CSE 574

Introduction to Machine Learning

PA 2

Spring 2019
04/13/2019

Handwritten Digit Classification

Project Report

Prepared By :

Md Moniruzzaman Monir (50291708)
Archisman Patra (50291555)
Tushar Gupta (50291152)

Introduction:

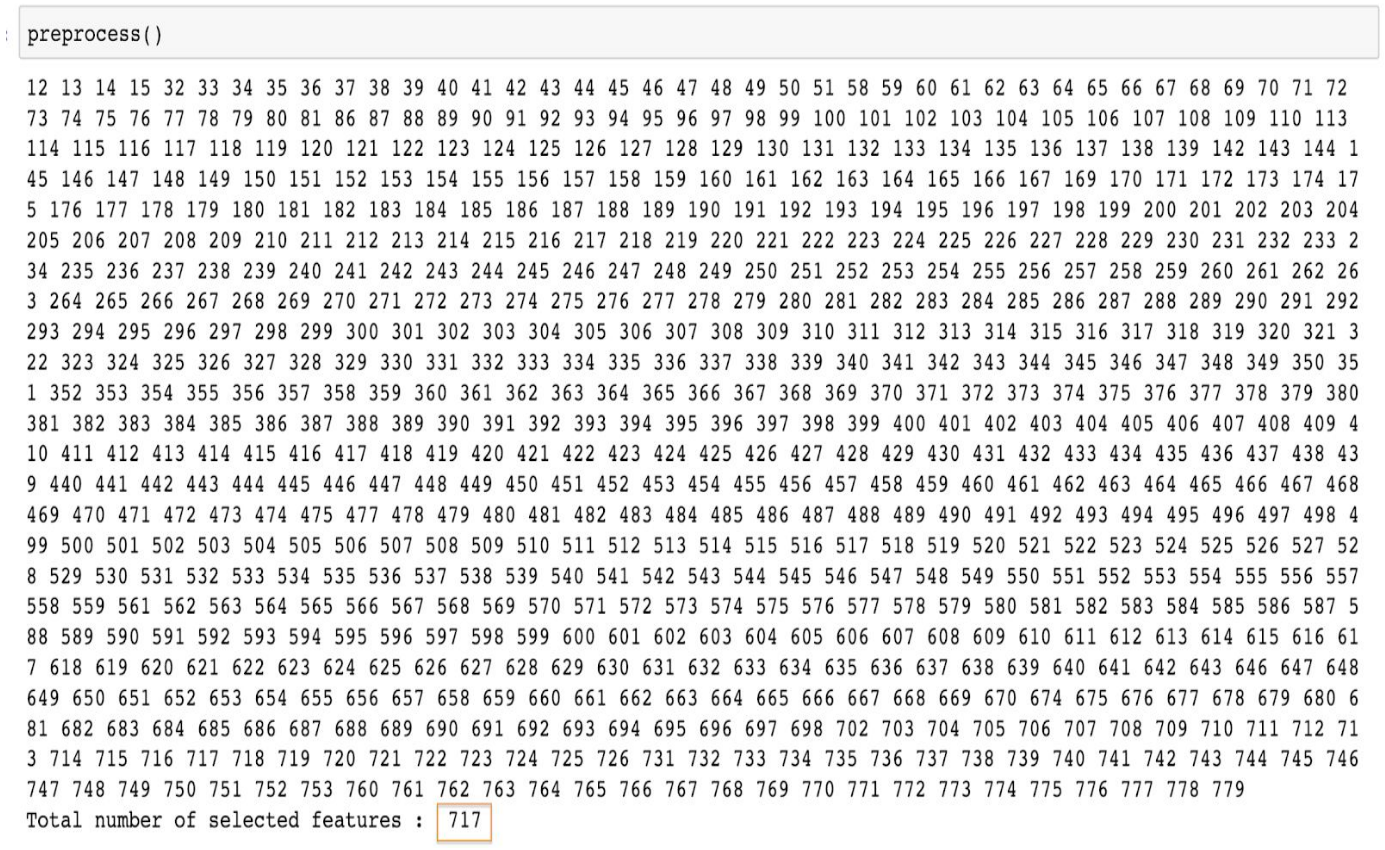
In this assignment, we have implemented a Multilayer Perceptron Neural Network and evaluated its performance in classifying handwritten digits on the **MNIST** Dataset. Then we used the same network to analyze a face dataset (**celebA**) to recognize who wore spectacles and compare the performance of the neural network against a deep neural network and a convolutional neural network using the TensorFlow Library.

Feature Selection:

In the dataset there are many features on which the values are exactly the same for all data points in the training set. With those feature the model cannot gain any more information about the variation between data points. So, we can ignore those features and choose the ones that are useful.

Total no. of features that are being used is 717.

Features indices are as follows:



```
preprocess()
```

```
12 13 14 15 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80 81 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 113
114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 142 143 144 1
45 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 169 170 171 172 173 174 17
5 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204
205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 2
34 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 26
3 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292
293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 3
22 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 35
1 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380
381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 4
10 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 43
9 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468
469 470 471 472 473 474 475 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 4
99 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 52
8 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557
558 559 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 5
88 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 61
7 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 646 647 648
649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 674 675 676 677 678 679 680 6
81 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 702 703 704 705 706 707 708 709 710 711 712 71
3 714 715 716 717 718 719 720 721 722 723 724 725 726 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746
747 748 749 750 751 752 753 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779
Total number of selected features : 717
```

How we choose hyper-parameter for Neural Network:

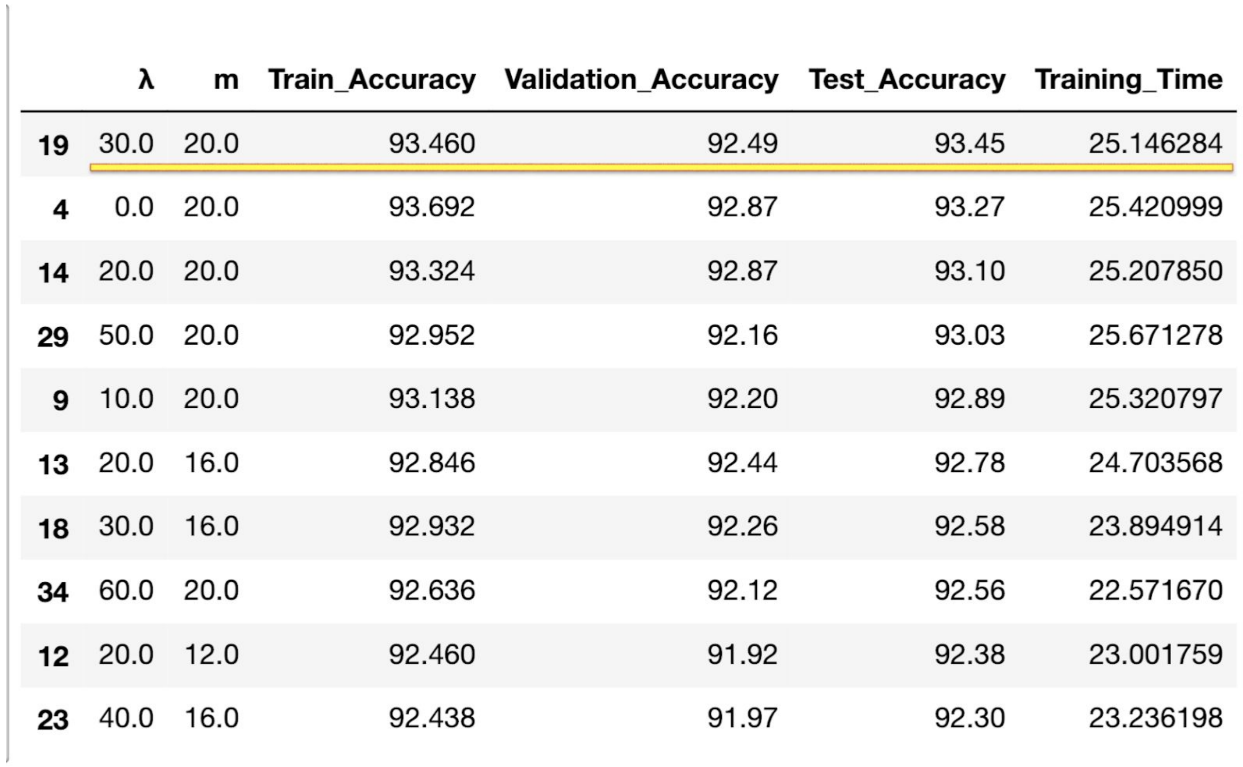
Below we explain how we choose the optimal hyper parameters (regularization coefficient, **λ** and no of hidden units, **m**) combination with supporting figures. We achieved the best combination by using **GridSearch** on different values of **λ** and no of hidden units (m).

We use the regularization in neural network to avoid the **overfitting** problem. We have also seen that different no hidden units affects the performance of the neural network. So, we varied the **m** from 4 to 20 with a step size of 4 and **λ** from 0 to 60 in increments of 10 (as recommended in the project manual).

Choosing Best Combination :

(varying **λ** from 0-60 with step size of 10 & m from 4-20 with step size of 4):

Following are the top hyper parameter combination that works best on Testing Data in this setting:



	λ	m	Train_Accuracy	Validation_Accuracy	Test_Accuracy	Training_Time
19	30.0	20.0	93.460	92.49	93.45	25.146284
4	0.0	20.0	93.692	92.87	93.27	25.420999
14	20.0	20.0	93.324	92.87	93.10	25.207850
29	50.0	20.0	92.952	92.16	93.03	25.671278
9	10.0	20.0	93.138	92.20	92.89	25.320797
13	20.0	16.0	92.846	92.44	92.78	24.703568
18	30.0	16.0	92.932	92.26	92.58	23.894914
34	60.0	20.0	92.636	92.12	92.56	22.571670
12	20.0	12.0	92.460	91.92	92.38	23.001759
23	40.0	16.0	92.438	91.97	92.30	23.236198

So, from the above table we can see that our optimum **λ** and **m** combination is 30 and 20, as this gives us the best accuracy on test data, that is **93.45 %.**

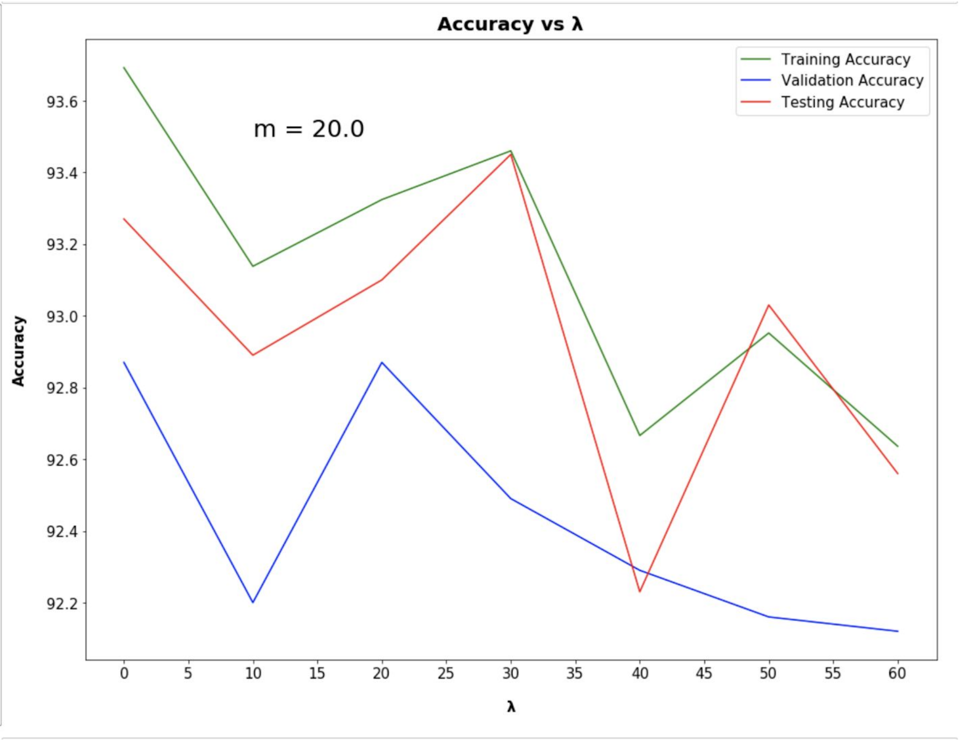
Now we vary **λ** by fixing the value of m to 20.

We can infer from the following table that the value of **λ** = 30 gives us optimum result.

```
rows_with_optimal_m = rows_with_optimal_m.sort_values(by=['λ'])
rows_with_optimal_m
```

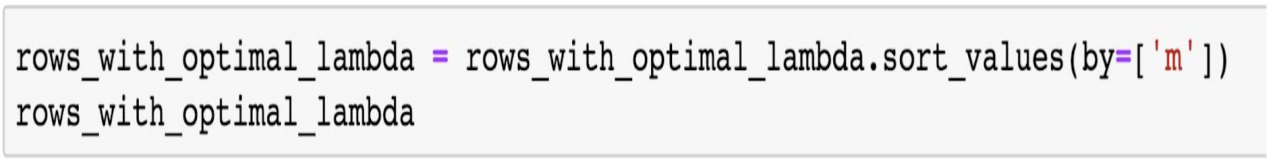
	λ	m	Train_Accuracy	Validation_Accuracy	Test_Accuracy	Training_Time
4	0.0	20.0	93.692	92.87	93.27	25.420999
9	10.0	20.0	93.138	92.20	92.89	25.320797
14	20.0	20.0	93.324	92.87	93.10	25.207850
19	30.0	20.0	93.460	92.49	93.45	25.146284
24	40.0	20.0	92.666	92.29	92.23	23.692462
29	50.0	20.0	92.952	92.16	93.03	25.671278
34	60.0	20.0	92.636	92.12	92.56	22.571670

Now we plot Accuracy with respect to different **λ** values ::

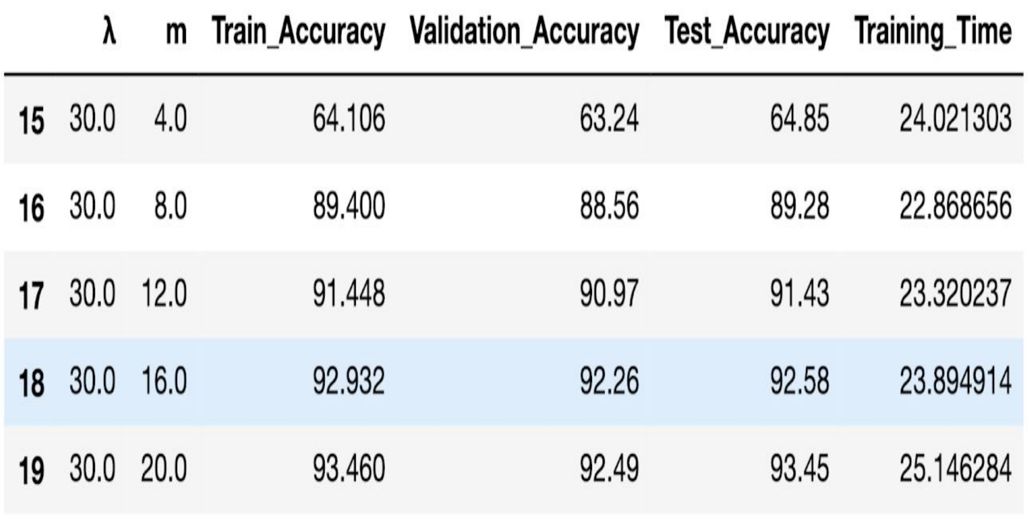


We see that the best testing accuracy is for **λ = 30.** As the **λ** value increases we see a little fluctuation in the training, validation and testing results. Theoretically, there should be a decrease in the accuracy after the optimum point due to underfitting. Our plot also justifies that as we see a overall decrease after that. However, there are little anomalies, which is justified as this is a practical setting and we get the same kind of anomaly in training, validation and testing data.

Now we fix the **λ** at 30 and vary the no of hidden units (m) ::



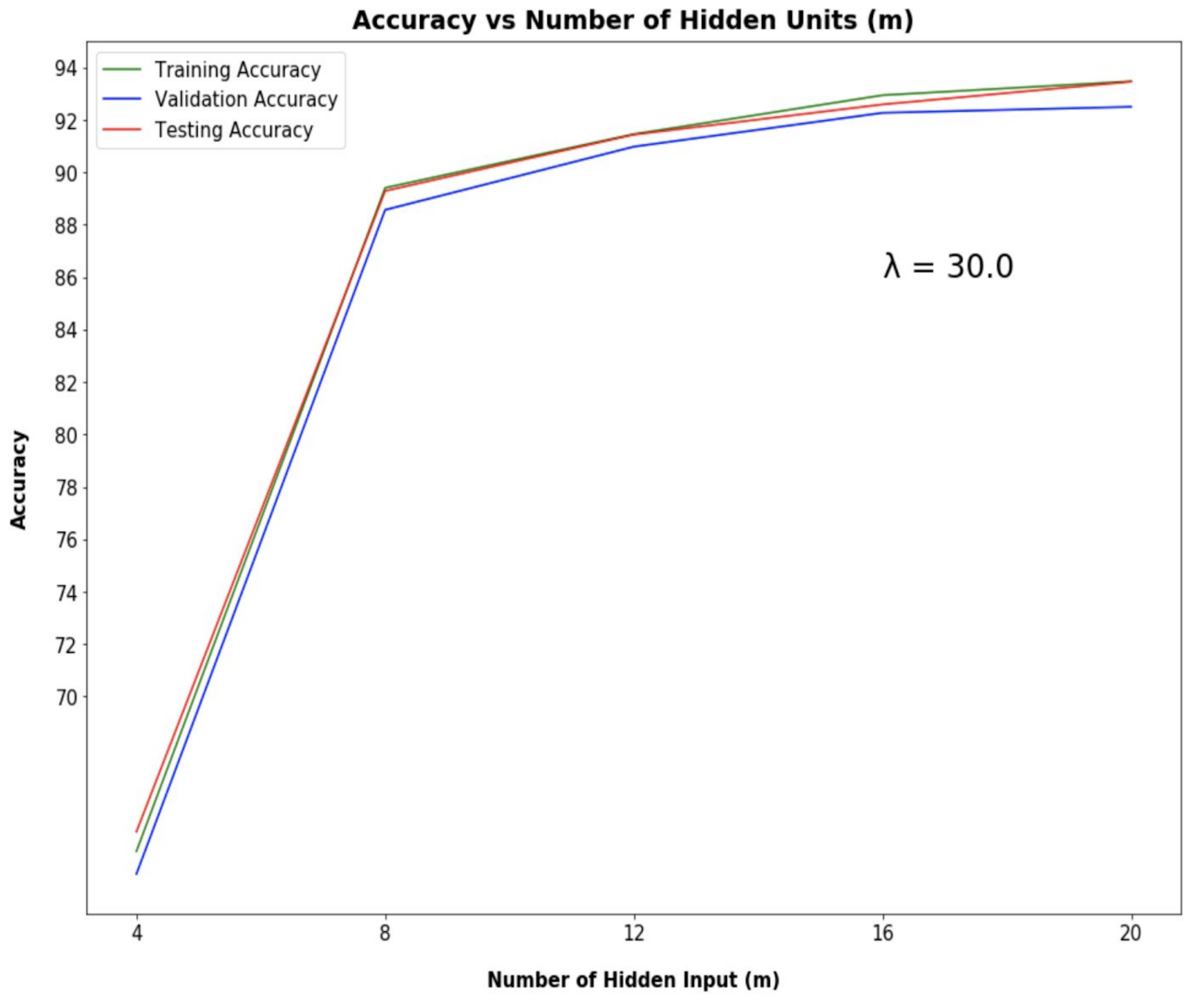
```
rows_with_optimal_lambda = rows_with_optimal_lambda.sort_values(by=['m'])
rows_with_optimal_lambda
```



	λ	m	Train_Accuracy	Validation_Accuracy	Test_Accuracy	Training_Time
15	30.0	4.0	64.106	63.24	64.85	24.021303
16	30.0	8.0	89.400	88.56	89.28	22.868656
17	30.0	12.0	91.448	90.97	91.43	23.320237
18	30.0	16.0	92.932	92.26	92.58	23.894914
19	30.0	20.0	93.460	92.49	93.45	25.146284

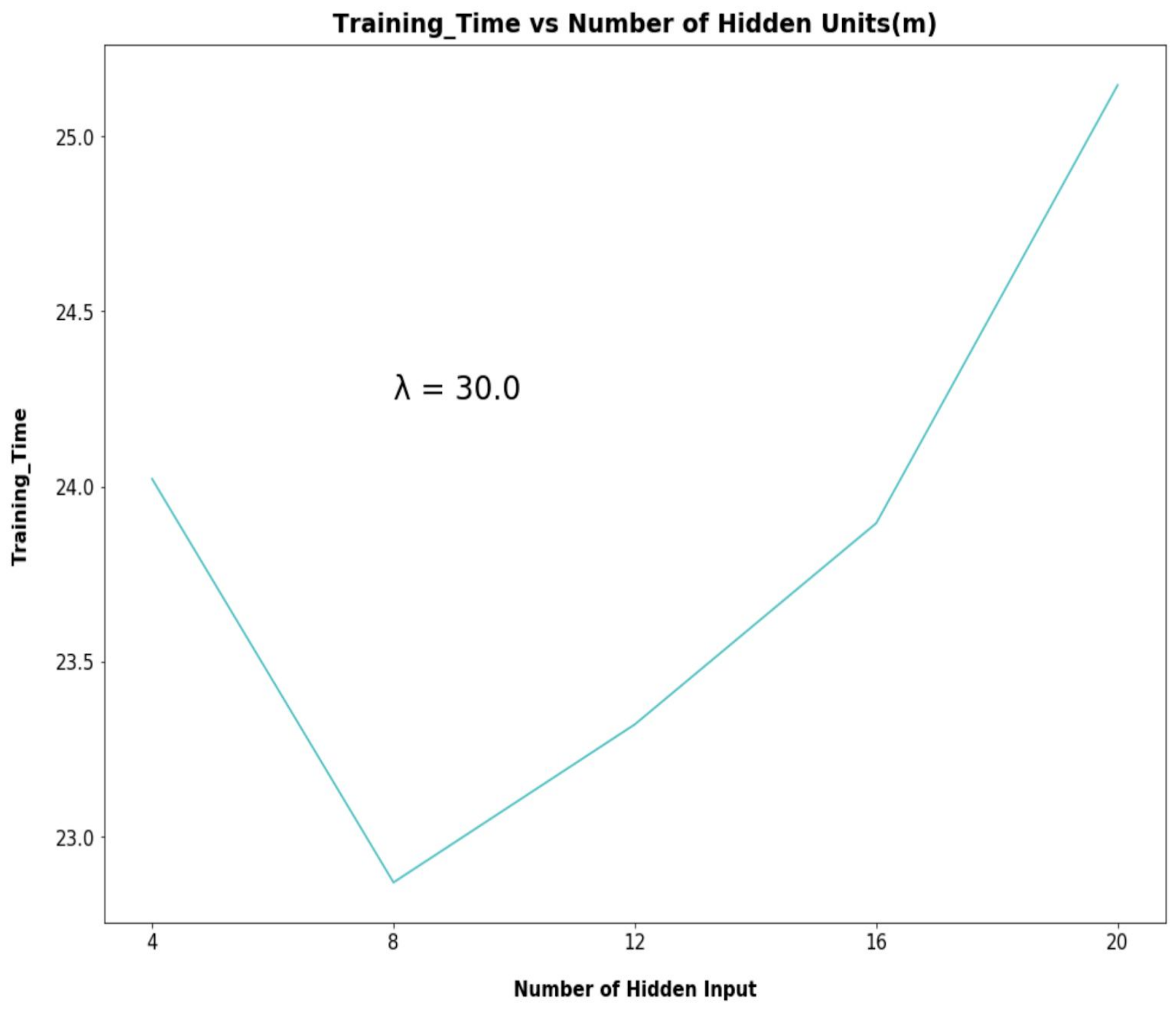
We can see that **m = 20** gives us best result.

Here we plot no of hidden units vs Accuracy:



We can conclude from the above plot that as the no of hidden unit (m) increases, accuracy also increases. There is a huge increase from 4 to 8 and then it increases slowly and from 16 to 20 the curve is almost flat. So, we can also infer that after certain number of hidden units there is no significant difference in accuracy even we increase m.

Now we plot the **Training time** vs **no of hidden units**:

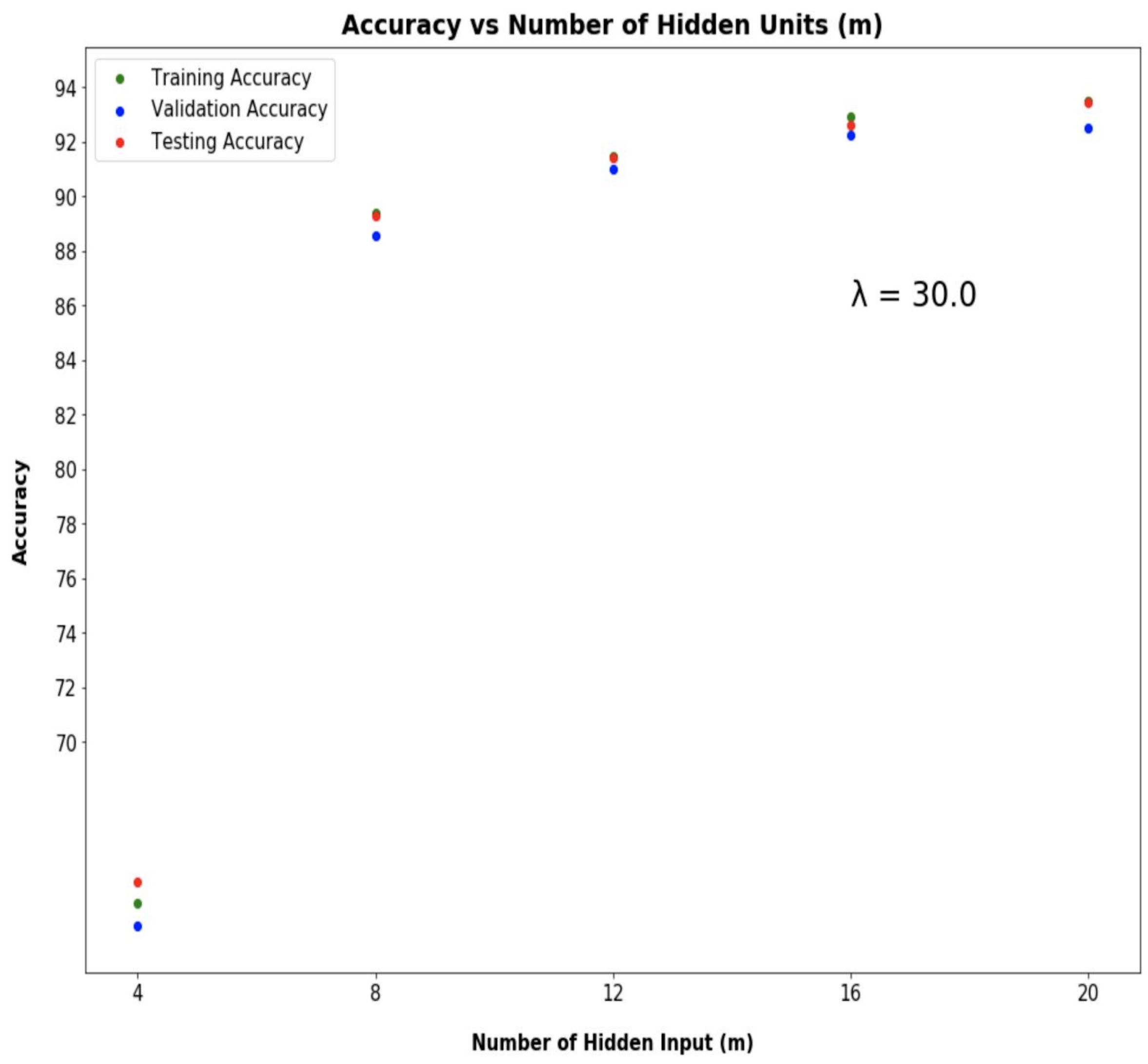


As depicted in the above plot, naturally the training time increases as we increase the no of hidden units in the model. So, we should be careful when increasing the no of hidden unit, if there is no significant change in the Accuracy then we should stop increasing m at that point as that will cost us huge time.

Accuracy of Classification Method on the Handwritten Digits Test Data using nnScript.py( ):

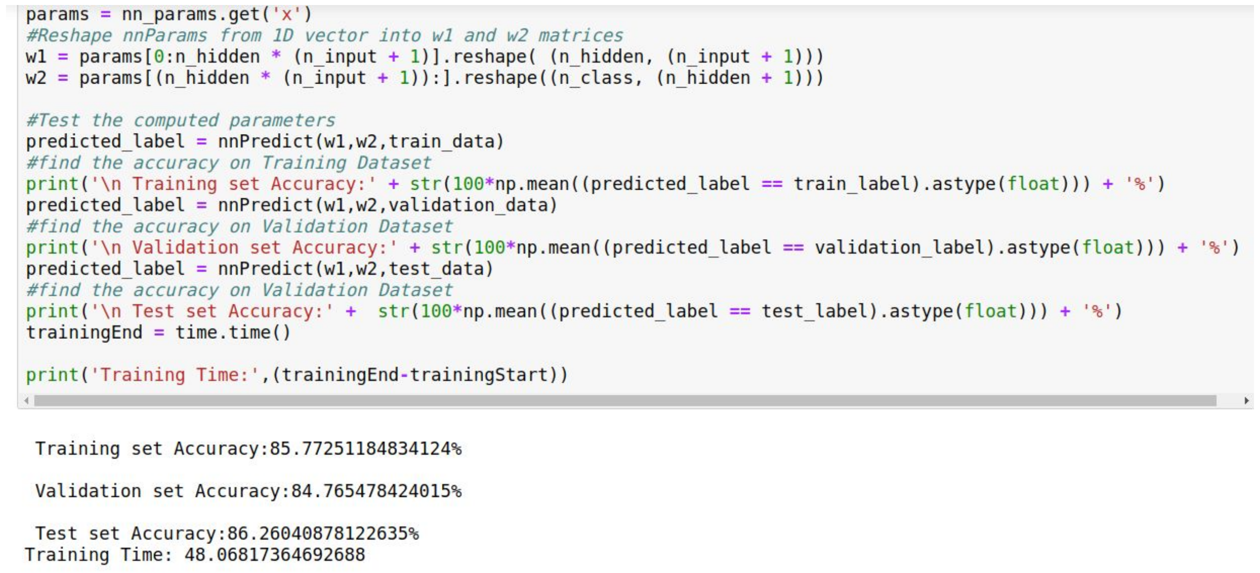
The best Accuracy we got on the Test Data: **93.45%**
Best Hyper Parameter combination: **λ = 30, m = 20**

Here is a scatter plot of Accuracy vs no of hidden units at **λ** = 30:



Accuracy of classification method on the celebA dataset:

We have used our implementations of the functions- sigmoid( ), nnObjFunc( ) and nnPredict( ) from the nnScript in the facennScript( ). Following are the result obtained by running the facennScript.py at **λ** = 10:



```
params = nn_params.get('x')
#Reshape nnParams from 1D vector into w1 and w2 matrices
w1 = params[0:n_hidden * (n_input + 1)].reshape( (n_hidden, (n_input + 1)))
w2 = params[(n_hidden * (n_input + 1)):].reshape((n_class, (n_hidden + 1)))

#Test the computed parameters
predicted_label = nnPredict(w1,w2,train_data)
#find the accuracy on Training Dataset
print('\n Training set Accuracy:' + str(100*np.mean((predicted_label == train_label).astype(float))) + '%')
predicted_label = nnPredict(w1,w2,validation_data)
#find the accuracy on Validation Dataset
print('\n Validation set Accuracy:' + str(100*np.mean((predicted_label == validation_label).astype(float))) + '%')
predicted_label = nnPredict(w1,w2,test_data)
#find the accuracy on Validation Dataset
print('\n Test set Accuracy:' +  str(100*np.mean((predicted_label == test_label).astype(float))) + '%')
trainingEnd = time.time()

print('Training Time:',(trainingEnd-trainingStart))
```

```
 Training set Accuracy:85.77251184834124%

 Validation set Accuracy:84.765478424015%

 Test set Accuracy:86.26040878122635%
Training Time: 48.06817364692688
```

Comparison of Neural Network with a Deep Neural Network (using TensorFlow) in terms of Accuracy and training time:

We compare and analyze the results in the **celebA** dataset between single layer Neural Network and the multilayer Deep Neural Network. We increase the number of layers from 2 to 7 (2, 3, 5, 7) and got the following results on Test Set:

No of Layers	Script	Accuracy (%)	Training Time (Sec)
1	facennScript.py	86.26	48.068
2	deepnnScript.py	81.19	70.259
3	deepnnScript.py	78.54	117.364
5	deepnnScript.py	76.04	124.820
7	deepnnScript.py	73.99	154.888

Inference:

- As expected, training time increases with the increase of no of hidden layer. Due to increase in the complexity of the model, computational time increases.
- We have also noticed that accuracy of the model decreases with the increase in number of hidden layers. As we see from the above table, this is because of the overfitting of the model, which increase the validation and training accuracy but decreases the testing accuracy.

Results from the Convolutional Neural Network in terms of Accuracy and Training Time:

We run the convolutional neural network on the given dataset and get the confusion matrix, our observations are as follows:

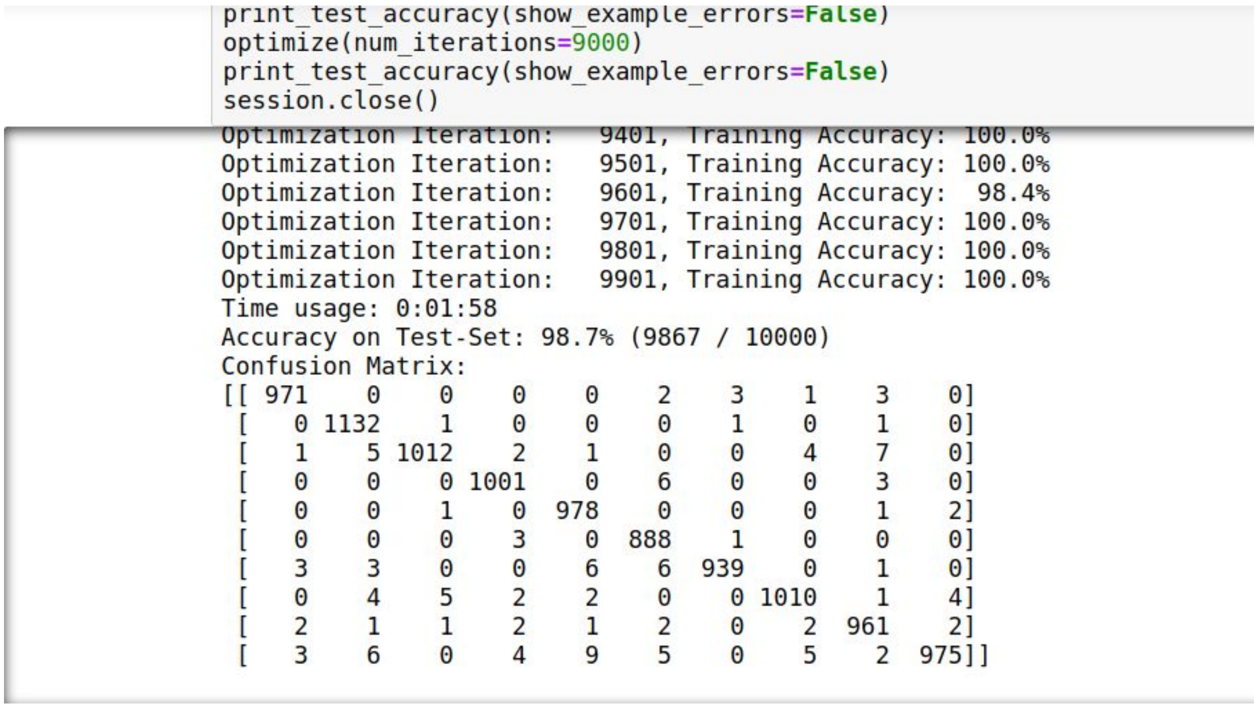
- We compare the accuracy of the CNN model and we see that the accuracy is much better than the single layer neural network and multilayer deep neural network. The accuracy of the CNN model is **98.7 %**

- We infer that this is because of the way CNN configures the model. CNN changes the hyper parameters depending upon the particular requirement of the image and independence of prior knowledge in selecting the features.

- The main reason for this accuracy is the use of filter matrix. As when create the filter by convolution we divide the input image into many parts of lower dimension and multiply with the filter of same dimension. So, we can change the image at every stage.

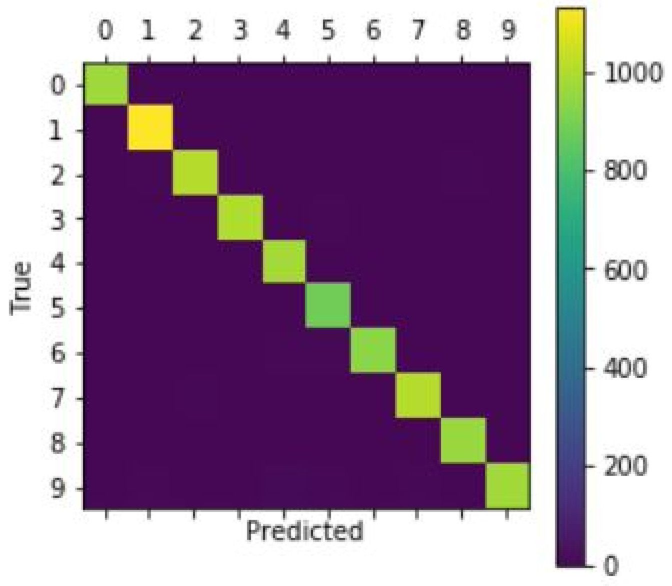
Training Time	Accuracy(%)
1 minute 58 seconds	98.7

Confusion Matrix for final Iteration:



```
print_test_accuracy(show_example_errors=False)
optimize(num_iterations=9000)
print_test_accuracy(show_example_errors=False)
session.close()
```

```
Optimization Iteration:   9401, Training Accuracy: 100.0%
Optimization Iteration:   9501, Training Accuracy: 100.0%
Optimization Iteration:   9601, Training Accuracy:  98.4%
Optimization Iteration:   9701, Training Accuracy: 100.0%
Optimization Iteration:   9801, Training Accuracy: 100.0%
Optimization Iteration:   9901, Training Accuracy: 100.0%
Time usage: 0:01:58
Accuracy on Test-Set: 98.7% (9867 / 10000)
Confusion Matrix:
[[ 971    0    0    0    0    2    3    1    3    0]
 [    0 1132    1    0    0    0    1    0    1    0]
 [    1    5 1012    2    1    0    0    4    7    0]
 [    0    0    0 1001    0    6    0    0    3    0]
 [    0    0    1    0  978    0    0    0    1    2]
 [    0    0    0    3    0  888    1    0    0    0]
 [    3    3    0    0    6    6  939    0    1    0]
 [    0    4    5    2    2    0    0 1010    1    4]
 [    2    1    1    2    1    2    0    2  961    2]
 [    3    6    0    4    9    5    0    5    2  975]]
```



From the above matrix we can see almost all the predictions are correct except a few and we get the final accuracy as 98.7%.