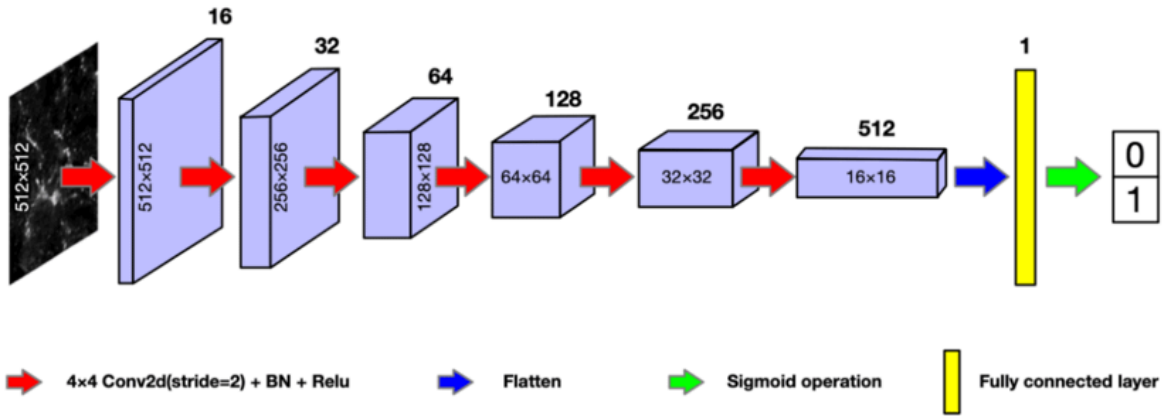
Classification with Deep Learning

Introduction

Deep Learning is a subset of machine learning that involves neural networks with many layers (deep networks). It is characterised by its ability to automatically learn hierarchical representations of data. The evolution of DL has significantly advanced the field of image processing, enabling machines to perform tasks such as image classification, object detection, and image generation with high accuracy.

Neural Networks are one of many methods of DL and excel at handling topological data such as images. Neural networks are computational models inspired by the human brain, composed of layers of interconnected nodes (neurons). These networks transform input data through a series of weighted connections and activation functions to produce an output.

Convolutional Neural Networks are specifically designed to handle grid-like data structures. They consist of multiple layers, primarily including convolutional layers, pooling layers, and fully connected layers. Lower layers capture basic features like edges and textures, while deeper layers capture more complex patterns and object parts.
By using small filters, CNNs exploit spatially local correlations in data, reducing the number of parameters compared to fully connected networks. This makes CNNs computationally efficient and suitable for large-scale image processing tasks. Scalability of CNNs are given by adapting the depth (number of layers) which allows the model to handle increasingly complex functions



The core functional layers of a CNN encompass Convolutional, Pooling and Dense Layers:

- **Convolutional Layers**:
    - Apply convolution operations to the input image using filters (typically 3x3).
    - Filters slide over the image to produce feature maps that highlight specific patterns like edges, textures, and shapes.
- **Pooling Layers**:
    - Reduce the spatial dimensions of feature maps, preserving the most important information while reducing computational complexity.
    - Common pooling techniques include max pooling (taking the max of a striding window) and average (taking the mean of a striding window) pooling.
- **Fully Connected Layers (Dense Layers)**:
    - After several convolutional and pooling layers, the high-level reasoning in the neural network is done via fully connected layers.
    - These layers flatten the feature maps and connect every neuron to every neuron in the next layer, similar to traditional neural networks.
- **Activation Functions**:
    - Introduce non-linearity into the model, allowing it to learn complex patterns.
    - ReLU (Rectified Linear Unit) is the most commonly used activation function in CNNs. For the final Dense layer that predicts the classification a softmax or sigmoid function is applied

Since its upbringing the field of Convolutional Neural Networks has seen some developments in the usage of layers, types of layers and other techniques. Down below we introduce three commonly known milestone architectures which we will employ for our classification.

LeNet-5
Developed in 1998 LeNet-5 is one of the earliest CNN architectures. It was primarily designed for handwritten digit recognition and demonstrated the potential of Convolutional Neural Networks (CNNs). LeNet-5 consists of seven layers: two convolutional layers, two pooling layers, and three dense layers.

AlexNet
Developed in 2012 AlexNet marked a significant breakthrough in the field of computer vision. AlexNet consists of five convolutional layers, followed by three fully connected layers. It employs ReLU activation functions, dropout for regularisation, and max-pooling layers. The use of ReLU activation functions addresses the vanishing gradient problem and the dropout layers prevent overfitting.

VGGNet
Developed in 2014 VGGNet emphasises simplicity and depth in network architecture design. VGGNet is known for its use of very small (3x3) convolution filters stacked in deep layers. The approach of using small convolution filters in deeper networks allowed for more complex features to be learned without a dramatic increase in the number of parameters.

LMA3P
The project groups approach of patching together a sensible selection of layers fit to the very challenge at hand.

Data Preparation

In order to train our CNNs we prepare the image dataset in the following way

1. Resize images (224 x 224) while batching them into buckets of 32.
2. Split into 3 datasets: Training (80%), validation (10%) and test (10%).
3. Calculate the class weights on the training dataset for later user
4. Apply data augmentation techniques to the training dataset:
   Rotation (18°), Zoom(5%) and Horizontal Flip

Model Definition

The definition of the models can be seen further below at the end of this section.

All models are being trained with the following parameters:
Optimizer: Adam
Loss function: Categorical Crossentropy
Metrics: Accuracy
Epochs: 25
Class Weights: Class weights as determined above

First we started out with LMAP3, for which we tried 3 different approaches and then took the best approach into the VGGNet and AlexNet:

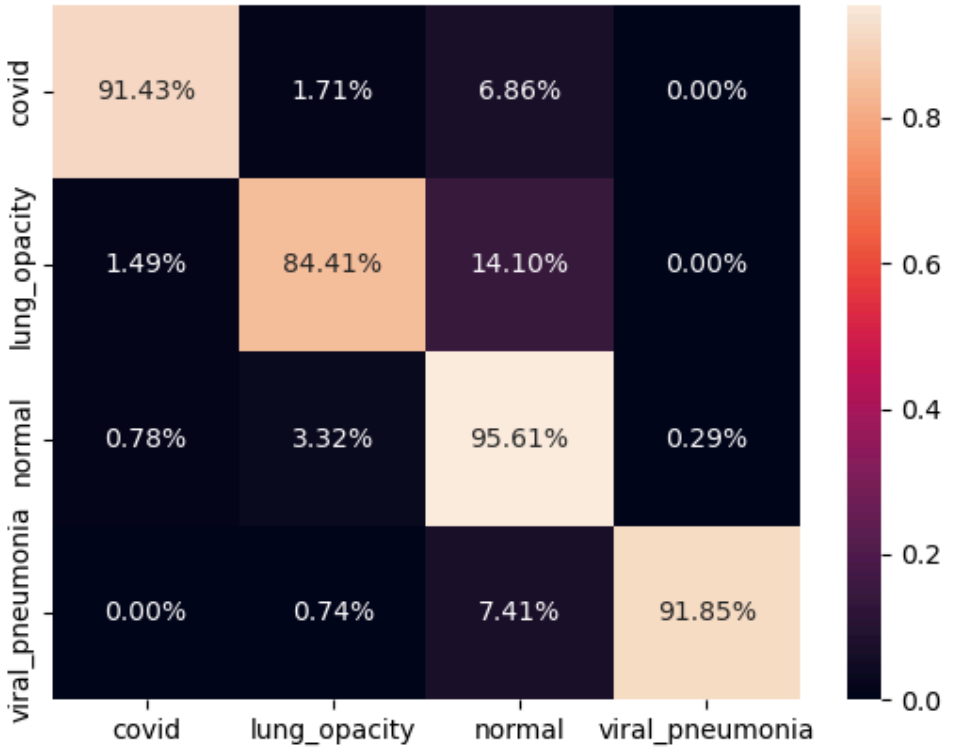
A. LMAP3: Full images without data augmentation
B. LMAP3: Full images with data augmentation
C. LMAP3: Only lung section of images with data augmentation
D. AlexNet: Full images with data augmentation
E. VGGNet: Full images with data augmentation

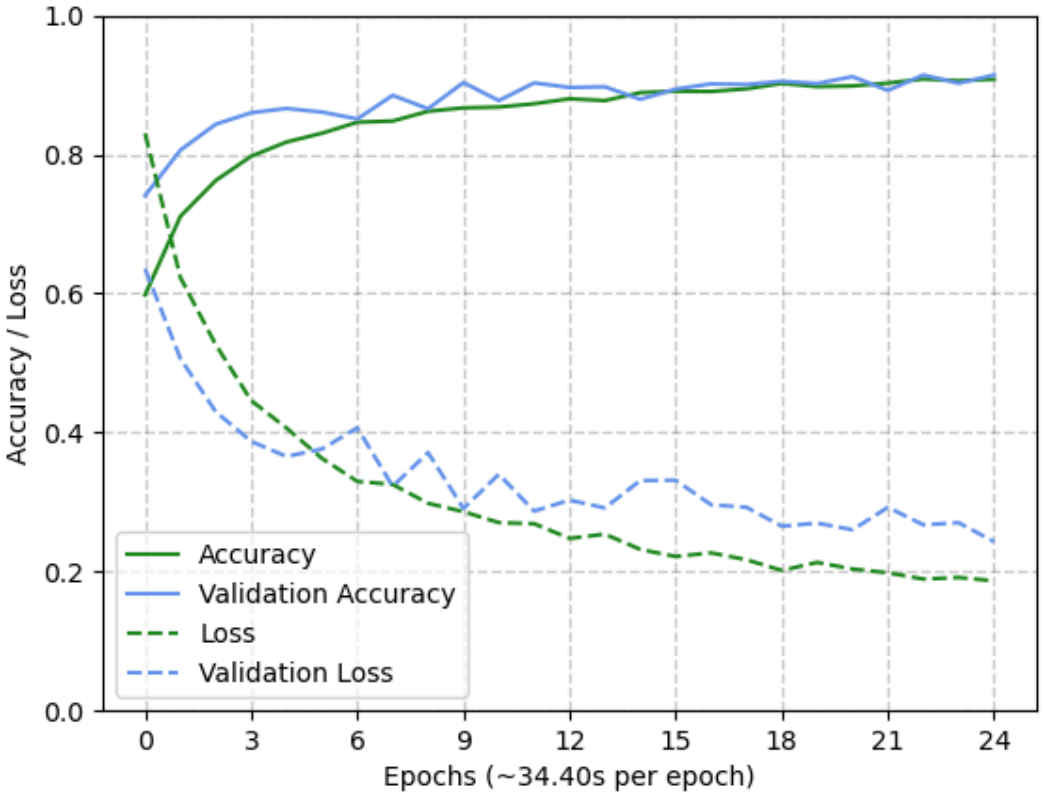
Then we let the trained models predict the classes of our test dataset, which it had never seen before. The table below shows the overall accuracy as well as the average computation time per epoch

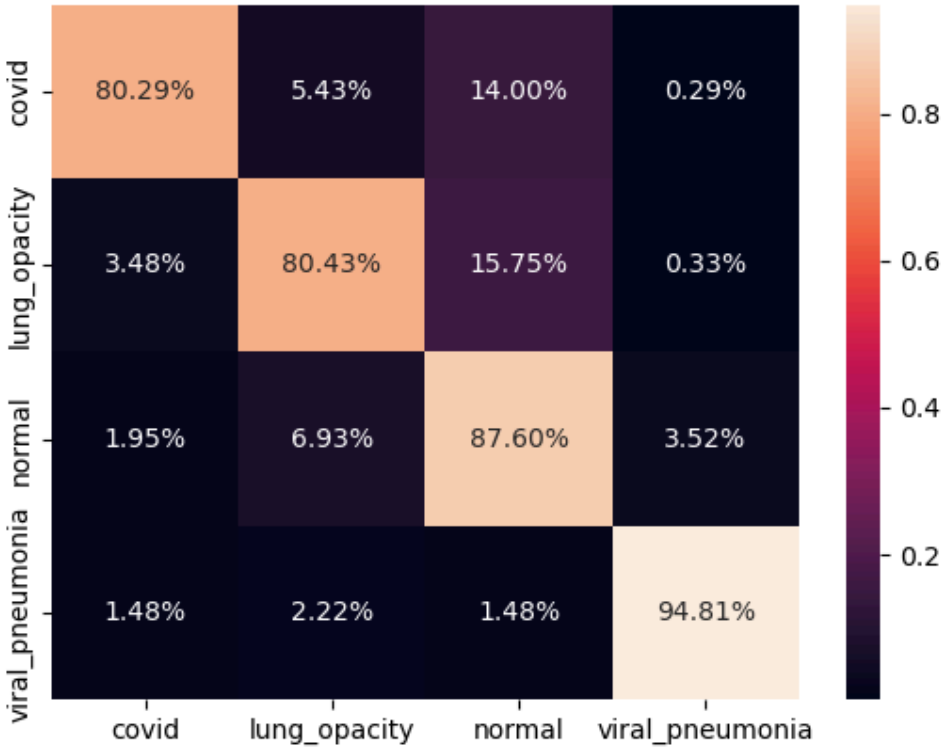
	A	B	C	D	F
Accuracy	0.90	0.91	0.83	0.85	0.75
Time / epoch	22.1 s	34.4 s	35.9 s	24.2 s	300 s

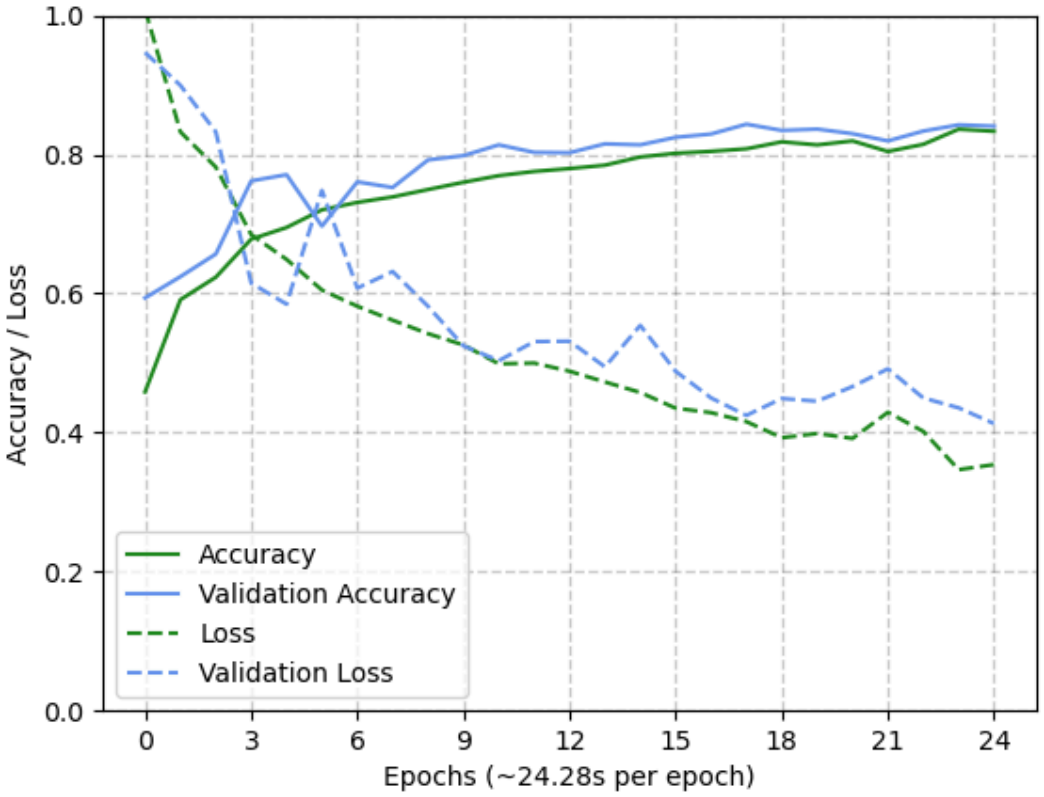
To see how the best model (B) and worst model (C) perform in more detail we take a look at the training progress as well as the respective confusion matrices.

The confusion matrices shows the percentage of predictions per row, i.e. per real class. A value of 91.73% means that this percentage of images of the real class have been labelled correctly.


covid
lung_opacity
normal
viral_pneumonia
91.43%
1.71%
6.86%
0.00%
1.49%
84.41%
14.10%
0.00%
0.78%
3.32%
95.61%
0.29%
0.00%
0.74%
7.41%
91.85%
covid
lung_opacity
normal
viral_pneumonia
0.8
0.6
0.4
0.2
0.0


Accuracy / Loss
Epochs (~34.40s per epoch)
Accuracy
Validation Accuracy
Loss
Validation Loss
1.0
0.8
0.6
0.4
0.2
0.0
0
3
6
9
12
15
18
21
24


covid
lung_opacity
normal
viral_pneumonia
80.29%
5.43%
14.00%
0.29%
3.48%
80.43%
15.75%
0.33%
1.95%
6.93%
87.60%
3.52%
1.48%
2.22%
1.48%
94.81%
covid
lung_opacity
normal
viral_pneumonia
0.8
0.6
0.4
0.2


Accuracy / Loss
Epochs (~24.28s per epoch)
Accuracy
Validation Accuracy
Loss
Validation Loss
1.0
0.8
0.6
0.4
0.2
0.0
0
3
6
9
12
15
18
21
24

```
Model: "AlexNet"
Layer               Output Shape        Param #
================================================
BatchNormalization (None, 224, 224, 1) 4
Conv2D              (None, 56, 56, 96)  11712
Activation          (None, 56, 56, 96)  0
MaxPooling2D        (None, 27, 27, 96)  0
Conv2D              (None, 7, 7, 256)   614656
Activation          (None, 7, 7, 256)   0
MaxPooling2D        (None, 3, 3, 256)   0
Conv2D              (None, 1, 1, 384)   885120
Activation          (None, 1, 1, 384)   0
Conv2D              (None, 1, 1, 384)   1327488
Activation          (None, 1, 1, 384)   0
Conv2D              (None, 1, 1, 256)   884992
Activation          (None, 1, 1, 256)   0
Flatten             (None, 256)         0
Dense               (None, 4096)        1052672
Dropout             (None, 4096)        0
Dense               (None, 4096)        16781312
Dropout             (None, 4096)        0
Dense               (None, 4)           16388
================================================
Total params: 21,574,344
```

```
Model: "LMAP3"
Layer                Output Shape             Param #
=================================================
BatchNormalization  (None, 224, 224, 1)      4
Conv2D              (None, 222, 222, 32)     320
MaxPooling2D        (None, 111, 111, 32)     0
Conv2D              (None, 109, 109, 32)     9248
MaxPooling2D        (None, 54, 54, 32)       0
Conv2D              (None, 52, 52, 32)       9248
MaxPooling2D        (None, 26, 26, 32)       0
Conv2D              (None, 24, 24, 64)       18496
MaxPooling2D        (None, 12, 12, 64)       0
Conv2D              (None, 10, 10, 64)       36928
MaxPooling2D        (None, 5, 5, 64)         0
Dropout             (None, 5, 5, 64)         0
Flatten             (None, 1600)             0
Dense               (None, 128)              204928
Dense               (None, 64)               8256
Dropout             (None, 64)               0
Dense               (None, 4)                260
=================================================
Total params: 287,688
```

```
Model: "VGG19"
Layer              Output Shape        Param #
=================================================
BatchNormalization (None, 224, 224, 3) 12
Functional         (None, 512)         20024384
Dense              (None, 4)           2052
=================================================
Total params: 20,026,448
```

Results

Two of the utilised architectures (LMAP3 and AlexNet) show a good match between the training and validation metrics, which means that they are not overfitting. The goal metric, accuracy, is above 0.8 in all cases, which is rather solid. An accuracy of 0.9 is reached for the LMAP3 model which is close to the best achieved values of ~0.95 found in available solutions on Kaggle.

With regard to the confusion matrix another point that can be observed is that the trained models seem to capture classes differently well. The biggest misclassification comes from images that belong to the category ‘Viral Pneumonia’ and ‘Lung Opacity’ but are labelled as Normal. It could be argued that these two symptoms are less distinguishable than the lungs affected by Covid. Since the majority of our samples are of class ‘Normal’ it could be that the linear class weight compensation is only efficient to a certain point.

Also it is very interesting to observe that VGG19, a usually highly functional image recognition network, achieves a minor result despite demanding 10 times the amount of computation time. At the same time the trained VGG19 model does present a severe gap between training and validation accuracy, implying that it improved steadily on the training data while not generalising well to unseen data. Overfitting is a common issue and can be addressed with several ways, e.g. data augmentation or the addition of dropout layers. However since both of these are implemented, and VGG19 is a supposedly high performing pretrained model, it is beyond the worth of the effort to investigate why this one performs so badly in our case.

Interpretability

Model interpretability in deep learning for image classification is crucial for understanding and trusting the decisions made by complex neural networks. As deep learning models, particularly Convolutional Neural Networks (CNNs), become more advanced and widely used, it becomes imperative to elucidate how these models arrive at their predictions. Interpretability techniques, such as saliency maps and Grad-CAM (Gradient-weighted Class Activation Mapping), help visualise which parts of an image are most influential in the model’s decision-making process. These methods provide insights into the model’s inner workings, highlighting important features and patterns that the network has learned. This not only aids in diagnosing and correcting potential biases or errors in the model but also builds confidence among users and stakeholders by demonstrating that the model’s decisions are

based on relevant and understandable aspects of the input data. For our project we focussed on the GradCAM approach

GradCAM (Gradient-weighted Class Activation Mapping)

Grad-CAM is a popular technique used for visualising and interpreting the decisions of Convolutional Neural Networks (CNNs), particularly in image classification tasks. It provides a way to produce visual explanations for predictions made by a CNN, highlighting the regions of the input image that are most relevant to the predicted class.

How Grad-CAM Works

1. **Select a Convolutional Layer**
   Typically, this is a layer near the end of the network, where high-level features are captured. The feature map of this layer will be visualised.
2. **Forward Pass**
   Predict the class scores (probabilities) before applying the final softmax function
3. **Compute Gradients**
   Compute the gradient of the score for the target class with respect to the feature maps of the selected convolutional layer. This gradient highlights how changes in the feature maps affect the class score.
4. **Global Average Pooling**
   Apply global average pooling to the gradients obtained in the previous step. This gives the importance weights for each feature map. These weights reflect the significance of each feature map in the decision-making process for the target class.
5. **Weighting Feature Maps**
   Multiply each feature map by its corresponding importance weight. This scales the feature maps according to their relevance to the target class.
6. **Generate Heatmap**
   Sum the weighted feature maps across the channel dimension to obtain a single-channel heatmap. This heatmap highlights the important regions in the input image.
7. **ReLU Activation**
   Apply the ReLU (Rectified Linear Unit) activation function to the heatmap to focus on the positive influences. Negative values are set to zero, as they typically represent regions that are less relevant or counter to the predicted class.
8. **Upsample to Original Image Size**
   Upsample the heatmap to the same size as the input image. This can be done using bilinear interpolation or other upsampling methods.
9. **Overlay on Input Image**
   The resulting heatmap is then overlaid on the input image to visualise which parts of the image contributed most to the classification. The regions with higher intensity in the heatmap are more influential in the model's prediction.

To this point the implementation of GradCAM does not yield a satisfying explanation for our model’s ways of working. We hope to implement it by the final report.