2.2. Machine learning models

Throughout the project, ~~different~~ various machine-learning algorithms will be used to train and evaluate the ~~effect of using~~ effectiveness of contrastive learning. This section provides a summary of these algorithms.

2.2.1. Logistic regression

- Logistic regression is classification and not as far as I know used to do true series prediction (linear regression is used for that
- Even for linear regression, the input is often not "historical"

Logistic regression is an established statistical model that is used to predict target values based on historical data. Logistic regression estimates the target values $\hat{\boldsymbol{y}}$ by applying the sigmoid function $\sigma(\cdot)$ to ~~the~~ ~~weighted~~ input data ~~with a constant factor b~~:

a linear transformation of the

this is intended as a "linear transformation"

$$\hat{y} = \sigma(z) \tag{2.1}$$

where z is calculated as:

$$z = \sum_i \boldsymbol{x_i} \times \boldsymbol{w_i} + b \tag{2.2}$$

and the sigmoid function is defined as

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

and maps values between $(-\infty, \infty)$ to a value between $(0, 1)$. The model takes in the input data x_i and multiplies each component of the input vector with a weight w_i, where the weights are optimised during training to maximise the likelihood criterion. The result is passed to the sigmoid/standard logistic function ~~to~~ which map ~~the results~~ it to between 0 and 1. The output can be seen as the probability that the particular input variable belongs to ~~a certain~~ the positive class, i.e. $\hat{y} = P(y = 1|\boldsymbol{x}, \boldsymbol{w})$ where $y = 1$ if it belongs to the ~~specified~~ positive class, and 0 otherwise. Logistic regression can be seen as a single ~~layer~~ node of a multi-layer perceptron with a sigmoid activation function as shown in Figure 2.1.

interpreted

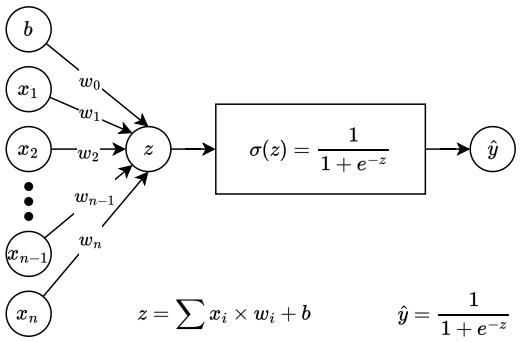


Figure 2.1: Logistic regression ~~is shown as a~~ multi-layer perception with input $\boldsymbol{x}$, a sigmoid activation function $\sigma(\cdot)$ and output $\hat{\boldsymbol{y}}$.

can be viewed as a single node of a

you do not have vector output in your figure.

2.2.2. Multi-layer perceptron

A neuron is a biological structure in the brain that transmits ~~information throughout the model~~. The artificial neuron, also referred to as a perceptron, is based on the biological neuron as shown in Figure 2.2. It receives input $\boldsymbol{x}$ from a source, adds a level of importance $\boldsymbol{w}$ to each input signal and generates outputs $\boldsymbol{y}$.

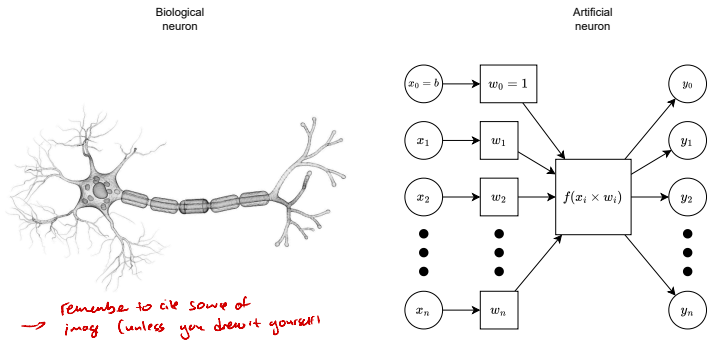


Figure 2.2: Biological and artificial neuron.

A multi-layer perceptron (MLP) consists of ~~multiple~~ of these neurons/perceptrons as shown in Figure 2.3. It generally consists of input $\boldsymbol{x}$, M hidden layers, an activation function $\sigma(\cdot)$, and the final output $\hat{y}$, where the final output can be either a single value for binary classification or multiple values for multinomial classification.

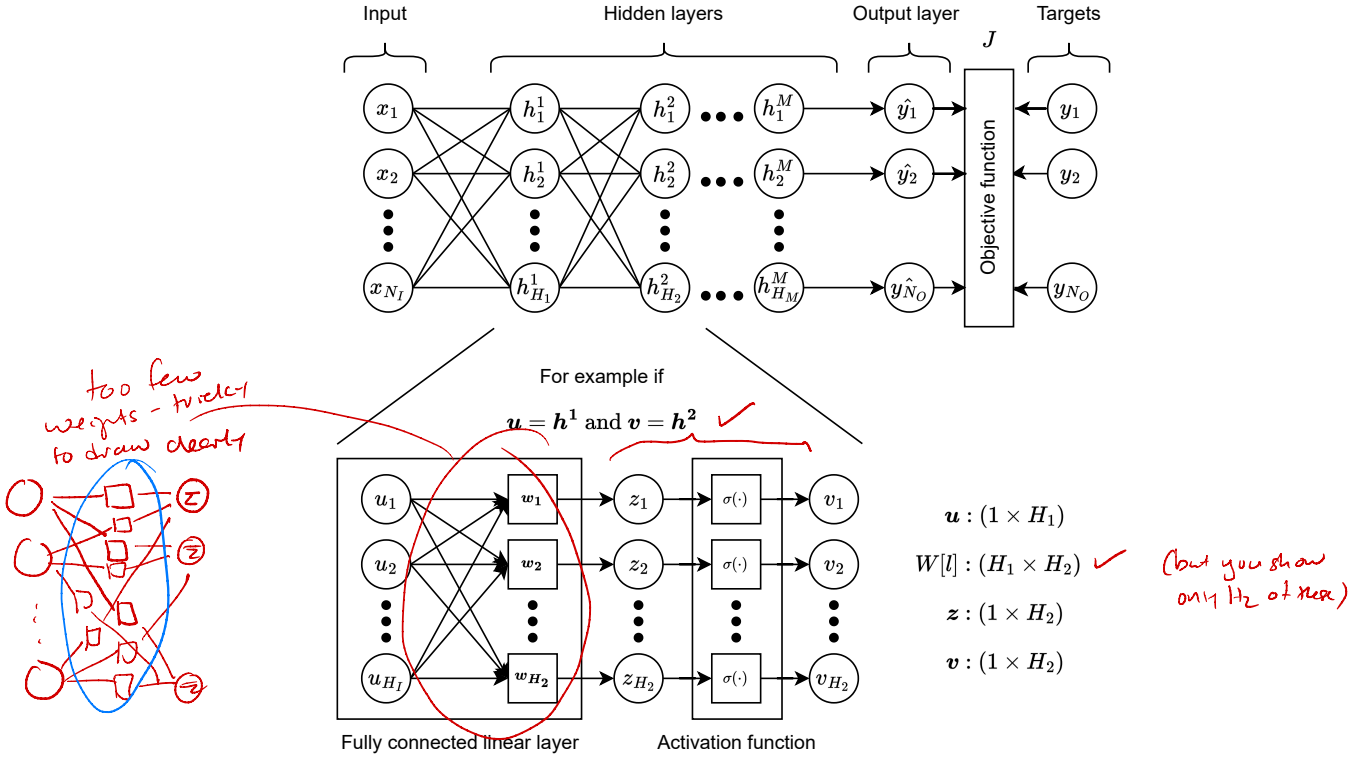


Figure 2.3: Multi-layer perceptron with input layer x with N_I inputs, M hidden layers each with $H_1, H_2, ..., H_M$ nodes, and predictions $\hat{\boldsymbol{y}}$.

A multi-layer perceptron is trained using a forward pass. First, the input data and the weights are initialised. For each layer in the MLP, the data $\boldsymbol{u}$ is multiplied by the weights $\boldsymbol{w}$. A non-linear transformation $\sigma(\cdot)$, referred to as the activation function, is applied to the result to generate the output of each layer $\boldsymbol{v}$. The output of the current layer is then used as input to the next layer and the output of the last layer is the predicted values generated by the model $\hat{\boldsymbol{y}}$. The predictions are compared to the true labels or targets using a cost function/objective function. For example, if the squared error is used as an objective function, then

$$J = \frac{1}{2}(\boldsymbol{y} - \hat{\boldsymbol{y}})^T(\boldsymbol{y} - \hat{\boldsymbol{y}}) \tag{2.4}$$

To optimise the weights of the multi-layer perceptron, the objective function is minimised - allowing for the best estimation of the true labels. This occurs during a process known as back-propagation. After the model has passed through the entire forward pass, the derivative of the objective function J relative to the weights for each layer $W[l]$ is calculated:

$$\frac{\partial J}{\partial W[l]} = \left[\frac{\partial J}{\partial \boldsymbol{w}_1[l]}, \frac{\partial J}{\partial \boldsymbol{w}_2[l]}, ..., \frac{\partial J}{\partial \boldsymbol{w}_{H_l}[l]}\right] \tag{2.5}$$

for node i in layer l:

$$\frac{\partial J}{\partial \boldsymbol{w}_i[l]} = \frac{\partial \boldsymbol{z}[l]}{\partial \boldsymbol{w}_i[l]} \times \frac{\partial \boldsymbol{v}[l]}{\partial \boldsymbol{z}[l]} \times \frac{\partial J}{\partial \boldsymbol{v}[l]} \tag{2.6}$$

These derivatives are stored and used during gradient descent to update the weights. The pseudo-code for the forward pass is provided in the Appendix, C.1. Commonly used activation functions include linear, sigmoid, softmax, tanh and ReLU. A summary of each of these activation functions is provided in the Appendix, A.1.

2.2.3. Convolutional neural network

A convolutional neural network (CNN) consists of one or more convolutional layers, filters, pooling layers and fully connected layers. It is deep deep-learning technique that takes advantage of the concept of filters to perform feature extraction on images. A simple convolutional network can be seen in Figure 2.4 to show how an image is processed, convoluted, pooled and then used as input to a normal feedforward network.

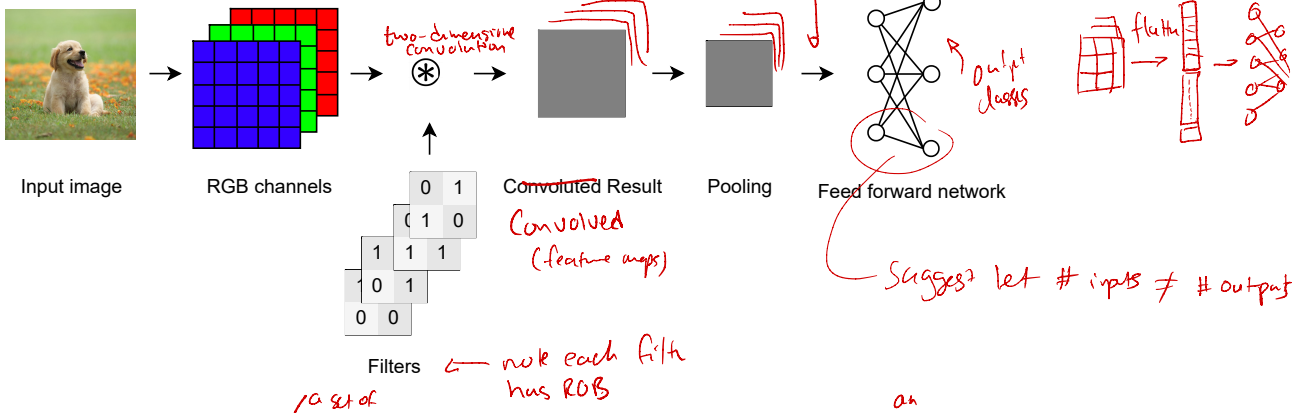
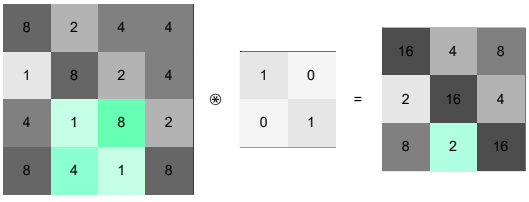


Figure 2.4: A convolutional neural network receives input in the form of images. The input image is composed of three channels: red, green and blue (RGB). These channels are then convolved with the filters to perform feature detection. Pooling is used to reduce the dimension of the resulting feature maps. Finally, the output of the convolutional network is fed to a fully connected feed-forward network.

A convolutional layer uses one or more filters for feature detection. Figure 2.5 shows an input matrix is convolved with a filter. The filter passes over the entire input matrix, performing an element-wise multiplication between the input pixels I and the filter pixels F, storing the sum of the result in the corresponding element in the feature map R.



$$R[n_i, n_j] = \sum_{i=n_i}^{n_i+F_{height}} \sum_{j=n_j}^{n_j+F_{width}} I[i,j]F[i-n_i, j-n_j]$$

Figure 2.5: Convolution explained using a diagonal line as a filter. The filter moves over the entire input image, each element in the input image is multiplied element-wise with the corresponding element in the feature map, and the total sum is stored as the output layer.

Mathematically, convolution is implemented using cross-correlation:

$$r_{I,F}[n_i, n_j] = \sum_{i=n_i}^{n_i+F_{height}} \sum_{j=n_j}^{n_j+F_{width}} I[i,j] \times F[i-n_i, j-n_j] \tag{2.7}$$

where a filter image F is passed over an image I to produce a smaller image with dimensions $(I_{width} - F_{width} + 1, I_{height} - F_{height} + 1)$. For example, if $n_i = 3, n_j = 2, F_{height} = 2, F_{width} = 2$ as shown in turquoise in the figure, then:

$$\begin{aligned} r_{I,F}[3,2] &= \textstyle\sum_{i=3}^{5} \sum_{j=2}^{4} I[i,j]F[i-3, j-2] \\ r_{I,F}[3,2] &= I[3,2]F[0,0] + I[3,3]F[0,1] + I[4,2]F[1,0] + I[4,3]F[1,1] \\ r_{I,F}[3,2] &= 1*1 + 8*0 + 4*0 + 1*1 \\ r_{I,F}[3,2] &= 2 \end{aligned} \tag{2.8}$$

Figure 2.6 shows a convolutional layer in more detail. It consists of an input image I with the respective RGB channels. The RGB channels are convolved with each filter F, to form convolved images R. The convolved images are then passed to an activation function similar to CNNs, resulting in a convoluted output matrix V. This entire process is referred to as a convoltional layer.

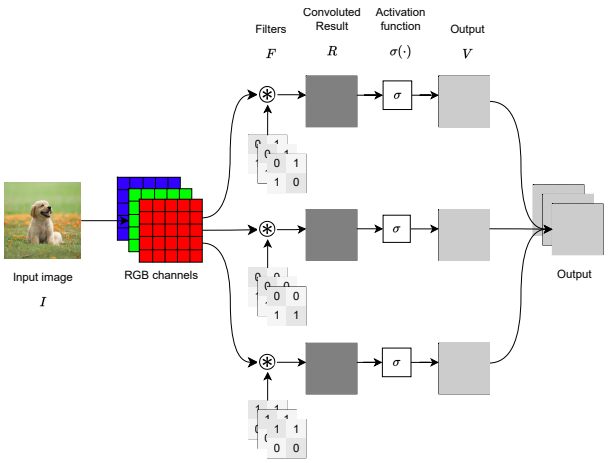


Figure 2.6

A pooling layer is used to reduce the dimension of the feature maps. Similar to the convolutional layer, pooling takes a mask with fixed dimensions and aggregates the image values that fall within the mask. Max-pooling obtains the maximum value in the feature map, while average-pooling obtains the average. The fully connected layers are as described above, where the output of the convolutional layers and pooling layers are used as input to the fully connected layers.

2.2.4. ResNet

A ResNet is a convolutional neural network that aims to solve the problem of vanishing gradients which plagues deep architectures by introducing Residual Blocks. A Residual Block consists of two convolutional layers with a skip connection as shown in Figure 2.7. The skip connection allows the model to bypass some of the convolutional layers during training, allowing for deeper models without influencing the performance.

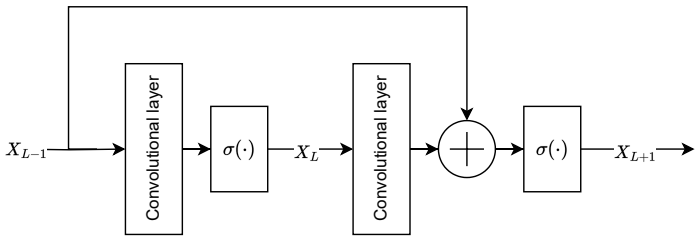


Figure 2.7: A resnet convolutional block showing the skip connection.

A Residual network consists of several of these Residual blocks in series. Figure 2.8 shows the general layout of a ResNet18 model (18 refers to the total number of connected layers). The model starts with a 7×7 convolutional layer followed by a max pooling layer. It then consists of four resnet convolutional blocks that are each repeated twice (for different ResNet architectures, the number of repeated blocks differs), where each convolutional block consists of two convolutional layers with the skip implemented as described above. The model ends with an average pooling layer and a fully connected layer.

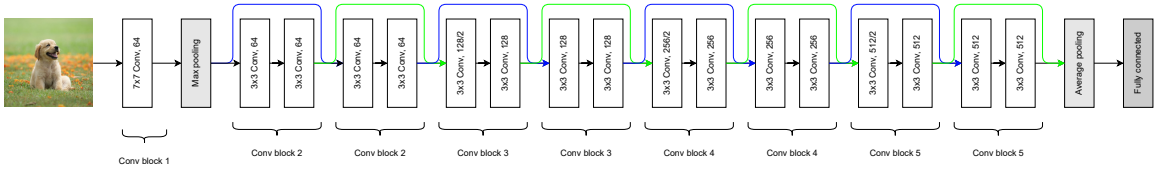


Figure 2.8: ResNet18 model consisting of a convolutional layer, a max pooling layer, four convolutional blocks (repeated twice), an average pooling layer and a fully connected layer.

2.2.5. Siamese network

A Siamese network is a neural network that consists of two sub-networks, which share the same weights. Figure 2.9 shows the layout of a basic Siamese network where the two sub-networks are shown as parallel branches. Typically, the input for this model will be a high dimensional representation, such as a mel spectrogram image, while the features generated will be a lower dimensional representation. Each sub-network consists of an encoder $f(\cdot)$, which returns a representation of the input, followed by a projection head $g(\cdot)$ which returns the features $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$. The encoder model typically consists of a ResNet model to encode the input images into representations, while the projection head uses a multi-layer perceptron to generate features. These features can then be used as input to another model, for example a classifier. A contrastive loss function is used to maximise the agreement between the two input samples.

Explain how x_i and x_j are generated.

explain why they should agree.

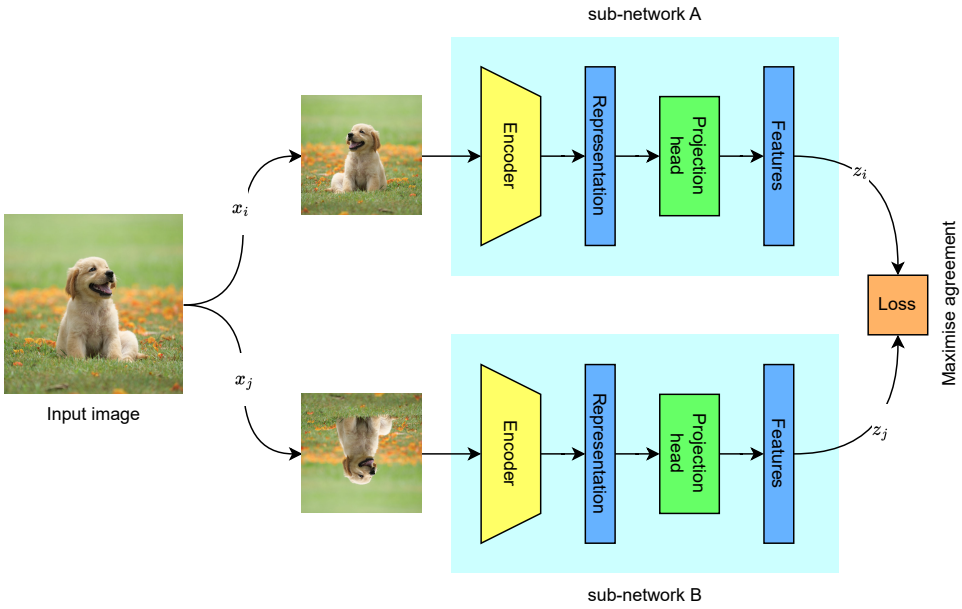


Figure 2.9: A general Siamese network takes in an image $\boldsymbol{x}$, creates a copy of the original input image $\boldsymbol{x_i}$ and an augmentation of the original image $\boldsymbol{x_j}$. Each sub-network consists of an encoder $f(\cdot)$, and a projection head $g(\cdot)$ which generates features $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$. The loss function uses the features generated to maximise the agreement between the two sub-networks.

Gradient collapse: Gradient collapse occurs when the gradients of the loss function used in Siamese networks become very small. This is a common phenomenon in contrastive learning where loss functions are used to minimise the distance between similar images. Each of the architectures discussed below has a unique way of ensuring that the architecture does not collapse.