# Lecture #19: Introduction to Deep Learning

**Vineet Edupuganti, Mihir Garimella, Ben Kotopka, Prathik Naidu, Rohan Suri, Trishiet Ray**
Department of Computer Science
Stanford University
Stanford, CA 94305
{ve5, mihirg, bkotopka, prathik.naidu, rsuri2, trishiet}@cs.stanford.edu

## 1 Introduction

So far in this class, we've covered a wide range of classical computer vision techniques, including edge detection, clustering methods, classifiers, and feature detectors/descriptors. While these techniques are still highly relevant to computer vision in research and industry, they share one major drawback: they rely on humans to hand-design features and choose classifiers that work well for the specific problem they're trying to solve. As a result, computer vision methods designed to solve one problem are often not readily adaptable to solving other problems, significant human effort and experimentation is required to build reasonably accurate vision pipelines, and even the best-designed pipelines are limited in the accuracy they can achieve simply because they were designed by humans and can therefore only pick up on patterns that human designers have picked up on first.

*Deep learning* is a relatively new method that is now a basic building block of modern computer vision research because it is free from these pitfalls. Instead of relying on hand-selected features and classifiers, deep learning approaches learn how to best digest and interpret data to solve any problem they're trained for. The key to these approaches is that they perform *end-to-end learning*, meaning that they learn how to directly map raw inputs (images in computer vision) to desired output (e.g., labels, segmented images, predictions). Deep learning approaches usually involve combining a small set of simple tools to build a network and then training that network on data for the specific problem you're trying to solve; they can often be adapted to different problems by simply swapping out the training data. Deep learning approaches are very loosely inspired by the brain in that they attempt to mimic neuronal activity in the neocortex of the human brain [**?** ].

## 2 Applications

Recently, deep learning has led to rapid progress on several problems in computer vision and related fields. In this section, we will provide a brief overview of progress and current state-of-the-art solutions for a small sampling of these problems.

### 2.1 Image Classification

*Image classification* is the problem of identifying all of the objects present in input images. One common benchmark for this task is performance on the ImageNet dataset [**?** ], a large, labeled dataset of 1,431,167 images containing 1,000 different objects (a 50× increase in the number of objects compared to previous classification datasets).

Initial state-of-the-art algorithms built for the ImageNet challenge were hand-tuned and used a series of carefully-selected, hardcoded features to tokenize test images before feeding this tokenized representation into a classifier, like an SVM, for final classification; for instance, the 2010 state-of-the-art for ImageNet was [**?** ], which managed to achieve a 28.2% top-5 error rate using a pipeline that first computed HOG and LBP descriptors for an image, coded and pooled them together, and then fed them into an SVM. However, so-called "shallow" approaches were quickly beaten by "deep"

approaches within the next few years; AlexNet [**?** ] in 2012 achieved a 16.4% error rate with a 8-layer network, VGG [**?** ] in 2014 achieved a 7.3% error rate with 19 layers, GoogLeNet [**?** ] later that year achieved a 6.7% error rate with 22 layers, and recently, in 2015, ResNet [**?** ] achieved a groundbreaking error rate of 3.57% with 152 layers—especially impressive considering that the error of a human performing the same classification challenge was only 5.1% [**?** ].

In general, this pattern indicates an exciting for the future of deep learning algorithms in computer vision; deep learning algorithms can rapidly outperform classical computer vision techniques—and even, in some sense, outperform humans—for very nuanced, difficult vision tasks, especially as the number of layers and network complexity is scaled up.

### 2.2 Object Detection

*Object detection* is an extension of the image classification task that aims to not only answer the question "what?" (i.e., recognize the objects in the image) but also answer the question "where?" (i.e., localize the recognized objects). Before deep convolutional neural networks (see Section 6 for more information on what these actually are), the mAP of "shallow" computer vision approaches on the PASCAL VOC dataset [**?** ] flattened out at around 40% in 2012, breaking the trend of continuous accuracy improvements in prior years and indicating that shallow approaches couldn't do much better for this task. However, deep learning approaches were able to increase this accuracy to around 52% just the next year, in 2013, and have been continuously demonstrating a rapid upward accuracy growth trend up to the current state-of-the-art performance for this task, an impressive 85.6% mAP from [**?** ].

### 2.3 Object Segmentation

*Object segmentation* involves first dividing an object into different semantic regions, usually representing different objects, and then determining what type of object each region contains. This is especially challenging to do generically with classical segmentation algorithms like the ones we learned in class—we need specific knowledge of the objects we're interested in detecting, and then must hand-design features tailored to each object using this prior knowledge (e.g., we know that donuts are circular with holes in the middle, so we design a feature that prioritizes these sorts of shapes). However, recent deep learning approaches have demonstrated impressive results to solving this difficult task of first segmenting an image and then identifying each object without having humans encode intuition about a small set of objects to look for; for example, [**?** ] uses three separate deep networks to achieve the results shown below.

### 2.4 Pose Estimation

*Pose estimation* involves tracking the skeleton of a human body configuration to understand what humans are doing across a sequence of images; [**?** ] uses convolutional neural networks to jointly learn to locate body parts and to then associate them with distinct individuals in an image.

### 2.5 Image Captioning

*Image captioning* is just what the name implies—algorithmically understanding the objects, actions, and relationships in an image to come up with a coherent English description of what that image depicts. Deep learning has demonstrated impressive results for this task: [**?** ] uses a combination of convolutional and recurrent neural networks to label images with a single caption, and [**?** ] uses a similar architecture plus an additional neural network layer that is responsible for localizing distinct objects/actions in an image to densely label an image with a caption for each object/action it depicts.

### 2.6 Other Computer Vision Tasks

*Visual question answering* is the task of answering plain-English questions about the objects/actions depicted in input images; for instance, given a picture of a pizza, an algorithm might be expected to answer "How many slices of pizza are there?" or "Is this a vegetarian pizza?". This problem is still considered quite challenging but one method, [**?** ], uses a recurrent neural network-based approach

that takes advantage of LSTM (long short-term memory) blocks to answer multiple-choice questions about an image's subjects and their actions.

*Image super-resolution* involves inferring a high-resolution version of a scaled-down input—this is an inherently difficult task because it involves generating details that you simply don't have to start out with, and making sure that this detail you generate is consistent throughout the image. [**?** ] uses a relatively recent network architecture called a generative adversarial network to recover realistic textures from downscaled images and managed to produce interpolated results that were closer to the original, high-resolution validation images than any other image super-resolution method.

### 2.7 Outside Computer Vision

Deep learning has also been used to demonstrate extremely impressive results to a variety of tasks outside of computer vision, including machine translation [**?** ], text generation [**?** ], speech recognition [**?** ], and speech synthesis [**?** ].

In addition, "deep reinforcement learning" approaches have demonstrated incredible success for several tasks. Deep reinforcement learning involves learning an optimal control policy (i.e., a series of actions to take given a particular state to reach an end goal) end-to-end through repeated trial-and-error, with the ultimate objective being to directly map state information (e.g., a screenshot of a video game) to the optimal action to take (e.g., the controller buttons to hit/directional pad inputs to supply). This approach has been used to demonstrate remarkable results for many game-based tasks in particular, including learning how to play Atari games [**?** ] and beating the world champion in the strategy board game Go [**?** ], long considered a benchmark of AI progress.

## 3 Motivation

Most deep learning problems are formulated as supervised learning tasks; data $X$ and labels $y$ are given, and the goal is to learn accurate predicted labels $\hat{y}$. Image classification (in which we want to place an image into one of several pre-specified categories) is a supervised learning task commonly addressed with deep learning approaches. The other approaches to image classification we've covered first extract hand-designed features, then use machine learning approaches (e.g. PCA or KNN) to learn models on this restricted feature set. Critically, the feature extractor is hand-designed and cannot be optimized during training.

A typical supervised image classification pipeline looks like the following: you start with a training set of images, across multiple different classes (for example: "airplane," "car," "bird," "cat," etc). Features are extracted from the images and fed into the neural network, which is trained on these labeled examples. Once a learned classifier is obtained, it can then take in the features from an unlabeled test image and make a classification prediction.

Most of the strategies covered in this class focused on reducing images to sets of hand-engineered features. These reduced feature sets encode (some of) the original information in the image and can be used to train classifiers through a variety of machine learning approaches.

Deep learning allows us to learn the features **jointly with the classifier**. This means we can achieve a much better model. Deep learning is motivated by the desire to derive informative features from something closer to first principles, in such a way that the feature extraction and classification modules are optimized together.

Deep learning models contain many layers, which learn hierarchical features. Lower layers learn simple features, while higher layers learn combinations of those.

## 4 Supervised Learning

### 4.1 Problem Formulation

Supervised learning is the process of learning a function from labeled training data $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$. In image classification, $x_i$ would be an image and $y_i$ would be an integer label (0 for dog, 1 for cat, etc.)

A loss function $L_i(y_i, \hat{y_i})$, given a training label $y_i$ and a corresponding perceptron $\hat{y_i} = wx_i$, specifies how well a classifier is currently performing. When the classifier predicts correctly ($y_i = \hat{y_i}$), the loss should be low; when it predicts incorrectly; ($y_i \neq \hat{y_i}$), the loss should be high. Loss over the entire dataset is an average of loss over examples:
$L = \sum_{i=1}^{N} L_i(y_i, \hat{y_i})$. When designing a model, you must choose the loss function. Some common examples are described below:

Given a set of training examples, the model will adjust its weights so as to minimize the value of its loss function. Oftentimes, this loss function will include a regularization term, R(w), that penalizes large weight values and limits model complexity. The model can then use these calculated weights to predict output labels or values for testing data. The testing error can subsequently be evaluated and used for model performance analysis.

The following equation provides a general solution to the problem of supervised learning.

$$W^* = \arg\min_W \frac{1}{N} \sum_{i=1}^{N} l(f(x_i, w), y_i) + R(w)$$

### 4.2 Linear Regression

Linear regression is a method of supervised learning where we attempt to model an input vector $x_i \in \mathbb{R}^{D_{in}}$ to an output vector $y_i \in \mathbb{R}^{D_{out}}$. Linear regressions attempts to model this through a simple linear transformation or matrix multiply:

$$f(x, W) = Wx$$

We can pose this as a learning problem where we attempt to learn the weights of $W$ by optimizing the a loss function which combines the euclidean distance $l(\hat{y}, y) = \frac{1}{2}||\hat{y} - y||_2^2$ with a regularizer $R(W)$. Using the frobenius regularizer, we can formally define this optimization problem as follows:

$$W^* = \arg\max_W \frac{1}{2N} \sum_{i=1}^{N} ||Wx_i - y||_2^2 + \lambda ||W||_{fro}^2$$

### 4.3 Softmax Classifier

Logistic regression (fitting a model that looks like $\frac{1}{1+e^{-(b_0+b_1 x)}}$ instead of $y = b_0 + b_1 x$ for linear regression) is similar to linear regression, except that the outcome (dependent variable) has only a limited number of possible values. Generalizing logistic regression to multi-class problems allows us to constrain our output to the range [0, 1], which can be treated as a probability.

A softmax classifier allows us to treat the outputs of a model as probabilities for each class. A common way of measuring distance between probability distributions is the Kullback-Leibler (KL) divergence, which can be interpreted as the negative log of the probability that the model makes a correct prediction for each y.

$$D_{KL} = \sum_y P(y) \log \frac{P(y)}{Q(y)} = -\log Prob[f(x_i, W) = y_i]$$

where P is the ground truth distribution and Q is the model's output score distribution. We then apply the softmax function, $\frac{e^{\hat{y}} k}{\sum_j e^{\hat{y}} j}$, to constrain our probability to the range [0, 1].

$$Prob[f(x_i, W) = y_i] = \frac{e^{\hat{y}} k}{\sum_j e^{\hat{y}} j}$$

### 4.4 Neural Networks

In some cases, a simple linear transformation is too simplistic of a function for our purposes and we would like to introduce a more complex model for our supervised learning task. At first, we may be tempted to merely introduce additional matrix multiplies as follows:

$$f(x, W_1, W_2) = W_2 W_1 x$$

This however, does not actually introduce any additional model complexity since $W_1 W_2$ can be written as just one matrix multiply where $W = W_1 W_2$, which brings us back to our original linear regression function, $f(x, W) = Wx$. Instead, we need to introduce nonlinearities between each matrix multiply. This produces the following new model where $W_1 \in \mathbb{R}^{H \times D_{in}}$ and $W_2 \in \mathbb{R}^{D_{out} \times H}$:

$$f(x, W_1, W_2) = W_2(\sigma(W_1 x))$$

In this model, a matrix multiply $W_1$ is followed by an element-wise non-linearity function $\sigma : \mathbb{R}^H \Rightarrow \mathbb{R}^H$, which is followed again by another $W_2$ matrix multiply. This is an example of a one-layer neural network. The introduction of the non-linearity function, $\sigma$, allows this model to be much more expressive and powerful than a simple linear model.

An activation function is a non-linear max function allows models to learn more complex transformations for features. Given that the use of the non-linearity function is a large part of what makes neural networks superior to basic linear models, a great deal of research has gone into finding the optimal function for use in these systems. The sigmoid function, defined as $\sigma(x) = \frac{1}{1+e^{-x}}$ has risen to popularity in part because it is continuous and easy to differentiate, while also bounding its inputs. Meanwhile, the rectified linear unit, defined as defined as $\sigma(x) = max(0, x)$ is another example of a popular activation function that has seen superb performance in literature as well as favorable efficiency. Other common functions include functions include tanh, Leaky ReLU, Maxout, and ELU.

The vector produced by each non-linearity (also called the activation function) is called a hidden layer. These types of models can be increased in size to include more matrix multiplies and thus more hidden states. The previous, one-layer neural network can be expanded to a two-layer neural network as follows:

$$f(x, W_1, W_2, W_3) = W_3 \sigma(W_2(\sigma(W_1 x))$$

Given a differentiable loss function (we can use the same one used in linear regression), this new neural network architecture can be optimized using a variety of optimization techniques. As the network is trained, the hidden layers begin to represent "learned features" regarding the input. Importantly, this is one of the primary motivations behind this type of model as it moves away from hand-crafted features.

## 5 Gradient Descent

Optimization of weights in a machine learning algorithm is key to building a more robust algorithm with predictive power. As described above, we ultimately want to minimize the following function to help determine the best weights:

$$g(w) = \frac{1}{N} \sum_{i=1}^{\infty} l(f(x_i, w), y_i) + R(w)$$

One approach to optimize this function is a closed form solution (directly solving through an equation) but this method can be highly expensive. Similarly, random search for the best weights is inefficient especially for high dimensions.

The goal of gradient descent is to efficiently search for optimized parameters that improve the preformance of the model rather than trying to randomly search through weights. Gradient descent works by calculating how the function $g(w)$ changes given small changes in the weight vector $w$.

The classic way to determine this change is by calculating the derivative $g'(w)$ assuming w is scalar, but if w is a vector (representing features in multiple dimensions) then we must calculate the gradient of $g(w)$ across each component of $w$

Based on this idea, we can come up with a straightforward algorithm for gradient descent that is useful in deep learning:

Initialize w randomly
**while** *true* **do**
    read current;
    Compute gradient $\nabla g(w)$;
    w = w - $\alpha * \nabla g(w)$
**end**

We can compute the gradient as the partial derivative of the loss function with respect to the weight: $\frac{dL}{dW} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dW}$ where $\hat{y} = Wx$ and $L = Loss(\hat{y}, y)$.

With this approach, we can iteratively update the weights of our model and use the learning rate ($\alpha$), which is a hyper-parameter that tells gradient descent how quickly to step in search for a minimum.

Backpropagation, short for "backward propagation of errors," is an algorithm used for gradient descent. The key insight is to visualize the computation as a graph: compute the forward pass to calculate the loss, then compute all gradients for each computation backwards

## 6 Convolutional Neural Networks

Convolutional neural networks are a very successful and commonly used subset of deep learning models, which adapt the convolutional filters discussed in earlier lectures. Starting from an original image (say, 32x32 with 3 color channels), the image is convolved with multiple filters. (In the example used in class, a 5x5 filter was used, only over the "valid" region of the image - so no overlap with the region outside the image; the output side length is 32-5+1 = 28.) Crucially, the parameters of these filters are optimized jointly via backpropagation - we are no longer limited to hand-chosen features.

Using convolutions gives us translational invariance - a feature is expected to mean the same thing no matter where in the image it appears. Multiple independent convolutional filters can be passed over the original image, and the outputs stacked to give what is essentially another image with more channels. Nonlinearities are applied as in other deep learning models, and further convolutional layers can be trained on this output to learn higher-level features (combinations of the lower-level features). This allows for the expressive models arising from simple components which are a hallmark of deep learning.

The GoogLeNet case study takes this approach one step further by building "Inception modules" - a particular pattern of layers which are then concatenated. To train this deep net, Szegedy et al. [**?** ] used "auxillary classifiers" to effectively try to predict image class based on only some of the GoogLeNet's layers; in training, a weighted sum of all outputs was used as the loss function.

## 7 Conclusion

Deep learning is a relatively recent paradigm in computer vision and related machine learning fields which co-trains feature extractors and classifiers instead of relying on human skill to identify salient features. Deep learning approaches are the state of the art in most computer vision problems today; however, in settings where training examples are expensive or inherently limited, other methods covered in this class can still be very useful, as human intuitions about a problem domain can be used to choose good sets of features.

## References

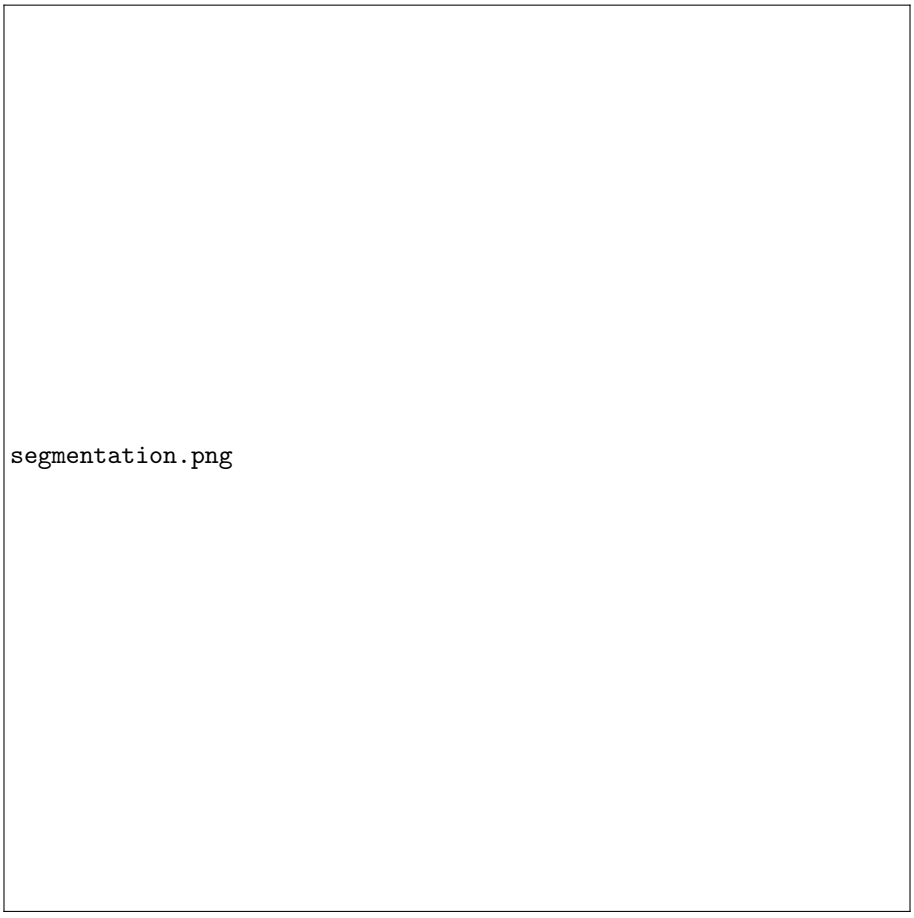


Figure 1: Object segmentation results from [**?** ].

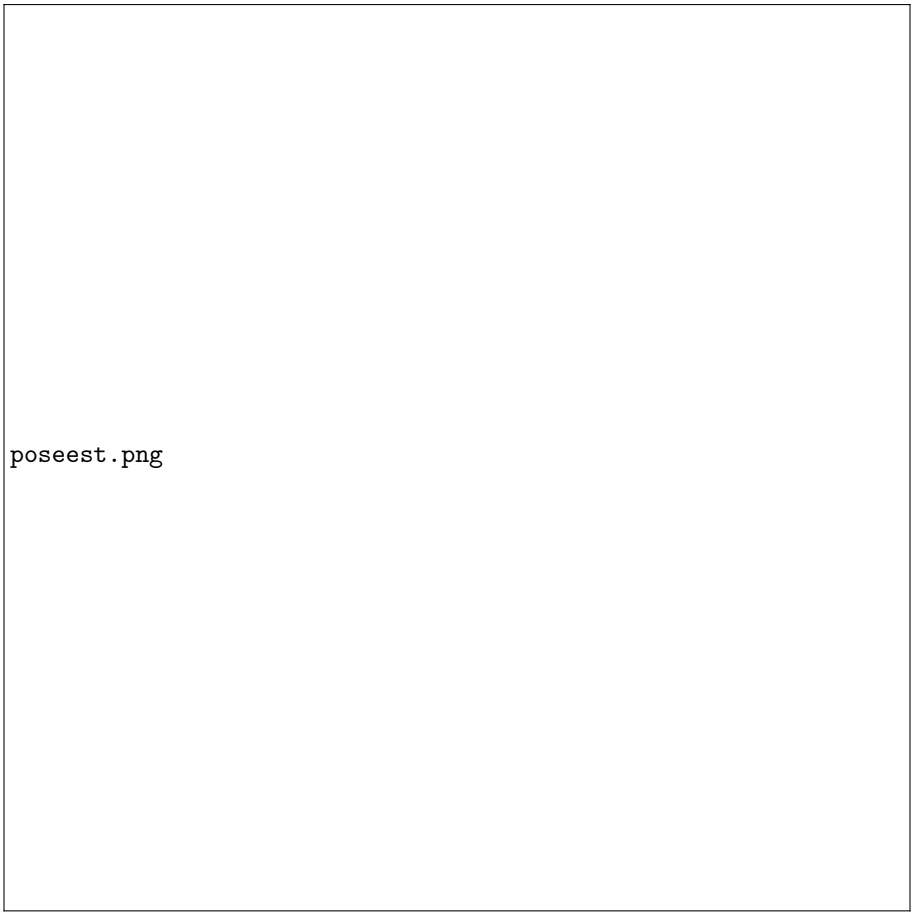


Figure 2: Pose estimation results from [**?** ]; here, identified human skeletons are labeled with bright colors to indicate pose.

densecap.png

Figure 3: Dense image captioning results from [**?** ].

Conventional_schematic.png

Figure 4: Image classification with "traditional" computer vision. (CS 131 lecture slide 19-28)

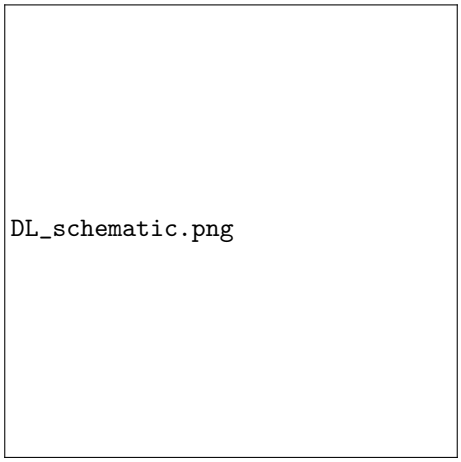


Figure 5: Image classification with deep learning. (CS 131 lecture slide 19-32)


Figure 6: Hierarchical features emerge in deep learning models. (CS 131 lecture slide 19-33)

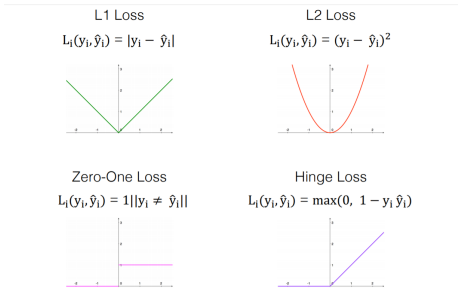


Figure 7: Common loss functions in machine learning (2019 CS 131 lecture slide 45)

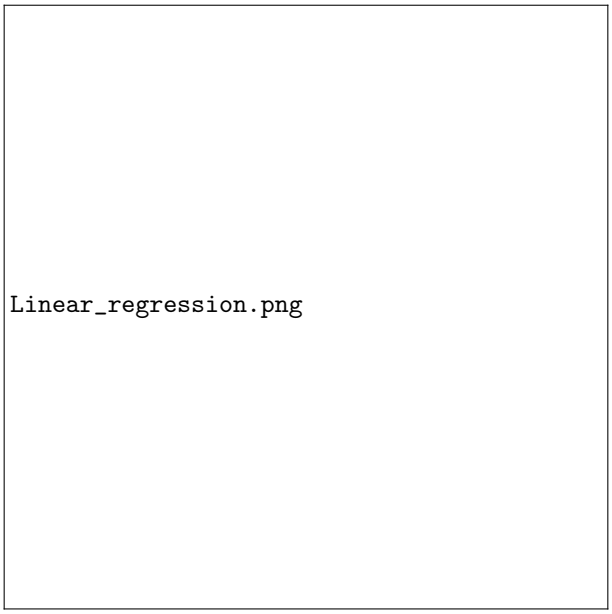


Figure 8: An example of a simple linear regression model being used to classify an image. Each row of $W$ represents the weights used to sum the input for each output class. (CS 131 lecture slide 19-42)


Figure 9: Depictions of the rectified linear unit and sigmoid activation functions (CS 131 lecture slide 48)

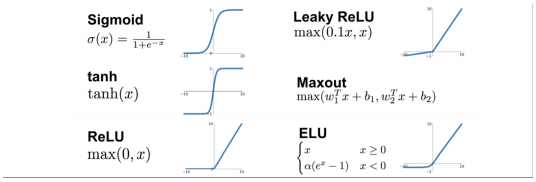


Figure 10: Depictions of other activation functions (2019 CS 131 lecture slide 91)

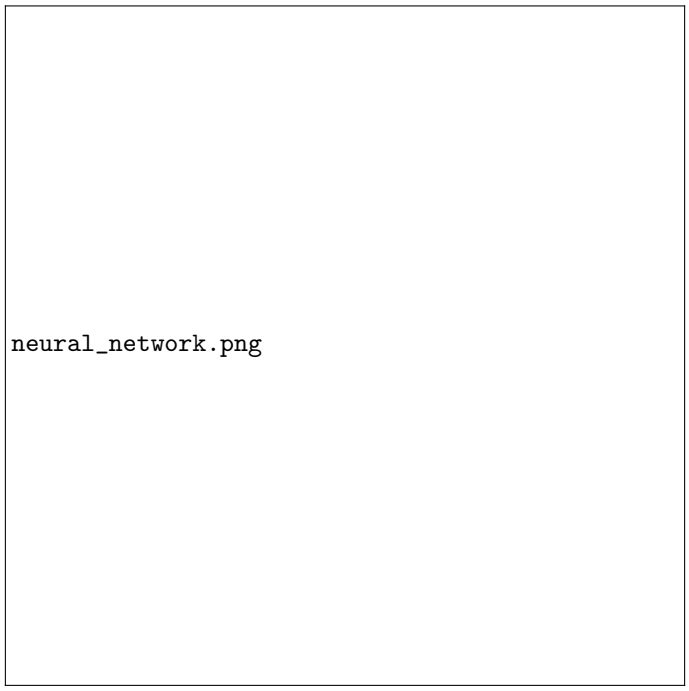


Figure 11: An graphical representation for one-layer and two-layer neural network architectures. (CS 131 lecture slide 19-52)


Figure 12: Extracting an activation map from a raw image in a CNN layer. (CS 131 lecture slide 19-97)

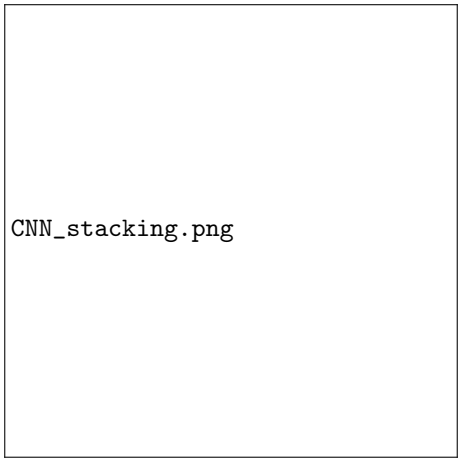


Figure 13: Stacking CNN layers to build a deeper model. (CS 131 lecture slide 19-100)

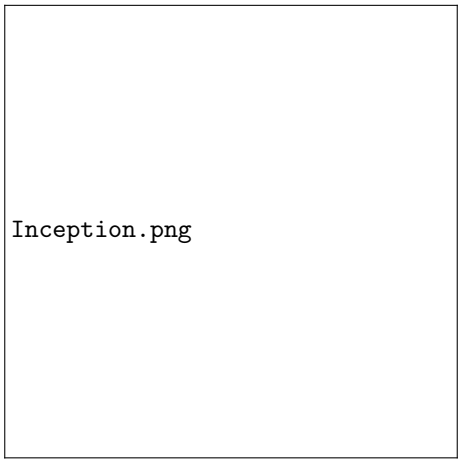


Figure 14: GoogLeNet architecture and Inception module. (CS 131 lecture slide 19-101)