

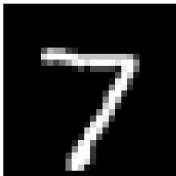

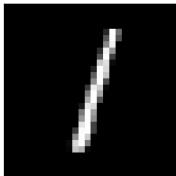
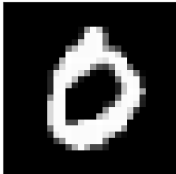
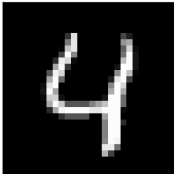
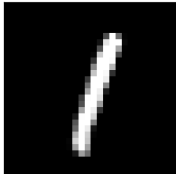
Project 5: Recognition using Deep Networks

Partner : Husain Khozema Gittham

Overview

This tasks begins the journey into deep learning and finally using python. It helps us to learn how to build, train, analyze and make changes to a neural network. We work on MNIST and Greek symbol dataset. In the greek dataset, instead of making the neural network classify we removed the last layer and used the output of the layer before that as features and ran own classification models on that like 1-NN, KNN.

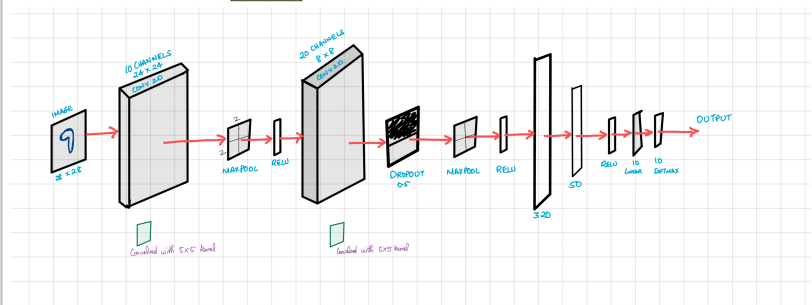
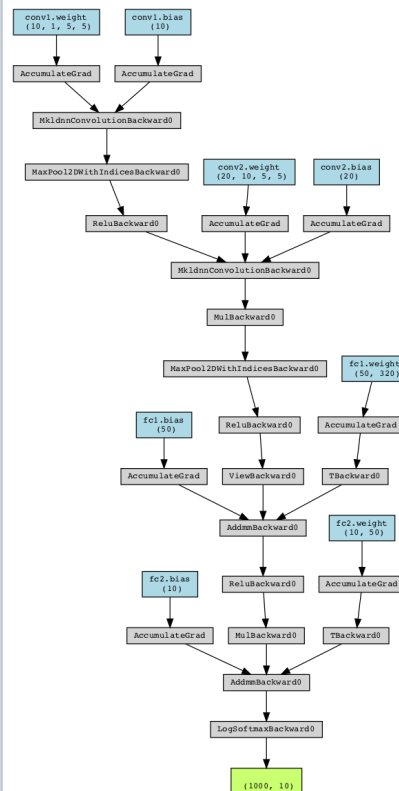
Task, respective explanation and results are attached in the table below

Problem		Result
Task 1: Build and train a network to recognize digits a)Get the MNIST digit data set	Downloaded the dataset and the first 6 images with the respective ground truth are	<div>Ground Truth 7</div>  <div>Ground Truth 2</div>  <div>Ground Truth 1</div>  <div>Ground Truth 0</div>  <div>Ground Truth 4</div>  <div>Ground Truth 1</div> 
b)Make your network code repeatable	Random seed set as 42 for pytorch package and disabled cudnn <code>torch.backends.cudnn.enabled = False</code>	

c) Build a network model

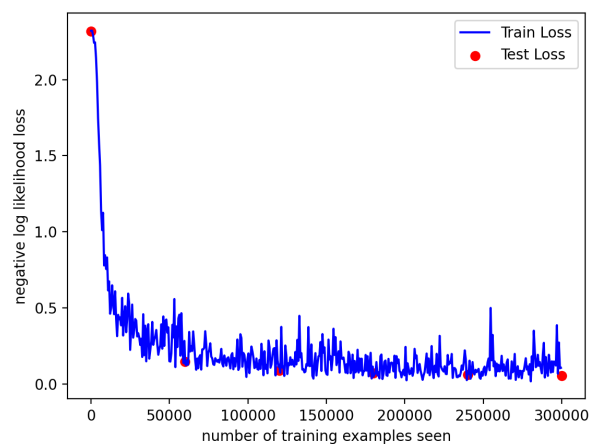
Used torchviz to visualize the neural network. Created a function, given a model and the data to the model, and creates a png file for the model.

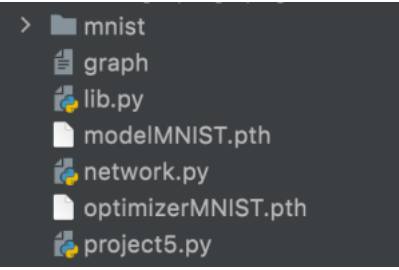




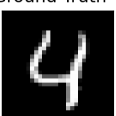
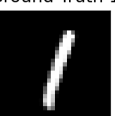













- **Input** : 28*28 1 channel(Gray Scale) image.
- A **convolution** layer with 10 5x5 filters. Produces 10*24*24 output.
- A **max-pooling** layer with a 2x2 window and a ReLU function was applied. Produces 10*12*12 output.
- A **convolution** layer with 20 5x5 filters. Produces 20*8*8 output.
- A **dropout** layer with a 0.5 dropout rate (50%)
- A **max-pooling** layer with a 2x2 window and a ReLU function applied. Produces 20*4*4 output.
- A flattening operation followed by a **fully connected Linear** layer that takes 320 inputs and converts to 50 output.
- A **fully connected Linear** layer with 10 nodes and the log_softmax function applied to the output. Produces 10 outputs.

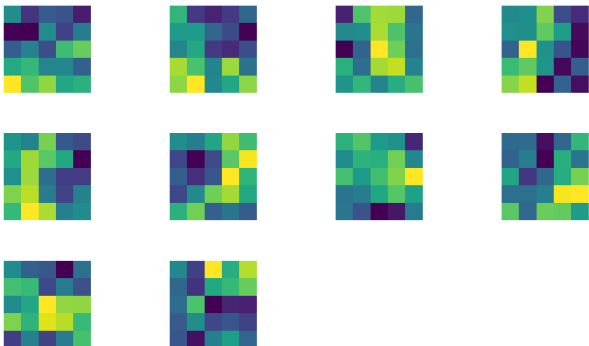
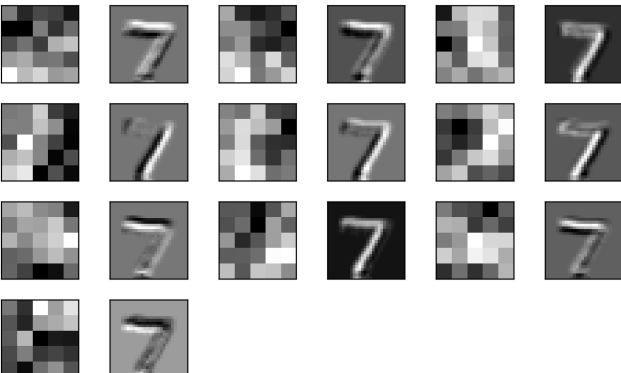


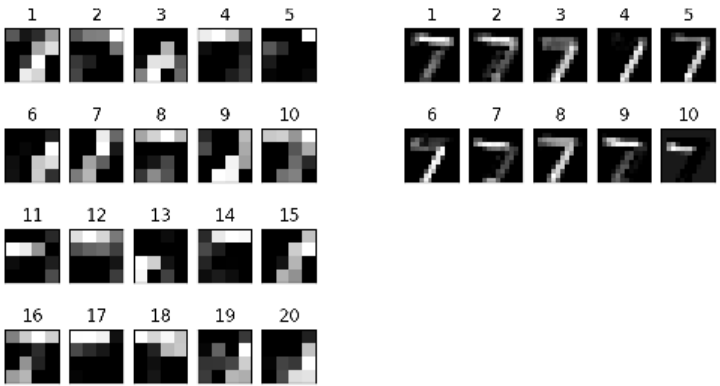












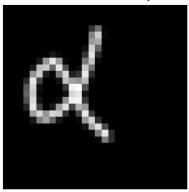
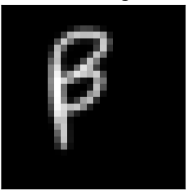
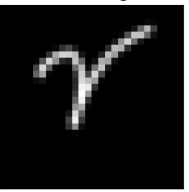
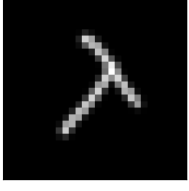
d) Train the model

Setting the batch_size of the training model as 64, and running the model for 5 epochs.


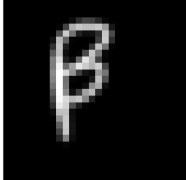
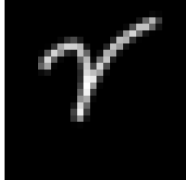
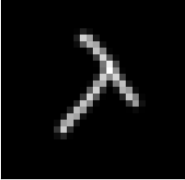

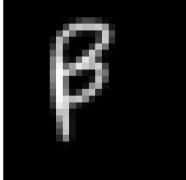
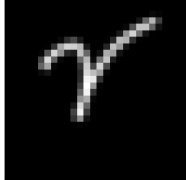
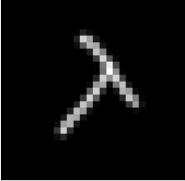


<p>e)Save the network to a file</p>	<p>Saving the network with the datasetName appended to it along with the optimizer.</p>	
<p>f)Read the network and run it on the test set</p>	<p>Loaded the saved model and given 10 images from the MNIST test set. Showing the comparison results and the classification in the results.</p>	<pre>Image 1 and values are [-18.14, -10.58, -12.34, -6.49, -9.4, -8.46, -20.83, -1.62, -11.21, -0.22] Image 2 and values are [-0.0, -29.88, -17.19, -29.36, -33.91, -15.49, -24.66, -23.37, -28.31, -23.84] Image 3 and values are [-2.62, -16.26, -11.6, -15.87, -15.94, -5.35, -0.08, -18.27, -5.96, -13.87] Image 4 and values are [-14.4, -19.49, -14.89, -13.94, -6.76, -8.9, -19.67, -7.95, -13.68, -0.0] Image 5 and values are [-0.01, -21.92, -9.63, -9.8, -14.59, -10.09, -24.77, -4.94, -17.02, -8.48] Image 6 and values are [-25.03, -0.0, -22.22, -18.23, -13.7, -15.6, -17.71, -19.5, -16.52, -19.17] Image 7 and values are [-9.56, -7.59, -10.55, -1.1, -13.47, -0.42, -9.55, -12.9, -4.69, -7.79] Image 8 and values are [-25.36, -35.74, -25.59, -11.25, -22.11, -15.99, -30.62, -21.76, -17.02, -0.0] Image 9 and values are [-40.12, -19.85, -19.24, -17.5, -29.99, -33.33, -48.48, -0.0, -22.49, -13.45] Image 10 and values are [-5.09, -9.56, -7.61, -1.03, -9.68, -0.7, -7.62, -8.33, -2.48, -2.89]</pre> <div> <div>Ground Truth 7 </div> <div>Ground Truth 2 </div> <div>Ground Truth 1 </div> <div>Ground Truth 0 </div> <div>Ground Truth 4 </div> <div>Ground Truth 1 </div> <div>Ground Truth 4 </div> <div>Ground Truth 9 </div> <div>Ground Truth 5 </div> </div>
<p>g)Test the network on new inputs</p>	<p>Runnin the model on hand-written digits. Created a pipeline to scale down the images to 28 x 28 and then intensities are inverted to make the background black and the digit white. The digits are predicted correctly except for 9, as 9 and 4 are almost close, I think having more data train the model might do the trick to classify that correctly as well.</p>	<div> <div>Ground Truth 0 </div> <div>Ground Truth 1 </div> <div>Ground Truth 2 </div> <div>Ground Truth 3 </div> <div>Ground Truth 4 </div> <div>Ground Truth 5 </div> <div>Ground Truth 6 </div> <div>Ground Truth 7 </div> <div>Ground Truth 8 </div> <div>Ground Truth 4 </div> </div>

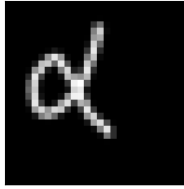
Task 2: Examine your network	<p>This task is to analyze the network we created. The results of printing the model is shown in the image. We can see the layers and the</p>	<pre> ----- Layer (type) Output Shape Param # ----- Conv2d-1 [-1, 10, 24, 24] 260 MaxPool2d-2 [-1, 10, 12, 12] 0 Conv2d-3 [-1, 20, 8, 8] 5,020 Dropout2d-4 [-1, 20, 8, 8] 0 MaxPool2d-5 [-1, 20, 4, 4] 0 Linear-6 [-1, 50] 10,050 Linear-7 [-1, 10] 510 ----- Total params: 21,840 Trainable params: 21,840 Non-trainable params: 0 ----- Input size (MB): 0.00 Forward/backward pass size (MB): 0.08 Params size (MB): 0.08 Estimated Total Size (MB): 0.16 ----- </pre>
a) Analyze the first layer	<p>Analyzing the filters(weights) of the first layer.</p>	
b) Show the effect of the filters	<p>Showing the effect of the filter analyzed above on the input data. We can see how digit 7 features are getting extracted. All the filters find the edges of the digit 7. The learning are random other than that, we can predict what would get learned, apart from the random guess than one of the features would help in identify edges like the sobels.</p>	

<p>c) Build a truncated model</p> <p>Extension 1</p>	<p>Created a subModel that has only the convolution layers of the original model. Weights aren't changed. We run the digit on the model and the result of them is shown in the first image. The 20 output channels of size 4x4 are shown.</p> <p>Truncated even the last layer and just kept one convolution layer, the output after running the classifier looks like the digit. The second photo belongs to this experiment</p>	
<p>Task 3: Create a digit embedding space</p>	<p>The task to remove the classifier layer and use features to classify the data tried adding two more greek symbols as part of an extension. Found discrepancies with tau symbol so removed it.</p> <p>Loading all the 36 images reduced the size of the greek symbol in the output, so showing only the first 12 symbols from the dataset.</p>	<div>Ground Truth alpha</div>  <div>Ground Truth alpha</div>  <div>Ground Truth alpha</div>  <div>Ground Truth alpha</div>  <div>Ground Truth alpha</div>  <div>Ground Truth alpha</div>  <div>Ground Truth alpha</div>  <div>Ground Truth alpha</div>  <div>Ground Truth alpha</div>  <div>Ground Truth beta</div>  <div>Ground Truth beta</div>  <div>Ground Truth beta</div> 
<p>a) Create a greek symbol data set</p>		<div>Ground Truth alpha</div>  <div>Ground Truth gamma</div>  <div>Ground Truth gamma</div>  <div>Ground Truth lambda</div> 

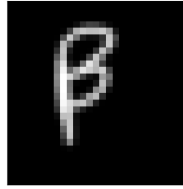
b)Create a truncated model	Showing the truncated model summary. The last layer, linear layer 6 shows the 50 output features coming from the network. These features are used later in the classification. Not storing the data in CSV storing it in the NumPy and using it.	<div><div>Layer (type)</div><div>Output Shape</div><div>Param #</div><div>=====</div><div>Conv2d-1</div><div>[-1, 10, 24, 24]</div><div>260</div><div>MaxPool2d-2</div><div>[-1, 10, 12, 12]</div><div>0</div><div>Conv2d-3</div><div>[-1, 20, 8, 8]</div><div>5,020</div><div>Dropout2d-4</div><div>[-1, 20, 8, 8]</div><div>0</div><div>MaxPool2d-5</div><div>[-1, 20, 4, 4]</div><div>0</div><div>Linear-6</div><div>[-1, 50]</div><div>16,050</div><div>=====</div><div>Total params: 21,330</div><div>Trainable params: 21,330</div><div>Non-trainable params: 0</div><div>-----</div><div>Input size (MB): 0.00</div><div>Forward/backward pass size (MB): 0.08</div><div>Params size (MB): 0.08</div><div>Estimated Total Size (MB): 0.16</div><div>-----</div></div>																																																																																																																																																				
c)Project the greek symbols into the embedding space	After running the model I have printed two examples of the 50 features for the alpha data. As there are 36 images for 4 greek symbols in our dataset, we would have 36*50 in total.	<div>1.805658849872998</div> <div>alpha</div> <div>tensor([-2.4106, -3.5967, -3.8531, -4.4440, -4.1268, -4.4440, -4.0881, -4.4440, -4.3093, -4.4440, -3.5944, -4.4440, -3.2512, -4.1154, -4.4440, -4.4440, -4.4440, -4.4440, -3.7500, -4.4440, -4.3757, -4.4440, -4.4440, -3.9352, -3.6825, -3.9228, -4.0671, -4.4440, -3.8978, -3.1199, -4.4440, -3.7789, -4.1664, -4.4440, -4.4440, -4.4440, -4.4440, -4.3400, -2.8685, -3.6130, -4.1263, -4.4440, -3.8831, -4.4440, -3.4378, -4.1781, -3.4421, -4.4440, -4.4440, -3.1999])</div> <div>0.8738411788799567</div> <div>alpha</div> <div>tensor([-2.6441, -3.4781, -4.0837, -4.5123, -4.2477, -4.5123, -4.0285, -4.2505, -4.1880, -4.5123, -3.2358, -4.5123, -3.0574, -4.0402, -4.5123, -4.5123, -4.5123, -4.5123, -3.4688, -4.5123, -4.0616, -4.5123, -4.5123, -3.6201, -3.5980, -3.6111, -4.4488, -4.5123, -3.6863, -3.0987, -4.5123, -3.5426, -4.1100, -4.0581, -4.5123, -4.3430, -4.5123, -4.0712, -3.1541, -3.5766, -4.1201, -4.5123, -4.1132, -4.5123, -3.6499, -4.5123, -3.8854, -4.5123, -4.5123, -3.1606])</div> <div>0.8738411788799567</div>																																																																																																																																																				
d)Compute distances in the embedding space	<p>Each greek symbol has 9 images. I have chosen the first image of each data set and calculated the SSD with respective to the rest of the images.</p> <p>The results are shown in the table on the right. The 0 SSD means we are trying to check the similarity with the same image. The header contains the least SSD among all excluding the same image.</p> <p>And we see that the SSD does a great job of identifying the correct label. That is 1NN seems to correctly classify the data. From the results we can also see that SSD (embeddings) is small when we are comparing from the same label. This means that KNN would work fine on this feature set.</p>	<table><tr><th>alpha (minSSD = 1.805658849872998)</th><th>beta (minSSD = 4.670998375611816)</th><th>gamma (minSSD = 1.6098189069416549)</th><th>lambda (minSSD = 64.91319096237203)</th></tr><tr><td>0.0</td><td>11.236256690695882</td><td>21.995351256791764</td><td>402.51968818577006</td></tr><tr><td>1.805658849872998</td><td>11.47205650160322</td><td>22.513688463834114</td><td>396.0073854699731</td></tr><tr><td>2.4588373585720547</td><td>12.93466503918171</td><td>20.318982689845143</td><td>407.4146935399622</td></tr><tr><td>6.45632137094799</td><td>11.41099588247016</td><td>18.763069452601485</td><td>416.7278552860953</td></tr><tr><td>11.360003832032817</td><td>14.806953816500936</td><td>6.351755948100617</td><td>426.54096222948283</td></tr><tr><td>4.504696165313362</td><td>14.716532756712695</td><td>14.26649178072148</td><td>414.8391750892624</td></tr><tr><td>5.899012240523007</td><td>14.873318413317065</td><td>18.415051099087577</td><td>403.97826842812356</td></tr><tr><td>3.2580893796985038</td><td>12.14097534220491</td><td>19.1499283055673</td><td>407.41841120203026</td></tr><tr><td>9.078647209224073</td><td>11.125744118772673</td><td>10.887375799618894</td><td>420.6762026026845</td></tr><tr><td>11.236256690695882</td><td>0.0</td><td>23.688047671399545</td><td>431.8344639944844</td></tr><tr><td>6.809145918814465</td><td>4.670998375611816</td><td>13.182425879178481</td><td>448.42275839298964</td></tr><tr><td>6.333121653362468</td><td>10.7179507301189</td><td>10.868441826912203</td><td>450.4535449008108</td></tr><tr><td>9.098301642225124</td><td>6.372945376031566</td><td>13.048754442228528</td><td>455.6987127959728</td></tr><tr><td>8.749914928688668</td><td>9.366699800841161</td><td>10.059022002504207</td><td>454.28180168403924</td></tr><tr><td>12.496342966682278</td><td>8.882956395682413</td><td>9.496843854729377</td><td>440.43290220573545</td></tr><tr><td>7.7509892585803755</td><td>7.569052855396876</td><td>11.452615171527214</td><td>438.8845079764724</td></tr><tr><td>10.646225764474366</td><td>10.26736030151369</td><td>9.536075360469113</td><td>448.71357809659094</td></tr><tr><td>6.077138318636571</td><td>8.053844882932026</td><td>11.855754105529059</td><td>445.14786748588085</td></tr><tr><td>21.995351256791764</td><td>23.688047671399545</td><td>0.0</td><td>432.3385785203427</td></tr><tr><td>23.570115624170285</td><td>26.907494563783985</td><td>2.0126119078627767</td><td>404.4644305827096</td></tr><tr><td>23.658429443719797</td><td>25.324691236950457</td><td>5.467963041965049</td><td>424.9898411356844</td></tr><tr><td>25.124794737261254</td><td>25.11058753694124</td><td>2.1122782317925157</td><td>429.225414602668</td></tr><tr><td>22.89047098332958</td><td>24.181782582920277</td><td>1.9509480225970037</td><td>427.74432673864067</td></tr><tr><td>21.252577806822956</td><td>16.839230951241916</td><td>6.25070136784052</td><td>447.31021362915635</td></tr><tr><td>23.357975811130018</td><td>26.14275644067675</td><td>1.6098189069416549</td><td>420.38076901063323</td></tr><tr><td>22.712654474540614</td><td>22.888569886097685</td><td>1.9948925194694311</td><td>434.05661666207016</td></tr><tr><td>20.679413510561062</td><td>22.93894039254519</td><td>1.6202049362373145</td><td>432.9539323132485</td></tr><tr><td>402.51968818577006</td><td>431.8344639944844</td><td>432.3385785203427</td><td>0.0</td></tr><tr><td>455.83282892405987</td><td>478.7090436052531</td><td>481.1236851782014</td><td>157.9397682135459</td></tr><tr><td>623.8891986645758</td><td>658.5111301436846</td><td>619.4588970149634</td><td>100.27985375247226</td></tr><tr><td>404.8207163121551</td><td>430.92807664209977</td><td>399.21444849751424</td><td>64.91319096237203</td></tr><tr><td>360.7531091426499</td><td>386.0194849893451</td><td>372.6534407986328</td><td>72.70007505360991</td></tr><tr><td>395.1390306212786</td><td>417.3877822555369</td><td>438.70889930124395</td><td>152.07210906036198</td></tr><tr><td>186.9617327718006</td><td>198.02926676743664</td><td>215.26200896780938</td><td>106.95834543555975</td></tr><tr><td>499.4871678310301</td><td>516.0016871783882</td><td>514.2681448012599</td><td>154.0449943691492</td></tr><tr><td>183.4170772060752</td><td>207.19765086375992</td><td>205.75657873926684</td><td>148.0446965061128</td></tr></table>	alpha (minSSD = 1.805658849872998)	beta (minSSD = 4.670998375611816)	gamma (minSSD = 1.6098189069416549)	lambda (minSSD = 64.91319096237203)	0.0	11.236256690695882	21.995351256791764	402.51968818577006	1.805658849872998	11.47205650160322	22.513688463834114	396.0073854699731	2.4588373585720547	12.93466503918171	20.318982689845143	407.4146935399622	6.45632137094799	11.41099588247016	18.763069452601485	416.7278552860953	11.360003832032817	14.806953816500936	6.351755948100617	426.54096222948283	4.504696165313362	14.716532756712695	14.26649178072148	414.8391750892624	5.899012240523007	14.873318413317065	18.415051099087577	403.97826842812356	3.2580893796985038	12.14097534220491	19.1499283055673	407.41841120203026	9.078647209224073	11.125744118772673	10.887375799618894	420.6762026026845	11.236256690695882	0.0	23.688047671399545	431.8344639944844	6.809145918814465	4.670998375611816	13.182425879178481	448.42275839298964	6.333121653362468	10.7179507301189	10.868441826912203	450.4535449008108	9.098301642225124	6.372945376031566	13.048754442228528	455.6987127959728	8.749914928688668	9.366699800841161	10.059022002504207	454.28180168403924	12.496342966682278	8.882956395682413	9.496843854729377	440.43290220573545	7.7509892585803755	7.569052855396876	11.452615171527214	438.8845079764724	10.646225764474366	10.26736030151369	9.536075360469113	448.71357809659094	6.077138318636571	8.053844882932026	11.855754105529059	445.14786748588085	21.995351256791764	23.688047671399545	0.0	432.3385785203427	23.570115624170285	26.907494563783985	2.0126119078627767	404.4644305827096	23.658429443719797	25.324691236950457	5.467963041965049	424.9898411356844	25.124794737261254	25.11058753694124	2.1122782317925157	429.225414602668	22.89047098332958	24.181782582920277	1.9509480225970037	427.74432673864067	21.252577806822956	16.839230951241916	6.25070136784052	447.31021362915635	23.357975811130018	26.14275644067675	1.6098189069416549	420.38076901063323	22.712654474540614	22.888569886097685	1.9948925194694311	434.05661666207016	20.679413510561062	22.93894039254519	1.6202049362373145	432.9539323132485	402.51968818577006	431.8344639944844	432.3385785203427	0.0	455.83282892405987	478.7090436052531	481.1236851782014	157.9397682135459	623.8891986645758	658.5111301436846	619.4588970149634	100.27985375247226	404.8207163121551	430.92807664209977	399.21444849751424	64.91319096237203	360.7531091426499	386.0194849893451	372.6534407986328	72.70007505360991	395.1390306212786	417.3877822555369	438.70889930124395	152.07210906036198	186.9617327718006	198.02926676743664	215.26200896780938	106.95834543555975	499.4871678310301	516.0016871783882	514.2681448012599	154.0449943691492	183.4170772060752	207.19765086375992	205.75657873926684	148.0446965061128
alpha (minSSD = 1.805658849872998)	beta (minSSD = 4.670998375611816)	gamma (minSSD = 1.6098189069416549)	lambda (minSSD = 64.91319096237203)																																																																																																																																																			
0.0	11.236256690695882	21.995351256791764	402.51968818577006																																																																																																																																																			
1.805658849872998	11.47205650160322	22.513688463834114	396.0073854699731																																																																																																																																																			
2.4588373585720547	12.93466503918171	20.318982689845143	407.4146935399622																																																																																																																																																			
6.45632137094799	11.41099588247016	18.763069452601485	416.7278552860953																																																																																																																																																			
11.360003832032817	14.806953816500936	6.351755948100617	426.54096222948283																																																																																																																																																			
4.504696165313362	14.716532756712695	14.26649178072148	414.8391750892624																																																																																																																																																			
5.899012240523007	14.873318413317065	18.415051099087577	403.97826842812356																																																																																																																																																			
3.2580893796985038	12.14097534220491	19.1499283055673	407.41841120203026																																																																																																																																																			
9.078647209224073	11.125744118772673	10.887375799618894	420.6762026026845																																																																																																																																																			
11.236256690695882	0.0	23.688047671399545	431.8344639944844																																																																																																																																																			
6.809145918814465	4.670998375611816	13.182425879178481	448.42275839298964																																																																																																																																																			
6.333121653362468	10.7179507301189	10.868441826912203	450.4535449008108																																																																																																																																																			
9.098301642225124	6.372945376031566	13.048754442228528	455.6987127959728																																																																																																																																																			
8.749914928688668	9.366699800841161	10.059022002504207	454.28180168403924																																																																																																																																																			
12.496342966682278	8.882956395682413	9.496843854729377	440.43290220573545																																																																																																																																																			
7.7509892585803755	7.569052855396876	11.452615171527214	438.8845079764724																																																																																																																																																			
10.646225764474366	10.26736030151369	9.536075360469113	448.71357809659094																																																																																																																																																			
6.077138318636571	8.053844882932026	11.855754105529059	445.14786748588085																																																																																																																																																			
21.995351256791764	23.688047671399545	0.0	432.3385785203427																																																																																																																																																			
23.570115624170285	26.907494563783985	2.0126119078627767	404.4644305827096																																																																																																																																																			
23.658429443719797	25.324691236950457	5.467963041965049	424.9898411356844																																																																																																																																																			
25.124794737261254	25.11058753694124	2.1122782317925157	429.225414602668																																																																																																																																																			
22.89047098332958	24.181782582920277	1.9509480225970037	427.74432673864067																																																																																																																																																			
21.252577806822956	16.839230951241916	6.25070136784052	447.31021362915635																																																																																																																																																			
23.357975811130018	26.14275644067675	1.6098189069416549	420.38076901063323																																																																																																																																																			
22.712654474540614	22.888569886097685	1.9948925194694311	434.05661666207016																																																																																																																																																			
20.679413510561062	22.93894039254519	1.6202049362373145	432.9539323132485																																																																																																																																																			
402.51968818577006	431.8344639944844	432.3385785203427	0.0																																																																																																																																																			
455.83282892405987	478.7090436052531	481.1236851782014	157.9397682135459																																																																																																																																																			
623.8891986645758	658.5111301436846	619.4588970149634	100.27985375247226																																																																																																																																																			
404.8207163121551	430.92807664209977	399.21444849751424	64.91319096237203																																																																																																																																																			
360.7531091426499	386.0194849893451	372.6534407986328	72.70007505360991																																																																																																																																																			
395.1390306212786	417.3877822555369	438.70889930124395	152.07210906036198																																																																																																																																																			
186.9617327718006	198.02926676743664	215.26200896780938	106.95834543555975																																																																																																																																																			
499.4871678310301	516.0016871783882	514.2681448012599	154.0449943691492																																																																																																																																																			
183.4170772060752	207.19765086375992	205.75657873926684	148.0446965061128																																																																																																																																																			

<p>e) Create your own greek symbol data</p>	<p>Have run 1NN on the features with the custom written symbols and below are the least SSD for each of the image.</p> <p>The calculated minimum SSD for each of the image are</p> <p>127.89654389083444 116.14244851097465 46.161773271349375 56.6454197174171</p> <p>Here the beta is wrongly classified.</p>	<div>Ground Truth alpha</div>  <div>Ground Truth gamma</div>  <div>Ground Truth gamma</div>  <div>Ground Truth lambda</div> 
<p>f) Extension 2(KNN classifier)</p> <p>Extension 3: Have added one more greek symbol</p>	<p>The KNN with K=9 almost gets the proper classification, only in the case of beta is it classified as gamma. The SSD is .00141 smaller than the gamma SSD, adding more values might help improve the system. The final SSD of all four results is given below.</p> <p>1315.489635540769 1208.1241607955308 450.96263690927395 1161.0843516781024</p> <p>The results of KNN and 1NN seems same WRT the output label. May be modification to the model or increasing the training data may help in correctly classifying it</p>	<div>Ground Truth alpha</div>  <div>Ground Truth gamma</div>  <div>Ground Truth gamma</div>  <div>Ground Truth lambda</div> 
<p>g) Extension 4: Evaluate more dimensions on task 3.</p>	<p>Tried changing the number of nodes in the fully connected instead of 50 tried increasing it to 90 and decreasing it to 30</p> <p>47.51130753464531 85.41704103164375 50.954944436321966 57.52047311607748 ['alpha', 'beta', 'gamma', 'lambda'] 476.0727699657764 921.5404625933443 505.5305667300199 790.3757818996091 ['alpha', 'beta', 'gamma', 'lambda']</p>	<p>90 layered Results</p>

Ground Truth alpha



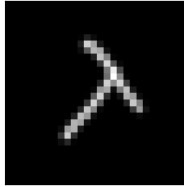
Ground Truth alpha



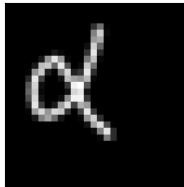
Ground Truth gamma



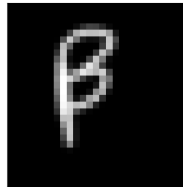
Ground Truth lambda



Ground Truth alpha



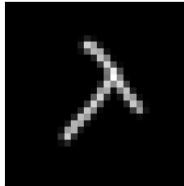
Ground Truth alpha



Ground Truth gamma

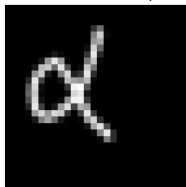


Ground Truth gamma

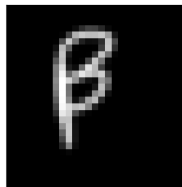


30 layerd Results

Ground Truth alpha



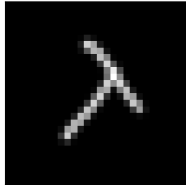
Ground Truth beta



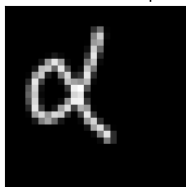
Ground Truth gamma



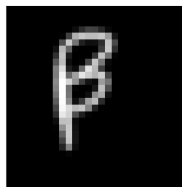
Ground Truth gamma



Ground Truth alpha



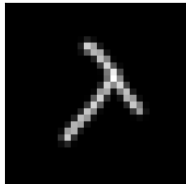
Ground Truth beta



Ground Truth gamma



Ground Truth lambda



Extension 5: There are many pre-trained networks available in the PyTorch package. Try loading one and evaluate its first couple of convolutional layers as in task 2.

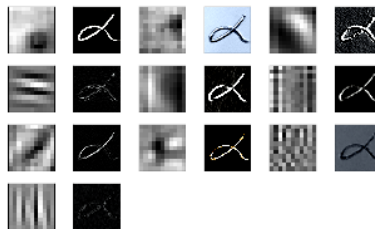
As seen from the layer results, as the layer increases the original image is almost not similar to the resultant of after that filter. Layer1 we can figure alpha more easily than Layer2.

```

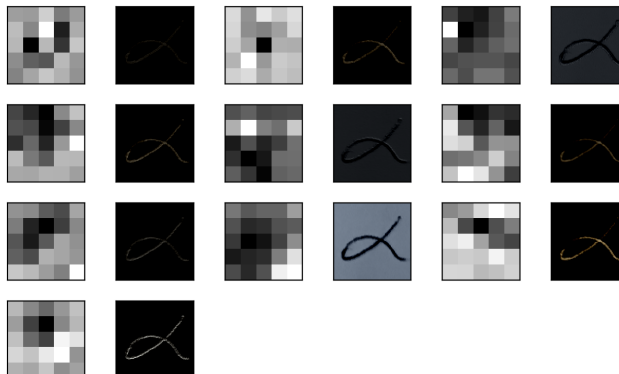
def make():
    (features): Sequential(
      (0): Conv2d(5, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64)
      (2): ReLU(inplace=True)
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (4): Conv2d(16, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (5): BatchNorm2d(128)
      (6): ReLU(inplace=True)
      (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (8): Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (9): BatchNorm2d(256)
      (10): ReLU(inplace=True)
      (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (output): Linear(4096, 1000)
    (classifier): Sequential(
      (0): Dropout(p=0.5, inplace=False)
      (1): Linear(4096, 1000)
      (2): ReLU(inplace=True)
      (3): Dropout(p=0.5, inplace=False)
      (4): Linear(1000, 1000)
      (5): BatchNorm1d(1000)
      (6): Linear(1000, 1000)
      (7): ReLU(inplace=True)
      (8): Linear(1000, 1000)
    )

```

Output of first conv2d



Output of second conv2d



Task 4: Design your own experiment

Checking the effect of batch size and the number of epochs simultaneously

Hypothesis: With a larger batch size there is more training data which results in better accuracy of the model.

Observation: The accuracy increases with increasing batch size and after certain point drops.

Hypothesis: With larger number of epochs we should get more accuracy, as we are training it multiple times, but after certain iterations it will start overfitting the data, resulting in decreased accuracy.

Observation: The accuracy of the model increases as a number of epochs increases.

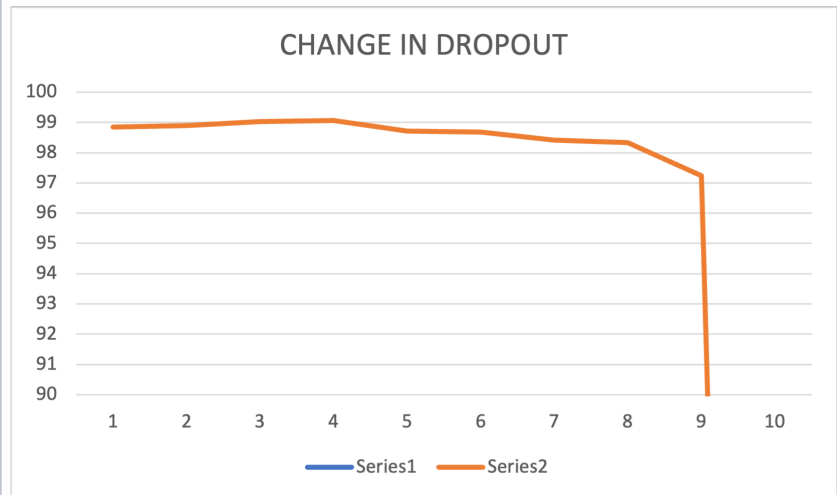
Hypothesis: The accuracy drops with an increase in the dropout probability.

Observation: We see a similar observation as the dropout increases the classification efficiency decreases and when the dropout is 1 we can see a severe drop in accuracy. On top of that we see that the accuracy increases as we move from 0.1 to 1 and after reaching some value between them it starts decreasing after that.

We have in total 60 variations of the network and see that

Number of Epochs	Batch_Train_Size	DropoutProbability	Accuracy
5	32	0.1	98.74
5	32	0.2	98.7
5	32	0.3	98.7
5	32	0.4	98.62
5	32	0.5	98.69
5	32	0.6	98.24
5	32	0.7	98.2
5	32	0.8	97.8
5	32	0.9	96.97
5	32	1	9.47
5	40	0.1	98.58
5	40	0.2	98.76
5	40	0.3	98.69
5	40	0.4	98.62
5	40	0.5	98.62
5	40	0.6	98.24
5	40	0.7	98.23
5	40	0.8	97.69
5	40	0.9	97.02
5	40	1	9.9
8	32	0.1	98.84
8	32	0.2	98.9
8	32	0.3	99.03
8	32	0.4	99.06
8	32	0.5	98.71
8	32	0.6	98.68
8	32	0.7	98.41
8	32	0.8	98.33
8	32	0.9	97.23
8	32	1	11.35
8	40	0.1	98.93
8	40	0.2	99.01
8	40	0.3	99.04
8	40	0.4	98.86
8	40	0.5	98.77
8	40	0.6	98.53
8	40	0.7	98.36
8	40	0.8	98.34
8	40	0.9	97.1
8	40	1	13.79
10	32	0.1	99
10	32	0.2	98.89
10	32	0.3	99.01
10	32	0.4	98.92
10	32	0.5	98.98
10	32	0.6	98.9
10	32	0.7	98.54
10	32	0.8	98.5
10	32	0.9	97.2
10	32	1	9.63
10	40	0.1	98.84
10	40	0.2	98.88
10	40	0.3	98.99
10	40	0.4	98.92
10	40	0.5	98.86
10	40	0.6	98.95
10	40	0.7	98.66
10	40	0.8	98.13
10	40	0.9	97.56
10	40	1	11.81

Here the Epoch is set at 8 and batch size as 32 and the variation in dropout is plotted.



Learning Outcome

This was my first time using pytorch and implementing deep neural network. It was tough assignment and I wish to do the extensions given in future as well couldn't work on it enough. Understood how the filters are getting generated and how we can visualize and see how the network generates those filters to create features that will help it to distinguish. Understood how the meta(hyper) parameters increase or decrease training time as well as accuracy.

Acknowledgement

- <https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/>
- <https://inside-machinelearning.com/en/easy-pytorch-to-load-image-and-volume-from-folder/>

- <https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/>
- <https://stackoverflow.com/questions/65617755/how-to-replicate-pytorch-normalization-in-opencv-or-numpy>
- And lot of googling quick bug fixes