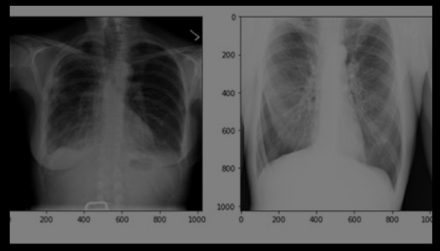
# Multi Label Chest X-Ray Classification



Georg Zhelev
Paul Stryck
Petrit Igrishtaj

# Agenda

- Task
- Result
- Our Approach

# 1. Task

**Classifying chest X-rays into 15 categories**

14 diseases + "no finding"

**Dataset**: NIH Chest **X-Ray** dataset

112,120 X-ray images with disease labels from 30,805 unique patients

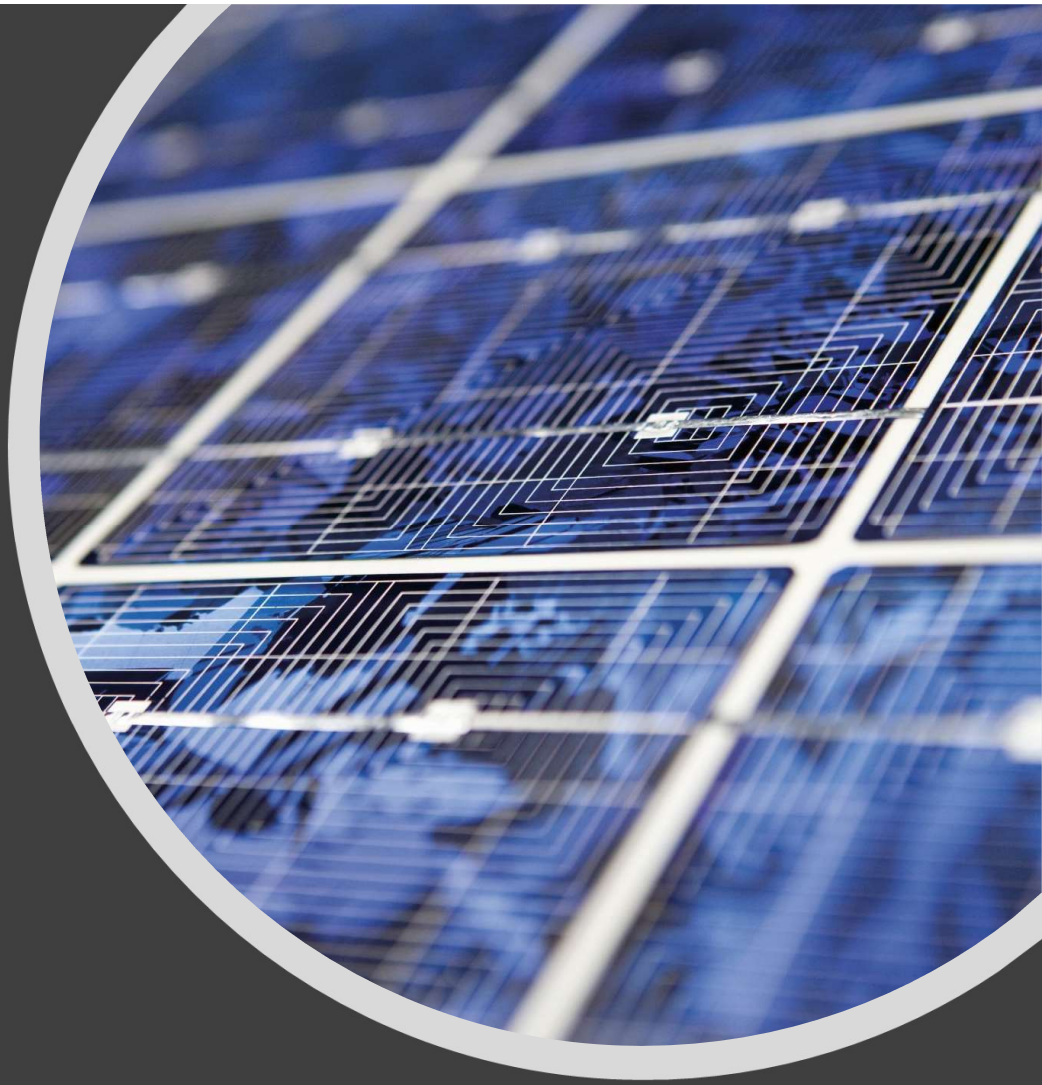
Labels algorithmically generated from patients' records

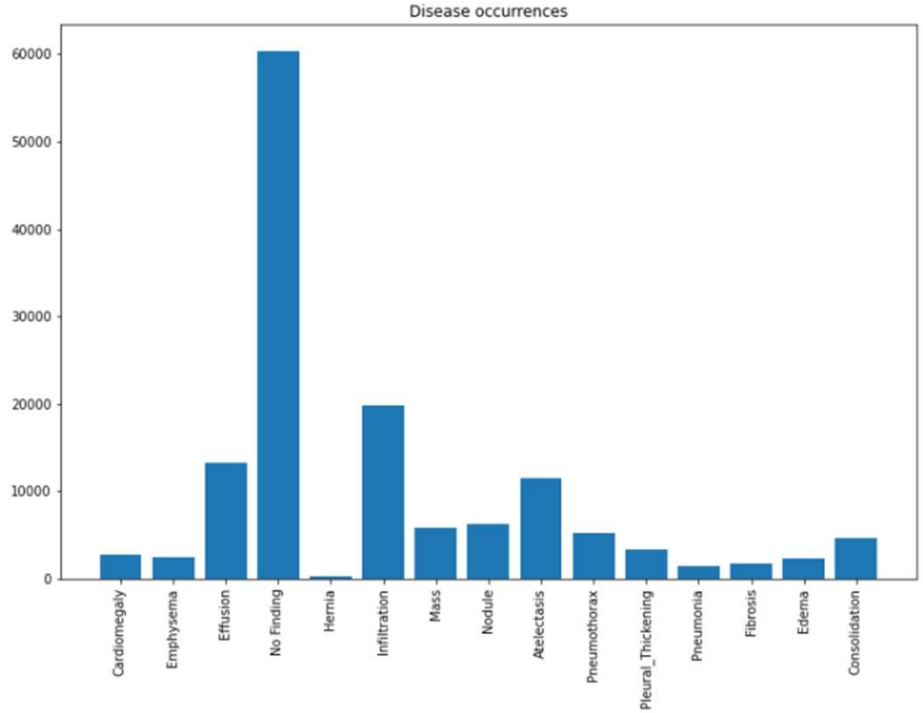
Class imbalances

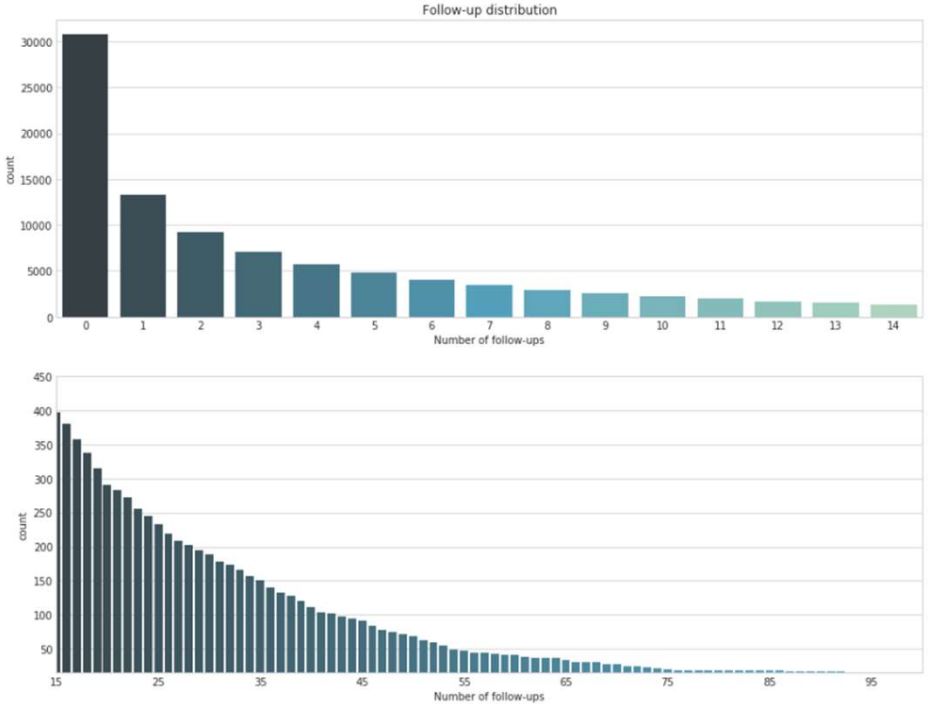
Correlation between diseases

**How**

Empirical evaluation of many neural network architectures




Disease occurrences
60000
50000
40000
30000
20000
10000
0
Cardiomegaly
Emphysema
Effusion
No Finding
Hernia
Infiltration
Mass
Nodule
Atelectasis
Pneumothorax
Pleural_Thickening
Pneumonia
Fibrosis
Edema
Consolidation


Follow-up distribution
count
Number of follow-ups
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
count
Number of follow-ups
15 25 35 45 55 65 75 85 95

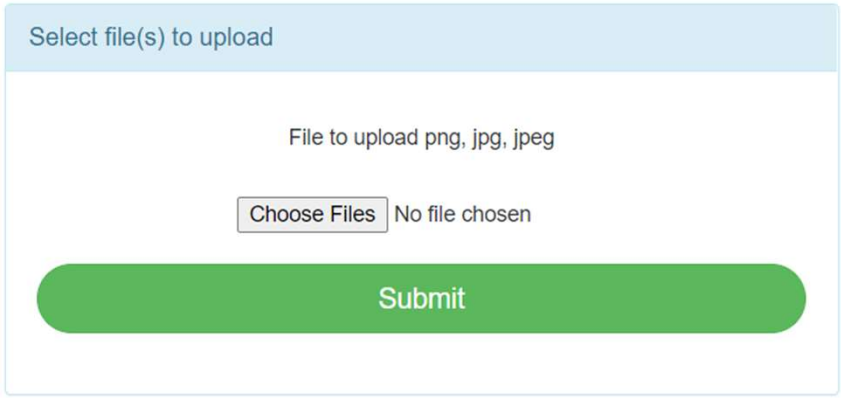
# 2. Final Products

1. Diagnosis Tool
2. Grad-CAM: Visual Explanations
3. Final Model
4. Final Model AUC Scores
5. Efficient Pipeline

# 2.1 Diagnosis Tool

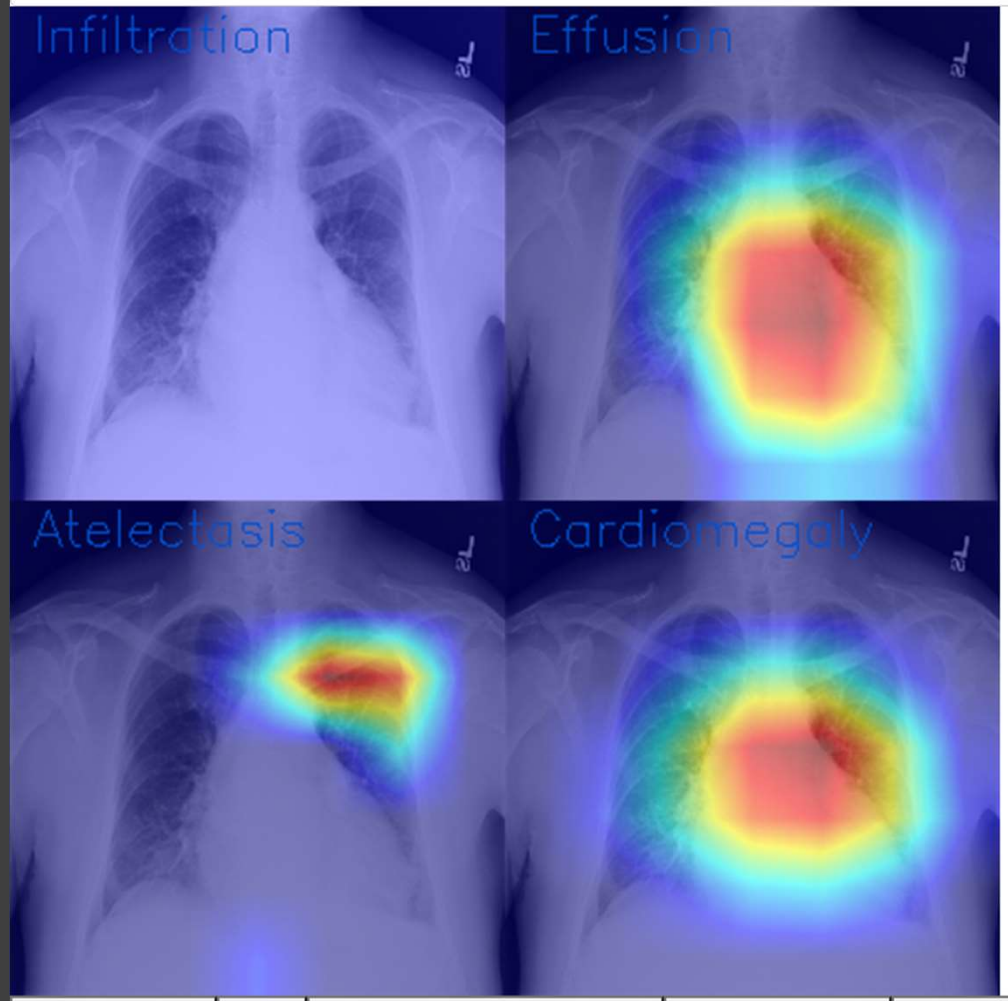
- Combines components from Flask and Dash
- A tool to dynamically upload and classify multiple chest  X-rays
- Tool is able to show all results and provide visual explanations via heatmaps (based on Grad-CAM)
- Runtime
  - Normal prediction: 2-3 seconds/image
  - Detailed view: 5-6 seconds/image

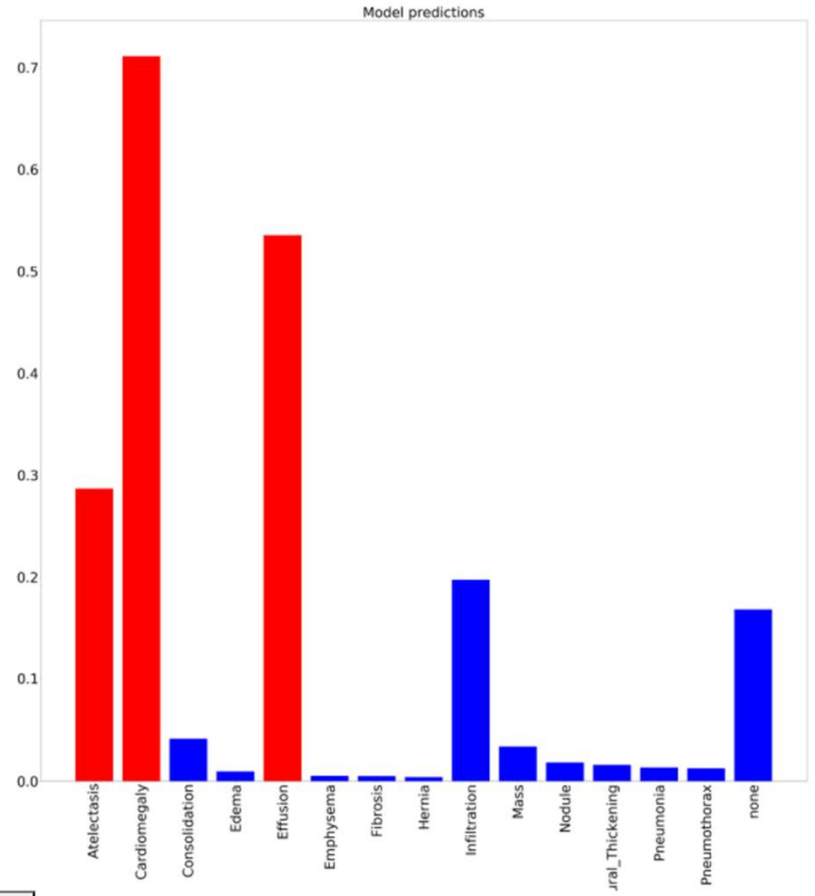
- Code:


Select file(s) to upload
File to upload png, jpg, jpeg
Choose Files
No file chosen
Submit

Heatmap_top4 00000001_002.png

Plot 00000001_002.png




Start predictions! | Home | Get detailed view of heatmaps! | Download Image 1 | Download Image 2

# 2.2 Grad-CAM: Visual Explanation

- Deep Neural Networks are hard to interpret
- Highlights important regions of the input image for a specific classification result
- Let y be the final result (before softmax) and k the number of activation maps A
- Neuron importance weight of class c:

$$\alpha_k^c = \overbrace{\frac{1}{Z}\sum_i\sum_j}^{\text{global average pooling}} \underbrace{\frac{\partial y^c}{\partial A_{ij}^k}}_{\text{gradients via backprop}}$$

- Resulting coarse heatmap:

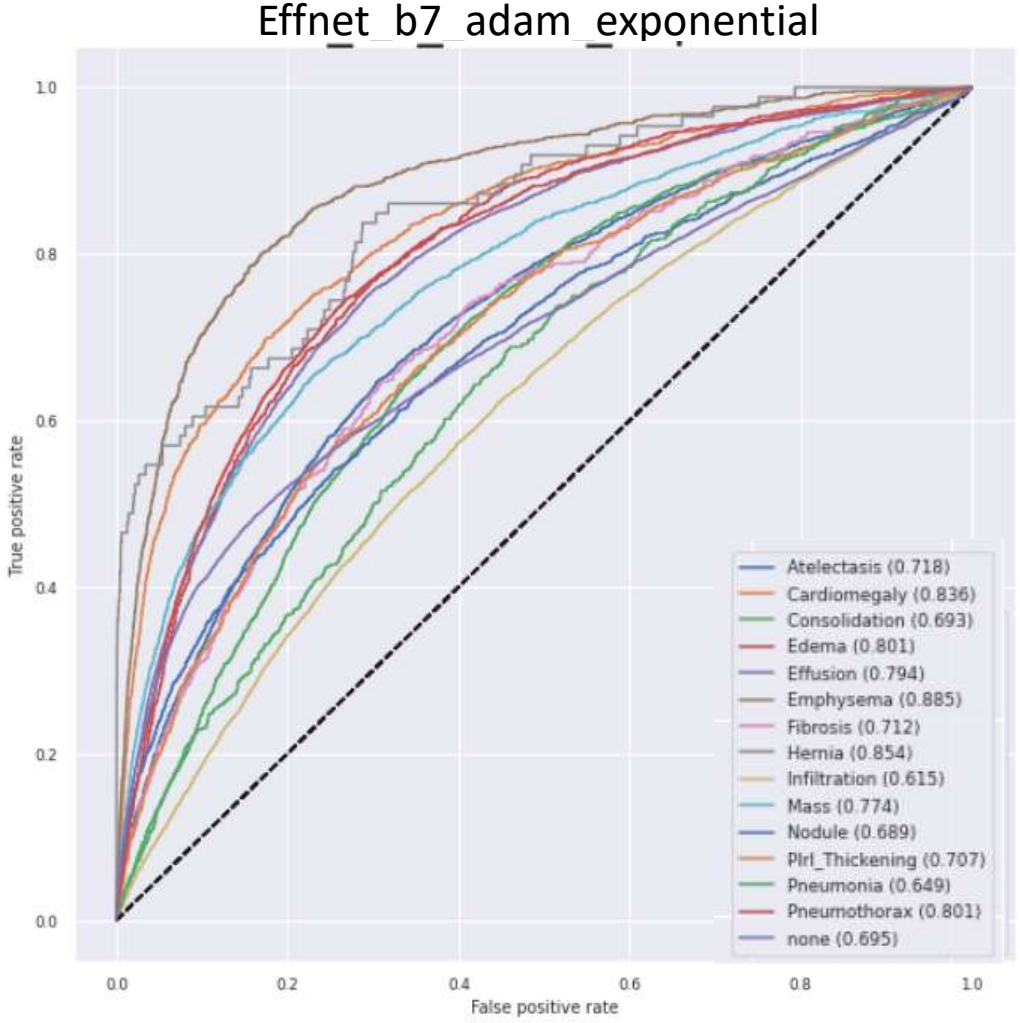
$$L^c_{\text{Grad-CAM}} = ReLU\underbrace{\left(\sum_k \alpha_k^c A^k\right)}_{\text{linear combination}}$$

https://arxiv.org/abs/1610.02391

# 2.3 Final Model

- Fine-tuned  Effnet_b7 pretrained on ImageNet
  - Optimizer: Adam
  - Learning Rate: 0.0001
  - Learning Rate Scheduler
    - Exponential
    - Gamma: 0.5
  - Epochs: 8
  - Average AUC score of 0.75 on 14 disease categories (averaged over 5 folds)

# 2.4 Final Model AUC Scores



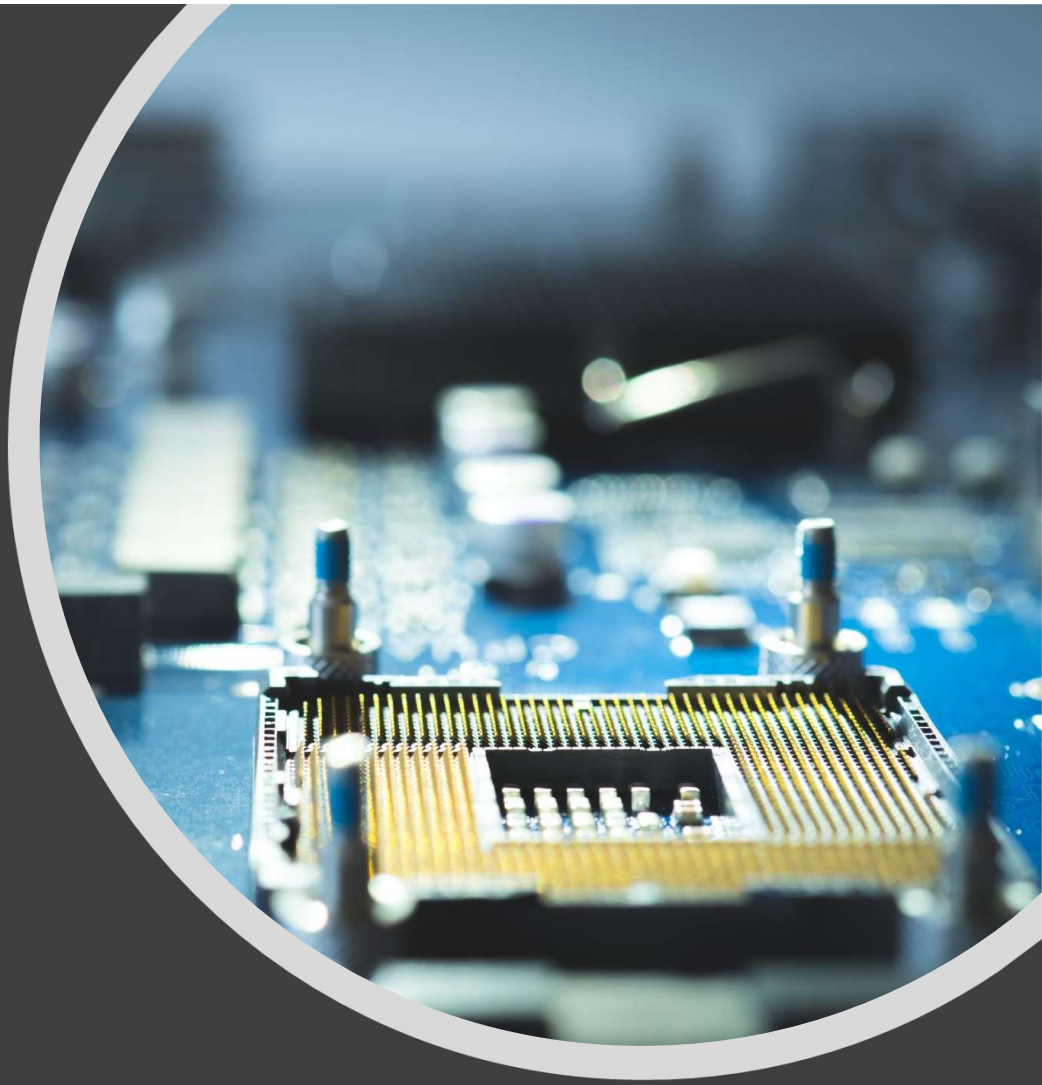
# 2.5 Efficient Pipeline

- Ability to change architecture and evaluate it seamlessly
- Allows training different neural networks
- Can be fine-tuned with a specific training regime
- Optimized it for speed
- Open sourced our code

- Code: PaulStryck/nih-chest-x-ray (github.com)

# 3. Our Approach

1. General Approach
2. Network Architectures
3. Learning Parameter Adjustments
4. 5-fold cross-validation to measure train/validation split impact
5. Data Augmentation to prevent early overfitting
6. Model Evaluation

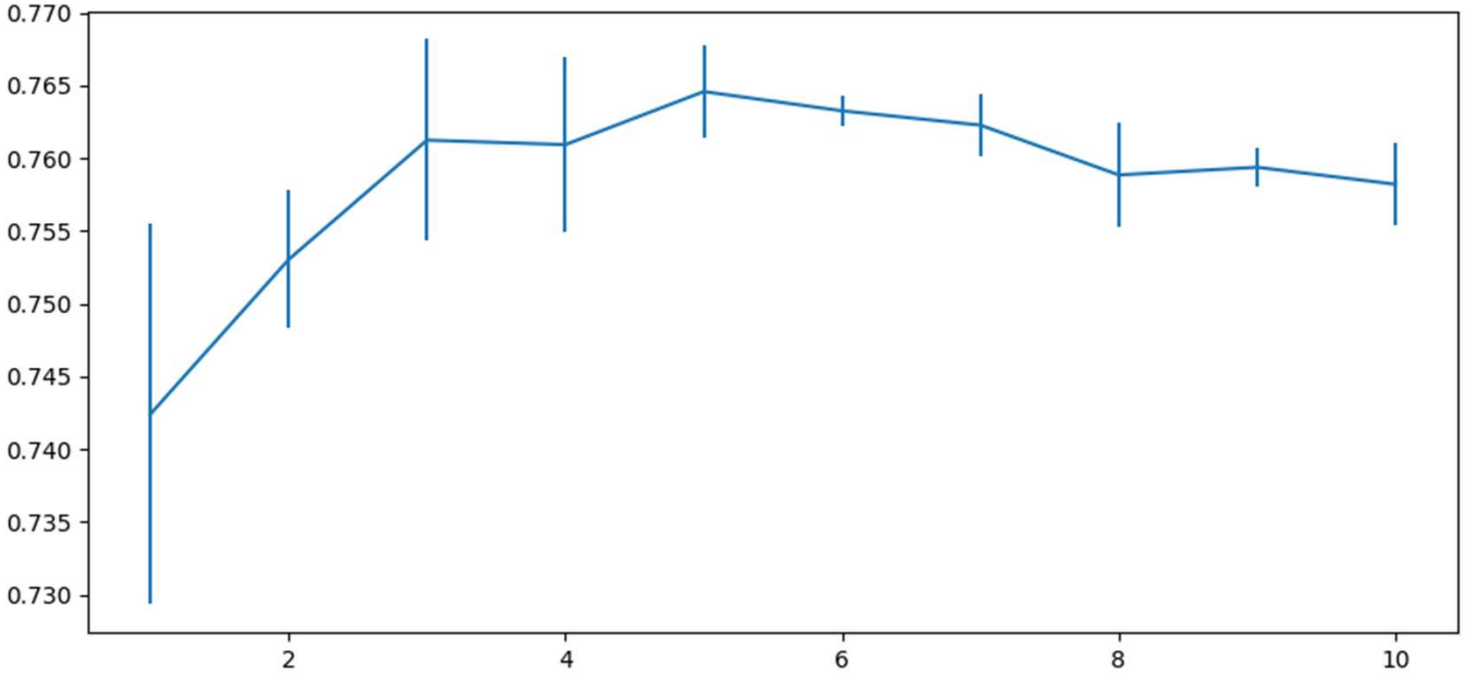


# 3.1 General Approach

- Empirical evaluation of many different NNs and hyperparameters

- Select NN
- Specify hyperparameters
  - Optimizer
  - Scheduler
  - Initial learning rate
  - Loss function in final layer
  - Potentially training different layers each epoch
- 5-fold CV training
- Evaluate network on predefined metrics
  - AUC score of receiver operating curve
- Repeat

# Example Output of Evaluation Script after Training Run




Average AUC score per epoch with std dev for 5 folds,
for ResNet50 with Adam Optimizer and StepLR

# Tuning

- Demand for a good Pipeline
  - Need a good pipeline where we can adjust these parameters independently and get results quickly
  - Implemented evaluation script to get comparable results
  - Highly optimised training infrastructure
    - Initial Keras implementation @ 0.2s/image
    - Final PyTorch implementation @ 0.002s/image
    - From 4:30h/epoch down to <5 min/epoch

# 3.2 Network Selection

- Not enough data to train deep NN from scratch
  - Thus: Fine-tune an existing network
- Choose pre-trained network and adjust final classification layer to our needs

- Literature review
  - Select previously well performing network architectures on NIH dataset
  - Recently published networks, performing well on ImageNet

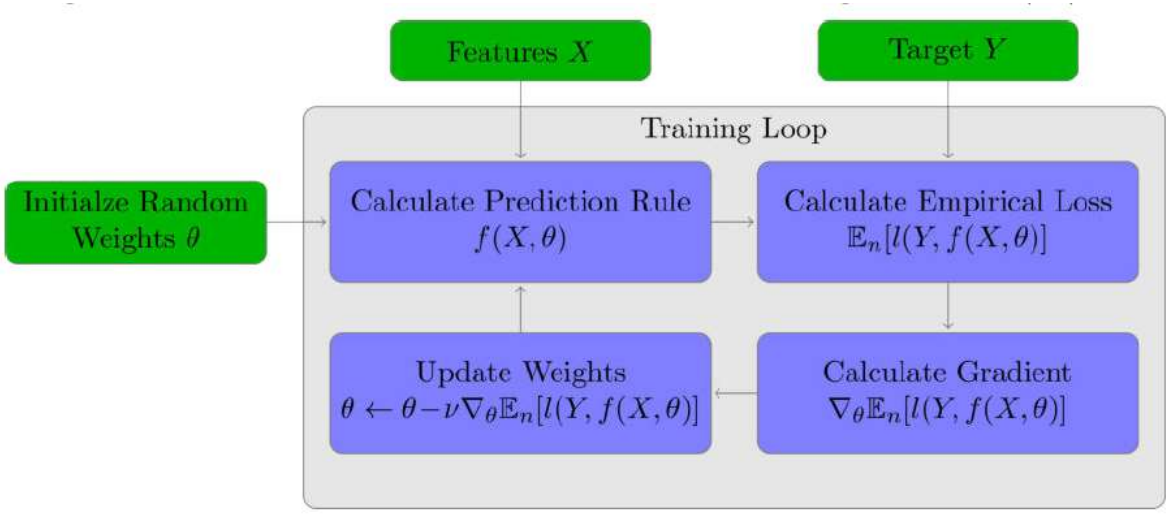
# 3.2 Network Selection

- Networks previously used successfully for NIH Chest X-Ray dataset
  - ResNet 50, 34, 18
  - DenseNet 161

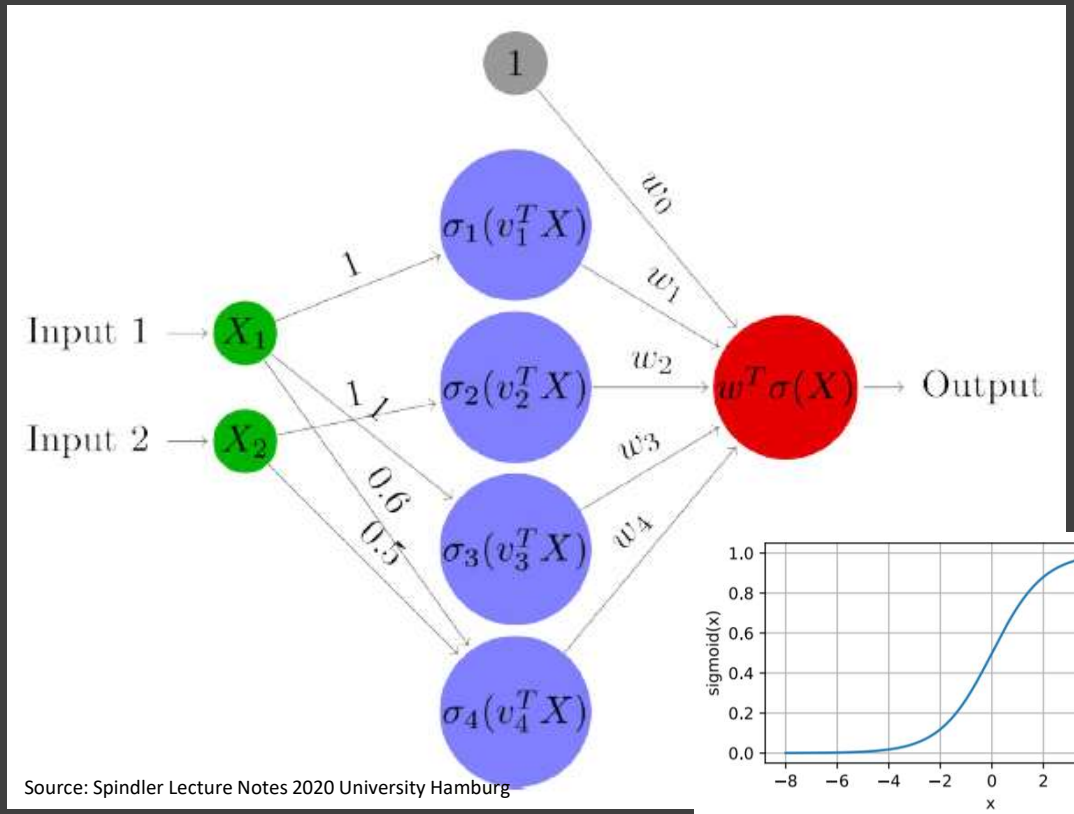
- Recent top performing NNs on ImageNet
  - EfficientNet B0, B7
  - GoogleNet

# 3.3 Learning Parameter Adjustments

1. Activation Functions
2. Schedulers
3. Loss Functions
4. Optimizers



# 3.3.1 Activation Functions




Source: Spindler Lecture Notes 2020 University Hamburg




$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$

- $\sigma$ is a threshold function
- Acertain input is required before neuro creates output
  - Different weights $w$ result in a different function of $X$
- Can construct several distinct transformed features by combining multiple of these basic building blocks
- NIH X-Ray: Patient can have multiple diseases/symptoms.
  - I.e. Effusion|Infiltration or Atelectasis | Effusion | Infiltration
- Last Layer: Sigmoid vs. Softmax
  - Binaray prediction: Sigmoid looks at each raw output value independently
  - Multi-Prediction: predict probabilities add up to 1
  - Chosen: Sigmoid

# 3.3.2 Schedulers

- Assist optimizer when closing in on the minimum

- StepLR
  - Reduce LR by gamma after n epochs
- Exponential Decay
  - Reduce LR by gamma after each epoch
- Reduce on Plateau
  - Reduce LR by gamma after validation loss stagnates for n epochs

# 3.3.3 Loss Functions

- Varying loss functions in final classification layer
- Class imbalance issues may be mitigated by loss function
- Considered loss functions after literature review
  - Binary Cross Entropy
  - Weighted Cross Entropy
  - Hamming Loss
  - Focal Loss

# Binary Cross Entropy Loss Function

$$\mathcal{L}(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log(p_{ij})$$
$$= -\frac{1}{n}\sum_{i=1}^{n}[y_i\log(p_i) + (1-y_i)\log(1-p_i)]$$

- Want to approximate a function $f0$
- Let $\mathcal{X}$ be the input space and $\mathcal{Y}$ be the target
- Learn a mapping $f_0 : \mathcal{X} \to \mathcal{Y}$, $x \mapsto y$.
- How $\mathcal{X}$,$\mathcal{Y}$ and $f0(\cdot)$ look like depends on the learning problem

$$f_0(x) := \mathbb{E}[Y|X = x].$$

- Assume that $f0$ is minimizing some average empirical loss $l$

$$f_0 := \arg\min_f \mathbb{E}[l(Y, f(X))].$$

- Classification setting: binary classification problem with $\mathcal{Y}$ = {0, 1}. Learn the conditional probability:

$$f(x) = P(Y = 1|X = x),$$

- Minimize loss

$$-\log\left(\prod_{i=1}^{n} P(Y_i|X_i)\right) = \sum_{i=1}^{n} \underbrace{-Y_1\log(f(X_i)) - (1-Y_i)\log(1-f(X_i))}_{=l(Y_i, f(X_i))},$$

- Final predicted class motivated by Bayes Classifier

$$\arg\max_{k\in\{1,\ldots,K\}} P(Y = k|X = x)$$

# Risk Minimization with Gradient Descent

- A neural network is learning a parametrized function

$$f : \mathcal{X} \times \Theta \to \mathcal{Y}$$
$$(x, \theta) \mapsto y,$$

- Where the $f$ is a fixed function and $\theta \in$ is parameter vector
- Learn the parameter vector $\theta$, which minimizes the risk (average loss)

$$\theta_0 := \arg\min_{\theta \in \Theta} \mathbb{E}[l(Y, f(X, \theta))].$$

- Since we do not know the distribution of ($Y$,$X$), we instead rely on the empirical measure to estimate $\theta$:

$$\arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} l(Y_i, f(X_i, \theta))$$

- This has no analytic solution for most cases therefore rely on gradient descent to minimize loss
- Take gradient of the loss of a random sample

$$\nabla_\theta \mathbb{E}[l(Y, f(X, \theta))].$$

- Gradient descent methods update the value of $\theta$ in the direction of the negative gradient

# 3.3.4 Optimizers

- Stochastic Gradient Descent
  - As the most common selection
- ADAM
  - For its good empirical results

# Adam

- Stochastic gradient decent method based on estimations of the first and second moments of the gradients
- Combines the advantages of AdaGrad and RMSProp

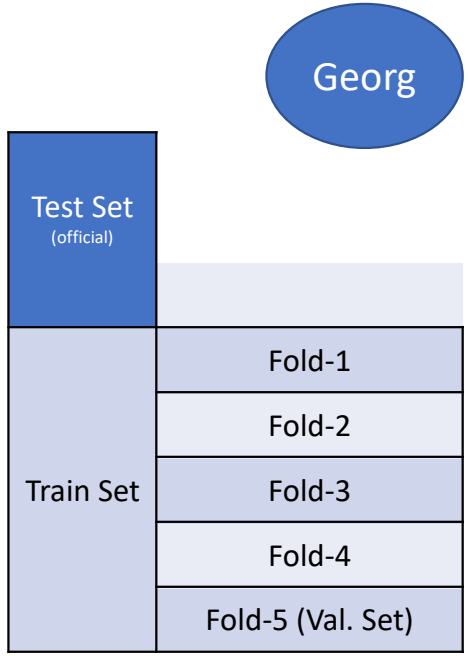
**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. $g_t^2$ indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

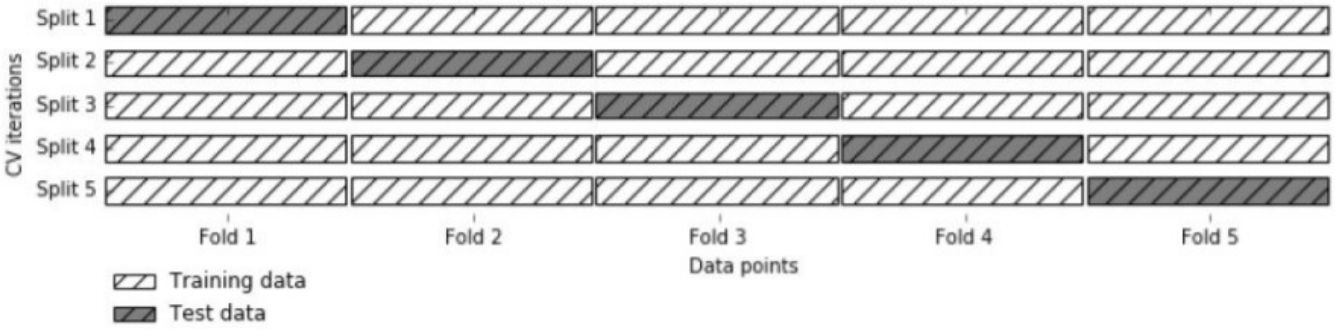
**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)

https://arxiv.org/pdf/1412.6980.pdf

# 3.4 5-Fold Cross-Validation

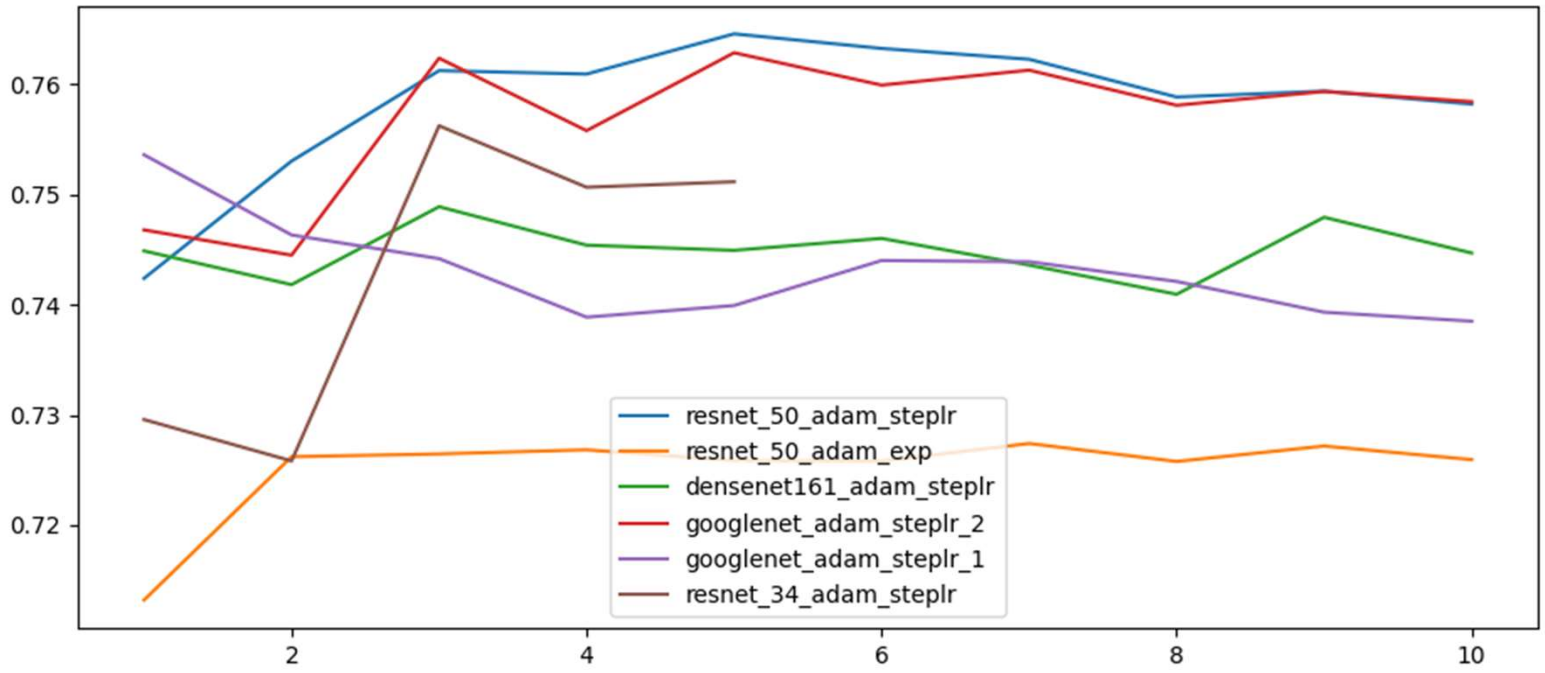
- Only small number of observations available
- Use k-fold cross-validation
  - Divide into 5 disjoint subsets of equal size
- Learn model 5 times
  - Alternate test set fold
- Average all 5 testing errors
- Group Split
  - Some patients more than one image
  - Split so images of patient are in the same split



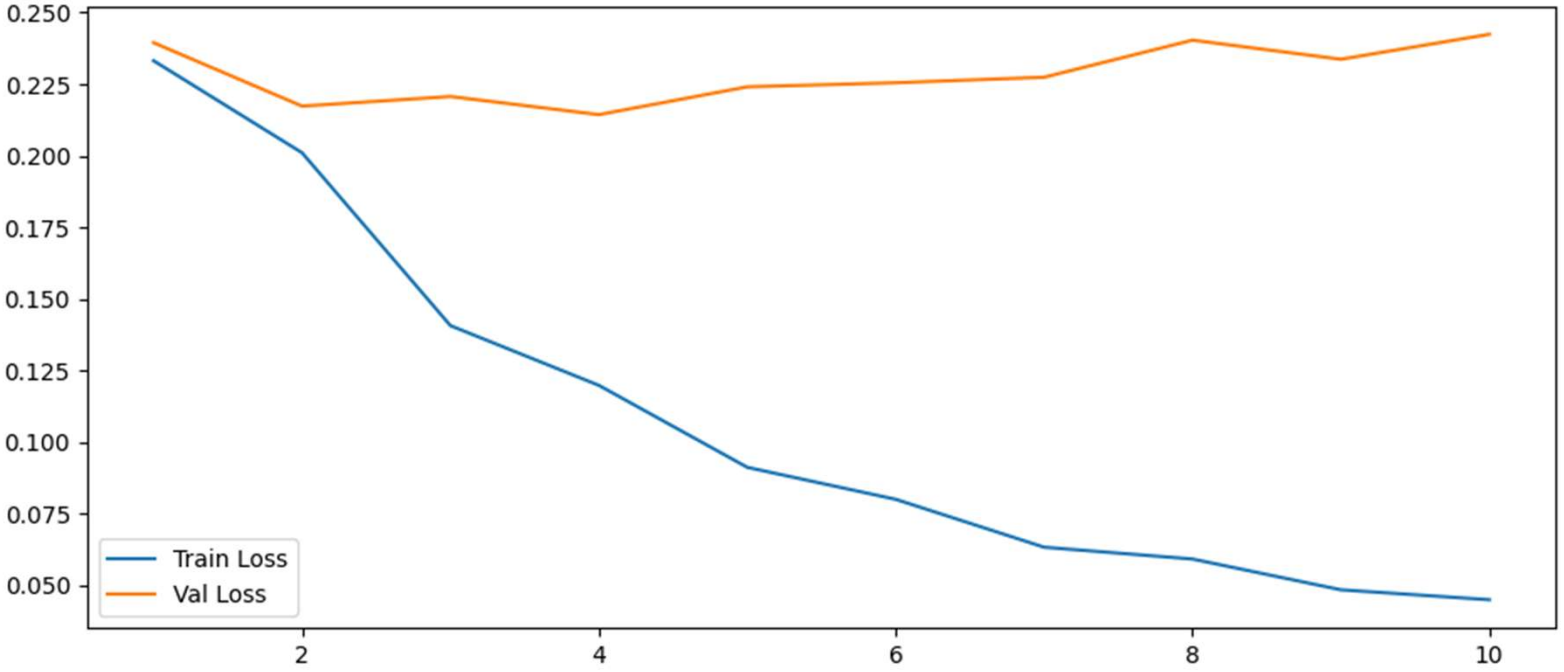

Source: Mueller and Guido (2017)

# Evaluation Script after Multiple Training Runs




Average AUC score per epoch averaged over 5 folds

# Evaluation Script after Individual Training Run




Validation vs Training Loss on Densenet161

# 3.5 Overfitting issues

- Data Augmentation
- Up/Down sampling

# 3.5.1 Data Augmentation

- Random changes to the training images producing similar, yet different, training examples
- Increases the size of the training dataset
- Used transformations:
  - Resizing
  - Random horizontal flips
  - Random rotations (-7 to 7 degrees)
- Horizontal flips and small rotations occur naturally in X-Rays
- Further augmentations would be purely artificial

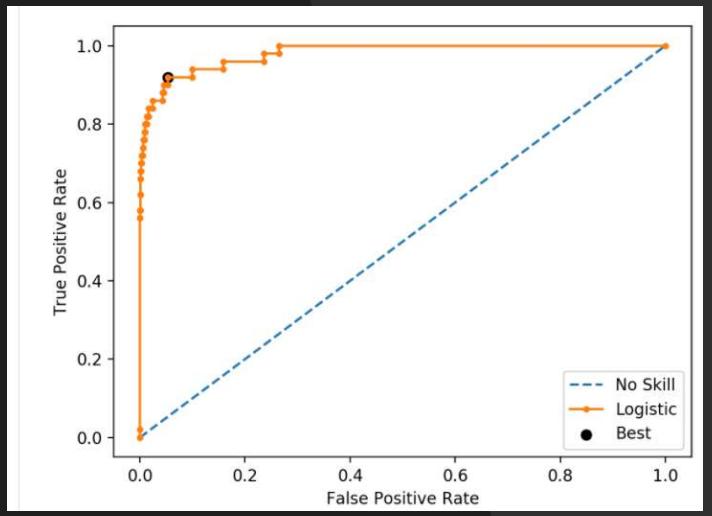
# 3.5.2 Up/Down Sampling

- High class imbalances lead to biased NN
- Artificially increase samples from infrequent categories
- Decrease samples from frequent categories
- Implemented via weighted random sampler in pytorch

# 3.6 Evaluation

- Different measurements taken
  1. Comparison table with variation
  2. Confusion Matrices
  3. AUC vs. Epoch
  4. ROC Curves

# ROC Curves and Confusion Matrices



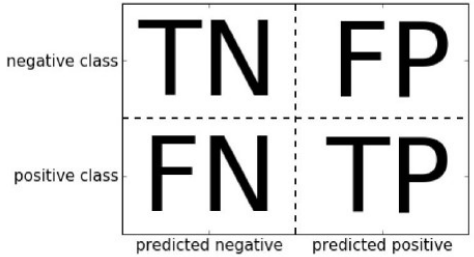

- Receiver Operating Characteristic (ROC)
  - Plot the TPR against the FPR

$$\text{Sensitivity} = TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{TN + FP}$$

    - Deals with skewed sample distributions
    - Avoids overfitting to a single class: rare diseases
  - Optimal threshold for ROC curve
    - Balances true positive and true negative rate similarly

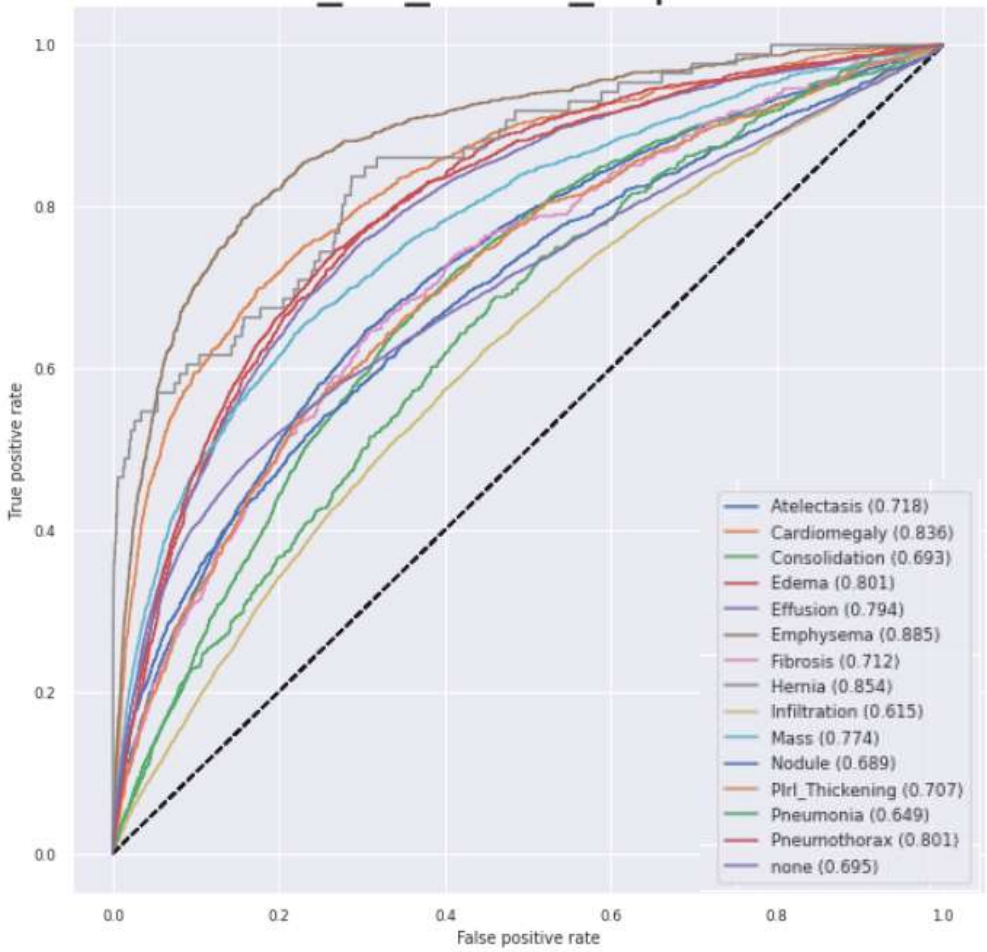
$$J = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} + \frac{\text{true negatives}}{\text{true negatives} + \text{false positives}} - 1$$

- Confusion Matrices
  - Bias towards FP preferable



# Effnet_b7_adam_exponential

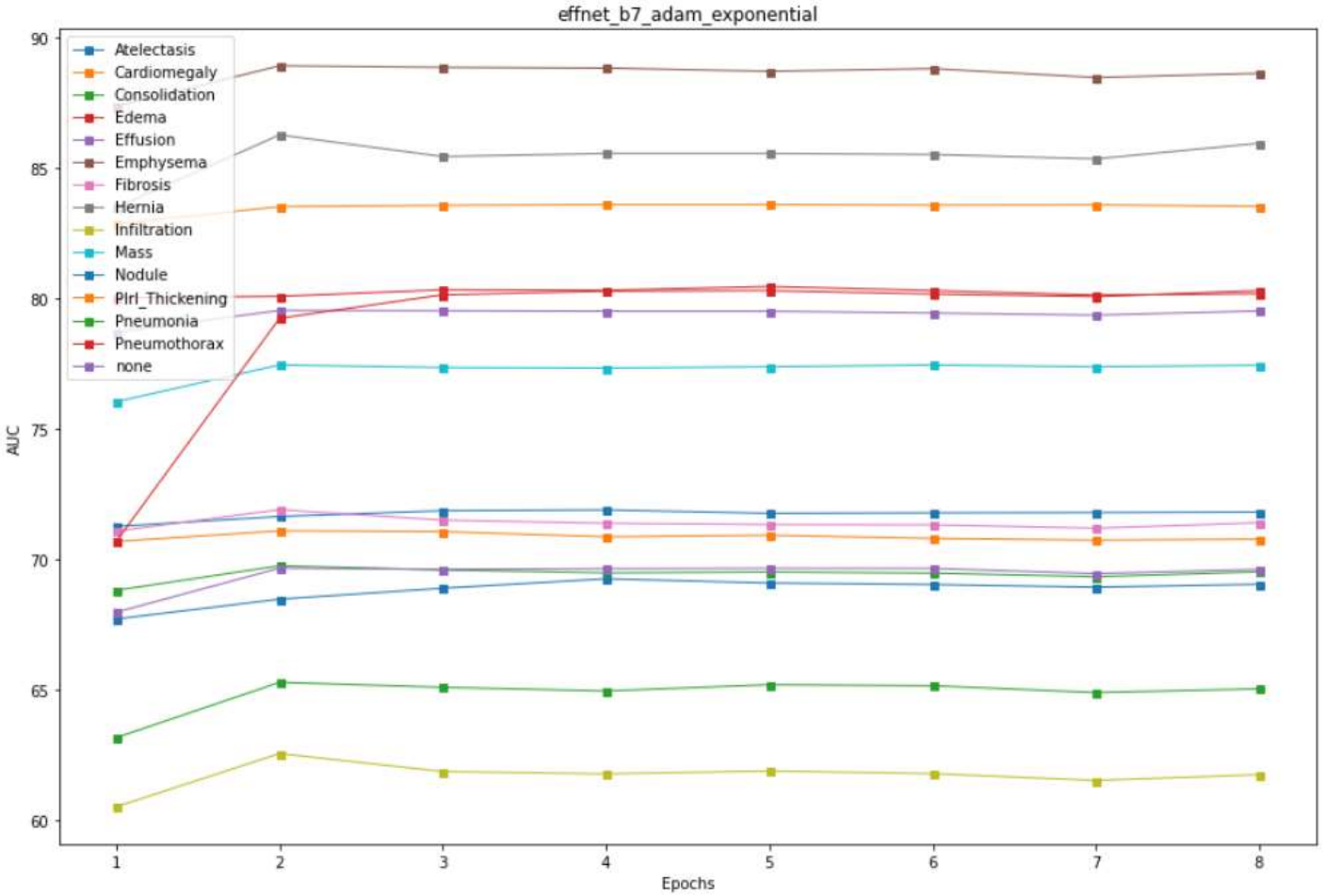
# Effnet _b7_adam_ exponential



# Comparison Table of All Trained Networks

| Pathology | effnet_b7_sgd_steplr | effnet_b7_adam_steplr | resnet_50_adam_exponential | resnet_50_adam_steplr | effnet_b0_adam_steplr | googlenet_adam_steplr_1 | densenet161_adam_steplr | effnet_b0_sgd_steplr | googlenet_adam_steplr_2 | resnet_34_adam_steplr | effnet_b7_adam_exponential |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Atelectasis** | 67.9 | 62.7 | 68.8 | 71.0 | 68.7 | 71.1 | 67.5 | 70.7 | 71.6 | 70.6 | 71.6 |
| **Cardiomegaly** | 79.3 | 85.6 | 83.5 | 82.5 | 85.1 | 82.0 | 84.0 | 81.8 | 84.9 | 85.0 | 83.5 |
| **Consolidation** | 68.6 | 69.7 | 69.9 | 65.8 | 68.6 | 70.1 | 71.1 | 70.2 | 68.4 | 69.3 | 69.8 |
| **Edema** | 80.3 | 77.5 | 79.7 | 81.2 | 81.3 | 82.1 | 77.8 | 82.5 | 80.0 | 80.3 | 80.1 |
| **Effusion** | 76.0 | 79.2 | 79.5 | 77.5 | 77.7 | 79.2 | 79.8 | 77.3 | 79.8 | 79.5 | 79.6 |
| **Emphysema** | 80.6 | 81.4 | 78.8 | 87.2 | 85.2 | 82.0 | 83.2 | 82.6 | 83.7 | 82.1 | 88.9 |
| **Fibrosis** | 74.4 | 75.5 | 73.4 | 73.0 | 76.1 | 76.7 | 76.1 | 78.7 | 76.6 | 76.1 | 71.9 |
| **Hernia** | 83.3 | 84.4 | 85.0 | 84.0 | 84.6 | 82.8 | 86.8 | 86.9 | 82.2 | 89.5 | 86.3 |
| **Infiltration** | 61.5 | 62.1 | 59.8 | 62.1 | 62.2 | 62.6 | 66.2 | 60.7 | 62.6 | 61.5 | 62.6 |
| **Mass** | 67.6 | 65.1 | 70.0 | 74.2 | 71.2 | 72.4 | 72.0 | 71.1 | 74.9 | 74.9 | 77.5 |
| **Nodule** | 62.4 | 59.3 | 63.6 | 66.2 | 65.8 | 67.2 | 66.9 | 65.5 | 66.3 | 66.5 | 68.5 |
| **Plrl_Thickening** | 68.9 | 69.2 | 70.4 | 70.3 | 69.0 | 72.6 | 70.5 | 70.1 | 68.2 | 69.2 | 71.1 |
| **Pneumonia** | 59.6 | 59.2 | 59.2 | 63.6 | 62.2 | 61.4 | 61.0 | 62.4 | 64.4 | 62.4 | 65.3 |
| **Pneumothorax** | 79.5 | 77.3 | 77.5 | 81.4 | 81.9 | 80.1 | 79.1 | 82.3 | 81.1 | 78.6 | 79.2 |
| **none** | 68.0 | 70.0 | 70.0 | 68.5 | 69.7 | 69.6 | 70.1 | 69.4 | 69.4 | 70.9 | 69.7 |

# Effnet_b7_adam_exponential



# Literature


1. Baltruschat, M., Nickisch, H., Grass, M., Knopp, T. & Saalbach, A. (2019): *Comparison of Deep Learning Approaches for Multi-Label Chest X-Ray Classification.* Nature Scientific Reports
2. Müller, A. & Guido, S. (2016): *Introduction to Machine Learning with Python*. O'Reilly Media, Sebastopol, 1st edition.
3. Spindler, M. (2020), Lecutre 2: Baisics of Deep Learning [PDF]. *Deep Learning an Introduction*, University of Hamburg, delivered 11. November 2020.