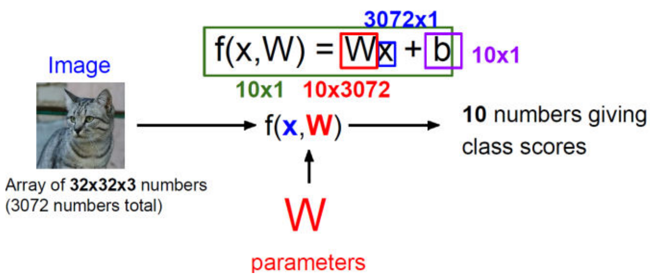
Assignment 4a. **(due on 20/02/2022)**

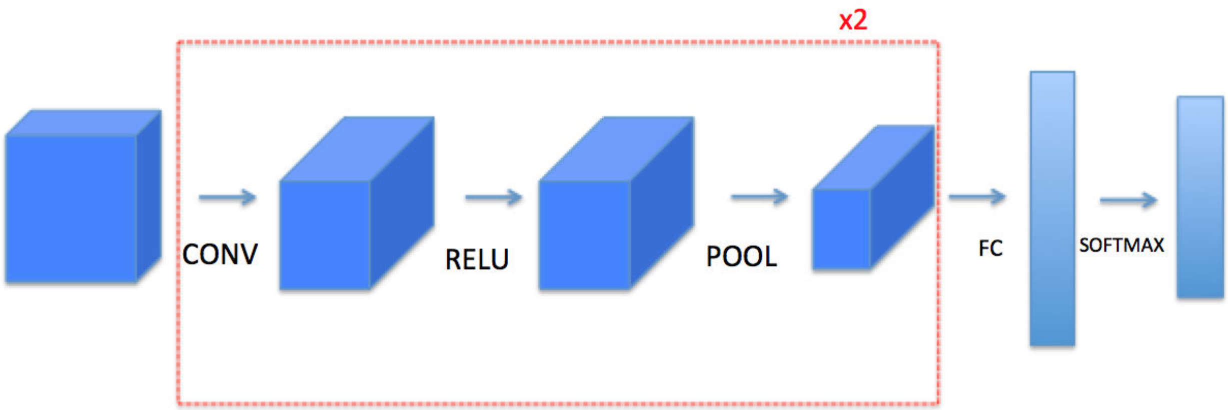
- Download and read train data from CIFAR 10 from 10 classes
- Use only one layer of neural network. Make the layer fully linear.
- **class_score** = W*X + b. Where W is the weight matrix, **X** is 3072 length vector. So b is also a 10 length vector for 10 classes. class_score is also a 10-vector. See lecture 2 page 32



- Don't use any sigmoid function in the neural network.
- Normalize the input image by dividing 255 each pixel value.
- Write forward(X) function to implement the forward calculation.
- Calculate the loss using the SVM_LOSS
- Using the calculated loss, use autograd and backward() to calculate the derivatives automatically.
- Write codes for a training module for 50 epochs to train the neural network.
- Report your accuracy on the test set
- Show confusion matrix of your test prediction
- After training, show the image for each row of the **W** to see the corresponding class pattern**.** Provide the title with the corresponding class name as shown in lecture 2 page 38.
- You can show each row as an image as follows.

Assignment 4b. **Due 27/02/2022** (Forward pass: 50% + Backward pass 50%)

1. Now we will design a simple convolutional neural network for classification. Use CIFAR 10 dataset with 10 classes.
2. You need to write codes to implement the following architecture.



3. Following function should be written to implement the following blocks.

```
def zero_pad(X, pad):
    """
    Argument:
        X -- python numpy array of shape (n_H, n_W, n_C) representing one  image
      pad -- integer, amount of padding around each image on vertical and horizontal
      dimensions

    Returns:
    X_pad -- padded image of shape (m, n_H + 2*pad, n_W + 2*pad, n_C)

def conv_single_step(a_slice_prev, W, b):
    """
    Apply one filter defined by parameters W on a single slice (a_slice_prev) of the output activation
     of the previous layer.

    Arguments:
    a_slice_prev -- slice of input data of shape (f, f, n_C_prev)
    W -- Weight parameters contained in a window - matrix of shape (f,
     f, n_C_prev)
    b -- Bias parameters contained in a window - matrix of shape (1, 1, 1)

    Returns:
      Z -- a scalar value, result of convolving the sliding window (W,  on a slice x of the input
      data
```

```python
    """
def conv_forward(A_prev, W, b, hparameters):
    """
    Implements the forward propagation for a convolution function

    Arguments:
    A_prev -- output activations of the previous layer, numpy array of shape (m, n_H_prev,
    n_W_prev, n_C_prev)
    W -- Weights, numpy array of shape (f, f, n_C_prev, n_C)
    b -- Biases, numpy array of shape (1, 1, 1, n_C)
    hparameters -- python dictionary containing "stride" and "pad"
    Returns:
    Z -- conv output, numpy array of shape (m, n_H, n_W, n_C)
    mem -- cache of values needed for the conv_backward() function
    """
def pool_forward(A_prev, hparameters):
    """
    Implements the forward pass of the pooling layer

    Arguments:
    A_prev -- Input data, numpy array of shape (m, n_H_prev, n_W_prev, n_C_prev)
    hparameters -- python dictionary containing "f" and "stride"

    Returns:
    A -- output of the pool layer, a numpy array of shape (m, n_H, n_W, n_C)
    mem -- cache used in the backward pass of the pooling layer, contains the input and
    hparameters
    """
def conv_backward(dZ, mem):
    """
    Implement the backward propagation for a convolution function

    Arguments:
```

```
    dZ -- gradient of the cost with respect to the output of the conv layer (Z), numpy array of
   shape (m, n_H, n_W, n_C)
    mem -- cache of values needed for the conv_backward(), output of conv_forward()

    Returns:
    dA_prev -- gradient of the cost with respect to the input of the conv layer (A_prev),
               numpy array of shape (m, n_H_prev, n_W_prev, n_C_prev)
    dW -- gradient of the cost with respect to the weights of the conv layer (W)
          numpy array of shape (f, f, n_C_prev, n_C)
    db -- gradient of the cost with respect to the biases of the conv layer (b)
          numpy array of shape (1, 1, 1, n_C)
    """

def create_mask_from_window(x):
    """
    Creates a mask from an input matrix x, to identify the max entry of x.

    Arguments:
    x -- Array of shape (f, f)

    Returns:
    mask -- Array of the same shape as window, contains a True at the position corresponding
    to the max entry of x.
    """
def pool_backward(dA, mem):
    """
    Implements the backward pass of the pooling layer

    Arguments:
    dA -- gradient of cost with respect to the output of the pooling layer, same shape as A
    mem -- cache output from the forward pass of the pooling layer, contains the layer's input
    and hparameters
    mode -- the pooling mode you would like to use, defined as a string ("max" or "average")
```

Returns:

dA_prev -- gradient of cost with respect to the input of the pooling layer, same shape as A_prev

"""