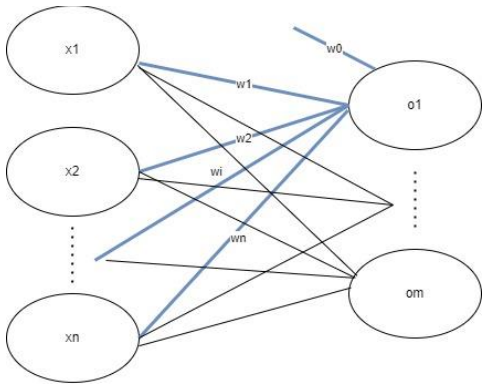
Learning hypotheses

ANN

In the case of ANN, there are several layers connected, and each layer is composed of several units. The idea is that each layer is connected with the previous layer, except the input layer. Each unit can be regarded as a perceptron in the sense that it is connected to all units from the previous layer. Each connection is actually a weight, and it can be visually represented as follows:

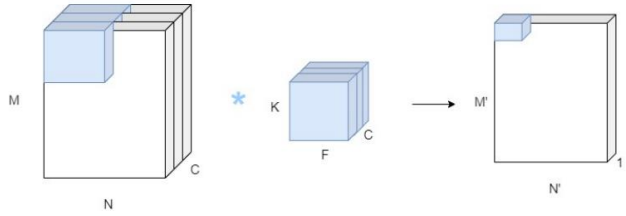


The output o1 is computed as w0 + w1*x1 + w2*x2 + … + wi*xi + … + wn*xn. An activation function should be applied on this value in order to introduce nonlinearities, otherwise everything can be reduced to a single polynomial. Also, w0 is usually called bias.

Therefore, all weights for a given layer can be stored in a (n+1) x m matrix, and this can repeat for several layers. A layer could also be seen as a stack of perceptrons.

CNN

The structure of the CNN is like the one of ANN, but instead of the perceptron as the building block, the convolution is the equivalent of the unit. If the weights of the perceptron represent the coefficients of a polynomial, here the weights represent a volume as follows:



The volume in the middle represent the actual weights of the convolution. They are usually called kernel. The depth of the kernel must be the same with the depth of the input volume. The idea is that the kernel is applied in a sliding window approach. As in the image, the kernel is overlapped on the input volume and the result is mapped to the same position in the output feature map. By applying the kernel, we mean that there is a pointwise multiplication, and the results are summed. As it was the case in ANN, a bias is added, and an activation function is applied. The depth of the output feature map is 1, therefore, in order to obtain

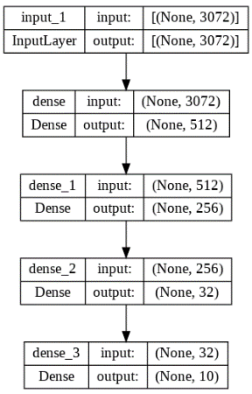
an output volume of depth C', C' kernels are stacked in a convolutional layer, in order to compute C' feature maps that form the output volume.

Details about the ML models/architectures

We use 2 types of architectures, both having around 1.7 million parameters in order to have a fair comparison.

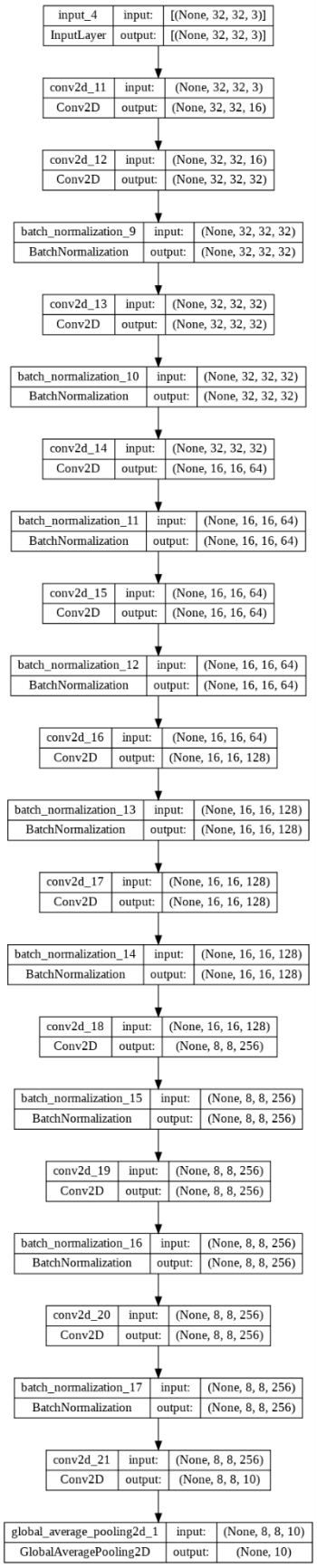
ANN

For the first type of model, we use an ANN with 4 dense layers, as depicted in the image bellow. All dense layers have ReLU activation. The model has 1,713,258 trainable parameters.



CNN

For the second architecture, we use a CNN with 21 layers, out of which 9 are batch normalization layers, one is a global average pooling and 11 convolutional layers with a total of 1,773,962 parameters, out of which 1,771,530 are trainable, and 2,432 are non-trainable.


input_4 input: [(None, 32, 32, 3)]
InputLayer output: [(None, 32, 32, 3)]
conv2d_11 input: (None, 32, 32, 3)
Conv2D output: (None, 32, 32, 16)
conv2d_12 input: (None, 32, 32, 16)
Conv2D output: (None, 32, 32, 32)
batch_normalization_9 input: (None, 32, 32, 32)
BatchNormalization output: (None, 32, 32, 32)
conv2d_13 input: (None, 32, 32, 32)
Conv2D output: (None, 32, 32, 32)
batch_normalization_10 input: (None, 32, 32, 32)
BatchNormalization output: (None, 32, 32, 32)
conv2d_14 input: (None, 32, 32, 32)
Conv2D output: (None, 16, 16, 64)
batch_normalization_11 input: (None, 16, 16, 64)
BatchNormalization output: (None, 16, 16, 64)
conv2d_15 input: (None, 16, 16, 64)
Conv2D output: (None, 16, 16, 64)
batch_normalization_12 input: (None, 16, 16, 64)
BatchNormalization output: (None, 16, 16, 64)
conv2d_16 input: (None, 16, 16, 64)
Conv2D output: (None, 16, 16, 128)
batch_normalization_13 input: (None, 16, 16, 128)
BatchNormalization output: (None, 16, 16, 128)
conv2d_17 input: (None, 16, 16, 128)
Conv2D output: (None, 16, 16, 128)
batch_normalization_14 input: (None, 16, 16, 128)
BatchNormalization output: (None, 16, 16, 128)
conv2d_18 input: (None, 16, 16, 128)
Conv2D output: (None, 8, 8, 256)
batch_normalization_15 input: (None, 8, 8, 256)
BatchNormalization output: (None, 8, 8, 256)
conv2d_19 input: (None, 8, 8, 256)
Conv2D output: (None, 8, 8, 256)
batch_normalization_16 input: (None, 8, 8, 256)
BatchNormalization output: (None, 8, 8, 256)
conv2d_20 input: (None, 8, 8, 256)
Conv2D output: (None, 8, 8, 256)
batch_normalization_17 input: (None, 8, 8, 256)
BatchNormalization output: (None, 8, 8, 256)
conv2d_21 input: (None, 8, 8, 256)
Conv2D output: (None, 8, 8, 10)
global_average_pooling2d_1 input: (None, 8, 8, 10)
GlobalAveragePooling2D output: (None, 10)

We use SparseCategoricalCrossentropy loss from Keras. (https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy).

This is suited for multi-class classification, and it receives the ground truth labels. We also directly pass the logits for numerical stability.

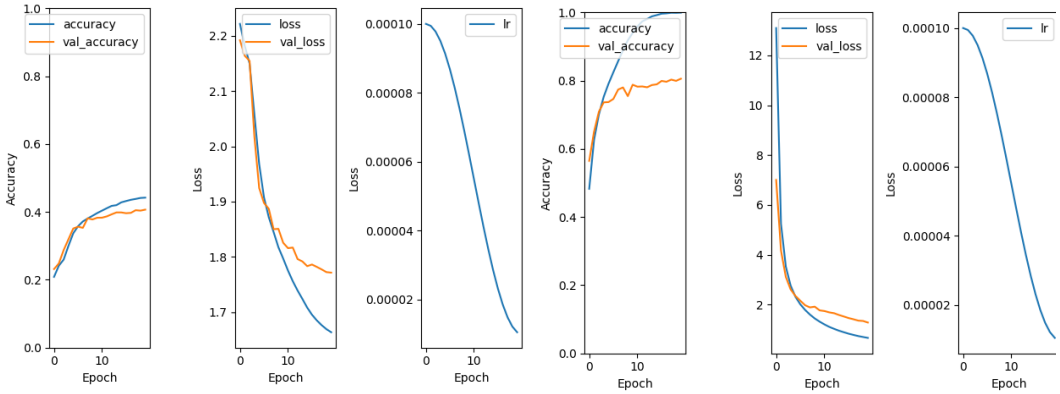
Mathematically speaking, the following formula is implemented behind the scenes:

$$L = -\sum_{c \in classes} p_{gt}^{c} log(p_{pred}^{c})$$

The sparse labels are turned into one-hot encodings, therefore p_{gt}^{c} will be 1 if c is the true class, 0 otherwise and p_{pred}^{c} represents the predicted probability for class c which is computed from the logits.

In our case c takes values from 0 to 9, because there are 10 classes.

The models are trained with a batch size of 32, for 20 epochs with a CosineAnnealing learning rate schedule from 1e-4 to 1e-5.



ANN training CNN training

Experimental results

We use accuracy, precision, recall, fscore and AUC in order to compare the 2 models.

We use cross-validation for evaluation on 6 splits. For precision, recall, fscore and AUC we perform a mean over the classes, since they have equal number of images in the dataset.

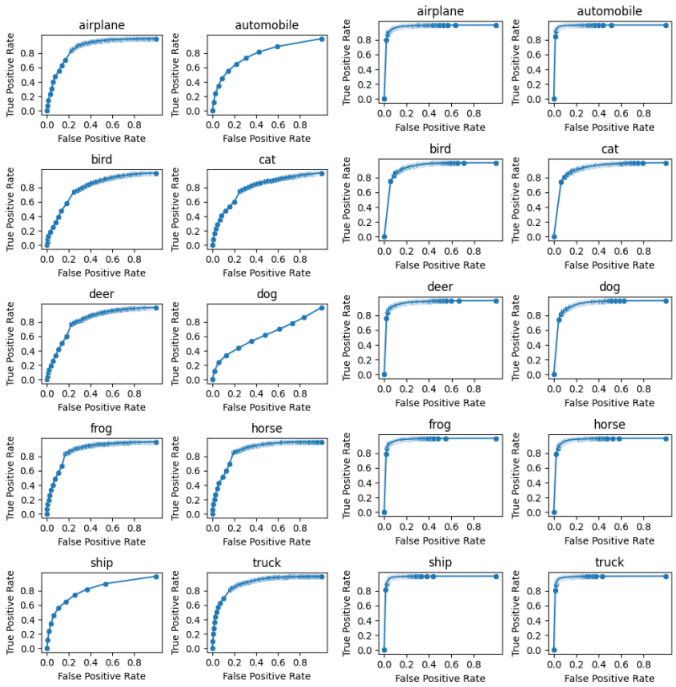
	ANN		CNN	
	mean	CI interval	mean	CI interval
Accuracy	0.48	0.04	0.79	0.005
Precision	0.43	0.08	0.79	0.005
Recall	0.48	0.04	0.79	0.005
Fscore	0.43	0.08	0.79	0.005

AUC	0.91	0.004	0.95	0.0006

Comparison

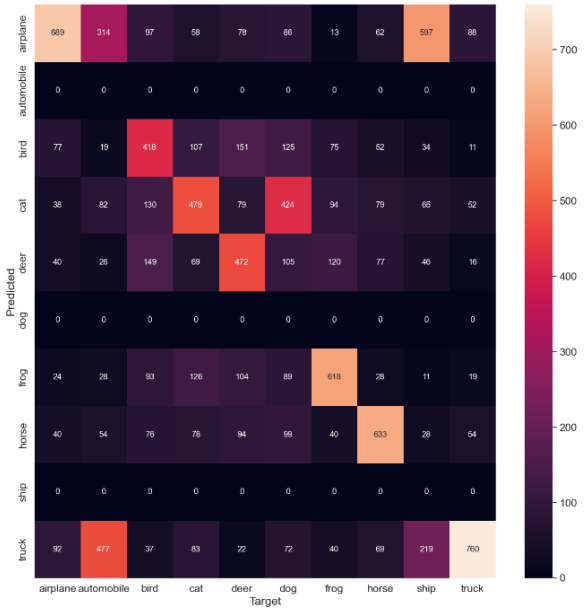
We can see from the table the CNN has better values for each metrics, and from the CI interval we can tell that it is more stable. Both models have the same number of parameters but, the CNN model is much deeper, with 21 layers, as opposed to the ANN with 4 layers. This is due to the superiority of deep learning.

In what follows we give some examples of ROC curves and confusion matrixes.

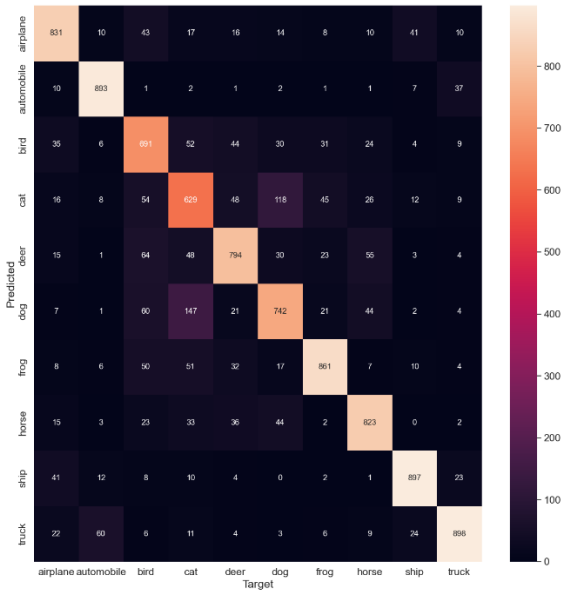


ANN CNN

We can also see from the curves that the CNN performs better.



ANN



CNN

Also, the confusion matrix for the ANN is much weaker, simply because there are rows with only with 0 values, meaning that the model didn't even predict those classes, and the values in the CNN case are centered on the main diagonal, meaning that the model actually predicted the target class.