1. 0 **Convolutional Neural Networks (CNNs)** are analogous to traditional ANNs in that they are comprised of neurons that self-optimise through learning. Each neuron will still receive an input and perform an operation (such as a scalar product followed by a non-linear function) - the basis of countless ANNs. From the input raw image vectors to the final output of the class score, the entire of the network will still express a single perceptive score function (the weight). The last layer will contain loss functions associated with the classes, and all of the regular tips and tricks developed for traditional ANNs still apply [1]

1.1 **Overall architecture**
CNN's are comprised of three types of layers. These are convolutional layers, pooling layers and fully-connected layers. When these layers are stacked, a CNN architecture has been formed. A simplified CNN architecture for MNIST classification is illustrated in Figure 1. [1]

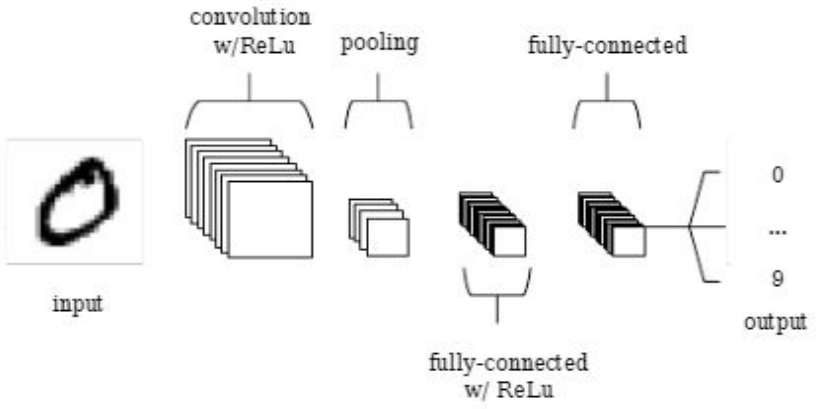


Fig. 1: An simple CNN architecture, comprised of just five layers
The basic functionality of the example CNN above can be broken down into four key areas.
1. As found in other forms of ANN, the **input layer** will hold the pixel values of the image.
2. The **convolutional layer** will determine the output of neurons of which are connected to local regions of the input through the calculation of the scalar product between their weights and the region connected to the input volume. The **rectified linear unit** (commonly shortened to ReLu) aims to apply an 'elementwise' activation function such as sigmoid to the output of the activation produced by the previous layer.
3. The **pooling layer** will then simply perform downsampling along with the spatial dimensionality of the given input, further reducing the number of parameters within that activation.
4. The **fully-connected layers** will then perform the same duties found in standard ANNs and attempt to produce class scores from the activations,

to be used for classification. It is also suggested that ReLu may be used between these layers, as to improve performance. [1]

1.2 **Dense connectivity.**

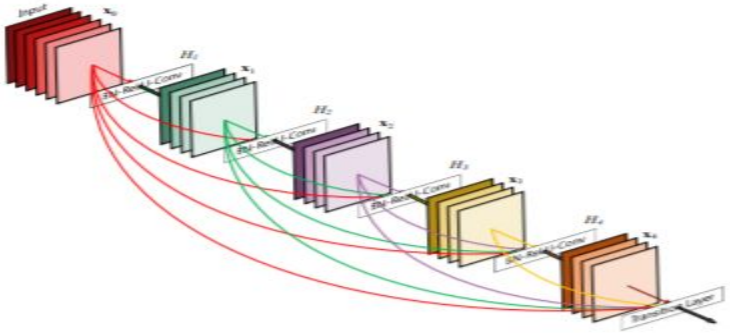


Figure 2: A **5-layer dense block** with a growth rate of k = 4. Each layer takes all preceding feature-maps as input

To further improve the information flow between layers we propose a different connectivity pattern: we introduce direct connections from any layer to all subsequent layers. Figure 2 illustrates the layout of the resulting DenseNet schematically. Consequently, the Lth layer receives the feature-maps of all preceding layers, x0, . . . , x l−1, as input:

$$x_L = H_L([x_0, x_1, \ldots, x_{L-1}]), \quad (2)$$

where [x0, x1, . . . , xL−1] refers to the concatenation of the feature-maps produced in layers 0, . . . , L−1. Because of its dense connectivity, we refer to this network architecture as Dense Convolutional Network (DenseNet). For ease of implementation, we concatenate the multiple inputs of Hl(·) in eq. (2) into a single tensor. [2]

2.1 **Experimental Results**

1. Simple Convolutional Neural Network:

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 1000)	0
embedding_1 (Embedding)	(None, 1000, 100)	672600
conv1d_1 (Conv1D)	(None, 996, 128)	64128
max_pooling1d_1 (MaxPooling1	(None, 199, 128)	0
conv1d_2 (Conv1D)	(None, 195, 128)	82048

max_pooling1d_2 (MaxPooling1 (None, 39, 128) 0

conv1d_3 (Conv1D) (None, 35, 128) 82048

max_pooling1d_3 (MaxPooling1 (None, 1, 128) 0

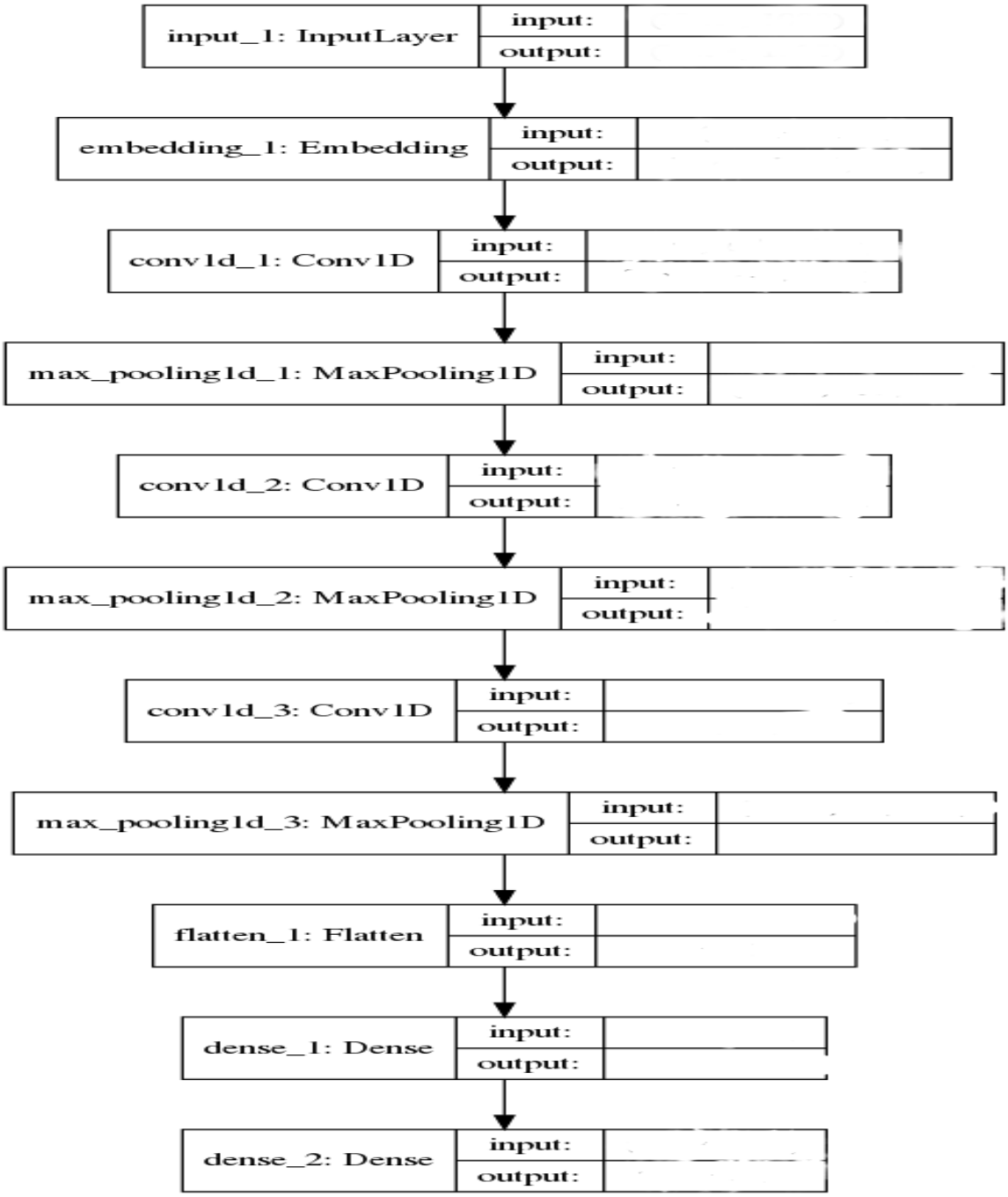
flatten_1 (Flatten) (None, 128) 0

dense_1 (Dense) (None, 128) 16512

dense_2 (Dense) (None, 2) 258

Total params: 917,594
Trainable params: 917,594
Non-trainable params: 0

As the above results show in the first model there are two dense layers with 128 and 2 output sizes and 16512 and 258 parameters. There are 3 pooling layers and 3 convolutional layer
The following diagram represents the simple convolutional neural network



2. Complex Convolutional Neural Network

Layer (type) Output Shape Param # Connected to

==========

==========

input_2 (InputLayer) (None, 1000) 0

embedding_1 (Embedding)	(None, 1000, 100)	672600	input_2[0][0]
conv1d_4 (Conv1D)	(None, 998, 128)	38528	embedding_1[1][0]
conv1d_5 (Conv1D)	(None, 997, 128)	51328	embedding_1[1][0]
conv1d_6 (Conv1D)	(None, 996, 128)	64128	embedding_1[1][0]
max_pooling1d_4 (MaxPooling1D)	(None, 199, 128)	0	conv1d_4[0][0]
max_pooling1d_5 (MaxPooling1D)	(None, 199, 128)	0	conv1d_5[0][0]
max_pooling1d_6 (MaxPooling1D)	(None, 199, 128)	0	conv1d_6[0][0]
concatenate_1 (Concatenate)	(None, 597, 128)	0	max_pooling1d_4[0][0] max_pooling1d_5[0][0] max_pooling1d_6[0][0]
conv1d_7 (Conv1D)	(None, 593, 128)	82048	concatenate_1[0][0]
max_pooling1d_7 (MaxPooling1D)	(None, 118, 128)	0	conv1d_7[0][0]
conv1d_8 (Conv1D)	(None, 114, 128)	82048	max_pooling1d_7[0][0]
max_pooling1d_8 (MaxPooling1D)	(None, 3, 128)	0	conv1d_8[0][0]
flatten_2 (Flatten)	(None, 384)	0	max_pooling1d_8[0][0]
dense_3 (Dense)	(None, 128)	49280	flatten_2[0][0]

_____
_____

dense_4 (Dense) (None, 2) 258 dense_3[0][0]

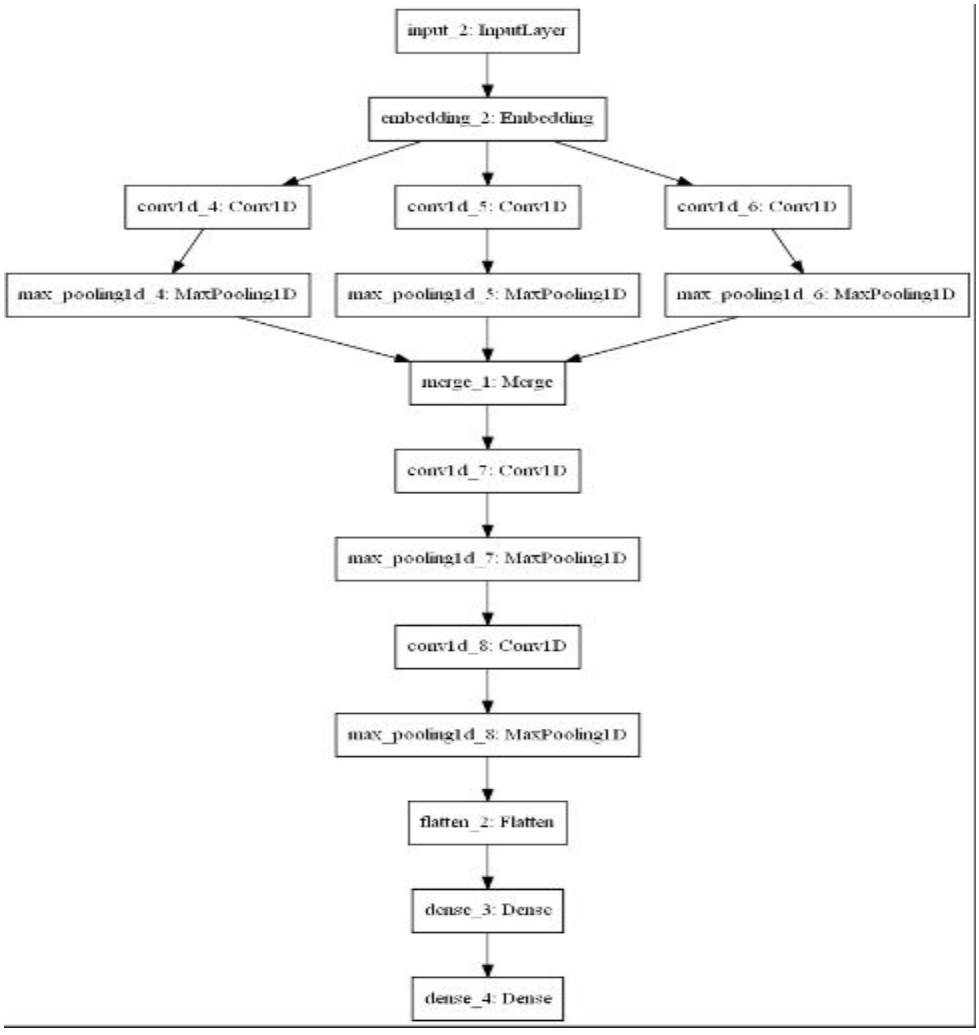
==========

==========

Total params: 1,040,218
Trainable params: 1,040,218
Non-trainable params: 0

_____

It has 4 dense layers , 8 convolutional and 8 max pooling layer
Following is the architecture



3.Long Short-term memory

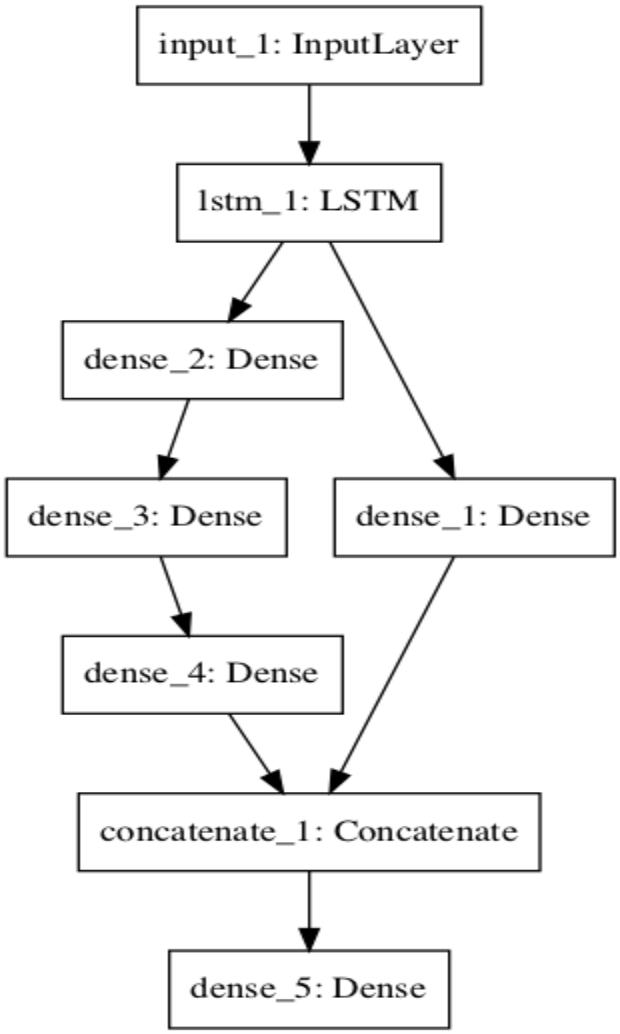
_____

Layer (type) Output Shape Param #

embedding_1 (Embedding)	(None, 1000, 100)	672600
dropout_1 (Dropout)	(None, 1000, 100)	0
conv1d_1 (Conv1D)	(None, 1000, 32)	16032
max_pooling1d_1 (MaxPooling1	(None, 500, 32)	0
conv1d_2 (Conv1D)	(None, 500, 64)	6208
max_pooling1d_2 (MaxPooling1	(None, 250, 64)	0
lstm_1 (LSTM)	(None, 100)	66000
batch_normalization_1 (Batch	(None, 100)	400
dense_1 (Dense)	(None, 256)	25856
dense_2 (Dense)	(None, 128)	32896
dense_3 (Dense)	(None, 64)	8256
dense_4 (Dense)	(None, 2)	130

Total params: 828,378
Trainable params: 828,178
Non-trainable params: 200

Following is the model architecture



3.0 **Datasets**

1. election dataset

SPLIT	TRAIN	VALIDATION	TEST
SIZE	1061	664	663
REAL NEWS	955	597	594
FAKE NEWS	106	67	69

	fake_news	tweet_id	retweet_c	user_frien	user_follo	user_favo	geo_coord	num_hash	num_ment	num_urls	num_media
count	136	1327	1327	1327	1327	1327	1327	1327	1327	1327	1327
mean	2.117647	7.79E+17	3633.335	8983.63	3509883	11724	0.83572	0.83572	1.945742	0.411454	0.232102
std	1.63326	5.60E+16	5305.272	33033.45	11817859	28691.64	1.180596	1.180596	1.021671	0.516212	0.422333
min	1	2.64E+17	1002	0	16	0	0	0	1	0	0
25%	1	7.89E+17	1370	254.5	31862	215	0	0	1	0	0
50%	1	7.95E+17	2090	779	194830	1563	0	0	2	0	0
75%	4	7.96E+17	3732	3326	926521.5	7190	1	1	2	1	0
max	5	7.96E+17	79092	578811	93936531	335194	9	9	11	2	1

DESCRIPTION OF ELECTION DATASET

2. LIAR

SPLIT	TRAIN	VALIDATION	TEST
SIZE	9727	6144	4095
FALSE	1892	1218	776
TRUE	1582	1008	668
MOSTLY-TRUE	1870	1142	820
HALF-TRUE	2000	1261	853
PANTS ON FIRE	798	503	336
BARELY TRUE	1585	1012	642

Column1	barely_true_c	false_c	half_true_c	mostly_true_c	pants_on_fire_c
count	10237	10237	10237	10237	10237
mean	11.53433623	13.28768194	17.13539123	16.43586988	6.202012308
std	18.97434865	24.11380828	35.84786188	36.15308885	16.12959871
min	0	0	0	0	0
25%	0	0	0	0	0
50%	2	2	3	3	1
75%	12	12	13	11	5
max	70	114	160	163	105

DESCRIPTION OF THE LIARDATASET

References

[1] O'Shea, K., & Nash, R. (2015, December 02). An Introduction to Convolutional Neural Networks. Retrieved June 24, 2020, from https://arxiv.org/abs/1511.08458v2

[2]G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 2261-2269, DOI: 10.1109/CVPR.2017.243.

[3]