adam | 0.001 | 0.5 | 128 | No | Step LR | Cross Entropy Loss | 85.37 | 79.52 |
| Decreased FC Units | Adam | 0.001 | 0.5 | 32 | No | Step LR | Cross Entropy Loss | 43.90 | 39.16 |
| Using Early Stopping | Adam | 0.001 | 0.5 | 64 | Yes | Step LR | Cross Entropy Loss | 61.59 | 60.24 |
| Different Optimizer: SGD | SGD | 0.001 | 0.5 | 64 | No | Exponential LR | Cross Entropy Loss | 93.90 | 91.57 |
| Different Scheduler | Adam | 0.001 | 0.5 | 64 | No | Cosine Annealing LR | Cross Entropy Loss | 68.29 | 69.88 |
| Different Loss Function | Adam | 0.001 | 0.5 | 64 | No | Step LR | Cross Entropy Loss Weighted | 68.29 | 69.27 |
| Different Optimizer: RMSProp | RMSProp | 0.001 | 0.5 | 64 | No | Reduce LR on Plateau | Cross Entropy Loss | 79.27 | 84.94 |
| Change Multiple Parameters | AdamW | 0.0005 | 0.6 | 96 | Yes | Reduce LR on Plateau | Cross Entropy Loss Weighted | 71.95 | 77.11 |
| Low Dropout and High Learning Rate | Adam | 0.01 | 0.2 | 64 | No | Exponential LR | Cross Entropy Loss | 85.37 | 84.93 |
| High Dropout and Low Learning Rate | Adam | 0.0001 | 0.7 | 64 | Yes | Cosine Annealing LR | Cross Entropy Loss | 49.40 | 45.73 |

### 3.3.3 Experimental Findings

A graph with lines and numbers

Description automatically generatedA graph of a graph

Description automatically generated with medium confidence

Figure 2: Plots of SGD Optimizer Experiment

Baseline Model: The baseline CNN model, employing Adam optimizer with a learning rate of 0.001, a dropout rate of 0.5, and 64 hidden units, performed modestly with test accuracy reaching 72.89%. This model did not use early stopping or a complex learning rate scheduler, resulting in stable but potentially underfit learning curves.

Learning Rate Variations: Increasing the learning rate to 0.01 resulted in significant underperformance (Test Acc: 53.61%), likely due to instability in training as seen in fluctuating loss values. Conversely, reducing the learning rate to 0.0001 also led to lower performance (Test Acc: 53.01%), indicating insufficient model updates per epoch.

Dropout Variations: Adjusting the dropout rate revealed that a decrease to 0.3 enhances the model’s ability to generalize, as evidenced by the highest test accuracy observed (85.54%). This suggests that the baseline model may be slightly over-regularized. Increasing the dropout rate to 0.7, however, decreased both the model's capacity to learn and its generalization performance (Test Acc: 65.66%).

Hidden Units: Doubling the hidden units to 128 improved learning capabilities (Test Acc: 79.52%), indicating that a higher model capacity was beneficial for this dataset. Reducing the hidden units to 32 severely degraded performance (Test Acc: 39.16%), underscoring the model's incapacity at lower complexity levels.

Early Stopping and Schedulers: Incorporating early stopping did not yield improvements, with a test accuracy of 60.24%. This could be due to premature stopping before necessary features could be learned. Using different schedulers like Cosine Annealing and Exponential LR slightly varied performances, suggesting that fine-tuning the scheduler based on validation loss trends could be beneficial.

Optimizer Variations: Switching to SGD with an exponential learning rate scheduler significantly boosted performance (Test Acc: 91.57%), highlighting the importance of optimizer choice in training dynamics. RMSProp also showed promising results, especially in conjunction with a learning rate scheduler that reduces the rate on plateaus (Test Acc: 84.94%).

Multi-parameter Adjustments: Configurations where multiple parameters were adjusted (e.g., different optimizers, learning rates, and dropout rates) generally showed varied results, emphasizing the need for careful tuning of hyperparameters. The combined adjustment using AdamW, a lower learning rate, and increased dropout reached a test accuracy of 77.11%, illustrating the delicate balance required in parameter settings.

## 3.4 CNN with LSTM

### 3.4.1 Model Definition

Finally, our group tried to integrate the convolutional neural network (CNN) and the long short-term memory (LSTM) network to take advantage of their respective advantages in feature extraction and sequential data processing. This innovative architecture aims to leverage the ability of CNNs to learn spatial features from image data and the ability of LSTMs to learn temporal dynamics. We hope to use CNN to extract features of CT scan images, and then LSTM can effectively capture the temporal or sequential dependencies between the high-level features extracted by CNN.

**CNN-LSTM Architecture:**

* **Convolutional Layers**: The initial layers consist of 2D convolutional operations that apply learnable filters to the input, capturing essential spatial information. Each convolutional layer is succeeded by batch normalization and ReLU activation function to stabilize learning and introduce non-linearities. A max-pooling layer follows to reduce spatial dimensions and enhance feature abstraction.
* **LSTM Layers**: The LSTM branch of the model receives a sequence of data transformed from the convolutional features, allowing it to learn from the temporal patterns present in the dataset. It is composed of LSTM units that are capable of maintaining a memory of past information, crucial for learning from sequences.
* **Dropout**: Dropout regularization is employed after each CNN and LSTM layer, aiming to prevent overfitting by randomly setting a fraction of input units to zero during training.
* **Fully Connected Linear Layers**: The outputs of both CNN and LSTM branches are then concatenated and passed through fully connected linear layers. These layers serve to integrate the learned spatial and temporal features, culminating in a final prediction output.

**Hyperparameters:**

The model's performance is contingent on several hyperparameters which are fine-tuned for optimal results:

* **dropout\_rate**: Controls the dropout regularization to prevent overfitting.
* **fc\_units**: Determines the size of the fully connected layers.
* **lstm\_units**: Specifies the number of units in the LSTM layers, which directly affects the model’s ability to learn temporal features.
* **num\_layers**: Configures the number of LSTM layers to build the depth of the model for learning complex patterns.

The model is instantiated and trained using a diverse combination of hyperparameters, followed by a random search to identify the most effective configurations for our task. Each set of hyperparameters is rigorously evaluated to deduce its impact on the model's performance, with a particular focus on validation accuracy as the metric of success.

### 3.4.2 Experiment Results

For Convolutional Neural Network + Long Short-Term Memory (CNN + LSTM), the experiments and their results are shown below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Experiment** | **Optimizer** | **Learning Rate** | **Dropout rate** | **Hidden units** | **Early Stopping** | **Scheduler** | **Loss function** | **Val. Acc.** | **Test**  **Acc.** |
| High Learning Rate - SGD | SGD | 0.01 | 0.5 | 64 | No | Step LR | Cross Entropy Loss | 80.49 | 81.33 |
| Weighted Loss | Adam | 0.001 | 0.5 | 64 | No | Step LR | Cross Entropy Loss Weighted | 65.24 | 65.06 |
| Increased LSTM Units | Adam | 0.001 | 0.3 | 64 | Yes | Step LR | Cross Entropy Loss | 82.32 | 82.53 |
| Reduced Dropout LSTM | Adam | 0.001 | 0.3 | 64 | Yes | Step LR | Cross Entropy Loss | 87.80 | 77.11 |
| Exponential Decay LR | Adam | 0.001 | 0.5 | 64 | Yes | Exponential LR | Cross Entropy Loss | 75.0 | 83.74 |

### 3.4.3 Experiment Findings

A graph of a graph

Description automatically generatedA graph with green and blue lines

Description automatically generated

Figure 3: Accuracy & Loss Plot of CNN + LSTM Model

**High Learning Rate with SGD:** This configuration utilized a significantly high learning rate of 0.01 with SGD as the optimizer. While often considered aggressive, this approach achieved reasonable success, reflected in test accuracies peaking at 81.33%. The model managed to stabilize the training loss, although the higher learning rate could have contributed to slightly erratic validation performance. Such a setup might be more sensitive to the initial settings and requires careful tuning of the learning rate scheduler to prevent overshooting the minima.

**Weighted Loss with Adam:** Incorporating a weighted loss function to address class imbalance, this setup utilized Adam with a standard learning rate of 0.001. Despite the theoretical advantages of handling skewed class distributions, the model underperformed, achieving a test accuracy of only 65.06%. This indicates potential overfitting to the majority class or an inability to generalize well across the minority classes, suggesting a need for better distribution of class weights or further hyperparameter tuning.

**Increased LSTM Units:** Enhancing the LSTM component by increasing the number of units provided a notable improvement in the model’s ability to capture temporal dependencies, as evidenced by a test accuracy of 82.53%. The addition of LSTM units appears to have helped in learning more complex patterns in the sequence data, which is crucial for tasks requiring understanding of temporal dynamics. The model also benefited from early stopping, preventing overfitting and leading to a stable learning process.

**Reduced Dropout:** Experimenting with a lower dropout rate of 0.3 resulted in the highest validation accuracy of 87.80% among the tests, although it experienced a drop in test accuracy to 77.11%. This discrepancy suggests that while the model was able to learn effectively from the training data, it might have slightly overfitted, as indicated by the lower performance on unseen test data. Adjusting the dropout rate could be key in balancing between learning sufficient representations and maintaining the model's generalization capabilities.

**Exponential Decay Learning Rate:** The application of an exponential decay on the learning rate aimed to combine the benefits of a high initial learning rate with the stability of a lower rate as training progresses. This model achieved the highest test accuracy of 83.74%, underscoring the effectiveness of this approach in managing learning rates dynamically throughout training phases. This method helped in fine-tuning the model gradually and avoiding drastic updates that could destabilize the learning process.

# 4. Experimental Evaluation – Custom Models

## 4.1 ResNet-152 Model

ResNet-152, specifically, is distinguished by its depth, featuring 152 layers. The architecture incorporates residual blocks, wherein the input to a block is added to its output, allowing for the direct flow of information through the network and mitigating the degradation problem encountered in training deeper networks. Due to its deep architecture and residual connections which help in superior feature extraction, ResNet-152 was chosen.

Additionally, ResNet-152 is known for its robustness to variations in input data and its ability to generalize well to unseen samples. Given the demographic diversity and inherent variability in lung cancer cases, having a model with strong generalization capabilities is crucial for achieving reliable and consistent classification results across different patient cohorts.

The particular weights we are initializing in the model have achieved a 82.284 acc@1 (on ImageNet-1K) and a 96.002 acc@5 (on ImageNet-1K).

After training/validating and testing on our dataset, the result was as follows:

A graph with green and orange lines

Description automatically generatedA graph with lines and numbers

Description automatically generated with medium confidence

Figure 4: Accuracy & Loss Plot of ResNet Model

The metrics obtained at the end of 30 epochs were:

1. Validation Accuracy: 93.90%
2. Test Accuracy: 97.59%

## 4.2 VGG-19 Model

One of VGG-19’s features is its simplicity and uniformity in design. It comprises 19 layers, predominantly consisting of 3x3 convolutional layers followed by max-pooling layers, with fully connected layers at the end. Additionally, it has been proven to demonstrate a remarkable performance on various computer vision tasks, including image classification.

VGG-19's deep architecture and hierarchical feature extraction capabilities make it well-suited for analyzing complex patterns in medical images. Its stacked convolutional layers enable it to capture intricate details and nuances present in CT scan images of lung tissue, allowing for accurate differentiation between benign, malignant, and normal cases.

The particular weights we are initializing in the model have achieved a 72.376 acc@1 (on ImageNet-1K) and a 90.876 acc@5 (on ImageNet-1K).

After training/validating and testing on our dataset, the result was as follows:

A graph of a graph

Description automatically generated with medium confidenceA graph with green and orange lines

Description automatically generated

Figure 5: Accuracy & Loss Plot of VGG Model

## 4.3 DenseNet-161 Model

DenseNet-161 is a deep convolutional neural network known for its dense connectivity pattern. Unlike traditional convolutional neural networks, DenseNet connects each layer to every other layer in a feed-forward fashion. This dense connectivity fosters feature reuse and encourages information flow throughout the network, leading to enhanced gradient propagation and alleviation of the vanishing gradient problem. With its 161 layers, DenseNet-161 can capture intricate features at multiple scales, making it highly effective for image classification tasks.

The unique dense connectivity structure of DenseNet-161 offers several advantages. By promoting feature reuse, it facilitates efficient parameter utilization, leading to reduced model redundancy and improved parameter efficiency. Additionally, DenseNet-161 tends to exhibit better performance with lower computational requirements compared to other deep architectures.

The downside to DenseNet is that it consumes a lot of memory and there is additional computation overhead during training.

However, using DenseNet-161 is justified for our case because of its ability to learn complex representations from limited data which could solve the issue with our highly imbalanced class dataset.

The particular weights we are initializing in the model have achieved a 77.138 acc@1 (on ImageNet-1K) and a 93.56 acc@5 (on ImageNet-1K).

After training/validating and testing on our dataset, the result was as follows:

A graph with green and blue lines

Description automatically generatedA graph of a graph

Description automatically generated with medium confidence

Figure 6: Accuracy & Loss plot of Densenet model

The metrics obtained at the end of 30 epochs were:

1. Validation Accuracy: 92.68%
2. Test Accuracy: 96.99%

## 4.4 MobileNetV3-Large Model

MobileNetV3-Large is an evolution of the MobileNet series, designed to provide efficient yet powerful deep neural network architectures specifically optimized for edge devices. It is essentially a balance between model size, computational efficiency, and accuracy.

One reason why MobileNetV3-Large was chosen for this task is because of its efficient building blocks, such as inverted residual blocks and linear bottlenecks. These blocks enable the network to achieve a high level of representational capacity while minimizing computational costs, making it well-suited for deployment on devices with limited computational resources.

The particular weights we are initializing in the model have achieved a 74.042 acc@1 (on ImageNet-1K) and a 93.56 acc@5 (on ImageNet-1K).

After training/validating and testing on our dataset, the result was as follows:

A graph with lines and numbers

Description automatically generated with medium confidenceA graph of a graph

Description automatically generated

Figure 7: Accuracy & Loss Plot of MobileNet Model

The metrics obtained at the end of 30 epochs were:

1. Validation Accuracy: 96.34%
2. Test Accuracy: 98.19%

## 4.5 Wide ResNet-101-2 Model

Wide ResNet-101-2 is characterized by its wider convolutional layers compared to traditional ResNet architectures. The "101-2" in its name denotes the number of layers and the widening factor, where the factor of "2" indicates that the number of channels in each layer is doubled compared to the standard ResNet.

By increasing the width of the network, it enhances the model's capacity to capture diverse features and patterns present in the data, leading to improved generalization and performance on various tasks. Additionally, Wide ResNet-101-2 mitigates the risk of overfitting by introducing more parameters while maintaining the overall simplicity of the architecture.

Furthermore, Wide ResNet-101-2 inherits the benefits of residual connections, which enable effective gradient propagation and training of deep networks. The wide architecture also facilitates efficient training by enabling better parameter utilization and reducing the risk of vanishing gradients.

The particular weights we are initializing in the model have achieved a 82.51 acc@1 (on ImageNet-1K) and a 96.02 acc@5 (on ImageNet-1K).

After training/validating and testing on our dataset, the result was as follows:

A graph with lines and numbers

Description automatically generated

A graph with a line graph

Description automatically generated with medium confidence

Figure : Accuracy & Loss Plot of Wide Resnet model

The metrics obtained at the end of 30 epochs were:

1. Validation Accuracy: 96.34%
2. Test Accuracy: 98.19%

## 4.6 GoogleNet Model

Inception-v3, developed by Google researchers, is an evolution of the original Inception architecture (GoogleNet).

One of the notable features of Inception-v3 is its use of deeper and wider convolutional layers compared to earlier versions. This allows the network to capture more complex and diverse features from input images, enhancing its ability to discriminate between different classes. Additionally, Inception-v3 incorporates various architectural enhancements, such as factorized convolutions and batch normalization, to facilitate more stable training and better generalization.

What’s interesting is that there are “bottleneck layers” which employ 1x1 convolutions to reduce the dimensionality of feature maps before applying larger convolutions, effectively reducing the number of parameters and computational complexity of the model.

Furthermore, Inception-v3 integrates auxiliary classifiers to encourage the propagation of gradients during training and mitigate the risk of vanishing gradients. These auxiliary classifiers serve as regularization mechanisms, helping to improve the overall robustness and performance of the network.

The particular weights we are initializing in the model have achieved a 77.294 acc@1 (on ImageNet-1K) and a 93.45 acc@5 (on ImageNet-1K).

After training/validating and testing on our dataset, the result was as follows:

A graph with lines and numbers

Description automatically generated

A graph with green and blue lines

Description automatically generated

Figure : Accuracy & Loss Plot for GoogleNet Model

The metrics obtained at the end of 30 epochs were:

1. Validation Accuracy: 96.95%
2. Test Accuracy: 98.19%

# 5. Future Work

## 5.1 Enhancing the Dataset

The current IQ-OTH/NCCD lung cancer dataset, while comprehensive, encompasses a total size of only 157 MB, which limits the variability and complexity of the data available for robust model training. To improve the model's accuracy and generalizability, we propose the following initiatives:

* **Data Augmentation**: Implement advanced augmentation techniques such as varying the window width and center levels in CT scans, applying geometric transformations (scaling, translation), and synthetic generation of CT scan images using Generative Adversarial Networks (GANs). These methods will enhance the dataset's diversity, simulating a wider variety of pathological conditions without requiring additional real patient data.
* **Dataset Expansion**: Increasing the physical size of the dataset is crucial. We aim to enlarge the current dataset by adding more annotated CT scans from varied sources. This could include collaboration with other medical institutions to incorporate their de-identified lung cancer CT scans, focusing on different stages and types of lung cancer across diverse populations.
* **International Collaboration**: Expand data collection efforts internationally to include CT scans from patients in different geographical locations and from varied demographic backgrounds. This will not only increase the size of the dataset but also improve the model's applicability on a global scale.
* **Longitudinal Data Collection**: Engage in longitudinal studies to gather follow-up scans from patients. This approach can provide additional insights into the progression of lung cancer, offering valuable data that can enhance detection algorithms.

These strategic enhancements will significantly increase the volume and diversity of the dataset, enabling more comprehensive training and validation of our deep learning models. By doing so, we aim to boost the diagnostic accuracy and reliability of our system, ensuring it can operate effectively across a broad range of clinical settings and populations.

## 5.2 Extending Model Capabilities

Our initial model leverages a convolutional neural network architecture which has shown promising results. However, future iterations could explore:

* **Advanced Neural Architectures**: Investigating the efficacy of more complex architectures such as ResNet, DenseNet, or even hybrid models that combine CNNs with Vision Transformer could potentially enhance feature extraction capabilities.
* **Transfer Learning**: Applying transfer learning techniques using pre-trained models on similar tasks (e.g., other medical imaging diagnostics) to improve training efficiency and potentially boost performance.

## 5.3 Application Development

**Diagnostic Tool Development**: Development of a user-friendly diagnostic tool that can be integrated into hospital systems. This tool would use the trained model to provide real-time analysis of CT scans, offering immediate support in clinical decision-making.

**Real-World Testing and Validation**: Conducting extensive testing in clinical environments to validate the model’s effectiveness and reliability in a real-world setting. This involves collaboration with oncologists and radiologists to assess practical utility and gather feedback for iterative improvements.

# 6. Conclusion

In this comprehensive study, we have effectively demonstrated the potential of deep learning techniques in enhancing lung cancer detection using the IQ-OTH/NCCD dataset. Our experiments with various neural network architectures, including Multi-Layer Perceptrons, Convolutional Neural Networks, and advanced hybrid models such as CNN with LSTM, underscore the robustness and adaptability of these methods in handling complex imaging data for accurate tumor classification.

The implementation of state-of-the-art models like ResNet-152, VGG-19, DenseNet-161, MobileNetV3-Large, Wide ResNet-101-2, and GoogleNet further enriched our understanding, highlighting the importance of architectural choices in achieving high accuracy. These models not only provided valuable benchmarks but also revealed critical insights into the dynamics of deep learning applied to medical imaging.

Through meticulous experimentation, we identified key factors influencing model performance, such as optimizer selection, learning rate adjustments, and dropout rates. The integration of these elements facilitated the fine-tuning of our models to optimize accuracy and generalizability. Particularly, the use of the Adam optimizer and various learning rate schedulers played a pivotal role in managing the training dynamics effectively, leading to significant improvements in model performance.

Looking forward, the expansion of the dataset and incorporation of more sophisticated data augmentation techniques are expected to further enhance the robustness and reliability of our models. Additionally, exploring more complex neural architectures and extending our models to integrate seamlessly within clinical workflows will pave the way for real-world applications.

# 7. Acknowledgements

We extend our sincere gratitude to Prof. Matthieu De Mari and Ngai-Man (Man) Cheung for generously sharing exemplars with us. Through careful examination of the repositories provided, we gained valuable insights into various aspects including approach, folder structure, and problem-solving strategies, which served as a significant source of inspiration for our own work.

Furthermore, we would like to express our appreciation to the teams who dedicated their time and effort to share their code and meticulously document their procedures. Your contributions have been instrumental in enriching our understanding and enhancing the quality of our research.

# 8. Reference

Ardila, D., Kiraly, A. P., Bharadwaj, S., Choi, B., Reicher, J. J., Peng, L., ... & Lungren, M. P. (2019). End-to-end lung cancer screening with three-dimensional deep learning on low-dose chest computed tomography. Nature medicine, 25(6), 954-961.

Hu, X., Gao, Y., Kang, Y., Liang, W., & Yang, X. (2020). Deep learning for lung cancer prognostication: a retrospective multi-cohort radiomics study. IEEE Journal of Biomedical and Health Informatics, 24(2), 531-541.

Armato III, S. G., McLennan, G., Bidaut, L., McNitt-Gray, M. F., Meyer, C. R., Reeves, A. P., ... & Gavrielides, M. A. (2011). The Lung Image Database Consortium (LIDC) and Image Database Resource Initiative (IDRI): a completed reference database of lung nodules on CT scans. Medical physics, 38(2), 915-931.

Aberle, D. R., Adams, A. M., Berg, C. D., Black, W. C., Clapp, J. D., Fagerstrom, R. M., ... & Gatsonis, C. (2011). Reduced lung-cancer mortality with low-dose computed tomographic screening. New England Journal of Medicine, 365(5), 395-409.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980.

# 9. Appendix

# Other experiments for CNN that was conducted were as follows:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Experiment** | **Optimizer** | **Learning Rate** | **Dropout rate** | **Hidden units** | **Early Stopping** | **Scheduler** | **Loss function** | **Val. Acc.** | **Test**  **Acc.** |
| CNN Optimized Dropout | Adam | 0.001 | 0.3 | 64 | Yes | Step LR | Cross Entropy Loss | 84.15 | 87.95 |
| CNN High Capacity | Adam | 0.001 | 0.5 | 128 | Yes | Reduce LR on Plateau | Cross Entropy Loss | 89.02 | 90.96 |
| CNN Optimized SGD | SGD | 0.005 | 0.3 | 64 | Yes | Exponential LR | Cross Entropy Loss | 70.73 | NA |
| CNN Advanced Regularization | Adam | 0.001 | 0.3 | 128 | Yes | Cosine Annealing LR | Cross Entropy Loss Weighted | 67.68 | 59.03 |
| CNN Combo Best Practices | RMSProp | 0.001 | 0.3 | 128 | Yes | Reduce LR on Plateau | Cross Entropy Loss | 53.05 | 75.90 |

# The accuracy and loss plot, in addition to the classification matrix outputted by a model can be viewed on our github. Please do check the ReadME. Additionally, for reproducibility, the weights of the model can be downloaded and used for inference.

# 