Chetan Gupta

A20378854

Machine Learning Final Project Report

Phase 1

Bush

Classifier	KNeighborsClassifier			
Parameters	n_neighbors=1			
	result1	result2	result3	mean result
fit_time	5.03912306	5.1881249	5.04586196	5.09103664
score_time	954.3408599	957.318552	957.8084559	956.4892893
test_f1	0.12834225	0.15434084	0.16085791	0.147847
test_precision	0.12182741	0.17910448	0.15228426	0.15107205
test_recall	0.13559322	0.13559322	0.17045455	0.147213663

Classifier	KNeighborsClassifier			
Parameters	n_neighbors=3			
	result1	result2	result3	mean result
fit_time	4.924582	5.02487421	5.31349206	5.087649423
score_time	953.802372	957.0233059	951.643811	954.1564963
test_f1	0.02690583	0.07106599	0.11374408	0.070571967
test_precision	0.06521739	0.35	0.34285714	0.25269151
test_recall	0.01694915	0.03954802	0.06818182	0.041559663

Classifier	KNeighborsClassifier			
Parameters	n_neighbors=5			
	result1	result2	result3	mean result
fit_time	5.07412386	4.95722389	5.18560195	5.072316567
score_time	961.2310932	957.529923	960.4045391	959.7218517
test_f1	0.04	0	0.06282723	0.034275743

test_precision	0.17391304	0	0.4	0.191304347
test_recall	0.02259887	0	0.03409091	0.018896593

SVC classifier parameters that returned a mean zero f1
SVC classifier parameters that returned a mean zero f1

C	kernel	degree	gamma
0.001	**linear**	N/A	N/A
0.0001	**linear**	N/A	N/A
0.1	**poly**	1	N/A
100	**poly**	4	N/A
10	**rbf**	N/A	0.0001
0.001	**rbf**	N/A	auto
0.01	**sigmoid**	N/A	0.001
0.1	**sigmoid**	N/A	0.001

(more rows and parameters)

Best (in terms of mean F1) SVC result I got				
Parameters	C=100	kernel=Linear		
	result1	result2	result3	mean result
fit_time	82.3658148	83.04409814	83.72238148	83.04409814
score_time	84.16569604	82.66568	81.16566396	82.66568
test_f1	0.66	0.66	0.63	0.65
test_precision	0.79822	0.83695	0.759562	0.798244
test_recall	0.568	0.526865	0.3686	0.487821667

William

Classifier	KNeighborsClassifier			
Parameters	n_neighbors=1			
	result1	result2	result3	mean result
fit_time	5.40667987	5.257689	5.76869702	5.47768863
score_time	965.0577281	971.516664	979.1235108	971.899301
test_f1	0.19047619	0.08695652	0.32	0.199144237
test_precision	0.66666667	0.16666667	0.5	0.444444447
test_recall	0.11111111	0.05882353	0.23529412	0.135076253

Classifier	KNeighborsClassifier			
Parameters	n_neighbors=3			
	result1	result2	result3	mean result
fit_time	5.49731708	5.59161901	5.70140481	5.5967803
score_time	983.175899	982.403609	982.14534	982.5749493
test_f1	0	0	0	0
test_precision	0	0	0	0
test_recall	0	0	0	0

Classifier	KNeighborsClassifier			
Parameters	n_neighbors=5			
	result1	result2	result3	mean result
fit_time	5.53675294	5.43971586	5.43185091	5.469439903
score_time	995.295315	981.9072731	971.7261169	982.976235
test_f1	0	0	0	0
test_precision	0	0	0	0
test_recall	0	0	0	0

SVC classifier parameters that returned a mean zero f1			
C	kernel	degree	gamma
0.001	linear	N/A	N/A
0.0001	linear	N/A	N/A
50	poly	2	N/A
50	poly	3	N/A
100	poly	4	N/A
1	rbf	N/A	0.001
0.001	rbf	N/A	auto
0.001	sigmoid	N/A	auto
10	sigmoid	N/A	0.0001

Best (in terms of mean F1) SVC result I got				
Parameters	C=100	kernel=Linear	??	
	result1	result2	result3	mean result
fit_time	12.64288545	11.85234189	11.63018417	12.04180384

score_time	11.13684869	10.3612899 8	10.181165 93	10.559768 2
test_f1	0.59	0.698	0.39855	0.4783549 77
test_precision	0.5698	0.65866	0.7585268	0.6995671
test_recall	0.36885	0.69898	0.3655	0.3834422 67

Results analysis: - KNN is highly susceptible to number of nearest neighbors chosen the more the number of neighbors the better the generalization for the model and vice versa.

KNN is also shown to perform very bad when the number of dimension increases. Which concluded "the curse of dimensionality" theory by Mr. Bellman, since in our number of dimensions grows to 64*64 image it very evident that KNN will not be a function approximator.

SVM model are very dependent feature techniques since in our method we used very basic feature selection which the vector representation of the grey scale image. We believe the feature representation didn't capture the gradients and its orientation in the image. Since CNNs are combine of many neurons and each neuron the output of the linear combination of the neurons on the previous layers. Inherently it performs operation as edge detection by convolving the image. They perform better function approximator to map the input to the output.

Having used advance feature selection techniques such as histogram of oriented gradients(hog) or sift detector we believe SVM could have performed much better.

Phase 2

Bush

Phase 1 results			
Classifier	**Parameters**	**Mean F1**	
KNeighborsClassifier	n_neighbors=1	0.147847	
KNeighborsClassifier	n_neighbors=3	0.070571967	
KNeighborsClassifier	n_neighbors=5	0.034275743	
SVC (Best result)	C=500, kernel=rbf, gamma=0.0001	0.647118153	
Phase 2 best results			

Best result for KNeighborsClassifier			
PCA parameters	N_componenets = 32		
KNeighborsClassifier parameters	n_neighbors=1		
Mean F1	0.165495487		
Best result for SVC			
PCA parameters	n_components = 4050		
SVC parameters	C=4050	kernel=rbf	gamma=auto
Mean F1	0.646344787		

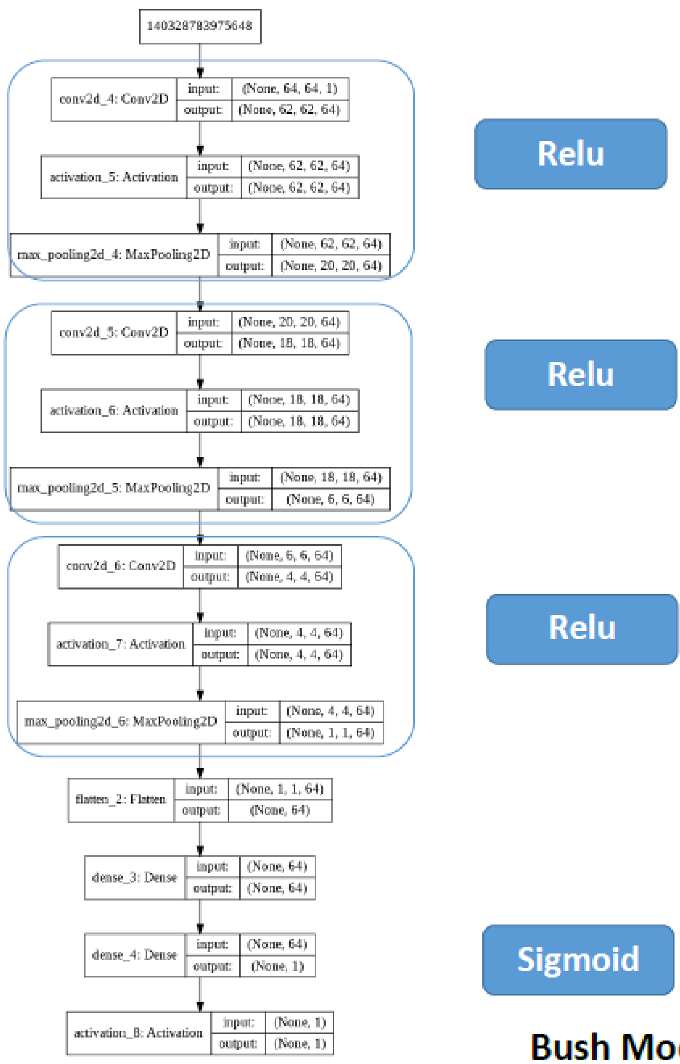
William

Phase 1 results		
Classifier	**Parameters**	**Mean F1**
KNeighborsClassifier	n_neighbors=1	0.199144237
KNeighborsClassifier	n_neighbors=3	0
KNeighborsClassifier	n_neighbors=5	0
SVC (Best result)	C=100, kernel=linear, gamma=auto	0.51728395
Phase 2 best results		
Best result for KNeighborsClassifier		
PCA parameters	n_components = 32	
KNeighborsClassifier parameters	n_neighbors=1	
Mean F1	0.21682099	
Best result for SVC		
PCA parameters	n_components = 2050	
SVC parameters	C=100, kernel=linear	
Mean F1	0.49259259	

Results analysis: - PCA on the training set, save the principal components that you use, and then use them to transform the points in your test set. This way the points in both sets end up in the same space, and you are not using any knowledge about your test set during training. entirely separate data set, just for computing the principal components. Then project both your training set and your test set into the space defined by those. When we talk about time analysis. It takes a lot time to learn and give results, but it also improves results as compared to KNN and SVM without PCA.

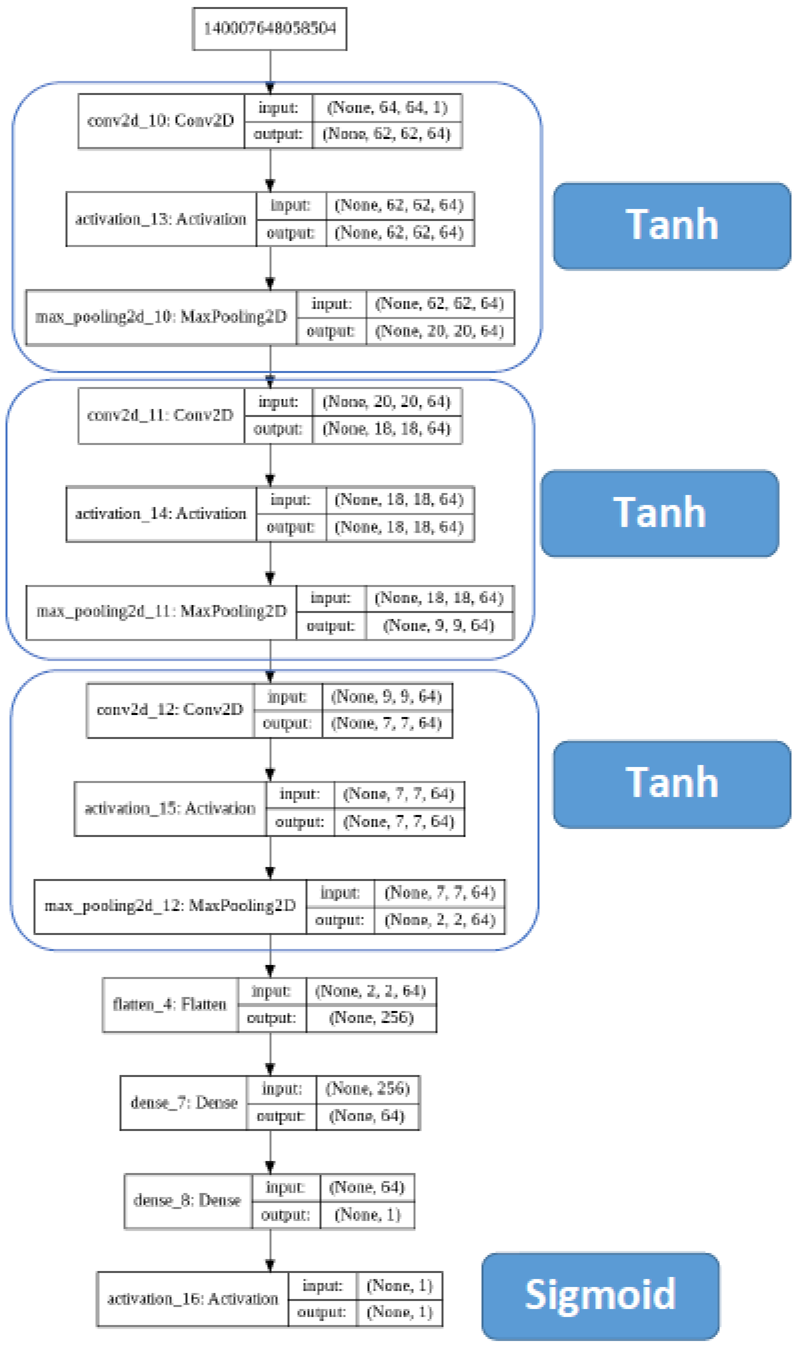
Phase 3

Bush



Bush Model

William Model



In Bush model I have used Adadelta optimizer and in William Adam optimizer with default learning rate has been used. I have used 30 epochs cycles in learn and validate. Following results has been captured by different experiments.

Bush f1

F1_test = 0.8668730644787165

F1_train = 1.0

William F1

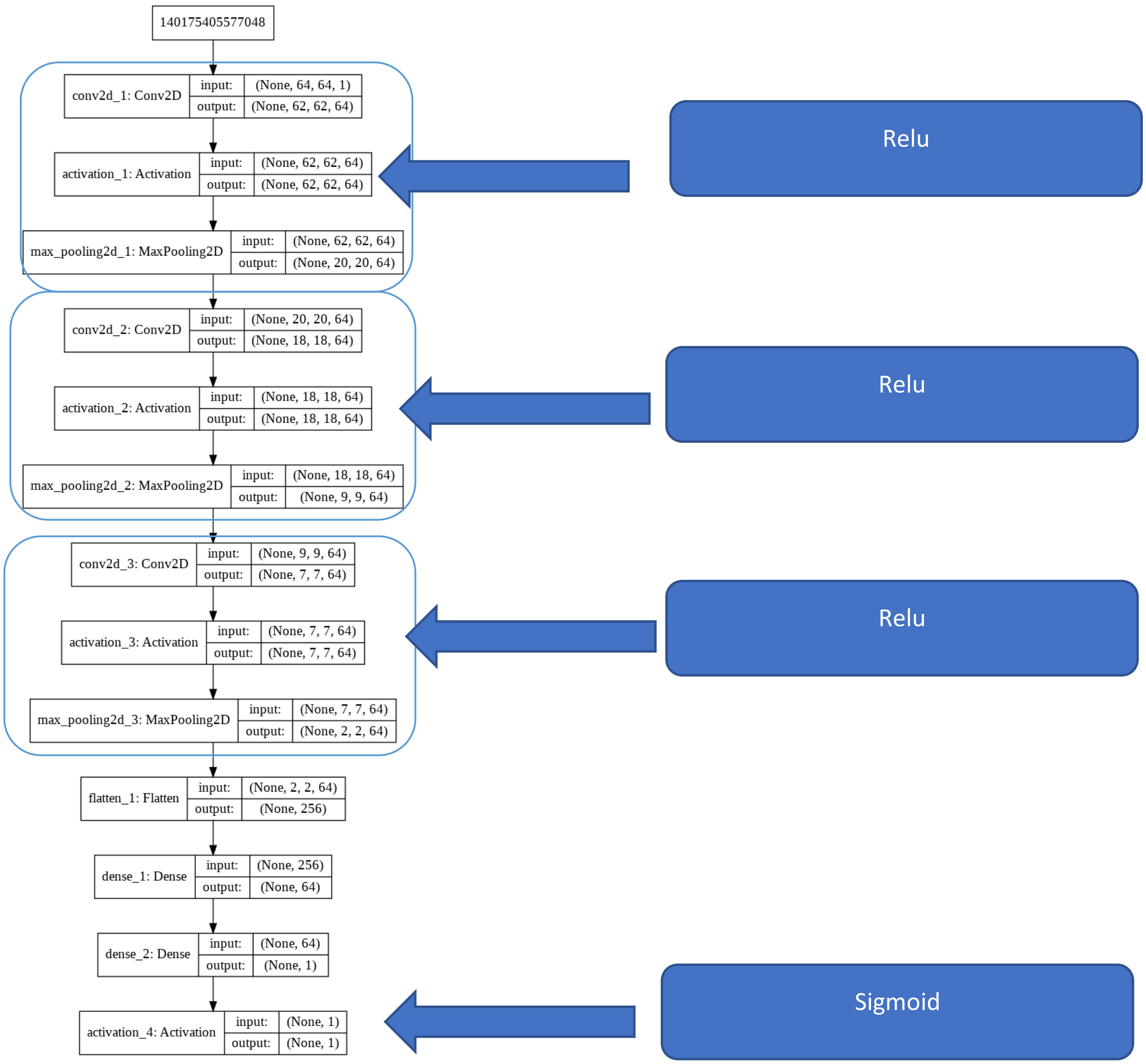
F1_test = 0.7857142801020407

F1_train = 1.0

Results analysis: - CNN helps in training and testing the model more efficiently as compare to KNN and SVM. . Since CNNs are combine of many neurons and each neuron the output of the linear combination of the neurons on the previous layers. Inherently it performs operation as edge detection by convolving the image. They perform better function approximator to map the input to the output.

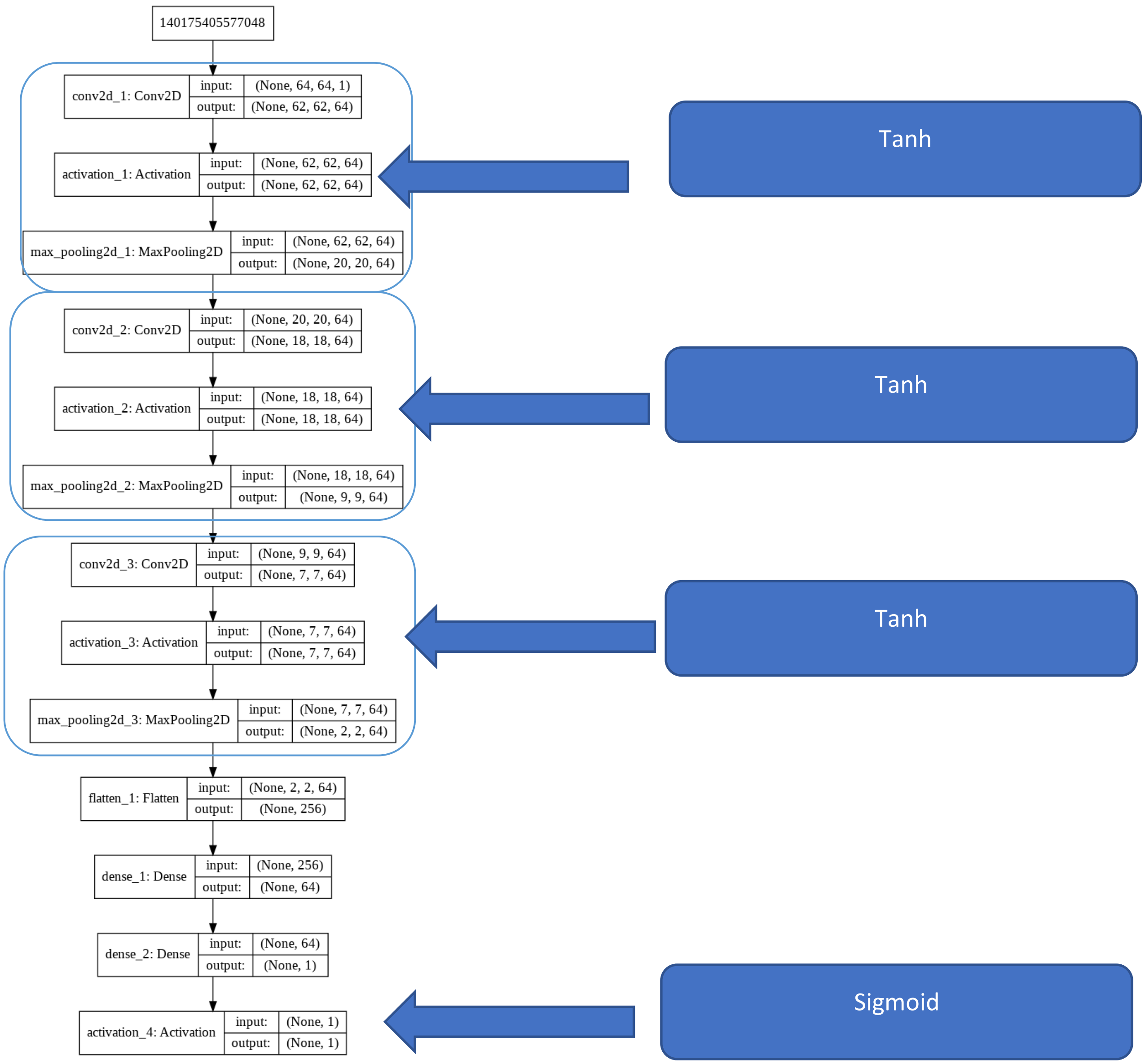
Phase 4

Bush



140175405577048
conv2d_1: Conv2D
input: (None, 64, 64, 1)
output: (None, 62, 62, 64)
activation_1: Activation
input: (None, 62, 62, 64)
output: (None, 62, 62, 64)
max_pooling2d_1: MaxPooling2D
input: (None, 62, 62, 64)
output: (None, 20, 20, 64)
conv2d_2: Conv2D
input: (None, 20, 20, 64)
output: (None, 18, 18, 64)
activation_2: Activation
input: (None, 18, 18, 64)
output: (None, 18, 18, 64)
max_pooling2d_2: MaxPooling2D
input: (None, 18, 18, 64)
output: (None, 9, 9, 64)
conv2d_3: Conv2D
input: (None, 9, 9, 64)
output: (None, 7, 7, 64)
activation_3: Activation
input: (None, 7, 7, 64)
output: (None, 7, 7, 64)
max_pooling2d_3: MaxPooling2D
input: (None, 7, 7, 64)
output: (None, 2, 2, 64)
flatten_1: Flatten
input: (None, 2, 2, 64)
output: (None, 256)
dense_1: Dense
input: (None, 256)
output: (None, 64)
dense_2: Dense
input: (None, 64)
output: (None, 1)
activation_4: Activation
input: (None, 1)
output: (None, 1)
Relu
Relu
Relu
Sigmoid

Williams



Configuration details

Cifar-10 F1 results

F1 Results	**Train**	1.0
	Test	0.550189908

		Bush	William
Activation function	Layer 1	Tanh	Tanh
	Layer 2	Tanh	Tanh
	Layer 3	Tanh	Tanh
	Output layer	Sigmoid	Sigmoid
Batch Size		32	64
Learning Rate		0.0001	0.0001
Epocs		30	30
F1 Results	Train	1	1
	Test	0.8125	0.6923076923076924
Cifar10 Training time		26 mins	20 mins
Transfer learning validation time		2.5 Mins	3 Mins

Dataset

The CIFAR-10 dataset

Link: - https://www.cs.toronto.edu/~kriz/cifar.html
https://keras.io/datasets/

Details: - The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Dataset preprocessing: -

Dataset imported from Keras,

Dataset of 50,000 32x32 color training images, labeled over 10 categories, and 10,000 test images.

- x_train, x_test: uint8 array of RGB image data with shape (num_samples, 3, 32, 32) or (num_samples, 32, 32, 3) based on the image_data_format backend setting of either channels_first or channels_last respectively.
- y_train, y_test: uint8 array of category labels (integers in range 0-9) with shape (num_samples,).

After importing dataset, dataset has been loaded and converting to greyscale and then resizing to 64*64 and then converted into binary. Then reshaping as required for using in model.

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

X_grey_train=tf.image.rgb_to_grayscale(x_train,name=None)
X_grey_test=tf.image.rgb_to_grayscale(x_test,name=None)

X_reshaped_train = tf.image.resize_images(X_grey_train,(64,64))
X_reshaped_test = tf.image.resize_images(X_grey_test,(64,64))

y_train_binary=list()
for i in range(len(y_train)):
 if y_train[i]==5:
  y_train_binary.append(1)
 elif y_train[i]!=5:
  y_train_binary.append(0)
y_test_binary=list()
for i in range(len(y_test)):
 if y_test[i]==5:
  y_test_binary.append(1)
 elif y_test[i]!=5:
  y_test_binary.append(0)
y_train_binary_array=np.asanyarray(y_train_binary)
type(y_train_binary_array)

y_test_binary_array=np.asanyarray(y_test_binary)
type(y_test_binary_array)

init=tf.global_variables_initializer()
sess=tf.Session()
sess.run(init)
X_reshaped_train=X_reshaped_train.eval(session=sess)
X_reshaped_test=X_reshaped_test.eval(session=sess)
```

Results analysis: - Transfer learning focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. In this problem statement I found out that Transfer learning didn’t affected a much in positive manner. Though the results are quite similar, but it doesn’t give a better result. This might be the case that as both datasets is not sufficient to learn the system to validate data more accurately. When we calculated F1 value which gave me high suspicion that recall is not doing good, but accuracy number and precision number are quite impressive.