Algonauts Challenge

Ibrahim Muhip Tezcan, Andrei Klimenok, Pelin Kömürlüoğlu
{itezcan, aklimenok, pkoemuerlueo}@uos.de
IANNwTF
Winter Semester 2022/23, Universität Osnabrück
April 1, 2023

Abstract

This project explores and experiments with different models to investigate their encoding accuracy in the Algonauts Challenge 2023. We experiment with pretrained models as well as training our own model, and furthermore, fine-tuning them. Our exploratory results show that larger models and models with residual connections perform better, and our experimental results demonstrate that training a model with a larger dataset improves the performance.

Keywords: *NSD, encoding models, algonauts challenge*

1 Introduction

1.1 Project Goals

Our project goal for the course *Implementing Artificial Neural Networks with TensorFlow* is to train models and experiment with different settings for artificial neural networks (ANNs) and see how they perform for the Algonauts Challenge 2023, which aims to build encoding models of the human visual cortex. Here, the project goals and challenge goals differ from each other. The main goal of the project is not to achieve the highest score for the challenge leaderboard, but rather to see how different settings contribute to the submission score - the correlation score. Therefore, we experiment with different training scenarios for our own implementation of AlexNet, such as being randomly initialized, fine-tuned, or trained from scratch. The performances for each scenario are compared. Hence, the project goal serves as a learning opportunity. On the other hand, the challenge goal is to obtain a comparably good score, hopefully within the top 3 in the leaderboard. For this goal, we hold an exploratory approach.

1.2 Algonauts Challenge

The task for Algonauts Challenge 2023 is to create an encoding model that can predict the fMRI activation of the brain from the input image [2]. The dataset of choice for this challenge is the Natural Scenes Dataset (NSD).

1.3 Natural Scene Dataset

The BOLD responses from the fMRI activity of 8 subjects were recorded while viewing natural scene images from the Common Objects in Context (COCO) dataset [1]. The dataset offers extensive data for high-quality fMRI images and is very suitable to train neural networks.

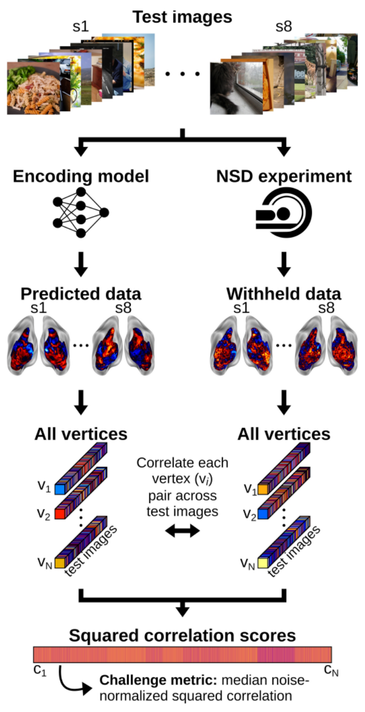

Test images
s1
s8
Encoding model
NSD experiment
Predicted data
Withheld data
s1
s8
s1
s8
All vertices
All vertices
v1
v2
vN
test images
Correlate each vertex (vi) pair across test images
Squared correlation scores
c1
cN
Challenge metric: median noise-normalized squared correlation

Figure 1: The visualization of the challenge. Taken from [2]

1.4 Evaluation Metric

The evaluation metric for the submission score in the Algonauts Challenge is calculated using the Pearson's correlation between the predicted data and the test data that is withheld from the challenge participants [2]. Therefore, the calculation of the results are only possible through submission. After the calculation of the correlation, each correlation score is normalized with noise-ceiling. The noise-ceiling suggests the variance of total prediction with the noise level in the data [2]. Finally, the median of the normalized squared correlations is used for the final submission score.

1.5 Encoding Models

Linear encoding models in computational neuroscience are used to find linear relations between sensory input and brain activation, thus predicting the brain activity of a subject when presented with certain stimuli. The encoding can be used in combination with decoding models that predict the stimuli from the brain activation, e.g. reconstruction of visual stimuli [9, 10]. In the challenge and this project, a computational model is used to extract features from images and these features are linearly mapped to the vertex responses, finally, the estimations for mapping are applied to generate fMRI activations [6]. To sum it up, we employ linear regressors to fit the extracted features from a computational model to the fMRI data.

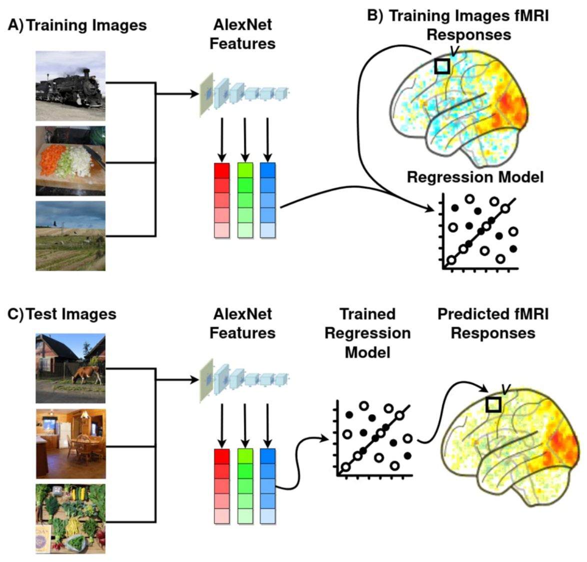


Figure 2: The linear encoding process. The features from images are extracted. These features are then mapped to corresponding fMRI responses and these estimations are applied to predict fMRI responses to test images. The figur is taken from [6] and the model is given as AlexNet since it is from the development kit.

2 Methods

2.1 Coding

The challenge provides a notebook [1] on Google Colab to get familiar with the data, encoding models, and submission as a part of the development kit [2]. The development kit uses PyTorch as the framework. We started by turning this notebook into Python modules and adapting it to Tensorflow. Since the initial code for the tutorial is written with PyTorch, our GitHub Repository also includes some scripts that work with PyTorch. However, for the purpose of this project, only the code with TensorFlow is going to be mentioned in further sections.

2.2 Repository

For the organization of our code, we have created a GitHub Repository[2]. Each folder of the repository holds the scripts for different functionalities.

Our GitHub Repository consists of Python modules, scripts and notebooks. The modules have various functions. The Data Processor modules process image datasets and load and partition the NSD dataset for training, validation, and test sets. The Encoders use ridge regression to predict fMRI data from given features and output the predictions as dictionaries for validation and test data for each hemisphere. The Evaluators investigate the performance of each ROI through all layers and print the best performance for each ROI with the information of which layer produces that score. These results are later written to a .json file and plotted. The Feature Extractors use incremental PCA to reduce the feature dimensions and vertically stack them in batches to avoid memory exhaustion. The Models load a model or implement the AlexNet architecure. Pipelines run the whole pipeline, given the parameters. Lastly, Utils include different utilities that are used throughout the pipeline such as writing and reading .json files. The notebooks have the pipelines and model creations as .ipnyb files. They import the functions from the modules, therefore they cannot be used without the modules.

2.3 Approach and Implementation

This section explains our approach. We introduce the features, models, datasets, and training methods of choice and reason the decisions.

2.3.1 Model Selections

AlexNet. With the implementation of optimized GPU usage, AlexNet revisits the convolutional networks and employs them for image classification tasks [5]. It remains to be a benchmark as a convolutional network for image tasks. The Algonauts Challenge uses a pretrained AlexNet as the baseline, therefore, we also use AlexNet for our experimentation. Aside from the initial trials with sourcing the model from Torch with and without ImageNet weights, we implement our own AlexNet model and train it on different datasets, such as COCO, Imagenette and CIFAR-100.

[1]`https://colab.research.google.com/drive/1bLJGP3bAo_hAOwZPHpiSHKlt97X9xsUw?usp=sharing`

[2]`https://github.com/iannwtf-algonauts/algonauts23`

VGG. VGG is a convolutional model that uses a small receptive field and has a very deep architecture [12]. Its significance is to achieve state-of-the-art performance by having weight layers up to 19. In this project, VGG-16 pretrained on ImageNet that is available in Keras is used.

EfficientNets. EfficientNets is a family of convolutional networks. It consists of models that are based on a mobile-sized model and scaled up using the compound scaling method where they can be scaled up in all three dimensions: depth, resolution, and width [13]. We have initially tested the performance of different members of the EfficientNets and decided to use EfficientNetB2 which is pretrained on ImageNet and available on Keras.

ResNet. ResNet tackles the degradation problem of the very deep convolutional models by introducing deep residual learning [3]. We are using ResNet-50 sourced from Keras with pretrained weights of ImageNet.

2.3.2 Training Methods

For the purpose of our project goal, we have various implementations of different models. Apart from the pretrained models that are outsourced from Torch Hub or Keras, we focused on training AlexNet from scratch with different datasets and fine tuning it.

Trained Models. The OrganizerBaseline is the score of an AlexNet trained on ImageNet. However, a pretrained version of AlexNet on ImageNet is not available on Keras but rather only on Torch Hub and since NSD consists of images from COCO [1], we chose to train AlexNet on COCO from scratch. Different from the data augmentation on ImageNet that was applied to train AlexNet [5], we follow another approach to handle the image data from COCO. The images are resized and normalized to the expectations depending on the size of the input layer. We also train AlexNet on Imagenette and CIFAR-100 since it was not computationally possible for us to train it on ImageNet due to the size of the dataset.

Additional to AlexNet, we have trained VGG-16 on COCO from scratch. Our aim with training another model is to compare these models. It is essential to train both models in the same conditions so that the comparison is valid. Similar to the data preparation to train VGG models [12], we have prepared the image data to have a resolution of 224x224.

Fine tuning. Another method to increase a model's performance and make it more precise for the chosen task is fine tuning. We use the fine tuning method from Keras [4]. It consists of the transfer learning method with freezing a pretrained model, adding layers and training those layers with the new dataset, and additionally fine tuning will unfreeze the whole model and train the model completely with the new dataset using a very small learning rate . We apply fine tuning on our AlexNet. We trained the model with Imagenette and fine tune it with COCO.

2.3.3 Statistical Techniques

Ridge regression. Ridge is a linear regression method that uses L2 regularization. With the large dimensions of data, ridge regression is preferred for prediction and estimation [7]. Hence, it becomes useful for our project to use ridge regression instead of basic linear regression due to the size of the dataset that is used in the challenge. We implement the ridge regression from the SciKit Learn[3].

Incremental Principal Component Analysis (Incremental PCA). PCA is a method to reduce the dimensionality of data and it is used to extract features from the data [11]. It is used with linear regression to create predictions from features. Incremental PCA takes batches of data to apply PCA in smaller increments rather than the whole data at once. This way the memory is not exhausted and the training can process smoothly. We use the function from SciKit Learn [4].

2.3.4 Layer Merging.

Another approach to producing better prediction results from our models is layer merging. With this method, we detect the best-performing layers of each ROI from the validation set and merge these best-performing ROI predictions together to produce the final predictions for the test set. For example, we do not pick only one layer, such as 'conv2d_5_pool' to generate predictions for all vertices. Instead, we get the predictions for 'V1v' from an early layer such as 'conv2d_2_pool', 'FFA' from a later layer such as 'batch_normalization_3', etc.

This improves results, because in most models (except models with residual connections) early layers of the model are better at predicting the early visual layers of the brain, such as V1, while later layers of the model are better at predicting the later layers of the brain, such

[3]`https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge`

[4]`https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.IncrementalPCA.html`

as FFA. By extracting features for each ROI separately, from their best-performing layers, we create a final prediction file that is more representative of the brain.

3 Results

Our results can be reported in two different categories. First, we have the results that aims to improve the challenge submission score. These models are pretrained and outsourced. They are used in an exploratory way to see the performance and ability to be tuned further. Second, we have our own trained models that serve for our project goal which is to pretrain and fine tune own models to observe their performances for experimental purposes.

3.1 Exploratory Results

In our exploratory submissions, we have tested AlexNet, VGG-16, ResNet-50, and EfficientNetB2. Our best results are obtained from EfficientNetB2.

The initial submissions were done with AlexNet which was sourced from Torch Hub, following the challenge tutorial. This test was to see whether we can reach the OrganizerBaseline score of 40.47%. The first submission with the AlexNet yielded a score of 10.34%. Later, we have adjusted the model with ridge regression, cross-validation, and used parameters from a grid search. This helped us to reach the score of 14.66%. Later, grid search and cross-validation are dropped, so they only appear here. With the further realizations, it occurred to us that we never gave the parameter to have the pretrained AlexNet when sourcing from the Torch Hub. Finally, we obtained 40.11% accuracy. This is a very close score to the OrganizerBaseline and this verifies the performance of the baseline model.

Next, we have tested VGG-16 pretrained on ImageNet. We sourced it from Keras. We wanted to be able to have a pretrained model to test using TensorFlow as AlexNet is not available on Keras. It achieved 40.46% accuracy directly, which is slightly above the OrganizerBaseline and the AlexNet.

As it was one of the available models on Keras, we selected ResNet-50 to see how a residual network performs in the challenge. ResNet-50 produced a score of 43.22%, slightly above of VGG-16.

As seen from the VGG-16, a very deep model can perform better than AlexNet. Therefore, we decided to use a scaled up model, and used EfficientNetB2 from EfficientNets. Our overall best result is from EfficientNetB2 with a score of 47.81%. EfficientNetB2 is also trained on ImageNet and sourced from Keras.

Model	Pretrained	Source	Score*
AlexNet	No	Torch	10.34
AlexNet	No	Torch	14.66
AlexNet	ImageNet	Torch	40.11
VGG-16	ImageNet	Keras	40.46
ResNet-50	ImageNet	Keras	43.22
EfficientNetB2	ImageNet	Keras	**47.81**

Table 1: The table of exploratory results for each model.
*scores are given in percentages

3.2 Experimental Results

Our experimental results contain the models we trained ourselves from scratch. The compiled models can be downloaded from MyShare folder[5]. We have implemented a model based on AlexNet architecture and first, we ran it on random weights. This model scored 15.12% accuracy and serves as the baseline for the further experiments with AlexNet. Without having to train our own AlexNet on the entire ImageNet dataset, we experimented with smaller datasets, CIFAR-100 and Imagenette, and additionally trained on COCO. Training the model on CIFAR-100 and Imagenette has improved the challenge score of the model, up to 21.39% and 22.00%, respectively. Next, we chose the better-performing model and applied transfer learning and fine tuning with COCO to the model trained with Imagenette. Fine tuning assisted the model to reach 26.78% accuracy.

Model	Pretrained	Fine tuned	Score*
AlexNet	CIFAR-100	No	21.39
AlexNet	Imagenette	No	22.00
AlexNet	COCO	No	**32.96**
AlexNet	Imagenette	COCO	26.78

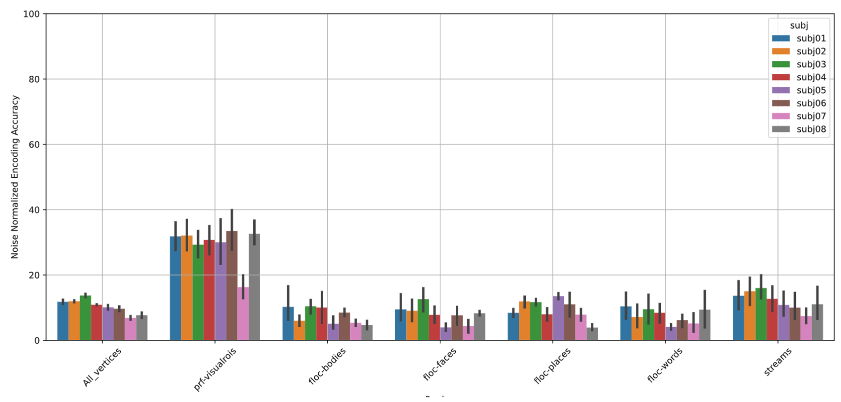
Table 2: The table of experimental results for each model.
*scores are given in percentages

However, training on COCO has resulted with the score of 32.96%, implying that a larger dataset increases the model performance.

4 Conclusion

In this project, we investigated different models and training methods to see the performances of neural networks in Algonauts Challenge 2023. We participated in

[5]`https://myshare.uni-osnabrueck.de/d/fbcb6c2079184184b3a8/`



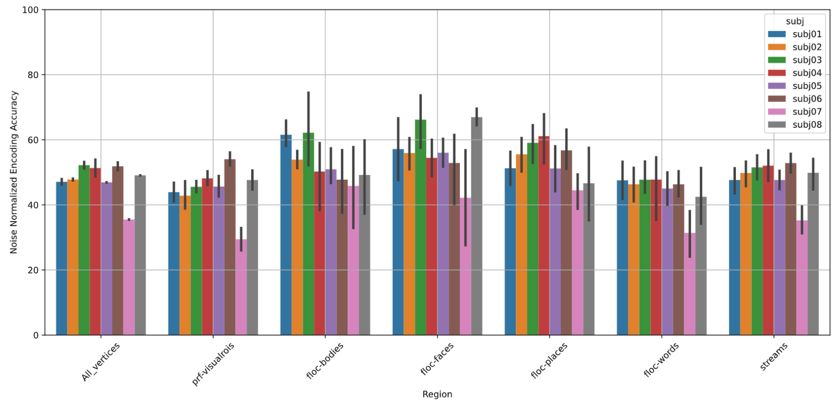


Figure 3: Top: The initials results of from the implementation of the baseline model which is an AlexNet with random weights. Bottom: The results of EfficientNetB2.
We can observe a good amount of increase from the baseline model in ROI with all subjects.

this challenge under the name "iannwtf", and on the day of the report submission, we hold 8th place with the challenge score of 47.81% obtained with EfficientNetB2 pretrained on ImageNet.

This project allowed us to train our own models with different datasets, experiment with pretrained models, and use methods outside of the course content such as merging the best layers, incremental PCA, ridge regression, transfer learning, and fine tuning.

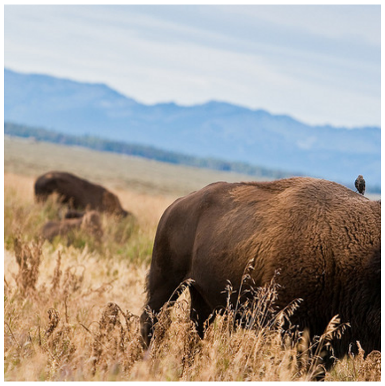
Our results show that our own AlexNet implementation without any training performs on par with, and even slightly better than, the untrained AlexNet from Torch Hub. However, we were not able to reach the performance of OrganizerBaseline or other pretrained models that we used. These models were trained on the entire ImageNet dataset, and the models except AlexNet are deeper and they rely on residual connections, which contributed to their performance [12, 13]. On the other hand, using a smaller dataset and fine tuning the model on another larger dataset is a promising and possible way of training models for object classification when the resources are not adequate to train on large datasets, especially if one is experimenting with different architectures and thus would benefit from getting results quickly.

Our intentions for the future of our participation in the Algonauts Challenge 2023 is to improve our work with the best layers of the models. Ma et al. [8] has implemented a mixed visual encoding model where they combine the extracted features from two different model and their layers to improve the accuracy for a voxel activity prediction task. Similarly, using our function to find the best layers, we can extract features from the best-performing layers of different models and apply linear regression mapping to this combination of features.

References

[1] Emily J. Allen et al. "A massive 7T fMRI dataset to bridge cognitive neuroscience and artificial intelligence". In: *Nature Neuroscience* 25.1 (Jan.

A



B

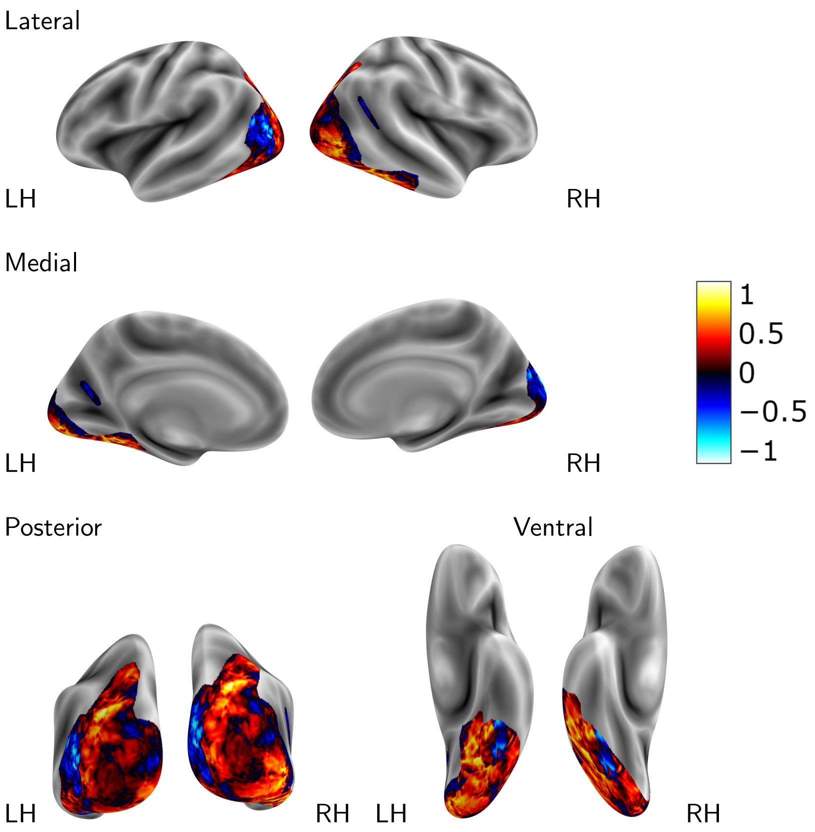


Figure 4: **(A)** An image from the test set for the Subject 3. **(B)** The predicted fMRI activation to the image in **(A)** produced from EfficientNetB2 in lateral, medial, posterior, and ventral views.

2022), pp. 116–126. ISSN: 1546-1726. DOI: 10.1038/s41593-021-00962-x.

[2] A. T. Gifford et al. "The Algonauts Project 2023 Challenge: How the Human Brain Makes Sense of Natural Scenes". In: (2023). DOI: 10.48550/ARXIV.2301.03198. URL: https://arxiv.org/abs/2301.03198.

[3] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[4] Keras. *Transfer Learning and Fine-tuning*. 2021. URL: https://keras.io/guides/transfer_learning/ (visited on 03/31/2023).

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[6] Alex Lascelles. *Encoding models*. URL: `http://algonauts.csail.mit.edu/encoding.html`.

[7] Sifan Liu and Edgar Dobriban. “Ridge regression: Structure, cross-validation, and sketching”. In: *arXiv preprint arXiv:1910.02373* (2019).

[8] Shuxiao Ma et al. “A Mixed Visual Encoding Model Based on the Larger-Scale Receptive Field for Human Brain Activity”. In: *Brain Sciences* 12.12 (2022), p. 1633.

[9] Thomas Naselaris et al. “Bayesian Reconstruction of Natural Images from Human Brain Activity”. en. In: *Neuron* 63.6 (Sept. 2009), pp. 902–915. ISSN: 08966273. DOI: `10.1016/j.neuron.2009.09.006`.

[10] Thomas Naselaris et al. “Encoding and decoding in fMRI”. In: *Neuroimage* 56.2 (2011), pp. 400–410.

[11] Jonathon Shlens. “A tutorial on principal component analysis”. In: *arXiv preprint arXiv:1404.1100* (2014).

[12] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[13] Mingxing Tan and Quoc Le. “Efficientnet: Rethinking model scaling for convolutional neural networks”. In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114.