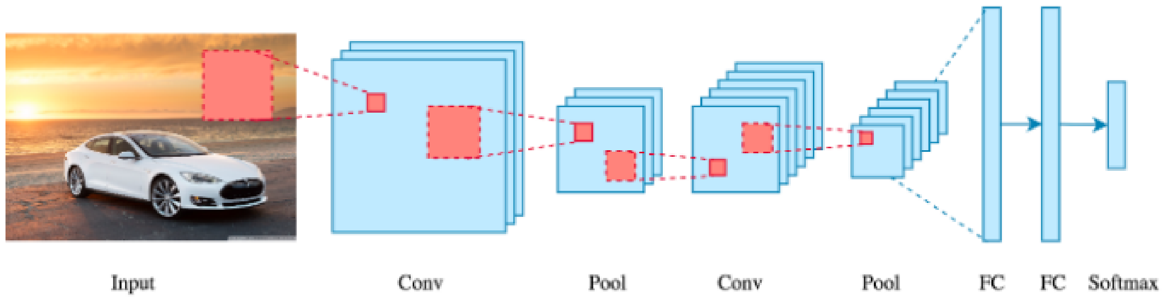
Question 2: Classification : Convolutional Neural Networks

CNN

CNN is now the go-to model for any problem related to the images. CNN's biggest benefit compared with its predecessors is that it identifies the essential features automatically without any human supervision.

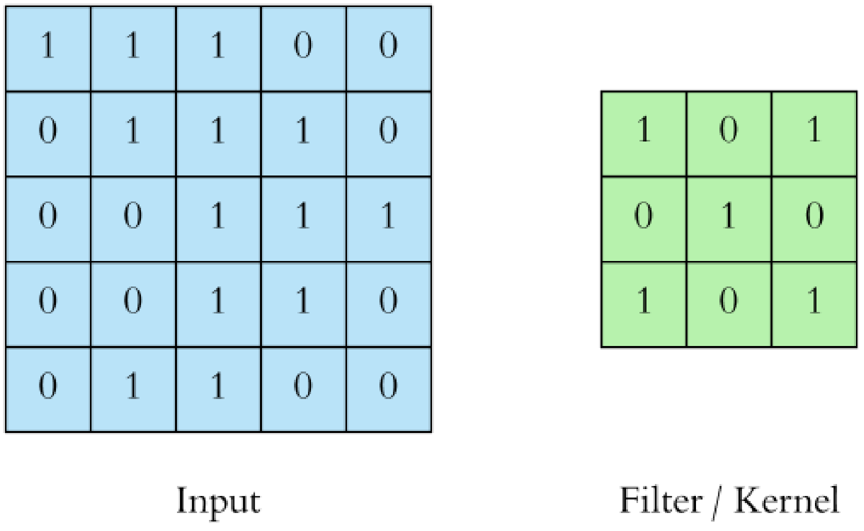
Exploring CNN Architecture



There is an input image that we work with is fed to the network.Then we perform a series of convolution + pooling operations, followed by several layers that are fully connected. If we perform classification of multiclass the output is softmax.

Convolutional Layer

CNN's core building block is the convolution layer. Convolution is a mathematical operation designed to merge two information sets. In our case, to produce a feature map, the convolution is applied to the input data using a convolution filter.



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input

1	0	1
0	1	0
1	0	1

Filter / Kernel

On the right is the convolution filter, also called the kernel.We perform the convolution operation by sliding this filter over the input. At every location, we do element-wise matrix multiplication and sum the result. This sum goes into the feature map.

So to capture minute details in the image small filter size can be used in the begining and gradually increasing the filter size in preceding layers would help to capture deatils for images which are not differentiated by small features.

Stride and Padding

Stride specifies how much we move the Convolution filter at each step.

If we want less overlap between the receptive fields, we can make bigger strides.But we will loose the inofrmation with increasing stride length.

Padding is commonly used in CNN to preserve the size of the feature maps, otherwise at each layer they would shrink, which is not desirable!

Pooling

Usually we do pooling after a convolution operation to reduce the dimensionality. This allows us to lower the number of parameters, both shortening the training time and combating overfitting.

The most common type of pooling is max pooling, which only takes the maximum value in the pooling window. It slides a window over its input and takes the maximum value in the window simply. Similar to a convolution, we specify the size and stride of the window.

So small pooling size can help in retaining more information but costs more training time.

Fully Connected

After the convolution + pooling layers we add a couple of fully connected layers to wrap up the CNN architecture.

the output of both convolution and pooling layers are 3D volumes, but a fully connected layer expects a 1D vector of numbers.

So we flatten the final pooling layer output into a vector and that becomes the input onto the fully connected layer. Flattening just arranges the 3D volume of numbers in a 1D vector.

Undersatanding CNN : A CNN model can be considered as a combination of two components: the feature extraction, and the classification part.

Feature extraction is performed by the convolution + pooling layers.

On top of these extracted features the fully connected layers act as a classifier and assign a probability for the input image

Ref : Convolutional Neural Networks

Ref : Understand the architecture of CNN

Creating 3-layered CNN model and training it

We will use keras sequential model to build CNN with 3 layers with basic parameters and then tune them to achieve god accuracy.

First Layer : 32 filters with Kernal = 3 and activation = 'relu'

Second Layer : 64 filters with Kernal = 3 and activation = 'relu'

Third Layer : 128 filters with Kernal = 3 and activation = 'relu'

Followed by Max Pooling of pool_size = 2 and then a drop out of 0.2 after each layer to drop neurons during the training phase of certain set of neurons which is chosen at random along with a final drop out of .5.

Before connecting to Dense network we flatten the out put of Max pooling to convert it to 1D array.

Then at last we will use a dense network for predicting the probability for each image and use softmax at the last stage.

Ref for building model : Convolutional Neural Networks for Beginners

```
cnn_model = Sequential()

# Try 32 fliters first then 64
cnn_model.add(Conv2D(32, (3, 3), input_shape = (28,28,1), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
cnn_model.add(Dropout(0.2))

cnn_model.add(Conv2D(64, (3, 3), input_shape = (28,28,1), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
cnn_model.add(Dropout(0.2))

cnn_model.add(Conv2D(128, (3, 3), input_shape = (28,28,1), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
cnn_model.add(Dropout(0.2))


# cnn_model.add(Conv2D(32,3, 3, activation='relu'))
# cnn_model.add(MaxPooling2D(pool_size = (2, 2)))

cnn_model.add(Flatten())
cnn_model.add(Dense(units = 4096, activation = 'relu'))
cnn_model.add(Dropout(0.5))
cnn_model.add(Dense(units = 5, activation = 'softmax'))
```

Design Choices made:

activation layer = relu -> considering ReLu as it is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations and also is not prone to vanishing gradients.

Ref : Understanding Activation Functions in Neural Networks

optimizer = Adam -> Adam is an adaptive learning rate method, which means, it computes individual learning rates for different parameters.It is regarded as one of the most robust optimizer.

Ref : Adam — latest trends in deep learning optimization

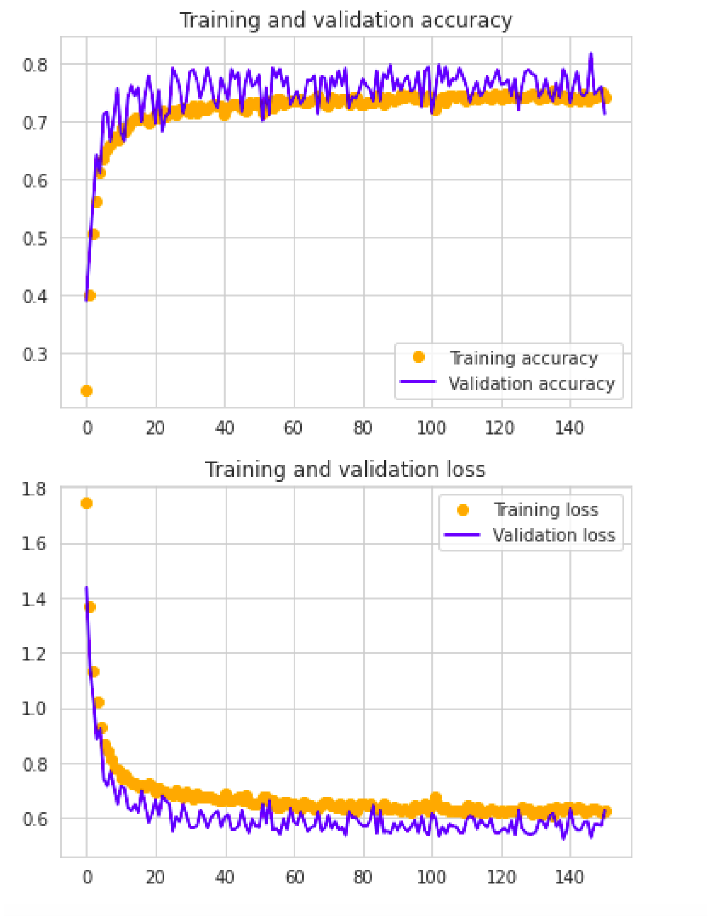
loss function = sparse_categorical_crossentropy -> As labels in my model are not hot encoded going with using sparse_categorical_crossentropy as loss function.

Ref : When to use sparse_categorical_crossentropy

```
cnn_model.compile(loss ='sparse_categorical_crossentropy', optimizer=Adam(lr=0.01),metrics =['accuracy'])

epochs = 551
history = cnn_model.fit(X_train, y_train, batch_size = 512, epochs = epochs,
                        verbose = 1, validation_data = (X_validate, y_validate))
```

samples = 10k

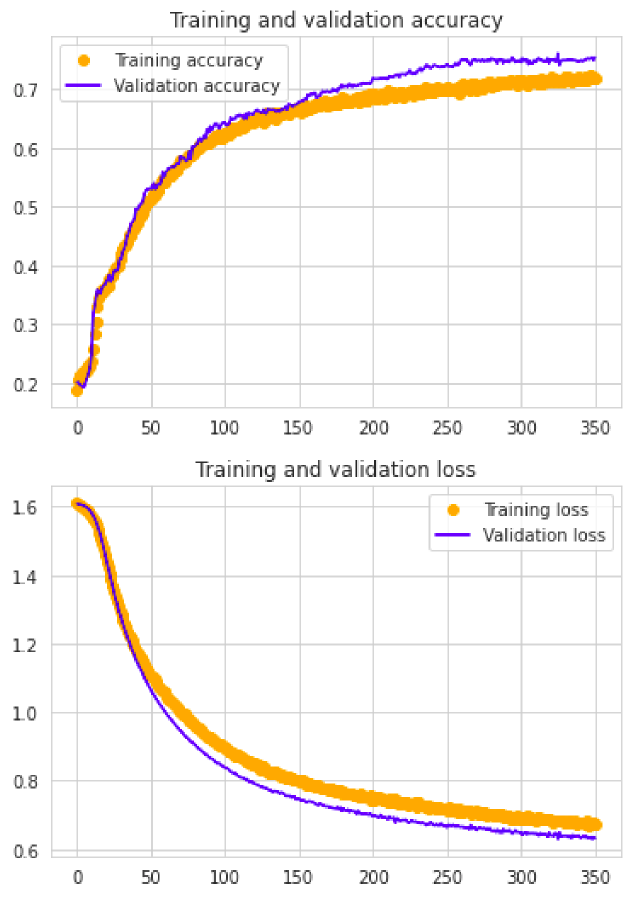
Lets observe the graphs for above model : epochs = 151



Test Accuracy : 0.707

It seems due to high learning rate of 0.01 the accuracy and loss are pretty fluctuated.This is not desirable as we will not be sure when the training stops we might end up with a less accurate model.The accuracies seems to be in upward trend and loss in downward trend.So we can still increase the number of epochs.

New modified parameters : lr = 0.00001 and epochs = 351



Test Accuracy : 0.728

We have fixed the fluctions and even now training accuracy is on a raise and loss is degrading.So we can still increase the number of epochs.But first lets play with filters and check if can get more training accuracy.

Lets try by increasing the size of filters layer by layer to check if our data set has important overall features than the minute differences.

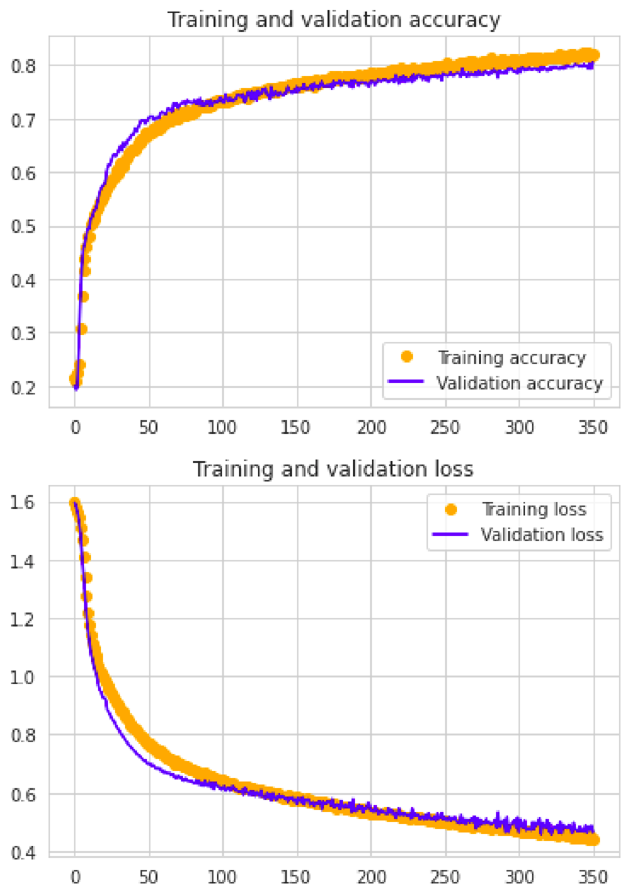
Changing filters stride length and introducing padding to the model.

So the modified model is a below:

```
cnn_model = Sequential()

# Try 32 fliters first then 64
cnn_model.add(Conv2D(32, (6, 6), input_shape = (28,28,1), activation='relu',strides=(1, 1),padding="same"))
cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
cnn_model.add(Dropout(0.2))

cnn_model.add(Conv2D(64, (7, 7), input_shape = (28,28,1), activation='relu',strides=(1, 1),padding="same"))
cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
cnn_model.add(Dropout(0.2))

cnn_model.add(Conv2D(128, (8, 8), input_shape = (28,28,1), activation='relu',strides=(1, 1),padding="same"))
cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
cnn_model.add(Dropout(0.35))
```

```
# cnn_model.add(Conv2D(32,3, 3, activation='relu'))
# cnn_model.add(MaxPooling2D(pool_size = (2, 2)))

cnn_model.add(Flatten())
cnn_model.add(Dense(units = 4096, activation = 'relu'))
cnn_model.add(Dropout(0.5))
cnn_model.add(Dense(units = 5, activation = 'softmax'))
```

lr = 0.00001 epochs = 351:

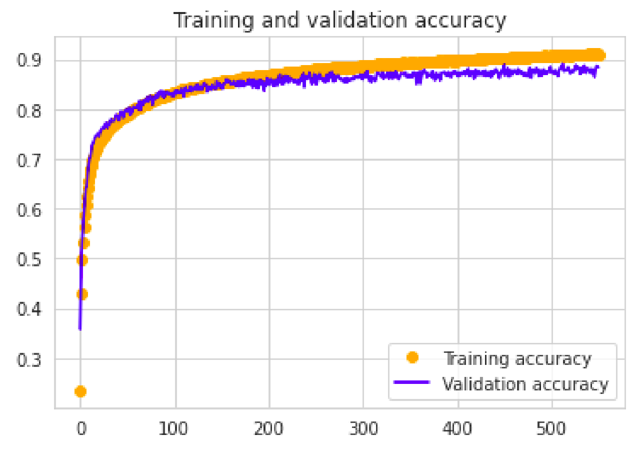


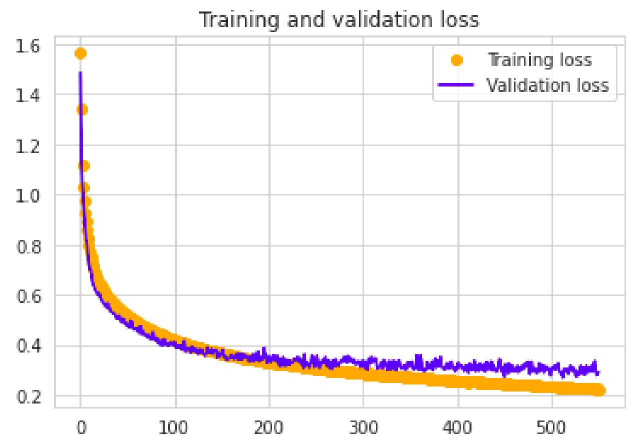
BAM it worked : Test Accuracy : 0.830

It seems we can still increase the number of epochs as accuracies are increasing and loss is decreasing.

Now lets train the same model for 60k samples with small validation and testing sets and observe the results.

Let us increase epochs to 550 as discussed above





Test Accuracy : 0.894

Lets train the whole model without any validation set and then predict the classes for test data and submit in kaggle

time taken for training CNN for all samples is 1360.5041358470917 - 551 epochs

time taken for predicting CNN for all samples is 0.52525

Kaggle highest Accuracy obtained for this model is : 0.91180

Understanding ResNet

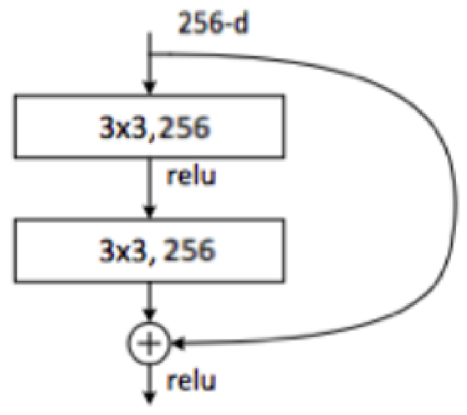
Vanishing gradient problem:

For neural networks with gradient-based learning methods and backpropagation each of the weights of the neural network receives an update proportional to the partial derivative of the error function as regards the current weight in each training iteration. The problem is that the gradient can in certain cases be vanishingly small, essentially preventing the weight from changing its value. In the worst case, this may completely stop the neural network from further training.

To solve this problem, the activation unit from a layer could be fed directly to a deeper layer of the network, which is termed as a **skip connection**.This forms the basis of **residual networks or ResNets**.

Residual block : A building block of a ResNet is called a residual block or identity block.A residual block is simply when the activation a layer is fast-forwarded to a deeper layer within the neural network.

This simple tweak allows training much deeper neural networks.



Example of a residual block

It can be written as two lines of code :

```
X_shortcut = X # Store the initial value of X in a variable
## Perform convolution + batch norm operations on X
X = Add()([X, X_shortcut]) # SKIP Connection
```

Building the model:

```
# Creating residual mapping using y
        y = Conv2D(num_filters,kernel_size=3,padding='same',strides=strides,
                   kernel_initializer='he_normal',kernel_regularizer=l2(1e-4))(x)
        y = BatchNormalization()(y)
        y = Activation('relu')(y)
        y = Conv2D(num_filters,kernel_size=3,padding='same',
                   kernel_initializer='he_normal',kernel_regularizer=l2(1e-4))(y)
        y = BatchNormalization()(y)
        if is_first_layer_but_not_first_block:
            x = Conv2D(num_filters,kernel_size=1,padding='same',strides=2,
                       kernel_initializer='he_normal',kernel_regularizer=l2(1e-4))(x)
        #Adding back residual mapping
        x = keras.layers.add([x, y])
        x = Activation('relu')(x)


# Add classifier on top.
x = AveragePooling2D()(x)
y = Flatten()(x)
outputs = Dense(num_classes,activation='softmax',kernel_initializer='he_normal')(y)

# Instantiate and compile model.
model = Model(inputs=inputs, outputs=outputs)
model.compile(loss='categorical_crossentropy',optimizer=Adam(lr=0.01),metrics=['accuracy'])
model.summary()
```

Design Choices made :

optimizer = Adam -> Adam is an adaptive learning rate method, which means, it computes individual learning rates for different parameters.It is regarded as one of the most robust optimizer.

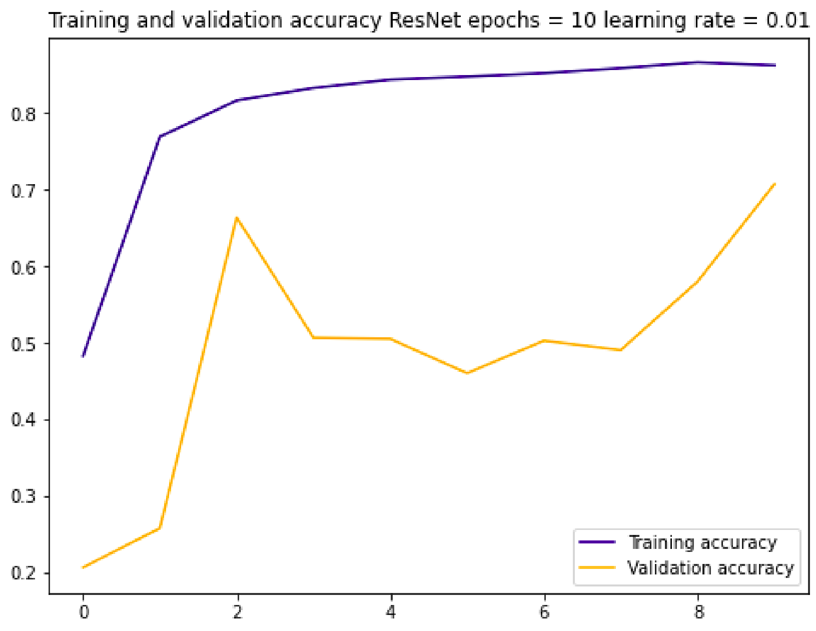
Ref : Adam — latest trends in deep learning optimization

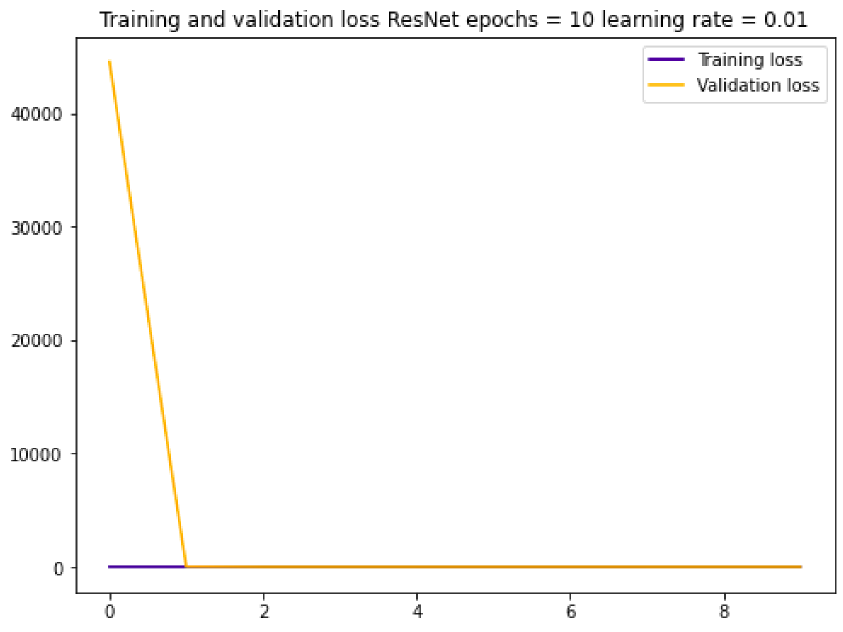
loss function = categorical_crossentropy -> categorical cross entropy is for multiclass classification.As labels in my model are one hot encoded going with using categorical_crossentropy as loss function.

Ref : When to use categorical_crossentropy

Lets train the model for epochs = 10 and learning rate = 0.01

time taken for training ResNet for epochs = 10 is 122.48458 sec





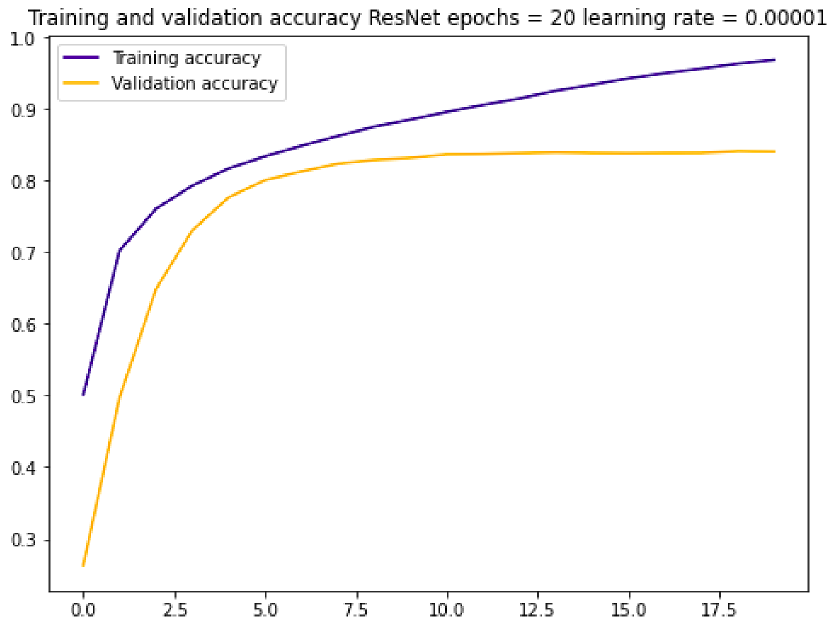
Test loss: 0.9423477164904277

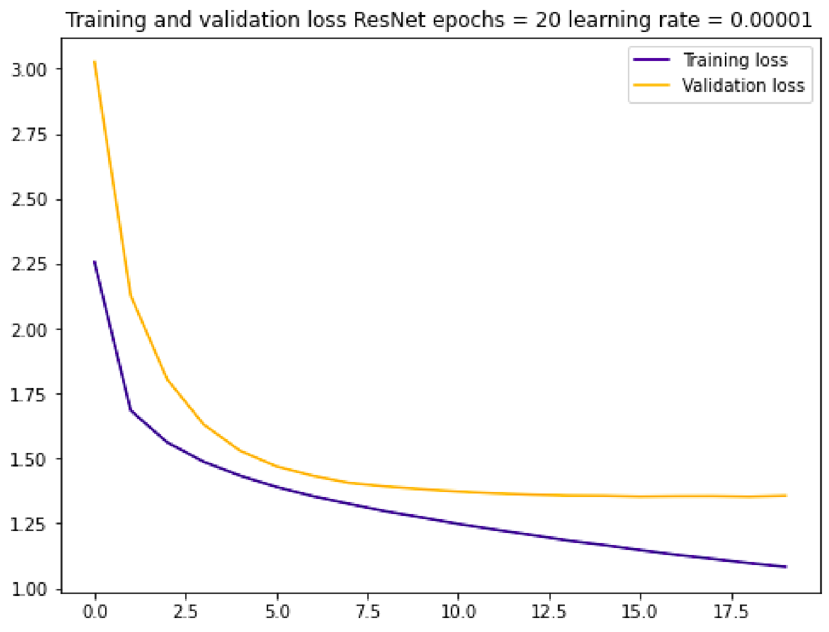
Test accuracy: 0.721666693687439

It seems tthat learning rate 0.01 is very high , so that it effecting accuracy and loss unpredictably.

Lets increase epochs = 20 and learning rate = 0.00001

time taken for training ResNet for epochs = 10 is 223.19536 sec



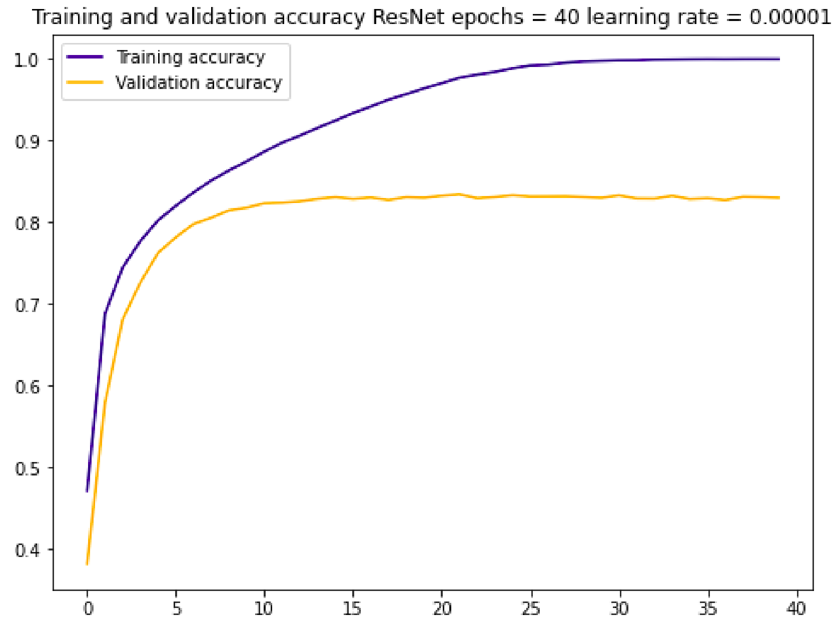


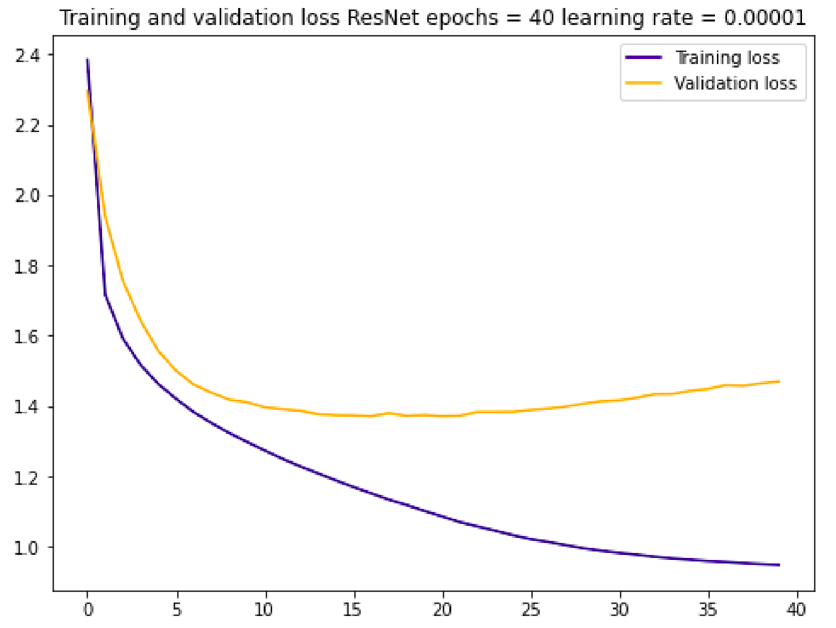
Test loss: 1.17467

Test accuracy: 0.8283

We can increase epochs to 40 and check if it gives better performance.

time taken for training ResNet for epochs = 40 is 426.80789 sec





Test loss: 1.460918

Test accuracy: 0.83499

We can observer the model is surely overfit.The training accuracy has reached one and validation loss started icreasing 25 epochs.

So lets train the model for whole data with epochs = 25 and learning rate = 0.00001

time taken for training ResNet for epochs = 25 for all data is 313.58110 sec

Kaggle accuracy : 0.83140

So lets train the model for whole data with epochs = 25 and learning rate = 0.00001

time taken for training ResNet for epochs = 25 for all data is 313.58110 sec

time taken for predicting ResNet is 2.2669 sec

Kaggle accuracy : 0.83140

References:

Ref : Vanishing gradient problem

Ref : Guide to Residual Networks (ResNet) in Keras

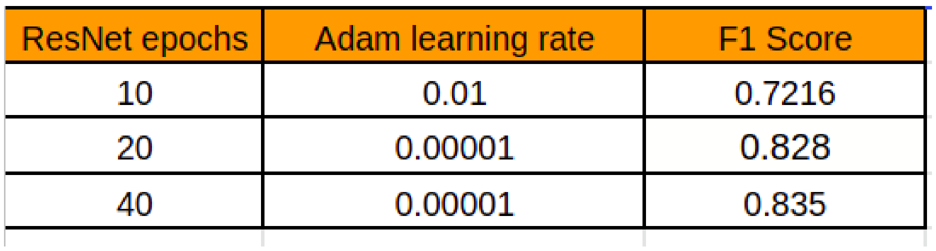
Ref : Understanding and Coding a ResNet in Keras

Metrics

Now lets discuss osme of the metrics observed during training and test process

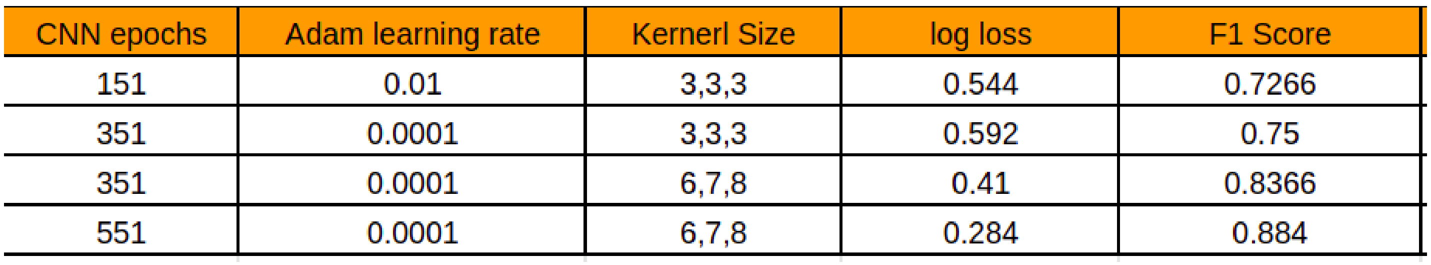
Metrics observed during:

ResNet training and testing process:



ResNet epochs	Adam learning rate	F1 Score
10	0.01	0.7216
20	0.00001	0.828
40	0.00001	0.835

CNN training and testing process:



CNN epochs	Adam learning rate	Kernerl Size	log loss	F1 Score
151	0.01	3,3,3	0.544	0.7266
351	0.0001	3,3,3	0.592	0.75
351	0.0001	6,7,8	0.41	0.8366
551	0.0001	6,7,8	0.284	0.884

Let's discuss CNN metrics in bit detail:

F1 score : is a measure of a test's accuracy. The F score is defined as the weighted harmonic mean of the test's precision and recall. This score is calculated according to:

$$F_1 = \left(\frac{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}}{2}\right)^{-1} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

With the precision and recall of a test taken into account. Precision, also called the positive predictive value, is the proportion of positive results that truly are positive. Recall, also called sensitivity, is the ability of a test to correctly identify positive results to get the true positive rate. The F score reaches the best value, meaning perfect precision and recall, at a value of 1. The worst F score, which means lowest precision and lowest recall, would be a value of 0.

Observations : We can notice from the above image that F1 score is increased through out parameter tuning.This implies that it has better precision and recall.

Ref : F1-Score

Log loss : Accuracy is the number of predictions where your predicted value equals the actual value. Due to its yes or no nature, accuracy is not always a good indicator. Log Loss takes into account the uncertainty of your prediction, based on how much it varies from the actual label. This gives us a more nuanced insight into our model's performance.

Observation : From the above image we can notice that log loss decreased as we tuned our parameters.The lesser the log loss the more certain our model is in predicting the output classes/labels.

Ref : Understanding Log loss

Ref : Log Loss wiki

Confusion Matrix :A confusion matrix is a summary of the results of a prediction over a classification issue. The number of predictions that are correct and incorrect is summarised with count values and broken down by class. That's the key to matrix confusion. The confusion matrix shows the ways in which your classification model is confused when it makes predictions.It gives us insight not only into the mistakes made by a classifier but, more importantly, the types of mistakes made.

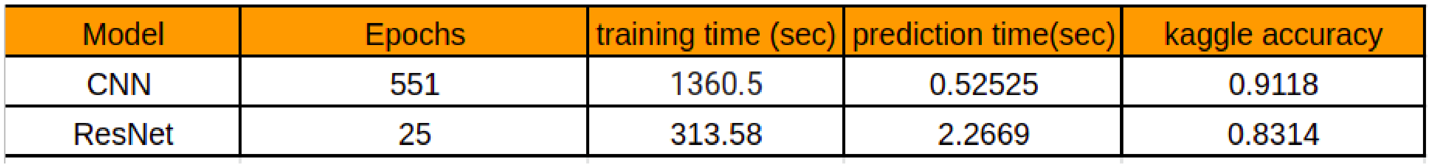
```
Confusion Matrix for CNN -> lr = 0.01 epochs = 151

[[112  16   1   0   0]
 [ 15  76  13   1   2]
 [  3  15  67  11   0]
 [  0  13  27  78   3]
 [  0   2  16  32  97]]


Confusion Matrix for CNN -> lr = 0.00001 epochs =351 kernels = 3,3,3

[[115   1   2   0   1]
 [ 36  72  18   0   1]
 [  6  25  59  15   1]
 [  1   3  20  76  11]
 [  0   1   4  22 110]]


Confusion Matrix for CNN -> lr = 0.00001 epochs = 351 kernels = 6,7,8

[[116   2   1   0   0]
 [ 20  97  10   0   0]
 [  3  14  79  10   0]
 [  0   2  11  89   9]
 [  0   0   2  14 121]]
```

Observations : We can notice that with tuning parameters and testing on test data, the model started classifying more tru posities for each class (observe the matrices). Example : class 5 : true positives increase from 97 -> 110 ->12

Ref : The Confusion Matrix: Unveiled

Ref : Confusion Matrix in Machine Learning

Comparing CNN and ResNet training and prediction times:



Model	Epochs	training time (sec)	prediction time(sec)	kaggle accuracy
CNN	551	1360.5	0.52525	0.9118
ResNet	25	313.58	2.2669	0.8314

- We can observe that prediction time of CNN is one-fourth of that of ResNet model.
- The time time by CNN for 551 epochs training is 1360 sec as whereas time taken by ResNet for just 25 epochs is 313.58 sec.
- For our dataset CNN with 3 layers is able to give higher accuracy than ResNet.
- We cannot decide wether CNN or ResNet is best based on this results.
- Apllication where a where deep Network like may be 1000 layers is need to be trained and used for prediction then we would definitly look at ResNet for that application.
- For this dataset CNN seems to be good enough to an accuracy of 91.18%

Highest Kaggle Score recieved using Deep Neural Networks is 91.18%

Team Name : The Beard Guy