Screen Recognition with Convolutional Neural Networks

Sofia Thomas   
NET ID: SXT220128  
CS 4375.5U1

Amulya Rayabhagi  
NET ID: AXR220242  
CS 4375.5U1

Noah Abraham   
NET ID: NXA220118  
CS 4375.5U1

Anthika Gunaselan   
NET ID: AMG22012  
CS 4375.5U1

***Abstract*—The purpose of our project was to apply a Convolutional Neural Network (CNN) to the intel image classifier dataset for scene recognition. Currently there is a trend in using advanced libraries to seamlessly create CNN’s however we were given the challenge of building our very own model using just the basic functions. Our model is built from scratch with the use of NumPy and PyTorch libraries for data processing, loading, and cleaning.**

**The dataset we used contained over 25,000 RGB images out of which we chose 10,000 to work with. Each image was already cleaned to be 150 x 150 pixels, however we had to downscale it to 64 x 64 pixels for usability in google colab (limited RAM). The images spanned from 6 different classes including buildings, forest, glacier, mountain, sea, and street, however our primary focus was on buildings, forest, glacier and mountain.**

**Our CNN was built using a multitude of techniques. We started with a convolutional layer using eight 3×3 filters for feature extraction. Then we used a ReLU activation for non-linearity. We also used a 2×2 max pooling layer to reduce spatial dimensions. Followed by that we had a flattening stage and finally a fully connected output layer with a Softmax activation for multi-class classification. We were able to use backpropagation in order to have iterative weight changes and our model was trained on stochastic gradient descent (SGD) and cross-entropy loss.**

**In order to get the best hyperparameters for our model we changed a variety of conditions. Some of the most varying objectives that we changed included: varying learning rates, epochs, and dataset sizes. The best model that we were able to create had a learning rate of 0.0043 over 60 epochs and was able to achieve an accuracy of 63.90%. We then were able to create metrics to understand the results that we had obtained by creating a confusion matrix, precision recall matrix and even ROC curves. These metrics show that four out of the six categories were able to train much better than those that were not in the reduced version of the training dataset.**

**Overall we were able to successfully build a manual CNN model without using any prebuilt libraries. The results that we were able to obtain show that even with constraints such as the usage of RAM we were able to get some optimal results. Some of our future recommendations for this project include, getting to train on higher quality images like the original 128 x 128 size by using a GPU instead of google colab. We could also try using data augmentation using torchvision.transforms during our preprocessing that would allow us to do random flips with our images. Finally the model that we created is considered to be a really shallow model as we looked at mostly shapes and edges. By increasing the number of layers and creating a more indepth model we could expand into higher level layers including textures, objects and scenes.**

***Keywords—convolutional neural network, NumPy, PyTorch, image classifier***

# Introduction

A convolutional neural network also known as the ConvNet or CNN is a type of feedforward neural network that learns features using filter optimization. While CNNs are advanced forms of the neural network, perceptrons and ANNs lay its foundation to create CNN.

A perceptron was the earliest form of the ANN, which was a single layer that took various inputs that were attached to weights which were summed up and passed through an activation function like the sigmoid function, Relu and others. The problem that they encountered was that perceptron was too simple and could only solve linearly separable problems. [1]

To overcome this problem, the advent of multi-layer perceptrons were created which are now known as ANNs. This contains three significant layers: input, hidden and the output layer which fully connects every neuron to the next layer and this works great for tabular or sequential data. But it struggles to apply to images. [1]

CNNs, on the other hand, are used to make predictions from data types such as text, images, and audio. The biggest difference between a regular Neural Network and a CNN is that, instead of looking at all the data at once like a regular neural network, CNNs use spatial patterns like textures, shapes, and edges.

CNNs have three main structures that we used when building our model: Convolutional Layer, Pooling Layer, and Fully Connected Layer. The convolutional layer extracts the features. The pooling layer shrinks the image by taking only the most important information, which helps the model focus on the bigger picture. It also reduces the size, allowing the network to learn faster and use less memory. Finally, the image is flattened into a list of numbers using the fully connected layer and passed through a normal neural network to make a prediction. [3]

The dataset that we chose to use for this project is the Intel Image Classification. It was created by Puneet Bansal in 2018. The dataset consists of around 25,000 images of size 150x150 distributed under 6 categories: 'buildings' -> 0, 'forest' -> 1, 'glacier' -> 2, 'mountain' -> 3, 'sea' -> 4, 'street' -> 5. Each image has 3 channels (RGB), totaling 150 × 150 × 3 = 67,500 features per image.

Our goal for the project was to create a CNN without using any built-in libraries that can accurately categorize the images given into these 6 categories. The network will be trained using cross-entropy loss and optimized using stochastic gradient descent.

# Theoretical and Conceptual Study

## History

CNNs work by learning features from matrix datasets, which is why they are so effective with visual datasets, like ours [3]. This effectiveness is why we chose to code a CNN for our visual dataset.

The idea behind CNNs originated from biology. The way neurons are connected in a CNN mirrors how neurons are connected in the visual cortex [3].

A 1968 paper by Huber and Russel on the visual cortices of a cat noted two types of visual cells: simple cells vs. complex cells. They created a model to show how the two types of cells work together to recognize patterns. It was this idea that inspired Kunihiko Fukushima to create the neocognitron model, a base model for the CNN that is used today [3].

The neocognitron model featured two types of layers used to recognize visual patterns. The S-layer (convolutional layer) features neurons that all share the same weights, which helps the model detect patterns. The features are then downsized in the C-layer. Here, noise is reduced and the features are more generalized. This ensures the model is general enough to recognize patterns across different images [3].

## Layers

A CNN is made up of different layers each of which help the model achieve its final output.

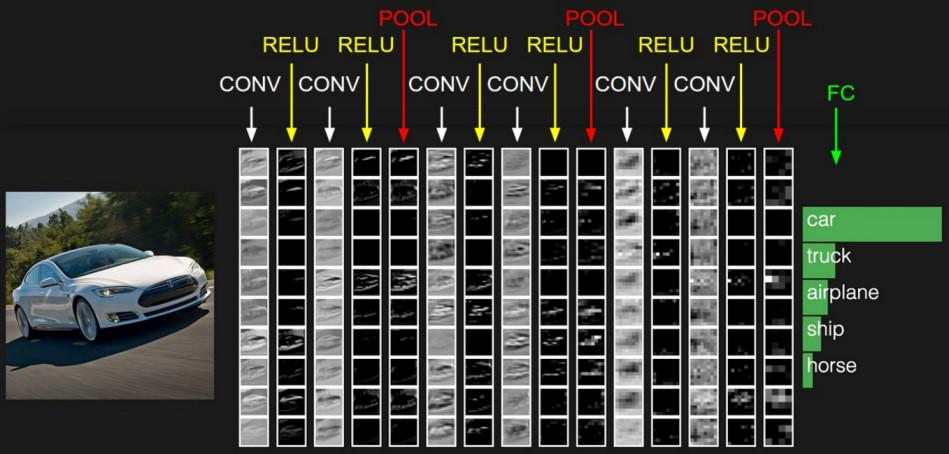
The convolutional layer uses filters that go over each image. This filter then creates a map which shows where the feature is present in the image [3].

The pooling layer shrinks the size of the feature maps and then takes the most relevant information from the feature maps to create a conclusion. The pooling layer ensures overfitting is avoided by using down-sampling. Pooling layers are often used after multiple convolutional layers. However, as of late, data scientists have began using smaller filters or not using a pooling layer, due to how the pooling layer may shrink a feature map too extremely [3]. Too much shrinking may lead to important information being lost.

The ReLU layer uses the activation function f(x) = max(0,x), which sets any negative values in the activation map equal to zero. It also brings nonlinearity to the model, meaning more complex relationships can be made between points. Fukushima also used ReLU in his model. Alternative activation functions that achieve the same outcome include the hyperbolic tangent function (f(x) = tanh(x)) and the sigmoid function [3].

The fully connected layer is the last layer in the model, which produces the final output based on the output from previous layers [3].

The loss layer measures the error of the model using a loss function. Loss functions such as mean squared error or entropy can give a better idea of how well the model is performing. This feedback is then used to backpropagate and improve the model. Backpropagation is the practice of taking steps “backward” in the model by updating the weights to reduce error [3].



1. Demonstration of how the convolutional, pooling, and ReLu layers can be used together. Source: [4]

# Results and Analysis

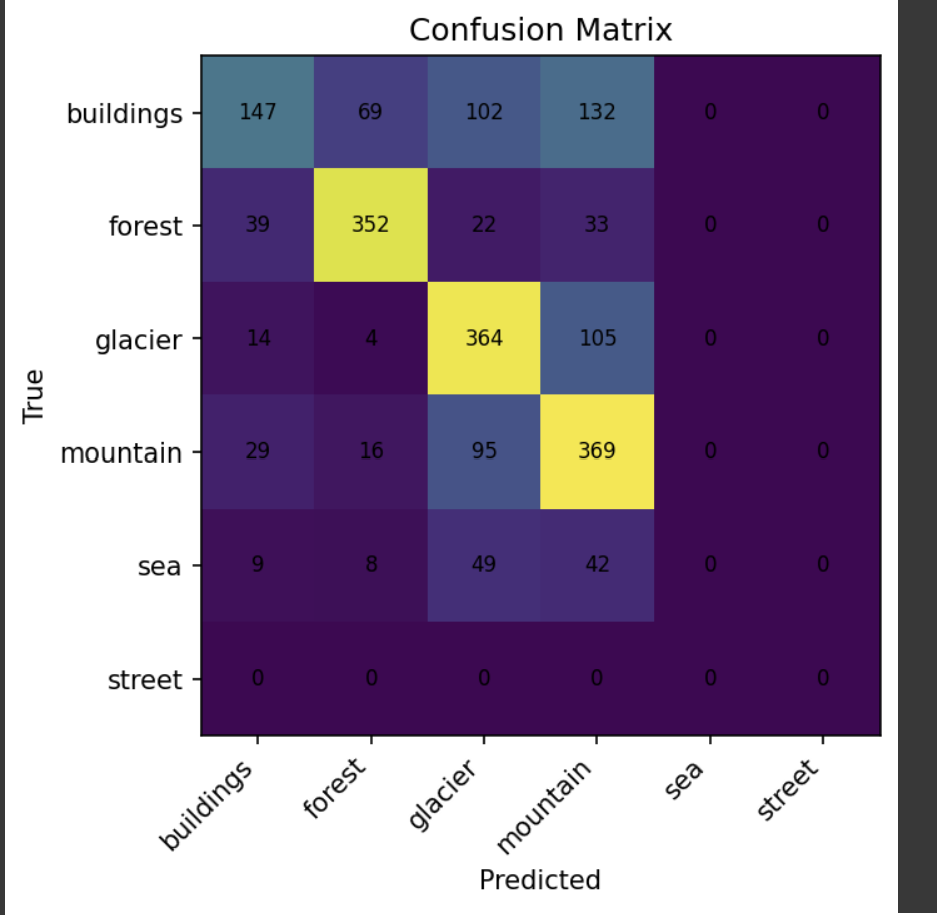
We evaluated our manually implemented CNN over multiple configurations, varying epochs, learning rates, and training sample sizes to understand the relationship between network parameters, computational constraints, and classification accuracy. All the results we measured on a held-out test set using accuracy, confusion matrices, precision-recall metrics, F1-scores, and ROC/AUC curves.

The model achieved an overall highest test accuracy of 63.90% for 80 epochs with a learning rate of .0043. We realized that that cost us a lot of RAM and we weren’t able to afford a costly runtime. The best average accuracy including considering our costly RAM consumption and time usage was 61.60% on 60 epoch, with the learning rate of .0043. This shows that while more training epochs allowed the network to refine weights further, limitations such as the usage of google collab forced us to prioritize memory efficiency over prolonged training.

We decided to look at the number of corrections compared to the incorrect predictions for each of the classes.

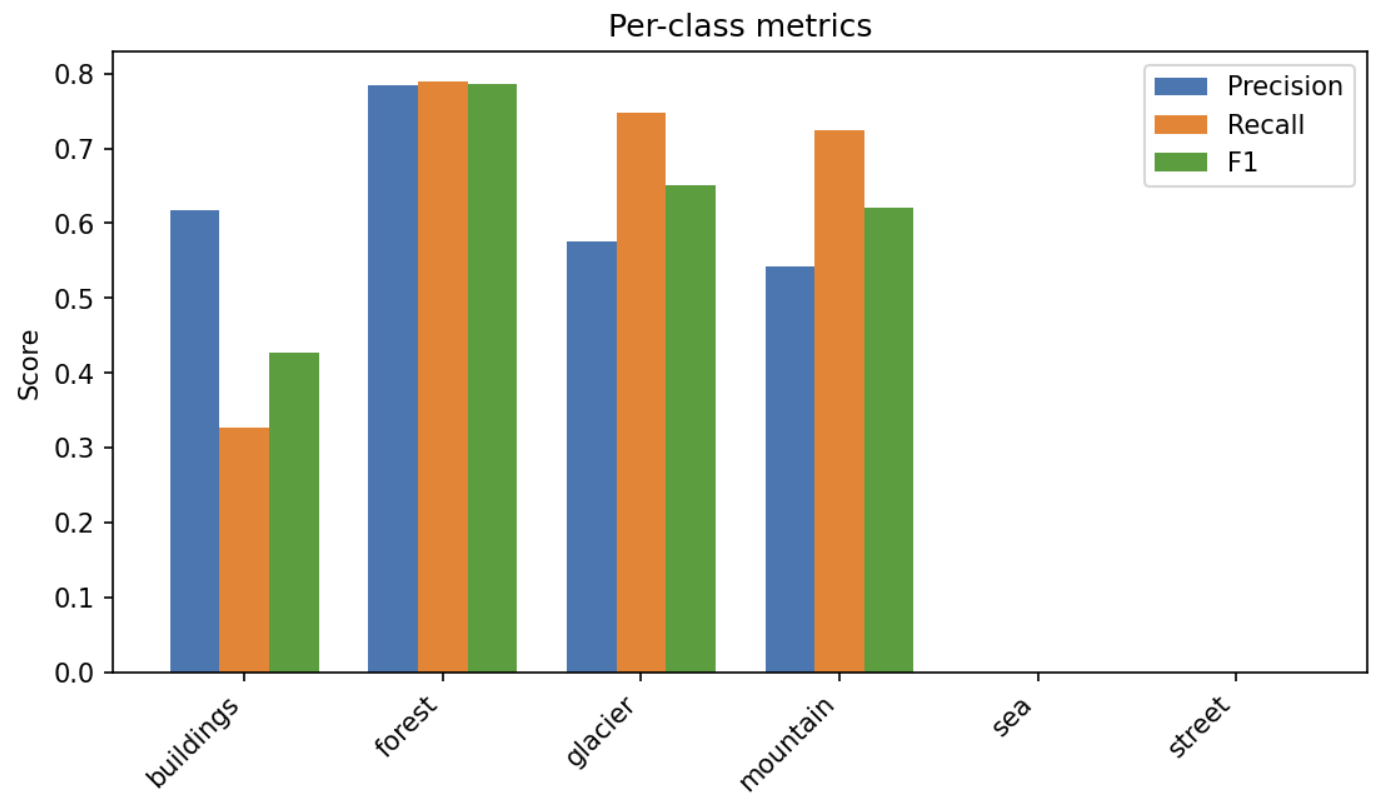


1. One of the experimental run that gave as a model accuracy on the test set of 60.15%



1. Confusion matrix of the various labels that show the predicted and true values of buildings, forest, glacier, mountain, sea and street.

As we reduced the sample size, our particular sample didn’t contain any street, so we classified only 5 of the 6 scenes. Based on the confusion matrix, we can analyze that the forest, glacier and mountain were the most accurately classified classes with their true values above 350. However, buildings on the other hand only accurately classified about 147 and the rest were incorrectly classified mostly as mountains. This may be due to similar structural edges or color palettes in certain lighting conditions. The sea took the worst hit as the model couldn’t classify the sea. This was partly because the sampled subset contained few or no sea images. Now taking a look out our precision recall and f-1 score.

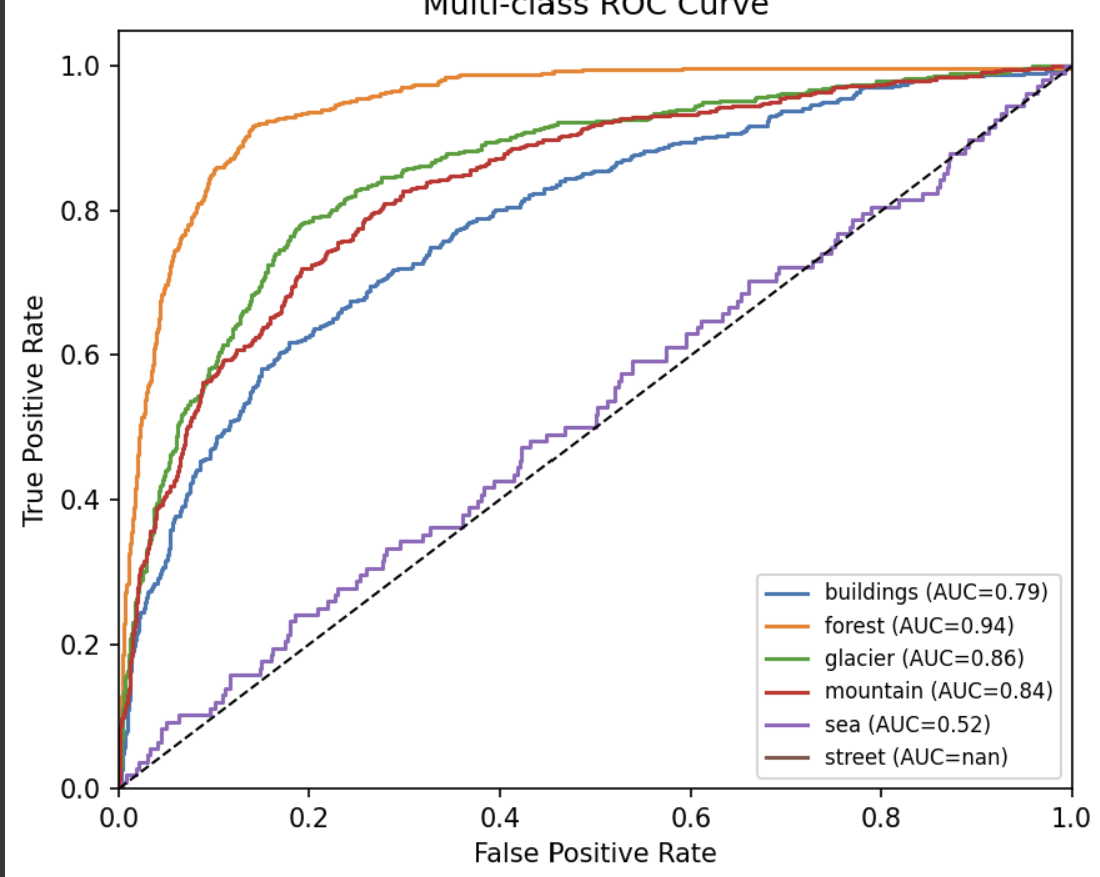


1. A graph that shows the precision, recall and F1 score of our model.

Similar to our confusion matrix, we can observe that the precision for forest is the hughes around 80% which means when the model predicts “forest” it accurately does that 80% of the time. The precision for the buildings, glaciers, and mountains have a range from 0.5 to 0.6.

Comparatively, forest again has the highest recall rate, but the glacier and mountain isn’t as far behind with above 70%. However the “buildings” recall rate is pretty low around 0.3 percent which can be connected to the confusion matrix.

Lastly the F-1 score shows once again the forest is the highest followed by glacier and mountain, with buildings being really low, and no evidence for sea and street. The imbalance shows the importance of dataset diversity and augmentation in future work.



1. A ROC Curve that plots the true positive rate against the false positive rate.

Finally we can take a look at the ROC curve that plots the true positive rate or the TPR against the false positive rate or the FPR at different thresholds. The AUC( area under the curve) is a good metric to help determine what a “good” model represents. As AUC is closer to 1 it shows that it’s a perfect model, while an AUC of 0.5 can indicate that it is the same as random guessing. Our main four classes: forest, glacier, mountain, and buildings have pretty high AUC metric which suggest that it does a great job in accurately distinguishing the scenes from each other most of the time.

Ultimately, we can observe that there are some key constraints we ran into when designing an effective model without the use of a pre-made library. Due to making the usage of RAM effective, we reduced the image size by 64 X 64 pixels instead of the 150 x 150 pixels. This caused the images to be more pixelated and removed fine details that may have been critical for distinguishing between certain classes such as the sea vs mountain.

We also faced class imbalance. Reduced subsets sometimes took out certain classes entirely, which skewed predictions and led to the absence of meaningful metrics for those classes. This in turn may have caused a decrease in our accuracy and other metrics.

Despite these constraints, we were able to achieve over 60% accuracy with a fully manual CNN under strict hardware limitations that demonstrates that the architecture is usable for scene recognition. However, the performance gap between our manual model and the optimized PyTorch models underscores the advantage of deeper architectures and hardware-accelerated computation.

# Conclusion and Future Work

## Conclusion

In this project we were able to successfully implement a CNN for the Intel Image Classification data set. In particular we were able to build the core infrastructure of the model without using any libraries and only used pytorch and numpy for data handling. This process allowed us to compare the process between manual implementation and library optimized architectures.

When doing our experiments of comparing the manual model to the library based one created using pytorch we were able to see that the one created using libraries took significantly less epochs ranging from 5 - 10 epochs for most of our tests, we were able to modify the number of convolution layers which we set to 3, added filters of (16, 32, 64), changed the optimizer between SDG and Adam, and use data augmentation to flip the images for better training and testing. When we implemented this many hyperparameters we were able to consistently get 80% accuracy with our model without running out of RAM that was given by Google Collab. However our manual model held up to the competitive accuracy and received approximately 60% accuracy. Based on the confusion matrix and ROC analysis confirmed that the model performed strongly in classifying forest, glacier, and mountain scenes but struggled with the sea and street categories. This was probably due to the image downscaling that had to be done which caused an imbalance in the data set.

After completing this project, we have gotten the opportunity to realize that even small changes go a long way when trying to find the best model parameters. Even when we made small changes to the learning rate or adjusting the number of epochs we were able to get better and better results every time we ran the model. This project taught us the fundamentals of how complex models like CNN can be broken down to small principles that we learned at the beginning of the semester and how we don’t need to rely on big frameworks to create a model. We hope that with better RAM and more data samples, this model can be improved significantly and can be used for a real-time scene detection scenario.

## Future Work

In the future we hope to improve on our already existing model in various ways to enhance performance, applicability and scalability of our project.

The biggest change that we would like to work to significantly improve our model is using higher resolution images and larger datasets. Instead of running our model on google colab we could use a GPU enabled environment that would allow us access to higher levels of RAM. This would allow us to train on higher pixelated images such as the original 150 x 150 sized images without having the need to downscale our images as much. By doing this the model would be able to capture more of the details from the images which in turn would allow for better feature extraction. In addition with the given increase in ram we could combine other datasets which would add to the number of features we could test while generalizing our model.

Our next recommendation is utilizing data augmentation in order to increase the generalization. By implementing changes such as random rotations, flips, brightness/contrast adjustments, and random cropping would allow for us to artificially expand our data set. This would help us reduce overfitting especially for the labels that didn’t perform so well such as the sea and street.

As you can see the model that we created is a pretty simple model which means that it is pretty shallow. Given the additional bandwidth we would recommend creating deeper CNN architectures. By stacking multiple convolutional layers, pooling layers, increasing the filter count, and utilizing batch normalization which was done in the Pytorch model we would be able to improve hierarchical feature learning. On top of that this would give us the opportunity to explore other architectures such as ResNet that would teach us how skip connections, depth, and filter scaling can affect classification accuracy.

In addition to creating a shallow network we were only able to experiment with the SDG optimizer. Instead we could use other optimization enhancements such as Adam, RMSProp which would help optimize our convergence speed and improve our training stability.

The hyperparameters that we chose for our experiments were done in a manual procedure. Instead if we did hyperparameter tuning either by using a grid search or a Bayesian optimization like we did in class would help us identify optimal learning rates, batch sizes, and epoch counts. This in turn could take a manual procedure that we did and turn it into a more efficient one.

Finally our dream application of how our model could be transformed is by creating real time scene detection applications. By utilizing the newer technologies that we have in place such as navigation systems, robots, environmental monitoring, and leveraging live camera feeds instead of static datasets we could potentially use models such as ours to find people who need to be rescued from various locations.

##### References

[1] GeeksforGeeks, “What is Perceptron | The simplest Artificial neural network,” *GeeksforGeeks*, Jul. 23, 2025. <https://www.geeksforgeeks.org/machine-learning/what-is-perceptron-the-simplest-artificial-neural-network/>

[2] GeeksforGeeks, “Introduction to Convolution Neural Network,” *GeeksforGeeks*, Jul. 11, 2025. <https://www.geeksforgeeks.org/machine-learning/introduction-convolution-neural-network/>

[3] Wikipedia contributors, “Convolutional neural network,” *Wikipedia*, Jul. 31, 2025. <https://en.wikipedia.org/wiki/Convolutional_neural_network#Building_blocks>

[4] Image of how convolutional, pooling, and ReLU layers work together from <https://www.slideserve.com/juanita/neural-networks>