UNIVERSITY OF TEXAS ARLINGTON

Programming Assignment 3: Using 2nd of 2 day quota

Student name: *Avinash Shanker, 1001668570*

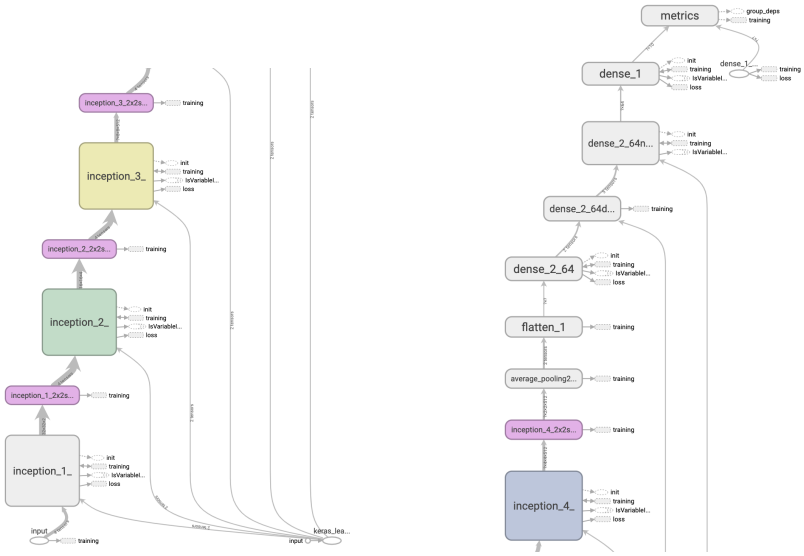
Due date: *Oct 15, 2019*

Inferences: Following points answered

- Describe the training process implemented in your code.
- Which loss function/optimization method is used?
- Is there any regularization used?
- Which evaluation metric is used and why? Like, top-k accuracy, average per-class accuracy, mean average precision (mAP).
- Contrast these metrics and discuss when they should be used.

(a) 10 Classes

Figure 1: CIFAR-10 dataset consists of 10 categories labeled as above



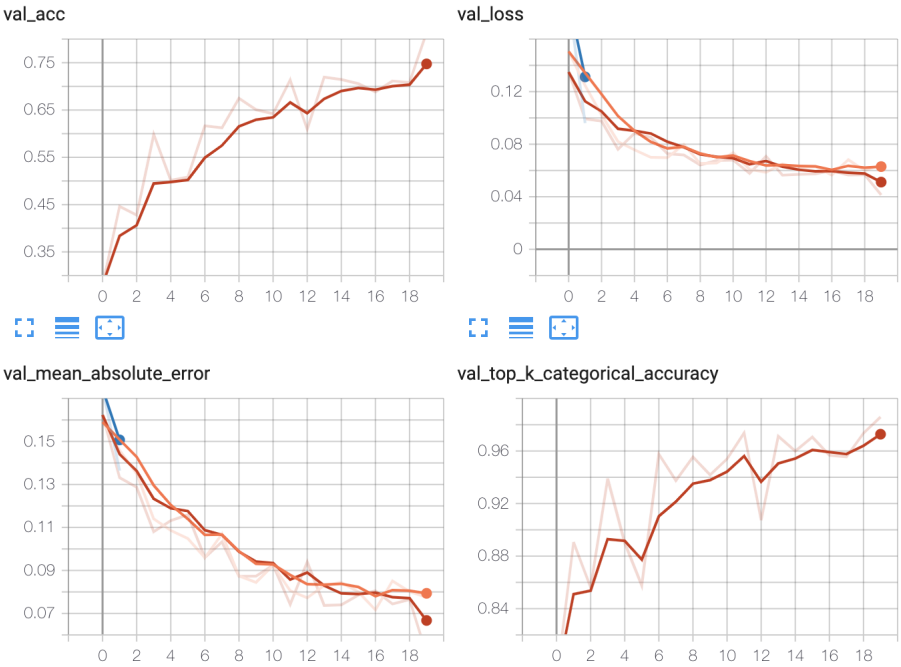
(a) Flow Graph (b) Flow Graph Cont.

Figure 2: My model is implemented using Google Inception Network. Graph generated has 4 inception layers which is obtained from Tensorboard-Graph of my implementation

Answer.

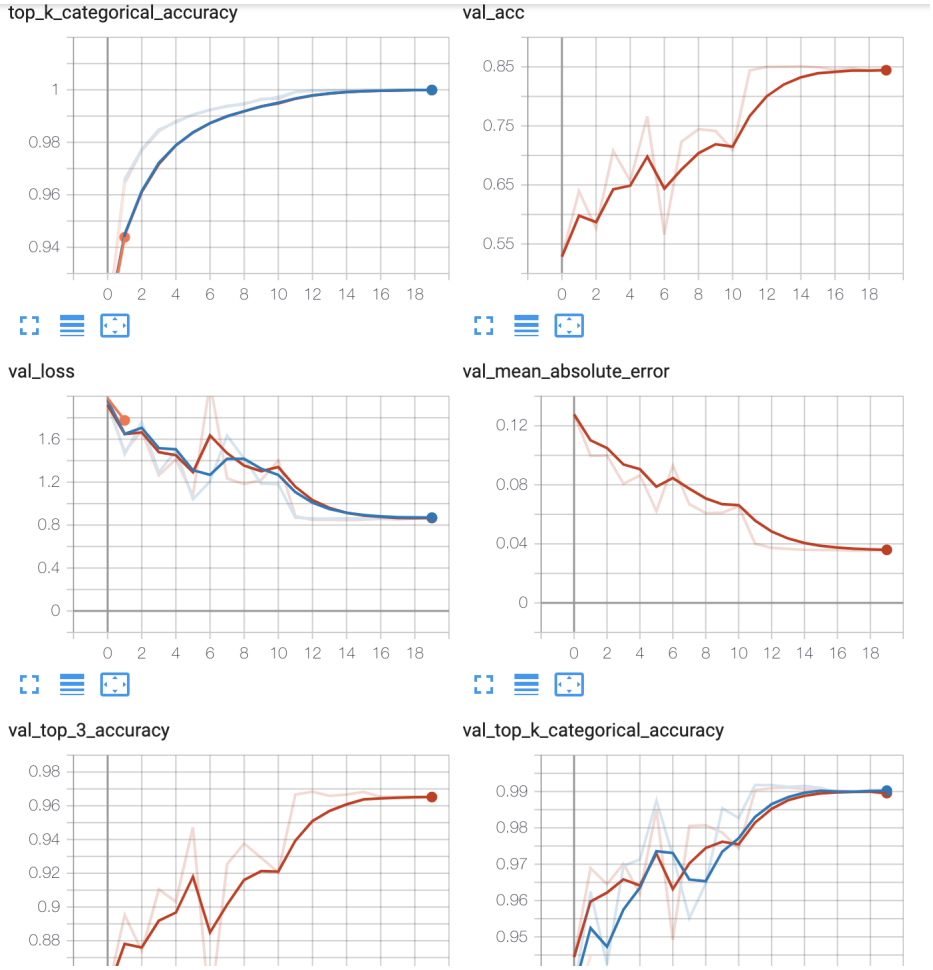
- Training process implemented in code by splitting the dataset into 50,000 training images and 10,000 test images of CIFAR-10 dataset. Shape of training dataset is x-train shape: (50000, 32, 32, 3)
- I have implemented two models to compare and contrast my work. First, simple 3 layer Convolution neural network(CNN) and second being, a 4 layer(Inception) Google Inception network.
- In Figure 1, inception layer is a combination of all those layers (1x1 Convolution layer, 3x3 Convolutions layer, 5x5 Convolutions layer) with their output filter banks concatenated into a single output vector forming the input of the next stage.
- Loss function used in both my models is categorical_crossentropy as we have 10 categories. I have tried binary_crossentropy as well. Though, loss functions had very close accuracy still categorical_crossentropy perfomed better. Also, its reccomended not to use binary for classes more than 2.
- Optimization, I have used Adam optimizer. As it is a Stochastic gradient descent optimizer which maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training.
- Regularization, I have tried both L1(Lasso Regression) and L2(Ridge Regression). But, L2 regularization provided better accuracy.
- I have compared model using top_k_categorical_accuracy, mean_absolute_error and Accuracy as metrices for my model. These have been captured in logs and also show in fig.

- Comparision to choose the suitable Loss function between categorical_crossentropy and mean_squared_error based on metrices accuracy, top_3_accuracy, top_k_categorical_accuracy, Mean Absolute Error to finalize.



(a) Validation metrices

Figure 3: Comparing metrices on Loss functions of mean_squared_error



(a) Validation metrices

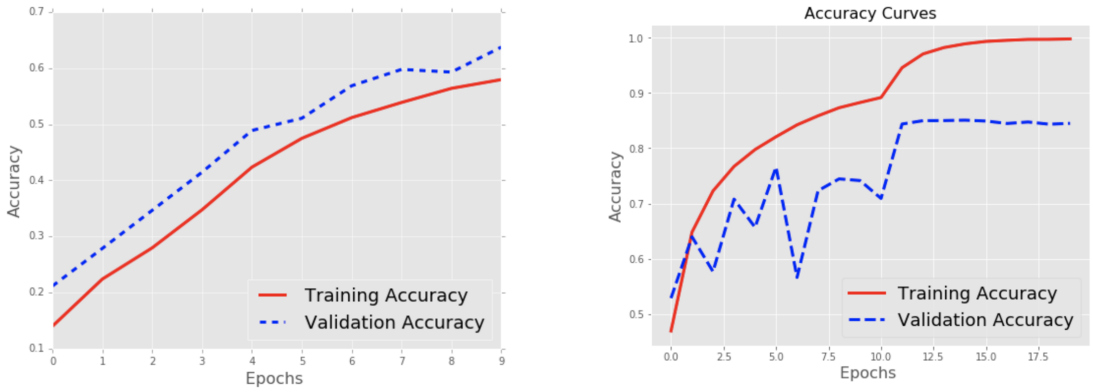
Figure 4: Comparing metrices on Loss functions of categorical_crossentropy

Comparison of Loss fn MSE and Categorical CrossEntropy		
Metric	Mean Squared Error Fig2	Cat CrossEntropy Fig3
Validation Accuracy	0.8135	0.8451
Validation Loss	0.04	0.8682
Mean Absolute Err	0.05	0.03548
Top k Categorical Accuracy	0.9859	0.9904

Explanation for choosing Categorical Cross entropy.

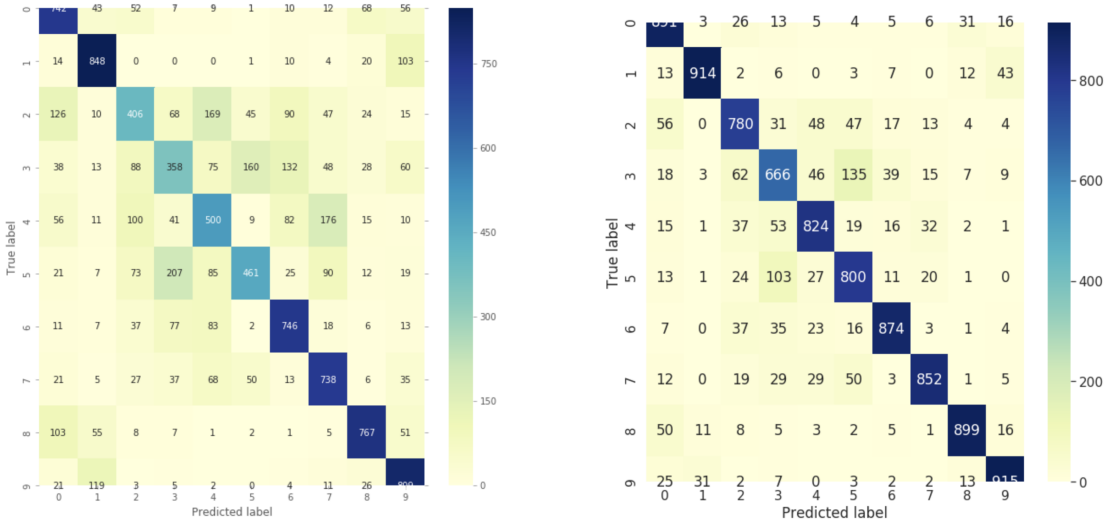
- By performing comparison based on mertices for loss function Mean Squared error and Categorical cross entropy.
- It can be inferred from graphs that, validation accuracy, mean absolute error and top-k categorical accuracy is best for loss function cross entropy than MSE.
- Hence, will be choosing Cross Entropy for both Simple MultiLayer CNN and Google Inception Network

- Comparison between a simple multi-layer CNN and multi-branch Google Inception Network. Decision to choose which model?



(a) 3 Layer Simple CNN (b) Google Inception network

Figure 5: Comparing Validation and Training accuracy of 3 Layer Simple CNN and multi-branch Google Inception network to choose a good model



(a) Confusion matrix for simple CNN (b) Confusion matrix for Inception network

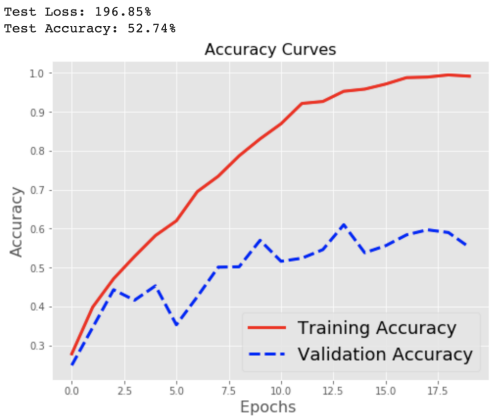
Figure 6: Comparison of confusion matrix between simple CNN and Google inception network. It can be clearly noticed that number of correct predictions(blue diagonal) is higher for 4layer Inception network

Explanation for choosing Multi-branch Google Inception Network over 3Layer CNN.

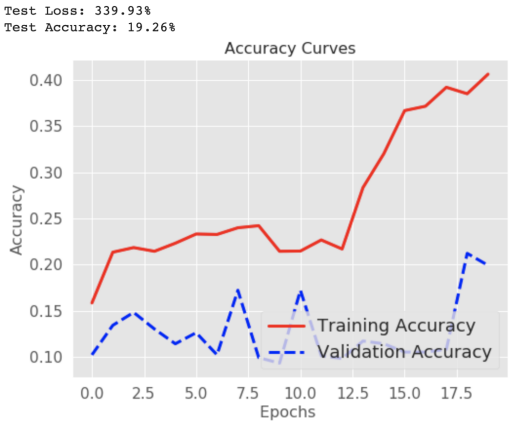
- Comparing the validation and training accuracy, simple 3 layer CNN has 64% validation accuracy. Whereas 4layer inception network has a validation accuracy of 84%
- Based on comparison of confusion matrix between simple CNN and Google inception network. It can be clearly noticed that number of correct predictions(blue diagonal) is higher for 4layer Inception network.

- Hence, from the above it can be inferred that Google Inception networks performs better.

- Discussion on adding residual connections and/or batch normalization.
- Results of multi-branch architecture in Google Inception networks.
- Results of tweaking hyper-parameters for training like learning rate, weight decay, training epochs.
- Justifying design of the model.
- Showing training curves and training/validation accuracy with best parameters.



(a) Without Batch normalization as input layers increases



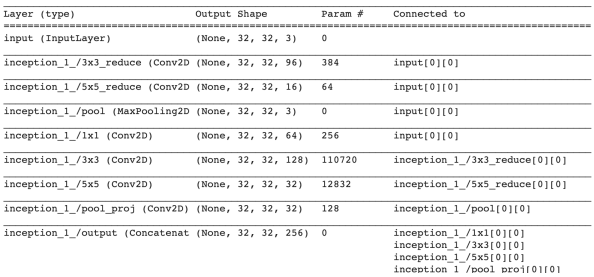
(b) Learning Rate 0.5

Figure 7: Tweaking Hyper-parameters, above are examples of not well optimised parameters which produce sluggish or low accuracy

Explanations and discussion for above.

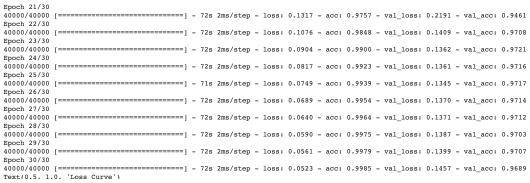
- For Adam optimizer when Learning rate is increased to 0.5, validation accuracy dropped to 19.26%. Whereas when accuracy was reset to a stable rate of 0.001 the validation accuracy increased to 84.8%. Hence, by trail and run LR of 0.001 suited best for my model.
- In my model when, batch normalization which was set as parameter to False, The validation accuracy dropped to 59.90.
- In my model, it was noticed that having low number of epoch, for epoch=5 the accuracy was 79.2%. But, the accuracy increased slowly as the number of epochs increased. For my code its set 20 epochs. Value greater that showed very less improvement in accuracy and highly processor intensive. This problem can also be called over fitting.
- My final model used is a multi branch google inception network. It has 4 inception layer. One layer consists of dimension inception1x1,inception3x3, inception5x5, inceptionPool convolution for faster and less expensive computation.

- I have used two activation function in my work, First for building layers. ReLu function and For Dense layer softmax function. And loss function is categorical_crossentropy optimizer is Adam(0.001) and finall metrics is accuracy.
- The best hyper-parameters presented the best output with a trade off between efficiency and accuracy is, Epoch=20, LearningRate=0.001, dropout=0.2, regularization=12
- The best case scenario in my model provided a accuracy of 84.5%



Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	(None, 32, 32, 3)	0	
inception_1_/3x3_reduce (Conv2D	(None, 32, 32, 96)	384	input[0][0]
inception_1_/5x5_reduce (Conv2D	(None, 32, 32, 16)	64	input[0][0]
inception_1_/pool (MaxPooling2D	(None, 32, 32, 3)	0	input[0][0]
inception_1_/1x1 (Conv2D)	(None, 32, 32, 64)	256	input[0][0]
inception_1_/3x3 (Conv2D)	(None, 32, 32, 128)	110720	inception_1_/3x3_reduce[0][0]
inception_1_/5x5 (Conv2D)	(None, 32, 32, 32)	12832	inception_1_/5x5_reduce[0][0]
inception_1_/pool_proj (Conv2D)	(None, 32, 32, 32)	128	inception_1_/pool[0][0]
inception_1_/output (Concatenat	(None, 32, 32, 256)	0	inception_1_/1x1[0][0] inception_1_/3x3[0][0] inception_1_/5x5[0][0] inception_1_/pool_proj[0][0]

(a) Layers created in inception model



```
Epoch 21/30
40000/40000 [==============================] - 72s 2ms/step - loss: 0.1317 - acc: 0.9757 - val_loss: 0.2191 - val_acc: 0.9461
Epoch 22/30
40000/40000 [==============================] - 72s 2ms/step - loss: 0.1076 - acc: 0.9848 - val_loss: 0.1409 - val_acc: 0.9708
Epoch 23/30
40000/40000 [==============================] - 72s 2ms/step - loss: 0.0904 - acc: 0.9900 - val_loss: 0.1362 - val_acc: 0.9721
Epoch 24/30
40000/40000 [==============================] - 72s 2ms/step - loss: 0.0817 - acc: 0.9923 - val_loss: 0.1361 - val_acc: 0.9716
Epoch 25/30
40000/40000 [==============================] - 71s 2ms/step - loss: 0.0749 - acc: 0.9939 - val_loss: 0.1345 - val_acc: 0.9717
Epoch 26/30
40000/40000 [==============================] - 72s 2ms/step - loss: 0.0689 - acc: 0.9954 - val_loss: 0.1370 - val_acc: 0.9714
Epoch 27/30
40000/40000 [==============================] - 72s 2ms/step - loss: 0.0640 - acc: 0.9964 - val_loss: 0.1371 - val_acc: 0.9712
Epoch 28/30
40000/40000 [==============================] - 72s 2ms/step - loss: 0.0590 - acc: 0.9975 - val_loss: 0.1387 - val_acc: 0.9703
Epoch 29/30
40000/40000 [==============================] - 72s 2ms/step - loss: 0.0561 - acc: 0.9979 - val_loss: 0.1399 - val_acc: 0.9707
Epoch 30/30
40000/40000 [==============================] - 72s 2ms/step - loss: 0.0523 - acc: 0.9985 - val_loss: 0.1457 - val_acc: 0.9689
Text(0.5, 1.0, 'Loss Curve')
```

(b) Batch running for 20 epochs

Figure 8: The above image shows the dimensions of convolution layers created in Google inception network

References.

- https://blog.stratospark.com/deep-learning-applied-food-classification-deep-learning-keras.html
- http://keras.io/optimizers/
- https://stackoverflow.com/questions/56851298/tensorboard-tensorflow-python-framework-errors-impl-notfounderror
- https://github.com/tensorflow/tensorboard/issues/2279
- https://seaborn.pydata.org/generated/seaborn.heatmap.html
- https://colab.research.google.com/github/Tzeny/cifar10/blob/master/Cifar10.ipynb
- https://www.programcreek.com/python/example/92956/keras.regularizers.l2
- http://faroit.com/keras-docs/2.0.2/getting-started/sequential-model-guide/
- https://adventuresinmachinelearning.com/keras-tutorial-cnn-11-lines/