1 IMAGE TRANSFORMATION

Following are the image transformations applied to the images present in the data-set.

1.1 Image Enhancement

Using the provided conversion formula for a single pixel of an image, we created a function enhance_Pixel() and then using the vectorize() method of the numpy library applied this operation over all the pixels of a image.

In the below example the maximum pixel is 250 and minimum is 0, therefore we can't see any changes reflected

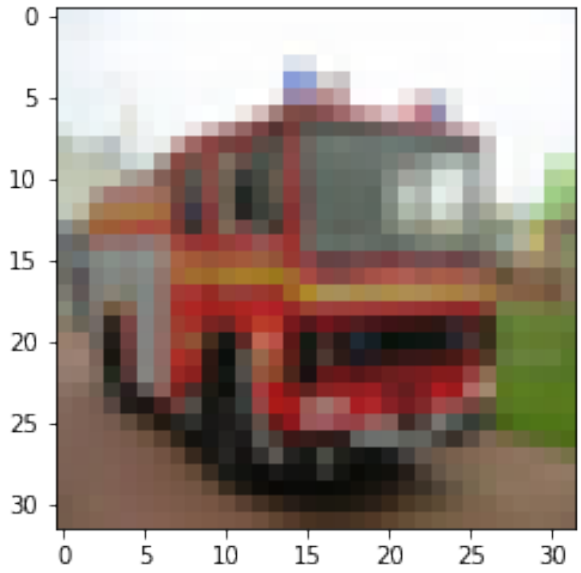


Figure 1: Original Image

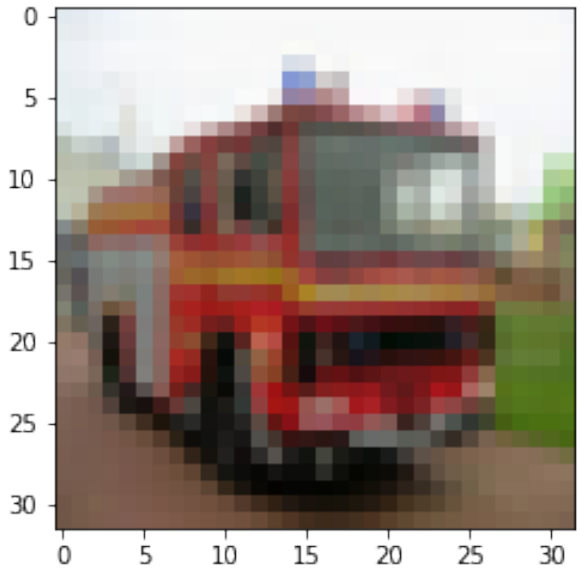


Figure 2: Converted Image

Figure 3: Image Enhancement

1.2 Posterization of Image

The value of the max is set to 200 and min to 50 for implementing posterization. Using the same method for enhancement operation, first a method for a single pixel is implemented and then vectorized over the whole image.

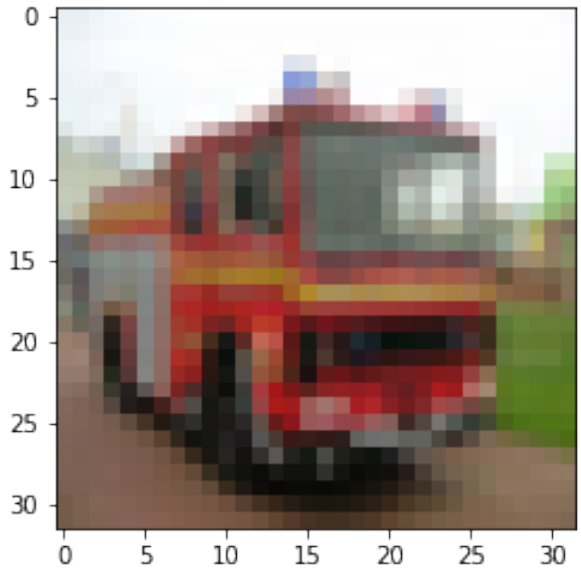


Figure 4: Original Image

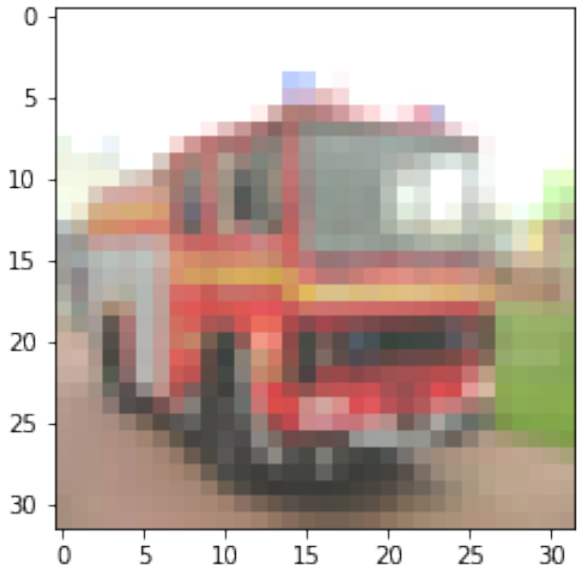


Figure 5: Converted Image

1.3 Random Rotate

To rotate an image by a certain angle, we chose the centre of rotation to the centre of image. Rotation matrix for an angle :

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

First we calculate the centre of the image, then for each pixel we find co-ordinates of pixel with respect to the centre of the image, then find the co-ordinate of pixel with respect to the rotated image by rotating them using the rotation matrix vector by the given angle gives a new position and then we add the pixel value to this new position (after making sure that the new position lies within the boundary of the image).

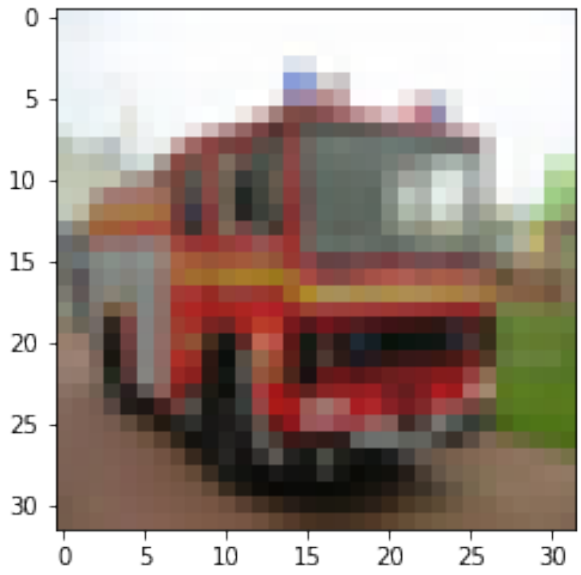


Figure 6: Original Image

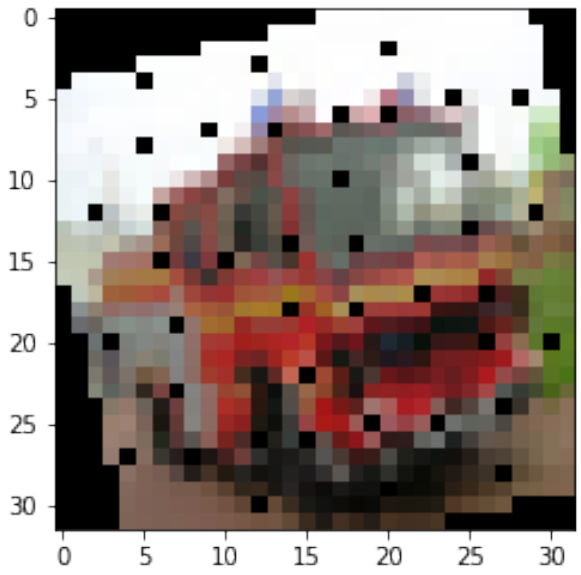


Figure 7: Converted Image

1.4 Contrast & Horizontal Flipping

The value of α is randomly generated between 0.5 and 2.0 and used to apply contrast to the image. Then the flipping operation is applied with a probability of 0.5 to the contrasted image.

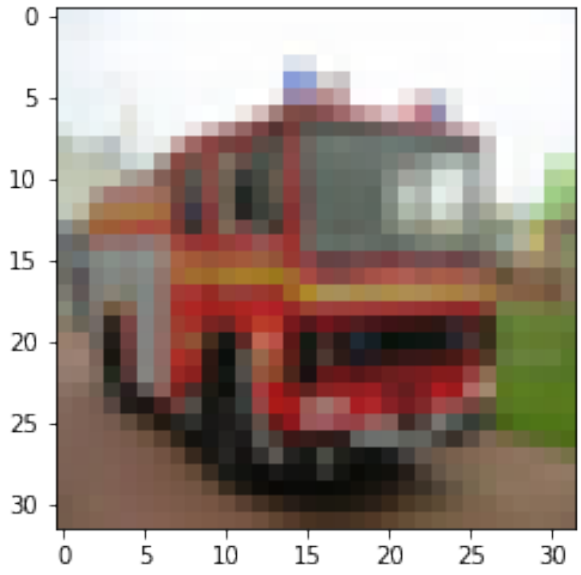


Figure 8: Original Image

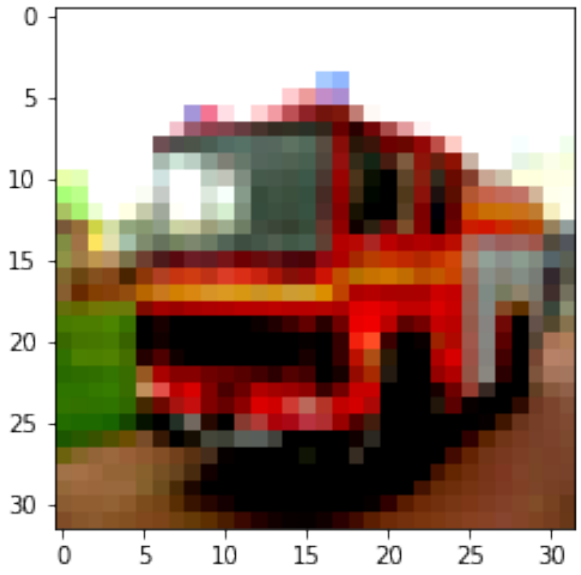
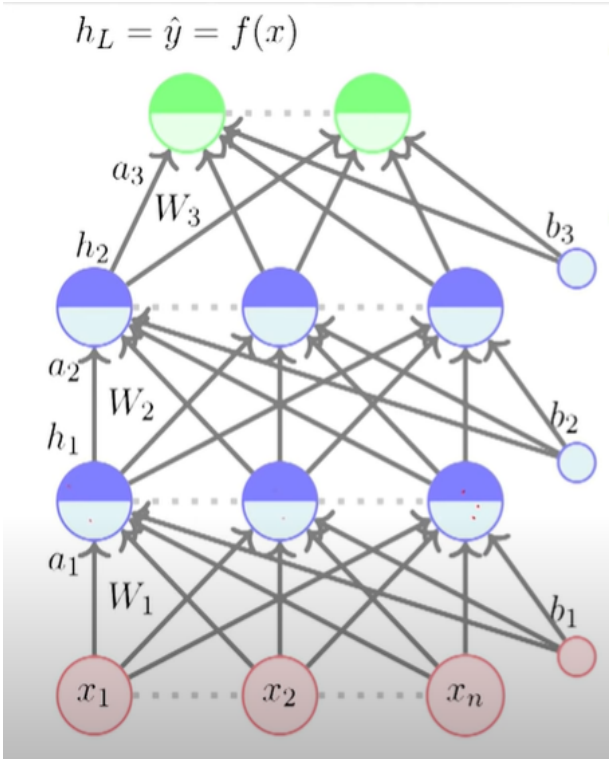


Figure 9: Converted Image

2 MLP IMPLEMENTATION

The input to the network is the vector image from the CI-FAR-10 data set, passed to the predefined feature extractor script. The vector belongs to R^{10} .The network contains 2 hidden layers containing 64 neurons each. The input layer is called the 0 layer and the output layer is called the L layer. The weights and biases for the first, second, and last layer are (W1 , B1) , (W2, B2) and (WL , BL) respectively.



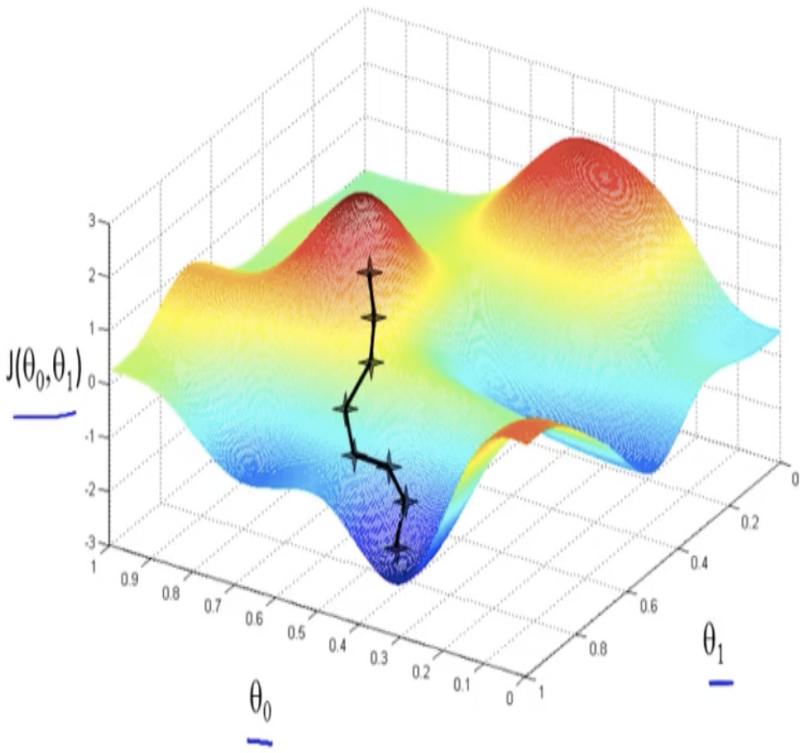
3 Output Function and Loss Function

The pre activation at the layer i is given by, $a^{(i)} = \mathbf{W}^{(i)} h^{(i-1)} + \mathbf{b}^{(i)}$

The activation at the layer i is given by, $h^{(i)} = g(\mathbf{a}^{(i)})$ where g is the activation function in order to bring non linearity to the model. In the above model relu activation function is used.

Activation at last output layer is different from the hidden layers activation function as in the image multi label classification a probability distribution is expected along all the outputs hence soft-max function is used
$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, 10.$

For training the network with respect to parameters w and b for all the layers gradient decent algorithm is used . Gradients of all the activation pre activation, weights and bias are calculated and then the model moves opposite to the gradient . The gradient decent algorithm is driven by a loss function , the loss function used is cross entropy. $\min_\theta \sum_y -\log(p(y;\theta))$



4 Required Gradient Quantities

- Gradient with respect to output units
- Gradient with respect to hidden units
- Gradient with respect to weights and biases

$$\underbrace{\frac{\partial \mathscr{L}(\theta)}{\partial W_{111}}}_{\text{Talk to the weight directly}} = \underbrace{\frac{\partial \mathscr{L}(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_3}}_{\text{Talk to the output layer}} \underbrace{\frac{\partial a_3}{\partial h_2} \frac{\partial h_2}{\partial a_2}}_{\text{Talk to the previous hidden layer}} \underbrace{\frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1}}_{\text{Talk to the previous hidden layer}} \underbrace{\frac{\partial a_1}{\partial W_{111}}}_{\text{and now talk to the weights}}$$

4.1 Gradient with respect to output units

In order to calculate gradients with respect to the output units ie al , we need to calculate with respect to the all the $\hat{y}$ i . as the loss function is $\frac{-1}{log(\hat{y}_l)}$ where $\hat{y}$ is the true prediction , hence it will be $\frac{-1}{log(\hat{y}_l)}$ only for $\hat{y}_l$ and rest zero . using the above information we can calculate the gradient with respect to al ie the output activation function as given below in the image.

$$\frac{\partial}{\partial a_{Li}} - log\widehat{yl} = \frac{-1}{\widehat{yl}} \frac{\partial}{\partial a_{Li}} \widehat{y}_l$$

$$= \frac{-1}{\widehat{yl}} \frac{\partial}{\partial aLi} softmax\,(a_L)_l$$

$$= \frac{-1}{\widehat{y}l} \frac{\partial}{\partial aL} \frac{e^{(a_L)}l}{\sum ie^{(aL)_i}}$$

$$= \frac{-1}{\widehat{y}l} \left(1(i=l)\widehat{y}l - \widehat{y}l\widehat{y}i\right)$$

$$= -\left(1_{(i=l)} - f\left(x\right)_i\right) \{here\widehat{y} = f\left(x\right)\}$$

$$\nabla a_L = -\left(e\left(l\right) - f\left(x\right)\right)$$

```
#compute gradeint w.r.t respect to output
grad_al = -(one_hot_Y-Yhat)
```

4.2 Gradient with respect to hidden units

To calculate the gradients for hidden units we need to calculate it with respect to the h_i ie all the activataion functions. as the activation function is connected to output function throgh all the preactivations of the next layers , we find the partial derivates with all the next layer preactivations and the sum them up.

$$\frac{\partial L\left(\theta\right)}{\partial h_{ij}} = \left(W_{i+1}, \cdot j\right)^T \nabla a_{i+1}, L\left(\theta\right)$$

$$\nabla h_i L\left(\theta\right) = \begin{bmatrix} \frac{\partial L\left(\theta\right)}{\partial h_{i1}} \\ \frac{\partial L\left(\theta\right)}{\partial hiz} \\ . \\ . \\ \frac{\partial L\left(\theta\right)}{\partial hi_{10}} \end{bmatrix} \begin{bmatrix} \left(W_{i+1}, .1\right)^T \nabla a_{i+1} \\ \left(W_{i+1}, \cdot 2\right)^T \nabla a_i + 1 \\ . \\ . \\ \left(W_{i+1}, \cdot 10\right)^T \nabla a_i + 1 \end{bmatrix}$$

$$= \left(W_{i+1}\right)^T \left(\nabla a_{i+1} L\left(\theta\right)\right)$$

$$\nabla a_i L\left(\theta\right) = \nabla h_i L\left(0\right) \odot \left[derivReLu\left(a_i\right)\right]$$

$$\frac{\partial L(\theta)}{\partial h_{ij}} = \sum_{m=1}^{64} \frac{\partial L(\theta)}{\partial a_{i+1,m}} + \frac{\partial a_{i+1,m}}{\partial h_{ij}}$$

which gives the following compact equation:

```
#compute gradients w.r.t respect to layer 2(activation)
grad_h2 = WL.transpose().dot(grad_al)
#compute gradients w.r.t respect to layer 2(pre-activation)
grad_a2 = numpy.multiply(grad_h2,deriv_reLU(a2))

#compute gradientes w.r.t respect to layer 1(activation)
grad_h1 = W2.transpose().dot(grad_a2)
#compute gradientes w.r.t respect to layer 1(pre-activation)
grad_a1 = numpy.multiply(grad_h1,deriv_reLU(a1))s
```

4.3 Graadient with respect to weights and biases

As we have calculted the gradeints with respect to the pre activation functions we can use the same to calculate gradients with respect to weights and biases . as in the above chain rule we have gradients with respect to pre activation . now we find gradeints with respect to weights for preactivation and club both.

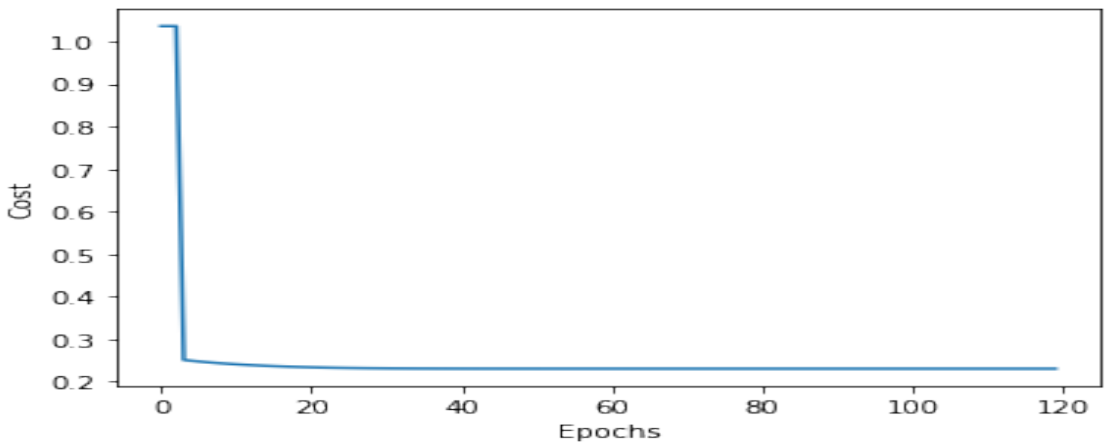
$$\nabla W_k = \frac{\partial L(\theta)}{\partial a_{ki}} \frac{\partial a_{ki}}{\partial W_{k,i,j}}$$

$$\nabla b_k = \nabla a_k$$

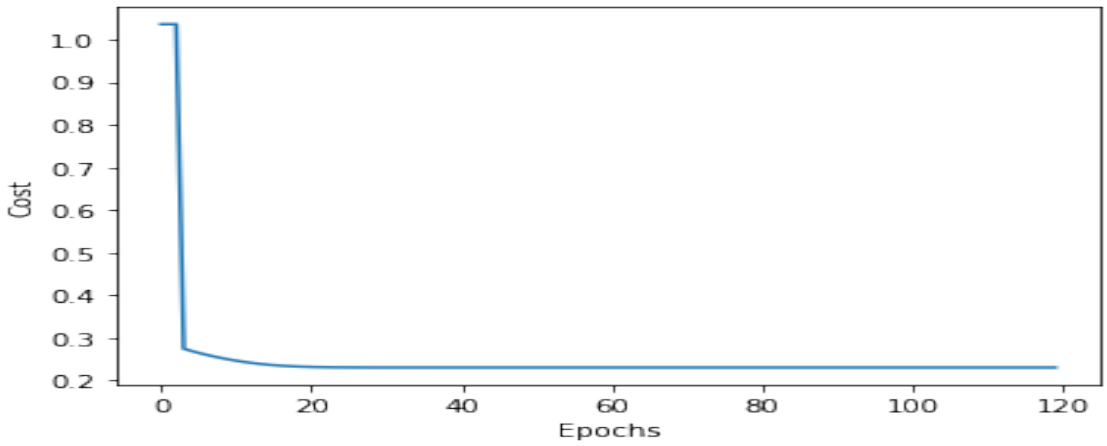
```
#output layer
grad_WL = grad_al.dot( h2.transpose() )
grad_bl = numpy.sum(grad_al,axis=1)

#2nd hidden layer
grad_W2 = grad_a2.dot(h1.transpose())
grad_b2 = numpy.array(numpy.sum(grad_a2,axis=1))

#first hidden layer
grad_W1 = grad_a1.dot(X.T)
grad_b1 = numpy.array(numpy.sum(grad_a1,axis=1))
```

5 Peformance and Comparison with other classifiers

Starting with epoch = 200 and learning rate = 0.1, instead of converging the loss started to bounce. Hence by decreasing the learning rate exponentially to 0.001, the cost stared tends to stabilize after which to achieve more accuracy the final learning rate finalized was 0.00002. After analyzing the loss graph more it was seen that after 120 epochs the loss function stabilized and a very small decrease was seen hence 120 epochs were finalized. //
Epoch vs cost for original training dataset:



Epoch vs cost for augmented training dataset:



5.1 MLP

Original Training Set	Original Testing Set	10 %
	Augmented Testing Set	10 %
Augmented Testing Set	Original Testing Set	10 %
	Augmented Testing Set	10 %

5.2 Logistic Regression Classifier

Original Training Set	Original Testing Set	30 %
	Augmented Testing Set	37.2 %
Augmented Testing Set	Original Testing Set	32 %
	Augmented Testing Set	37.7 %

5.3 KNN Classifier

Original Training Set	Original Testing Set	10 %
	Augmented Testing Set	31.2 %
Augmented Testing Set	Original Testing Set	30 %
	Augmented Testing Set	32.6 %

5.4 SVM Classifier

Original Training Set	Original Testing Set	33.2 %
	Augmented Testing Set	38.6 %
Augmented Testing Set	Original Testing Set	40.3 %
	Augmented Testing Set	42.7 %

5.5 Decision Tree Classifier

Original Training Set	Original Testing Set	18.6 %
	Augmented Testing Set	22.8 %
Augmented Testing Set	Original Testing Set	20.2 %
	Augmented Testing Set	23.7 %