Deep Learning for Pneumonia Diagnosis: Models and Insights

Iakovos Tsalparas
IHU Dept. of Data Science
The Hague, The Netherlands
itsalparas@ihu.edu.gr

Nikolaos Mavriopoulos
IHU Dept. of Data Science
Larissa, Greece
nmavriopoulos@ihu.edu.gr

ABSTRACT

This report presents a comprehensive analysis of several machine learning models developed for a classification task.

The task involves predicting the class labels for a given dataset of X- ray images into 3 categories: normal, bacterial pneumonia and viral pneumonia. Various deep learning architectures models, including EfficientNet, MobileNet, and EfficientNetV2, were employed to address this classification problem. The report outlines the preprocessing steps, model development process, experimental results, and discussions on the findings.

Through extensive experimentation and evaluation, the report aims to provide insights into the effectiveness of different models and approaches for the given classification task to predict by an X-ray if a person is healthy, has bacterial pneumonia or viral pneumonia.

1. INTRODUCTION

Pneumonia is an infection that remains a significant global health concern, particularly affecting vulnerable populations such as children, the elderly, and individuals that have reduced immune systems for different kind of reasons. Accurate diagnosis of pneumonia is critical for timely treatment and reducing associated morbidity and mortality.

Traditional methods of pneumonia diagnosis, including physical examination, laboratory tests, and chest X-rays, are valuable but often time-consuming and require specialized expertise. In recent years, deep learning techniques has offered promising ways for enhancing pneumonia diagnosis through automated image analysis of X-rays.

This report explores the application of deep learning models for pneumonia diagnosis, focusing on the utilization of convolutional neural networks (CNNs) trained on medical imaging data like X-rays. Specifically, we look how the performance of various CNN architectures, including different types of EfficientNet, EfficientNetV2, MobileNet, and MobileNetV2, is in accurately detecting pneumonia from a given dataset of chest X-ray images.

Through a systematic evaluation of multiple deep learning models, we aim to evaluate their efficacy, identify strengths and limitations, and provide insights into optimizing model performance. Also, we analyse the impact of model architecture, hyperparameters, and training strategies on diagnostic accuracy.

Because we had a high imbalanced dataset of chest X-rays images, to make all models more robust and improve generalization we employed data augmentation for the training and the test data at start with batch size 32 for some models and with batch size 40 and 45 for some other models to the finish. We did that by increasing the diversity and complexity of augmentation techniques.

Our aim is that the findings of this report may contribute to the body of literature on deep learning-based approaches for pneumonia diagnosis and help improve clinical decision-making and patient outcomes. Ultimately, the goal is to facilitate the development of robust and scalable deep learning models that can support healthcare professionals in diagnosing pneumonia more effectively.

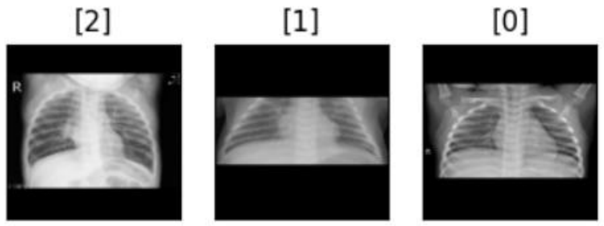

Fig. 1. An example of training images of the dataset where:
class_id = 0 if the image corresponds to a subject without disease
class_id = 1 if the image corresponds to a patient with bacterial pneumonia
class_id = 2 if the image corresponds to a patient with viral pneumonia

2. MODELS DESCRIPTION

The following CNN models has been used for the final submission after soft voting in order to improve the final accuracy. They represent a wide range of architectures, each offering unique advantages and trade-offs in terms of performance, efficiency, and deployment considerations for image classification problems specifically.

All models have input shape: (224,224,3) because models are trained on datasets like ImageNet, where images are typically resized to 224x224. Using the same input shape ensures compatibility with these pre-trained weights, allowing for transfer learning. Also, this resolution manages a balance between having enough detail to capture important features in the image and keeping the computational complexity manageable. While larger input sizes can potentially improve model accuracy by capturing more detailed information, they come with increased computational costs. Using this we can have balance between speed and accuracy.

All models have Swish activation function, which is know for its effectiveness in deeper models compared to traditional functions like ReLu.

Batch normalization is applied after each dense layer in order to stabilize the training process by reducing internal covariate shift, which refers to the change in the distribution of layer inputs during training and allow the model to converge faster during training. This is because the optimization process is less prone to vanishing or exploding gradients, leading to more stable updates of model parameters. Batch normalization, also reduces the sensitivity of the models to weight initialization, allowing for the use of higher learning rates without the risk of diverging or failing to converge. This can accelerate the training process and make it less dependent on fine-tuning hyperparameters. Finally, by doing batch normalization helps the model generalize better to unseen data.

At every model we monitor for early stopping validation loss during training with patience of 13 epochs (the number of 13 is chosen as best after testing with other patience numbers) in order to stop, because it helps to identify the point at which the model's validation loss has stopped improving. Also, with that way we can save computational resources by stopping the training process when it's no longer making significant improvements and prevent overfitting. Validation loss is our guide for hyperparameter tuning too. By comparing the validation losses of models trained with different hyperparameters, we select the combination that manages the best generalization performance. Overall, mainly but not by rule, the models with the lowest validation loss were chosen as they demonstrate the best generalization ability on unseen data.

Also, for every model we set at Model Checkpoint Callback save best only to True so we can ensure that only the weights of the model that corresponds to the epoch with the best performance on the monitored metric (validation loss) are saved. This helps to save storage space and ensures that we retain the best-performing version of our model. In the Model Checkpoint callback, we set save weights only to True in order to save only the model's weights (i.e., the learned parameters) rather than the entire model too. This is preferred because it's more lightweight and easier to manage. At Early Stopping callback we set restore best weights to True to ensure that after early stopping is triggered and training is stopped prematurely, the model's weights are reverted to the state they were in when the model had the best performance on the monitored metric during training.

Other hyperparameters like the number of the epochs, L1 and L2 at Kernel Regularization, Dropout rate, all the hyperparameters of the Learning Rate Reduction of each model were chosen empirically after consecutives tests in order to have higher accuracy.

We used pre-trained models, because, they have already undergone training on large datasets, which often helps to prevent overfitting if they fine-tune them on a specific task, as they start with meaningful weight initializations. Also, at all models we used Adam optimizer with different learning rate and decay at some models because the results that we got were much better than using SGD or RMSProp**.**

The EfficientNetB1, B2 and B3 were selected because they are models known for their excellent performance and efficiency for image classification problems like ours. By choosing different variants like B1, B2, and B3, and testing different hyperparameters we explored models with varying depths and complexities, allowing us to have a balance between model performance and computational cost.

Furthermore, EfficientNetV2B0, B1, B3, M, M_3 were selected, because EfficientNetV2 is an evolution of the EfficientNet architecture, incorporating improvements to further enhance performance. Our selection of different variants in EfficientNetV2 allows us to compare the performance of these newer models against the original EfficientNet. Here, like before different hyperparameters were tested between these models in order to achieve the highest possible accuracy. Lastly, MobileNet were used because we wanted to have a model designed specifically for mobile and vision applications. By including MobileNet in our selection, we explored a lightweight model option suitable for resource-constrained environments. Also, we must say that the combination of those models after constant testing with different hyperparameters, with the time we had, proved to be the best and that's why we keep only them for the final soft voting of the classification problem.

More versions of those models (EfficientNetB0, EfficientNetB2, EfficientNet3_2, EfficientNetV2B2 and MobileNetV2) with different or same hyperparameters and other models (VGG16, ResNet and AlexNet) were tested but removed from final soft voting because they dropped the final accuracy significantly or they tended to overfit to the dataset.

1.1 EfficientNet

It is first introduced in Tan and Le, 2019 is among the most efficient models that reaches State-of-the-Art accuracy on common image classification tasks. By introducing a heuristic way to scale the model, EfficientNet provides a family of models (B0 to B7) that represents a good combination of efficiency and accuracy. Each model in this series is progressively deeper, and wider, and has a higher resolution than its predecessor, allowing for incremental performance improvements. For all EfficientNet models the parameters are:

Total params: 4973734 (18.97 MB)
Trainable params: 4929663 (18.81 MB)
Non-trainable params: 44071 (172.16 KB)

EfficientNetB1:

- **Architecture**: This classification model is optionally loaded with weights pre-trained on ImageNet. It is based on a scaling method that uniformly scales network depth, width, and resolution. It consists of multiple blocks, each containing convolutional layers, batch normalization, and Swish activation functions. His architecture is designed to automatically learn and extract hierarchical features from images, making them well-suited for tasks like object detection, classification, and segmentation.
- **Advantages**: It achieves state-of-the-art performance on various image classification problems with fewer parameters compared to traditional CNNs. Its efficient use of resources makes it suitable for deployment on resource-constrained devices.

EfficientNetB3:

- **Architecture**: It follows the same architecture principles as EfficientNetB1 but with a larger network size, offering increased capacity and potentially higher accuracy.
- **Advantages**: It offers higher capacity and potentially better performance than EfficientNetB1 due to its larger size.
- **Training Process**: The training process for is similar to that of EfficientNetB1.

EfficientNetV2B0:

Released in 2021, EfficientNetV2B0 is an updated version of the EfficientNet architecture, having advancements in architecture design and training techniques. It retains the efficiency and effectiveness of the original EfficientNet while further improving performance on various computer vision benchmarks. It is designed to be more robust and scalable.

- **Architecture**: It is an improved version of the EfficientNet architecture.
- **Advantages**: It offers improved performance over the original EfficientNet architecture, thanks to its redesigned layers and attention mechanism.

EfficientNetV2B1:

- **Architecture**: Similar to EfficientNetV2B0 but with a larger model size, offering increased capacity and potentially better performance.
- **Advantages**: It provides higher capacity and potentially better performance than EfficientNetV2B0 due to its larger size.

EfficientNetV2B3:

- **Architecture**: With further increased depth and width, EfficientNetV2B3 pushes the boundaries of performance for the EfficientNet family. It is designed to handle more complex patterns and features in images, making it well-suited for demanding tasks such as fine-grained classification and object detection.
- **Advantages**: It provides higher capacity and potentially better performance than EfficientNetV2B0 and V2B1 due to its larger size.

EfficientNetV2M:

- **Architecture**: Targeted at mobile and embedded devices, EfficientNetV2M is optimized for deployment on resource-constrained platforms. It maintains a good balance between performance and model size, enabling efficient inference on devices with limited computational resources.
- **Advantages**: It offers a more lightweight architecture compared to other variants, making it suitable for deployment on mobile and edge devices.

1.2 MobileNet

- **Architecture**: Developed in 2017, MobileNet is designed for mobile and embedded vision applications. It utilizes depth-wise separable convolutions to reduce the computational cost while maintaining accuracy, making it ideal for real-time image recognition on devices with limited resources. It is a simple but efficient and not very computationally intensive convolutional neural networks for mobile vision applications. It is widely used in many real-world applications which includes object detection, fine-grained classifications, face attributes, and localization. For MobileNet the parameters are:
  Total params: 4021955 (15.34 MB)
  Trainable params: 3998019 (15.25 MB)
  Non-trainable params: 23936 (93.50 KB)
- **Advantages**: MobileNet offers lightweight and efficient inference, making it suitable for deployment on resource-constrained devices.

At the table below we show the hyperparameters of every model we used at the final soft voting:

MODEL	DROP OUT	ACTIVATION FUNCTION	OPTIMIZER	REGULA RIZATION	LOSS FUNCTION	EPOCHS	EARLY STOPING	LR REDUCTION
Efficient NetB1	0.3	Softmax	Adam with lr 1e-3, decay 1e-3*0.1	L1=1e-5 L2=1e-3	Categorical Cross-entropy	50	Patience of 25 epochs	Pat. of 2 ep., fact. of 0.3, min lr of 0.00000001
Efficient NetB3	0.3	Softmax	Adam Optimizer	L1=1e-5 L2=1e-3	Binary Cross-entropy	50	Patience of 22 epochs	Pat. of 2 ep., fact. of 0.3, min lr rate of 0.00000001
Efficient NetB3_1	0.2	Softmax	Adam Optimizer	L1=1e-6 L2=1e-4	Binary Cross-entropy	50	Patience of 21 epochs	Pat. of 2 ep., fact. of 0.3, min lr of 0.00000001
Efficient NetB3_3	0.5	Softmax	Adam Optimizer	L1=1e-5 L2=1e-3	Binary Cross-entropy	40	Patience of 29 epochs	Pat. of 2 ep., fact. of 0.3, min lr of 0.00000001
Efficient NetB3_4	0.3	Softmax	Adam Optimizer	L1=1e-5 L2=1e-3	Binary Cross-entropy	50	Patience of 22 epochs	Pat. of 2 ep., fact. of 0.3, min lr of 0.00000001
Efficient NetB3_5	0.5	Sigmoid	Adam Optimizer	L1=1e-5 L2=1e-3	Binary Cross-entropy	20	Patience of 20 epochs	Pat. of 2 ep., fact. of 0.3, min lr of 0.00000001
Efficient NetB3_6	0.2	Softmax	Adam Optimizer	L1=1e-5 L2=1e-3	Binary Cross-entropy	50	Patience of 21 epochs	Pat. of 2 ep., fac. of 0.3, min lr of 0.00000001
Efficient NetB3_7	0.3	Softmax	Adam Optimizer	L1=1e-5 L2=1e-3	Binary Cross-entropy	70	Patience of 21 epochs	Pat. of 2 ep., fact. of 0.3, min lr of 0.00000001
Efficient NetV2B0	0.3	Softmax	Adam Optimizer with lr if 1e-3, decay of 1e-3*0.1	L1=1e-5 L2=1e-3	Categorical Cross-entropy	25	Patience of 24 epochs	Patt. of 2 ep., fact. of 0.3, min lr of 0.00000001
Efficient NetV2B1	0.3	Softmax	Adam Optimizer with lr if 1e-3, decay of 1e-3*0.1	L1=1e-5 L2=1e-3	Categorical Cross-entropy	25	Patience of 23 epochs	Pat. of 2 ep., fact. of 0.3, min lr of 0.00000001
Efficient NetV2B3	0.3	Softmax	Adam Optimizer with lr if 1e-3, decay of 1e-3*0.1	L1=1e-5 L2=1e-3	Categorical Cross-entropy	30	Patience of 30 epochs	Pat. of 2 ep., fact. of 0.3, min lr of 0.00000001
Efficient NetV2M	0.2	Softmax	Adam Optimizer with lr if 1e-3, decay of 1e-3*0.1	L1=1e-5 L2=1e-3	Categorical Cross-entropy	50	Patience of 50 epochs	Pat. of 2 ep., fact. of 0.3, min lr of 0.000001
Efficient NetV2M_2	0.3	Softmax	Adam Optimizer with lr if 1e-3, decay of 1e-3*0.1	L1=1e-5 L2=1e-3	Categorical Cross-entropy	10	Patience of 10 epochs	Pat. of 2 ep., fact. of 0.3, min lr of 0.000001
Efficient NetV2M_3	0.5	Softmax	Adam Optimizer with lr if 1e-3, decay of 1e-3*0.1	L1=1e-5 L2=1e-3	Categorical Cross-entropy	40	Patience of 40 epochs	Pat. of 2 ep., fact. of 0.3, min lr of 0.000001
MOBILE NET	0.3	Softmax	Adam Optimizer with lr if 1e-3, decay of 1e-3*0.1	L1=1e-5 L2=1e-3	Categorical Cross-entropy	30	Patience of 26 epochs	Pat. of 2 ep., fact. of 0.3, min lr of 0.00000001

3. COMPARITIVE EXPERIMENTS AND RESULTS

We have trained multiple models trying each time different hyperparameters in order to solve our classification problem with the best possible accuracy. By judging on the results each time we found out that the best hyperparameters for this problem have the following best epoch during training of each model. Also we saved the weights of each epoch in order to do the final soft voting.

Model	Epoch	Loss	Acc	Val_Loss	Val_Ac
Efficient NetB1	19	0.0695	0.9862	0.6602	0.6808
Efficient NetB3	20	0.0493	0.9826	0.3763	0.8651
Efficient NetB3_1	16	0.0848	0.9843	0.4214	0.8587
Efficient NetB3_3	22	0.0428	0.9850	0.3380	0.8822
Efficient NetB3_4	21	0.0513	0.9788	0.3580	0.8522
Efficient NetB3_5	15	0.0805	0.9724	0.3380	0.8694
Efficient NetB3_6	17	0.0603	0.9738	0.3714	0.8608
Efficient NetB3_7	13	0.0782	0.9707	0.3105	0.8694
Efficient NetV2B0	18	0.1346	0.9658	0.6215	0.8501
Efficient NetV2B1	10	0.4418	0.8599	0.4663	0.8437
Efficient NetV2B3	26	0.5119	0.8221	0.4674	0.8373
Efficient NetV2M	28	0.1750	0.9907	0.6943	0.8694
Efficient NetV2M_2	10	0.6284	0.8996	0.7974	0.8415
Efficient NetV2M_3	40	0.2025	0.9717	0.5901	0.8672
MOBILE NET	18	0.0855	0.9872	0.5661	0.8694

After submitting each model separately, the accuracy score based on the test data on the Kaggle competition that each one has managed is the following:

EffnetB1 : 0.81678
EffnetB3 : 0.80993
EffnetB3_1 : 0.83390
EffnetB3_3 : 0.82020
EffnetB3_4 : 0.82705
EffnetB3_5 : 0.82876
EffnetB3_6 : 0.81678
EffnetB3_7 : 0.83390
EffnetV2B0 : 0.82363
EffnetV2B1 : 0.82020
EffnetV2B3 : 0.79965
EffnetV2M : 0.82363
Effnet V2M_2 : 0.81849
Effnet V2M_3 : 0.80308
MobileNet : 0.81506

As we can see the results of each model separately even after many tries of tuning right the hyperparameters has not been so good so in order to increase accuracy significantly. So in order to have improve accuracy we have tried soft voting by averaging the predictions of all those models to create an ensemble prediction. By doing that technique we improve the overall performance compared to using individual models alone by having total score of 0.84246 for the test data at Kaggle(50%).

4. CONCLUSIONS

In this classification problem, we have explored different machine learning models to accurately predict whether a person has pneumonia by virus, pneumonia by bacteria or he is healthy based on some X-rays. With experimentation and analysis, we found out that using ensemble methods has significantly enhanced the model's performance.

We have tried different ensemble strategies, but soft voting has been the most effective approach. By aggregating the probabilistic predictions from the multiple models we have trained before, soft voting provides a robust and good mechanism for producing final predictions. We can conclude that the by doing soft voting we yielded a notable improvement in classification accuracy compared to individual classifiers.

This has happened because by using soft voting we reduced the impact of the weaknesses inherited in any single model since the probabilistic averaging allowed the ensemble to make more balanced and informed predictions.

In conclusion, the use of soft voting as an ensemble method substantially improved the classification accuracy. This shows the importance of ensemble techniques in machine learning. Future work could explore further optimization of different models and investigate other ensemble methods to continue advancing the predictive performance of the model.

5. REFERENCES

[1] https://keras.io/api/applications/efficientnet/
[2] https://keras.io/api/applications/mobilenet/
[3] https://arxiv.org/abs/1905.11946
[4] https://blog.roboflow.com/what-is-efficientnet/
[5]https://github.com/codesteller/train-efficientnet-with-imagenet
[6]https://www.tensorflow.org/api_docs/python/tf/keras/applications/efficientnet
[7] https://paperswithcode.com/method/efficientnet
[8]https://en.wikipedia.org/wiki/Convolutional_neural_network
[9] https://arxiv.org/abs/2104.00298
[10]https://github.com/qubvel/efficientnet/blob/master/efficientnet/model.py
[11]https://keras.io/api/keras_cv/models/backbones/efficientnetv2/