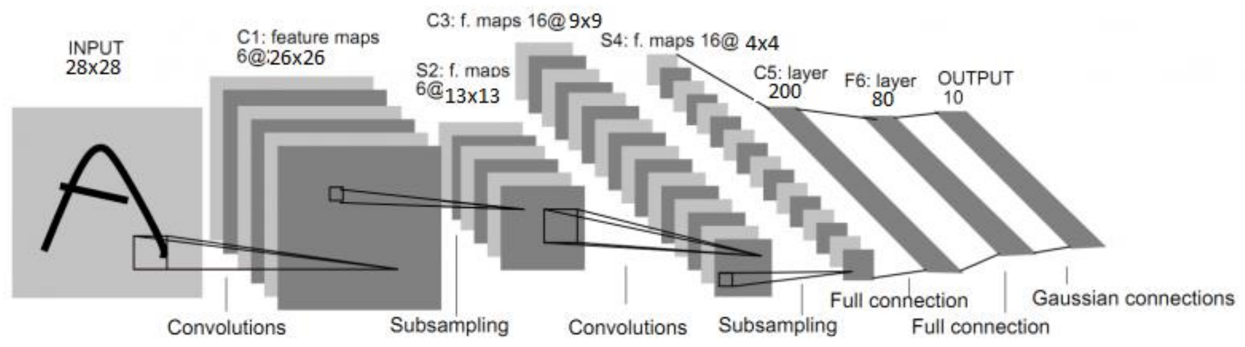
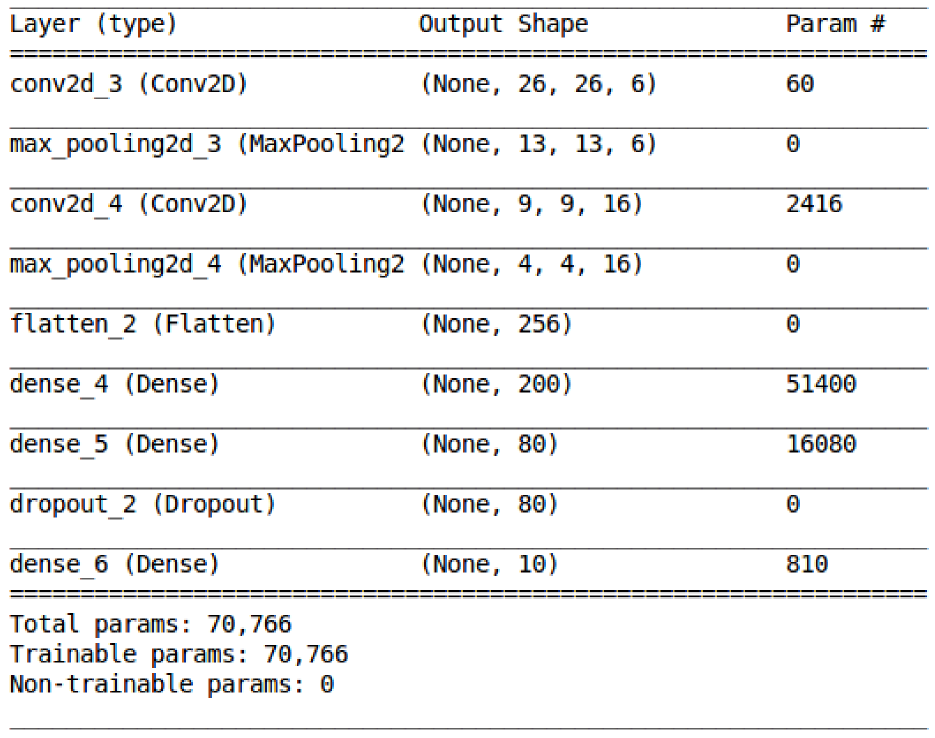
Comparing NN architectures:

#	#Epoch	Filter size	#Filte-rs	Batch size	Dropout	Regularization and Optimizer	Dense layers	Training Accuracy	Test Accuracy
1	20	5x5	6, 16	64	No	No, Adam	200, 80	99.07%	97.4%
2	20	5x5	6, 16	32	No	No, Adam	200, 80	99.5%	97.5%
3	20	5x5	6, 16	32	No	No, Adam	120, 85	98.9%	96.4%
4	30	3x3, 5x5	6, 16	32	0.5	(0.01) L2, Adam	200, 80	99.77%	98.2%
5	20	5x5	6, 16	64	No	RBF	200, 80	97.9%	96.95%
6	20	5x5	6, 16	16	No	(0.01) L2, Adam	200, 80	99.5%	97.5%

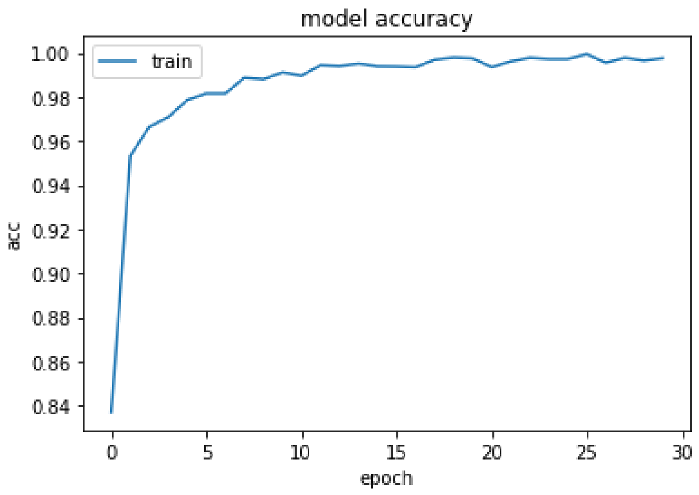
Model Used #4:

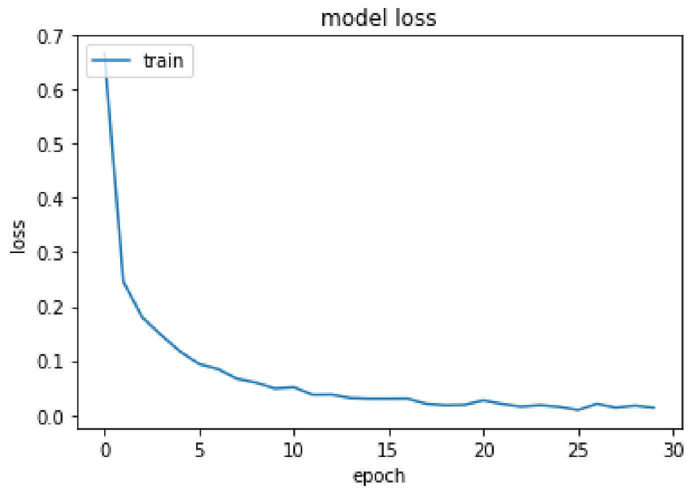




```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 6)         60
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 6)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 9, 9, 16)          2416
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 16)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 200)               51400
_________________________________________________________________
dense_5 (Dense)              (None, 80)                16080
_________________________________________________________________
dropout_2 (Dropout)          (None, 80)                0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                810
=================================================================
Total params: 70,766
Trainable params: 70,766
Non-trainable params: 0
_________________________________________________________________
```

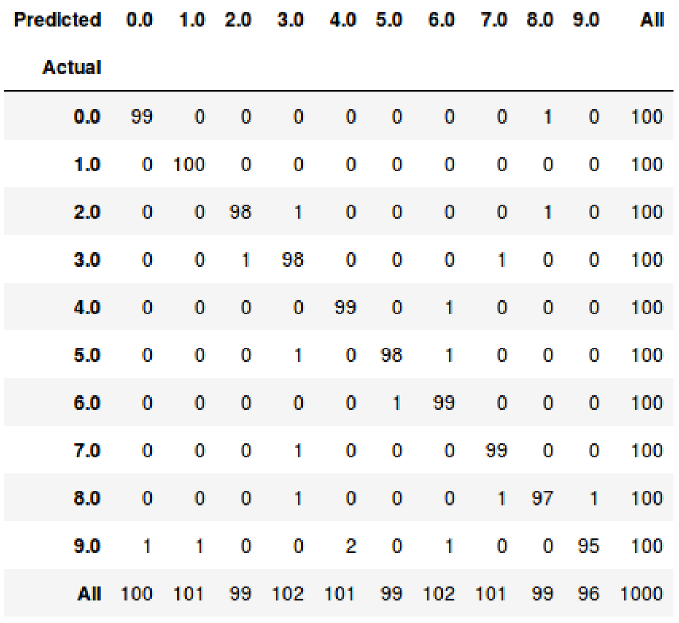
Plotting the Model loss and accuracy vs epochs





Comment:

This shows that the model loss saturates after the 12th epoch and gives very good result which is proves from the following confusion matrix.



Predicted	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	All
Actual											
0.0	99	0	0	0	0	0	0	0	1	0	100
1.0	0	100	0	0	0	0	0	0	0	0	100
2.0	0	0	98	1	0	0	0	0	1	0	100
3.0	0	0	1	98	0	0	0	1	0	0	100
4.0	0	0	0	0	99	0	1	0	0	0	100
5.0	0	0	0	1	0	98	1	0	0	0	100
6.0	0	0	0	0	0	1	99	0	0	0	100
7.0	0	0	0	1	0	0	0	99	0	0	100
8.0	0	0	0	1	0	0	0	1	97	1	100
9.0	1	1	0	0	2	0	1	0	0	95	100
All	100	101	99	102	101	99	102	101	99	96	1000

As we can see, it is very accurate but there are still some confusion with 4, 6 and 9 as they are very close to each other and the MNIST data is reduced, therefore there are small errors and mislabeling

And for the computational time it took 285 seconds for each epoch and for testing it took 10 seconds.

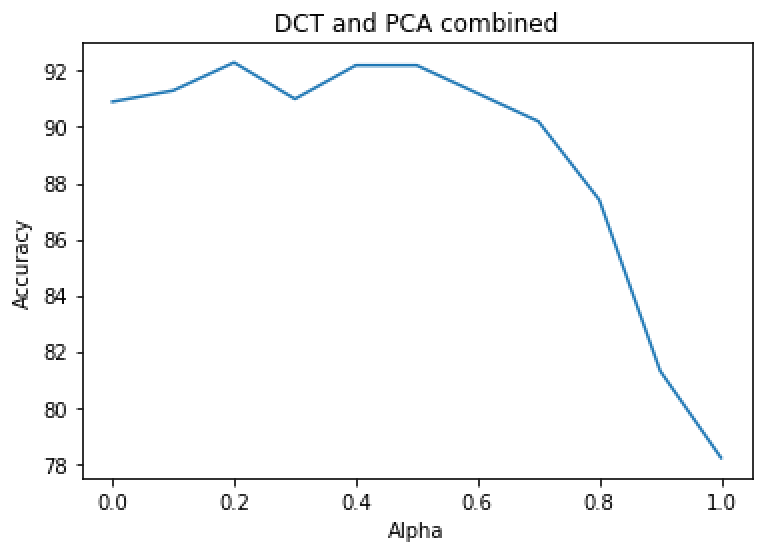
Combining DCT vs PCA: [α*DCT + (1- α)*PCA] and using K-means

α	Time of training	Time of prediction	Accuracy
0	4.87 sec	1.7 sec	90.9%
0.1	4.9 sec	1.7 sec	91.3%
0.2	4.84 sec	1.75 sec	92.3%
0.3	5 sec	1.7 sec	91%
0.4	4.9 sec	1.7 sec	92.2%
0.5	5.15 sec	1.73 sec	92.2%
0.6	4.9 sec	1.716 sec	91.2%
0.7	5.09 sec	1.7 sec	90.2%
0.8	5.3 sec	1.7 sec	87.4%
0.9	5.3 sec	1.7 sec	81.3%
1	6.3 sec	1.9	78.2%

Comment:

After using weight α we found out from the experiment that combining both DCT and PCA achieves higher accuracy that PCA and DCT alone, also the process indicates that PCA behaves slightly better than DCT

Equation: DCT*α + PCA*(1- α)



Best value for alpha is around 0.2

Model type	Features generation	Time of training	Time of prediction	Accuracy
K-means 1 cluster	Autoencoder 10 features	0.2149 sec	0.2204 sec	68.1 %
	Autoencoder 20 features	0.243 sec	0.2417 sec	73.5 %
	Centroid features	0.1048 sec	0.1505 sec	39.1 %
	DCT	0.1575 sec	0.1835 sec	59.0 %
	PCA	0.1909 sec	0.3548 sec	74.6 %
K-means 2 cluster	Autoencoder 10 features	0.6302 sec	0.4122 sec	72.4 %
	Autoencoder 20 features	0.6786 sec	0.4124 sec	78.1 %
	Centroid features	0.211 sec	0.2994 sec	47.4 %
	DCT	0.8128 sec	0.3320 sec	63.8 %
	PCA	1.1443 sec	0.2988 sec	80.1 %
K-means 4 cluster	Autoencoder 10 features	0.1509 sec	0.7499 sec	77.7 %
	Autoencoder 20 features	1.669 sec	0.8415 sec	80.8 %
	Centroid features	0.4457 sec	0.5447 sec	56.4 %
	DCT	1.3830 sec	0.5454 sec	70.3 %
	PCA	2.0916 sec	0.4627 sec	84.0 %
K-means 8 cluster	Autoencoder 10 features	1.3082 sec	1.4513 sec	79.3 %
	Autoencoder 20 features	1.7365 sec	1.522 sec	85.4 %
	Centroid features	0.7509 sec	1.0431 sec	64.0 %
	DCT	1.8654 sec	0.8386 sec	75.0 %
	PCA	2.3143 sec	0.8857 sec	89.5 %
K-means 16 cluster	Autoencoder 10 features	1.9183 sec	2.7627 sec	80.4 %
	Autoencoder 20 features	1.8665 sec	2.804 sec	88.1 %
	Centroid features	1.5426 sec	1.976 sec	70.5 %
	DCT	2.2383 sec	1.6592 sec	78.2 %
	PCA	2.6674 sec	1.6466 sec	91.8 %
1 GMM	Autoencoder 10 features	1.9795 sec	1.7749 sec	68.1 %
	Autoencoder 20 features	0.7902 sec	0.2434 sec	73.5 %
	Centroid features	0.5094 sec	0.2465 sec	39.1 %
	DCT	0.3606 sec	0.1775 sec	59.0 %
	PCA	0.9161 sec	0.1598 sec	74.5 %
2 GMM	Autoencoder 10 features	0.3852 sec	0.1796 sec	69.6 %
	Autoencoder 20 features	3.0359 sec	0.3434 sec	76.2 %
	Centroid features	0.7093 sec	0.373 sec	45.8 %
	DCT	1.7436 sec	0.2937 sec	65.1 %
	PCA	5.0936 sec	0.2963sec	80.0 %
4 GMM	Autoencoder 10 features	8.2123 sec	0.3169 sec	76.2 %
	Autoencoder 20 features	9.5174 sec	0.7396 sec	77.9 %
	Centroid features	1.8107 sec	0.7264 sec	53.4 %
	DCT	5.6600 sec	0.7784 sec	70.9 %
	PCA	9.4406 sec	0.5454 sec	83.0 %

SVM - linear kernels	Autoencoder 10 features	4.4774 sec	0.4924 sec	82.7 %
	Autoencoder 20 features	5.7885 sec	0.0679 sec	89.1 %
	Centroid features	4.4105 sec	0.0626 sec	62.0 %
	DCT	2.4367 sec	0.1107 sec	82.0 %
	PCA	5.1712 sec	0.2524 sec	90.3 %
SVM nonlinear kernels (Poly)	Autoencoder 10 features	10.2875 sec	0.193 sec	77.0 %
	Autoencoder 20 features	11.4055 sec	0.3847 sec	68.7 %
	Centroid features	2.4326 sec	0.5152 sec	81.1%
	DCT	2.3865 sec	0.0749 sec	92.1 %
	PCA	5.5565 sec	0.5134 sec	97.4 %

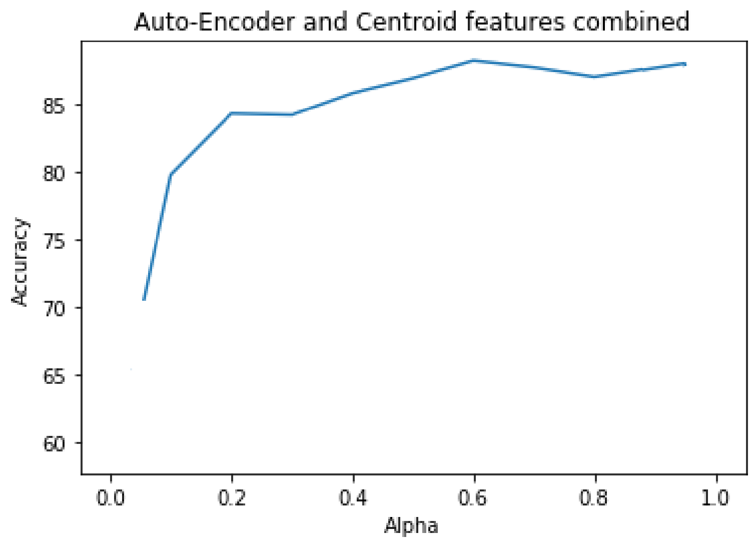
Combining AE and Centroid features Then using K-means

α	Time of training	Time of prediction	Accuracy
0	1.62 sec	1.91 sec	59.2%
0.1	1.59 sec	1.83 sec	79.8 %
0.2	1.46 sec	1.64 sec	84.3 %
0.3	1.92 sec	1.59 sec	84.2 %
0.4	1.72 sec	1.63 sec	85.8 %
0.5	2.38 sec	1.94 sec	86.9 %
0.6	1.96 sec	1.795 sec	88.2 %
0.7	1.81 sec	1.63 sec	87.7 %
0.8	2.31 sec	1.68 sec	87.0 %
0.9	1.82 sec	1.61 sec	87.7 %
1	1.91 sec	1.68 sec	70.5 %

Comment:

After using weight α we found out from the experiment that combining both Centroid features and Auto-Encoder achieves higher accuracy than any of them alone, also the process indicates that Auto encoder behaves better than Centroid features

*Equ: AE*α + Centroid*(1- α)*



Best alpha is around 0.6

3.1 Conclusions:

- Classification of Handwriiten numbers is an important part of day to day services and industries, using machines can increase efficiency of this process, descision of which algorithm to use depends on accuracy of this algorithm and processing time of it in our problem next points were noticed:
- As the number of clusters increases the accuracy increases.
- As the number of clusters increases computational time increases.
- GMM gives better accuracy for the same number of clusters over the KMeans, with more computational time in dct and pca features.
- Concatenation of the PCA and DCT didn't increase the accuracy dramatically, it almost remained the same as using PCA only.
- Centroid features was the worst among the feature reduction algorithms in most clustering algorithms however it did a better job with nonlinear svm than autoencoder.
- Using Autoencoder has the upper hand regarding its accuracy since it tailors the output features on the type of the input it was trained on.
- As the number of output nodes in the encoder increases accuracy increases.
- Autoencoder has relatively slower timing due to time it takes to be trained on dataset rather than being a ready mathematical transformation.
- Nonlinear classification can hurt the algorithm rather than benefit from it.
- SVM with non-linear kernel was the most powerful algorithm of classification among the algorithms for this problem regarding accuracy using pca features.
- Autoencoder can benefit from deeping the layers but that will cause the model to be computationally to be more expensive in terms of time and memory usage.
- As the number of epochs loops increases loss function decreases and gets slower as learning algorithm reaches the minimum point.
- CNN is the most powerful technique obviously from the very high accuracy achieved due to the forward and back propagation which changes the parameters and makes it learn a lot of features about the input.
- Dropouts helps achieve better accuracy and faster computations, as it removes nodes depending on a probability
- L2 regularization is an efficient way to reduce over fitting and improves the test accuracy
- Increasing no. of epochs help at first, but then the loss settles down minimum

- With a small batch size, the gradients are only a *very* rough approximation of the true gradients. So it’ll take a lot longer to find a good solution. Generally Smaller batch sizes are OK, but will take a bit longer.
- Using larger mini-batches allows network to reduce the variance of your stochastic gradient (mini batch of size 1) updates (by taking the average of the gradients in the mini-batch), and this in turn allows you to take bigger step-sizes
- Larger mini-batches are very attractive computationally. In addition to easy parallelism across processors/machines, they can give much better throughput.
- However small mini-batches down to stochastic gradient update parameters by adding (minus sign) the gradient computed on a *single* instance of the dataset. Since it's based on one random data point, it's very noisy and may go off in a direction far from the batch gradient. However, the noisiness is exactly what we want in non-convex optimization, because it helps escaping from saddle points or local minima.
- Batch size of 32 allowed as to avoid big noise in search for minimum as with batch size of 16, and take good optimization step towards minimum loss and higher accuracy as with batch size of 64.
- Transfer learning was used to learn further more with more epochs after training one model and benefit from weights it learned from earlier training.