Functional Approximation of V4 Neural Responses

Eli Pugh
epugh@stanford.edu

Abstract

In this paper I examine state-of-the-art models used for approximating neural responses in V4 given different stimuli. I then design a model that outperforms the current state-of-the-art models on the Pasupathy ShapeLAB dataset for this task. This model collates many successful techniques used in other successful models. In addition, I analyze these methods and provide insight to why the combination of these methods forms a more performant model.

1 Introduction

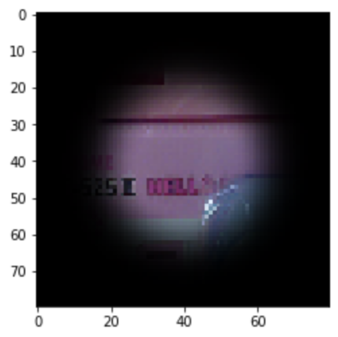
One of the greatest challenges in neuroscience today is understanding how the brain processes and understands stimuli. A very important aspect of this understanding is how the brain encodes sensory input into latent space. One way we can analyze this ability to represent items in latent space is to model this process and analyze different methods to see if we can create networks that perform this transformation in a similar fashion. Not only would an accurate and robust model give great insight into neural encodings, but it would also be of value as a way to quickly construct stimuli that will fire chosen neurons in order to further study structure in the brain.

One such way to approach this problem is by measuring neural spikes as a response to stimuli. There have been many research efforts towards this goal focusing in different areas of the brain. For this paper, I will focus on the visual cortex V4. The reason for focusing on this area is that its central point in the ventral stream means that its neuronal firings are past the early stages that are thought to pick up details such as edges, but before the later stages where neural embeddings are thought to represent more broad features. Studying neural spike trains in the V4 is a great place to start for researchers looking for representation of visual stimuli in latent space. [8]

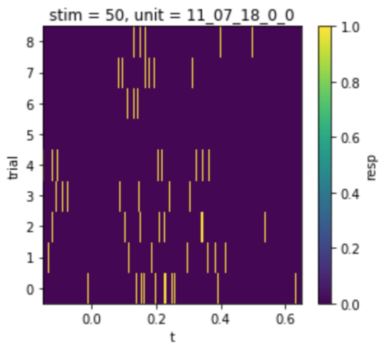
2 Data

There are many datasets used for neural response approximation, and the one used for this research is from the Pasupathy ShapeLAB in the Department of Biological Structure at the University of Washington. The Pasupathy lab is an electrophysiology lab that studies how visual input is processed. Their main focus is on processing in the ventral stream with specific attention to the V4 visual cortex. This V4 located in the center of the ventral stream, where it's thought that representations transform from more simple features like edges to more general patterns further downstream. [8]

Pasupathy lab collected this data by placing an electrode into the V4 of a macaque. They then showed the macaque up to 601 images and recorded the neural spikes through the electrode during this process. Each image was shown 3-20 times for 300 milliseconds each time, with a rest of 250 milliseconds in between images. The dataset used gives the average square root of the number of spikes between 50 and 350 milliseconds after the image first appeared to account for latency. The square root is taken to lower variance. Each image is randomly drawn from the 2012 ILSVRC validation set of images. The images are 80x80 pixels, and are processed to have a soft circular alpha Gaussian blur applied to the edges with a standard deviation of 16 pixels. [4]



Stimulus



Spike Trains

There were many reasons for choosing this dataset to work with for these experiments. Often times datasets such as this are not made publicly available, yet the Pasupathy lab released this data. In addition, they posted a competition on Kaggle.com with awards and monetary prizes, drawing interest from many field experts who were able to compete to build the best model. This means that there is a base of work from the computational neuroscience community to build off of, as well as a wealth of baselines we can use to compare performance.

2.1 Metrics

For this task I felt that it was most appropriate to stick to the standard metric introduced during the challenge hosted on Kaggle, which was correlation of the square root of predicted responses to the square root of observed responses. This is measured by root mean squared error (RMSE). Taking the root of the spikes is useful to reduce variance, and this is a natural metric in many regression tasks. In addition, using the most widely used metric allows direct comparison to other models.

3 Baseline Models

We use multiple baselines to benchmark our results, and describe them all here in detail. This allows us to see the performance of many different models used on this task, and compare these results later in the discussion section. In addition, we let this section replace a typical 'Previous Work' section, as it shows the most successful models at this task and explores popular methods in response estimation.

3.1 RGB Regression

The first baseline is a simple model that simply regresses on the average color on the image. This model is formed by calculating the total amount of red, green, and blue in an image, and then fitting a simple linear combination of these three elements to get a good estimate of the response activity. This is done quickly and easily with a closed-form calculation.

The first step is calculating the average value of the red, green, and blue channels of all images in the training set. Let the $n \times 3$ matrix A have entry a_{ij} corresponding to the i^{th} stimulus and the j^{th} color channel, and let y be the n-dimensional vector with entry y_i the true label of stimulus i. From this we form the optimization problem

$$\text{minimize} \quad |Ax - y| \qquad \text{with solution} \qquad x = (A^T A)^{-1} A^T y$$

where $(A^T A)^{-1} A^T$ is the Moore-Penrose inverse or pseudoinverse. This for an example a_i, the predicted root number of spikes is then calculated by $a_i \cdot x$.

Note that this is simplified for clarity to show the method on predicting one neuron's root number of firings. This is done done for each neuron that is needed. For this task, x has dimension $n \times 3 \times 18$ and y has dimension $n \times 18$.

This simple solution does well for a fairly naive method with a score of 0.809, placing 24th in the Kaggle competition. We will see more context in later baseline models, but the best performing models have an error in the range of 0.71 to 0.75.

3.2 AlexNet, Clever Features, and Extra-Trees[1]

The second baseline is a model created by Pierre Karashchuk and Tony Bigelow that placed 3rd in the Kaggle competition. Their approach first begins by computing multiple clever features of the data. They keep raw RGB values, but also convert the pixels to LAB color space. LAB color space is lightness l from 0 to 100, a from green (-) to b (+), and b from blue (-) to yellow (+). [3]

They also use the power spectrum of the 2-dimensional Fourier Transform, also sometimes called the power spectral density (PSD) of an image. The 2-dimensional Discrete Fourier Transform of an $N \times N$ image is given by

$$F(k,l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j)\, e^{-\tau 2\pi (ki+lj)/N}$$

where $f(i,j)$ is the image in the spatial domain, and τ is the basis function.

In addition, Karashchuk and Bigelow use a discrete wavelet transformation of the image using a Gabor filter. For brevity, the wavelet decomposition algorithm is not included, but (Shahbahrami, 2012) gives a thorough look at 2-dimensional discrete wavelet transform algorithms. [6]

the final features used were simple statistics of the RGB and LAB images, such as the mean and standard deviation of pixel values for each channel.

After featurizing, they also fed each RGB image through AlexNet and cached the response of each of the 5 convolutional layers. Figure 1 below shows the AlexNet architecture. It is a deep convolutional network with 5 channels that are combined with dense layers to a 1000-dimensional final representation. [2]

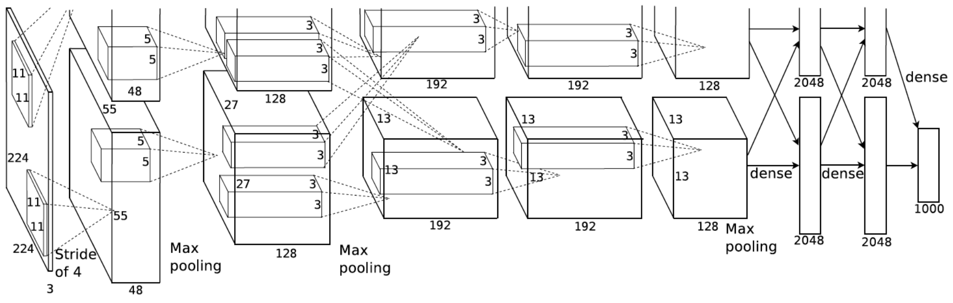


Figure 1: AlexNet

[1]This model and code is available at https://github.com/lambdaloop/v4-challenge

After computing these many different feature representations, Karashchuk and Bigelow use Principal Component Analysis (PCA) on the features that have large embedding spaces. This ensures that their model won't overfit the data, while still preserving important information in the image.

Finally, this representation is fed through an extra-trees regressor, which fits multiple decision trees on sub-samples of the data before and then averaging them together. [1]

This approach placed Karashchuk and Bigelow 3rd on the Kaggle leaderboard with a score of 0.736.

3.3 CNN and Data Augmentation[2]

The winner of the Kaggle competition, Oleg Polosin, decided to use a deep convolutional network, yet he cited overfitting as one of the biggest obstacles. His model architecture is shown below in Figure 2.

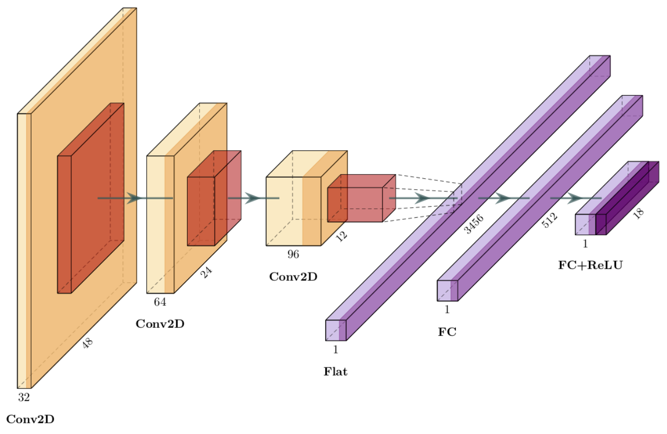


Figure 2: Polosin's model

As the figure shows, Polosin used 3 convolutional layers fed into 3 fully connected layers. This added complexity with a small amount of data is likely to overfit, and to solve this problem he used data augmentation and regularization.

Because of the limited data given in this challenge, Polosin used data augmentation to create new examples that were slight modifications of the original examples. He found that the distortions that helped the model generalize most efficiently were saturation and rotations by small angles. In addition to these, he also tried vertical and horizontal flips, but these seemed too distant from the original images to be good training data. For his augmentation he used rotations between -0.1 and 0.1 radians, as well as saturation multiplications from 0.7 to 1.3.

In addition to data augmentation, Polosin used regularization to keep his model from overfitting. After experimenting with L2 regularizations, batch normalization, and dropout, he concluded that he achieved the best results by only regularizing the dense layers with dropout.

4 Approach

4.1 Model Architecture

My first design decision was on a general model architecture. I decided to pursue a deep convolutional network similar to Polosin's model over an extra-trees regressor. I chose to use a CNN

[2]This model and code is available at https://github.com/apls777/kaggle-uw-neural-data-challenge

because it preserves spatial structure in the image much more than Karashchuk and Bigelow's model. By computing so many features, they were forced to perform PCA on many of their embeddings, and weren't able to really preserve positional relation between pixels in the image in the same way that a convolution does. Seeing how convolutional networks achieve state-of-the-art results on many computer vision tasks, I thought that this model not only made sense intuitively, but also because of it's empirical success.

4.2 Data Augmentation

One potential problem with using a deep convolutional architecture is that it's fairly complex given the small amount of training data, and bound to overfit if not carefully trained. To overcome this problem, I performed data augmentation, again similarly to Polosin's solution. I found that after experimenting with hyperparameters, I agreed that his use of slight rotations helped reduce overfitting, yet vertical or horizontal flips and large rotations severely weakened the link between stimuli and neural responses.

4.3 Featurization

Inspired by the featurization from Karashchuk and Bigelow's model, I decided that my network should not only take advantage of the spatial structure of the images, but also incorporate features that the convolutions might have a harder time picking out. In order to achieve this, I concatenated three additional featurization techniques to the original image.

My first processing step was to convert the pixels to LAB color space, as Karashchuk and Bigelow did in their own preprocessing. This preserves dimensionality and structure as desired, and it also represents the image in a way that more closely mirrors human perception. Unlike RGB and CMYK color spaces, LAB was designed to match human perception with respect to percieved similarity. This is true with respect to many metrics, most notably the squared error

$$\begin{aligned} |C_1 - C_2| &= \sqrt{(l_1 - l_2)^2 + (a_1 - a_2)^2 + (b_1 - b_2)^2} \quad \text{(LAB)} \\ |C_1 - C_2| &= \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2} \quad \text{(RGB).} \end{aligned}$$

Since LAB color matches human perception of similarity and distance of colors than RGB, this can help the model better predict human neural response patterns to the images as well.

The second preprocessing step was to convert to HSB color space. HSB stands for hue, saturation, and brightness. This color space also aligns much better with human perception than RGB and CMYK, but is not quite as closely aligned as LAB is. The advantages to HSB are that it is quickly and easily computed from the original RGB color space, yet it still represents the image in a different, informative way. [3]

The last preprocessing step was to add an edge detection layer using the Sobel operator. [7] The Sobel operator is a way of approximating gradients by convolution over an image. For a source image I, the we can approximate the magnitude of the gradient G using the Sobel operator:

$$G = \sqrt{G_x^2 + G_y^2}, \qquad G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I, \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I$$

By quickly estimating the magnitude of the gradient, we see where the image changes colors and brightness quickly, giving highlights of where there are edges in the photo. Knowing where edges are in the photo is highly useful to convolutional networks, and it can take several kernels and layers to learn to detect edges effectively. By including this information in the input, it's possible increase the efficiency of the network considerably.

After the 3 featurized images are computed, I stack them with the original image. This means that the input to the convolutional network is still a 80×80 pixel image where each pixel now has 12 channels rather than the original 3.

4.4 Implementation Details

My implementation details are fairly standard. I began by choosing most hyperparameters to be the same that Polosin used, and then tweaked them as needed to see if I could gain performance. I found that he probably performed cross validation since I didn't end up seeing performance benefits to changing almost all parameters. I stuck with his convolution architecture of 5×5 kernels at each level, and increasing the number of channels from 32 to 64 to 128. The structure of this model is the same as Polosin's, as shown in Figure 2. Given the complexity of the model and my added features, I predicted that without precautions the model would overfit the data. To remedy this, I adjusted regularization to be a bit higher in the fully-connected layers. In addition, I raised the dropout rate to 0.45 in the fully connected layers.

Even though I increased the number of input features to the model by a factor of 4, I only saw a modest amount of overfitting, which didn't seem to hurt overall performance too badly. In addition, with more regularization there was less separation between the training and development accuracy, but the overall accuracy was lower, so I decided to allow the model to overfit some.

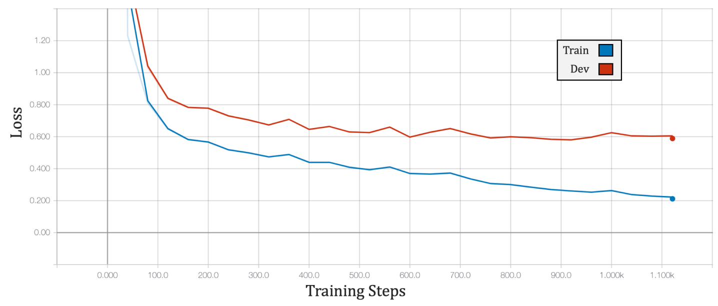


Figure 3: Training Loss

In addition, notice that the model training loss and root mean squared error (RMSE) follow nearly the same trajectory with only scaling differences. This is what one would hope to see, as the loss function is the same as our evaluation function, except for that the loss function includes a touch of regularization.

Training curves are shown below. My model trained for roughly 1,100 steps, or 60 minutes on a standard Macbook Pro laptop with a quad-core Intel i7 at 2.2 GHz.

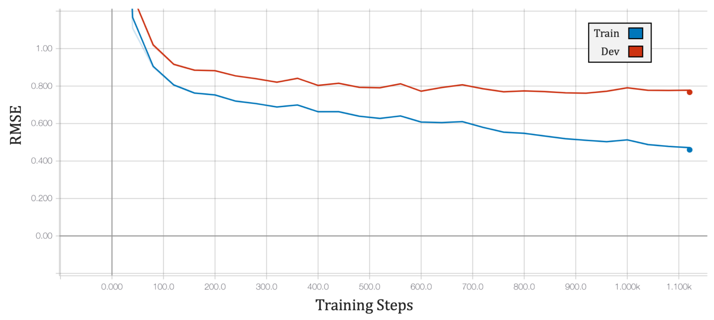


Figure 4: RMSE

My implementation of this model will be available in the near future at `github.com/elipugh`.

5 Results

While private test sets are not released to evaluate my model on, I have evaluated using the public portion of the dataset by holding out an evaluation set that was not used during training. I fully acknowledge that these results may not fully generalize to the private test set, but getting results on the development set is a fairly good proxy for how the model will perform on another unseen set.

In order to ensure that I got a truly good comparison to other models using the development set, I trained both Polosin's model as well as Karashchuk & Bigelow's model locally to confirm their results.

Root Mean Squared Error Comparison

	Dev Set	*Private Test Set*
RGB Regression	0.921	0.809
Karashchuk & Bigelow	0.754	0.733
Polosin	0.751	0.715
Pugh	**0.742**	–

6 Discussion

It's unsurprising that I was able to achieve state-of-the-art results on this task given that other top finishers released their models. This allowed me to base most design decisions off of their work, and use many of their insights to boost my performance.

In addition, these experiments and comparisons show a few clear trends that can lend insights into the brain's mapping into V4.

My first observation is that the models that retained the most amount of spatial structure were at a great advantage. This seems very intuitive, since images only make sense to humans because of their arrangement. An image with the pixels flattened into a line is not interpretable by humans, so it is reasonable that the neuronal firings would be more accurately predicted with structured input.

Another thing that seemed to help models was featurization that made them more representative of the way humans perceive images. Featurization techniques like changing color space or identifying edges are really insightful techniques. These show that if we want to predict neural spikes, we should interpret the data as similarly to humans (or macaques) as possible.

7 Future Work

It seems that this area of neuroscience is particularly primed for advancements given the fast-paced technological advancements of electrode arrays to measure cortical signals as well as rapid development in the field of machine learning. There are many directions that I can see being particularly useful for researchers to explore in the near future.

One really interesting direction that could use more exploring is the use of this kind of data for stimulus reconstruction. Recently Akbari et al. used motor cortex neuronal data to generate synthetic speech. [5] It would be really interesting to see this same goal for images. This might also identify different areas and ways of storing visual information in the brain. For example, reconstructing an image after flashing a subject a picture seems to be a much different task than trying to reconstruct just by asking the subject to think about a subject in the image.

Lastly, on a less technical note, it would be very advantageous to encourage interdisciplinary research from neuroscientists and statistical learning experts. Venues like Kaggle allow researchers from around the world to collaborate and learn and combine ideas in a way that can really accelerate the cutting-edge of research in many fields. I hope to see more publicly-available, high-quality neural data in the near future, so that more students and researchers in other areas can contribute to the field.

References

[1] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, Apr 2006.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[3] Wojciech Mokrzycki and Maciej Tatol. Perceptual difference in l * a * b * color space as the base for object colour identfication. 08 2009.

[4] Timothy D. Oleskiw, Amy L. Nowack, and Anitha Pasupathy. Joint coding of shape and blur in area v4. In *Nature Communications*, 2017.

[5] Brian N. Pasley, Stephen V. David, Nima Mesgarani, Adeen Flinker, Shihab A. Shamma, Nathan E. Crone, Robert T. Knight, and Edward F. Chang. Reconstructing speech from human auditory cortex. *PLOS Biology*, 10(1):1–13, 01 2012.

[6] Asadollah Shahbahrami. Algorithms and architectures for 2d discrete wavelet transform. *The Journal of Supercomputing*, 62(2):1045–1064, Nov 2012.

[7] Irwin Sobel. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 2014.

[8] Leslie G. Ungerleider, Thelma W. Galkin, Robert Desimone, and Ricardo Gattass. Cortical Connections of Area V4 in the Macaque. *Cerebral Cortex*, 18(3):477–499, 06 2007.