AMRITA VISHWA VIDYAPEETHAM

Project Name: Food Classification From Images

Subject Name: Deep Learning and Neural Networks
Subject Code: 15CSE380

IDLI

Project By:

M.M.Shobandri(BL.EN.U4CSE17076)

N.Prudhvi (BL.EN.U4CSE17092)

G.Sreekanth (BL.EN.U4CSE17034)

G.Danussh(BL.EN.U4CSE17032)

ABSTRACT:

Diet is very important in human life. Obtaining adequate nutrition from everyday meals is essential for our health.

To know what we eat, we often make a record of everyday meals. Such food recording is usually a manual exercise using textual description, but manual recording is tedious and time consuming. To overcome this difficulty, there have been attempts to assist food recording by using information technology. Image recognition of food items would be a good solution to food recording. Taking a picture would then be a sufficient record.

So our Food image classifier helps in differentiating food items and classifying it.

- We can upload our Image of Food and this application gives us the food item name.
- The product is mainly a Image classifying using Deep Learning concepts.

Our project include 4 food classes:

1. Apple pie
2. French fries
3. Chicken curry
4. Fried rice

We Implemented our project using CNN

The convolutional neural network (CNN) is a class of deep learning neural networks. CNNs represent a huge breakthrough in image recognition. They're most commonly used to analyze visual imagery and are frequently working behind the scenes in image classification. A CNN convolves (not convolutes...) learned features with input data and uses 2D convolutional layers. This means that this type of network is ideal for processing 2D images.

A CNN works by extracting features from images. This eliminates the need for manual feature extraction. The features are not trained! They're learned while the network trains on a set of images. This makes deep learning models extremely accurate for computer vision tasks.

```
model = Sequential()
```

#created a model in sequential class with stack of layers

```
model.add(Conv2D(filters = 32, kernel_size = (5,5), strides = 2, padding = 'Same', activation ='relu', input_shape = (224,224,3), kernel_initializer='he_normal'))
```

#Input layer with kernel size 5*5 and 32 features are extracted. The amount of movement between applications of the filter to the input image is referred to as the stride, Different size filters will detect different sized features in the input image and in turn will result in differently sized feature maps.

Padding preserve spatial dimensions of the volume such that the output volume size matches the input volume size, by setting the value to the "same".
Relu activation function is used.
Kernel initializer :This parameter controls the initialization method which is used to initialize all the values in the Conv2D class before actually training the model.

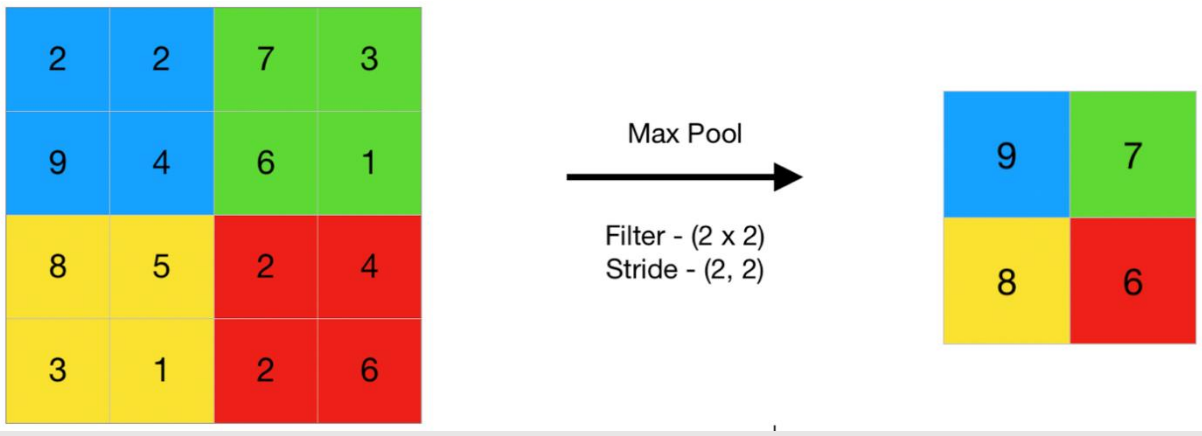
Initializers allow you to pre-specify an initialization strategy, encoded in the Initializer object, without knowing the shape and dtype of the variable being initialized.

Input shape consists image parameters: height,width,depth as (224,224,3)

```
model.add(Conv2D(filters = 32, kernel_size = (5,5), strides = 2, paddin
g = 'Same', activation ='relu',kernel_initializer='he_normal'))
model.add(MaxPool2D(pool_size=(2,2)))
```

#Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

Implementation:



```
model.add(Dropout(0.2))
```

#Dropout: A Simple Way to Prevent Neural Networks from Overfitting,it refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random.

```
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same', ac
tivation ='relu',kernel_initializer='he_normal'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same', ac
tivation ='relu',kernel_initializer='he_normal'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.2))
model.add(Conv2D(filters = 128, kernel_size = (2,2),padding = 'Same', a
ctivation ='relu',kernel_initializer='he_normal'))
model.add(Conv2D(filters = 128, kernel_size = (2,2),padding = 'Same', a
ctivation ='relu',kernel_initializer='he_normal'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.2))
model.add(Conv2D(filters = 256, kernel_size = (2,2),padding = 'Same', a
ctivation ='relu',kernel_initializer='he_normal'))
model.add(Conv2D(filters = 256, kernel_size = (2,2),padding = 'Same', a
ctivation ='relu',kernel_initializer='he_normal'))
```

```
model.add(GlobalAveragePooling2D())
```

Global Average Pooling is an operation that calculates the average output of each feature map in the previous layer. This fairly simple operation reduces the data significantly and prepares the model for the final classification layer. It also has no trainable parameters – just like Max Pooling
the elimination of all these trainable parameters also reduces the tendency of over-fitting, which needs to be managed in fully connected layers by the use of dropout.

```
model.add(Dense(512, activation = "relu",
kernel_initializer='he_normal'))
```

Dense layer is the regular deeply connected neural network layer. It is most common and frequently used layer. Dense layer does the below operation on the input and return the output.

```
output = activation(dot(input, kernel) + bias)
```

```
model.add(Dropout(0.2))
```

```
model.add(Dense(4, activation = "softmax"
,kernel_initializer='he_normal',kernel_re
gularizer=l2()))
```

Softmax activation is basically the normalized exponential probability of class observations represented as neuron activations. In a multi-class classification, ‘Cross-Entropy’ is cut out for usage along with Softmax.

```
model.compile(optimizer = 'Adam' , loss =
 "categorical_crossentropy", metrics=["ac
curacy"])
```

#compiled the network with adam optimizer

Adam is an adaptive learning rate **optimization** algorithm that's been designed specifically for training deep neural networks. ... The algorithms leverages the power of adaptive learning rates methods to find individual learning rates for each parameter

It is a multiclass problem, you have to use categorical_crossentropy. Also labels need to converted into the categorical format.

Accuracy is the proportion of true results among the total number of cases examined

```
history = model.fit_generator(train_gener
ator,steps_per_epoch=3000/16,
                              validation_
data=test_generator,validation_steps=1000
/16,
                              epochs=40,
callbacks=[checkpointer, reduceLR, earlys
topping])
```

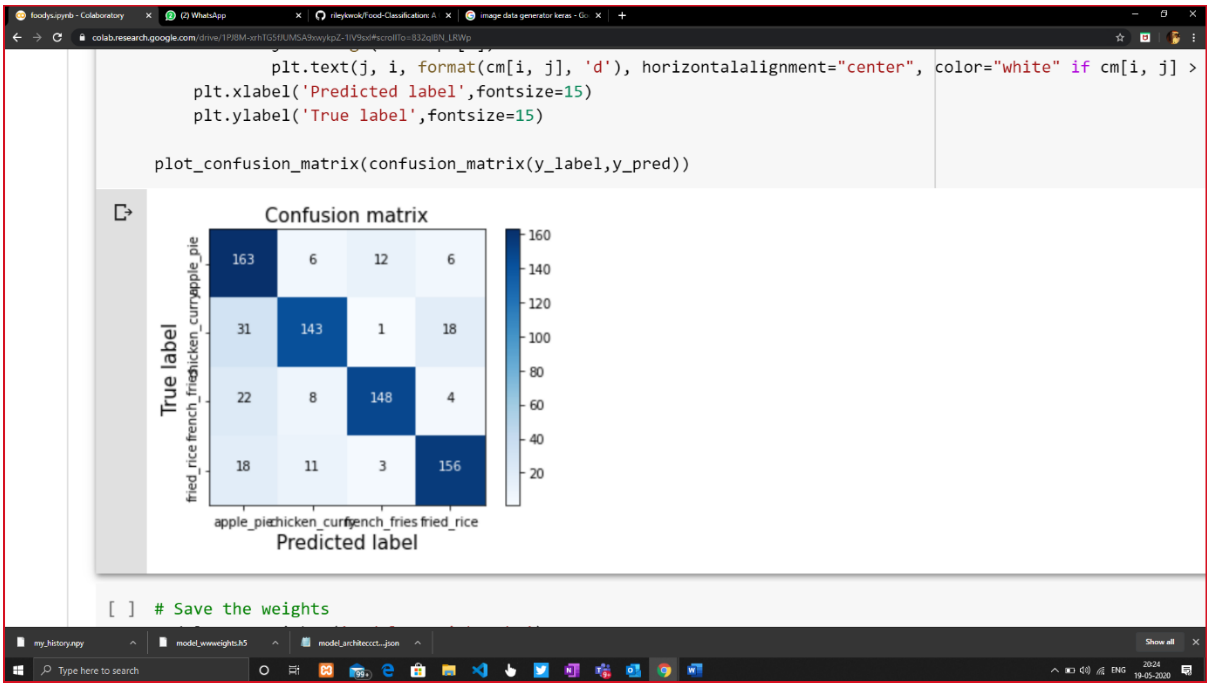
callbacks Periodically save your model to disk

The checkpoint may be used directly, or used as the starting point for a new run, picking up where it left off.

learning rate to be reduced when training is not progressing.

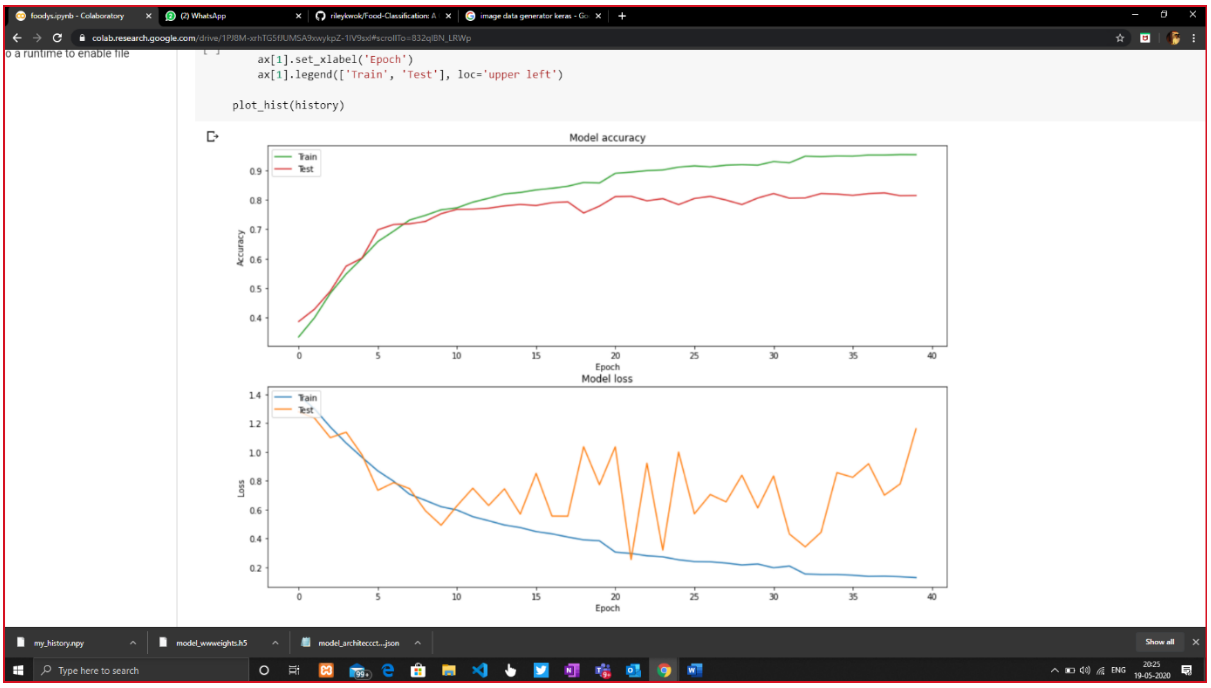
Early stopping is basically stopping the training once your loss starts to increase (or in other words validation accuracy starts to decrease).

steps_per_epoch
value as the total number of training data points divided by the batch size. Once Keras hits this step count it knows that it’s a new epoch.

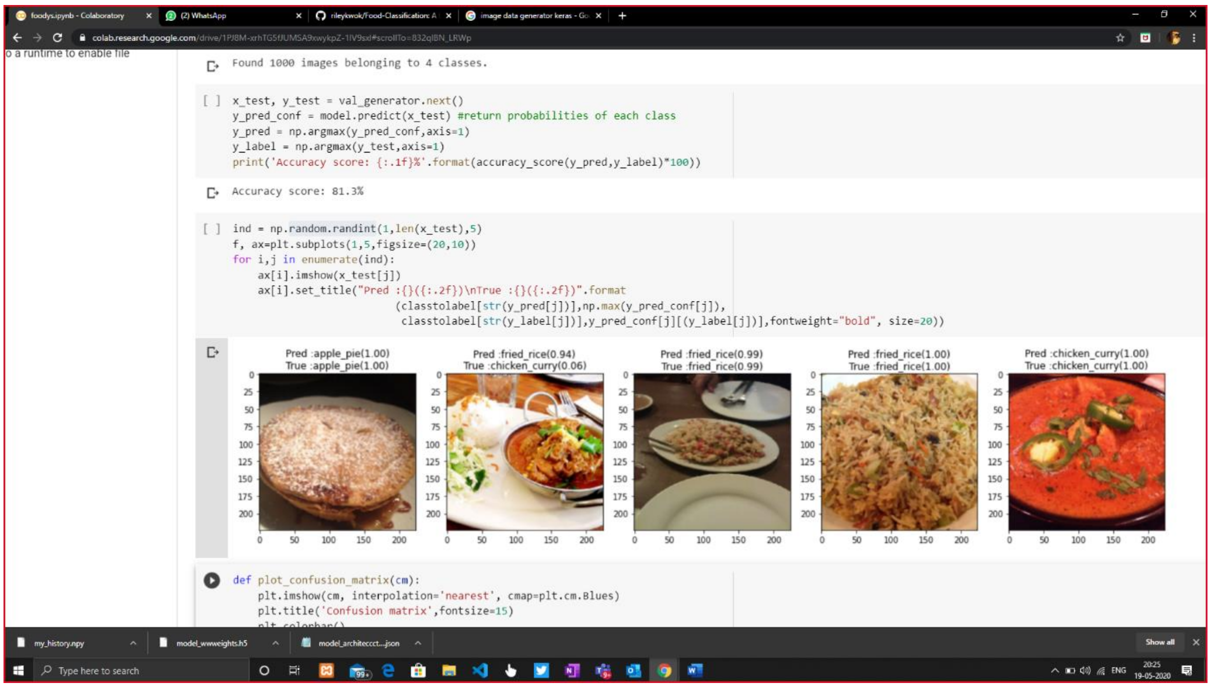


```
            plt.text(j, i, format(cm[i, j], 'd'), horizontalalignment="center", color="white" if cm[i, j] >
    plt.xlabel('Predicted label',fontsize=15)
    plt.ylabel('True label',fontsize=15)

plot_confusion_matrix(confusion_matrix(y_label,y_pred))
```

```
[ ] # Save the weights
```



```
    ax[1].set_xlabel('Epoch')
    ax[1].legend(['Train', 'Test'], loc='upper left')

plot_hist(history)
```



```
Found 1000 images belonging to 4 classes.
```

```
x_test, y_test = val_generator.next()
y_pred_conf = model.predict(x_test) #return probabilities of each class
y_pred = np.argmax(y_pred_conf,axis=1)
y_label = np.argmax(y_test,axis=1)
print('Accuracy score: {:.1f}%'.format(accuracy_score(y_pred,y_label)*100))
```

```
Accuracy score: 81.3%
```

```
ind = np.random.randint(1,len(x_test),5)
f, ax=plt.subplots(1,5,figsize=(20,10))
for i,j in enumerate(ind):
    ax[i].imshow(x_test[j])
    ax[i].set_title("Pred :{}({:.2f})\nTrue :{}({:.2f})".format
                        (classtolabel[str(y_pred[j])],np.max(y_pred_conf[j]),
                         classtolabel[str(y_label[j])],y_pred_conf[j][(y_label[j])],fontweight="bold", size=20))
```

```
def plot_confusion_matrix(cm):
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Confusion matrix',fontsize=15)
```

Tools:

Keras

Mat.lib

Dataset:

We are using food101 dataset present in Kaggle

Python libraries used:

1. Numpy
2. Matplotlib
3. Sklearn
4. Keras
5. Mlextend

Modules used

`random.randint()`

`randomly choosing 5 images for testing`

`Imagedatagenerator()`

It is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset

`Plt.subplots()`

Used for plotting graph with required parameters

Problems Encountered

Problem: overfitting problem

Solution: Inorder to overcome this problem we used dropout layer

Problem: Accuracy was less initially
Solution: feature extraction is increased

Result

The proposed work is to build a food image classifier
To know what we eat, we often make a record of everyday meals. Such food recording is usually a manual exercise using textual description, but manual recording is tedious and time consuming. To overcome this difficulty, there have been attempts to assist food recording by using information technology.

Contibution of each team member:

G.Danuushh :
Collecting the data ,dividing into test and train and analysizing the image size required for training process

M.M.Shobanadri
Generating similar images (shear,rotating,zoom,flip)-actions
Creating convolutional network layers

N.prudhvi
Compiling
Training
Testing

G.sreekanth
Json architecture building
Classification report generation
Generating accuracy and loss graphs
Confusion matrix

Scope of future work

The CNN is capable of train highly non-linear data, and for that incontrast, it takes more computational time to train the network. However, the performance matters a lot, and once the system is properly trained, the system can produce the results in less time. The images are properly preprocessed and all kinds of images are tested with CNN

A multi-level classification approach (hierarchical approach) is suitable to avoid mis-classifications when the number of classes is more.