1 Definitions

Convolution Neural Network: A type of neural network that is particularly well-suited for image and video processing tasks. CNNs consist of the following building blocks:

1. **Convolutional Layer:** Involves sliding a small window of fixed size across the input data and computing the dot product between the kernel and the overlapping portion of the input at each position. The result of each dot product is then summed to produce a single output value, which is stored in a new output.

   (a) **Kernel/Filter:** A small multidimensional array of weights that is used to scan over the input data during the convolution operation

   (b) **Stride:** The number of pixels or steps by which a filter is moved across the layer input data

   (c) **Channel:** An additional dimension within a layer of the network that represents a specific feature or attribute of the data. For example, an RGB image has three channels that correspond to the red, green, and blue color channels. The output of a convolutional layer usually has many different channels to allow the network to learn to detect different patterns in each channel. For example, the output of the first convolutional layer may learn to detect edges at different angles in each channel.

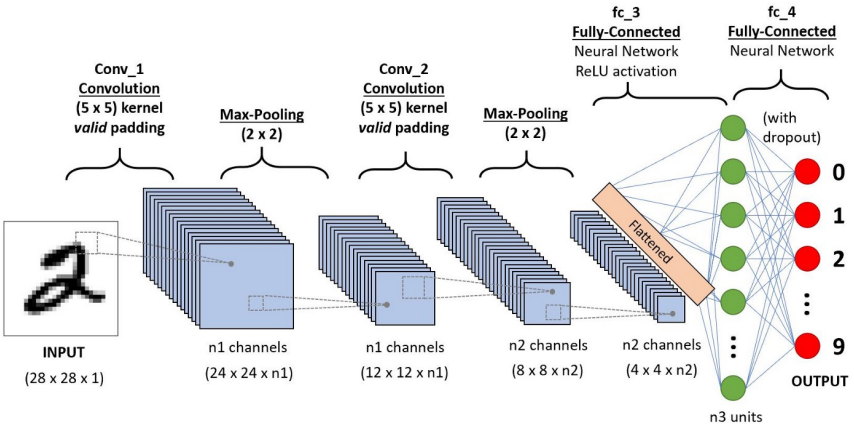
   (d) **Padding:** Extra space added to the outsides of the layer input (filled with zeros or a copy of the data on the border) to make the output dimensions have the desired size.

2. **Pooling Layer:** Used to reduce the spatial dimensions of the data. Two popular types of pooling are max pooling and average pooling.

   (a) **Max pooling:** Similar to a convolution, but in this case, the maximum value at each kernel location is selected. Analogous to picking the brightest pixel in a grid of small regions of the image and tossing the rest.

   (b) **Average pooling:** The average value of all the pixels at each kernel location is selected. Analogous to smoothing out an image while reducing its size.

3. **Fully-connected layer:** Typically a CNN will end with one or more fully-connected layers. These layers effectively perform classification or regression based on the features coming from the convolutional layers.



2 CNNs and not the kind on TV

2.1 Conceptual Questions

1. What are some benefits of CNNs over fully connected (also called dense or linear) layers?

   CNNs are good for image-related machine learning tasks because they learn the kernels that do feature engineering. Additionally, CNNs, through the use of convolutions and pooling, take advantage of local spatial coherence. This refers to the tendency of neighboring pixels in an image to have similar values or characteristics. CNNs are also parameter efficient.

2. How are the weights in a CNN updated during training?

   Backpropogation, the same as fully connected layers.

3. What are the parameters of a convolutional layer?

   The values in the kernels (a.k.a. weights or filters) and the bias term per output channel. The kernel sizes are num_input_channels by kernel_width by kernel_height for each output channel.

4. What are the hyperparameters of a convolutional layer?

   Stride size, padding size, filter size.

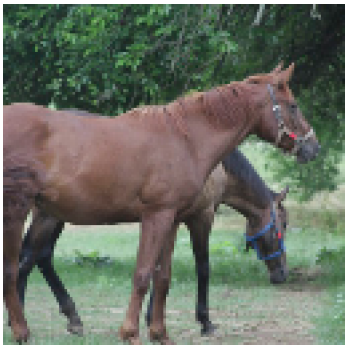
5. Can convolutions only be applied to 2-D shapes like images?

   Nope. Convolutions are very common on 1-D signals like sound or a stock market time series. You can also combine 2-D space with time and do convolutions for video. Or even higher dimensional data like 4-D cardiac MRI data.

2.2 Shapes and Parameters

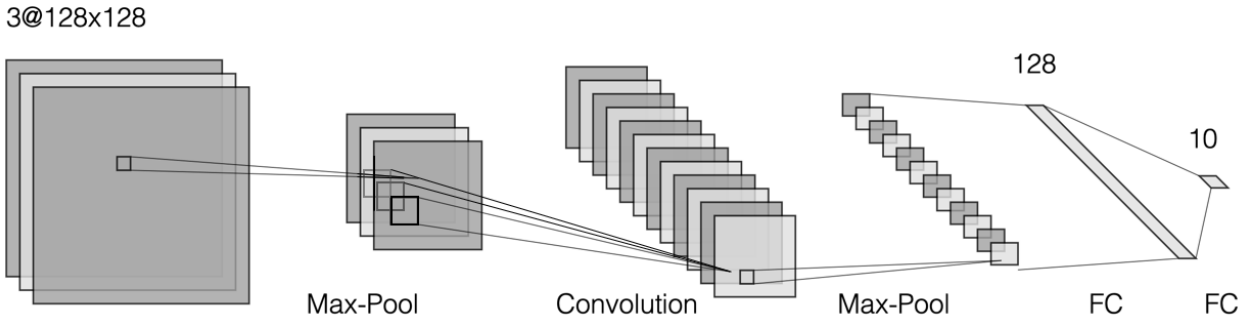
Torch the Triceratops is working on his 10-315 final project and is looking into the MS-COCO dataset. His goal is to classify images that belong to one of ten possible classes (i.e. `[cat, dog, bird, turtle, ..., horse]`). The images come in RGB format (one channel for each color) and are downsampled to dimension 128×128.

The following illustrates one such image from the MS-COCO dataset.



Torch constructs a convolutional neural network that has the following structure: the input is first max-pooled (not common) with a 2×2 filter with stride 2. The results are then sent to a convolutional layer that uses a 17×17 filter (uncommonly large) with stride 1 and 12 output channels. Those values are then passed through a max-pooling layer with a 3×3 filter with stride 3. The result is then flattened and passed through a fully-connected layer (with ReLU activation, not shown) with 128 hidden units, followed by a fully connected layer (softmax activation, not shown) with 10 hidden units. The final 10 hidden units represent the categorical probability for each of the ten classes. With enough labeled data, Torch plans on using some optimizer like SGD to train this model using backpropagation.

Note: Assume we have bias terms in all neural network layers unless explicitly stated otherwise.



1. What is the shape of the output tensor after max pooling?

   With a stride of two the size is cut in half in each direction. Unlike convolution, pooling is applied independently per channel.
   $3 \times 64 \times 64$

2. What is the shape of the output tensor after the convolutional layer?

   Thinking in just one dimension, we can fit a window of width 17 within the larger width of 64 and then shift it by one 47 more times before it hits the right edge.
   $12 \times 48 \times 48$

3. What is the shape of the output tensor after the second max pooling layer?

   $12 \times 16 \times 16$

4. What is the general equation for the shape of the output tensor if a filter of size $F \times F$ with stride S and padding P is applied to a tensor with width W and height H?

   output width $= \frac{W+2P-F}{S} + 1$

   output height $= \frac{H+2P-F}{S} + 1$

5. How many parameters are in this network for the first max pooling layer?

   Zero. Max-pooling layers do not have any parameters, since a fixed operation is performed on the input.

6. How many parameters are in this network for the convolutional layer?

   $12 \times (3 \times 17 \times 17) + 12 = 10416$

7. How many parameters are in this network for both fully connected layers?

   $(3072 \times 128 + 128) + (128 \times 10 + 10) = 394634$

8. From these parameter calculations, what can you say about convolutional layers and fully connected layers in terms of parameter efficiency (the ratio between the number of parameters from some layer type and the total number of parameters)? Why do you think this is the case?

Total number of parameters = 10416 + 394634 = 405050

Percent from convolutional components = $\frac{10416}{405050} = 2.57\%$

Percent from fully connected components = $\frac{394634}{405050} = 97.43\%$

Convolutional layers are much more parameter efficient, mainly because we are reusing the convolutional filter repeatedly for each convolutional layer (we only need to train one kernel per channel per layer). In comparison, the fully connected layer requires all nodes between two layers to be fully connected.

3 Neural Network Applications

3.1 How Smiley is Smiley?

We've provided you a dataset of faces with numerical smiley-ness scores. In this activity we want to build a regression model based on the degree of a smile.

1. What a good feature we can look at to determine the degree to which a face image is smiling?

   curvature of the mouth

2. Given the problem statement, construct a general scoring system you could give the different types of smiles. Hint: consider a frowny, neutral, and smiley face, what numerical values would you give each of these?

   Frown: -1
   Neutral: 0
   Smile: 1

3. Now we will take the curvature of the mouth specifically and develop a linear regression model that can take in a image of a face as input and predict the 'smiley-ness' score.
   In order to do so we must construct a weights matrix. The faces notebook contains starter code for this activity. Complete the cells marked with 'TODO'. What weights did you come up with and what mean square loss did you obtain?

   Frown pixel weight:
   $w[16, 14] = -1.0/255$
   $w[17, 14] = -1.0/255$

   Neutral:
   $w[18, 14] = 0.0/255$

   Mostly smiley pixel weight:
   $w[19, 14] = 1.0/255$
   $w[20, 14] = 1.0/255$

   Loss: 0.16870186082276048

3.2 Word Embeddings

3.2.1 Definitions

Some terms that may be helpful to know while going through this activity

1. **Corpus**: A corpus refers to a large and structured set of texts in a specific language or domain, typically used for linguistic analysis, training of language models, and other NLP tasks.
2. **Token**: In NLP, a token represents a single unit of a sequence, typically a word or a punctuation mark, obtained by splitting the input text based on predefined rules.
   For example: In the sentence "The cat sat on the mat.", the tokens are: "The", "cat", "sat", "on", "the", "mat", and ".".
3. **Word Embedding**: A word embedding is a technique used to represent words from a vocabulary in a lower-dimensional vector space

3.2.2 Word2Vec Activity

For this activity we will be walking through the word embeddings notebook, and implementing portions of it, namely the similarity and objective functions.

We are given the lines in Green Eggs and Ham and want to develop word embeddings for the corpus from that story. Namely, our goal is to figure out the likely next token, given a specific token.

1. What technique can we use to determine the closeness of two tokens/words? How can we use it?

   We can use cosine similarity to determine the similarity between two vectors.

   Cosine similarity measures the cosine of the angle between them. It ranges from -1 to 1, where 1 indicates perfect similarity (the vectors point in the same direction), 0 indicates no similarity (the vectors are orthogonal), and -1 indicates perfect dissimilarity (the vectors point in opposite directions).

   We can simplify this calculation to determine the dot product between the two vectors. That is, the vectors are similar if the dot product is large.

2. After implementing the similarity function and completing the objective implementation, run the training loop. What final loss do you obtain for the entire corpus at the start and after 100 epochs?

   Start: 5.7253
   100 epochs: 3.5561

3.3 minGPT