Natural Image Processing
(PROJECT WORK)

Rohit B19EE098

Jai Arora B19EE094

Yogesh Nema B19EE092

Abstract— Here we have a Dataset which consists of 60000 images which are 32 x 32 pixels size and all these are made up of three colours that are red green and blue with the values lying between 0 to 256. Here we have in dataset 10 columns and these are different features according to different images.

I. INTRODUCTION

One of the way to classify images is Convolution Neural Network . A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

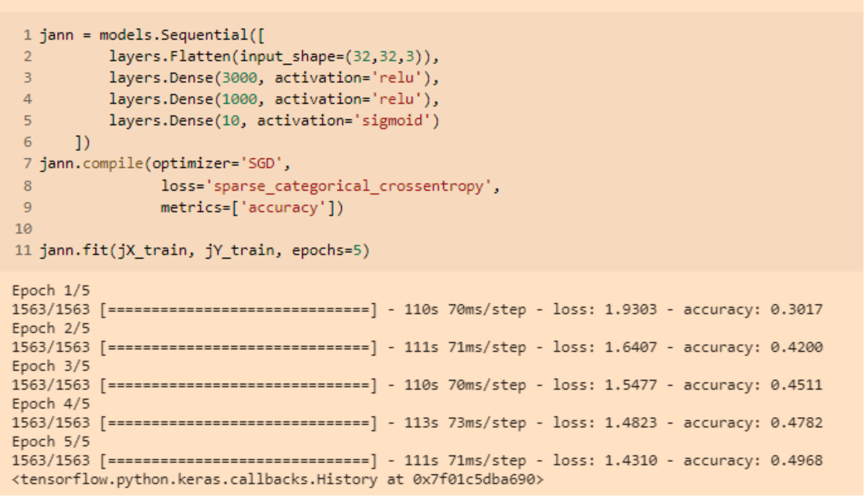
The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

In statistics, the logistic model (or logit model) is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This can be extended to model several classes of events such as determining whether an image contains a cat, dog, lion, etc. Each object being detected in the image would be assigned a probability between 0 and 1, with a sum of one.

The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems.
In KNN(k-nearest neighbors) we use KNeighborsClassifier set parameter as leafsize=20 metrics type minkowski and n_neighbour =40
We also apply ensemble learning and cross validation techniques to get good results .

FULL ANALYSIS

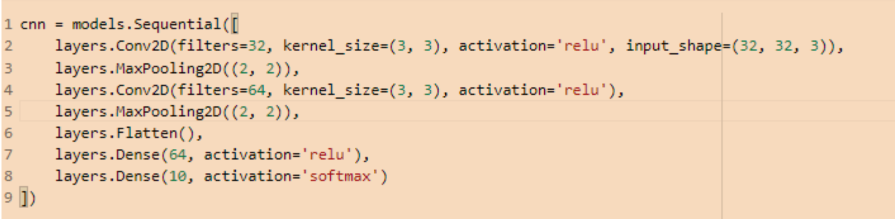
Firstly we are here to discuss the Results from our CNN Model



```
jann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='sigmoid')
    ])
jann.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

jann.fit(jX_train, jY_train, epochs=5)
```

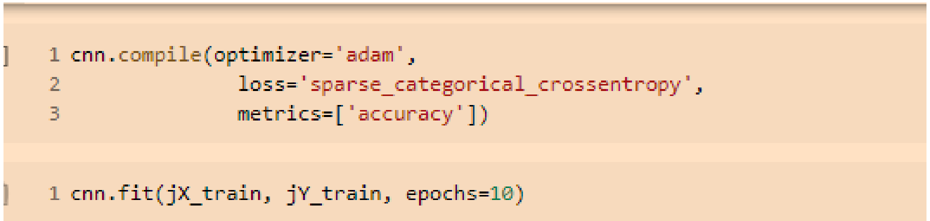
```
Epoch 1/5
1563/1563 [==============================] - 110s 70ms/step - loss: 1.9303 - accuracy: 0.3017
Epoch 2/5
1563/1563 [==============================] - 111s 71ms/step - loss: 1.6407 - accuracy: 0.4200
Epoch 3/5
1563/1563 [==============================] - 110s 70ms/step - loss: 1.5477 - accuracy: 0.4511
Epoch 4/5
1563/1563 [==============================] - 113s 73ms/step - loss: 1.4823 - accuracy: 0.4782
Epoch 5/5
1563/1563 [==============================] - 111s 71ms/step - loss: 1.4310 - accuracy: 0.4968
<tensorflow.python.keras.callbacks.History at 0x7f01c5dba690>
```

Here we have our ANN Model and we got the accuracy of 49%.



```
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

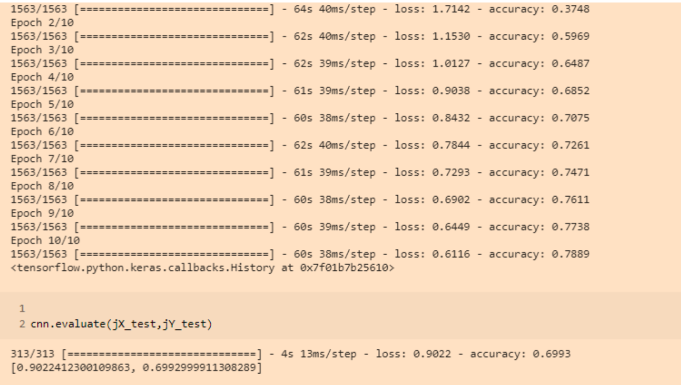
Here we implemented the CNN model with activation function RELU and on addition we have done the convolution part and max pooling to get the max value on 2 X 2 matrix. Than we have used 64 Neurons and with another such functions we have implemented the CNN Model.



```
cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
cnn.fit(jX_train, jY_train, epochs=10)
```

Here we have trained the model with training data.

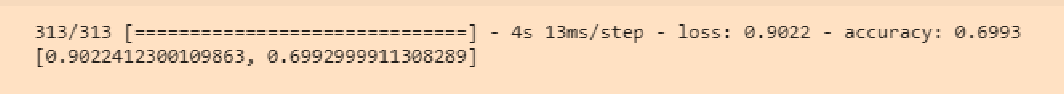


```
1563/1563 [==============================] - 64s 40ms/step - loss: 1.7142 - accuracy: 0.3748
Epoch 2/10
1563/1563 [==============================] - 62s 40ms/step - loss: 1.1530 - accuracy: 0.5969
Epoch 3/10
1563/1563 [==============================] - 62s 39ms/step - loss: 1.0127 - accuracy: 0.6487
Epoch 4/10
1563/1563 [==============================] - 61s 39ms/step - loss: 0.9038 - accuracy: 0.6852
Epoch 5/10
1563/1563 [==============================] - 60s 38ms/step - loss: 0.8432 - accuracy: 0.7075
Epoch 6/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.7844 - accuracy: 0.7261
Epoch 7/10
1563/1563 [==============================] - 61s 39ms/step - loss: 0.7293 - accuracy: 0.7471
Epoch 8/10
1563/1563 [==============================] - 60s 38ms/step - loss: 0.6902 - accuracy: 0.7611
Epoch 9/10
1563/1563 [==============================] - 60s 39ms/step - loss: 0.6449 - accuracy: 0.7738
Epoch 10/10
1563/1563 [==============================] - 60s 38ms/step - loss: 0.6116 - accuracy: 0.7889
<tensorflow.python.keras.callbacks.History at 0x7f01b7b25610>
```

```

cnn.evaluate(jX_test,jY_test)
```

```
313/313 [==============================] - 4s 13ms/step - loss: 0.9022 - accuracy: 0.6993
[0.9022412300109863, 0.6992999911308289]
```

Got the Accuracies at different Epochs.



```
313/313 [==============================] - 4s 13ms/step - loss: 0.9022 - accuracy: 0.6993
[0.9022412300109863, 0.6992999911308289]
```

Finally The accuracy is 69%.

KNN Analysis

Analysis of KNN data set
First we split the dataset into train and test
Then we apply some preprocessing step like scale the data and reshape the data
Then we apply knn function on data

```
# Create a kNN classifier instance.
from sklearn.neighbors import KNeighborsClassifier
# the Classifier simply remembers the data and does no further processing
classifier = KNeighborsClassifier(algorithm='auto', leaf_size=20, metric='minkowski',n_neighbors=4)
#fit the model
classifier.fit(X_train, y_train)
```

And then predict the y_test

```
accuracy_score(y_tes, y_pred)
```

I got accuracy as 0.27

After apply cross validation

```
array([0.267, 0.258, 0.275, 0.294, 0.266])
```

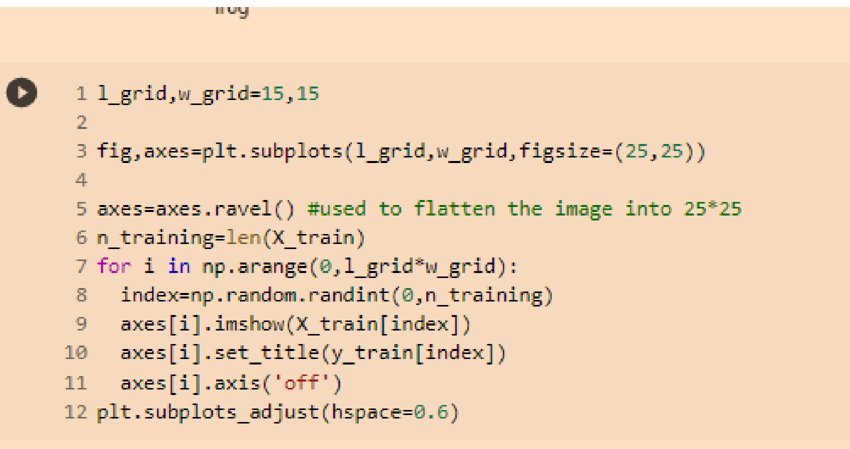
For different value of k i got this array

```
ff=[]
for k in range(1,100,5):
classifier = KNeighborsClassifier(n_neighbors=k)
classifier.fit(X_train, y_train)
y_predq = classifier.predict(X_test)
ff.append(accuracy_score(y_tes, y_predq))
print(ff)
```

[0.274, 0.282, 0.262, 0.268, 0.266, 0.266, 0.246, 0.252, 0.252, 0.246, 0.24, 0.238, 0.242, 0.236, 0.24, 0.226, 0.228, 0.222, 0.216, 0.222]

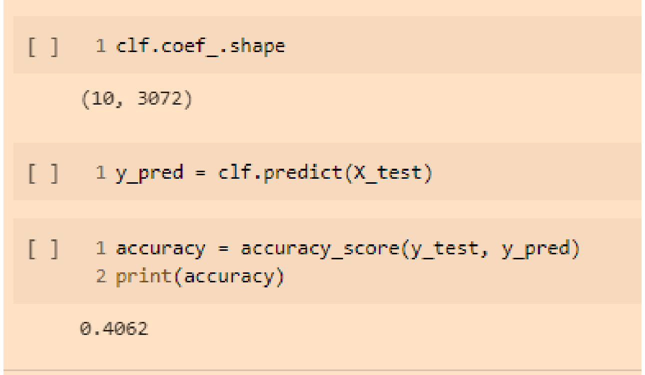
27 % accuracy from KNN Model.

Now Results From Logistics Regression Model



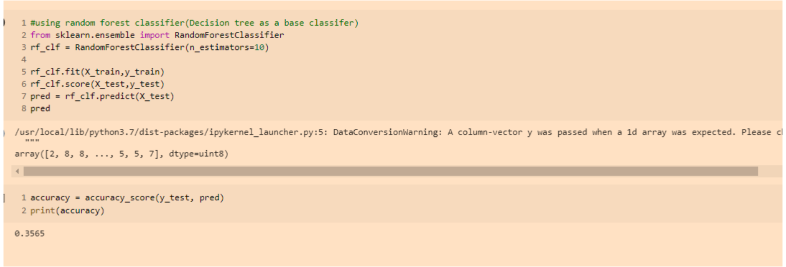
```
l_grid,w_grid=15,15

fig,axes=plt.subplots(l_grid,w_grid,figsize=(25,25))

axes=axes.ravel() #used to flatten the image into 25*25
n_training=len(X_train)
for i in np.arange(0,l_grid*w_grid):
  index=np.random.randint(0,n_training)
  axes[i].imshow(X_train[index])
  axes[i].set_title(y_train[index])
  axes[i].axis('off')
plt.subplots_adjust(hspace=0.6)
```

Training the Logistics Regression Model



```
[ ]  clf.coef_.shape

     (10, 3072)

[ ]  y_pred = clf.predict(X_test)

[ ]  accuracy = accuracy_score(y_test, y_pred)
     print(accuracy)

     0.4062
```

Got the 40% accuracy.



```
#using random forest classifier(Decision tree as a base classifer)
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(n_estimators=10)

rf_clf.fit(X_train,y_train)
rf_clf.score(X_test,y_test)
pred = rf_clf.predict(X_test)
pred

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please cl
  """
array([2, 8, 8, ..., 5, 5, 7], dtype=uint8)

accuracy = accuracy_score(y_test, pred)
print(accuracy)

0.3565
```

35% accuracy in Random Forest.