# SKIL IV - Visual Decoding with EEG

Isaac Van Benthuysen, Jack Caputo,

Physics Department, Stevens Institute of Technology

## 1 Introduction

### 1.1 *Electroencephalography*

An electroencephalogram (EEG) is a recording of the electrical activity in a person's brain. When neurons fire, they generate action potentials: a rapid depolarization and repolarization of the cell that creates a millivolt pulse. While the firing of an individual neuron is difficult to detect, the aggregate voltage from many neurons can be detected by electrodes placed on the scalp. A major limitation of EEG is the low signal to noise ratio, making the signals difficult for humans to interpret. Recent developments in machine learning have allowed EEG to be used for neural decoding, the practice of determining what is happening in a subject's brain based on their brain activity.

### 1.2 *Visual Decoding*

Visual Decoding entails determining what a person is looking at based on their brain activity. In recent years, various groups have achieved success in decoding and even reconstructing visuals, but the majority of these experiments have relied on functional magnetic resonance imaging (fMRI), which is expensive, immobile, requires a relatively time consuming setup, and has low temporal resolution. EEG, on the other hand, is significantly cheaper, portable, easy to set up, and has high temporal resolution. While EEG lacks spatial resolution, the pattern recognition capabilities of neural networks enables them to make use of this data for visual decoding. In the past few years, a number of groups have been successful in reconstructing visual imagery by first using a convolutional neural network decode, then a diffusion model to reconstruct. The first step in complete visual reconstruction is decoding what exactly a person is visualizing, and our intent is to achieve successful visual decoding by use of a convolutional neural network.

### 1.3 *Machine Learning*

Machine Learning (ML) is a general term which refers to the use of statistical techniques to analyze data in such a way that some algorithm can learn from patterns in that data, with the goal of having the algorithm make predictions on new data based on the patterns it has discovered. Neural Networks (NNs) are a core subject in Machine Learning. Based on how real neurons work, NNs are a way of connecting some input to some output, with a number of in-between steps. As a quick example, imagine a visual stimulus: light is focused through the eye and hits the retina (input), that information is taken to the brain and then processed in some complex fashion before the person is able to identify what they just saw (output). An abstract diagram of this general process is given in Figure 1, where we call the circles "nodes" and the lines "weights". Deep learning is a subset of ML and simply uses NNs with multiple layers to accomplish the task of ML. The most popular type of NN is the Convolution Neural Network (CNN), its name arising from the use of convolutions. As our project used the CNN approach, we will break that specific architecture down in detail. There are

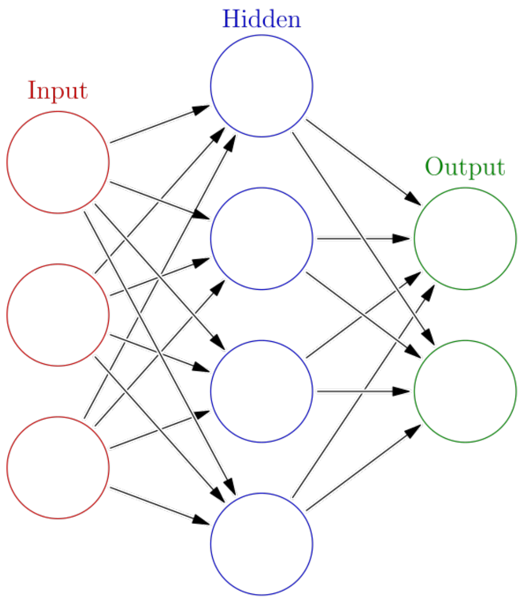

**Figure 1.** Abstract diagram of the general structure of NNs.

three main components in a CNN, and each is handled by a different layer: feature extraction, handled by convolutions; dimension compression, handled by pooling; and conversion to final output, mostly handled by the fully connected/dense layers. The convolutional layers actually create the patterns that the model learns from by extracting "features" of its input. This makes it the most important component of a CNN, and why it was given its name. The pooling layers simply decrease the size of the input while retaining the information. The dense layers contort its input so that we can more easily control the weights. All of this is quite theoretical, so it will be useful to see an example of a simple CNN.

### 1.4 *CNN Example*

Take the most simple CNN imaginable, Figure 2, comprised of the input, then a convolution layer, then a pooling layer, then a dense layer. We want to make a model to classify cats and dogs, and the input we consider is a 30x30 pixel picture of a cat. The image is in black and white, so the values of each pixel will be between 0 and 255, defining a grayscale. First, we do feature extraction on the input picture. What this actually looks like is a small grid called a "kernel," or "filter" - say, 3x3 - moving across the entire image. This kernel has real numbers in its pixels, and the operation is that we sum the products of each overlapping pixel (shown in Figure 3). This incorporates the convolution in its graphical sense which is easy to visualize; namely, "one thing sliding over the other." The convolutional layer is created by the outputs of different kernels over the input. Each slice is the result of a different

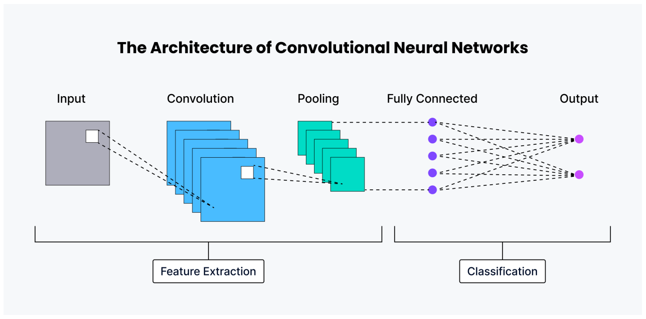

**Figure 2.** Diagram of a simple CNN.

kernel acting on the input: detecting horizontal or vertical edges, sharpening or blurring the image, translating certain pixels, etc. Let us use 16 unique kernels in our example. These features define the model's ability to characterize the input. In a realistic model, there are multiple convolutional layers, so the preceding convolutional layers are combined by using more kernels to extract increasingly complex features (e.g. vertical and horizontal edge detections combining to detect corners) until us humans no longer understand what features are being extracted. Some kernels may be useless - their output produces no new information. But we do an exhaustive search of as many kernels as is computationally reasonable because, as mentioned, there may be features which we humans cannot detect but which are essential for the model to work. Anyways, we now have our convolution layer. Note also that both dimensions of our image have decreased by 2, resulting in 16 28x28 outputs; this would be higher if we used a larger kernel. The pooling layer takes some sort of average to decrease

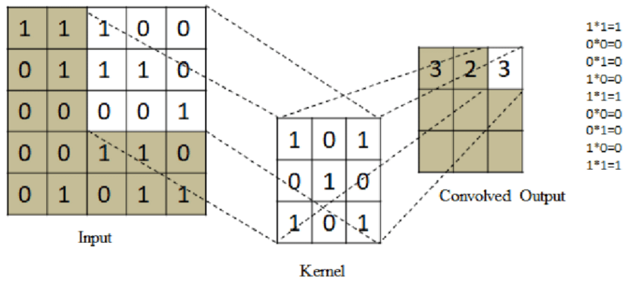

**Figure 3.** Visualization of the action of a kernel. Note the math to the side: we sum the product of overlapping pixels.

the dimensions while preserving the information. We might use a 2x2 MaxPool with strides of 2: MaxPool keeps the maximum value in the 2x2, and strides tells us how many pixels the pooling kernel must move before doing another iteration. Having the stride equal the dimensions means that there will be no overlap in the pooling kernels. Each 2x2 pooling kernel outputs into 1 pixel, so we are left with 16 14x14 outputs - there will be as many outputs here as convolution kernels, so the more features we extract, the pool layer will be correspondingly larger. Next, we "unwrap," "unroll," or "flatten" the 16 14x14 outputs. There is information present, but it is a bit difficult to extract, so we flatten our 16x14x14 elements from the pooled output to a 3136-dimensional vector. Now, we connect that vector to the final outputs, or "classification layer." As our model classifies cats and dogs, the final output will be a 2-dimensional vector - one dimension for each option. The information in the 3136-vector is extremely coupled - through the convolutions, information from the original image has become mixed and combined, so although the 3136-vector contains no discernable pattern to a human, it is still extremely sensitive to the initial image input. Perhaps the simplest solution to ensure we consider all information in the dense layer is to attach weights from each element of the 3136-vector to the 2-vector, and that is exactly what the "dense" or "fully connected" layer does. Finally, the magic of the CNN occurs in the "backpropagation." In this step, each weight is adjusted according to some statistical rule which minimizes the "loss" of the model, or equivalently, maximizes its "correctness" - given cat, the model should predict "cat." This is when the model is actually trained, a process in which there is no shortage of "hyperparameters" to choose from.

### 1.5 *Additional CNN Features*

There are many features to add to a model, but we will only cover here what we attempted. Instead of MaxPool, we could have something like AvgPool, which returns the average of the 2x2 pixels instead of returning the maximum. There are others still, but all use some mathematical rule to decrease the size of the data. Notably, the purpose of this is solely to rearrange the computation more efficiently. Instead of using 8 convolution kernels on a 30x30 image, we can use 32 kernels on a 15x15 image, ultimately giving more information with equal compute. "Dropout" is the practice of excluding certain nodes during the training phase, and the nodes in question are usually in the flattened layer, but can also be in other layers. We restrict some of the information the model uses to make its predictions in the hopes that it will become more robust. The analogy here is to cover a quarter of a cat's face: humans can still determine it is a cat, even with significant amounts of data missing. Though we have been using images as input, there is nothing unique about them - researchers have used CNNs on frequency signals to classify sounds, or on EEG signals in the time domain to classify what those signals represent.

## 2 Methods

### 2.1 *Software*

Our CNN was created using TensorFlow in Python. TensorFlow is one of the most popular software libraries for creating NNs, and as such it supports a wide range of ML-related tasks. We utilized the Keras API, which implements TensorFlow at a higher level and made the coding much more feasible for beginners like us. Keras has extensive documentation which was often referenced when creating the code for our CNN. Also of great importance was the Keras Tuner. Implemented within Keras, it is a library which automates the exhaustive process of determining the best hyperparameters for a model. Briefly, during backpropagation, there are many "hyperparameters" that must be chosen which affect the efficacy of the model (activation functions, loss functions, learning rates, etc.). The ideal set of hyperparameters maximizes the model's ability to predict the correct classification, but there seems to be no simple way of determining the ideal set. Therefore, the standard solution is to do an exhaustive search, which is greatly aided by Keras Tuner.

### 2.2 *Data*

To train our model, we used the publicly available dataset THINGS EEG2 (Gifford et al., 2022). The data was obtained by presenting a series of images to subjects and using EEG to record the activity in the occipital and parietal cortices. The data consists of 1654 image classes of 10 images each, with 4 repetitions performed for each of the 10 subjects. 17 channels were recorded for 100 time points. For the training, validation, and testing split, one repetition from each image was sent to either the validation or test set, for an overall 75% training, 15% validation, and 10% testing split. We limited the training to 20 classes, so our test set contained 6000 recordings, validation contained 1200, and test contained 800. We chose 20 specific classes based on 20 classes chosen by another group that we intend to compare our results with. We used the preprocessed data from THINGS EEG2. Additionally, we normalized data and randomized the order of samples prior to training and testing.

### 2.3 *Model Structure*

Our model consists of two blocks of a 2D convolution layer, layer normalization layer, 2D max pooling layer, and 2D spatial dropout layer; then a flattening layer, followed by two dense layers. The structure is visualized and better detailed in Figure 4. Convolutions and pooling have been described previously. Layer normalization simply re-normalizes the data after the convolution on a sample-by-sample basis (rather than as a whole batch). Spatial dropout differs from regular dropout in that it drops entire 2D feature maps, rather than individual elements, helping to promote independence between feature maps. We used the Adam optimizer and the Sparse Categorical Crossentropy loss function.

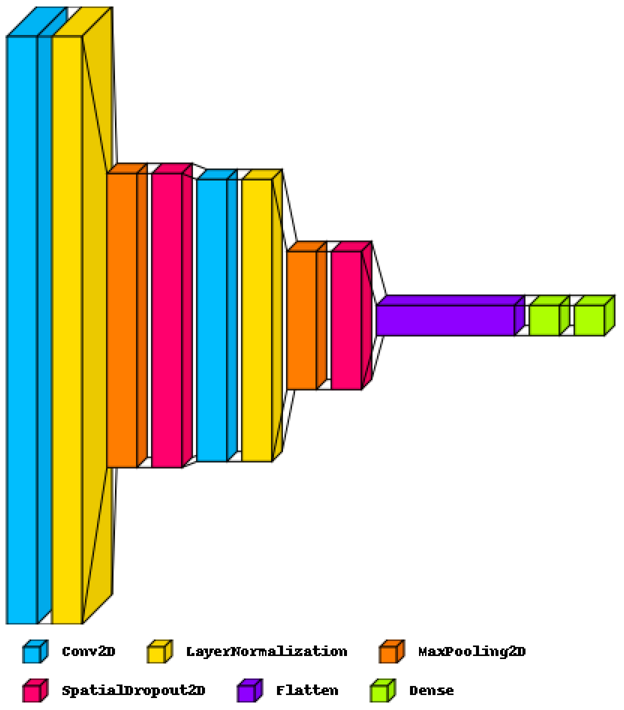

**Figure 4.** Block diagram abstraction of our CNN model.

## 3 Results

Our model achieved an overall test accuracy of 31.62% across all subjects and classes. The model performed best on the classes of man (65%), daisy (42.5%), and tennis_ball (42.5%). The model performed worst on school_bus (10%), dog (15%), and fish (17.5%). The most notably confused classes were panda (labeled as tennis_ball 25% of the time) and cherry (also confused with tennis_ball 25% of the time). This model appears to have a preference for classifying things as tennis balls, shown in Figure 5. Across the various models we trained and tested, we found that certain models seem to have some type of preference. Our earlier models seemed to like pizza and tigers. Accuracy and loss curves from training are

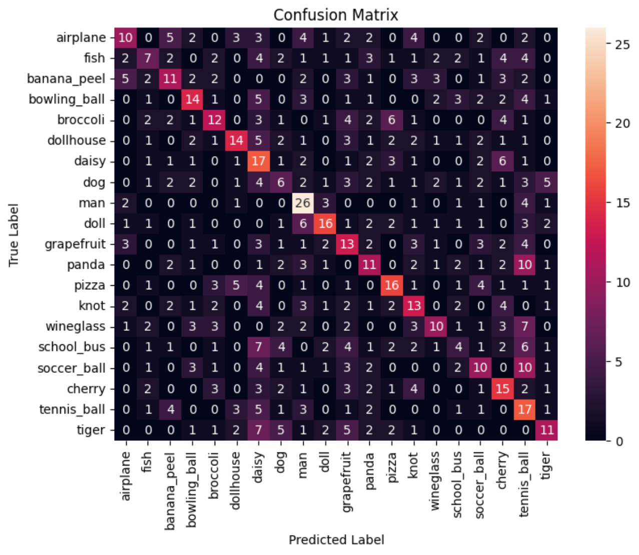

**Figure 5.** Predicted classifications vs. true classifications. A perfectly diagonal matrix is ideal.

shown in Figure 6 and Figure 7. Ideally, accuracy curves converge to 100% and loss to 0%. As can be seen, validation accuracy begins to plateau, and validation loss begins to diverge from loss. These are indicators of overfitting, when the model essentially "memorizes" training data and fails to generalize to new samples, so we kept the number of epochs low and implemented early stopping in order to prevent this.

## 4 Discussion

Our comparison to average total accuracy across all subjects and classes is shown in 1. Ours significantly outperforms many of the other CNNs, except for EEGNet, which beats ours by 2.78

Although we tested many hyperparameter combinations, there is no guarantee that the most effective combination we discovered is the most effective combination that exists. However, the more pressing issue, we believe, deals with the underlying philosophy of using hyperparameter tuning. In our testing, changing a couple of hyperparameters could alter the accuracy of our model by several percent, which was significant. We find it unsettling, frankly, that the effectiveness of a model can be so sensitive to the choice of hyperparameters. That being said, a proper analysis must be made for each unique model: perhaps a certain activation function gives the model a higher accuracy when the inputs are square rather

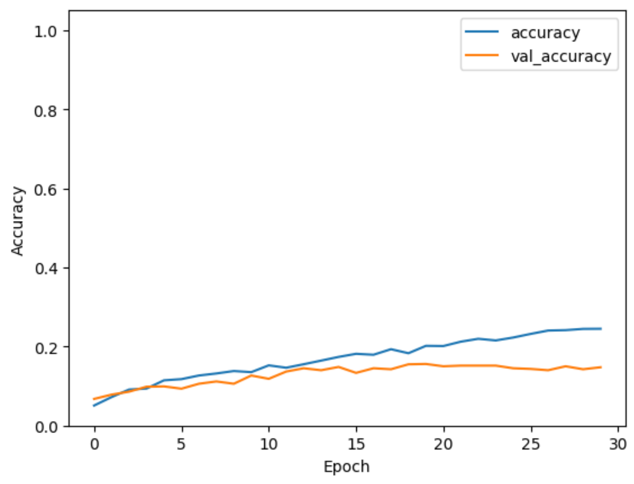

**Figure 6.** Accuracy of our model.

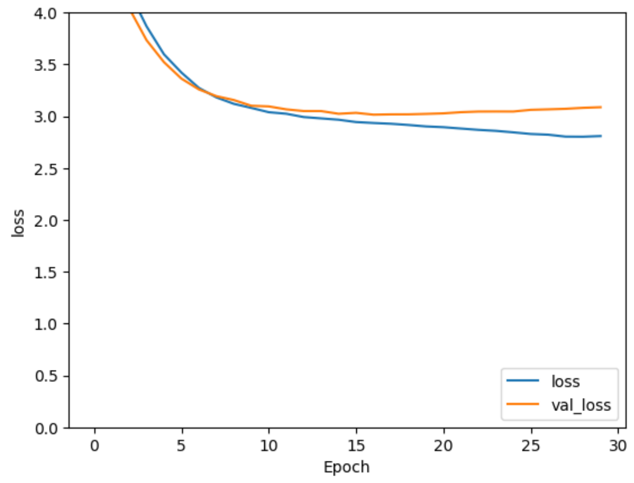

**Figure 7.** Loss of our model. The test and validation accuracy lines should stay close; divergence implies overfitting of model.

**Table 1**
Comparison of accuracies of models (figures from Guenther et al., 2024).

| Model | Accuracy (percent) |
|---|---|
| EEGNet | 34.4 |
| **Ours** | **31.62** |
| TSception | 22.6 |
| EEG Conformer | 21.2 |
| EEG ChannelNet | 15.6 |
| EEG-to-Image | 6.4 |
| Random guessing | 5.0 |

than rectangular, or with few final outputs rather than many. We did not do such an analysis. Nonetheless, no resource we discovered sufficiently explained how to do such an analysis, nor questioned why models are so sensitive to the choice of hyperparameters; they simply reinforced that it was necessary. We believe that CNNs are not the proper way to classify EEG data. This is foremost evidenced by the fact that our highest accuracy value was 30%. Although this is still 6x better than random guessing, there are NNs and indeed CNNs used in professions with accuracies >95%. Furthermore, the training method for CNNs – supervised learning – is simply not agreeable with EEG data and its collection process. Although we used a large dataset, the amount of data was much less than is commonly used to train neural networks. Supervised learning algorithms generally require astronomical amounts of data to perform to the best of their capabilities, but large amounts of EEG data are time consuming to collect. They also need to be trained on distinct classes. The variety of classes needed to reconstruct a simple everyday scene is quite large, and the model needs hundreds of examples of each of these many classes in order to perform well. Many groups have used CNNs on EEG data and continue to do so, but CNNs and supervised learning do not seem to lend themselves to these paradigms.

## 5 Conclusion

We trained and tested numerous CNNs with the goal of being able to classify EEG data, corresponding to the visual stimuli of participants gathered in other studies. Our best model performed relatively well, with a test accuracy of 31.62% surpassing the overall accuracy of many other visual decoding CNNs. Despite being relatively successful, 31.62% is not a very high accuracy, and the best CNN we compared to only reached 34.4%. This was with a mere 20 object classes, far less than the number of different objects people see when looking at their surroundings, and as more classes are added, overall accuracy only drops. Additionally, CNNs need incredible amounts of data to train, and getting such quantities of EEG data is difficult. This suggests that CNNs trained via supervised learning on discrete classes are not the way to visual decoding, a view which is reinforced by researchers (Wilson et al., 2024). As this is an emergent field, not many other methods have been tried. A paradigm like reinforcement learning, in which the model teaches itself through trial and error, minimizing loss and maximizing reward, could be beneficial to explore. Alternatively, a model capable of decoding and reconstructing basic colors, textures, and shapes could work better than a model trained on specific classes, as theoretically, any object seen can be reconstructed from simple colors and forms. While our model was quite successful relative to other models like it, it is not worthwhile to put effort into improving a model dependent on a paradigm that simply does not lend itself to the data being used. Models alternative to convolutional neural networks, taught via supervised learning, need to be explored in the field of visual decoding.

## 6 Appendix

Code is available at our project's Github.

## Acknowledgements

This research received support during the PEP398 SKIL IV course, instructed by Professor Rainer Martini, PhD at Stevens Institute of

Technology.

## References

Gifford, Alessandro T. et al. (2022). “A large and rich EEG dataset for modeling human visual object recognition”. In: *NeuroImage* 264, p. 119754. ISSN: 1053-8119. DOI: `10.1016/j.neuroimage.2022.119754`. URL: `https://www.sciencedirect.com/science/article/pii/S1053811922008758`.

Guenther, Sven, Nataliya Kosmyna, and Pattie Maes (July 2024). “Image classification and reconstruction from low-density EEG”. In: *Scientific Reports* 14.1 AB - none, p. 16436. ISSN: 2045-2322. DOI: `10.1038/s41598-024-66228-1`. URL: `https://doi.org/10.1038/s41598-024-66228-1`.

Wilson, Holly et al. (2024). “Feasibility of decoding visual information from EEG”. In: *Brain-Computer Interfaces* 11.1-2, pp. 33–60. DOI: `10.1080/2326263X.2023.2287719`. eprint: `https://doi.org/10.1080/2326263X.2023.2287719`. URL: `https://doi.org/10.1080/2326263X.2023.2287719`.