רשתות נוירונים לתמונות | תרגיל 1

שם: תמוז גיטלר | ת`ז: 312243439

2 באפריל 2023

Practical Questions:

Question 1

A) train and test losses

Both of the models are using the same parameters:

1. epochs: 30
2. batch size: 32
3. lr: 0.01

except for the number of filters that changes between the two models.

The green line represent a Model using 3 filters and it has 3172 parameters.

The orange line represent a Model using 20 filters and it has 46590 paramete.

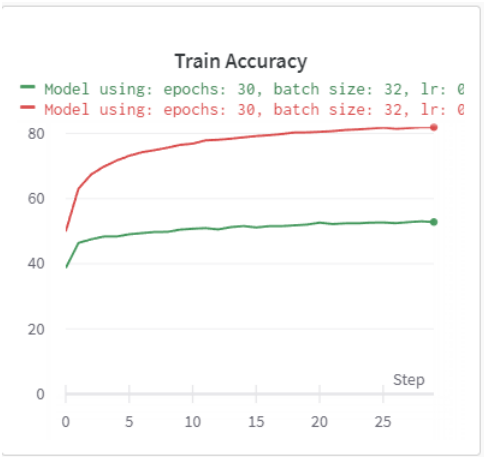
we can notice that the orange model is getting better (smaller) loss than the green model over the train set. but when we look on the test set, we see that the blue model overfitts the training set because the test loss starts rissing.

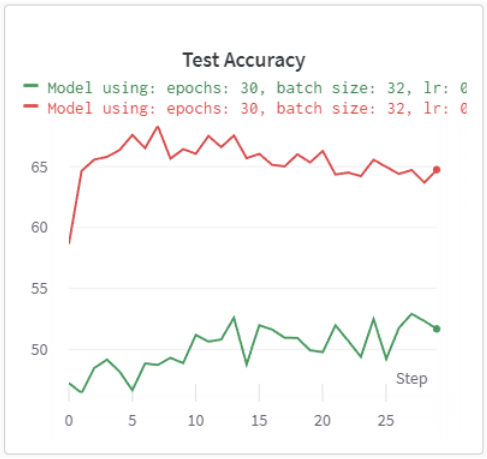
another way of seeing the overfitt that accours is by looking on the train accuracy of the orange model which is around 82% on the train set but on the test set the accuracy is only around 52%.

The green model train accuracy after 30 epochs is around 53% and the test accuracy is around 52%, this is because the model underfitts the data, by increasing the number of filters the network will be able to learn the data better.

Train and Test losses:

Train and Test Accuracys:





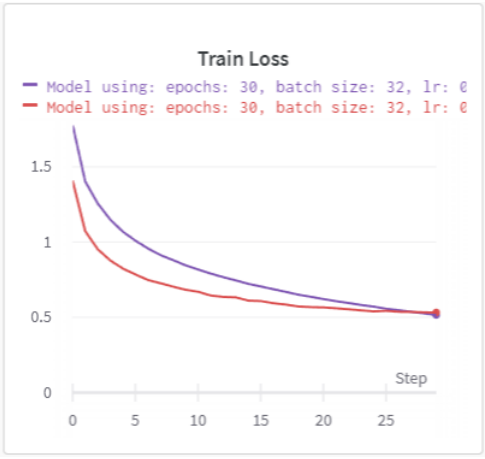
B) Configuration resulting lowest test error

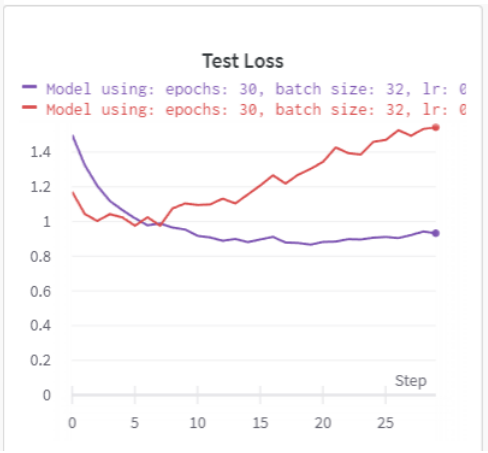
Finding the configuration that results the lowest test error is hard task, i tried several paremetrs and this is the best model i found

The configruation i found is a Model (purple line) using: epochs: 20, batch size: 32, lr: 0.001 and 20 filters the number of parameters the model uses is 46590.

C) Configuration resulting lowest test error

For comparison i added the orange model which uses the same parameters but lr of 0.01 instead of 0.001. Altough both of the models have simillar training loss this model generalize better and doesnt overfits which results lower test loss (purple line)





The models accuracy on the test set was around 70%

Question 2

Non linear CNN	linear CNN

`def forward(self, x):`
`  x = self.pool(F.relu(self.conv1(x)))`
`  x = self.pool(F.relu(self.conv2(x)))`
`  x = torch.flatten(x, 1) # flatten all dimensions except batch`
`  x = self.fc1(x)`
`  return x`
	`def forward(self, x):`
	`  x = self.conv1(x)`
	`  x = self.conv2(x)`
	`  x = torch.flatten(x, 1)`
	`  x = self.fc1(x)`
	`  return x`

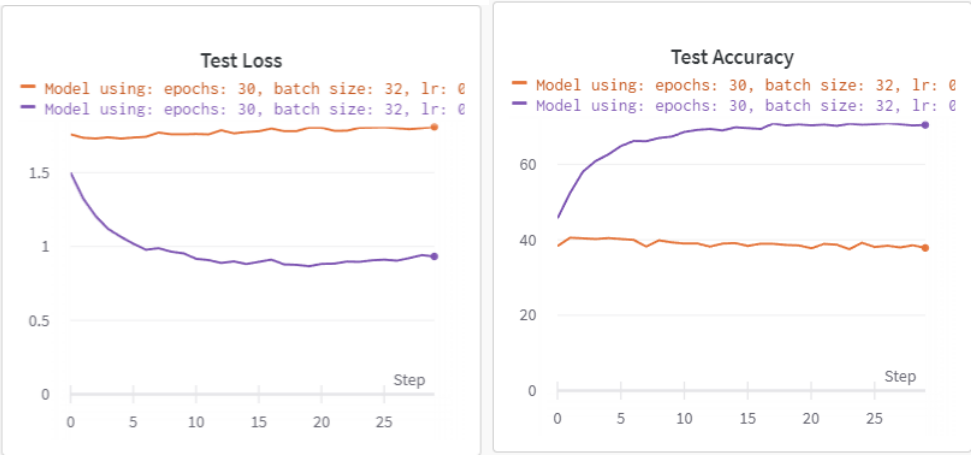
The non linear component of my CNN is the activation function called ReLU which maps negative values to zero and leaves positive values unchanged and the pooling layer which was MaxPool2d, I used both of them in lines 2,3 (after convolution).

Without non-linear activation functions and pooling, the CNN would not be able to learn complex, non-linear patterns in the data and would result in a significant loss of performance.

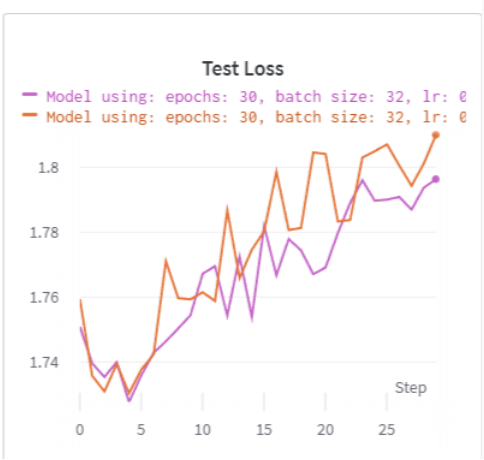
In the graph below im using the same model:

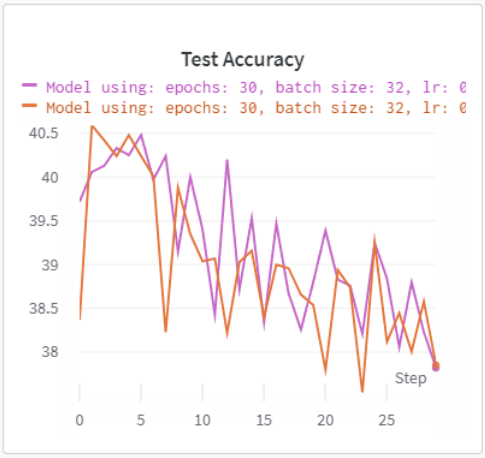
epochs: 20, batch size: 32, lr: 0.001 and 20 filters.

We can see that the same model without the non linear layers (the orange line) got higher loss (worse performance) hence less accuracy (37% instead of 70%).



I tried to compare two networks but with different number of filters 6 or 20

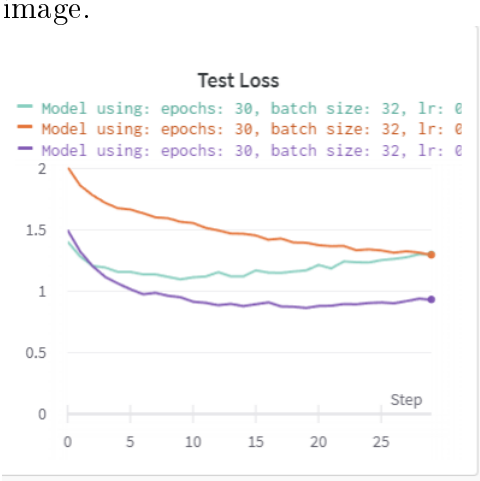


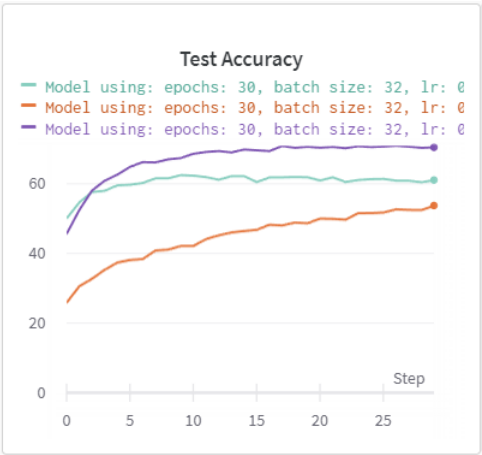


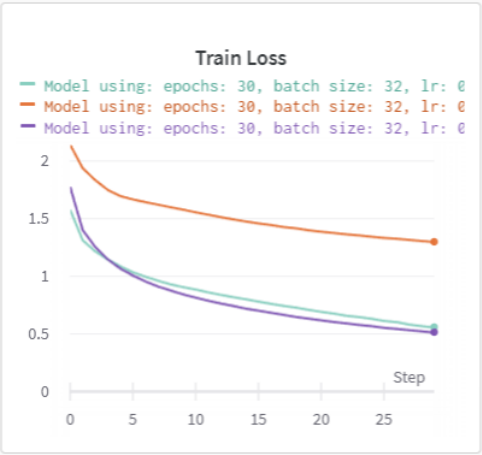
as you can see using more filters (the orange line) didnt improve the results, as you can see both of the models had 37% accuracy.

Question 3

In general:
The receptive field of a CNN refers to the region in the input image that influences the output of a particular neuron in the network. A larger receptive field means that each neuron in the network is sensitive to a larger region of the input image, which can help the network capture more global and abstract features of the input image.







In the following models i used the same hyper parameters: epochs: 30, batch size: 32, lr: 0.001, filter size: 20
purple line - the model from previous question
light blue - model (i) that uses a FC layer on the activations of the first conv layer (relu without pooling).
orange line - model (ii) that uses a “global average pooling” operator and then applys an FC layer.
when comperring between the two models from this section (3) i deduce that the light blue model which uses FC on the first conv layer performs better than the blue model.

Both of the networks didnt perfom better than the one in part 1 (purple line).
My assumption is that the purple model performed better than the orange line is because it uses MaxPool2d which helps to reduce the spatial dimensions of the input while retaining the most important features.
The purple model performed better than the light blue line because the purple model is more complex - it uses 2 conv layers in comparission to the light blue model which uses only 1 conv layer without pooling.
altough the light blue and the purple model got simillar Train loss, the purple model has lower Test loss which indicates that he generalizes better.

Theoretical Questions:

Question 1

Given linear operator L
and the following condition: $L[x(i+k)](j) = L([x(i)](j+k)$
i will show that $L[x(i+k)](j)$ corresponds to a convolution:
$L[x(i+k)](j) \underbrace{=}_{linearity\ of\ L} L[\Sigma_n x(n)\delta(i+k-n)](j) = \Sigma_n L[x(n)\delta(i+k-n)](j) \underbrace{=}_{LTI} \Sigma_n L[x(n)\delta(i-n)](j+k) =$
$= \Sigma_n x(n) L[\delta(i-n)](j+k) \overset{*}{=} (x*g)(j+k)$
denote $L[\delta(t)] = g(t)$, we get that $\Sigma_n x(n) L[\delta(i-n)]$ is a convolution between x and g

Question 2

When reshaping a 2D activation map (resulting from a convolutional layer) and feeding it into an FC layer the order of the reshaping **doesnt matter** because the resulting 1D vector will be multiplied by a fully connected weight matrix, when reshaping in different order the learned weight matrix will learn the same weights but they will be located in different locations in the matrix. but the result of the multipied vector with the matrix will be the same.

Question 3

LTI - refers to the ability of the network to recognize the same pattern or feature regardless of its position in the input image, which means that if a pattern appears in one part of the image, the same pattern translated to another part of the image should be recognized in the same way by the network.

A. The ReLU activation function:

No, because ReLU isnt linear operator, and LTI mean that the operator has to be Linear operator.
I will show that the second rule of linear operator doesnt apply:
f(cx)=cf(x) for all x and all constants c
for example:
$\underbrace{ReLU(-3 \cdot 5)}_{f(cx)}$=ReLu(-15)=0!=-15=-3·5=$\underbrace{-3 \cdot ReLU(5)}_{c \cdot f(x)}$
we got that $f(cx)! = c \cdot f(x)$

B. strided pooling layer:

No, A strided pooling layer is not LTI because it subsamples the input and discards information about the precise location of features in the input. This means that if an input feature is shifted by a small amount, the output of the strided pooling layer may change.

C. The addition of a bias:

No, because bias addition isnt linear operator, and LTI mean that the operator has to be Linear operator.
No, I will show that the first rule of linear operator doesnt apply:
1. f(x+y)=f(x)+f(y) for all x and y
for example, given f(x)=x+$\underbrace{b}_{bias}$

$$f(x) + f(y) = x + b + y + b = x + y + 2b$$

$$f(x + y) = x + y + b$$

we got that $f(x) + f(y)! = f(x + y)$ thus its not linear operator.

D. Multiplication with a fully-connected matrix:

The fully connected matrix is a weight matrix, when shifting the flatten vector and multiplying it with the same matrix we will get different results, which means that the Multiplication with the weight matrix is not TI.
example: where $e_{1-shifted}$ is equal to e_1 but shifted

$$A \cdot e_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$A \cdot e_{1-shifted} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

we got that $A \cdot e_1! = A \cdot e_2$ thus its not LT!