Harmful Brain Activity Classification with Deep Learning

Loc Nguyen
Oregon State University
nguypham@oregonstate.edu

Amogh Sawant
Oregon State University
sawantam@oregonstate.edu

Ju Hyun Kim
Oregon State University
kimjuhyu@oregonstate.edu

Abstract

Electroencephalography (EEG) has emerged as a powerful tool for monitoring brain activity non-invasively, offering insights into various neurological disorders and cognitive processes. In recent years, the application of deep learning techniques to EEG data has shown promising results in tasks such as brain-computer interfacing, seizure detection, and cognitive state recognition. In this study, we train different deep learning models on the EEG data provided by Critical Care EEG Monitoring Research Consortium. The evaluation criteria for our model is based on the Kullback Liebler divergence between the predicted probability and the observed target.

1. Introduction

Recently, EEG monitoring are being done manually by specialized neurologists. However, this process is a major bottleneck for multiple reasons: labor intensive, expensive, prone to fatigue-related errors and suffers from reliability issues between different reviewers, even when those reviewers are experts. For this project, we aim to address this bottleneck and automate the process of EEG analysis using Deep Learning. This automation will help doctors and brain researchers detect seizures and other types of brain activity that can cause brain damage, so that they can give treatments more quickly and accurately. Our model will be used to generate prediction for submission to HMS - Harmful Brain Activity Classification hosted by Harvard Medical School on Kaggle.[3]

1.1. Electroencephalography - EEG

An electroencephalogram (EEG) is a test that measures electrical activity in the brain using small, metal discs (electrodes) attached to the scalp. Brain cells communicate via electrical impulses and are active all the time, even during asleep. For this project, we will use the EEG data provided by Critical Care EEG Monitoring Research Consortium to train different learning models. Some samples from the dataset can be found in the following Figure 1:

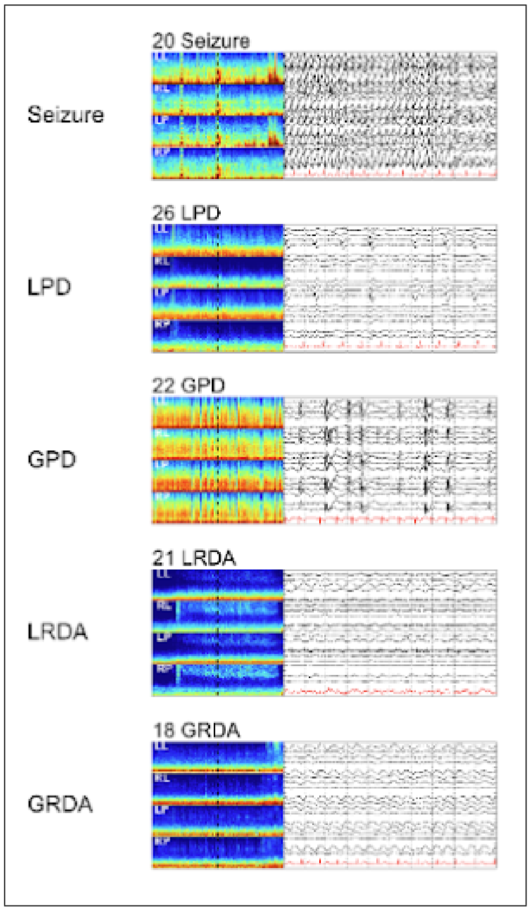


Figure 1: Sample EEGs from dataset and their labels.

1.2. EEG Classification with Deep Learning

There are six patterns of interest for this project: seizure (SZ), generalized periodic discharges (GPD), lateralized periodic discharges (LPD), lateralized rhythmic delta activity (LRDA), generalized rhythmic delta activity (GRDA), or "other". An example visualization of each pattern can be found in Figure 1. We used the data to train and predict the classifications of these patterns using the following deep learning model architectures: EfficientNetv2, Resnet50, YOLOv8.

1.3. Evaluation Metric: Kullback-Leibler divergence

Kullback-Leibler divergence [8] is a statistical measurement from information theory that is commonly used to quantify the difference between one probability distribution from a reference probability distribution. For each *eeg_id* in the test set, the model must predict a probability for each of the 6 patterns.

2. Related works

In this paper, we conduct experiments utilizing three distinct backbones for our foundational neural network: ResNet50v2 [1] (Residual Network with 50 layers, version 2), YOLOv8 [6](You Only Look Once, version 8), and EfficientNetv2b2 [10] (Efficient Network, version 2). Each of these networks employs pretrained weights from ImageNet [7], indicating that they have been pre-optimized for image classification tasks. In the following subsections, we will discuss the objectives and the unique advantages offered by these various backbones.

2.1. ResNet50v2

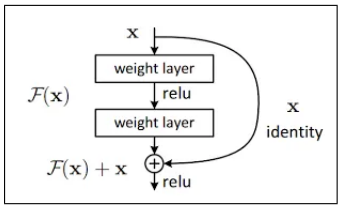

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2: Residual learning: a building a block [1]

Resnet50v2, an iteration of the original ResNet [1] (Residual Networks) architecture, is a convolutional neural network designed for image recognition and classification tasks. Resnet50v2 incorporates several improvements over its predecessor, primarily through the use of identity mapping in residual connections. This architecture comprises 50 layers and is known for its ability to solve the vanishing gradient problem, allowing for the training of deeper networks by utilizing shortcut connections that skip one or more layers. These shortcut connections enable the network to perform identity mapping, with the added layers learning residual functions with reference to the layer inputs, thus facilitating deeper network architectures without degradation in training performance. Resnet50v2 has demonstrated significant effectiveness in various image recognition benchmarks, making it a popular choice in the field of deep learning for computer vision.

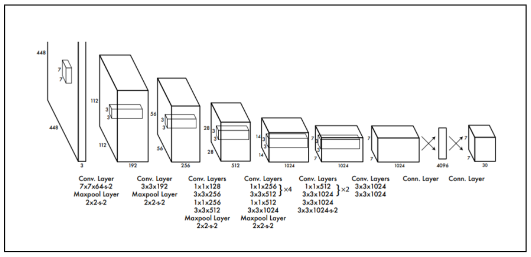


Figure 3: YOLO Architecture [6]

2.2. YOLOv8

YOLOv8 [6] (You Only Look Once, version 8) is the latest iteration in the YOLO [5] series of real-time object detection systems. Building on the principles of its predecessors, YOLOv8 further optimizes speed and accuracy in detecting objects within images. This model is designed to process images in a single pass, making predictions on various object attributes, including class probabilities and bounding box coordinates, thus enabling it to achieve real-time performance. YOLOv8 incorporates advancements in network architecture, training procedures, and algorithm optimizations, making it one of the fastest and most accurate object detection models available. It is especially well-regarded for its application in scenarios requiring real-time processing and high accuracy, such as surveillance, autonomous vehicles, and interactive systems.

2.3. EfficientNetv2b2

EfficientNetv2b2 is a part of the EfficientNetv2 [10] series, a next-generation architecture for convolutional neural networks that emphasizes efficiency and adaptability. Introduced by Tan and Le, EfficientNetv2 improves upon the original EfficientNet [9] models by optimizing training speed and model scaling. The ”v2b2” variant refers to a specific configuration within this series, designed to offer a balance between computational efficiency and model accuracy. EfficientNetv2 introduces innovations such as progressive learning, where the model uses smaller images at the start of training and gradually increases the resolution as training progresses. This approach, along with refinements in the model’s architecture, such as enhanced compound scaling and the use of fused convolution operations, enables EfficientNetv2 models to achieve higher accuracy with fewer parameters and reduced training time. EfficientNetv2b2, in particular, is tailored for applications where model efficiency is paramount without significantly compromising on the accuracy of tasks like image classification.

3. Technical Approach

Our primary objective is to construct a neural network capable of producing a probability distribution from input images. It is important that the output probability distribution closely approximates the ground truth distribution. Consequently, we have selected the Kullback-Leibler (KL) Divergence [8] as our loss function, aiming to minimize this loss as much as feasible. We use the term score and loss interchangeably from now on. In an attempt to develop an optimal model, we perform experiments on different architectures, each pre-trained for image classification by utilizing the ImageNet [7] weights, with different configuration. In this section, we explain the input data format, output data format, loss function and Learning Rate schedule in detail, laying a foundation for the reader.

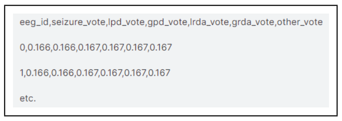
3.1. Input

The input to our model comprises EEG segments sourced from the Kaggle competition [3], which have undergone expert annotation or classification. These annotations vary in agreement levels among experts, ranging from complete consensus to disagreement. Segments with high consensus are termed ”idealized” patterns, while cases where experts are evenly split between labels are termed ”proto patterns.” Additionally, segments where experts are roughly divided between two labels are labeled as ”edge cases.”

These EEG segments are initially in image format and undergo preprocessing, including reducing the number of channels for improved model compatibility. Furthermore, data augmentation techniques such as masking frequency and temporal data are applied to augment the dataset. Image translation and random cutouts are also utilized to further enhance the dataset's diversity and aid in robust model training.

3.2. Output

The evaluation criterion for submissions relies on the Kullback-Liebler divergence between predicted probabilities and observed targets. For every EEG ID within the test set, predictions must be provided for each vote column. The submission file must include a header and have the specified format in Figure 4.



```
eeg_id,seizure_vote,lpd_vote,gpd_vote,lrda_vote,grda_vote,other_vote
0,0.166,0.166,0.167,0.167,0.167,0.167
1,0.166,0.166,0.167,0.167,0.167,0.167
etc.
```

Figure 4: Output Format

3.3. Loss Function

We decide to use KL divergence [8] because it is widely used to calculate the distance between two probability distributions, which is what we want to calculate. It measures how much one probability distribution diverges from other. Here, we do not need to calculate the accuracy as the goal is to minimize the distance between two distributions. We use the following definition of the KL divergence.

$$D_{KL}(P||Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

Here, $\mathcal{X}$ represents the set of all possible states of the variable x, and $P(x)$ and $Q(x)$ are the probability density functions of P and Q, respectively.

3.4. Learing Rate warm-up

Learning rate warm-up is a technique used in training deep learning models where the learning rate is gradually increased from a small value to its desired value at the beginning of training. This approach helps stabilize the training process by allowing the model to initially explore a wider range of parameter space, preventing it from getting stuck in poor local minima. By gradually ramping up the learning rate, the model can converge faster and more reliably to a good solution, especially when dealing with large, complex datasets or architectures. For this project, we'll be using the following learning rate schedule with warm up for the first 3 epochs from figure 5:

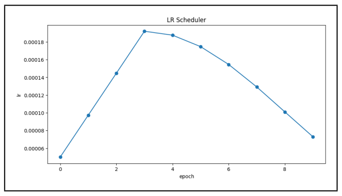


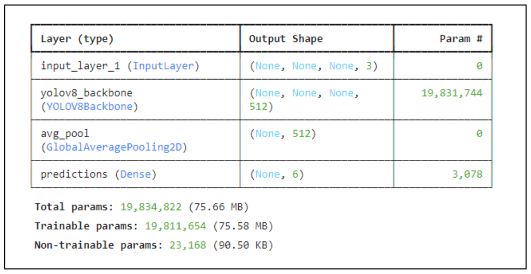
Figure 5: Learning rate schedule

4. Experiment

In this section, we provide an in-depth examination of the various network backbone architectures and configurations employed in our study. Our objective is to explain the methodology behind the selection and optimization of these models, aiming to achieve the highest performing model attainable. This involves a detailed discussion on the decision for choosing specific architectures, the adjustments made to enhance model performance, and the criteria used to evaluate the performance of each configuration. Note that we use keras [2] to utilize these various backbones for our models.

4.1. YOLOv8 backbone

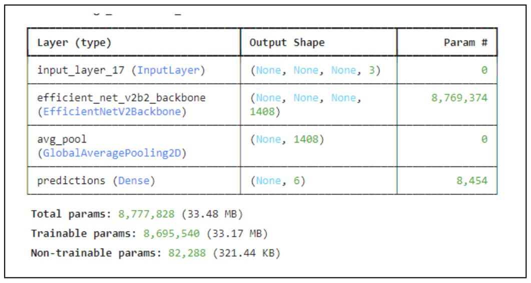
We began with the YOLOv8 backbone model [6], comprising over 19 million parameters. Table 1 outlines the structure of this model, including an input layer and an extensive YOLOv8 backbone, with a global average pooling layer. Notably, a connected layer was created for classification output. The substantial parameter count of the model



Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, None, None, 3)	0
yolov8_backbone (YOLOV8Backbone)	(None, None, None, 512)	19,831,744
avg_pool (GlobalAveragePooling2D)	(None, 512)	0
predictions (Dense)	(None, 6)	3,078

```
Total params: 19,834,822 (75.66 MB)
Trainable params: 19,811,654 (75.58 MB)
Non-trainable params: 23,168 (90.50 KB)
```

Table 1: YOLOv8 backbone model structure



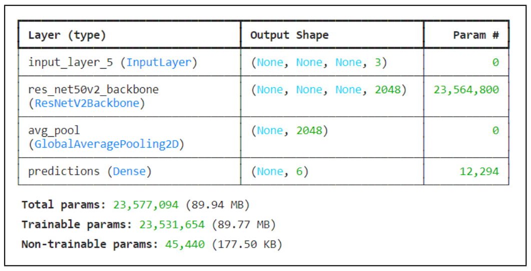
Layer (type)	Output Shape	Param #
input_layer_17 (InputLayer)	(None, None, None, 3)	0
efficient_net_v2b2_backbone (EfficientNetV2Backbone)	(None, None, None, 1408)	8,769,374
avg_pool (GlobalAveragePooling2D)	(None, 1408)	0
predictions (Dense)	(None, 6)	8,454

```
Total params: 8,777,828 (33.48 MB)
Trainable params: 8,695,540 (33.17 MB)
Non-trainable params: 82,288 (321.44 KB)
```

Table 3: EfficientNetv2b2 backbone model structure

exacerbated overfitting and prolonged our training sessions excessively. Despite training for 40 epochs, we observed minimal improvement in results, prompting concern. To balance memory constraints and gradient update stability, we set the batch size to 64.

4.2. ResNet50v2 backbone



Layer (type)	Output Shape	Param #
input_layer_5 (InputLayer)	(None, None, None, 3)	0
res_net50v2_backbone (ResNetV2Backbone)	(None, None, None, 2048)	23,564,800
avg_pool (GlobalAveragePooling2D)	(None, 2048)	0
predictions (Dense)	(None, 6)	12,294

```
Total params: 23,577,094 (89.94 MB)
Trainable params: 23,531,654 (89.77 MB)
Non-trainable params: 45,440 (177.50 KB)
```

Table 2: ResNet50 backbone model structure

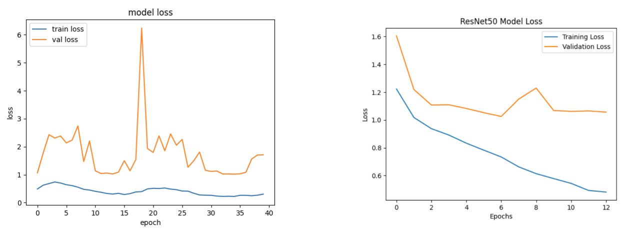
Table 2 displays the considerable depth of ResNet50 backbone, boasting over 23 million parameters. However, this size proved too large for our task, resulting in increased model complexity. The intricate structure of the model strained our computational resources during training. Despite attempting a batch size of 64, we encountered an 'out of memory' error due to the combined size of data batches and model parameters exceeding the GPU's memory capacity. Consequently, we reduced the batch size to 16, effectively reducing the memory load per training iteration.

4.3. EfficientNetv2b2 backbone

EfficientNetv2b2 [10] features a hierarchical architecture comprising multiple blocks of convolutional layers, each carefully designed to balance model complexity and computational efficiency. It incorporates novel improvements over its predecessor, EfficientNet [9], including enhanced depthwise separable convolutions, efficient attention mechanisms, and advanced normalization techniques. This architecture enables EfficientNetv2b2 to efficiently capture hierarchical features at different scales while minimizing computational overhead. Additionally, its modular design facilitates scalability, allowing for easy adjustment of model size and complexity to suit various computational resources and task requirements. The model architecture we're using is shown in Table 3 with over 8 millions parameters, which is significantly less compared to YOLOv8 and ResNet50v2 backbones. With this EfficientNetv2b2 backbone structure, we trained the model on three different batch size: 32, 64 and 128 and three different kernel sizes: 3x3, 5x5, 7x7.

4.4. Results

In our experiments, we find that the model using EfficientNetv2b2 backbone outperforms models using YOLOv8 [6] and ResNet50 [1] backbones by a large margin. The plots in Figure 6 shows the training progress of the YOLOv8 model over 40 epochs tuned in 64 batch size and ResNet50 model over 13 epochs tuned in 16 batch size.

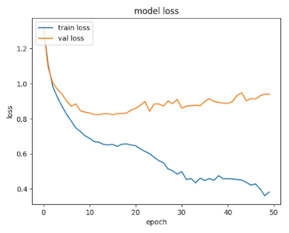


YOLOv8 ResNet50

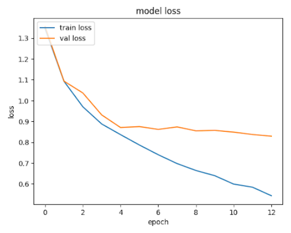
Figure 6: Training loss and Validation loss for YOLOv8 and ResNet50 backbone models

The structure of YOLOv8 model led to an arduous training process. The model's size and complexity presented substantial challenges in optimization. The extensive parameter count of the model contributed to increased complexity, making it challenging for the network to learn finer details in EEG images. Furthermore, the training process for YOLOv8 was notably prolonged, resulting in limited progress, with only a 0.92 loss achieved. Despite experimenting with varying epochs, our efforts yielded no substantial improvements. The combined factors of model size, training duration, and limited performance made it not ideal for our EEG image processing task.

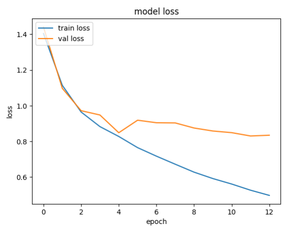
The results of the Resnet50 backbone model in Figure 6 show reductions in training loss, while the verification loss showed fluctuation patterns. These fluctuations can signal the onset of over-aggregation, where the model learns the training data too closely and fails to generalize it to new unseen data. Therefore, we got the final verification loss of 0.85. The complexity of the Resnet50 hindered the network's ability to effectively learn the finer details present in EEG images. Additionally, the reduced batch size for solving memory constraint problems has further hindered networks from learning and generalizing effectively. As a result, we encountered challenges in adequately training the model and achieving satisfactory performance on our EEG image dataset.



Batch size = 32



Batch size = 64

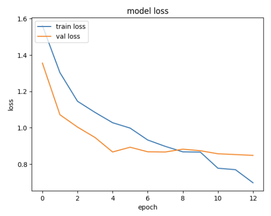


Batch size = 128

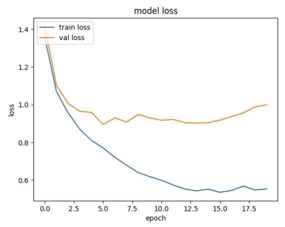
Figure 7: Training Loss and Validation Loss for batch size 32, 64 and 128 over 13 epochs from left to right for EfficientNet-v2b2

Training EfficientNetv2b2 offers distinct advantages in terms of cost-effectiveness, efficiency, and performance optimization. With its streamlined architecture and efficient utilization of computational resources, EfficientNetv2b2 enables faster training times compared to other architectures, making it a cost-effective choice for large-scale image classification tasks. Shown in figure 7 are the model training and validation losses over 13 epochs of when training the model on batch size 32, 64, and 128. Notably, when trained with a batch size of 64, EfficientNetv2b2 achieves an optimal loss of 0.64 on Kaggle [3] submission, outperforming models trained with batch sizes of 128 and 32, which yield losses of 0.66 and 0.82, respectively.

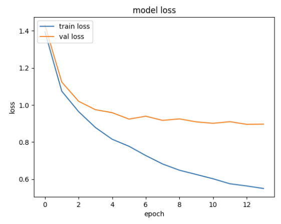
After determining that a batch size of 64 yields optimal performance for our specific problem statement, we conducted experiments to assess the impact of different kernel sizes. Our findings indicate that a kernel size of 3x3 proves most effective for our dataset (See Figure 8), which comprises images of EEG signals characterized by high-frequency data. These high frequencies are spatially proximate within the images. Utilizing kernel sizes larger than three presents potential drawbacks. While larger kernels capture more intricate details, they tend to overlook smaller, more crucial details, particularly relevant in our context. Moreover, larger kernel sizes introduce image blurring in subsequent layers due to the aggregation of information over broader areas, consequently obscuring finer details. Additionally, it is worth noting that larger kernel sizes require more computational power, while smaller kernels are generally more efficient in most cases. [4]



Kernel size = 3x3



Kernel size = 5x5



Kernel size = 7x7

Figure 8: Training Loss and Validation Loss for kernel size 3x3, 5x5 and 7x7 over 13 epochs from left to right for EfficientNet-v2b2

After learning the best configuration for our EfficientNetv2b2 backbone model with ImageNet [7] weights, we decided to increase the epochs to 50. We pick out the best performing model in those 50 epochs and use it to generate a prediction for submission to Kaggle.[3] Figure 9 shows the traning loss and validation loss of our best model over 50 epochs of training. Figure 10 is the final result of our best performing model. Our model achieved an optimal loss of 0.57.

5. Conclusion

In conclusion, our experimentation with EfficientNetv2b2 backbone, utilizing a batch size of 64 and kernel size of 3x3 trained over 50 epochs, has yielded good results, outperforming YOLOv8 and ResNet50 models. With

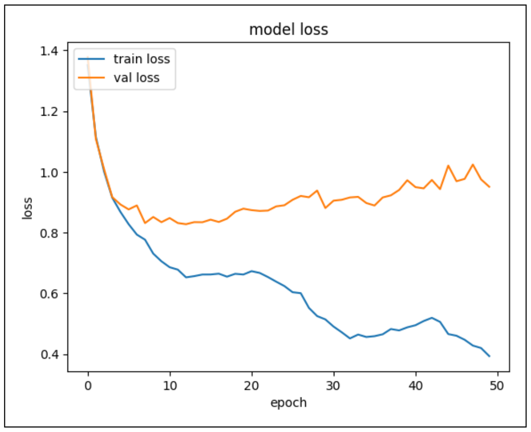
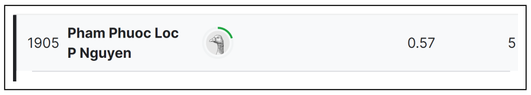


Figure 9: Training Loss and Validation Loss for batch size 64 over 50 epochs of best model



1905 Pham Phuoc Loc P Nguyen 0.57 5

Figure 10: Kaggle final result for best performing model

an optimal loss of 0.57, our EfficientNetv2b2 backbone model implementation achieved a notable rank of 1905 on Kaggle.[3] This underscores the effectiveness of EfficientNetv2b2 in image classification tasks, highlighting its superior performance and potential for further advancements in machine learning research and application.

6. Future Work

In future work, several improvement can be made to further enhance the performance and capabilities of our model. Firstly, investigating the impact of different augmentation strategies and regularization techniques could help improve generalization and robustness, potentially leading to even lower loss values and higher ranks on Kaggle. Additionally, fine-tuning hyperparameters such as learning rate schedules, optimizer choices, and model architecture variations may uncover optimal configurations that maximize performance. Lastly, incorporating 3 to 4 of the top performing models using Ensemble might significantly improve the prediction results and further decrease loss.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[2] J. Heaton. Applications of deep neural networks with keras, 2022.

[3] C. Y. A. C. S. D. J. S. M. B. W. Jin Jing, Zhen Lin. Hms - harmful brain activity classification, 2024.

[4] D. Li, L. Li, Z. Chen, and J. Li. Shift-convnets: Small convolutional kernel with large kernel effects, 2024.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection, 2016.

[6] D. Reis, J. Kupec, J. Hong, and A. Daoudi. Real-time flying object detection with yolov8, 2023.

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[8] J. Shlens. Notes on kullback-leibler divergence and likelihood, 2014.

[9] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[10] M. Tan and Q. V. Le. Efficientnetv2: Smaller models and faster training, 2021.