Design and Implementation choices of Model

BASIC STRUCTURE OF OUR APPROACH(DESIGN)

1. Train→ 60000 with labels
2. Test→ 10000 without labels
3. df_x→ reshaped features→ (60000,28,28,1)
4. df_y→ categorical labels→ (60000,5)
5. Resizing the image as per the architecture.
6. Split into train and validate→ x_train, x_test, y_train, y_test(80:20)
7.
    1. x_train: y_train=48000: features and labels
    2. x_test: y_test=12000: features and labels
8. All features Normalized.
9. Preprocessing the unlabeled test data.
10. Apply Models:
    a. CNN
    b. RESNET
    c. LENET
    d. VGG-16
    e. ALEXNET
11. For each Model:
    a. Time complexity for each combination of parameters.
    b. Accuracy for each combination.
    c. Plots:
        a. Training Loss vs Epoch
        b. Validation Loss vs Epoch
        c. Training Accuracy vs Epoch
        d. Validation Accuracy vs Epoch
        e. Classification Accuracy vs Loss(Training)
        f. Classification Accuracy vs Loss(Validation).

APPROACHES FOLLOWED

1. Convolution Neural Network with custom layers and multiple tweaks.
2. RESNET published Architecture.
3. LENET Architecture.
4. VGG 16
5. ALEXNET

Note:

1. **Some tweaks are made in all the Architecture. For which we will discuss the results and analysis.**
2. **Random seed(1337) with shuffle=False is used in all the architectures.**

CONVOLUTION NEURAL NETWORKS

Neural networks are made up of the neurons which consists of the weights and the biases. All the neurons are capable of receiving inputs and performs the weighted sum on them, transfer the same to an activation function and provides the output label.

Convolution Layer: The convolution layer is the building layer of the neural network. It consists of the set of filters. Each filter is independently convolved with the image.

Filter: Array of numbers called weights or parameters.

For instance:

Input Image: 28*28*1

Filter: 5*5*1 (F-depth=I-depth)

Output: 24*24*1 (depth=no. of filters)

The output received after convolving the layer is called the feature map.

Pooling layer: It reduces the spatial size of the representation to reduce the number of parameters and computational complexity in the network. They operate on the feature map independently.

We will use max pooling with 2*2 filters mostly.

RELU: After each convolution layer, the convention is to apply the non-linear activation layer to introduce the non-linearity to the system which was computing linear operations during the convolution layers i.e. the element wise multiplications and summations.

Earlier non-linear functions of tanh and sigmoid were used but ReLU layers work much faster and also helps to solve the problem of vanishing gradient.

FULLY CONNECTED LAYERS: It is the fully connected layer which is at the end. All the neurons in this layer are fully connected to all the

activations in the previous layers. It basically looks at the output of the previous layer i.e. activation map of high-level features and determine which feature correlate to which class.

Other parameters used:

1. **Stride:** We usually increase the stride to 3 if we don't want overlapped input values during the convolution.
2. **Padding:** If we convolve a 32*32*3 input volume using 3 (5*5*3) filters we will end up having output volume of 28*28*3 in first layer and losing some low-level features. Thus, padding is required to preserve the features.
3. **Parameter choice:** It depends on the size of the dataset and the complexity. We need to find proper combination of parameters that create the abstractions ta a proper scale.

IMPORTANT CODE BLOCKS FOR CNN

1. IMPORTS AND DATA LOAD

In [70]:
```
import numpy as np
np.random.seed(1337)
import keras
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers.normalization import c
from keras.losses import categorical_crossentropy

import tensorflow as tf

from sklearn.model_selection import train_test_split
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt

for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/ece657a-w20-asg3-part2/train.csv
/kaggle/input/ece657a-w20-asg3-part2/testX.csv
```

2. DATA RESHAPING INTO 28*28*1 & CATEGORICAL LABELS FORMATION.

Reshaping the dataset into 28*28--

In [74]:
```
df_x = train.iloc[:,2:].values.reshape(len(train),28,28,1)
#Storing the labels in y
y = train.iloc[:,1].values
```

In [75]:
```
df_x.shape,y.shape
```

Out[75]:
```
((60000, 28, 28, 1), (60000,))
```

In [76]:
```
train['Label'].unique()
```

Out[76]:
```
array([4, 0, 1, 2, 3])
```

categorical labels--

In [77]:
```
#Converting labels to categorical features
df_y = keras.utils.to_categorical(y,num_classes=5)
```

3. SPLITTING THE DATASET

Splitting into train and val set--

In [81]:
```
x_train, x_test, y_train, y_test = train_test_split(df_x,df_y,test_size=0.2,random_state=42)
```

In [82]:
```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[82]:
```
((48000, 28, 28, 1), (12000, 28, 28, 1), (48000, 5), (12000, 5))
```

4. NORMALIZING THE DATASET

Normalizing the dataset--

In [83]:
```
x_train=x_train.astype('float32')
x_test=x_test.astype('float32')
#rescaling it between 0 to 1
x_train /=255
x_test /=255
```

5. PREPROCESSING THE TEST DATA

Preprocessing the real test data--

In [84]:
```
# for test data

#Storing Pixel array in form length width and channel in df_x_test
df_x_test = test.iloc[:,1:].values.reshape(len(test),28,28,1)
df_x_test=df_x_test.astype('float32')
#rescaling it between 0 to 1
df_x_test /=255
```

Here, we have finished our data preparation for the CNN models.

Note: We tried multiple combination of layers with multiple parameters and we have set the CNN Model-1 as the baseline for our analysis. So, we will be improving this model by tweaking it with parameters to get the better results on the validation set.

1. CNN MODEL-1: This CNN model consists of:

Layer	Count
Convolution	4
Maxpooling	2
Dropout	3
Dense	3
Optimizer	Adam()
Epoch	50
Batch Size	32

MODEL 1------ Adam() | BS: 32 | Epoch: 50

In [86]:

```
model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (28,28,1)))
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))


model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.25))
model.add(Dense(5, activation = "softmax"))


model.compile(optimizer = keras.optimizers.Adam() , loss = "categorical_crossentropy", metrics=
["accuracy"])
```

SUMMARY

In [87]:

```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 28, 28, 32)        25632
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_14 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_15 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_9 (Dense)              (None, 256)               803072
_________________________________________________________________
dropout_16 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_10 (Dense)             (None, 5)                 1285
=================================================================
Total params: 886,245
Trainable params: 886,245
Non-trainable params: 0
_________________________________________________________________
```

FITTING THE MODEL

```
start=timeit.default_timer()

a=model.fit(x_train,y_train,batch_size = 32,epochs = 50,verbose=1,validation_data=(x_test,y_test),shuffle=False)

stop=timeit.default_timer()
```

ACCURACY AND TIME

RESULT --

Epoch 50 | BS: 32 | val acc: 88.80% | val loss: 0.3498 |Time Taken 626.66

```
print('Time Taken',stop-start)
```

```
Time Taken 626.6638785159957
```

```
Epoch 50/50
48000/48000 [==============================] - 12s 256us/step - loss: 0.1652 - accuracy:
0.9345 - val_loss: 0.3498 - val_accuracy: 0.8880
```

MODEL METRICS

Making dataframe

In [93]:

```
train_acc_1=a.history['accuracy']
train_loss_1=a.history['loss']
val_acc_1=a.history['val_accuracy']
val_loss_1=a.history['val_loss']

num=len(a.history['accuracy'])
eplist=np.arange(1,num+1,step=1)
eplist=eplist.tolist()
```

DATAFRAME

	Epoch	train_acc	train_loss	val_acc	val_loss
0	1	0.750646	0.596504	0.837750	0.403372
1	2	0.823812	0.431185	0.853500	0.359961
2	3	0.844312	0.381224	0.865750	0.329141
3	4	0.855812	0.354485	0.869250	0.319300
4	5	0.864625	0.334641	0.870167	0.312544
5	6	0.869563	0.317614	0.873333	0.301360

PLOTS

FOR REST OF THE MODELS, WE WILL ANALYZE THE GRAPHS AT THE END OF EACH APPROACH.

Epoch vs Train loss

In [98]:
```
import seaborn as sns;
fig= plt.figure(figsize=(10,8))
sns.set(style="darkgrid")
ax=sns.lineplot(x='Epoch',y='train_loss',data=history_df1)
plt.title("Epoch vs Train loss")

plt.xlabel("Epoch")
plt.ylabel("Training Loss")
plt.legend(["Training Loss"])
plt.show()
```

Epoch vs Val loss

In [99]:
```
import seaborn as sns;
fig= plt.figure(figsize=(10,8))
sns.set(style="darkgrid")
ax=sns.lineplot(x='Epoch',y='val_loss',data=history_df1)
plt.title("Epoch vs val loss")

plt.xlabel("Epoch")
plt.ylabel("val Loss")
plt.legend(["Val Loss"])
plt.show()
```

Epoch vs Train accuracy

In [100]:
```
import seaborn as sns;
fig= plt.figure(figsize=(10,8))
sns.set(style="darkgrid")
ax=sns.lineplot(x='Epoch',y='train_acc',data=history_df1)
plt.title("Epoch vs train accuracy")

plt.xlabel("Epoch")
plt.ylabel("train accuracy")
plt.legend(["Train accuracy"])
```

Epoch vs Val accuracy

In [101]:
```
import seaborn as sns;
fig= plt.figure(figsize=(10,8))
sns.set(style="darkgrid")
ax=sns.lineplot(x='Epoch',y='val_acc',data=history_df1)
plt.title("Epoch vs Val accuracy")

plt.xlabel("Epoch")
plt.ylabel("Val accuracy")
plt.legend(["Val accuracy"])
plt.show()
```

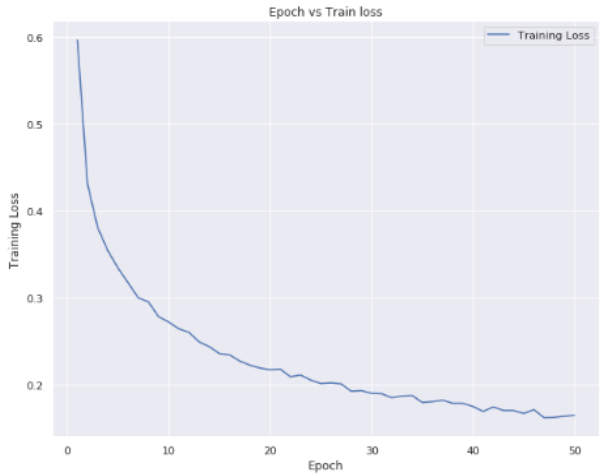
Train accuracy vs Train loss

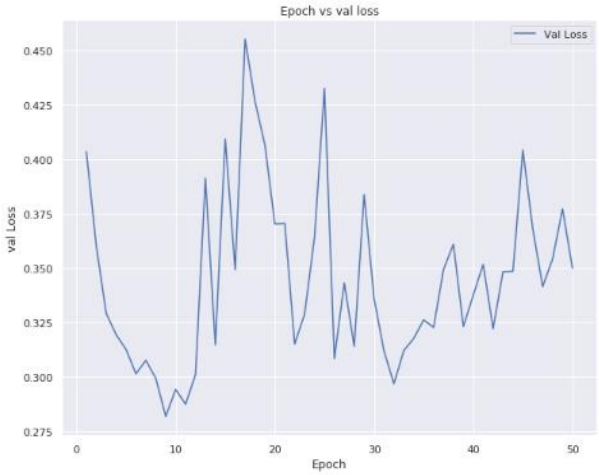
In [102]:
```
import seaborn as sns;
fig= plt.figure(figsize=(10,8))
sns.set(style="darkgrid")
ax=sns.lineplot(x='train_acc',y='train_loss',data=history_df1)
plt.title("Train accuracy vs Train loss")

plt.xlabel("Train Accuracy")
plt.ylabel("Train Loss")
plt.legend(["Train accuracy vs Train loss"])
plt.show()
```

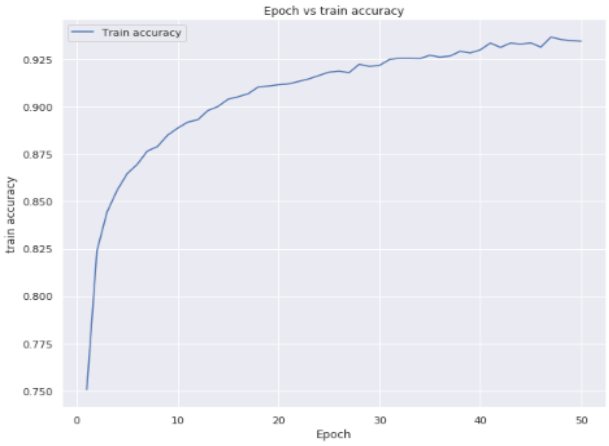
Val accuracy vs Val loss

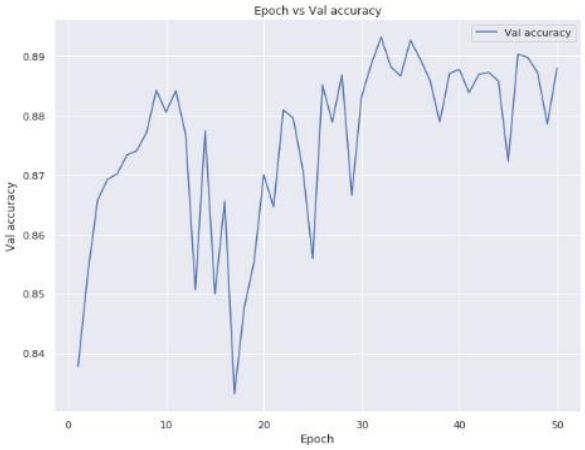
In [103]:
```
import seaborn as sns;
fig= plt.figure(figsize=(10,8))
sns.set(style="darkgrid")
ax=sns.lineplot(x='val_acc',y='val_loss',data=history_df1)
plt.title("Val accuracy vs Val loss")

plt.xlabel("Val Accuracy")
plt.ylabel("Val Loss")
plt.legend(["Val accuracy vs Val loss"])
plt.show()
```

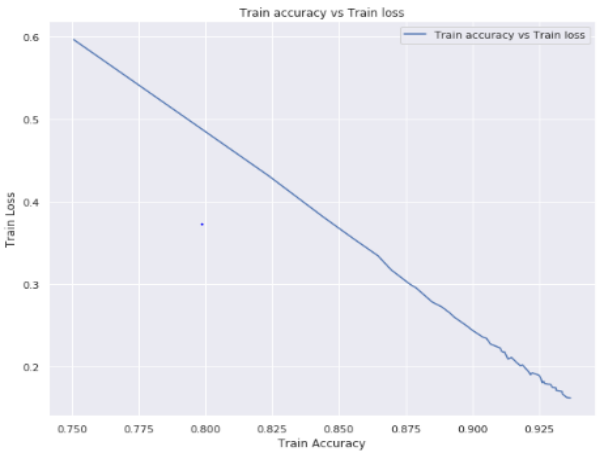
FOR REST OF THE MODELS, WE WILL ANALYZE THE GRAPHS AT THE END OF EACH APPROACH.

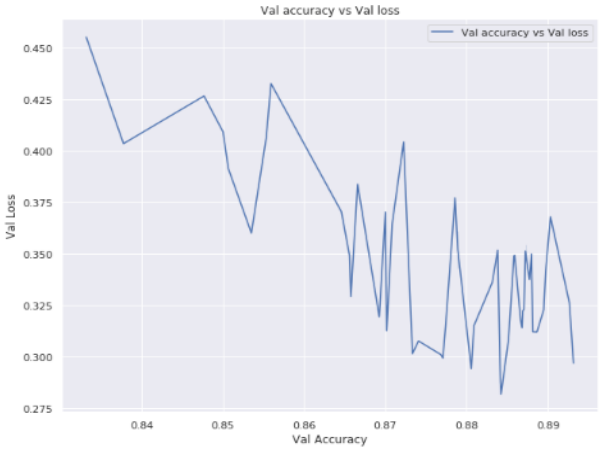












UNLABELED TEST DATA PREDICTION

```
In [105]: df_x_test.shape
Out[105]: (10000, 28, 28, 1)

In [106]: target_classes = model.predict_classes(df_x_test,verbose=1)
          10000/10000 [==============================] - 1s 67us/step

In [107]: target_classes=pd.DataFrame(target_classes)
```

2. CNN MODEL-2: This CNN model consists of:

Layer	Count
Convolution	4
Maxpooling	2
Dropout	3
Dense	3

Optimizer	Adam()
Epoch	50
Batch Size	32
Callback	ReduceLROnPlateau

+

ReduceLROnPlateau (ADDITION IN THIS MODEL)

```
learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

MODEL 2------ Adam() | BS: 32 | Epoch: 50 | callback | learning_rate_reduction

In [111]:

```
model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (28,28,1)))
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))


model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.25))
model.add(Dense(5, activation = "softmax"))

# Set a learning rate annealer

learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)



model.compile(optimizer = keras.optimizers.Adam() , loss = "categorical_crossentropy", metrics=
["accuracy"])
```

SUMMARY

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 28, 28, 32)        25632
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dropout_17 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_18 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_11 (Dense)             (None, 256)               803072
_________________________________________________________________
dropout_19 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_12 (Dense)             (None, 5)                 1285
=================================================================
Total params: 886,245
Trainable params: 886,245
Non-trainable params: 0
_________________________________________________________________
```

FITTING THE MODEL

In [113]:

```
import timeit
start=timeit.default_timer()

a=model.fit(x_train,y_train,batch_size = 32,epochs = 50,verbose=1,validation_data=(x_test,y_tes
t),callbacks=[learning_rate_reduction],shuffle=False)

stop=timeit.default_timer()
```

ACCURACY AND TIME

RESULT --

Epoch 50 | BS: 32 | val acc: 0.9068 | val loss: 0.2798 |Time Taken 620.27

In [115]:
```
print('Time Taken',stop-start)
```

```
Time Taken 620.2746239570042
```

```
Epoch 50/50
48000/48000 [==============================] - 13s 267us/step - loss: 0.1081 - accuracy:
0.9569 - val_loss: 0.2798 - val_accuracy: 0.9068
```

MODEL METRICS

	Epoch	train_acc	train_loss	val_acc	val_loss
0	1	0.753187	0.587544	0.832750	0.404939
1	2	0.828375	0.422969	0.852500	0.362263
2	3	0.843583	0.381160	0.872167	0.321167
3	4	0.857583	0.350440	0.872667	0.309248
4	5	0.865625	0.329842	0.881083	0.289230

PLOTS

```
import seaborn as sns;
fig= plt.figure(figsize=(10,8))
sns.set(style="darkgrid")
ax=sns.lineplot(x='Epoch',y='train_loss',data=history_df1)
plt.title("Epoch vs Train loss")

plt.xlabel("Epoch")
plt.ylabel("Training Loss")
plt.legend(["Training Loss"])
plt.show()
```

NOTE: PLOTS WILL BE SHARED IN ANALYSIS SECTION DURING ANALYSIS.

3. CNN MODEL-3: This CNN model consists of:

--SAME ARCHIITECTURE AS MODEL-2 WITH CHANGED DROPOUT TO 0.50—

Layer	Count
Convolution	4
Maxpooling	2
Dropout	3
Dense	3

Optimizer	Adam()
Epoch	50
Batch Size	32
Callback	ReduceLROnPlateau

+

ReduceLROnPlateau

```
learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

+

DROPOUT=0.50

MODEL 3------ Adam() | BS: 32 | Epoch: 50 | callback | learning_rate_reduction | Dropout 0.50 from 0.25 in last layer

In [138]:

```
model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (28,28,1)))
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))


model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.50))
model.add(Dense(5, activation = "softmax"))

# Set a learning rate annealer
```

```
learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)



model.compile(optimizer = keras.optimizers.Adam() , loss = "categorical_crossentropy", metrics=
["accuracy"])
```

SUMMARY

In [139]:
```
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_32 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 28, 28, 32)        25632
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dropout_26 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_27 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
flatten_9 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_17 (Dense)             (None, 256)               803072
_________________________________________________________________
dropout_28 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_18 (Dense)             (None, 5)                 1285
=================================================================
Total params: 886,245
Trainable params: 886,245
Non-trainable params: 0
```

ACCURACY AND TIME

RESULT --

Epoch 50 | BS: 32 | val acc:0.9020 | val loss: 0.2937 |Time Taken 620

In [142]:
```
print('Time Taken',stop-start)
```

```
Time Taken 620.1422442439944
```

NOTE: PLOTS IN ANALYSIS SECTION

4. CNN MODEL-4: This CNN model consists of:

Layer	Count
Convolution	4
Maxpooling	2
Dropout	3
Dense	3

Optimizer	Adam()
Epoch	50
Batch Size	32
Callback	ReduceLROnPlateau

+

ReduceLROnPlateau

```
learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

+

LeakyReLU

Model 4-- Adam() | BS: 32 | Epoch: 50 | LeakyReLU | callback | learning_rate_reduction | Dropout 0.25

In [166]:

```
from keras.layers.advanced_activations import LeakyReLU

model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (28,28,1)))
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))


model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(LeakyReLU(alpha=0.1))
model.add(Dropout(0.25))
model.add(Dense(5, activation = "softmax"))
```

```
# Set a learning rate annealer

learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)

model.compile(optimizer = keras.optimizers.Adam() , loss = "categorical_crossentropy", metrics=
["accuracy"])
```

SUMMARY

In [167]:
```
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_40 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
conv2d_41 (Conv2D)           (None, 28, 28, 32)        25632
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dropout_32 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_42 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
conv2d_43 (Conv2D)           (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_33 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
flatten_11 (Flatten)         (None, 3136)              0
_________________________________________________________________
dense_21 (Dense)             (None, 256)               803072
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dropout_34 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_22 (Dense)             (None, 5)                 1285
=================================================================
Total params: 886,245
Trainable params: 886,245
Non-trainable params: 0
_________________________________________________________________
```

ACCURACY AND TIME

RESULT -- ¶

Epoch 50 | BS: 32 | val acc: 0.9048 | val loss: 0.3230 |Time Taken 618.72

In [170]:
```
print('Time Taken',stop-start)
```

```
Time Taken 618.7266924010037
```

5. CNN MODEL-5: This CNN model consists of:

Layer	Count
Convolution	4
Maxpooling	2
Dropout	3
Dense	3

Optimizer	SGD(lr=0.01,decay=1e-6,momemum=0.9,nesterov=True)
Epoch	50
Batch Size	32
Callback	ReduceLROnPlateau

+

ReduceLROnPlateau

```
learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

Model 5-- SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True) | BS: 32 | Epoch: 50 | callback | learning_rate_reduction | Dropout 0.25

In [191]:

```
model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (28,28,1)))
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))


model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.25))
model.add(Dense(5, activation = "softmax"))
```

```
# Set a learning rate annealer

learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)



model.compile(optimizer = keras.optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=Tru
e) , loss = "categorical_crossentropy", metrics=["accuracy"])
```

SUMMARY

In [192]:
```
model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_44 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
conv2d_45 (Conv2D)           (None, 28, 28, 32)        25632
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dropout_35 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_46 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
conv2d_47 (Conv2D)           (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_36 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
flatten_12 (Flatten)         (None, 3136)              0
_________________________________________________________________
dense_23 (Dense)             (None, 256)               803072
_________________________________________________________________
dropout_37 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_24 (Dense)             (None, 5)                 1285
=================================================================
Total params: 886,245
Trainable params: 886,245
Non-trainable params: 0
_________________________________________________________________
```

TIME AND ACCURACY

RESULT --

Epoch 50 | BS: 32 | val acc: 0.9078 | val loss: 0.2696 |Time Taken 534.0414721150009

In [195]:
```
print('Time Taken',stop-start)
```

```
Time Taken 534.0414721150009
```

6. CNN MODEL-6 This CNN model consists of:

Layer	Count
Convolution	4
Maxpooling	2
Dropout	3
Dense	3

Optimizer	SGD(lr=0.01,decay=1e-6,momemum=0.9,nesterov=True)
Epoch	50
Batch Size	128
Callback	ReduceLROnPlateau

Model 6-- SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True) | BS: 128 | Epoch: 50 | LeakyReLU | callback | learning_rate_reduction | Dropout 0.25

In [216]:

```
model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (28,28,1)))
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))


model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.25))
model.add(Dense(5, activation = "softmax"))
```

```
# Set a learning rate annealer

learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)



model.compile(optimizer = keras.optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=Tru
e) , loss = "categorical_crossentropy", metrics=["accuracy"])
```

SUMMARY

In [217]:

```
model.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_48 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
conv2d_49 (Conv2D)           (None, 28, 28, 32)        25632
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dropout_38 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_39 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
flatten_13 (Flatten)         (None, 3136)              0
_________________________________________________________________
dense_25 (Dense)             (None, 256)               803072
_________________________________________________________________
dropout_40 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_26 (Dense)             (None, 5)                 1285
=================================================================
Total params: 886,245
Trainable params: 886,245
Non-trainable params: 0
```

TIME AND ACCURACY

RESULT --

Epoch 50 | BS: 32 | val acc: 0.9073 | val loss: 0.2311 |Time Taken 201

```
In [220]: print('Time Taken',stop-start)

Time Taken 201.29215921900322
```

7. CNN MODEL-7 This CNN model consists of:

Layer	Count
Convolution	4
Maxpooling	2
Dropout	3
Dense	3

Optimizer	SGD(lr=0.01,decay=1e-6,momemum=0.9,nesterov=True)
Epoch	50
Batch Size	32
Callback	ReduceLROnPlateau
Normalization	BatchNormalization()

Model 7-- SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True) | BS: 32 | Epoch: 50 | callback | learning_rate_reduction | Dropout 0.25 | Batch Normalizer()

In [248]:

```
model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (28,28,1)))
model.add(BatchNormalization())
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Dropout(0.25))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(BatchNormalization())


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))
```

```
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.25))
model.add(Dense(5, activation = "softmax"))

# Set a learning rate annealer

learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)



model.compile(optimizer = keras.optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=Tru
e) , loss = "categorical_crossentropy", metrics=["accuracy"])
```

SUMMARY

In [249]:

```
model.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_56 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
batch_normalization_5 (Batch (None, 28, 28, 32)        128
_________________________________________________________________
conv2d_57 (Conv2D)           (None, 28, 28, 32)        25632
_________________________________________________________________
batch_normalization_6 (Batch (None, 28, 28, 32)        128
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dropout_44 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_58 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
batch_normalization_7 (Batch (None, 14, 14, 64)        256
_________________________________________________________________
conv2d_59 (Conv2D)           (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_30 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_45 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
batch_normalization_8 (Batch (None, 7, 7, 64)          256
_________________________________________________________________
flatten_15 (Flatten)         (None, 3136)              0
_________________________________________________________________
flatten_15 (Flatten)         (None, 3136)              0
_________________________________________________________________
dense_29 (Dense)             (None, 256)               803072
_________________________________________________________________
dropout_46 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_30 (Dense)             (None, 5)                 1285
=================================================================
Total params: 887,013
Trainable params: 886,629
Non-trainable params: 384
_________________________________________________________________
```

TIME AND ACCURACY

RESULT --

Epoch 50 | BS: 32 | val acc: 0.9112 | val loss: 0.2488 |Time Taken 927.5281197939985

In [252]:

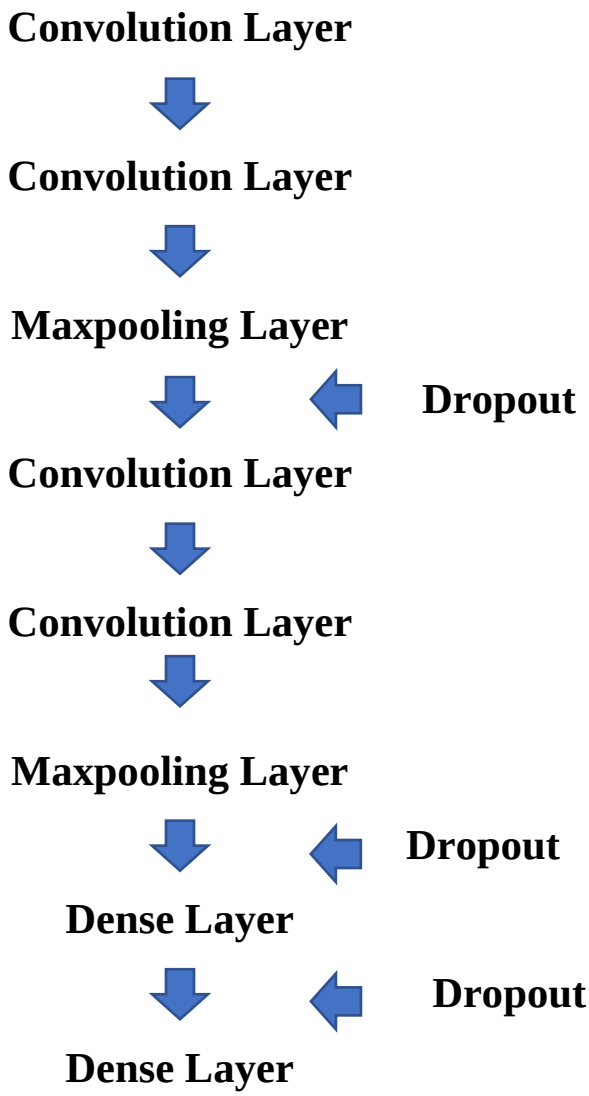
```
print('Time Taken',stop-start)
```

```
Time Taken 927.5281197939985
```

RESULTS & DETAILED ANALYSIS

DESIGN CHOICES | PARAMETER COMPARISON | ACCURACY| TIME COMPLEXITY| NETWORK ARCHITECTURE| OPTIMIZERS | PLOTS

Our Network Architecture:



Note: Subsequent layers were added which are shown below with all the parameters.

Models: Above we have stated that are taking model 1 as the base model after trying multiple models and architectures. We tried to improve the results of our base model by performing minor tweaks in the hyperparameters.

First, let's get the high-level idea of our models again.

1. **Model 1:** Validation accuracy→ 88.80%

Filters	32
Kernel size	(5,5)
Padding	Same
Activation	Relu
Input shape	28*28*1
Pool_size	(2,2)
Dropout	0.25
Activation	Softmax
BatchSize	32
Epoch	50
Optimizer	Adam()

2. **Model 2:** We saw the learning stagnating in the previous model, to overcome it we are using the callback which will adjust the learning rate i.e. reduce it by the factor of 3 if no improvement in val_accuracy is seen per 3 epochs . Validation accuracy → 90.68%

ReduceLrOnPlateau	(Factor=3,min_lr=0.00001, monitor='val_accuracy', patience=3)
Filters	32
Kernel size	(5,5)
Padding	Same
Activation	Relu
Input shape	28*28*1
Pool_size	(2,2)
Dropout	0.25

Activation	Softmax
BatchSize	32
Epoch	50
Optimizer	Adam()

3. **Model 3:** By trying the callback above, we got a decent improvement in the percentage. Now we are increasing the dropout to prevent the overfitting because the training accuracy was 95.69%. Validation Accuracy → 90.20%

ReduceLrOnPlateau	(Factor=3,min_lr=0.00001, monitor='val_accuracy', patience=3)
Filters	32
Kernel size	(5,5)
Padding	Same
Activation	Relu
Input shape	28*28*1
Pool_size	(2,2)
Dropout	0.50
Activation	Softmax
BatchSize	32
Epoch	50
Optimizer	Adam()

Optimizer used: Adam() computes individual adaptive learning rates for different parameters. It may fail to converge under specific settings.Thus , posing a need to shift to SGD().

4. **Model 4:** Model 3 didn't improve the validation accuracy. Now we are retaining the model 2 and tweaking its parameters again, neglecting the model 3.

When activation is ReLU, there exists a dying relu problem which suggests the elimination of some relu neurons for all the inputs affecting the performance of the

neural network which can be improved by using LeakyReLU [In this there is a change in the slope causing the leak resulting in the extended range for relu.]

Validation accuracy= 90.48%

ReduceLrOnPlateau	(Factor=3,min_lr=0.00001, monitor='val_accuracy', patience=3)
Filters	32
Kernel size	(5,5)
Padding	Same
Activation	Relu
Input shape	28*28*1
Pool_size	(2,2)
Dropout	0.25
Activation	Softmax
BatchSize	32
Epoch	50
Optimizer	Adam()
LeakyReLU	Alpha=0.1

5. **Model 5:** Model 4 didn't improve the validation accuracy. Now we are again retaining the model 2 and tweaking its parameters again, neglecting the model 4. Now we are using the SGD instead of Adam(). Validation Accuracy=90.78% .

SGD(lr=0.01,decay=1e-6,momentum=0.9,nesterov=True)

- Learning rate is 0.01 which will still be reduced by the factor of 3 because of the callback.
- We are applying the nesterov momentum.
- Momentum can accelerate the SGD in the relevant direction.

ReduceLrOnPlateau	(Factor=3,min_lr=0.00001, monitor='val_accuracy', patience=3)
Filters	32
Kernel size	(5,5)
Padding	Same
Activation	Relu
Input shape	28*28*1
Pool_size	(2,2)
Dropout	0.25
Activation	Softmax
BatchSize	32
Epoch	50
Optimizer	SGD()

Result: SGD(params) resulted in a slight increase in the accuracy to 90.78%

Now we will proceed with this model as our new baseline neglecting Adam() and keeping the dropout 0.25 for all and eliminating LeakyReLU but retaining the callback for learning rate reducer.

6. **Model 6:** Model 5 provided an improvement in the accuracy. Validation accuracy=90.73%

Following are the changes so far.

1. Neglect Adam() use SGD().
2. Proceed with Dropout=0.25.
3. Eliminate LeakyReLU.
4. Retain Callback for LR reduction with same parameters.

ReduceLrOnPlateau	(Factor=3,min_lr=0.00001, monitor='val_accuracy', patience=3)
Filters	32
Kernel size	(5,5)
Padding	Same
Activation	Relu
Input shape	28*28*1
Pool_size	(2,2)
Dropout	0.25
Activation	Softmax
BatchSize	128
Epoch	50
Optimizer	SGD()

7. Model 7: Increasing the batch size did not help much in the accuracy but reduced the time complexity by a factor of 3.

Now we are using the batch normalization on model 5. Batch normalization reduces the amount by what the hidden unit values shift around i.e. covariance shift.

ReduceLrOnPlateau	(Factor=3,min_lr=0.00001, monitor='val_accuracy', patience=3)
Filters	32
Kernel size	(5,5)
Padding	Same
Activation	Relu
Input shape	28*28*1
Pool_size	(2,2)
Dropout	0.25
Activation	Softmax
BatchSize	32
Epoch	50
Optimizer	SGD()

Here in the above architecture, we have achieved the validation accuracy of 91.12% which is decent improvement form our base model 1 accuracy which was 88.80% .

Overall Analysis of parameters and layers:

1. We used the kernel of size (5,5) and padding='same' to retain the low-level features which resulted in good accuracy of 88.80% in the base model itself.

2. Activation in the con2D layer is used as 'relu' to introduce the non-linearity to the system which was computing linear operations during the convolution layers i.e. the element wise multiplications and summation.

3. Learning rate reducer(ReduceLROnPlateau) provided a great improvement in the accuracy by 1.88%.

4. Increasing the dropout did not improve the accuracy at all as there was no overfitting happening in the model.

5. LekyReLU didn't result in a better accuracy given the fact that ReLU neurons weren't being inactive(Not confirmed).

6. SGD() provides better accuracy than Adam() in our case with the reduction in the time complexity as well. It reduced the training accuracy
   i.e. From 95.69%(Model 2)→95%(Model 5).

7. Batch Normalization resulted in an average increase of 0.50% in the accuracy on the validation set. As it allows each layer of a network learn independent of the other layers.

8. We have used the max_pooling with the pool size of (2,2) which reduces the spatial size of the representation to reduce the number of parameters and computational complexity in the network.

9. Softmax outputs the probability distribution function which we used in the final dense layer for our multi class problem.

OVERALL PERFROMANCE OF VARIOUS MODELS

	T-Acc	T-loss	Val-Acc	Val-Loss	Time
Model 1	**93.45**	**0.16**	**88.80**	**0.35**	**626**
Model 2	**95.69**	**0.10**	**90.68**	**0.27**	**620**
Model 3	**94.74**	**0.13**	**90.20**	**0.29**	**620**
Model 4	**96.84**	**0.08**	**90.48**	**0.32**	**618**
Model 5	**95.30**	**0.11**	**90.78**	**0.27**	**534**
Model 6	**92.0**	**0.19**	**90.73**	**0.23**	**201**
Model 7	**95.14**	**0.13**	**91.12**	**0.25**	**927**

- SGD
- Adam
- There is no sign of overfitting in our model 7 which can be seen and it can also be analyzed from the graphs.
- In Model 7, loss remains decreasing for most of the epochs with the increase in the accuracy with no bigger fluctuations for the both.
- Batch Normalization improved the overall accuracy from 90.73% to 91.12 on the validation set.

Runtime performance analysis

MODEL 1	**626.66**
MODEL 2	**620.27**
MODEL 3	**620.14**
MODEL 4	**618.72**
MODEL 5	**534.04**
MODEL 6	**201**
MODEL 7	**927.52**

Analysis:

1. Time taken by the network is almost similar when the optimizer was Adam() which can be seen above in the first 4 Models.

2. From 1 to 4 we can conclude that adding the LeakyRelu and tweaking some other parameters didn't increase the time complexity.

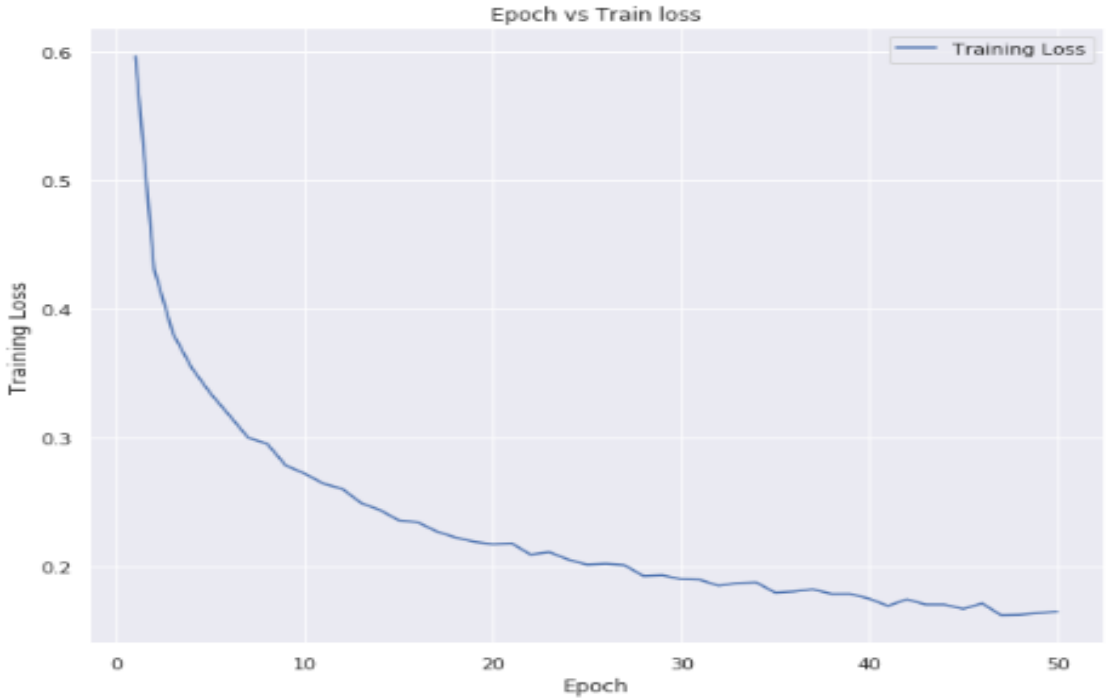
3. In model 5, when SGD() was used, there was a drastic reduction in the time by a factor of almost 3 which can be seen in the table.

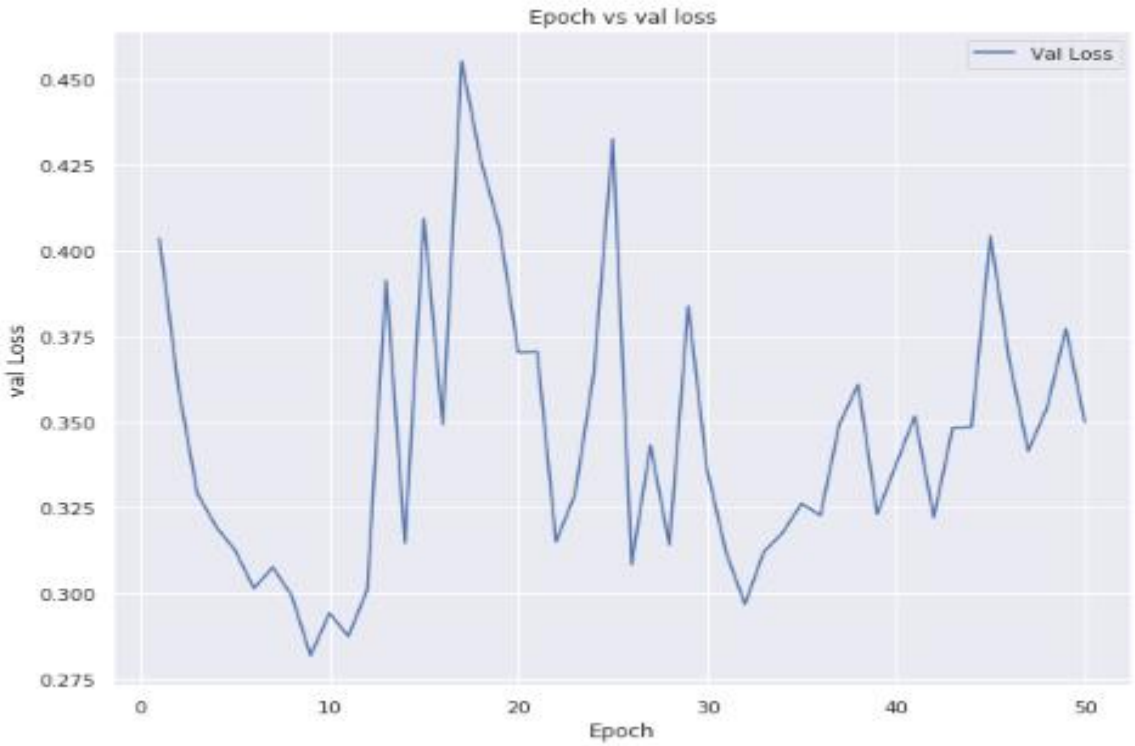
4. As we know that SGD() only computes on a small subset of dataset thus helping the network to compute faster.

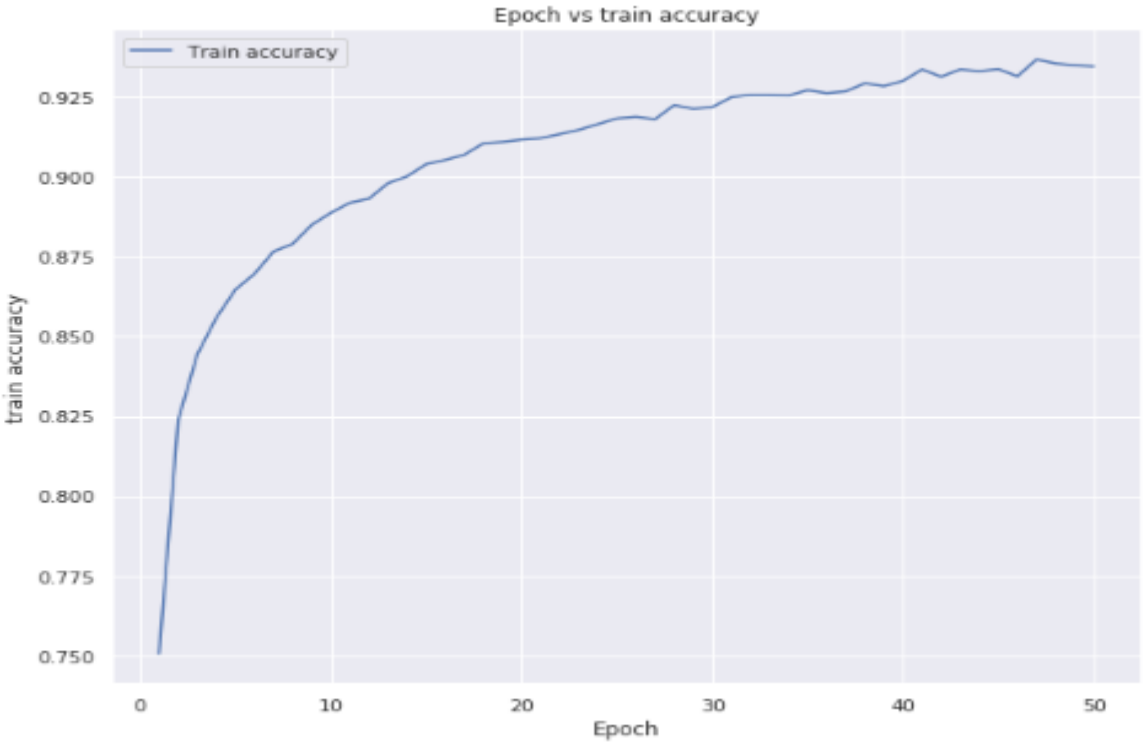
5. Runtime was highest when we used batch normalization which can be seen in the model 7 with a runtime of 927 seconds.

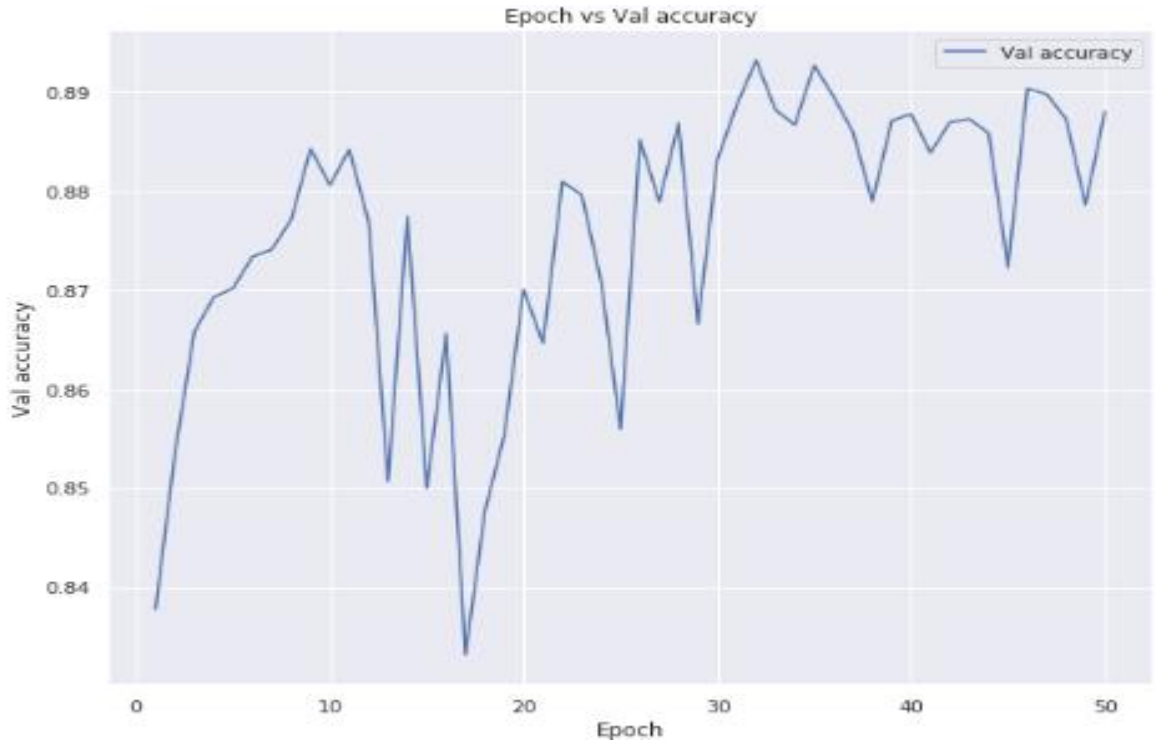
PLOTS

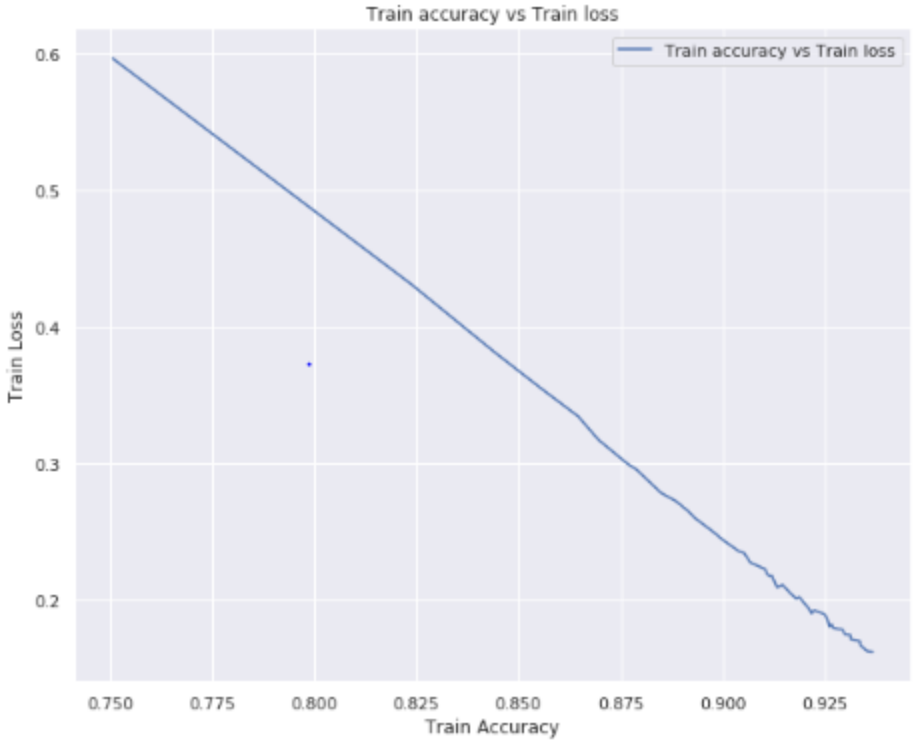
1. MODEL-1 => Baseline Model

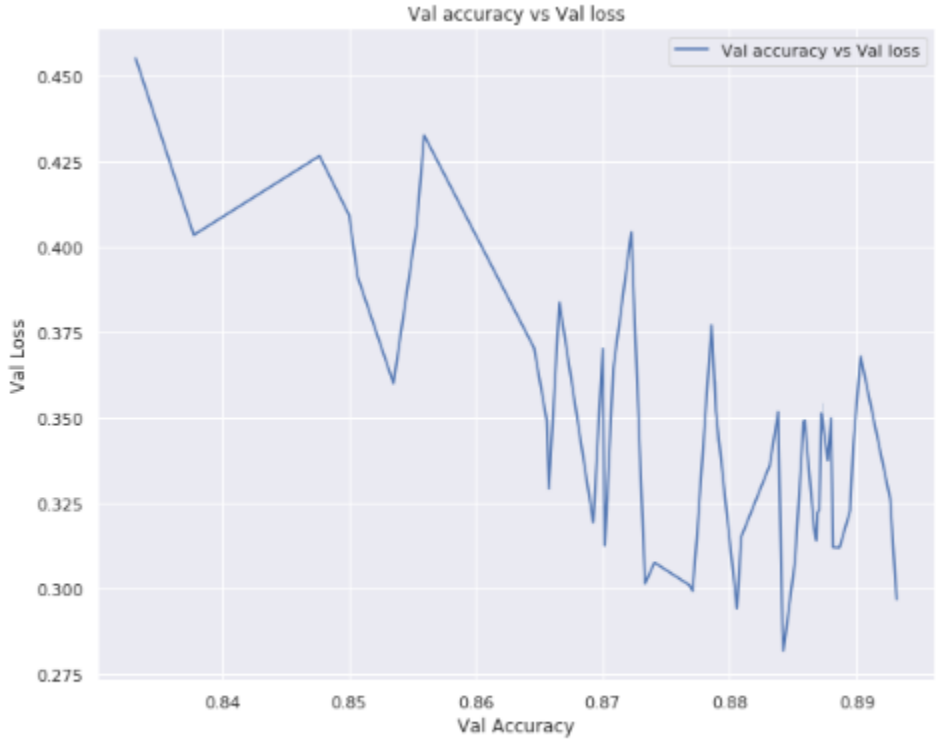




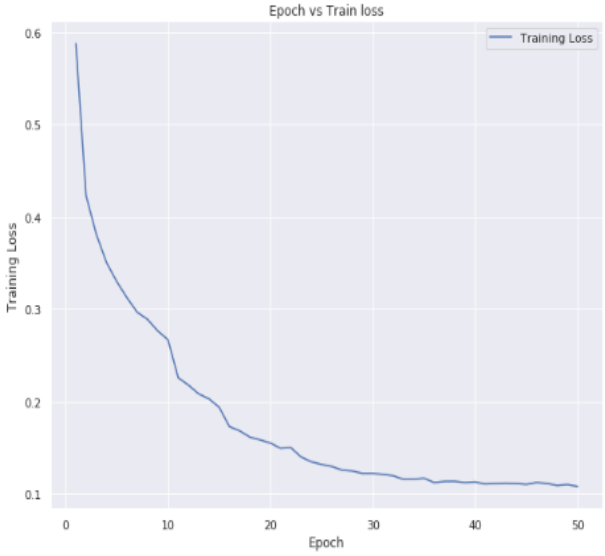

Epoch vs train accuracy
Train accuracy
train accuracy
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0
10
20
30
40
50
Epoch

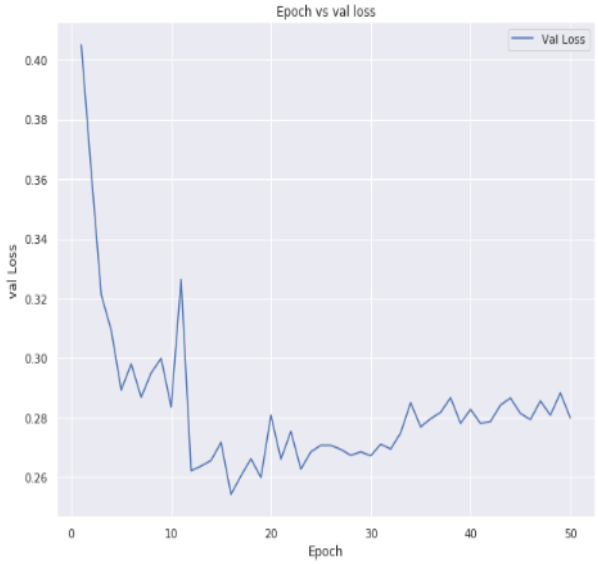

Epoch vs Val accuracy
Val accuracy
Val accuracy
0.89
0.88
0.87
0.86
0.85
0.84
0
10
20
30
40
50
Epoch

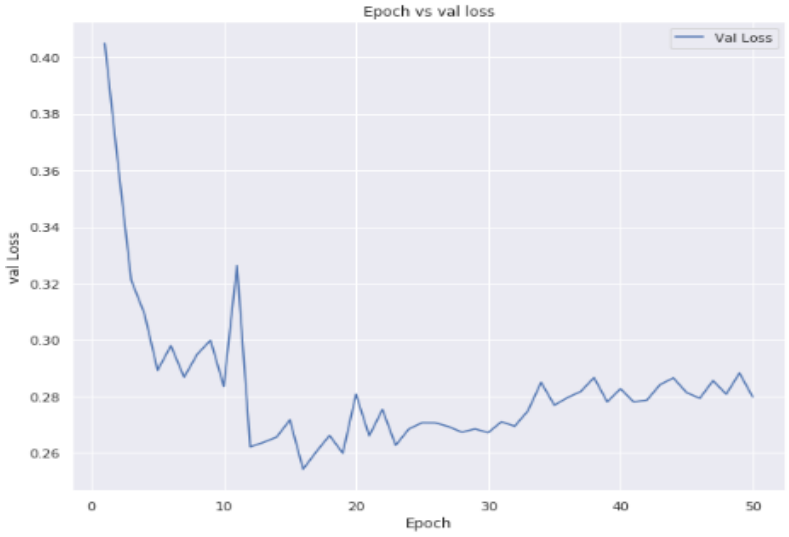

Train accuracy vs Train loss
Train accuracy vs Train loss
0.6
0.5
0.4
0.3
0.2
Train Loss
0.750
0.775
0.800
0.825
0.850
0.875
0.900
0.925
Train Accuracy

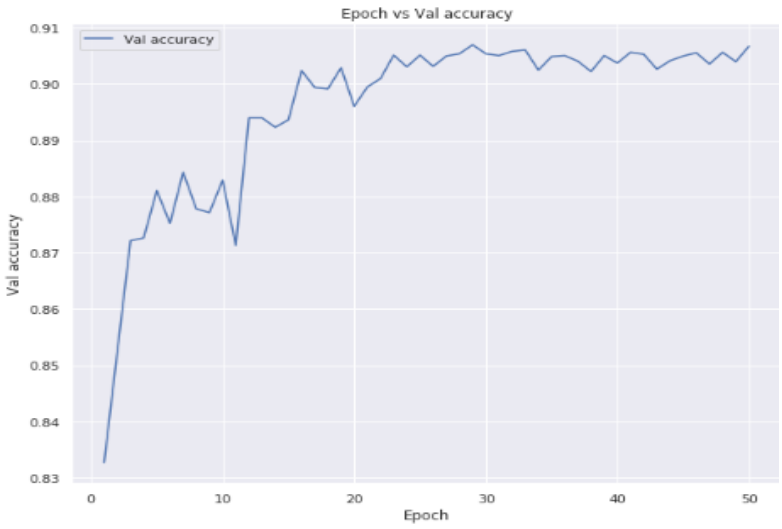

Val accuracy vs Val loss
Val accuracy vs Val loss
0.450
0.425
0.400
0.375
0.350
0.325
0.300
0.275
Val Loss
0.84
0.85
0.86
0.87
0.88
0.89
Val Accuracy

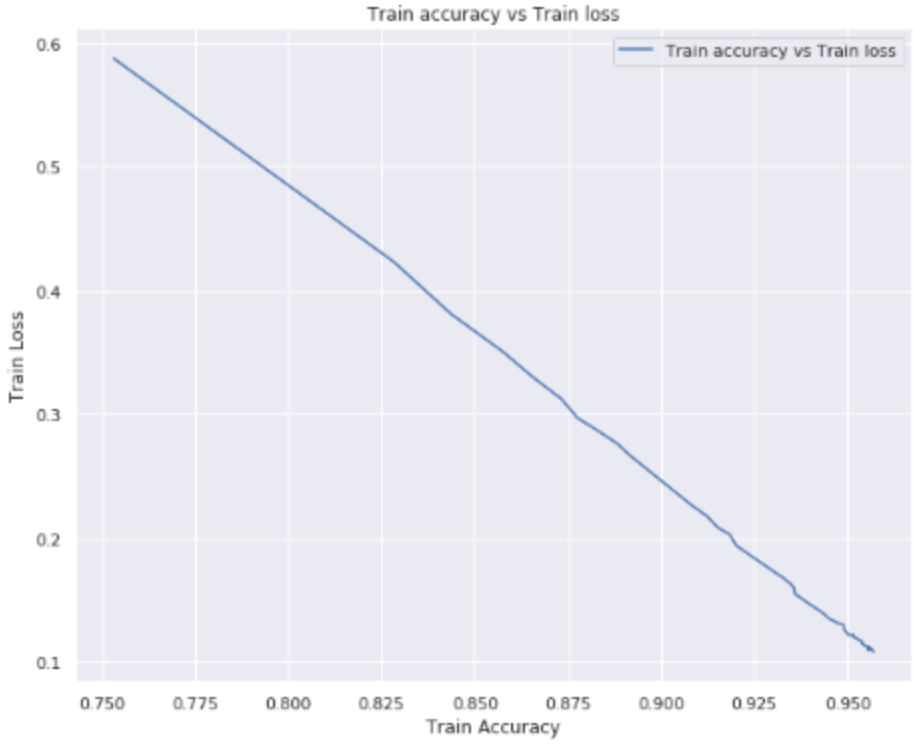
MODEL 2

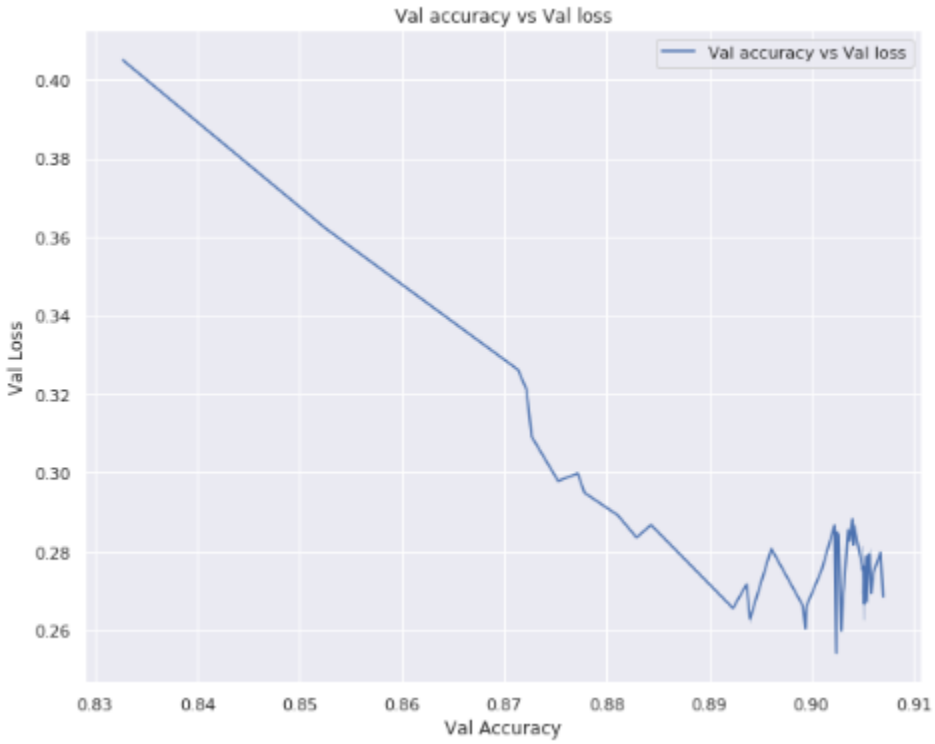

Epoch vs Train loss
Training Loss
Training Loss
Epoch
0.6
0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50


Epoch vs val loss
Val Loss
val Loss
Epoch
0.40
0.38
0.36
0.34
0.32
0.30
0.28
0.26
0
10
20
30
40
50

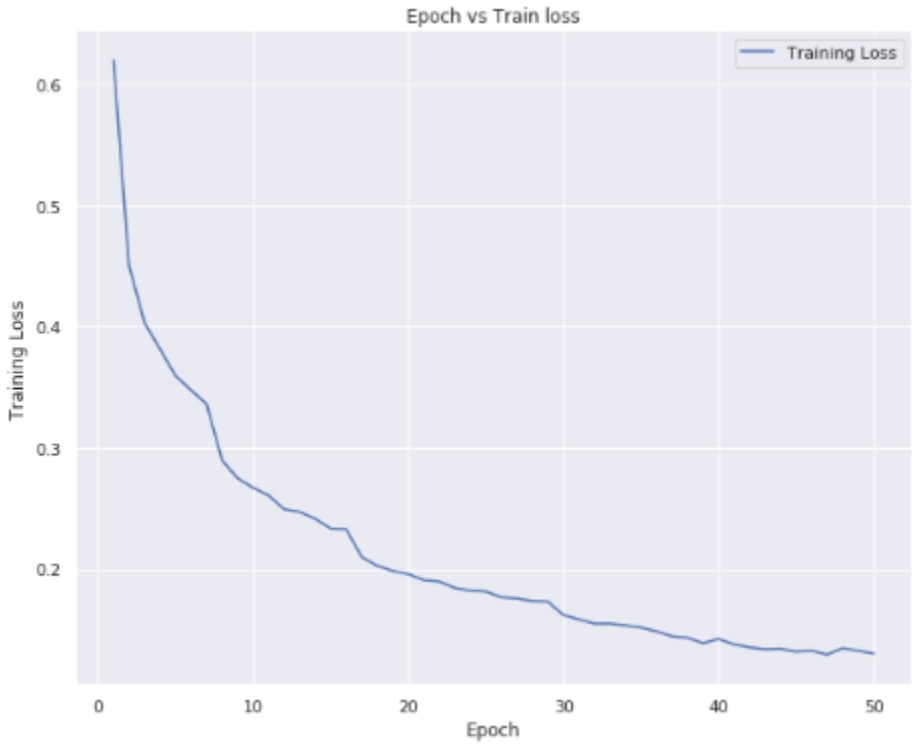

Epoch vs val loss
Val Loss
val Loss
Epoch
0.40
0.38
0.36
0.34
0.32
0.30
0.28
0.26
0
10
20
30
40
50

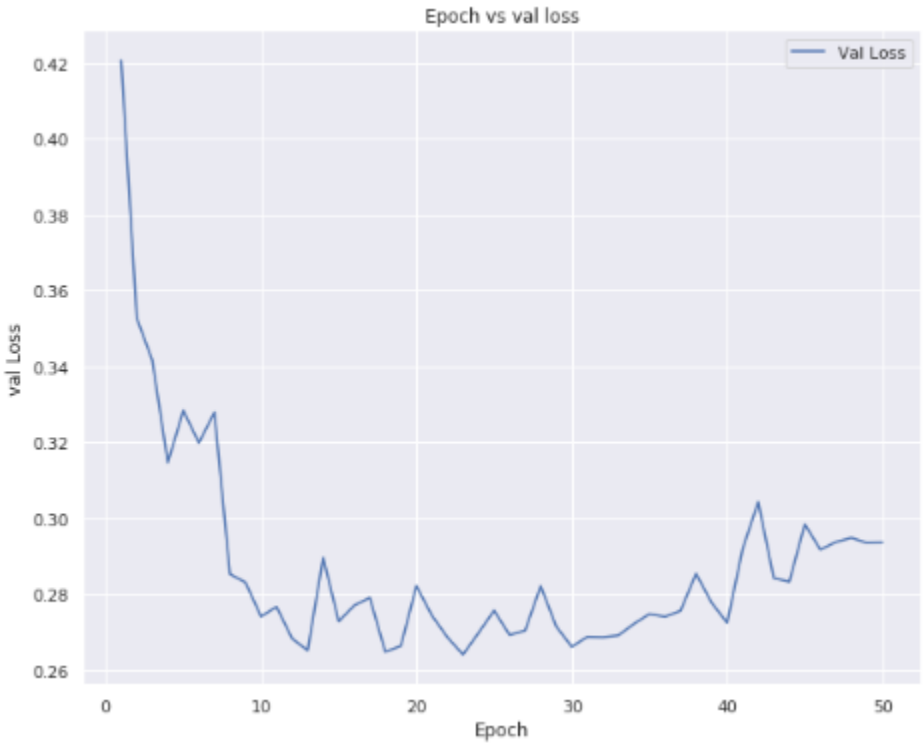

Epoch vs Val accuracy
Val accuracy
Val accuracy
Epoch
0.91
0.90
0.89
0.88
0.87
0.86
0.85
0.84
0.83
0
10
20
30
40
50

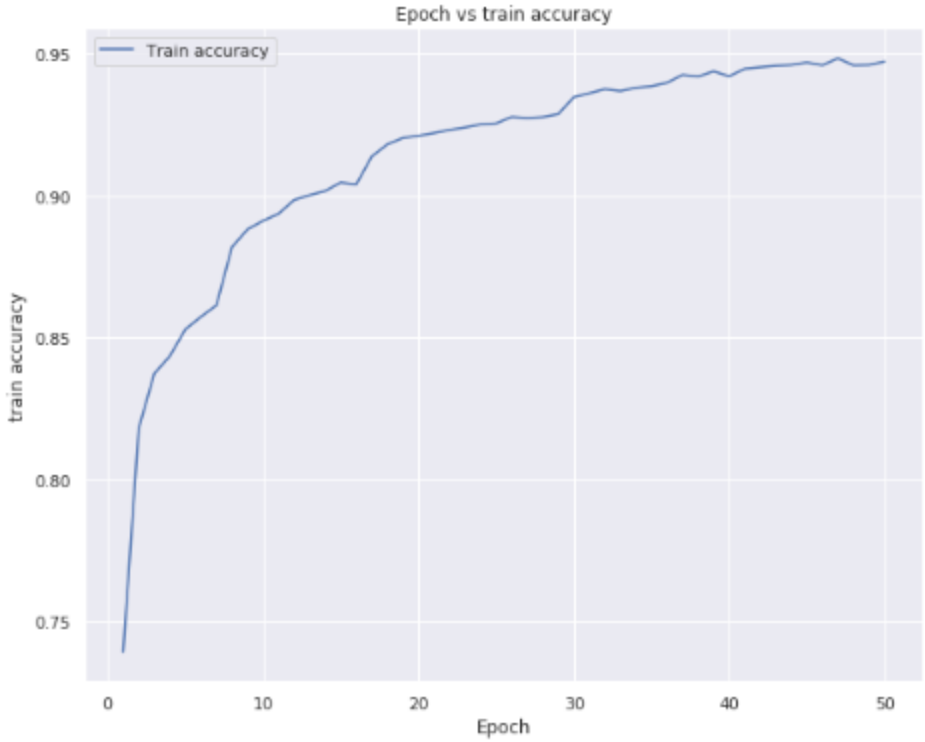

Train accuracy vs Train loss
Train accuracy vs Train loss
0.6
0.5
0.4
0.3
0.2
0.1
Train Loss
0.750
0.775
0.800
0.825
0.850
0.875
0.900
0.925
0.950
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
0.40
0.38
0.36
0.34
0.32
0.30
0.28
0.26
Val Loss
0.83
0.84
0.85
0.86
0.87
0.88
0.89
0.90
0.91
Val Accuracy

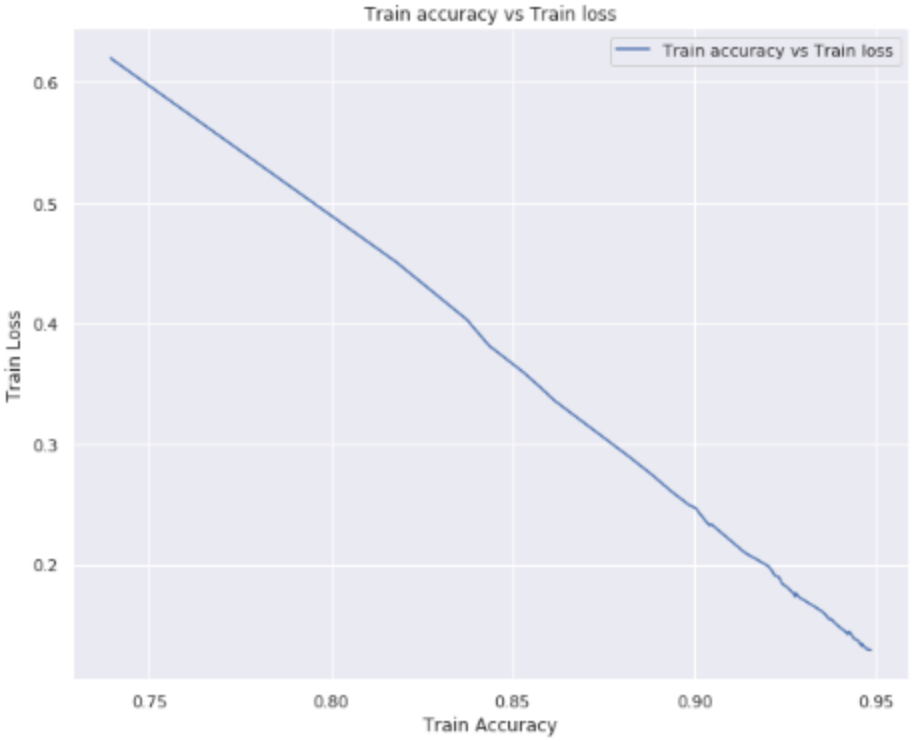
MODEL 3

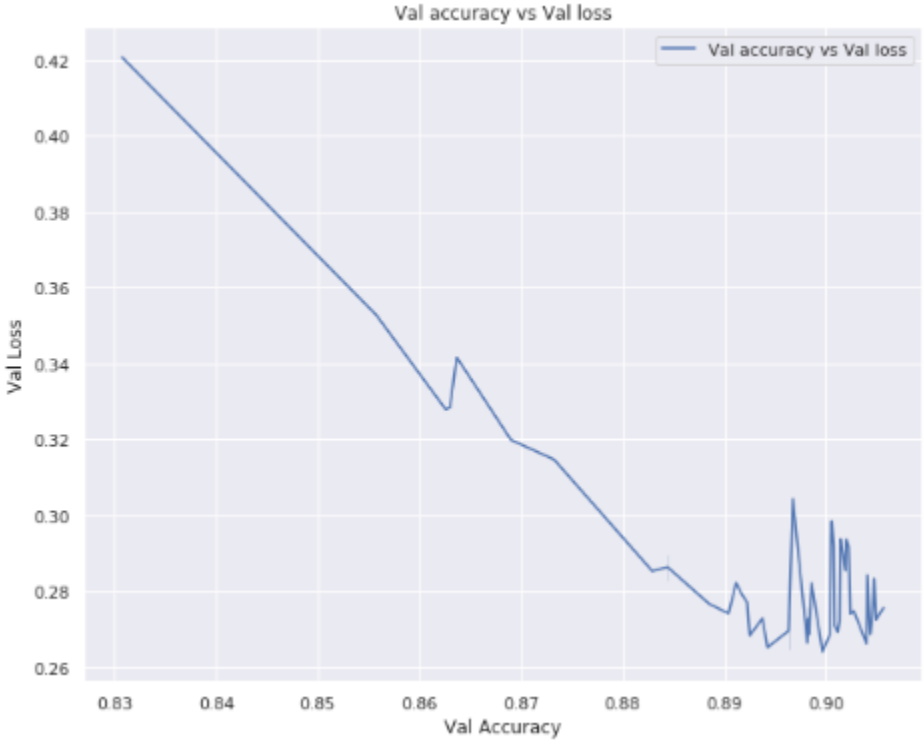

Epoch vs Train loss
Training Loss
0.6
0.5
0.4
0.3
0.2
Training Loss
0
10
20
30
40
50
Epoch


Epoch vs val loss
Val Loss
0.42
0.40
0.38
0.36
0.34
0.32
0.30
0.28
0.26
val Loss
0
10
20
30
40
50
Epoch

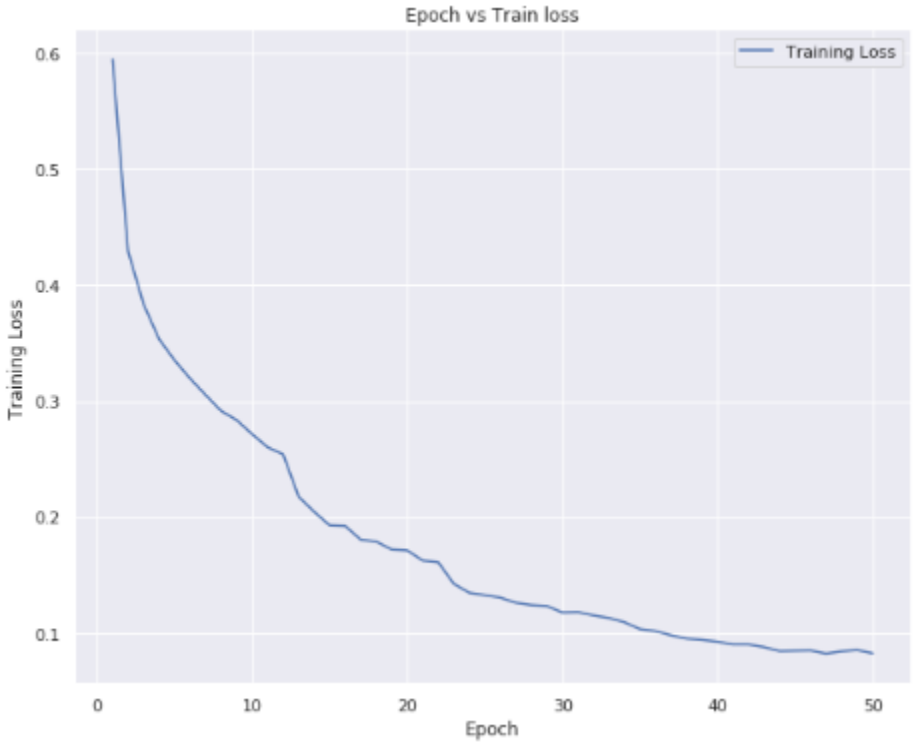

Epoch vs train accuracy
Train accuracy
train accuracy
Epoch
0.95
0.90
0.85
0.80
0.75
0
10
20
30
40
50

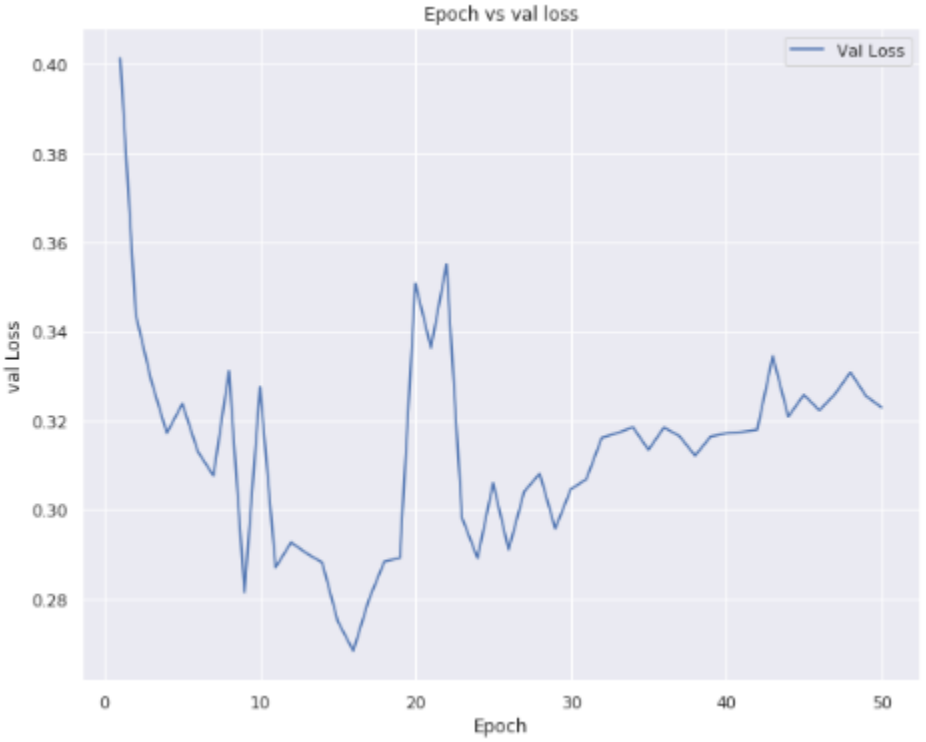

Epoch vs Val accuracy
Val accuracy
Val accuracy
Epoch
0.90
0.89
0.88
0.87
0.86
0.85
0.84
0.83
0
10
20
30
40
50

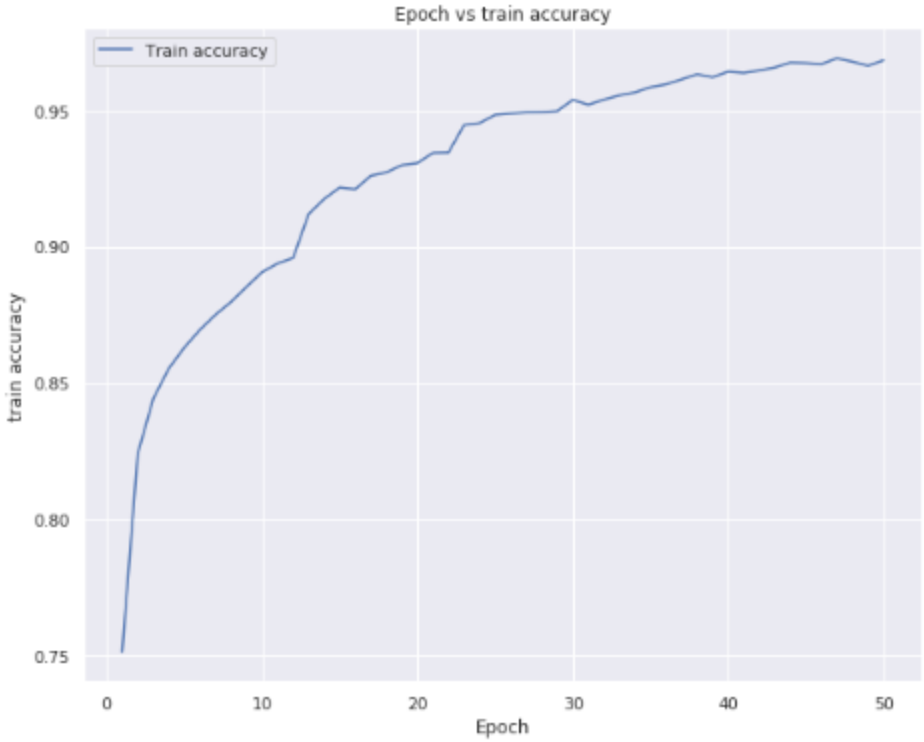

Train accuracy vs Train loss
Train accuracy vs Train loss
0.6
0.5
0.4
0.3
0.2
Train Loss
0.75
0.80
0.85
0.90
0.95
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
0.42
0.40
0.38
0.36
0.34
0.32
0.30
0.28
0.26
Val Loss
0.83
0.84
0.85
0.86
0.87
0.88
0.89
0.90
Val Accuracy

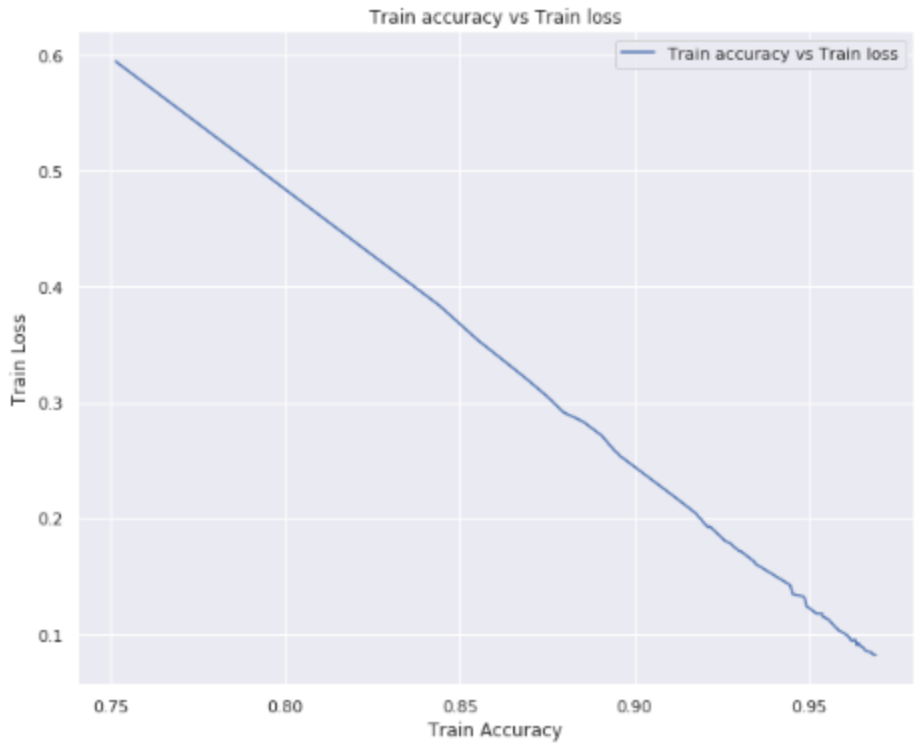
MODEL-4

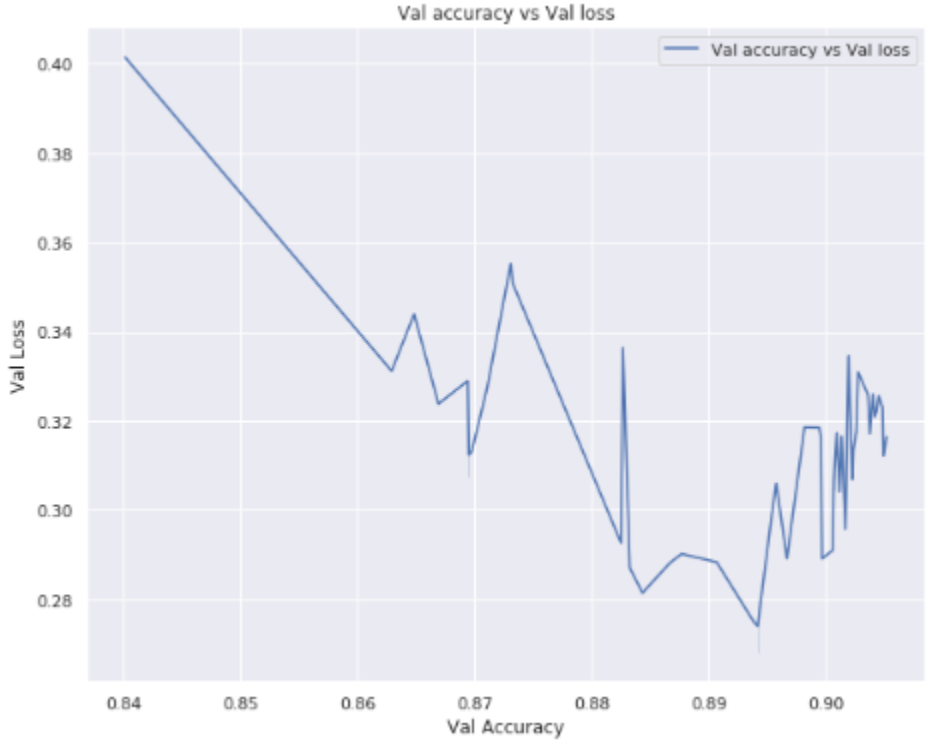

Epoch vs Train loss
Training Loss
Training Loss
0.6
0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50
Epoch


Epoch vs val loss
Val Loss
val Loss
0.40
0.38
0.36
0.34
0.32
0.30
0.28
0
10
20
30
40
50
Epoch

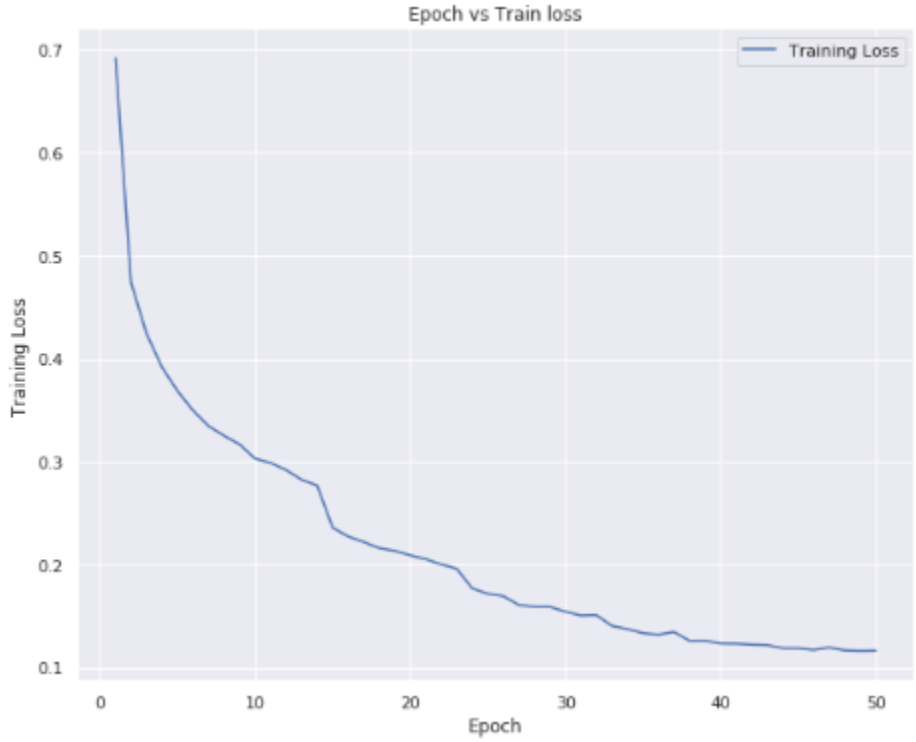

Epoch vs train accuracy
Train accuracy
train accuracy
0.95
0.90
0.85
0.80
0.75
0
10
20
30
40
50
Epoch

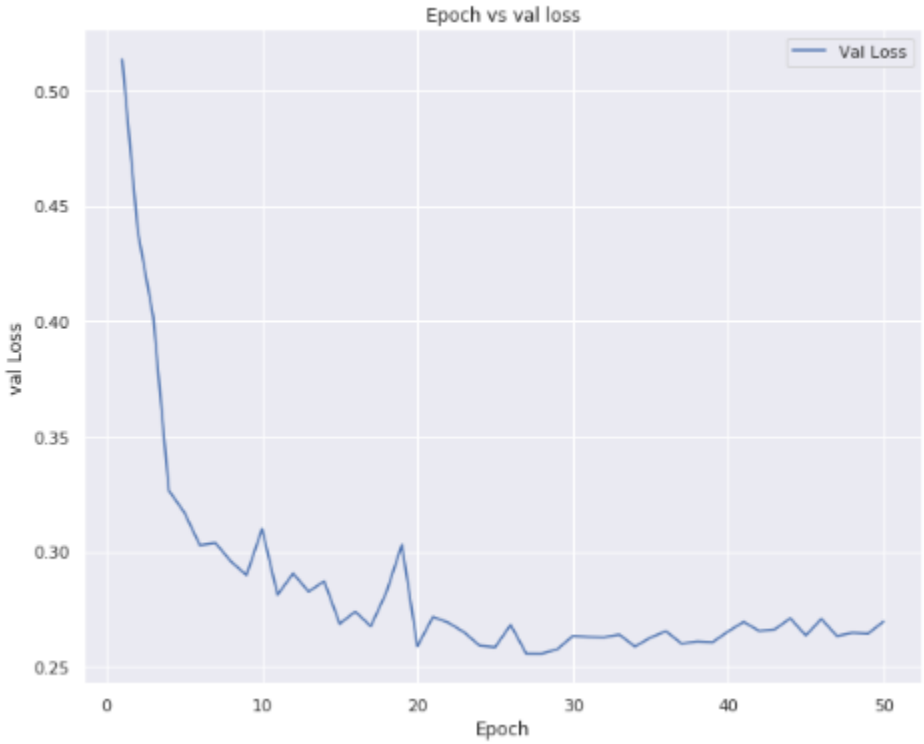

Epoch vs Val accuracy
Val accuracy
Val accuracy
0.90
0.89
0.88
0.87
0.86
0.85
0.84
0
10
20
30
40
50
Epoch

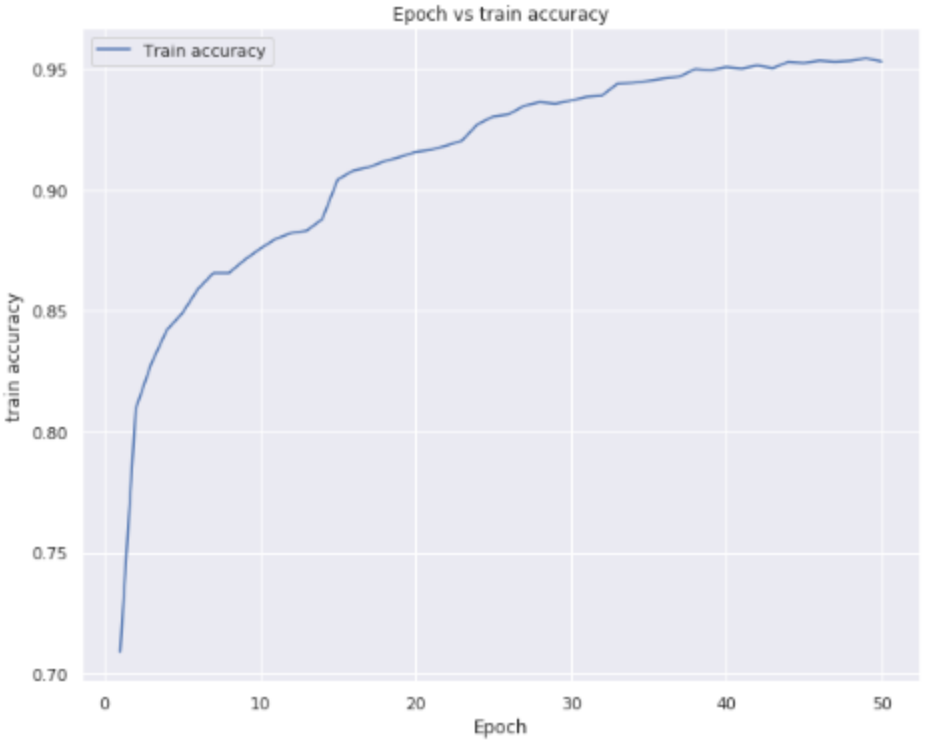

Train accuracy vs Train loss
Train accuracy vs Train loss
0.6
0.5
0.4
0.3
0.2
0.1
Train Loss
0.75
0.80
0.85
0.90
0.95
Train Accuracy

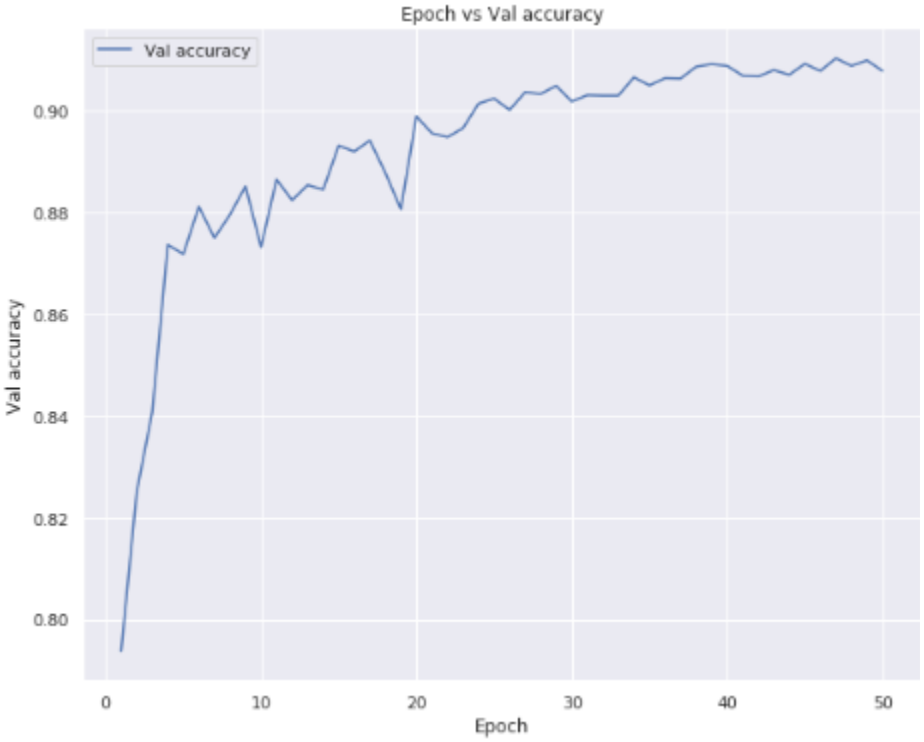

Val accuracy vs Val loss
Val accuracy vs Val loss
0.40
0.38
0.36
0.34
0.32
0.30
0.28
Val Loss
0.84
0.85
0.86
0.87
0.88
0.89
0.90
Val Accuracy

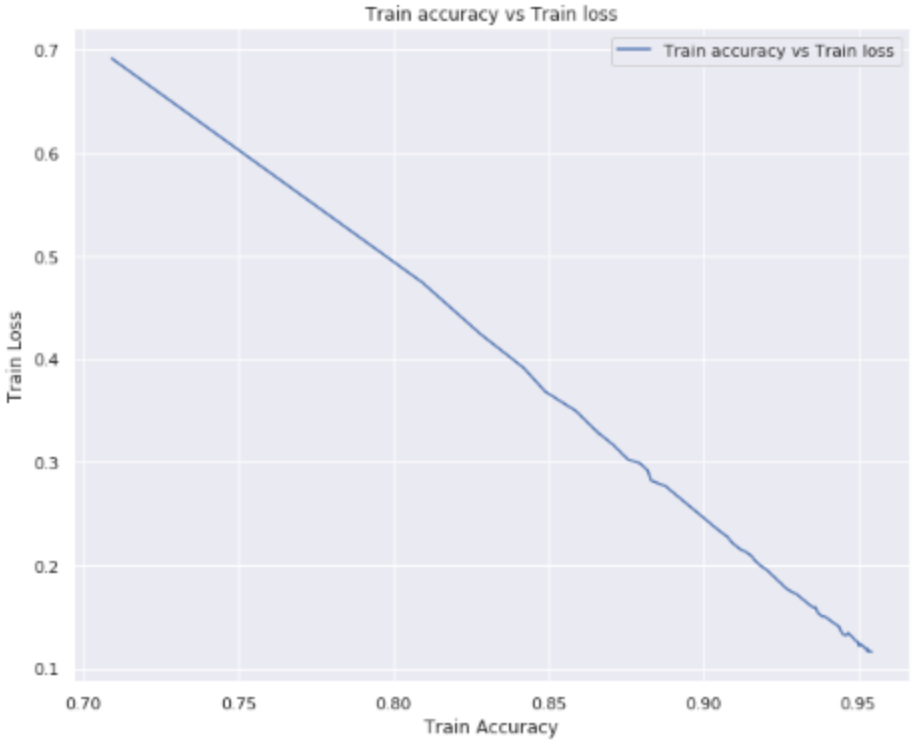
MODEL 5

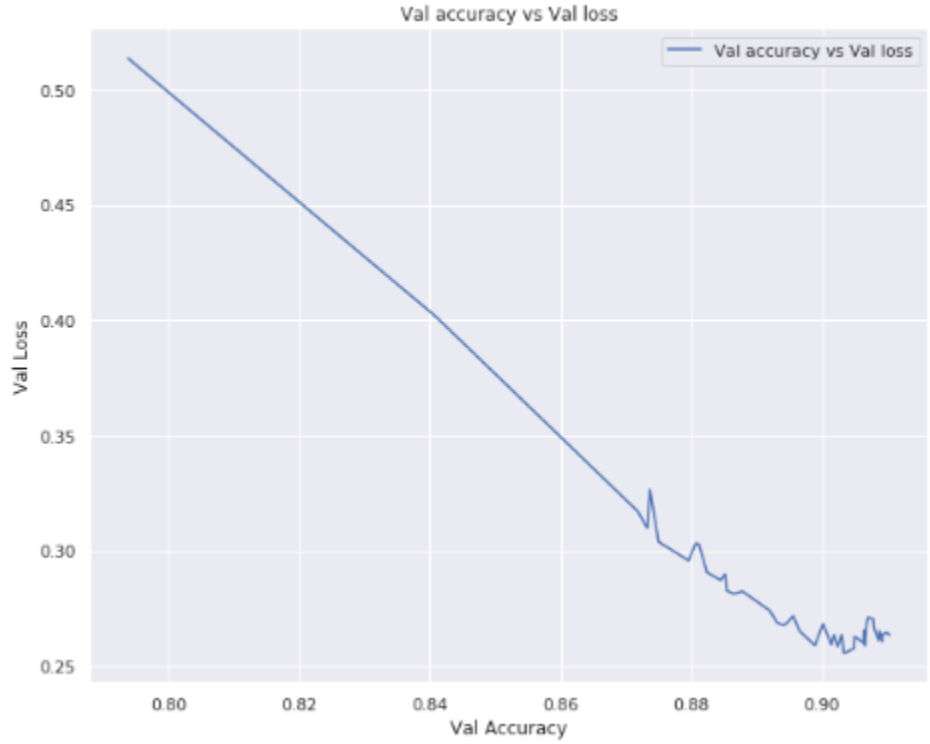

Epoch vs Train loss
Training Loss
Training Loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50
Epoch


Epoch vs val loss
Val Loss
val Loss
0.50
0.45
0.40
0.35
0.30
0.25
0
10
20
30
40
50
Epoch

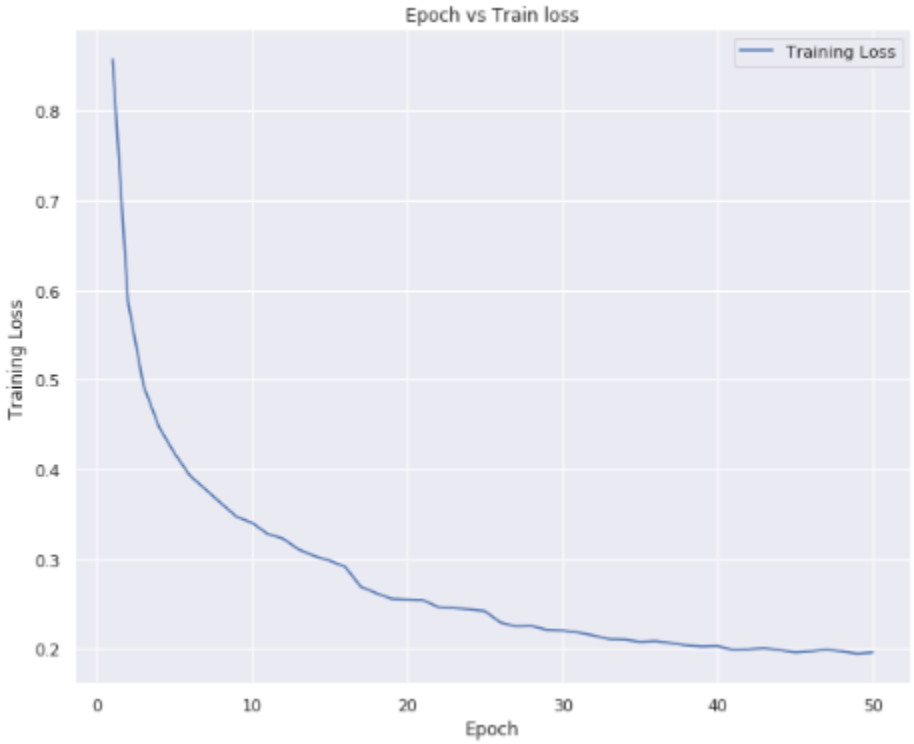

Epoch vs train accuracy
Train accuracy
train accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0
10
20
30
40
50
Epoch

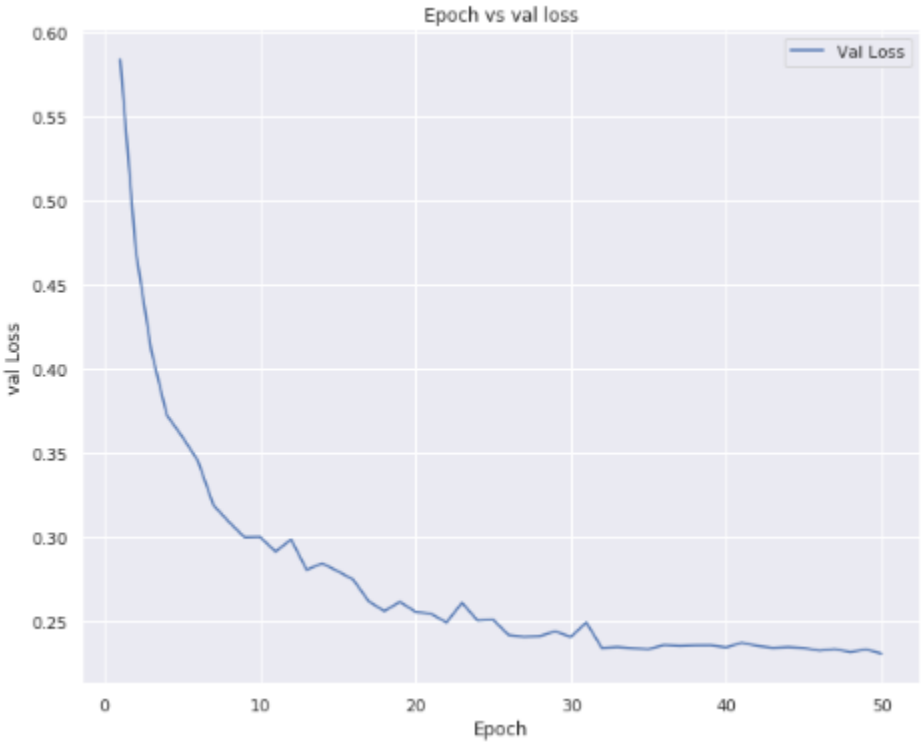

Epoch vs Val accuracy
Val accuracy
Val accuracy
0.90
0.88
0.86
0.84
0.82
0.80
0
10
20
30
40
50
Epoch

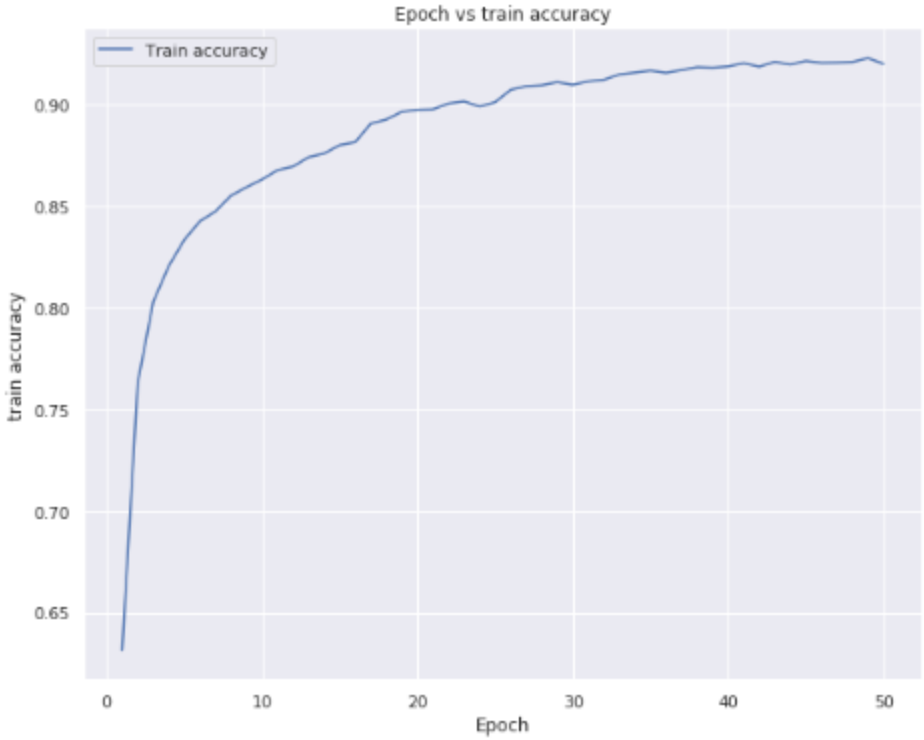

Train accuracy vs Train loss
Train accuracy vs Train loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
Train Loss
0.70
0.75
0.80
0.85
0.90
0.95
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
0.50
0.45
0.40
0.35
0.30
0.25
Val Loss
0.80
0.82
0.84
0.86
0.88
0.90
Val Accuracy

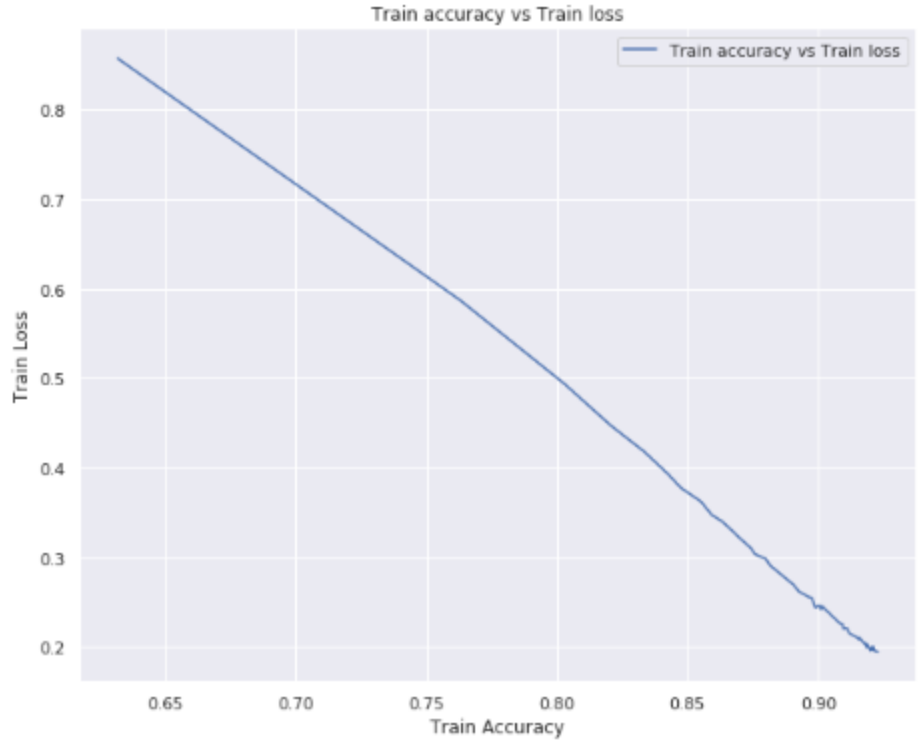
MODEL 6

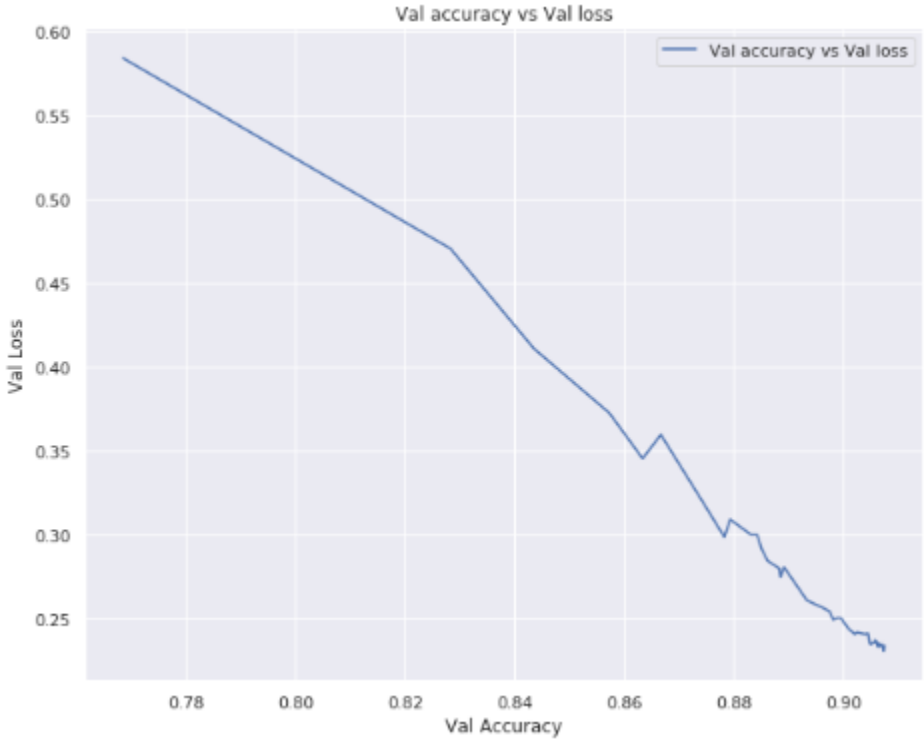

Epoch vs Train loss
Training Loss
Training Loss
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
50
Epoch


Epoch vs val loss
Val Loss
val Loss
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0
10
20
30
40
50
Epoch

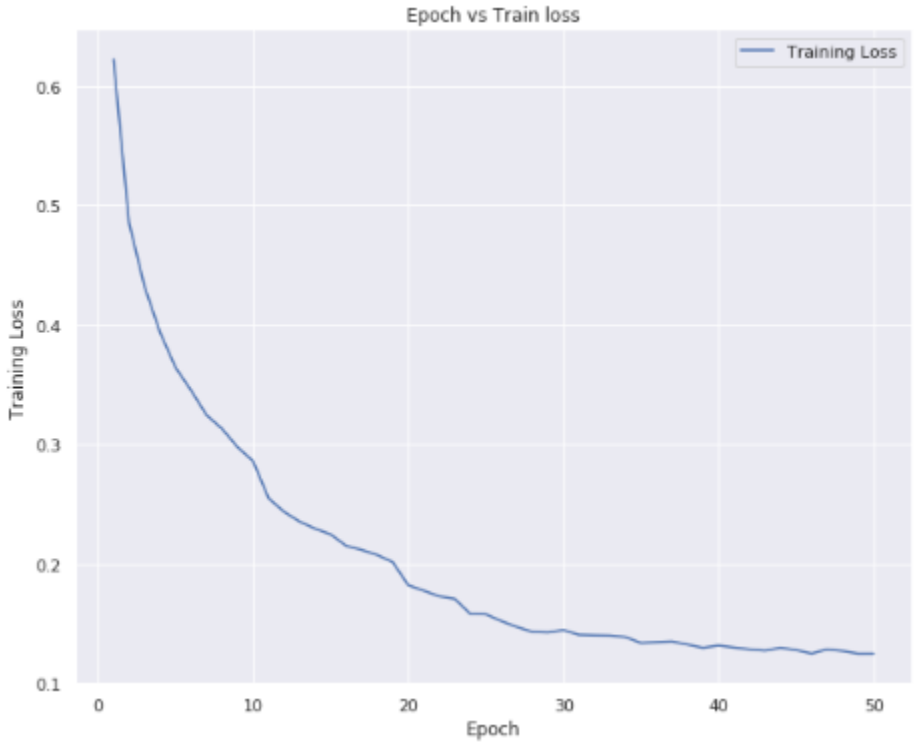

Epoch vs train accuracy
Train accuracy
0.90
0.85
0.80
0.75
0.70
0.65
train accuracy
0
10
20
30
40
50
Epoch

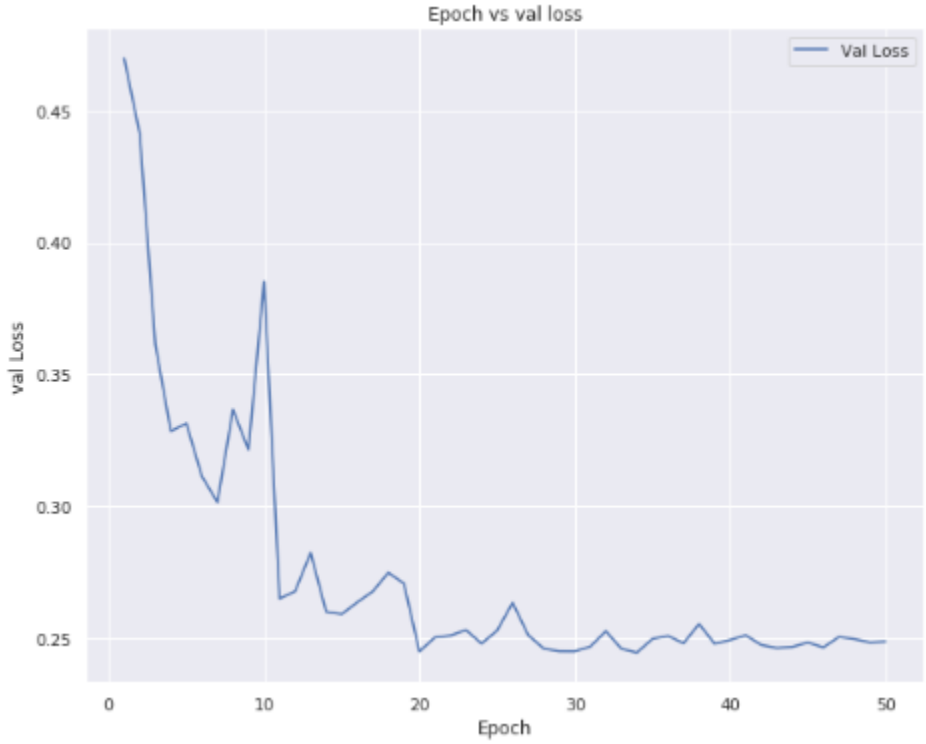

Epoch vs Val accuracy
Val accuracy
0.90
0.88
0.86
0.84
0.82
0.80
0.78
Val accuracy
0
10
20
30
40
50
Epoch

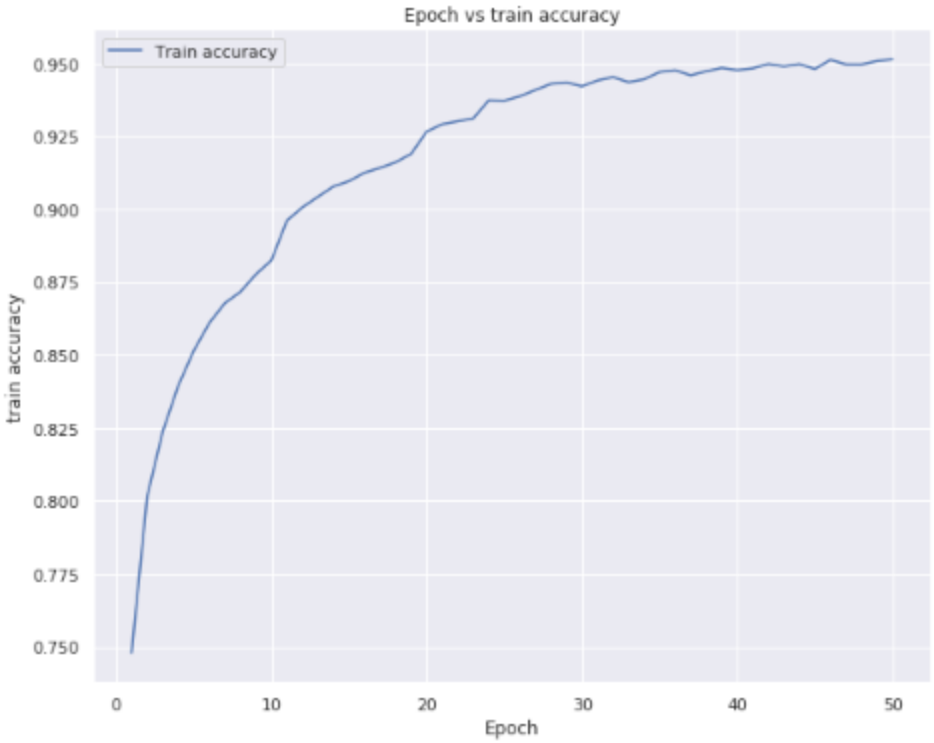

Train accuracy vs Train loss
Train accuracy vs Train loss
Train Loss
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.65
0.70
0.75
0.80
0.85
0.90
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
Val Loss
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.78
0.80
0.82
0.84
0.86
0.88
0.90
Val Accuracy

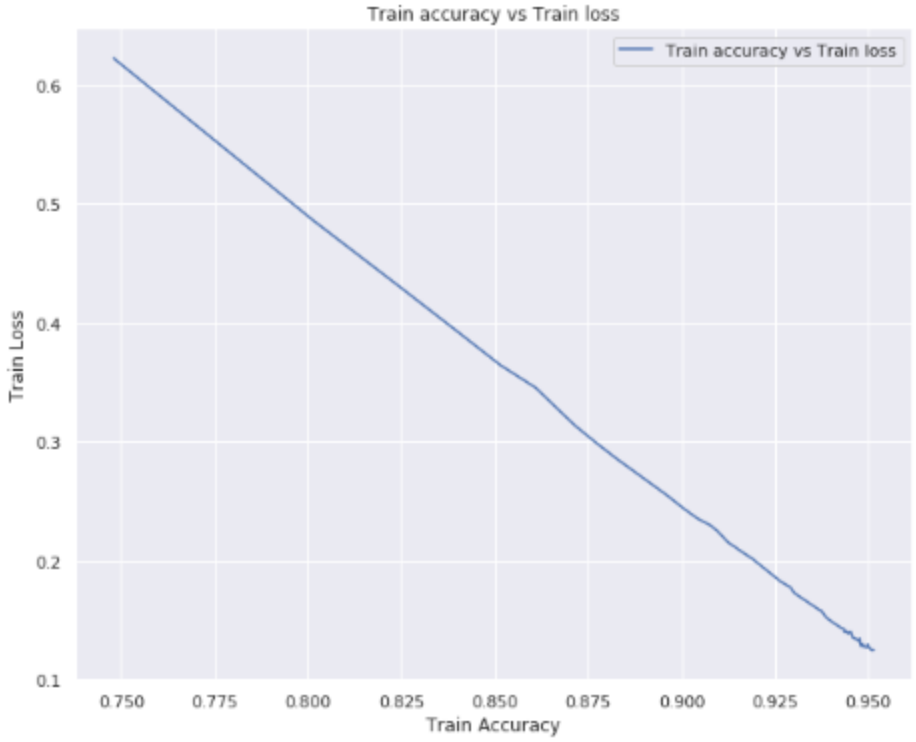
MODLE 7

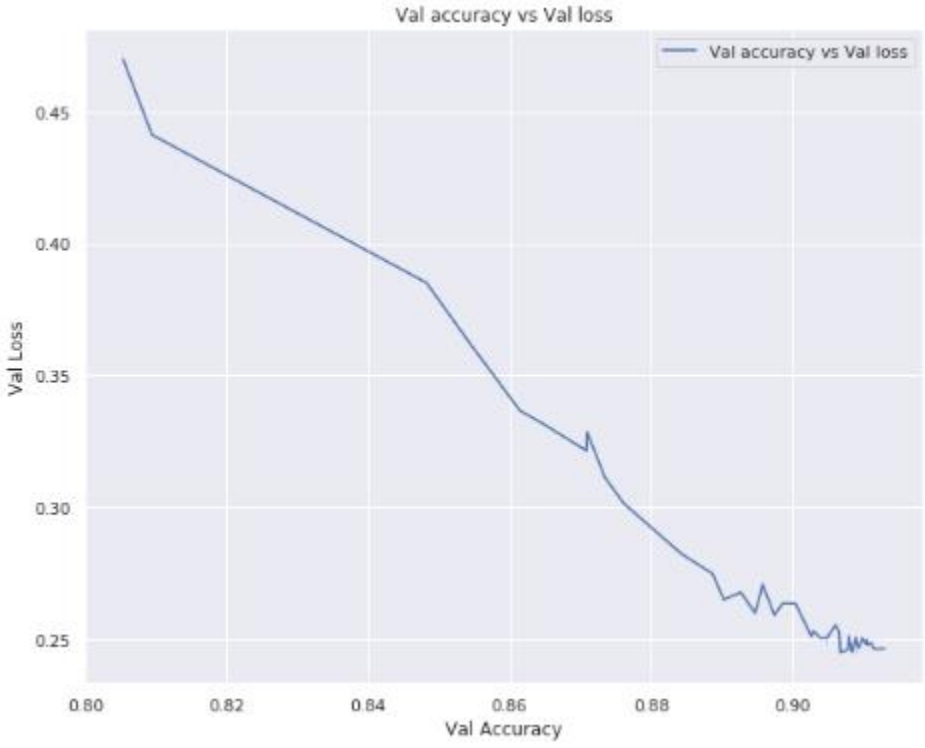

Epoch vs Train loss
Training Loss
Training Loss
0.6
0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50
Epoch


Epoch vs val loss
Val Loss
val Loss
0.45
0.40
0.35
0.30
0.25
0
10
20
30
40
50
Epoch

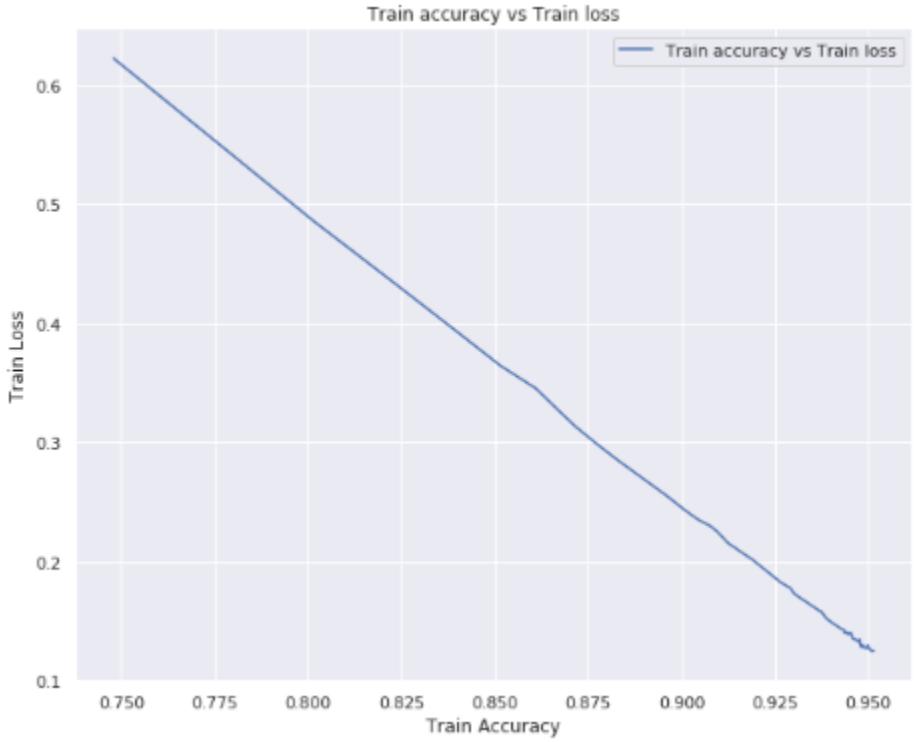

Epoch vs train accuracy
Train accuracy
train accuracy
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0
10
20
30
40
50
Epoch

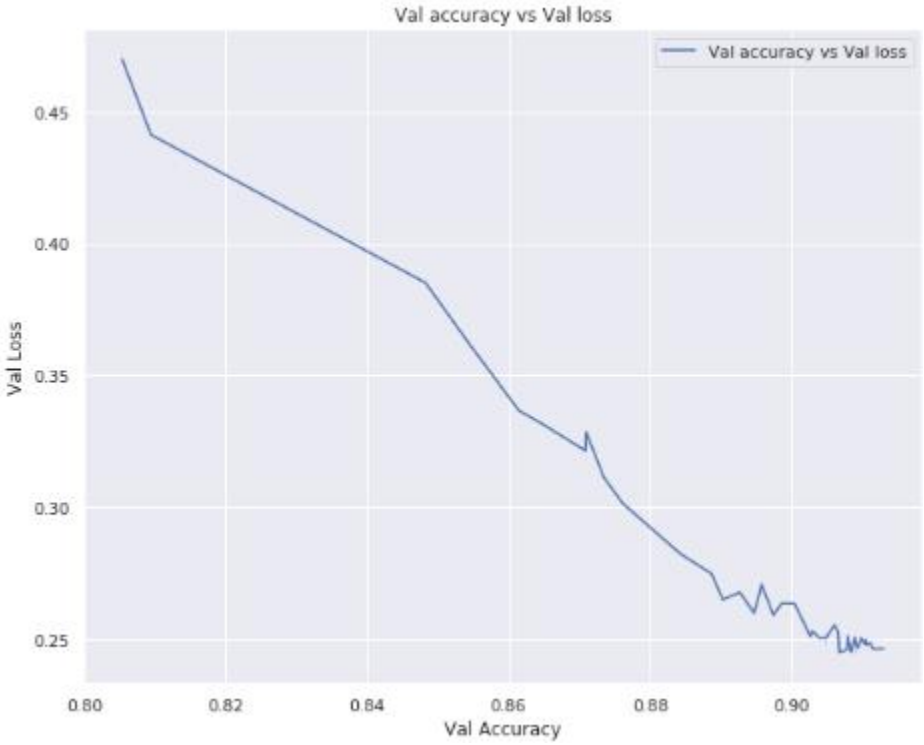

Epoch vs Val accuracy
Val accuracy
Val accuracy
0.90
0.88
0.86
0.84
0.82
0.80
0
10
20
30
40
50
Epoch


Train accuracy vs Train loss
Train accuracy vs Train loss
Train Loss
0.6
0.5
0.4
0.3
0.2
0.1
0.750
0.775
0.800
0.825
0.850
0.875
0.900
0.925
0.950
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
Val Loss
0.45
0.40
0.35
0.30
0.25
0.80
0.82
0.84
0.86
0.88
0.90
Val Accuracy

ANALYZING THE BEST MODEL(7)



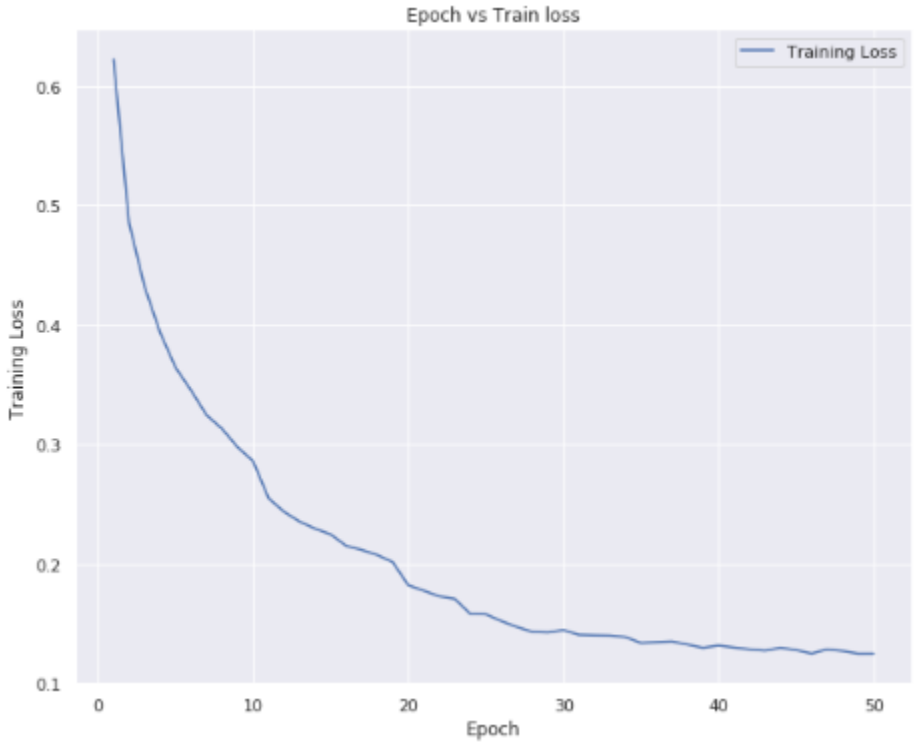


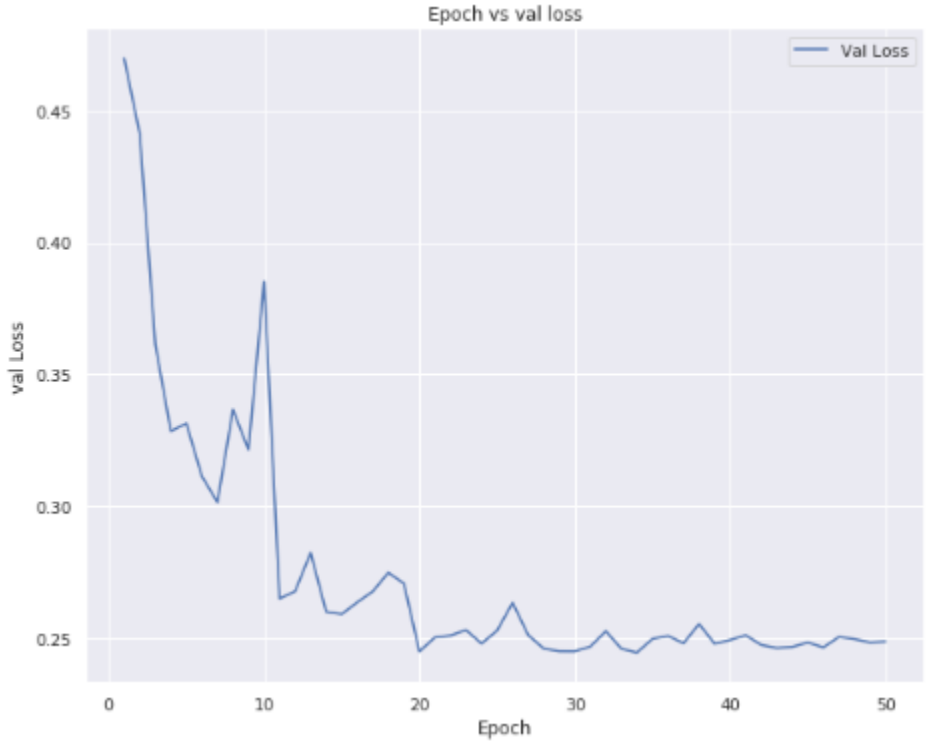
Analysis:

- It was found that using SVD() with batch normalization and callback for learning rate reduction provided the increase in the accuracy from 88.80% to 91.12 % for validation set.

- **In Train loss vs Train accuracy,**
  - It can be clearly seen from the graph that there is a steep increase in the accuracy from 75% to 95.14% .
  - Loss decreased from 0.65 to 0.12 for the training classification with little accumulation of loss values near the baseline of the loss i.e. 0.15 and at the accuracy of more than 90% irrespective of the epochs[for above graphs.]

- **In Test loss vs Test accuracy,**
  - It can be seen that the accuracy for the test set increased from 80% to 91.12% with large number of accuracies accumulating between 88% and 91%.
  - Loss also decreased heavily in the beginning from 0.47 to 0.23 and accumulated very well at 0.25 for large number of epochs.
  - After the 28^{th} epoch, the accuracy is more than 90%.
- Thus, from the graphs, it can be said that our model is generalizing well on the validation data.
- It can be concluded form the graph that our model is not overfitting.


Epoch vs Train loss
Training Loss
Training Loss
0.6
0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50
Epoch


Epoch vs val loss
Val Loss
val Loss
0.45
0.40
0.35
0.30
0.25
0
10
20
30
40
50
Epoch

Analysis:

- **Training Epoch vs Loss:**
  - The loss decreases from 0.61 to 0.11 approximately from Epoch 0 to Epoch 50.
  - There is no Epoch where the loss has visibly increased with a large difference.
- **Testing Epoch vs Loss:**
  - Testing loss decreased from 0.47 to 0.25 within the complete range of Epochs.
  - The stabilization in the loss was achieved when the 28th Epoch was reached.
  - On reaching the 9th Epoch, there is a gradual increase in the loss from 0.32 to 0.38 which can be seen.

	Epoch	train_acc	train_loss	val_acc	val_loss
0	1	0.748000	0.622076	0.805333	0.469666
1	2	0.801542	0.485535	0.809417	0.441103
2	3	0.823750	0.432063	0.854583	0.361485
3	4	0.839500	0.393738	0.871000	0.328491
4	5	0.851500	0.364274	0.864917	0.331434
5	6	0.861000	0.345169	0.873500	0.311421
6	7	0.867813	0.324478	0.876167	0.301561
7	8	0.871667	0.312995	0.861500	0.336631
8	9	0.877688	0.297640	0.870917	0.321426
9	10	0.882500	0.285921	0.848250	0.385105

BENEFIT OF USING SOME TRAINING DATA AS VAL DATA

- Train→ 60000 with labels
- Test→ 10000 without labels
- df_x→ reshaped features→ (60000,28,28,1)
- df_y→ categorical labels→ (60000,5)
- Resizing the image as per the architecture.
- Split into train and validate→ x_train, x_test, y_train, y_test(80:20)
    - x_train: y_train=48000: features and labels
    - x_test: y_test=12000: features and labels

- **We have trained our model on Training set with different metrics and validated it on the validation set which we made in the beginning from the training set**.
- **Thus, we were able to tweak different parameters and checked the performance of the model on those metrics using the validation data.**
- **Moreover, here we used the callback as learning rate reducer which reduced the learning rate by monitoring the validation data accuracy when we were validating the model.**
    - tf.keras.callbacks.ReduceLROnPlateau(
      monitor='val_loss', factor=0.1, patience=10, verbose=0, mode='auto',
      min_delta=0.0001, cooldown=0, min_lr=0, **kwargs
)
      where monitor is the quantity to be monitored and it comes from the train test split that we have done to validate our model.

1. Model has to be trained with some data having labeled input with known outputs. Once the training of the model is done, the model has to be tested on the test data which is not exposed earlier to the model during training. To achieve this, we need to split the dataset into train and test sets using: sklearn.model_selection.train_test_split giving required parameters.

2. If test data is exposed during the training of the model, it will result in over fitting and overestimating the accuracy of the model when the same training

data is tested. This model will not generalize well on the new test data. As a result, the model will become potentially unusable to deploy for practical applications. This problem is also known as Data Leakage.

3. To analyze the generalization of the model we need to test the model with novel and unseen data. Thus, there comes a need of the test dataset.

    1. In order to solve the problem of data leakage, we can divide the dataset into train-validate-test sets.

    2. Thus, the validation dataset which is taken from the training set can be used as the initial test set before performing the actual testing on test dataset.
    3. Overall, we split the dataset into train and test sets to prevent over-fitting, data leakage and isolate the test dataset so that it is not exposed to model while the model is learning.

Example:

 # splitting the dataset into train and test sets in 80:20.

X_train,X_test,y_train,y_test=train_test_split(train.data,train.target,test_size =0.20,train_size=0.80,random_state=42)

Kaggle score

1. Highest accuracy achieved on Kaggle till now is 90.9% with Model 7 i.e.
    a. SGD optimizer
    b. Callback for learning rate reduction
    c. Batch Normalization after each convolution layer i.e. 4 Batch normalizations.
2. Improvement can be done on this score by using more layers, using regularizer, by using the optimizers such as Adadelta, Adagrad.
3. Our Runtime performance with few layers is very impressive and there is no complexity involved in the model for getting this score.

SOME PARAMETER EXPLANATIONS

SGD:

- It stands for Stochastic Gradient Descent. Stochastic means the system is associated with a random probability. So, SGD works on some samples of dataset rather than whole dataset. While calculating the gradient SGD uses batch of only 1 for each iteration which is computationally less expensive as compared to other optimizers.
- So, In SGD we calculate gradient of the cost function of only 1 sample for each iteration which is chosen randomly. Although it takes more iterations to reach the minima because of the randomness of the samples taken per iteration, it is still computationally less expensive than most of the gradient descent used to optimize a learning algorithm.
- Convergence path of SGD is very noisy as compared to other gradient descent methods .Default parameters are learning_rate=0.01, momentum=0.0, nesterov=False. Here momentum is the parameter that takes SGD in suitable direction and calms oscillations. nesterov parameter means whether we want to apply nesterov momentum or not.

ADAM:

- Adam calculates learning rates for each parameter using first and second moment of gradient. So, it is computationally less expensive as compared to **some optimizers** and it can result in high accuracy as well. It uses squared gradients to change the learning rate and it benefits from Gradient by using changing average of the gradient instead of gradient.

$$\text{Moment}(m_n)=E[X^n]$$

- Here $E[X^n]$ is the expected value of a random variable[X] raised to power n [where n is n^{th} moment]. In Adam process, the parameters which are updated are invariant to rescaling of gradient. So, algorithm will converge irrespective of fact that objective function remains same or changes. Adam requires second order derivative to be calculated which results in increase of cost. The default parameters are learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False. Where amgrad basically means whether we choose to apply AMSGrad variant of Adam.

ADADELTA:

- Adadelta is a robust optimizer that manages learning rate based on current gradient updates rather than including all the gradient updates from the past. Adadelta is updated dynamically over the time. It requires no manual training of parameters like learning rate.

$$E[g^2]_t=\gamma E[g^2]_t-1+(1-\gamma)g^2_t$$

- Here $E[g^2]_t$ is the running average at time say t depends only on previous average and current gradient. The default parameters are learning_rate=1.0, rho=0.95 .Here rho is adadelta decay factor which tells us fraction of gradient to keep in each iterative step.

BATCH NORMALIZATION:

- Batch normalization normalizes the input that is presented to each layer in convolutional Neural Network to resolve the problem of internal covariance shift. We normalize the data before using CNN to have general distribution with zero mean and unitary variance and hence make sure that all the input data is in same range of values. But some abnormalities may arise between layers because of changing distribution of actions and this problem is known as **internal covariance shift.** This dampens learning because now each layer must adjust to the new distribution. So, batch normalization is done to counter this **problem of internal covariance shift.**
- **Steps in batch normalization** that are followed in training stage:
    - Mean and variance of input to layers is calculated
    - Normalize the input using previous calculated batch statistics
    - Scale and shift to find the output of the layer

Learning Rate:

- Learning rate basically means how the model adapts to the loss problem it encounters. Now basically when we use optimizers, error gradient is calculated and then weights of the neural network layers are updated using back propagation. The portion in which weights are changed is called the **learning rate .**
- In other words if learning rate is very less then weights are updated by a very small amount and it takes large number of epochs to arise at minimum loss and hence converge slowly while if the learning rate is very high the model converges quickly as weights are updated by a large amount.
- Usually there is trade-off between the two as quick convergence also means that algorithm is not learning properly and taking harsh steps while if convergence is slow that means algorithm is taking a whole lot of time to reach the minima and hence generalize. Value of learning rate is between 0.0 to 1.0.

Callbacks:

- Callback is set of functions that are applied in training stage to have control over the training procedure. Callback gives us internal state view of the model and its statistics during training. Using Callbacks, when accuracy and loss of a model is at satisfactory level, we can stop the training, we can save the model after individual epochs, we can also adapt learning rate over the time etc. Early stopping is excellent way to avoid overfitting. **Some metrics under early stopping are:**

- **Monitor**: Here we mention the value that we want to monitor. (val_accuracy)
- **Min_delta**: It refers to minimum change in value which is monitored. For example: if min_delta is 0.5 and change in the monitored value is less than 0.5, the training will be stopped.
- **Patience**: Number of epochs with no improvement that will be tolerated before training process is stopped.
- **restore_best_weights:** it is set to true if we want to save the best weights once training is stopped.
- **Factor:** By which the learning rate will be reduced.

Note: ReduceLROnPlateau -> our usage.

Dropout:

- Sometimes in a neural network, specific nodes might always be giving the best answer. This makes model neglect the other nodes and hence , their weights may not be updated accordingly. When we use dropout in our model, we drop certain percentage of nodes and also their connection. So, in a way we forbid some nodes, and this gives chance to other nodes and forces them to learn. Hence drop out is a mechanism used to **prevent overfitting.**

Flatten:

- Flatten layer changes the shape of the data from a 2D vector of matrices into correct form for dense layer to interpret. It is generally used after the max pooling or average pooling in a convolution neural network used to convert the pooled feature maps to a single column as required by the dense layer.

Note: All the parameters used are also described where they are used.

VGG

ARCHITECTURE

- It is a 16-layer architecture consisting of:
  - Convolution Layers→ 13 Layers
  - Max Pooling Layers→ 5 Layers
  - Activation Layers and
  - Fully connected Layers→ 3 Layers

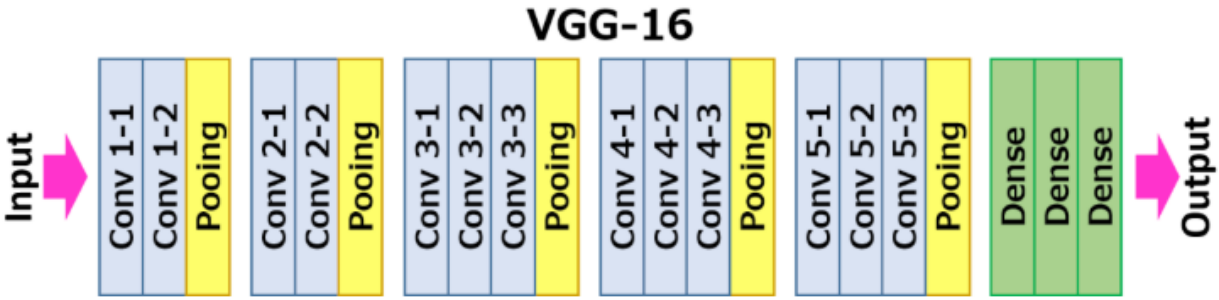


Fig 1: Basic Architecture of VGG 16

- **Convolution Layers:**
  - **Layer 1→ 64**
  - **Layer 2→ 128**
  - **Layer 3→ 256**
  - **Layer 4→ 512**
  - **Layer 5→ 512**

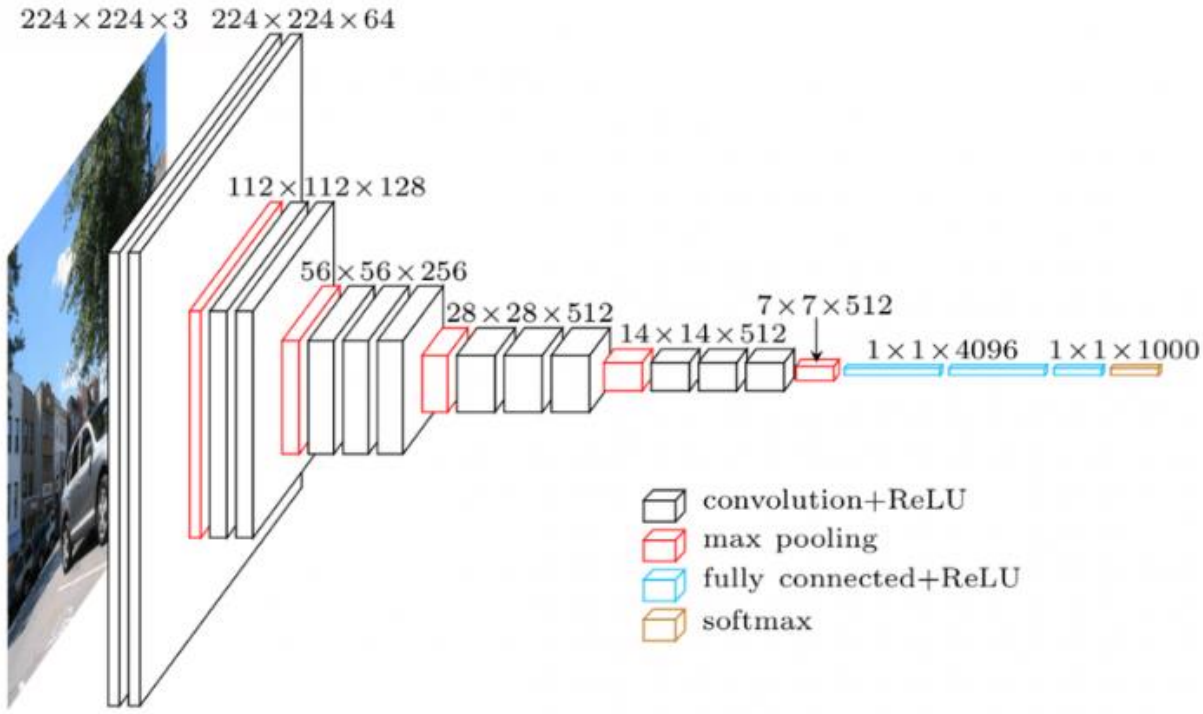


Fig. 2: VGG 16

- **Convolution Layers used only 3*3 filters whereas the maxpooling layers used only 2*2 filters.**

CODE BLOCKS AND ARCHITECTURE

IMPORTS AND DATA LOAD

VGG 16 MANUAL --

In [68]:

```
from __future__ import print_function
import numpy as np
np.random.seed(1337)
import keras
from keras.layers import MaxPooling2D, AveragePooling2D, Input, Flatten ,Dropout,ZeroPadding2D,
Dense,Convolution2D,BatchNormalization,Activation
from keras.optimizers import Adam,SGD, Adadelta
from keras.models import Model
from sklearn.model_selection import train_test_split
import keras,os
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D , Flatten

import matplotlib.pyplot as plt
%matplotlib inline
# This Python 3 environment comes with many helpful analytics libraries installed
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/ece657a-w20-asg3-part2/testX.csv
/kaggle/input/ece657a-w20-asg3-part2/train.csv
```

RESIZING THE IMAGE(Rest steps similar to CNN)

Resizing the data to 48*48--

In [26]:
```
#RESHAPE according to vgg
from keras.preprocessing.image import img_to_array, array_to_img
x_train = np.asarray([img_to_array(array_to_img(im, scale=False).resize((48,48))) for im in x_t
rain])
x_test = np.asarray([img_to_array(array_to_img(im, scale=False).resize((48,48))) for im in x_te
st])
# train_x = preprocess_input(x)
x_train.shape, x_test.shape
```

Out[26]:
```
((48000, 48, 48, 1), (12000, 48, 48, 1))
```

Note: We are not reshaping it into 48*48 not 224*224.

Resizing the real test data into 48*48--

In [126]:
```
df_x_test = np.asarray([img_to_array(array_to_img(im, scale=False).resize((48,48))) for im in d
f_x_test])
```

In [127]:
```
df_x_test.shape
```

Out[127]:
```
(10000, 48, 48, 1)
```

MODEL -1: (Tweaked model)

- **We have tweaked the model by changing layers and parameters to get the best accuracy while minimizing the time complexity.**

Layer and dropouts	Count
Convolution	10
Maxpooling	4
Dropout	2 (0.5)
Dense	3

Optimizer	SGD()
Epoch	50
Batch Size	64

- We are using 10 convolution layers with 4 max pooling layers with SGD() as the optimizer which we concluded as better than the Adam() for our dataset as of now.
- Batch size is increased to 64 as we are increasing the number of layers.

MODEL 1--SGD() | BS: 64 | EPOCH: 50 ----------

In [129]:

```
model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(48,48,1)))
model.add(Conv2D(filters=64, kernel_size=(3,3),activation="relu"))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=64, kernel_size=(3,3),activation="relu"))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=128, kernel_size=(3,3),activation="relu"))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=128, kernel_size=(3,3),activation="relu"))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=256, kernel_size=(3,3),activation="relu"))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=256, kernel_size=(3,3),activation="relu"))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=256, kernel_size=(3,3),activation="relu"))
model.add(MaxPooling2D((2,2), strides=(2,2)))
```

```
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=512, kernel_size=(3,3),activation="relu"))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=512, kernel_size=(3,3),activation="relu"))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=512, kernel_size=(3,3),activation="relu"))
model.add(MaxPooling2D((2,2), strides=(2,2)))

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

In [131]:

```
from keras.optimizers import Adam
from keras import optimizers
opt = Adam()
sgd = SGD()

model.compile(optimizer=sgd, loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])
```

SUMMARY

In [130]:

```
model.summary()
```

```
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d_146 (ZeroPadd (None, 50, 50, 1)         0
_________________________________________________________________
conv2d_250 (Conv2D)          (None, 48, 48, 64)        640
_________________________________________________________________
zero_padding2d_147 (ZeroPadd (None, 50, 50, 64)        0
_________________________________________________________________
conv2d_251 (Conv2D)          (None, 48, 48, 64)        36928
_________________________________________________________________
max_pooling2d_95 (MaxPooling (None, 24, 24, 64)        0
_________________________________________________________________
zero_padding2d_148 (ZeroPadd (None, 26, 26, 64)        0
_________________________________________________________________
conv2d_252 (Conv2D)          (None, 24, 24, 128)       73856
_________________________________________________________________
zero_padding2d_149 (ZeroPadd (None, 26, 26, 128)       0
_________________________________________________________________
conv2d_253 (Conv2D)          (None, 24, 24, 128)       147584
_________________________________________________________________
max_pooling2d_96 (MaxPooling (None, 12, 12, 128)       0
_________________________________________________________________
zero_padding2d_150 (ZeroPadd (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_254 (Conv2D)          (None, 12, 12, 256)       295168
_________________________________________________________________
zero_padding2d_151 (ZeroPadd (None, 14, 14, 256)       0
```

```
_________________________________________________________________
conv2d_255 (Conv2D)          (None, 12, 12, 256)       590080
_________________________________________________________________
zero_padding2d_152 (ZeroPadd (None, 14, 14, 256)       0
_________________________________________________________________
conv2d_256 (Conv2D)          (None, 12, 12, 256)       590080
_________________________________________________________________
max_pooling2d_97 (MaxPooling (None, 6, 6, 256)         0
_________________________________________________________________
zero_padding2d_153 (ZeroPadd (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_257 (Conv2D)          (None, 6, 6, 512)         1180160
_________________________________________________________________
zero_padding2d_154 (ZeroPadd (None, 8, 8, 512)         0
_________________________________________________________________
conv2d_258 (Conv2D)          (None, 6, 6, 512)         2359808
_________________________________________________________________
zero_padding2d_155 (ZeroPadd (None, 8, 8, 512)         0
_________________________________________________________________
conv2d_259 (Conv2D)          (None, 6, 6, 512)         2359808
_________________________________________________________________
max_pooling2d_98 (MaxPooling (None, 3, 3, 512)         0
_________________________________________________________________
zero_padding2d_156 (ZeroPadd (None, 5, 5, 512)         0
_________________________________________________________________
conv2d_260 (Conv2D)          (None, 3, 3, 512)         2359808
_________________________________________________________________
zero_padding2d_157 (ZeroPadd (None, 5, 5, 512)         0
_________________________________________________________________
```

```
_________________________________________________________________
conv2d_261 (Conv2D)          (None, 3, 3, 512)         2359808
_________________________________________________________________
zero_padding2d_158 (ZeroPadd (None, 5, 5, 512)         0
_________________________________________________________________
conv2d_262 (Conv2D)          (None, 3, 3, 512)         2359808
_________________________________________________________________
max_pooling2d_99 (MaxPooling (None, 1, 1, 512)         0
_________________________________________________________________
flatten_20 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_58 (Dense)             (None, 4096)              2101248
_________________________________________________________________
dropout_23 (Dropout)         (None, 4096)              0
_________________________________________________________________
dense_59 (Dense)             (None, 4096)              16781312
_________________________________________________________________
dropout_24 (Dropout)         (None, 4096)              0
_________________________________________________________________
dense_60 (Dense)             (None, 5)                 20485
=================================================================
Total params: 33,616,581
Trainable params: 33,616,581
Non-trainable params: 0
_________________________________________________________________
```

FITTING THE MODEL

In [132]:
```
import timeit
start=timeit.default_timer()
a=model.fit(x_train,y_train,batch_size=64,epochs = 50,verbose=1,validation_data=(x_test,y_tes
t),shuffle=False)
stop=timeit.default_timer()
```

ACCURACY AND TIME

RESULT:

Epoch 50/50 |val_loss: 0.6540 - val_accuracy: 0.8603 | Time taken 1751.951351571999

In [133]:
```
print('Time taken',stop-start)
```

```
Time taken 1751.951351571999
```

PLOTS

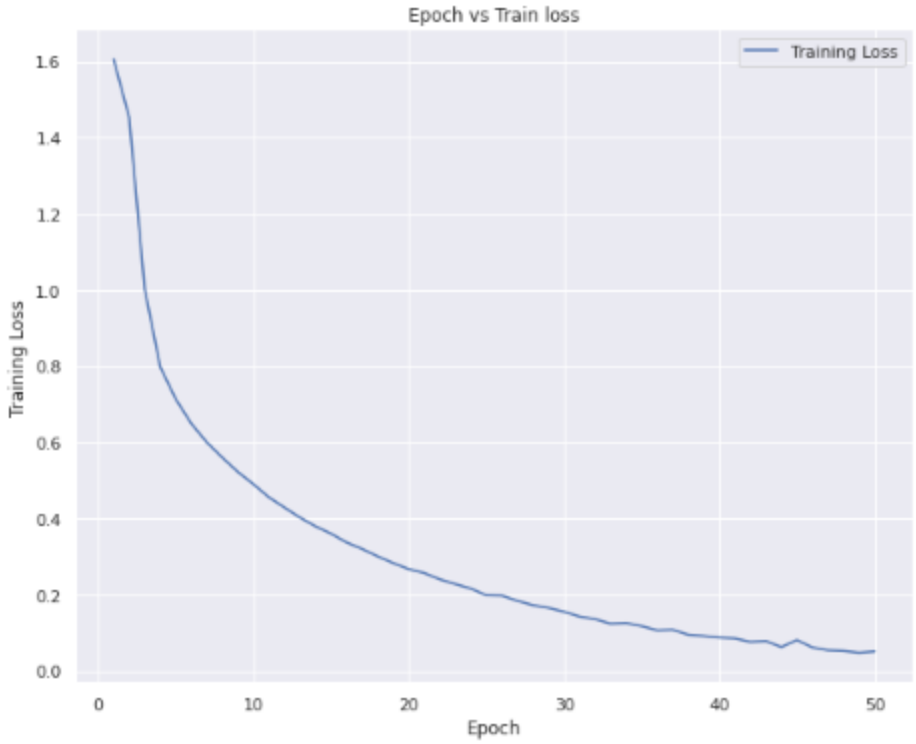
PLOTS

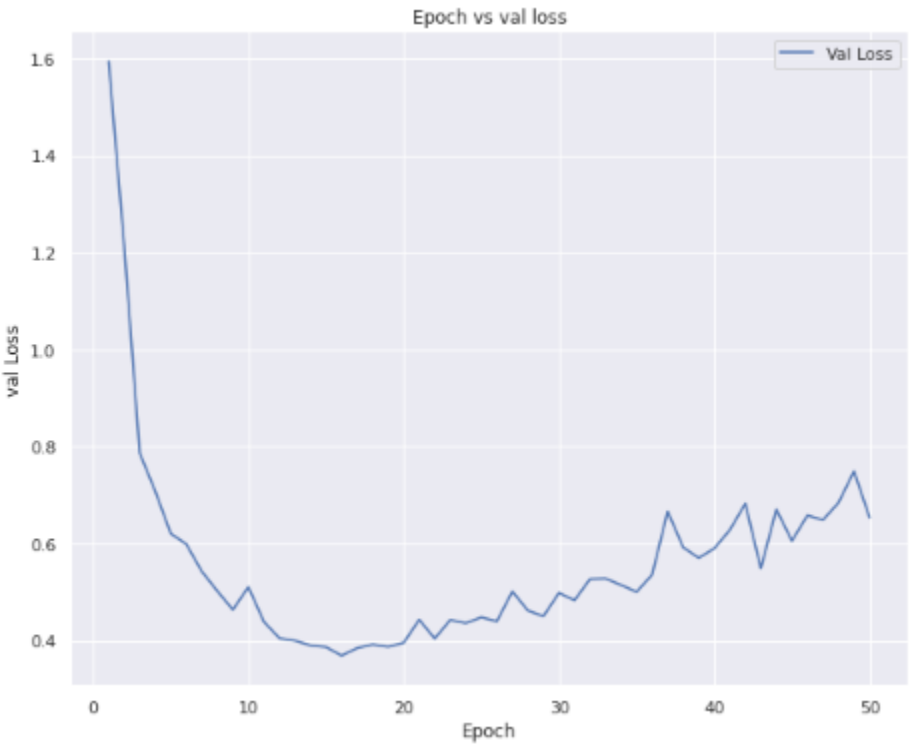
In [134]:
```
a.history
```

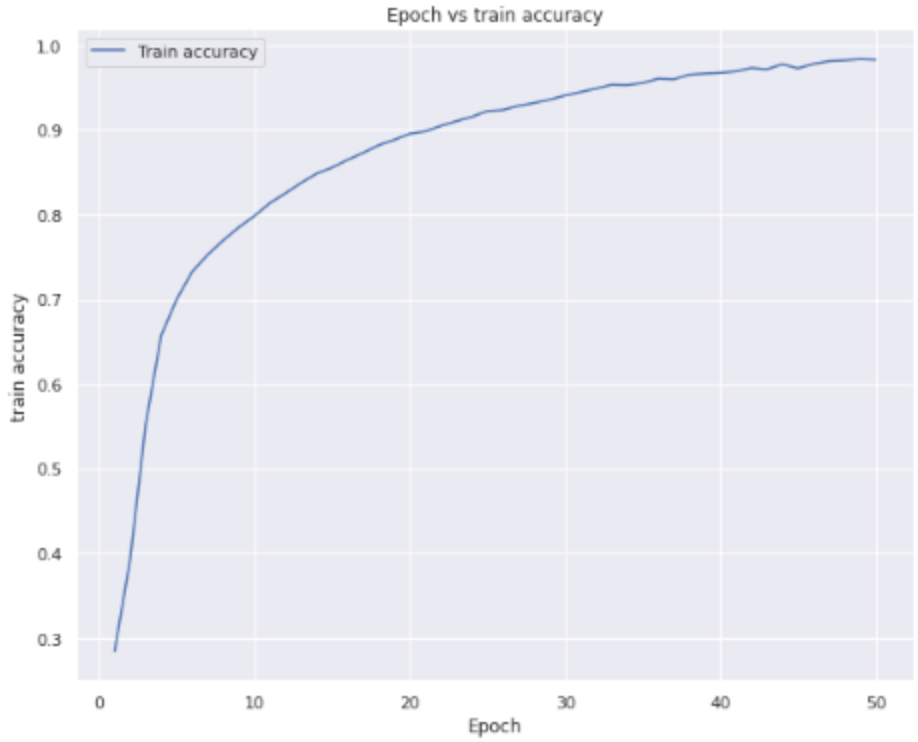
Epoch vs Train loss

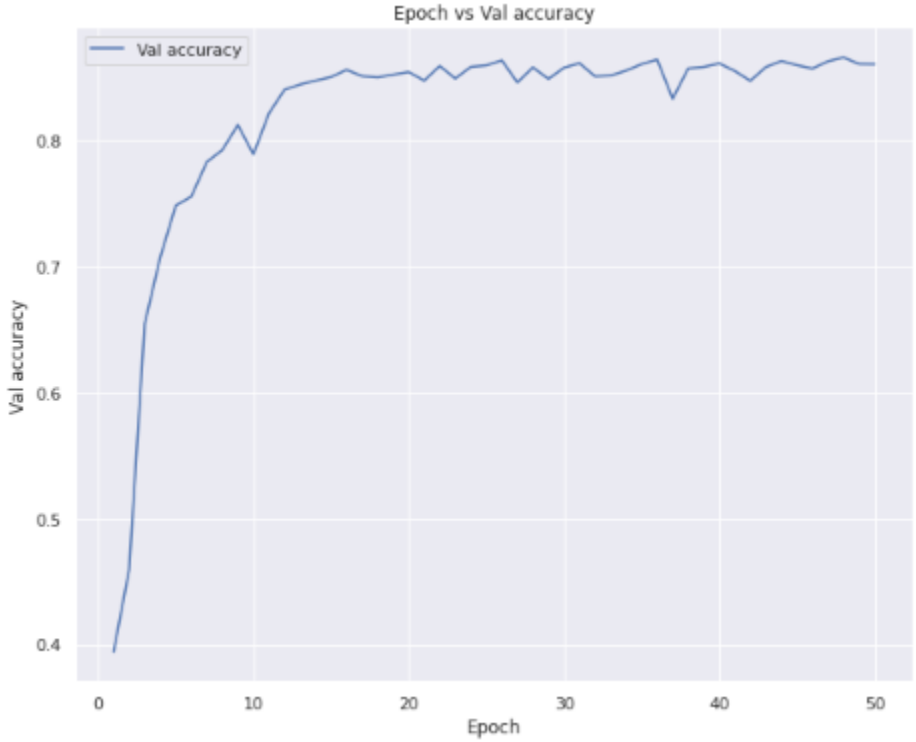
In [141]:
```
import seaborn as sns;
fig= plt.figure(figsize=(10,8))
sns.set(style="darkgrid")
ax=sns.lineplot(x='Epoch',y='train_loss',data=history_df1)
plt.title("Epoch vs Train loss")

plt.xlabel("Epoch")
plt.ylabel("Training Loss")
plt.legend(["Training Loss"])
plt.show()
```

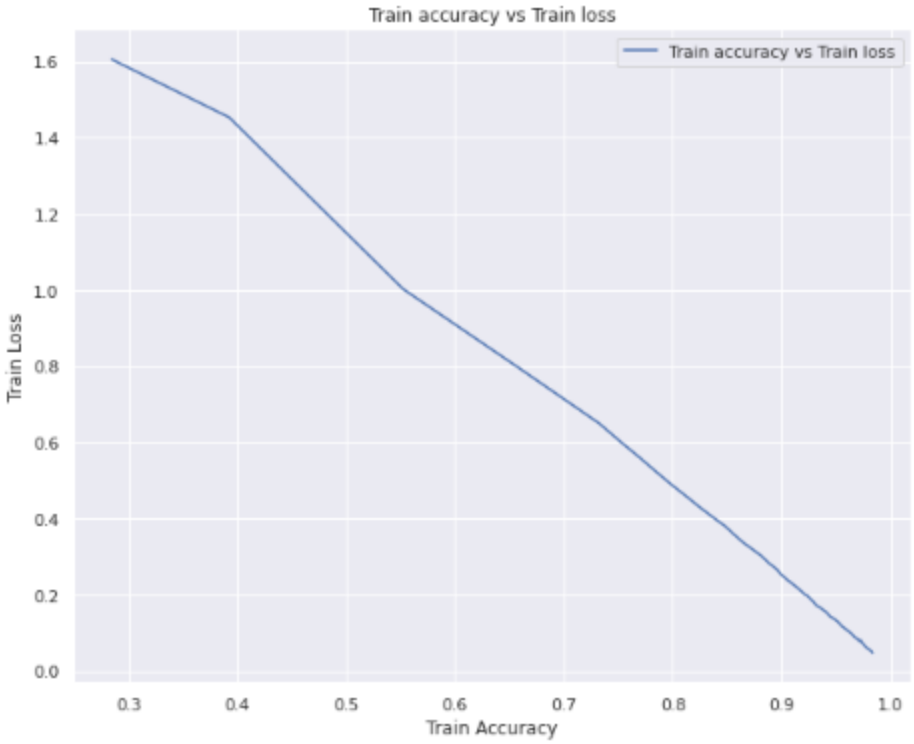
Note: Similarly we do the rest of the plots.

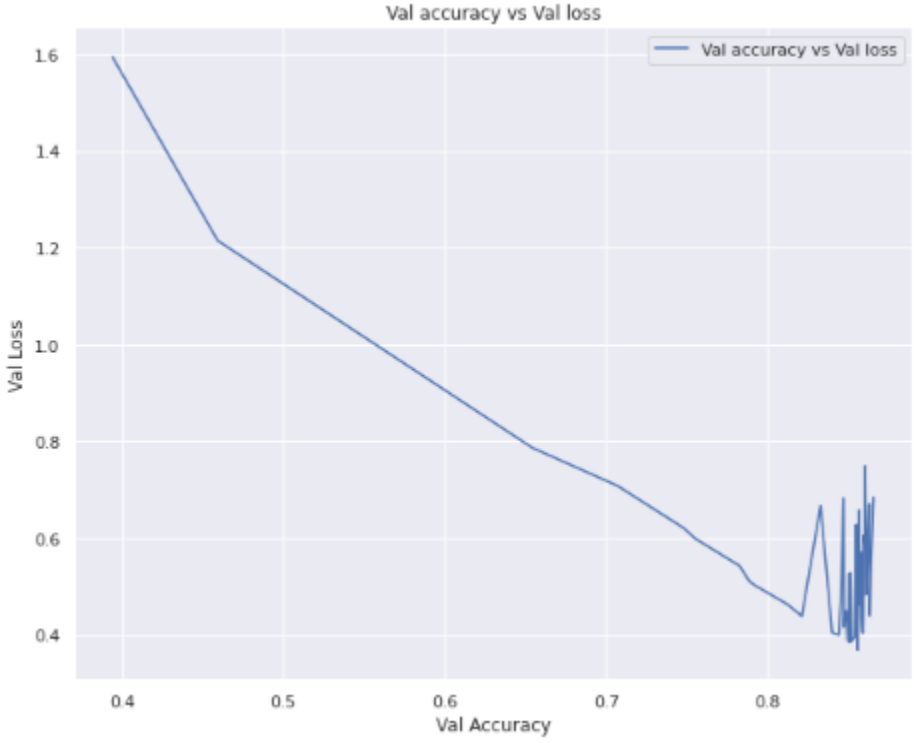

Epoch vs Train loss
Training Loss
Training Loss
Epoch
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50


Epoch vs val loss
Val Loss
Val Loss
Epoch
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
10
20
30
40
50


Epoch vs train accuracy
Train accuracy
train accuracy
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
10
20
30
40
50


Epoch vs Val accuracy
Val accuracy
Val accuracy
Epoch
0.8
0.7
0.6
0.5
0.4
0
10
20
30
40
50


Train accuracy vs Train loss
Train accuracy vs Train loss
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Train Loss
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
1.6
1.4
1.2
1.0
0.8
0.6
0.4
Val Loss
0.4
0.5
0.6
0.7
0.8
Val Accuracy

Note: we tried the Model 1 by changing the learning rate and momentum, but it didn't work well. Code can be found in the notebook.

MODEL -3: (Tweaked model)

Layer and dropouts	Count
Convolution	10
Maxpooling	4
Dropout	2 (0.5)
Dense	3

Optimizer	Adam()
Epoch	50
Batch Size	64

- We are using 10 convolution layers with 4 max pooling layers with Adam() as the optimizer.

- Batch size is increased to 64 as we are increasing the number of layers.

MODEL 3--Adam() | BS: 64 | EPOCH: 30 ----------

In [216]:

```
model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(48,48,1)))
model.add(Conv2D(filters=64, kernel_size=(3,3),activation="relu"))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=64, kernel_size=(3,3),activation="relu"))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=128, kernel_size=(3,3),activation="relu"))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=128, kernel_size=(3,3),activation="relu"))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=256, kernel_size=(3,3),activation="relu"))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=256, kernel_size=(3,3),activation="relu"))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=256, kernel_size=(3,3),activation="relu"))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=512, kernel_size=(3,3),activation="relu"))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=512, kernel_size=(3,3),activation="relu"))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=512, kernel_size=(3,3),activation="relu"))
model.add(MaxPooling2D((2,2), strides=(2,2)))
```

```
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=512, kernel_size=(3,3),activation="relu"))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=512, kernel_size=(3,3),activation="relu"))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(filters=512, kernel_size=(3,3),activation="relu"))
model.add(MaxPooling2D((2,2), strides=(2,2)))

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

In [217]:

```
model.summary()
```

```
Model: "sequential_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d_276 (ZeroPadd (None, 50, 50, 1)         0
_________________________________________________________________
conv2d_380 (Conv2D)          (None, 48, 48, 64)        640
_________________________________________________________________
zero_padding2d_277 (ZeroPadd (None, 50, 50, 64)        0
_________________________________________________________________
conv2d_381 (Conv2D)          (None, 48, 48, 64)        36928
_________________________________________________________________
max_pooling2d_145 (MaxPoolin (None, 24, 24, 64)        0
_________________________________________________________________
zero_padding2d_278 (ZeroPadd (None, 26, 26, 64)        0
```

```
_________________________________________________________________
conv2d_382 (Conv2D)          (None, 24, 24, 128)       73856
_________________________________________________________________
zero_padding2d_279 (ZeroPadd (None, 26, 26, 128)       0
_________________________________________________________________
conv2d_383 (Conv2D)          (None, 24, 24, 128)       147584
_________________________________________________________________
max_pooling2d_146 (MaxPoolin (None, 12, 12, 128)       0
_________________________________________________________________
zero_padding2d_280 (ZeroPadd (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_384 (Conv2D)          (None, 12, 12, 256)       295168
_________________________________________________________________
zero_padding2d_281 (ZeroPadd (None, 14, 14, 256)       0
_________________________________________________________________
conv2d_385 (Conv2D)          (None, 12, 12, 256)       590080
_________________________________________________________________
zero_padding2d_282 (ZeroPadd (None, 14, 14, 256)       0
_________________________________________________________________
conv2d_386 (Conv2D)          (None, 12, 12, 256)       590080
_________________________________________________________________
max_pooling2d_147 (MaxPoolin (None, 6, 6, 256)         0
_________________________________________________________________
zero_padding2d_283 (ZeroPadd (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_387 (Conv2D)          (None, 6, 6, 512)         1180160
_________________________________________________________________
zero_padding2d_284 (ZeroPadd (None, 8, 8, 512)         0
_________________________________________________________________
conv2d_388 (Conv2D)          (None, 6, 6, 512)         2359808
_________________________________________________________________
zero_padding2d_285 (ZeroPadd (None, 8, 8, 512)         0
_________________________________________________________________
conv2d_389 (Conv2D)          (None, 6, 6, 512)         2359808
_________________________________________________________________
```

```
max_pooling2d_148 (MaxPoolin (None, 3, 3, 512)         0
_________________________________________________________________
zero_padding2d_286 (ZeroPadd (None, 5, 5, 512)         0
_________________________________________________________________
conv2d_390 (Conv2D)          (None, 3, 3, 512)         2359808
_________________________________________________________________
zero_padding2d_287 (ZeroPadd (None, 5, 5, 512)         0
_________________________________________________________________
conv2d_391 (Conv2D)          (None, 3, 3, 512)         2359808
_________________________________________________________________
zero_padding2d_288 (ZeroPadd (None, 5, 5, 512)         0
_________________________________________________________________
conv2d_392 (Conv2D)          (None, 3, 3, 512)         2359808
_________________________________________________________________
max_pooling2d_149 (MaxPoolin (None, 1, 1, 512)         0
_________________________________________________________________
flatten_30 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_88 (Dense)             (None, 4096)              2101248
_________________________________________________________________
dropout_43 (Dropout)         (None, 4096)              0
_________________________________________________________________
dense_89 (Dense)             (None, 4096)              16781312
_________________________________________________________________
dropout_44 (Dropout)         (None, 4096)              0
_________________________________________________________________
dense_90 (Dense)             (None, 5)                 20485
=================================================================
Total params: 33,616,581
Trainable params: 33,616,581
Non-trainable params: 0
_________________________________________________________________
```

In [218]:

```
from keras.optimizers import Adam
from keras import optimizers
# adam = Adam()
model.compile(optimizer=Adam(), loss=keras.losses.categorical_crossentropy, metrics=['accurac
y'])
```

Adam() Fails due to the large number of parameters in the VGG network. use sgd.

RESULT:

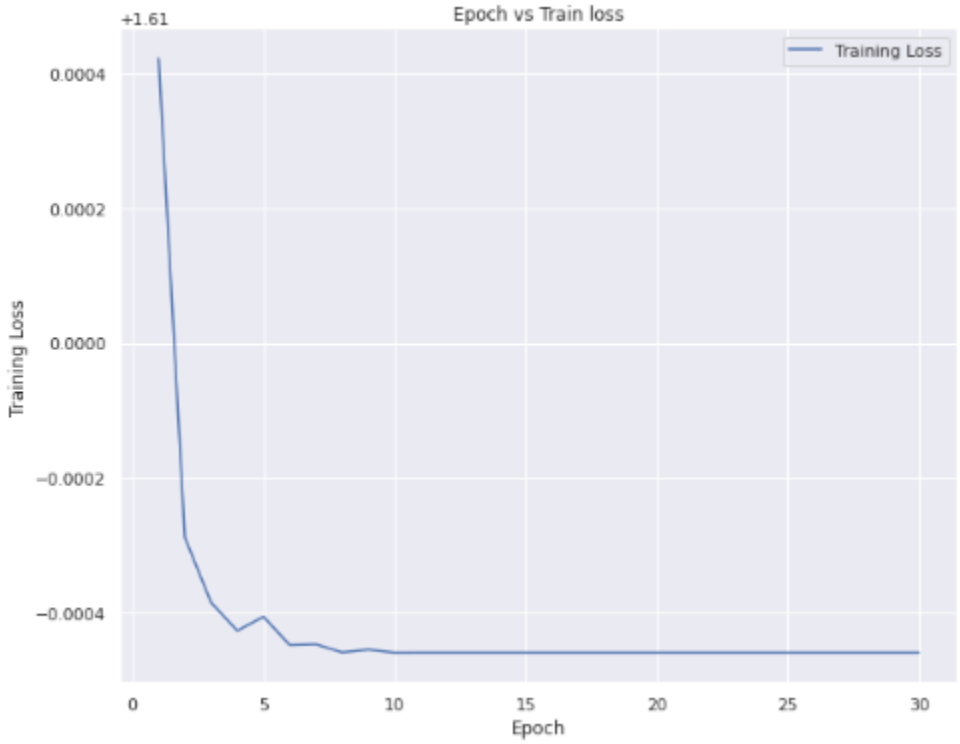
Epoch 50/50 |val_loss: 1.6096 - val_accuracy: 0.1970 | Time taken 1238.6372289899991

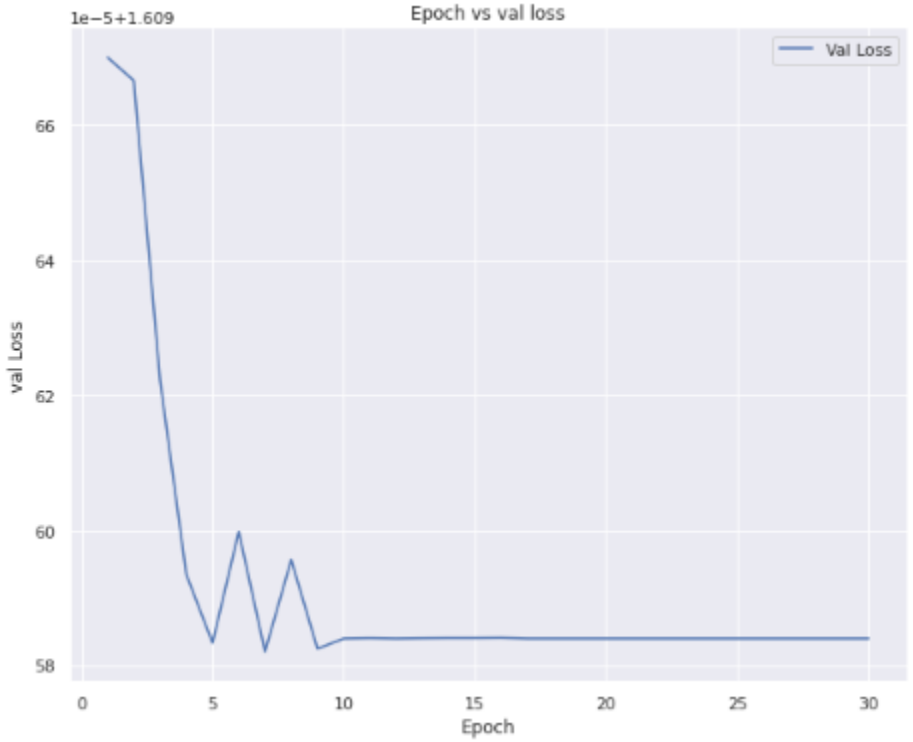
In [221]:
```
print('Time taken',stop-start)
```

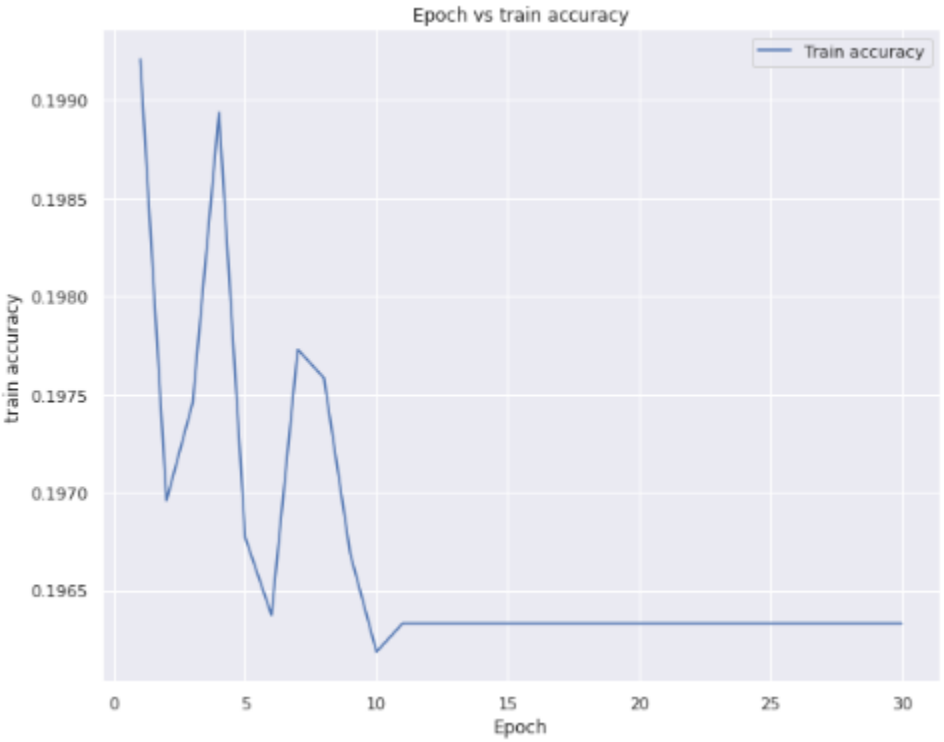
```
Time taken 1238.6372289899991
```

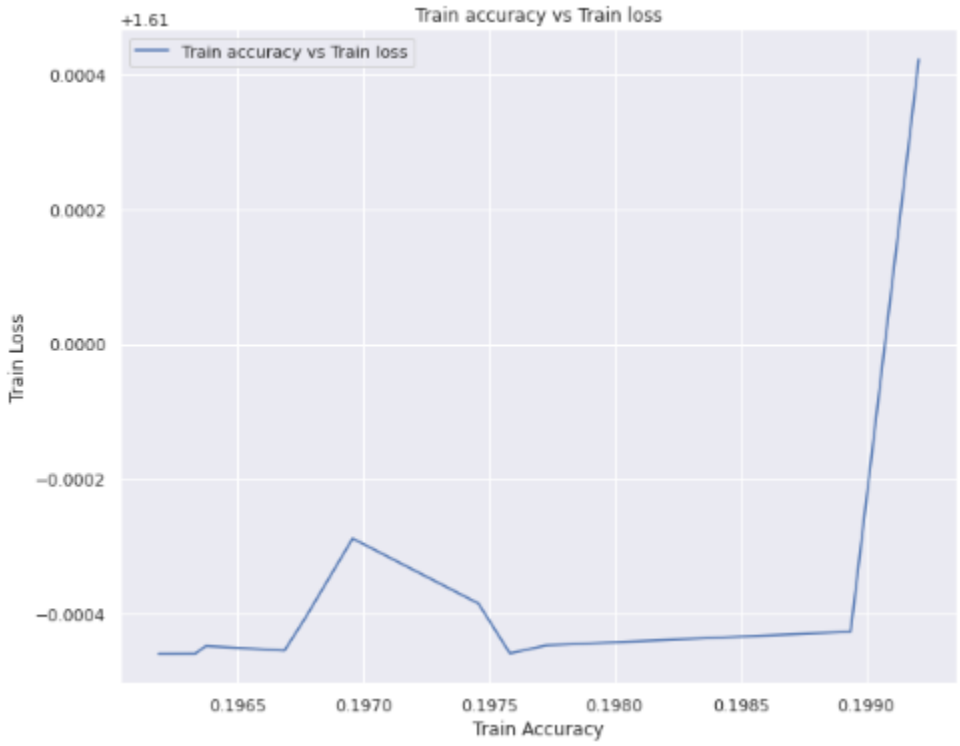
PLOTS

In [222]:
```
c.history
```




Epoch vs val loss
1e−5+1.609
Val Loss
66
64
62
60
58
val Loss
0
5
10
15
20
25
30
Epoch

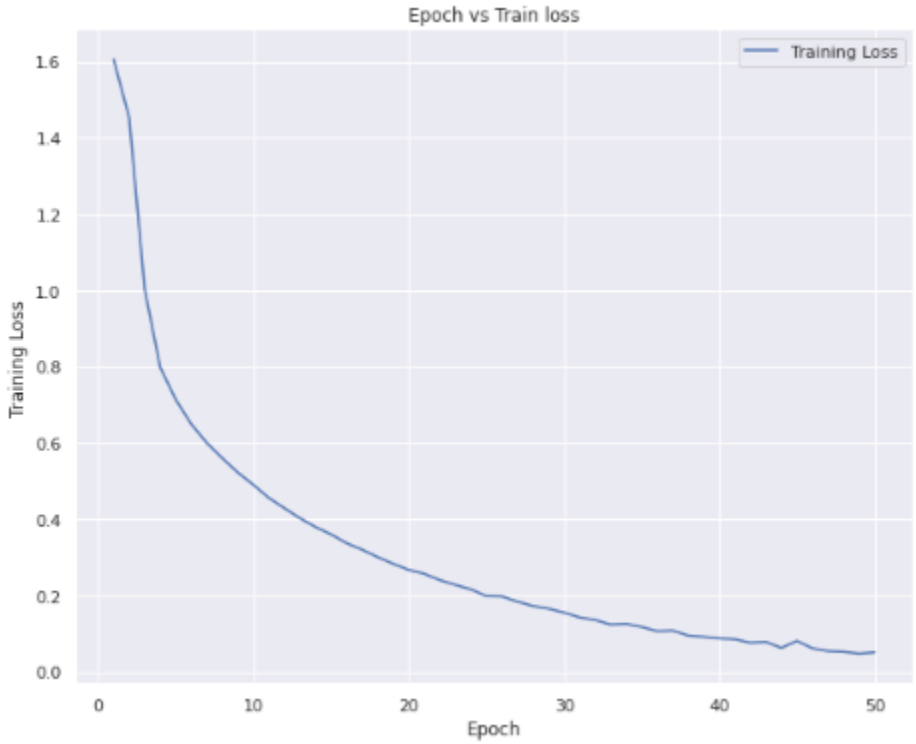

Epoch vs train accuracy
Train accuracy
0.1990
0.1985
0.1980
0.1975
0.1970
0.1965
train accuracy
0
5
10
15
20
25
30
Epoch

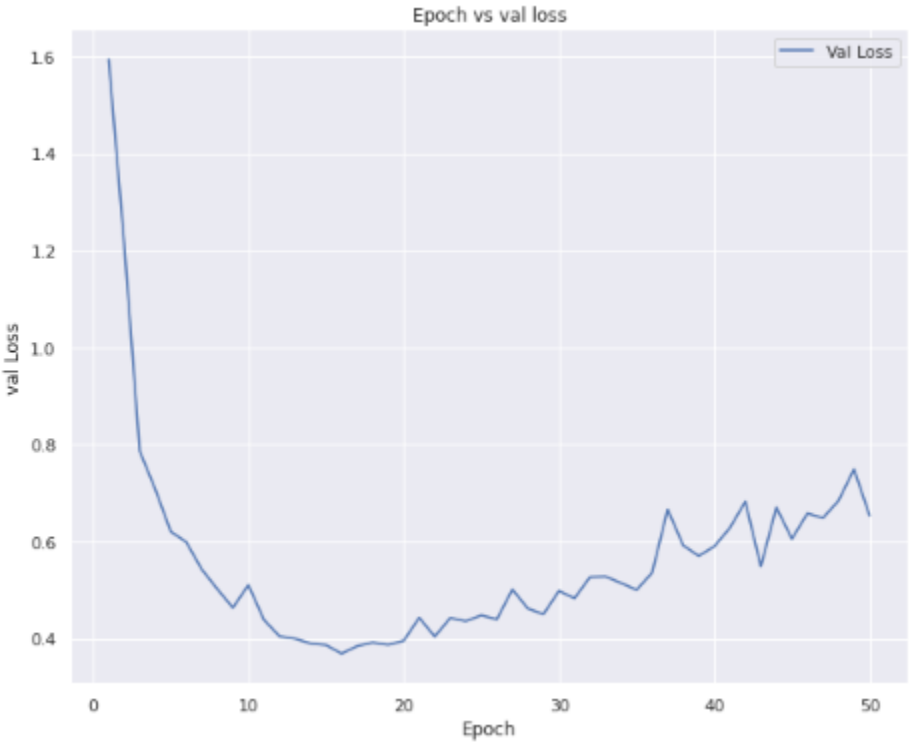


RESULTS AND ANALYSIS

1. From the above results we can see that SGD() works much better than Adam() when the number of layers are increased which was also true in the previous architecture of our CNN.
2. At that time there was less difference in the accuracy but as the layers and parameters are increased, Adam can be seen failing for our dataset.
3. Accuracy achieved by the SGD() in this tweaked model is 86.03% with time of 1751 to converge which is comparatively not better than our CNN. Our CNN Model 7 took around 900 seconds to converge.

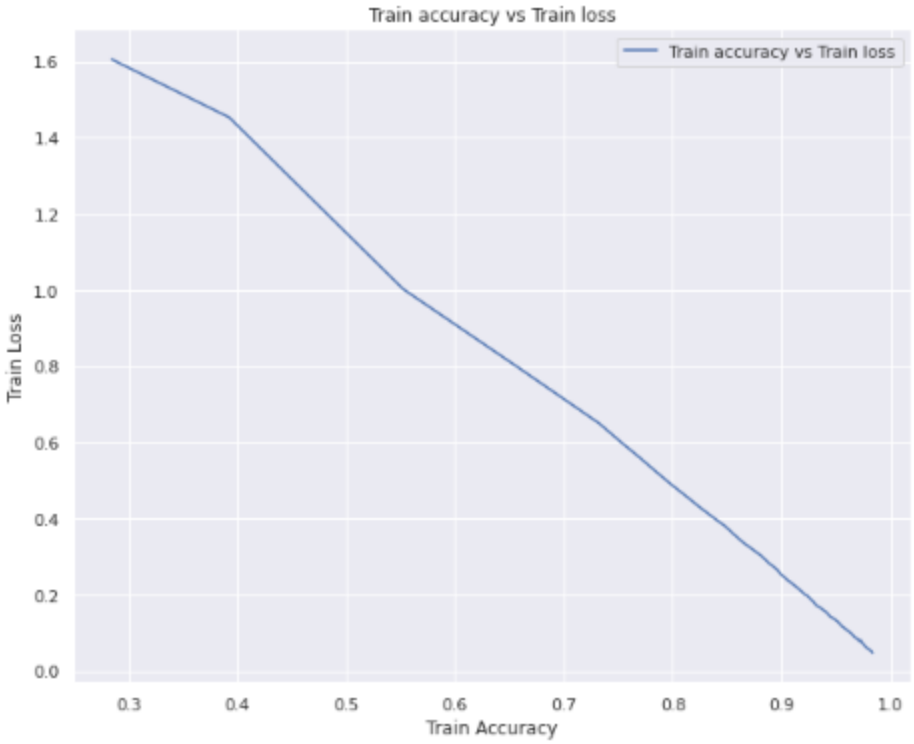
GRAPH ANALYSIS OF BEST MODEL (1) SGD()

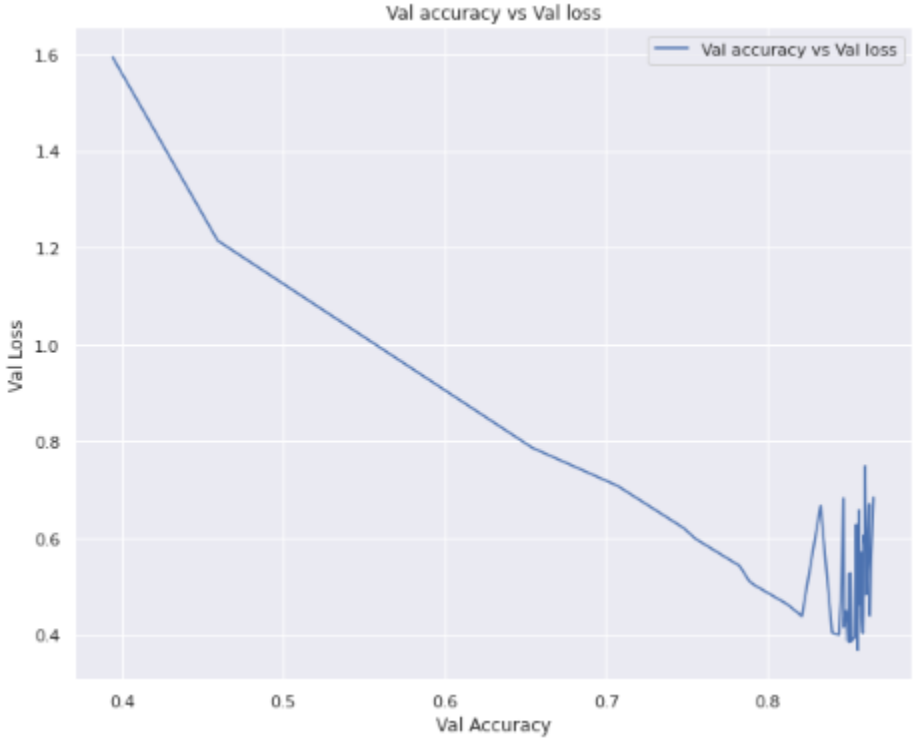




Analysis:

- **Training Epoch vs Loss:**
  - The loss decreases from 1.6 to 0.05 approximately from Epoch 0 to Epoch 50 which is a big decrease of 96% in the whole cycle.
  - 68.7% of the decrease in the loss can be seen in the first 10 epochs from 1.6 to 0.50.
  - The training loss stabilizes at 0.10 between epoch 40 and epoch 50.
- **Testing Epoch vs Loss:**
  - Testing loss decreased from 1.6 to 0.65 within the complete range of Epochs which is a decrease of 59%.
  - The Testing loss decreased from 1.6 to 0.5 during the first 10 epochs which is a decrease of 68.7%.
  - The loss again increased from 0.4 (lowest loss) to 0.65 from the epoch 20 to epoch 50.
  - It can be seen that till 20 epochs, loss was decreasing and after that the model contributed in degrading the validation loss.


Train accuracy vs Train loss
Train accuracy vs Train loss
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Train Loss
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
1.6
1.4
1.2
1.0
0.8
0.6
0.4
Val Loss
0.4
0.5
0.6
0.7
0.8
Val Accuracy

Analysis:

- **In Train loss vs Train accuracy,**
  - The loss has decreased from 1.6 to 0.05 with an increase in the accuracy from 24% to 98.28% within the complete range of epochs.
  - At 10^{th} epoch, the loss is 0.50 and the accuracy is 80%.

- **In Test loss vs Test accuracy,**
  - The loss has decreased from 1.6 to 0.65 with an increase in the accuracy from 30% to 86.03% within the complete range of epochs.
  - At epoch=10, loss is 0.50 which is same as for the training set and the accuracy is around 79%.
  - From epoch 11 to epoch 50, the accuracy has increased from 80% to 86% and the loss has also increased continuously from 0.40 to 0.70.
  - We can see that our model is fighting with the overfitting because both the loss and accuracy increased within the range of 11- 50(Epochs).

ALEXNET

ARCHITECTURE

Layer Type	Size	Number of Kernels	Number of Neurons
Image input	224×224×3		150,528
Convolution	11×11×3	96	253,440
ReLU			
Channel normalization			
Pooling			
Convolution	5×5×48	256	186,624
ReLU			
Channel normalization			
Pooling			
Convolution	3×3×256	384	64,896
ReLU			
Convolution	3×3×192	384	64,896
ReLU			
Convolution	3×3×192	256	43,264
ReLU			
Pooling			
Fully connected			4096
ReLU			
Dropout			
Fully connected			4096
ReLU			
Dropout			
Fully connected			80
Softmax			
Classification			

FIG:3 BASIC ALEXNET STRUCTURE

- Alexnet model consists of 5 Convolution layers and 3 Dense layer or fully connected layers. These multiple convolution layers help to extract essential features of the image.

- The first Convolution layer consists of 96 kernels of size 11 X 11 X 1. Here 1 represents number of channel(s). Since the images in our dataset are grayscale hence 1.

- The first Convolutional Layer is followed by Max pooling layer which help us to extract sharpest or low-level features of an image like edges, points etc. Max pooling layer helps to down sample the width and height of tensors . The size of the pooling window used is 3 X 3.

- Then we have 3 convolutional layers that are connected directly .

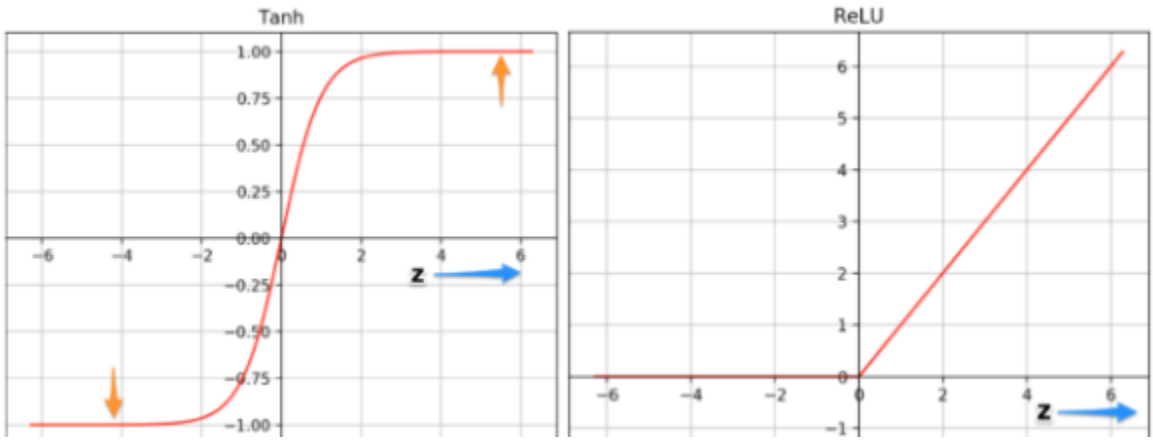
- These 3-convolution layers is again connected to a max- pooling layer.

- The max pooling layer feeds into 2 fully connected layers which in turns feeds into softmax classifier with 5 labels as according to our dataset.

- For non-linear perspective we have ReLU added after all convolution and fully connected layers. The importance of ReLU is that it sets all the negative values in the matrix of image to 0, other values are kept constant. This is really important because we are dealing with the non-linear functions.

- ReLUs help deep CNN to be trained much faster than the tanh or sigmoid which are also called as the saturation activation functions.

- We don’t have normalization in alexnet because researcher found it to be ineffective .



- We can easily see that the tanh function saturates at very high and very low levels of z , which makes the gradient descend slow(The slope is close). The ReLU function slope on other hand is still high for large values of z which allows optimization to converge quickly.

- To reduce overfitting, we have used dropout with a probability of 0.5. Using dropout a neuron is dropped which does not contributes in backward and forward propagation . The dropouts increase the steps taken to reach convergence but they reduce overfitting which improves accuracy for the model as general.

CODE BLOCKS AND DESIGN

DATA PREPARATTION

1.--Resizing the data into 48*48--

2.--Normalizing the data--

In [81]:

```
from keras.preprocessing.image import img_to_array, array_to_img
x_train = np.asarray([img_to_array(array_to_img(im, scale=False).resize((48,48))) for im in x_t
rain])
x_test = np.asarray([img_to_array(array_to_img(im, scale=False).resize((48,48))) for im in x_te
st])


input_size = (img_size, img_size,1)

# Normalize data.
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)

print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
print('y_train shape:', y_train.shape)
```

```
x_train shape: (48000, 48, 48, 1)
x_test shape: (12000, 48, 48, 1)
48000 train samples
12000 test samples
y_train shape: (48000,)
```

Note: we are using 48*48*1 image

-- Reshaping the final test data--

In [160]:

```
# for test data
#Storing Pixel array in form length width and channel in df_x_test
df_x_test = test_data.iloc[:,1:].values.reshape(len(test_data),28,28,1)
```

In [161]:

```
df_x_test.shape
```

Out[161]:

```
(10000, 28, 28, 1)
```

1. --Resizing the data into 48*48--

2. --Normalizing the test data--

In [163]:
```
df_x_test = np.asarray([img_to_array(array_to_img(im, scale=False).resize((48,48))) for im in df_x_test])



df_x_test=df_x_test.astype('float32')

#rescaling it between 0 to 1
df_x_test /=255
```

MODEL-1

Layer and dropouts	Count
Convolution	5
Maxpooling	3
Dropout	2 (0.5)
Dense	3(4096,4096,5)

Optimizer	SGD()
Epoch	50
Batch Size	64

Again, we are taking our optimizer as SGD(), we will try this architecture on Adadelta and Adam as well.

----------MODEL 1---------- SGD() | BS: 64 | EPOCH: 50

In [87]:

```
model= Sequential()
model.add(Conv2D(96,(11,11), strides=1, activation='relu',input_shape=(48,48,1)))
model.add(MaxPooling2D(pool_size=3, strides=2))

model.add(Conv2D(256, (5,5), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=3, strides=2))

model.add(Conv2D(384, (3,3),padding='same', activation='relu'))
model.add(Conv2D(384, (3,3), padding='same', activation='relu'))
model.add(Conv2D(256,  (3,3),padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=3, strides=2))

model.add(Flatten())

model.add(Dense(4096, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(4096, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(5, activation = "softmax"))
model.compile(optimizer = keras.optimizers.SGD() , loss = "categorical_crossentropy", metrics=
["accuracy"])
model.summary()
```

SUMMARY

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 38, 38, 96)        11712
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 18, 18, 96)        0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 18, 18, 256)       614656
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 8, 8, 384)         885120
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 8, 8, 384)         1327488
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 8, 8, 256)         884992
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 3, 3, 256)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_10 (Dense)             (None, 4096)              9441280
_________________________________________________________________
dropout_7 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_11 (Dense)             (None, 4096)              16781312
_________________________________________________________________
dropout_8 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_12 (Dense)             (None, 5)                 20485
=================================================================
Total params: 29,967,045
Trainable params: 29,967,045
Non-trainable params: 0
_________________________________________________________________
```

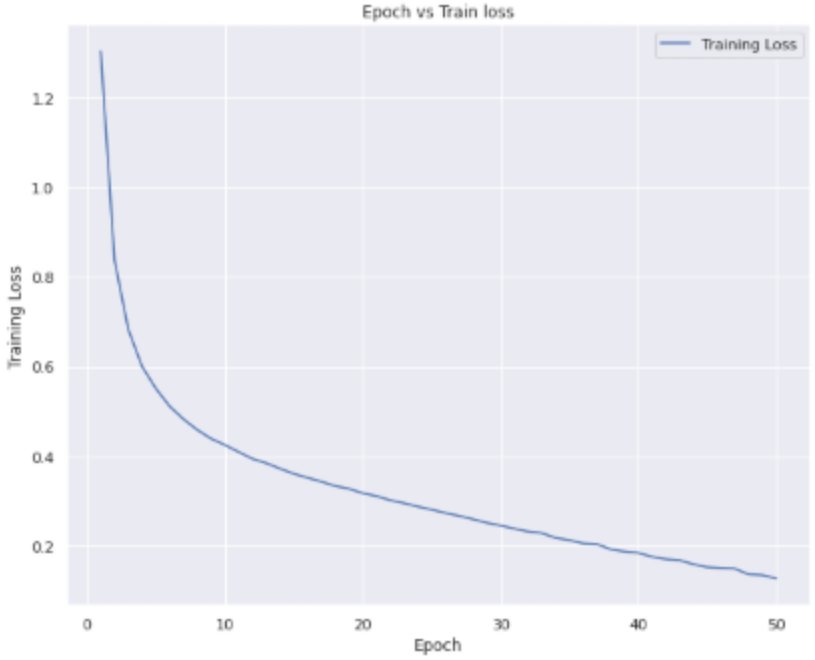
TIME AND ACCURACY

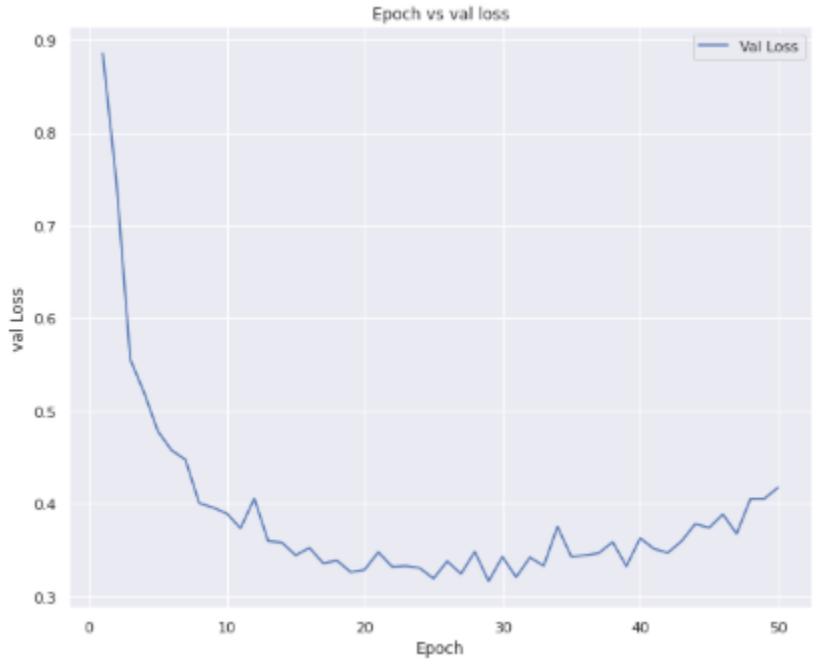
RESULT:

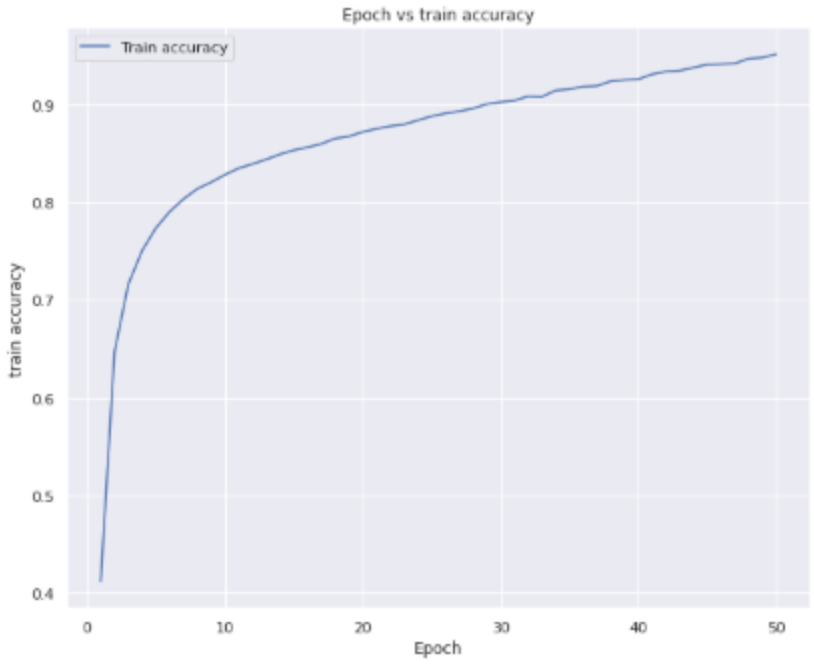
Epoch 50/50 |val_loss: 0.4172 - val_accuracy: 0.8742 | Time taken 737.9889082519994

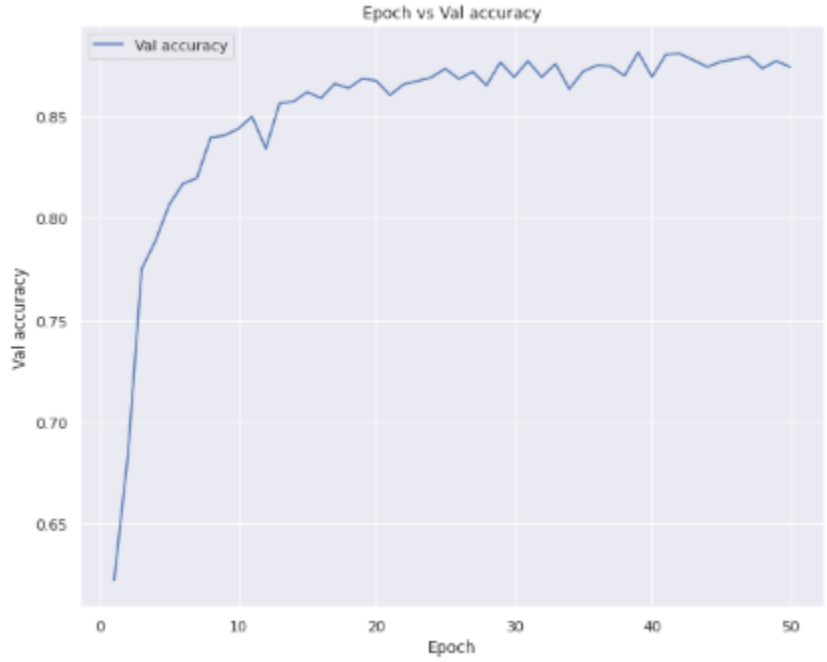
```
print('Time taken',stop-start)
```

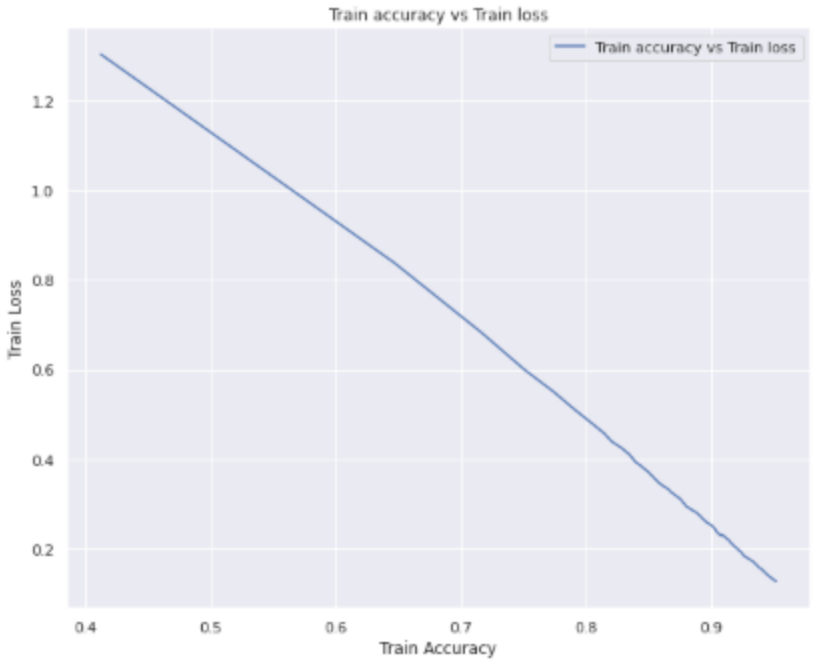
```
Time taken 737.9889082519994
```

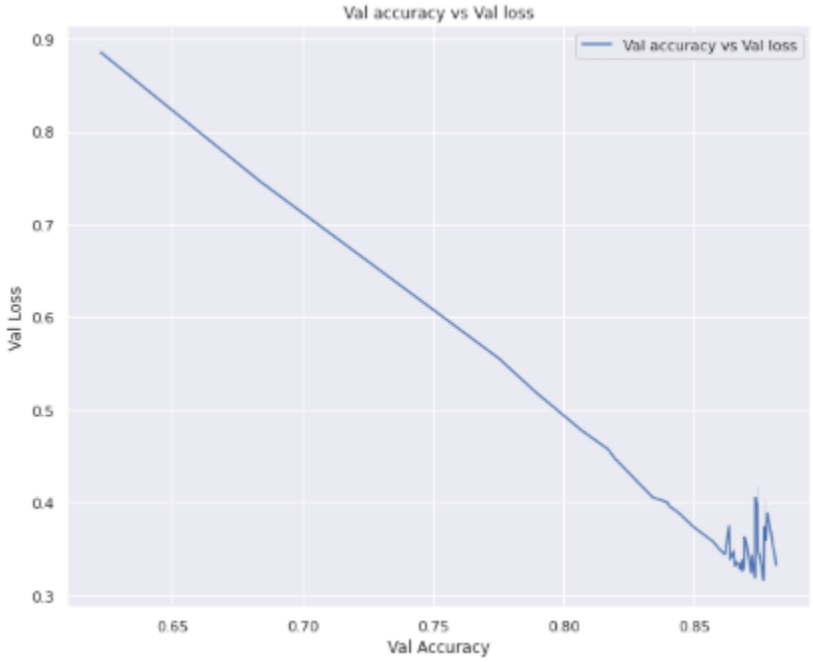

Epoch vs Train loss
Training Loss
Training Loss
1.2
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
50
Epoch


Epoch vs val loss
Val Loss
val Loss
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
10
20
30
40
50
Epoch


Epoch vs train accuracy
Train accuracy
train accuracy
0.4
0.5
0.6
0.7
0.8
0.9
0
10
20
30
40
50
Epoch


Epoch vs Val accuracy
Val accuracy
Val accuracy
0.65
0.70
0.75
0.80
0.85
0
10
20
30
40
50
Epoch


Train accuracy vs Train loss
Train accuracy vs Train loss
Train Loss
1.2
1.0
0.8
0.6
0.4
0.2
0.4
0.5
0.6
0.7
0.8
0.9
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
Val Loss
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.65
0.70
0.75
0.80
0.85
Val Accuracy

MODEL 2

----------MODEL 2---------- Adam() | BS: 64 | EPOCH: 50

In [174]:

```
model= Sequential()
model.add(Conv2D(96,(11,11), strides=1, activation='relu',input_shape=(48,48,1)))
model.add(MaxPooling2D(pool_size=3, strides=2))

model.add(Conv2D(256, (5,5), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=3, strides=2))

model.add(Conv2D(384, (3,3),padding='same', activation='relu'))
model.add(Conv2D(384, (3,3), padding='same', activation='relu'))
model.add(Conv2D(256,  (3,3),padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=3, strides=2))

model.add(Flatten())

model.add(Dense(4096, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(4096, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(5, activation = "softmax"))
model.compile(optimizer = keras.optimizers.Adam() , loss = "categorical_crossentropy", metrics=
["accuracy"])
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_26 (Conv2D)           (None, 38, 38, 96)        11712
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 18, 18, 96)        0
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 18, 18, 256)       614656
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 8, 8, 256)         0
```

```
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 8, 8, 384)         885120
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 8, 8, 384)         1327488
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 8, 8, 256)         884992
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 3, 3, 256)         0
_________________________________________________________________
flatten_6 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_16 (Dense)             (None, 4096)              9441280
_________________________________________________________________
dropout_11 (Dropout)         (None, 4096)              0
_________________________________________________________________
dense_17 (Dense)             (None, 4096)              16781312
_________________________________________________________________
dropout_12 (Dropout)         (None, 4096)              0
_________________________________________________________________
dense_18 (Dense)             (None, 5)                 20485
=================================================================
Total params: 29,967,045
Trainable params: 29,967,045
Non-trainable params: 0
_________________________________________________________________
```

RESULT:

Epoch 50/50 |val_loss: 0.4484 - val_accuracy: 0.8602 | Time taken 964.7407229679993

In [176]:

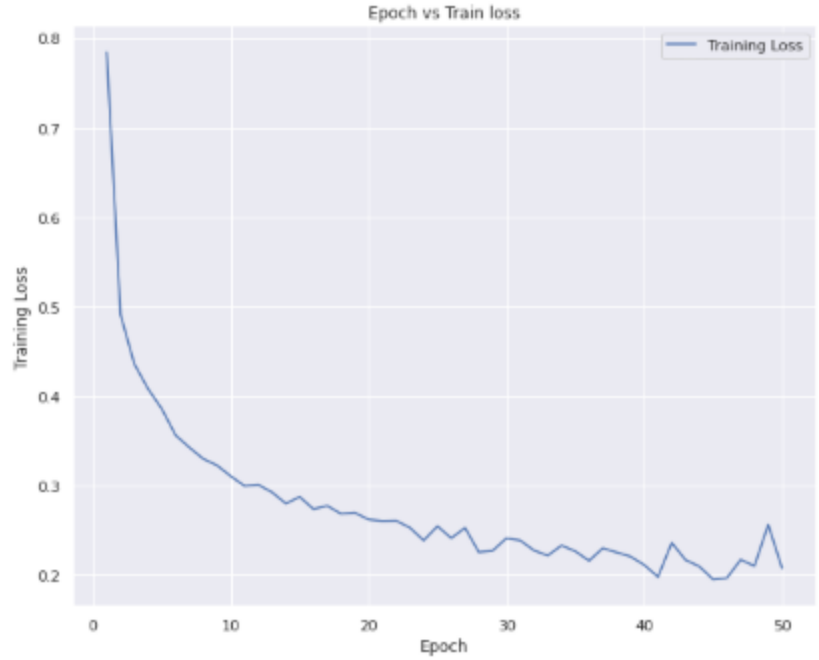
```
print('Time taken',stop-start)
```

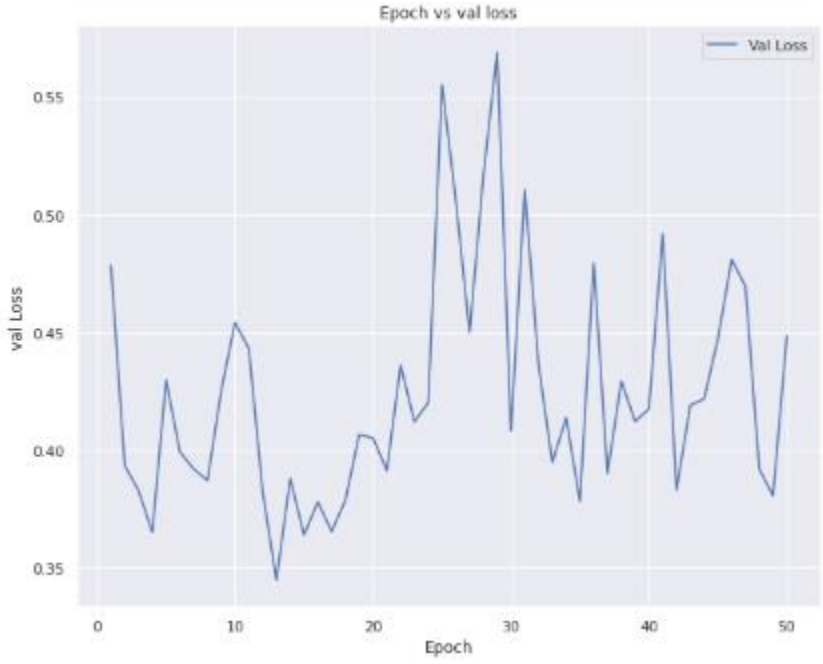
```
Time taken 964.7407229679993
```

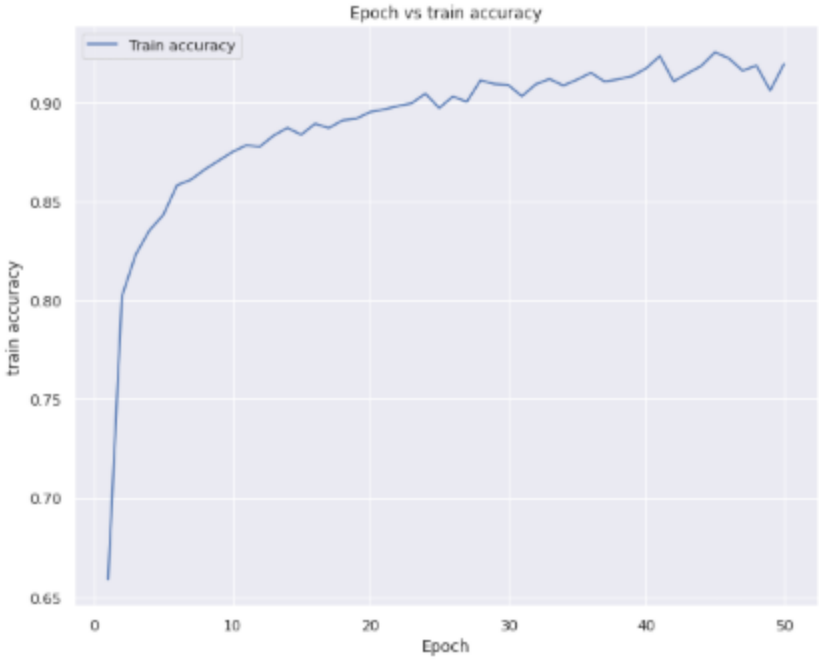
PLOTS

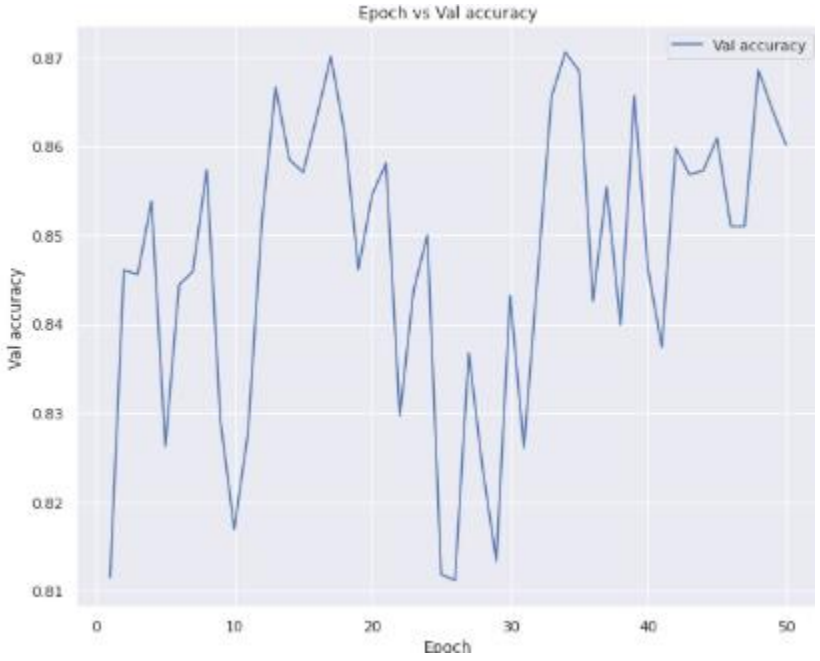
In [177]:

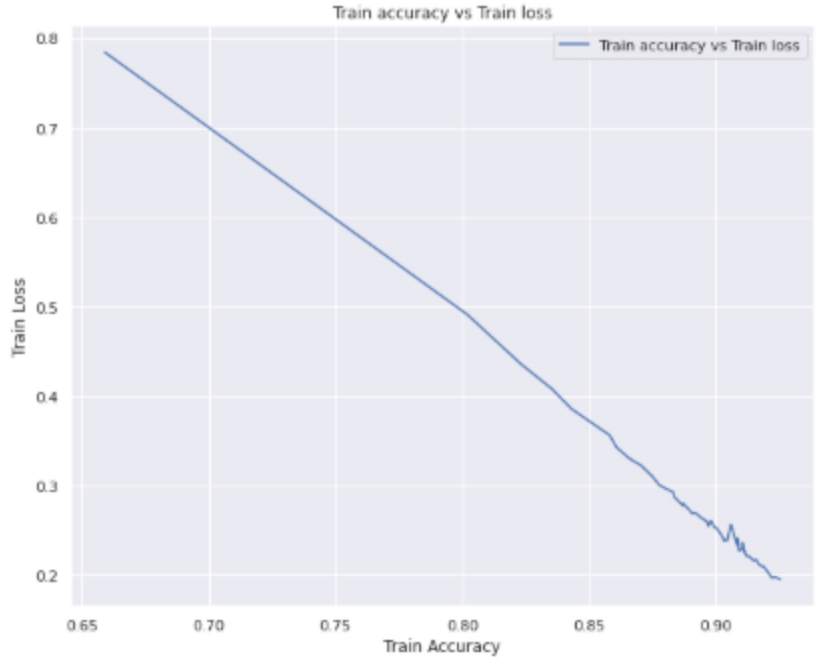
```
b.history
```

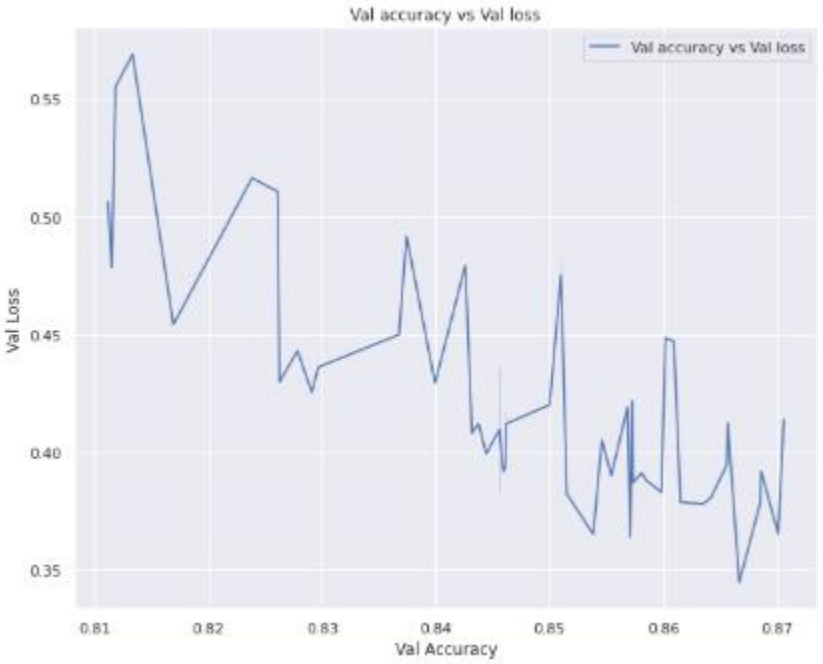

Epoch vs Train loss
Training Loss
Training Loss
Epoch
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
50


Epoch vs val loss
Val Loss
val Loss
Epoch
0.55
0.50
0.45
0.40
0.35
0
10
20
30
40
50


Epoch vs train accuracy
Train accuracy
train accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0
10
20
30
40
50
Epoch


Epoch vs Val accuracy
Val accuracy
Val accuracy
0.87
0.86
0.85
0.84
0.83
0.82
0.81
0
10
20
30
40
50
Epoch


Train accuracy vs Train loss
Train accuracy vs Train loss
Train Loss
Train Accuracy
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.65
0.70
0.75
0.80
0.85
0.90


Val accuracy vs Val loss
Val accuracy vs Val loss
Val Loss
Val Accuracy
0.55
0.50
0.45
0.40
0.35
0.81
0.82
0.83
0.84
0.85
0.86
0.87

MODEL 3

Here we are using Adadelta as our optimizer.

----------MODEL 3---------- Adadelta() | BS: 64 | EPOCH: 50

In [214]:

```
model= Sequential()
model.add(Conv2D(96,(11,11), strides=1, activation='relu',input_shape=(48,48,1)))
model.add(MaxPooling2D(pool_size=3, strides=2))

model.add(Conv2D(256, (5,5), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=3, strides=2))

model.add(Conv2D(384, (3,3),padding='same', activation='relu'))
model.add(Conv2D(384, (3,3), padding='same', activation='relu'))
model.add(Conv2D(256,  (3,3),padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=3, strides=2))

model.add(Flatten())

model.add(Dense(4096, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(4096, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(5, activation = "softmax"))
model.compile(optimizer = keras.optimizers.Adadelta() , loss = "categorical_crossentropy", metr
ics=["accuracy"])
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_51 (Conv2D)           (None, 38, 38, 96)        11712
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 18, 18, 96)        0
_________________________________________________________________
conv2d_52 (Conv2D)           (None, 18, 18, 256)       614656
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 8, 8, 256)         0
```

```
_________________________________________________________________
conv2d_53 (Conv2D)           (None, 8, 8, 384)         885120
_________________________________________________________________
conv2d_54 (Conv2D)           (None, 8, 8, 384)         1327488
_________________________________________________________________
conv2d_55 (Conv2D)           (None, 8, 8, 256)         884992
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 3, 3, 256)         0
_________________________________________________________________
flatten_11 (Flatten)         (None, 2304)              0
_________________________________________________________________
dense_31 (Dense)             (None, 4096)              9441280
_________________________________________________________________
dropout_21 (Dropout)         (None, 4096)              0
_________________________________________________________________
dense_32 (Dense)             (None, 4096)              16781312
_________________________________________________________________
dropout_22 (Dropout)         (None, 4096)              0
_________________________________________________________________
dense_33 (Dense)             (None, 5)                 20485
=================================================================
Total params: 29,967,045
Trainable params: 29,967,045
Non-trainable params: 0
_________________________________________________________________
```

RESULT:

Epoch 50/50 |val_loss: 1.1783 - val_accuracy: 0.8524 | Time taken 1089.7482127199983

In [216]:

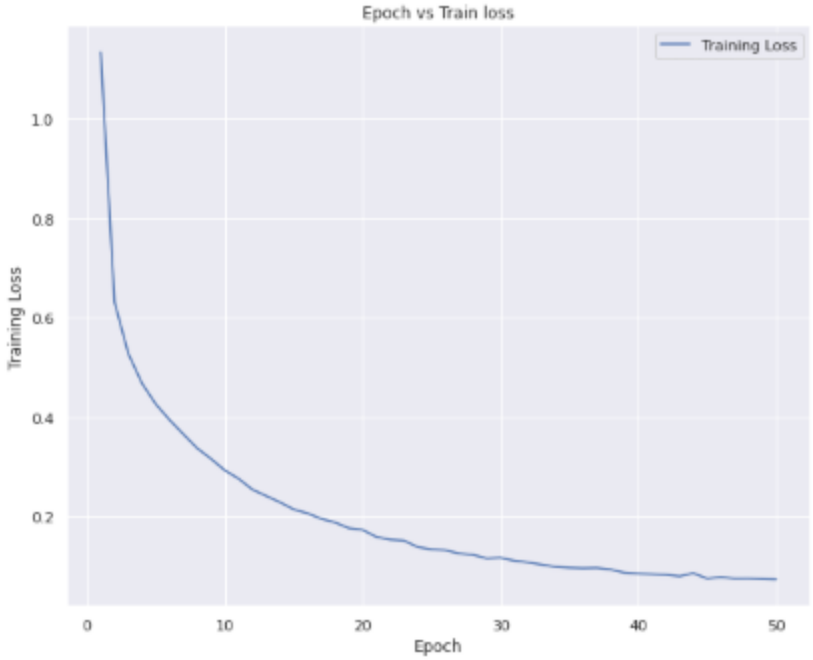
```
print('Time taken',stop-start)
```

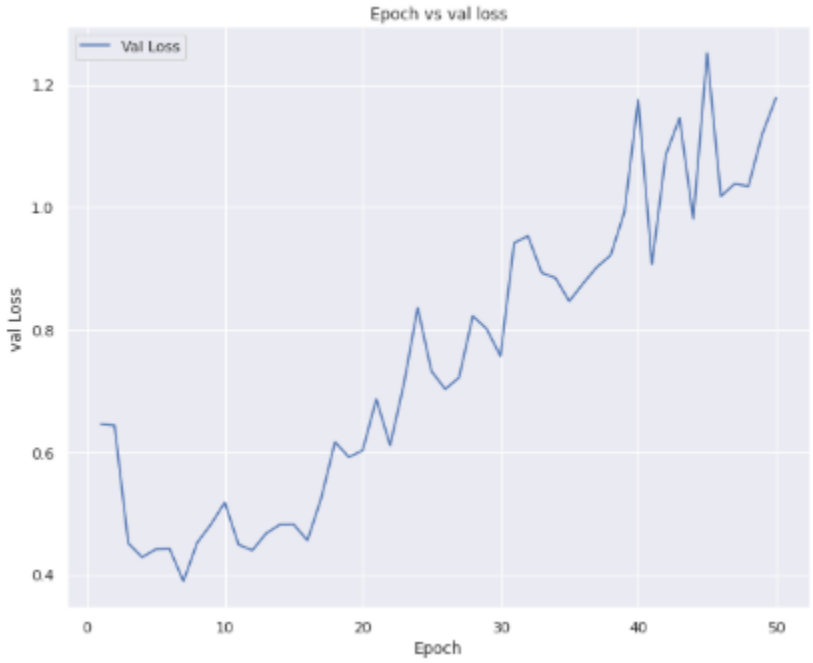
```
Time taken 1089.7482127199983
```

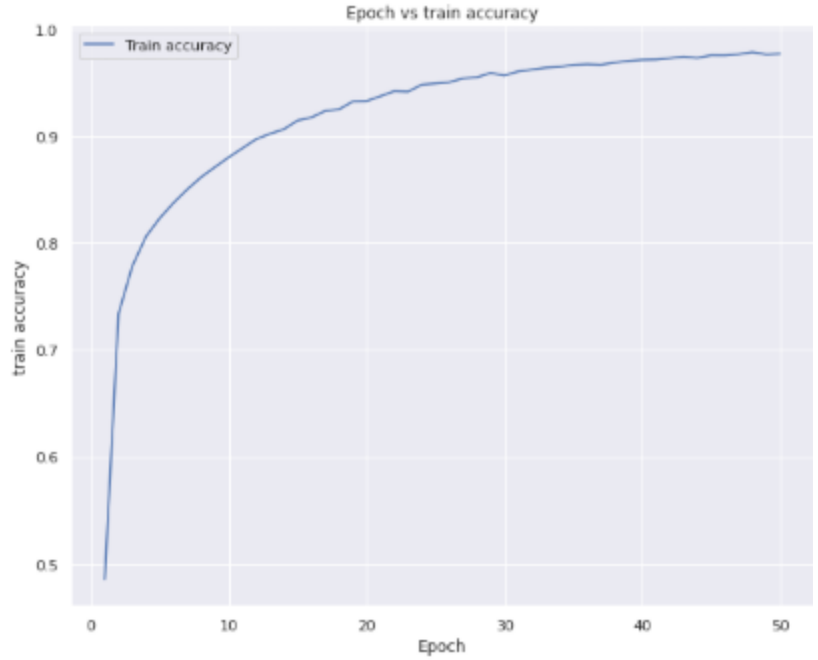
PLOTS

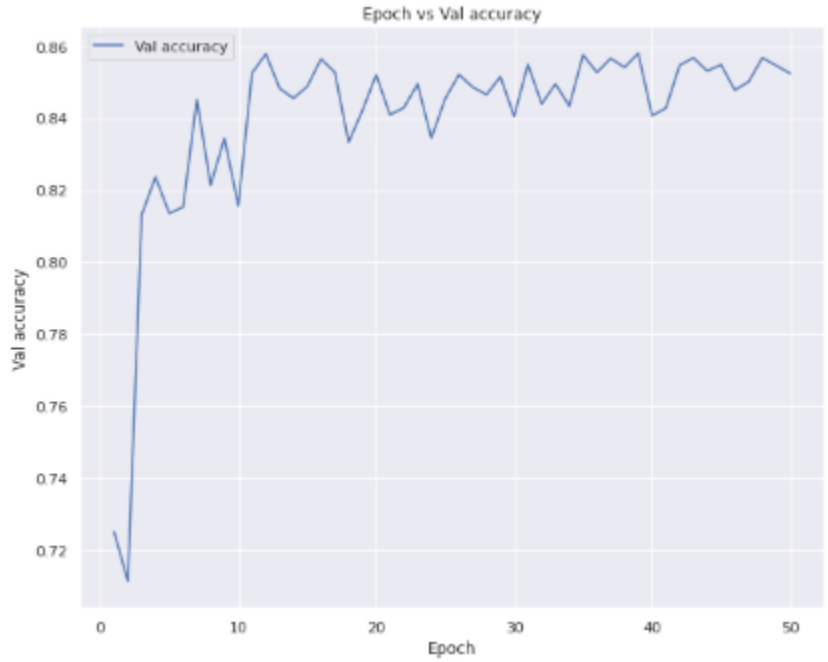
In [217]:

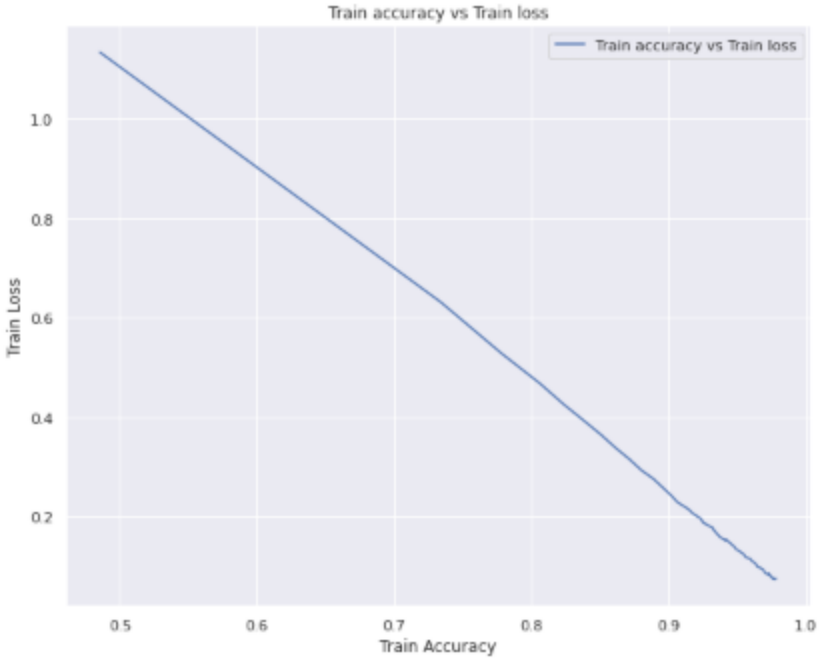
```
c.history
```

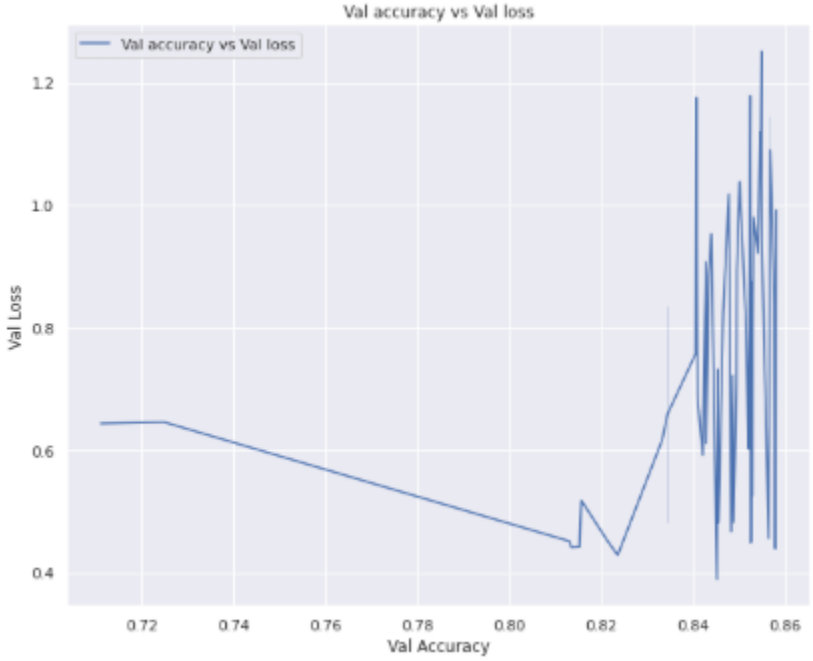

Epoch vs Train loss
Training Loss
Training Loss
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
50
Epoch


Epoch vs val loss
Val Loss
val Loss
1.2
1.0
0.8
0.6
0.4
0
10
20
30
40
50
Epoch


Epoch vs train accuracy
Train accuracy
train accuracy
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0
10
20
30
40
50


Epoch vs Val accuracy
Val accuracy
Val accuracy
Epoch
0.86
0.84
0.82
0.80
0.78
0.76
0.74
0.72
0
10
20
30
40
50


Train accuracy vs Train loss
Train accuracy vs Train loss
1.0
0.8
0.6
0.4
0.2
Train Loss
0.5
0.6
0.7
0.8
0.9
1.0
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
1.2
1.0
0.8
0.6
0.4
Val Loss
0.72
0.74
0.76
0.78
0.80
0.82
0.84
0.86
Val Accuracy

RESULTS AND ANALYSIS

Runtime analysis

MODELS WITH OPTIMIZERS	TIME
SGD()	737
Adam()	965
Adadelta()	1089

- As seen in the previous models, model with SGD() takes less time to converge which can also be seen here.

[Adadelta scales the learning rate on the basis of the historical gradient while taking into account only recent time window(not the whole history). Secondly, it uses component that serves an acceleration term, that accumulates historical updates which is similar to momentum.]

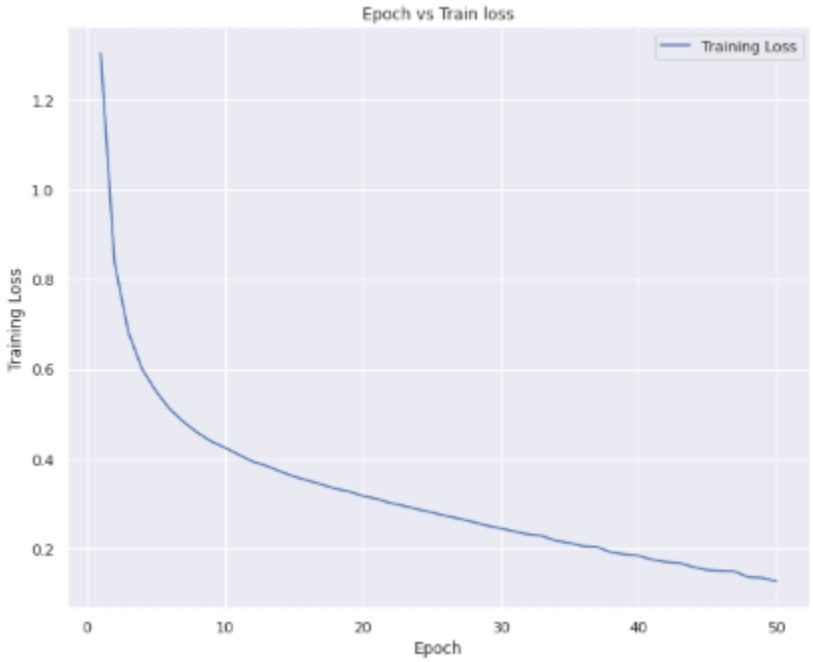
- We can conclude that the time taken by the SGD() is least for Alexnet as the SGD() takes the subset of samples and perform the computations.

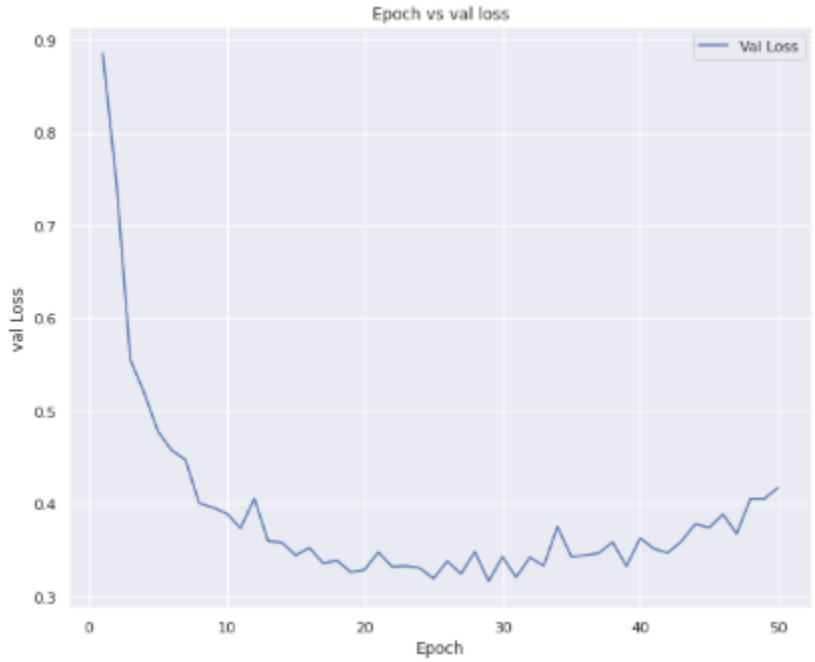
PERFORMANCE OF MODELS

	Val Accuracy	Time
MODEL 1	**87.42%**	**737.98**
MODEL 2	**86.02%**	**964.74**
MODEL 3	**85.24%**	**1089.74**

- Performance of Alexnet with SGD() as the optimizer is 87.42% and the runtime for the same is also less than the other two.
- Architecture with Adam as optimizer is not converging well for our dataset which can be seen.

GRAPHICAL ANALYSIS (MODEL 1)

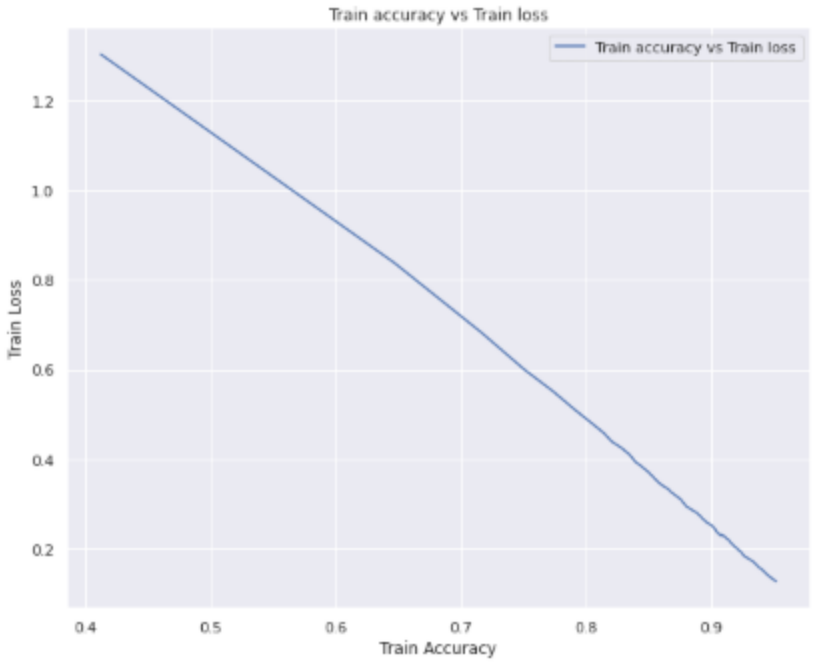


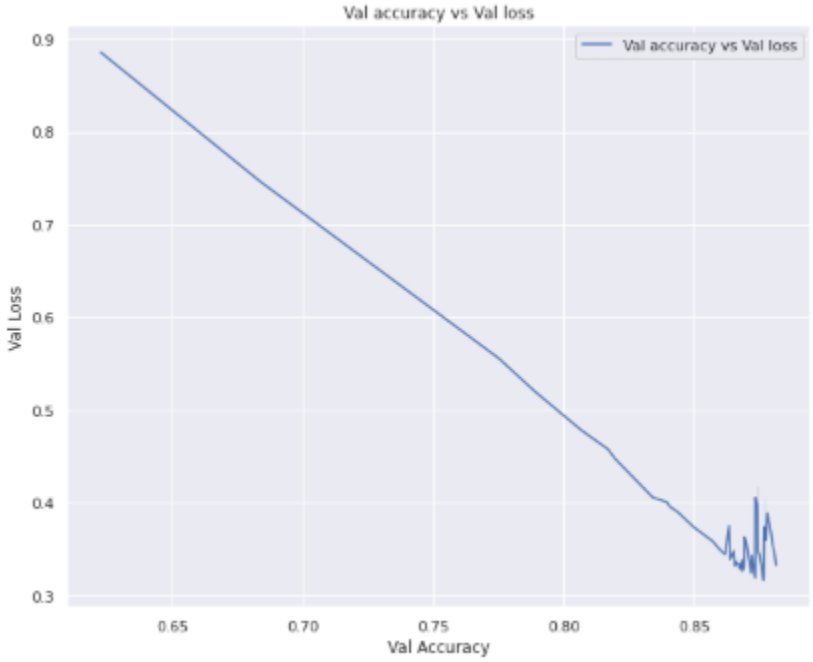


Analysis:

- **Training Epoch vs Loss:**
  - 92% of the decrease in the loss can be seen from 1.3 to 0.1 within the range of all the epochs.
  - 68% of loss is decreased from 1.3 to 0.42 with in the first 10 epochs.
- **Testing Epoch vs Loss:**
  - Testing loss decreased from 0.9 to 0.40 for the first 10 epochs.
  - From epoch 10-epoch 30, the loss remained stabilized within the range of (0.35,0.40) with an initial decrease.
  - The loss again increased from 0.35(lowest loss) to 0.42 from the epoch 30 to epoch 50.

It can be seen that loss stabilized between 20-30 epochs at 0.35 for the test set. If the accuracy is decreased after these levels where the loss is increasing, then there might be a chance of overfitting and our model is diverging. We will analyze this in next plots.


Train accuracy vs Train loss
Train accuracy vs Train loss
Train Loss
1.2
1.0
0.8
0.6
0.4
0.2
0.4
0.5
0.6
0.7
0.8
0.9
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
Val Loss
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.65
0.70
0.75
0.80
0.85
Val Accuracy

Analysis:

- **In Train loss vs Train accuracy,**
  - The loss has decreased from 1.3 to 0.1 with an increase in the accuracy from 40% to 97% within the complete range of epochs.

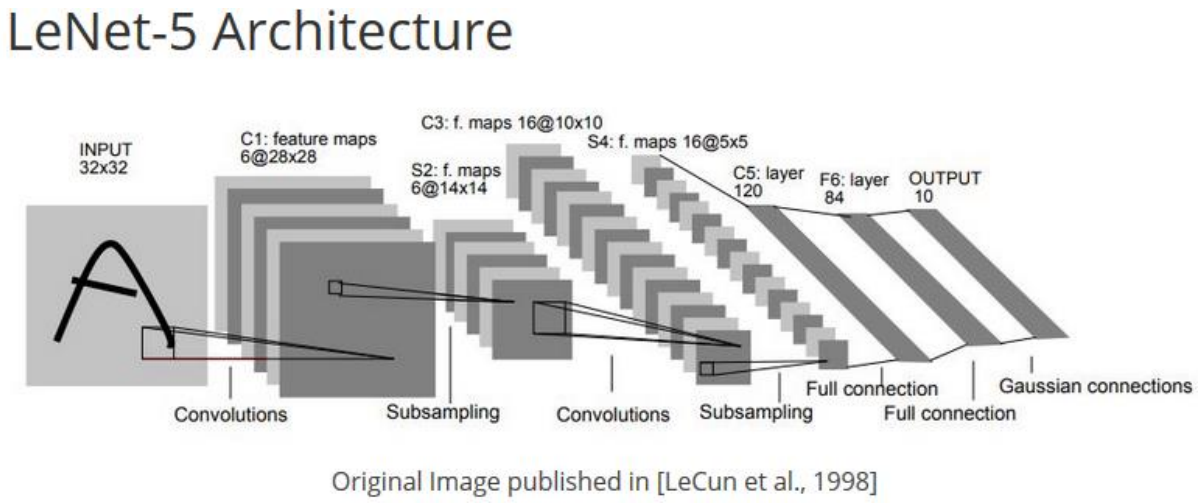
- **In Test loss vs Test accuracy,**
  - The loss has decreased from 0.9 to 0.35 with an increase in the accuracy from 63% to 87.42% within the complete range of epochs.
  - For large number of epochs, loss between the range of 0.3-0.4 can be seen at accuracy levels of 85% and more.

From both the graphs, we concluded that in the last 10 epochs, our model faces the problem of slight overfitting. Loss as well as accuracy increased which shows that our model is fighting well with the problem of overfitting with the regularization techniques.

LENET-5

ARCHITECTURE

- The LeNet-5 is a type of neural network which consists of 2 sets of convolution layer each followed the average pooling layer which helps to extract essential features of the image. The image dataset supplied to us consist of only grey-scale images and hence channel is one. These are then followed by a convolution layer which is essentially used for flattening .the dataset which is then fed to two fully connected layer and finally SoftMax classifier which helps in improving accuracy and correct classification of images into respective labels.



- The first Convolution layer has 6 filters of size 5x5 which act as feature as feature maps. The stride taken here for same is 1.The padding here is taken as 0. The image dimensions changes from 32x32x1 to 28x28x6.

- The output is the fed into Average pooling Layer with kernel size=(5,5) and stride here taken is 2. The filters chosen for this purpose are 6. Which changes the image dimensions into 14x14x6 as padding taken as 0. The average pooling layer helps to reduce variance and complexity as we have to deal with lower dimensional space.

- The third layer which is added is actually the Convolution layer with filter size (5x5) and there are 16 filters in total. The point here to note is that only 10 out of 16 feature maps or filters are connected to 6 filters of the previous layer. There is abrupt change in the number of filters to make sure that we don't have symmetric structure of the classifier. The stride taken in this step is 1.
- The output of this layer is fed into an average pooling layer with 16 filters of size (2x2) and stride taken here is 2. The padding is taken as 0.The dimension of output now becomes (5x5x16).
- The fifth layer in consideration form the fully connected layer with 120 filters of size (1x1).Each of these filters are connected to the 400 possible connections to the previous node.
- The sixth layer in consideration is fully connected layer with 84 nodes and the parameters considered for training is Weight, Bias.(the parameters trained=((120x84) +84)
- Finally comes the output layer which is also a dense layer with 5 output for 5 classes in our dataset. Here SoftMax activation function is used which changes numbers into probabilities whose total is 1.One important factor to note is SoftMax used here is not a black box and is specially used for multi class classifier problem.

MODEL 1

MODEL 1 → adam = Adam(lr=0.01) | epoch = 80 | batch_size= 64

In [18]:
```
import keras
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.losses import categorical_crossentropy
```

In [20]:
```
model = Sequential()
model.add(Conv2D(6, kernel_size=(5,5), padding='same', activation='tanh', input_shape=(28, 28,
1), strides=(1,1)))
model.add(AveragePooling2D(pool_size=(2,2), strides=(1,1), padding='valid'))
model.add(Conv2D(16, kernel_size=(5,5), padding='valid', activation='tanh'))
model.add(AveragePooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
model.add(Flatten())
model.add(Dense(120, activation='tanh'))
model.add(Dense(84, activation='tanh'))
model.add(Dense(5, activation='softmax'))
```

In [21]:

```
model.build()
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_3 (Average (None, 27, 27, 6)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 23, 23, 16)        2416
_________________________________________________________________
average_pooling2d_4 (Average (None, 11, 11, 16)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1936)              0
_________________________________________________________________
dense_4 (Dense)              (None, 120)               232440
_________________________________________________________________
dense_5 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_6 (Dense)              (None, 5)                 425
=================================================================
Total params: 245,601
Trainable params: 245,601
Non-trainable params: 0
_________________________________________________________________
```

In [23]:

```
adam = Adam()
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer=adam)
```

Time Taken 332.455398782

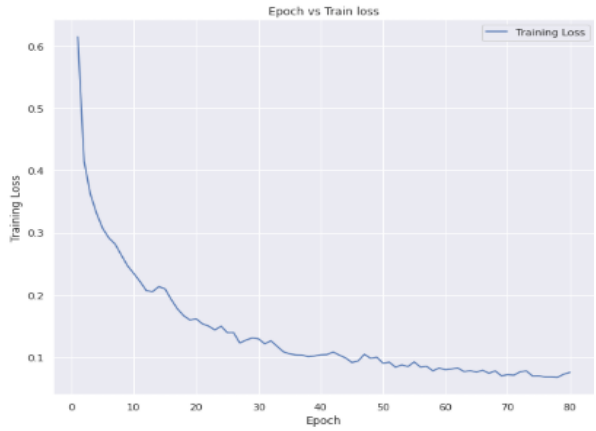
RESULT | MODEL 1 | VAL DATA

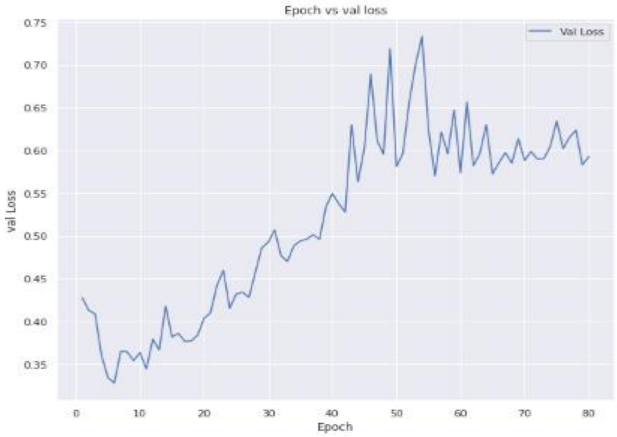
Epoch 80/80 | 48000/48000 | val_loss: 0.5924 - val_accuracy: 0.8618

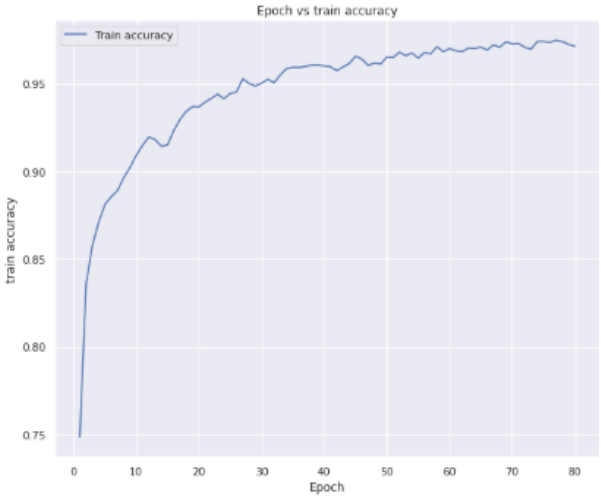
PLOTS

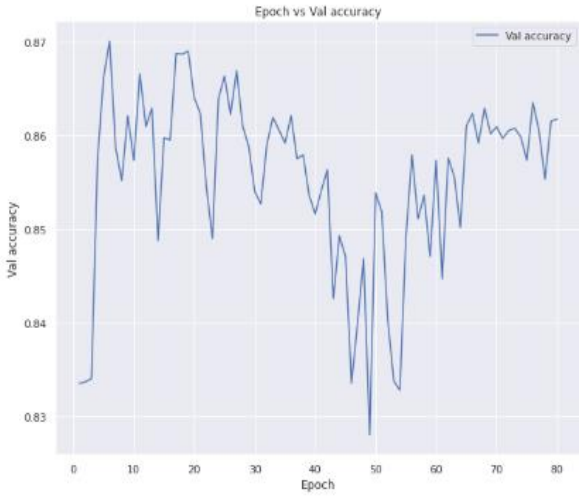
In [26]:

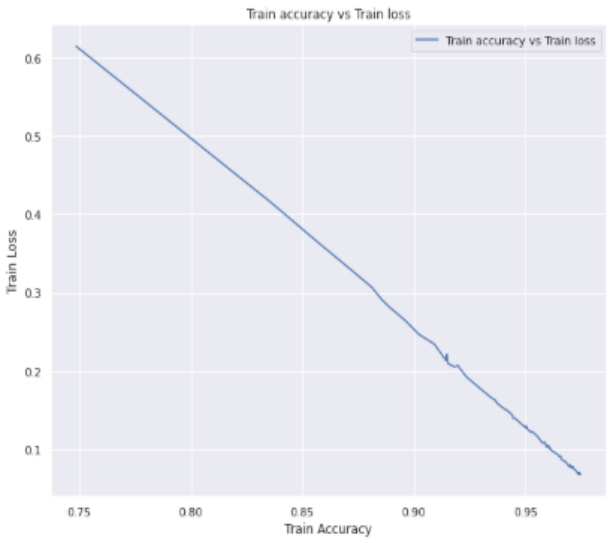
```
a.history
```

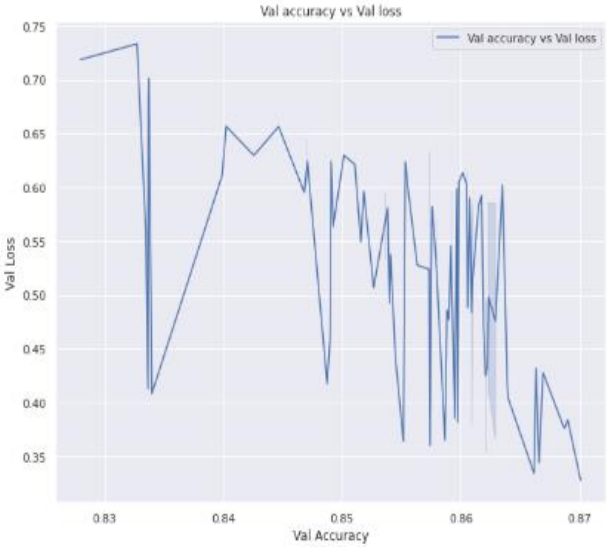

Epoch vs Train loss
Training Loss
Training Loss
Epoch


Epoch vs val loss
Val Loss
Val Loss
Epoch


Epoch vs train accuracy
Train accuracy
train accuracy
Epoch


Epoch vs Val accuracy
Val accuracy
Val accuracy
Epoch


Train accuracy vs Train loss
Train accuracy vs Train loss
Train Loss
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
Val Loss
Val Accuracy

MODEL 2

MODEL 1 → adam = Adam(lr=5e-4) i.e. 0.00050 | epoch = 80 | batch_size= 64

In [19]:

```
import keras
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.losses import categorical_crossentropy


model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), padding='same', activation='relu', input_shape=
(28, 28, 1)))
model.add(MaxPooling2D(strides=2))
model.add(Conv2D(filters=48, kernel_size=(5,5), padding='valid', activation='relu'))
model.add(MaxPooling2D(strides=2))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dense(5, activation='softmax'))
```

In [20]:

```
model.build()
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 48)        38448
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 48)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1200)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               307456
_________________________________________________________________
dense_2 (Dense)              (None, 84)                21588
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 425
=================================================================
Total params: 368,749
Trainable params: 368,749
Non-trainable params: 0
_________________________________________________________________
```

In [21]:

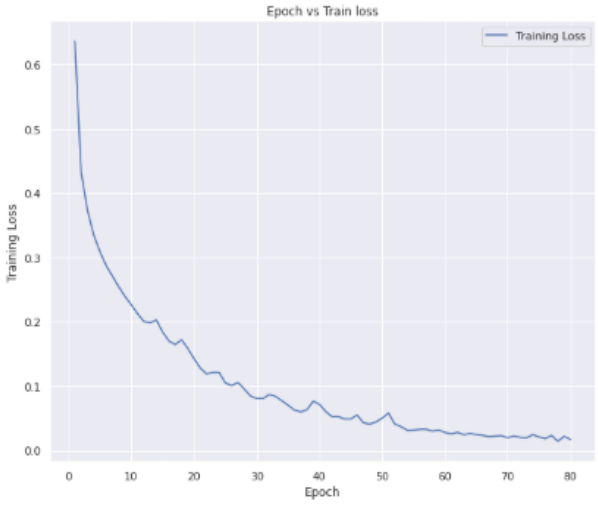
```
adam = Adam(lr=5e-4) # 0.00050
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer=adam)
```

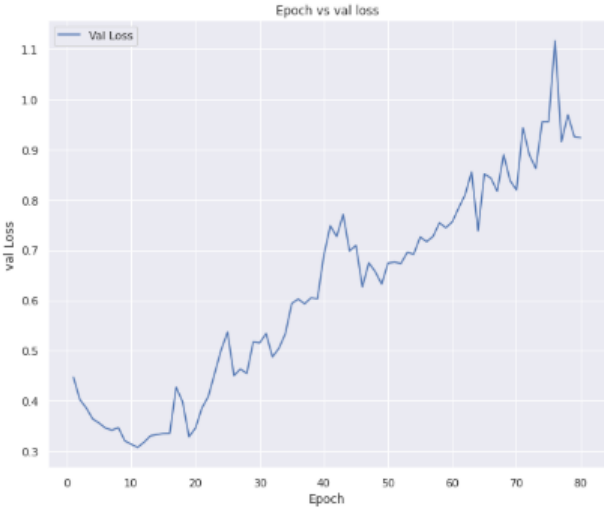
Time Taken 319.34

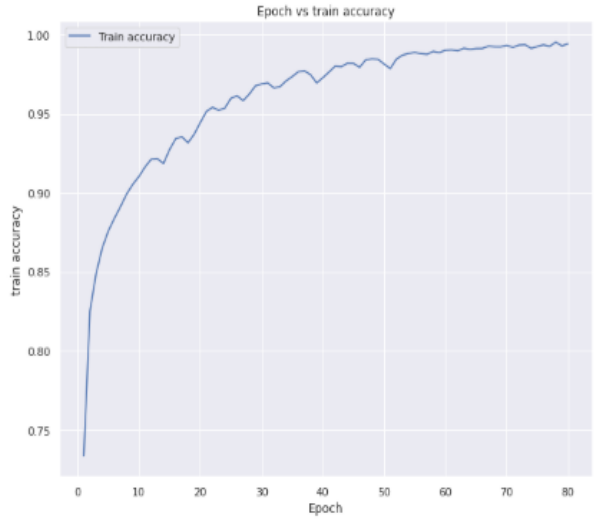
RESULT | MODEL 1 | VAL DATA

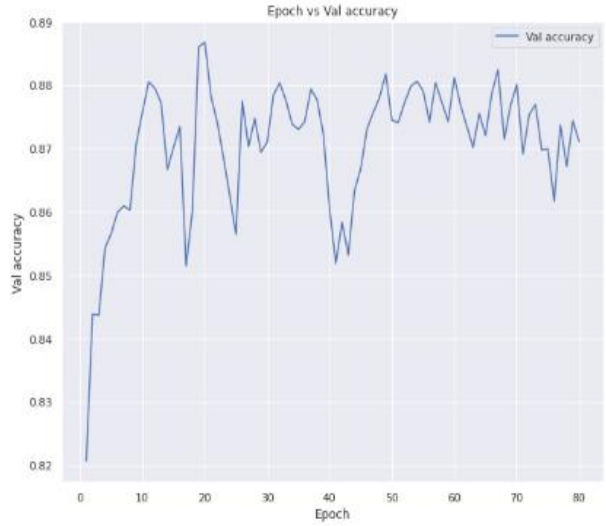
Epoch 80/80 | 48000/48000 | val_loss: 0.9235 - val_accuracy: 0.8711

PLOTS

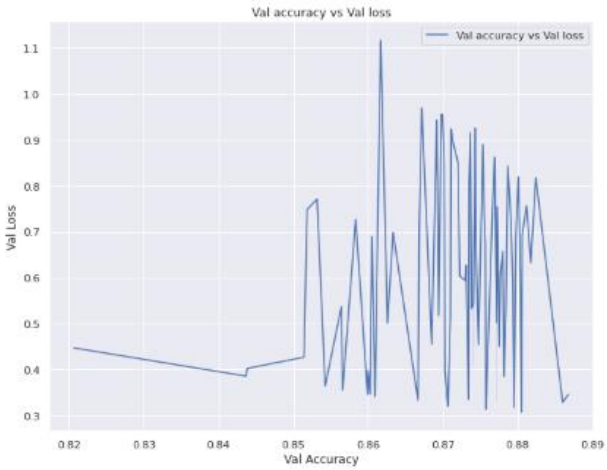

Epoch vs Train loss
Training Loss
Training Loss
Epoch


Epoch vs val loss
Val Loss
val Loss
Epoch


Epoch vs train accuracy
Train accuracy
train accuracy
Epoch


Epoch vs Val accuracy
Val accuracy
Val accuracy
Epoch


Train accuracy vs Train loss
Train accuracy vs Train loss
Train Loss
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
Val Loss
Val Accuracy

MODEL-3

Model 2 | adam = Adam(0.01) | epoch = 80 | batch_size= 64

In [51]:

```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), padding='same', activation='relu', input_shape=
(28, 28, 1)))
model.add(MaxPooling2D(strides=2))
model.add(Conv2D(filters=48, kernel_size=(5,5), padding='valid', activation='relu'))
model.add(MaxPooling2D(strides=2))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dense(5, activation='softmax'))
```

In [52]:

```
model.build()
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 48)        38448
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 5, 5, 48)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1200)              0
_________________________________________________________________
dense_4 (Dense)              (None, 256)               307456
_________________________________________________________________
dense_5 (Dense)              (None, 84)                21588
_________________________________________________________________
dense_6 (Dense)              (None, 5)                 425
=================================================================
Total params: 368,749
Trainable params: 368,749
Non-trainable params: 0
_________________________________________________________________
```

In [53]:
```
adam = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer=adam)
```

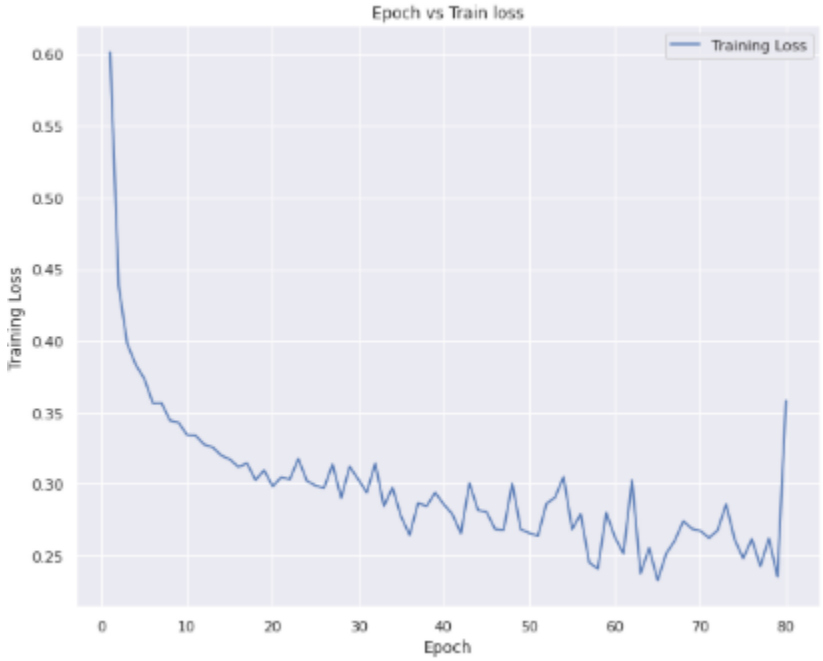
In [55]:
```
print('Time Taken',stop-start)
```

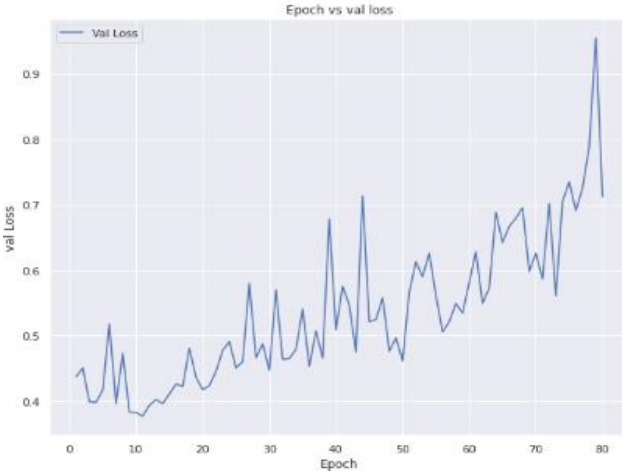
```
Time Taken 323.2294229859981
```

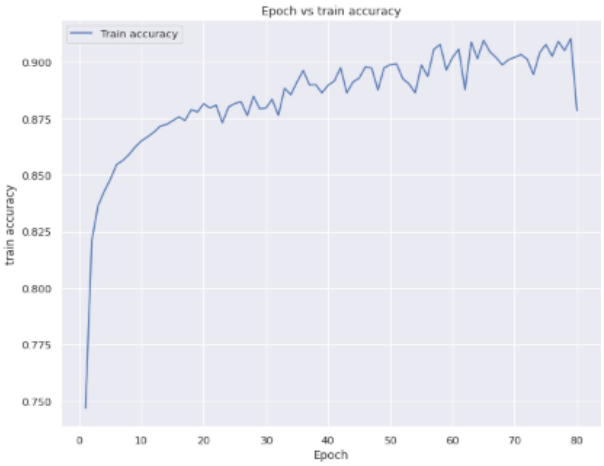
Time Taken 323.2294229859981

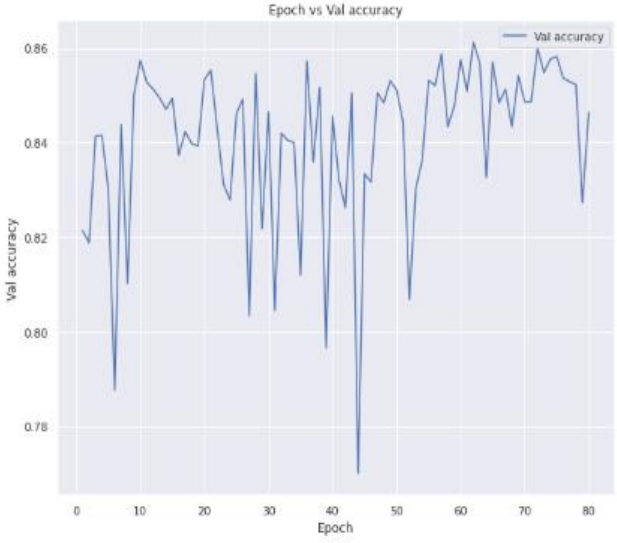
RESULT | MODEL 2 | VAL DATA ¶

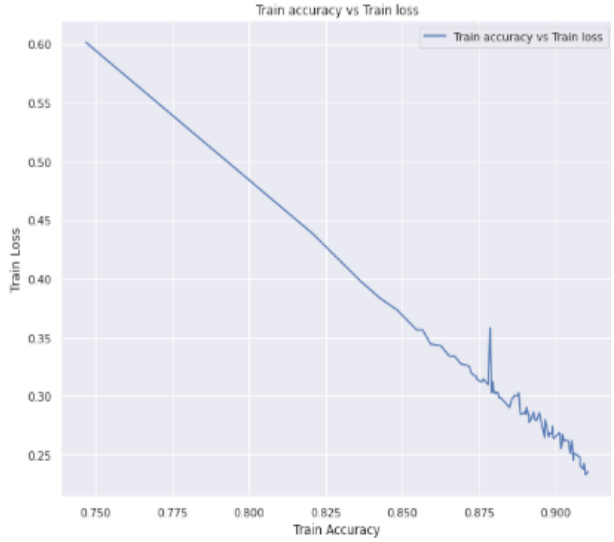
Epoch 80/80 | 48000/48000 | val_loss: 0.7129 - val_accuracy: 0.8464

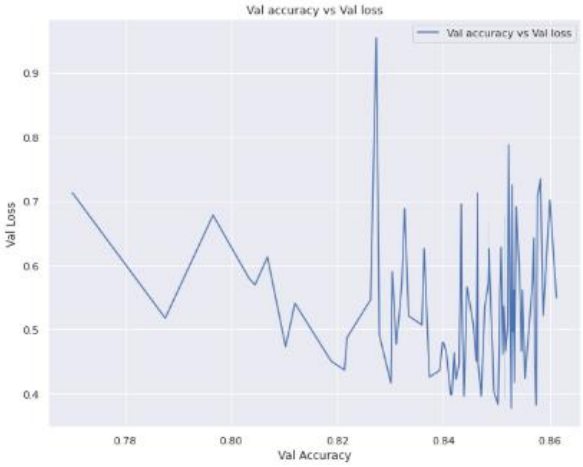



Epoch vs val loss
Val Loss
val Loss
Epoch


Epoch vs train accuracy
Train accuracy
train accuracy
Epoch


Epoch vs Val accuracy
Val accuracy
Val accuracy
Epoch


Train accuracy vs Train loss
Train accuracy vs Train loss
Train Loss
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
Val Loss
Val Accuracy

MODEL-4

Model 3 | adam = Adam(0.001)| epoch = 150 | batch_size= 32

In [79]:

```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), padding='same', activation='relu', input_shape=
(28, 28, 1)))
model.add(MaxPooling2D(strides=2))
model.add(Conv2D(filters=48, kernel_size=(5,5), padding='valid', activation='relu'))
model.add(MaxPooling2D(strides=2))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dense(5, activation='softmax'))
```

In [80]:

```
model.build()
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 10, 10, 48)        38448
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 5, 5, 48)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1200)              0
_________________________________________________________________
dense_7 (Dense)              (None, 256)               307456
_________________________________________________________________
dense_8 (Dense)              (None, 84)                21588
_________________________________________________________________
dense_9 (Dense)              (None, 5)                 425
=================================================================
Total params: 368,749
Trainable params: 368,749
Non-trainable params: 0
_________________________________________________________________
```

In [83]:

```
print('Time Taken',stop-start)
```

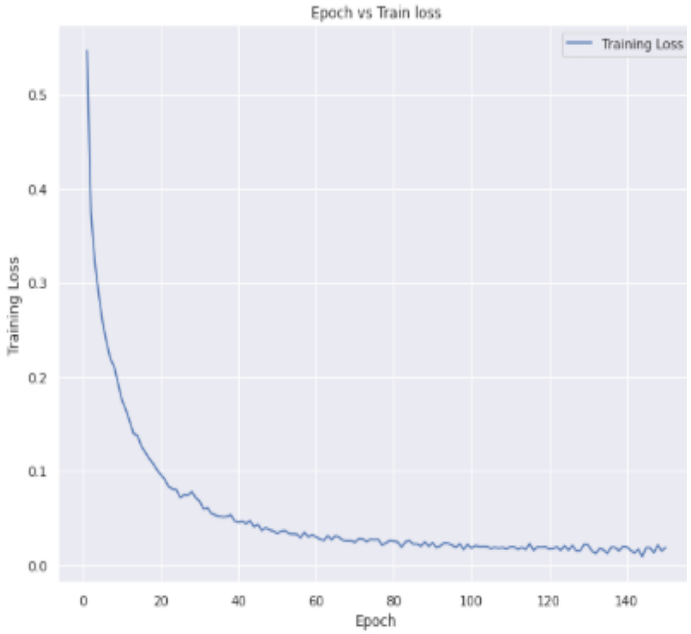
```
Time Taken 1098.0633210080014
```

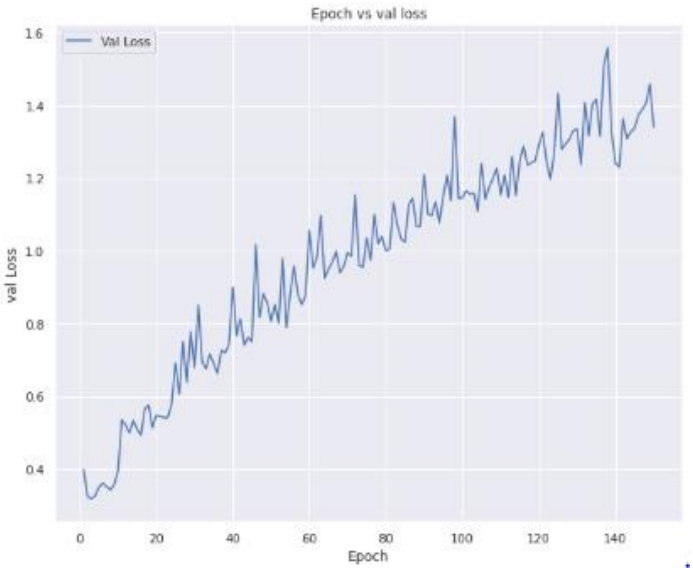
Time Taken 1098.0633210080014

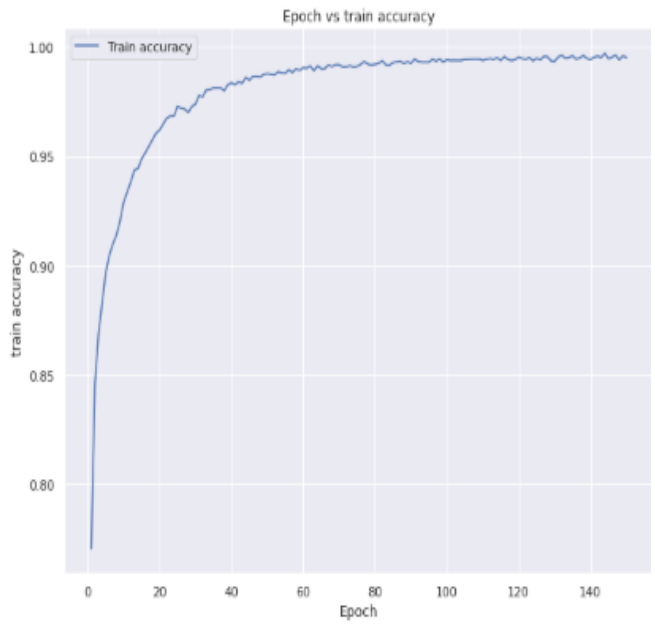
RESULT | MODEL 3 | VAL DATA

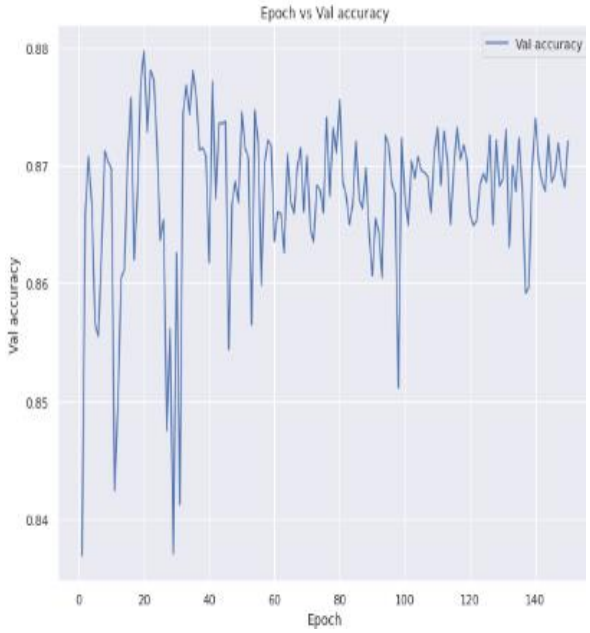
Epoch 150/150 | val_loss: 1.34 | val_accuracy: 0.8721

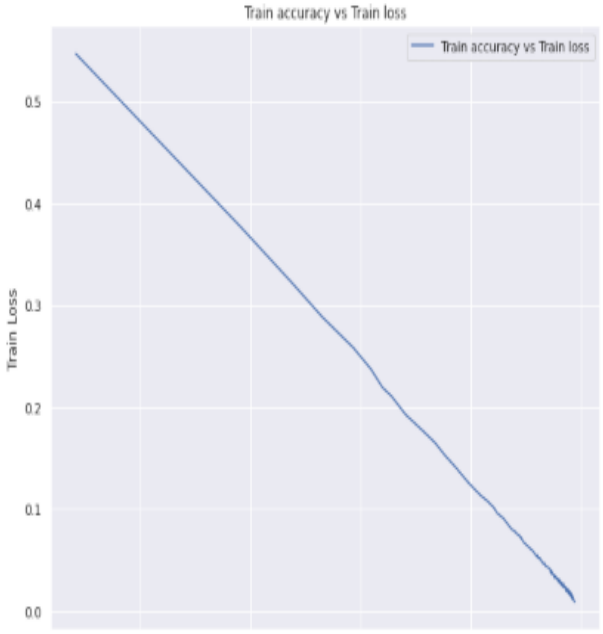
PLOTS

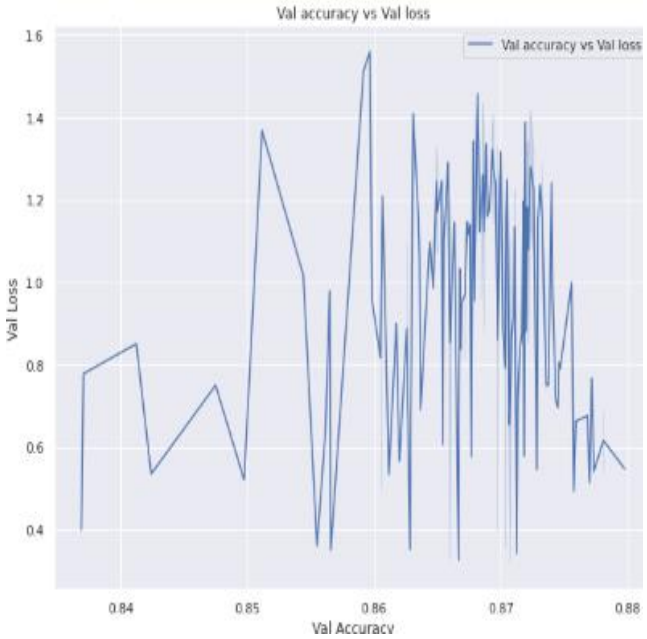





Epoch vs train accuracy
Train accuracy
train accuracy
Epoch
1.00
0.95
0.90
0.85
0.80
0
20
40
60
80
100
120
140


Epoch vs Val accuracy
Val accuracy
Val accuracy
Epoch
0.88
0.87
0.86
0.85
0.84
0
20
40
60
80
100
120
140


Train accuracy vs Train loss
Train accuracy vs Train loss
Train Loss
0.5
0.4
0.3
0.2
0.1
0.0


Val accuracy vs Val loss
Val accuracy vs Val loss
Val Loss
Val Accuracy
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.84
0.85
0.86
0.87
0.88

MODEL-5

Changing optimizer to sgd

Model 4 | SGD i.e. 0.0001 | epoch = 200 | batch_size= 32

In [109]:

```
import keras
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.losses import categorical_crossentropy
from keras import optimizers

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), padding='same', activation='relu', input_shape=
(28, 28, 1)))
model.add(MaxPooling2D(strides=2))
model.add(Conv2D(filters=48, kernel_size=(5,5), padding='valid', activation='relu'))
model.add(MaxPooling2D(strides=2))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dense(5, activation='softmax'))
```

In [110]:

```
model.build()
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 10, 10, 48)        38448
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 5, 5, 48)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 1200)              0
_________________________________________________________________
dense_10 (Dense)             (None, 256)               307456
_________________________________________________________________
dense_11 (Dense)             (None, 84)                21588
_________________________________________________________________
dense_12 (Dense)             (None, 5)                 425
=================================================================
Total params: 368,749
Trainable params: 368,749
Non-trainable params: 0
_________________________________________________________________
```

In [111]:

```
sgd = optimizers.SGD(lr=0.0001)
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer=sgd)
```

In [113]:

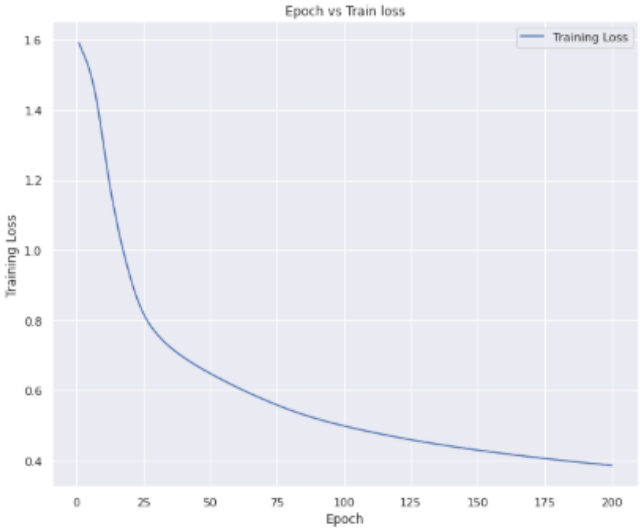
```
print('Time Taken',stop-start)
```

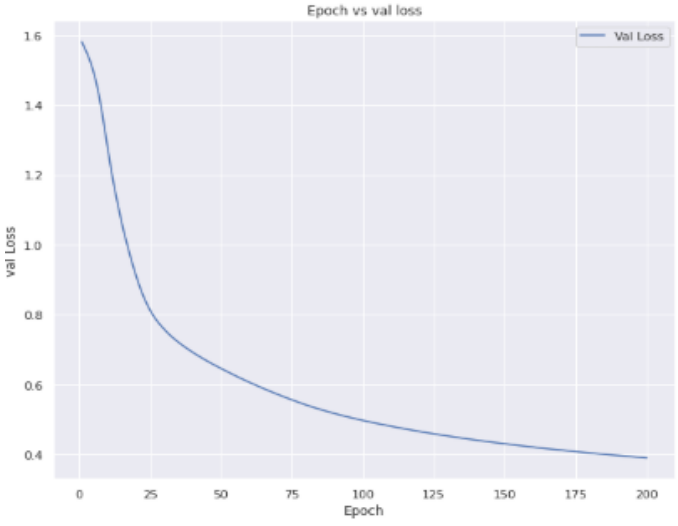
```
Time Taken 1193.4481554810009
```

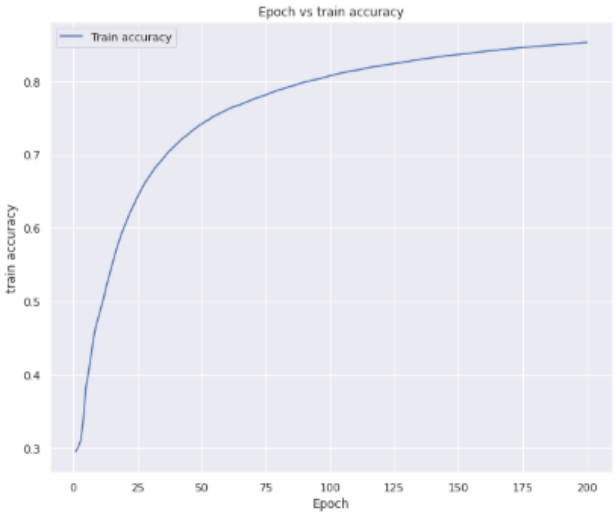
Time Taken 1193.4481554810009

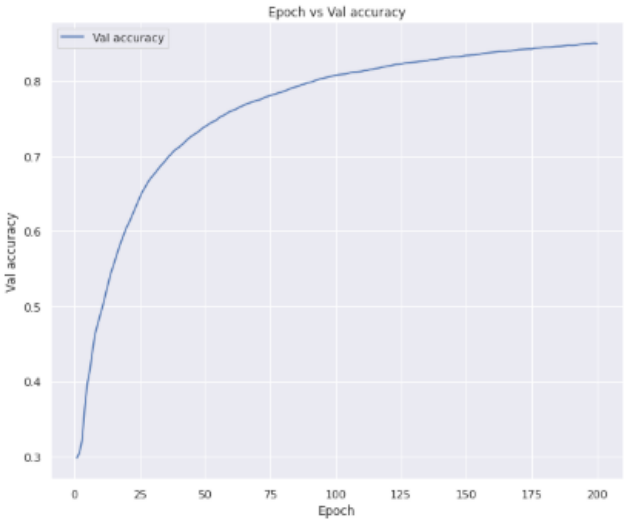
RESULT | MODEL 4 | VAL DATA

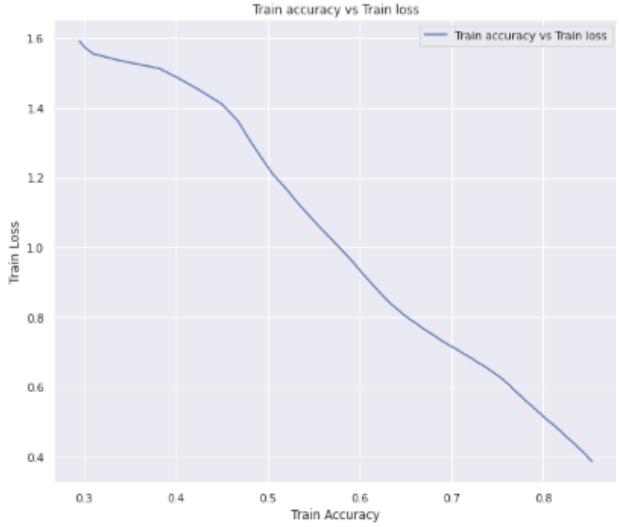
Epoch 200/200 | val_loss: 0.3898 | val_accuracy: 0.8498

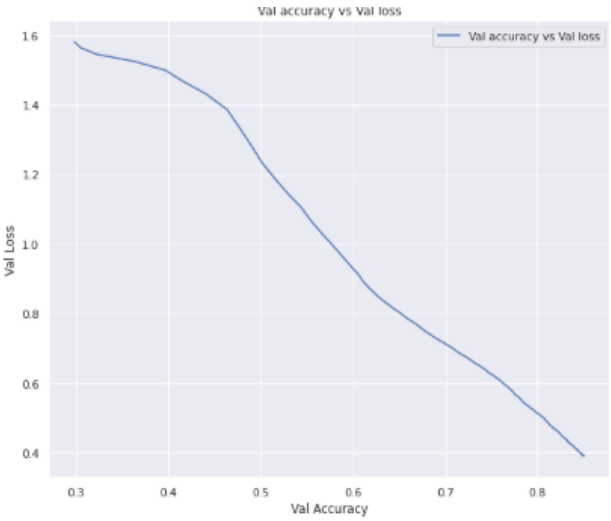





Epoch vs train accuracy
Train accuracy
train accuracy
Epoch
0.3
0.4
0.5
0.6
0.7
0.8
0
25
50
75
100
125
150
175
200


Epoch vs Val accuracy
Val accuracy
Val accuracy
Epoch
0.3
0.4
0.5
0.6
0.7
0.8
0
25
50
75
100
125
150
175
200


Train accuracy vs Train loss
Train accuracy vs Train loss
Train Loss
Train Accuracy
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0.3
0.4
0.5
0.6
0.7
0.8


Val accuracy vs Val loss
Val accuracy vs Val loss
Val Loss
Val Accuracy
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0.3
0.4
0.5
0.6
0.7
0.8

DESIGN AND RESULTS

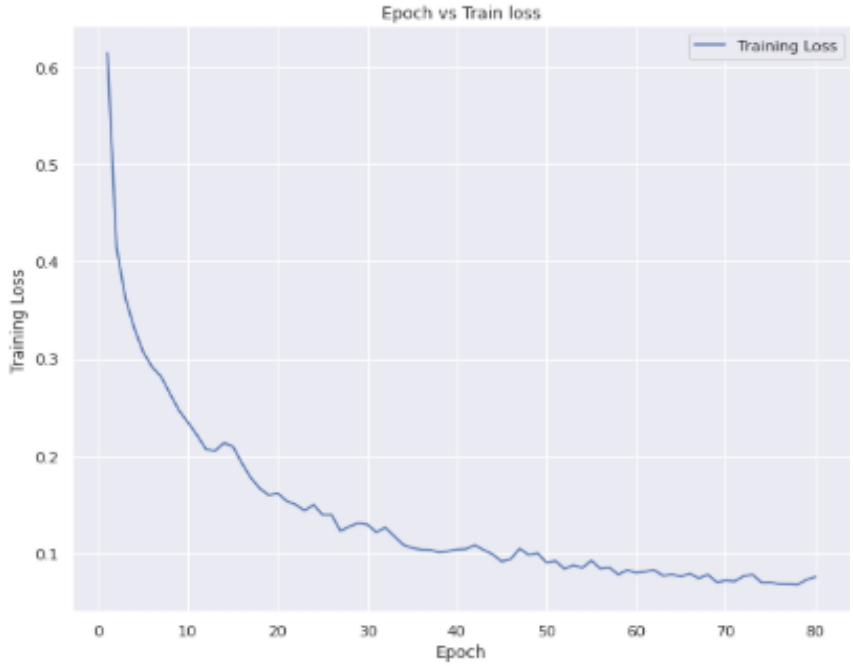
Model 2,3,4,5 are the tweaked models with changed filters and hyperparameters.

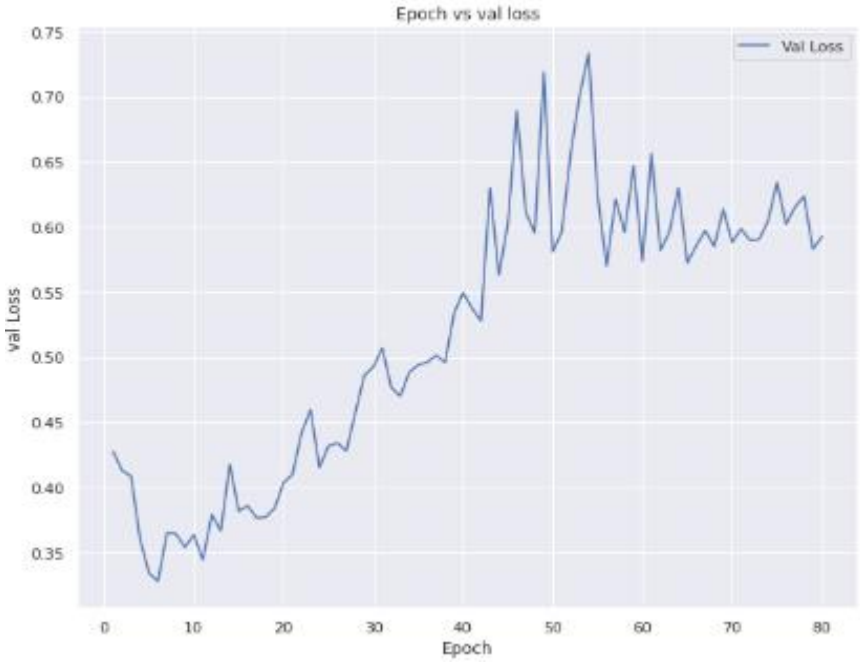
	Val Acc.	Val loss	Time	Architectural Changes	Optimizer
MODEL 1	86.18%	0.5924	332	Lenet 5 architecture	Adam(0.01)
MODEL 2	87.11%	0.9235	319.34	C1-32,C2-48,d1-256	Adam(.0005)
MODEL 3	84.64%	0.712	323	C1-32,C2-48,d1-256	Adam(0.01)
MODEL 4	87.21%	1.34	1098	C1-32,C2-48,d1-256	Adam(0.001)
MODEL 5	84.98%	0.38	1193	C1-32,C2-48,d1-256	SGD(0.0001),epoch=200

Analysis:

- Time taken by the Adam is significantly less than the SGD in Lenet given a smaller number of layers i.e. one third of time taken by SGD().
- Basic Lenet architecture provides a decent accuracy of 86.18% with a loss of 0.59 on validation data.
- Our tweaked model 4 provided the better accuracy of 87.21% but the loss has also increased drastically making the model not suitable given the same parameters with few tweaks.
- SGD() doesn't seem to add anything better to the accuracy when the number of layers are less.
- Here, we can choose the original Lenet architecture as the best architecture to proceed with.

Graphical Analysis



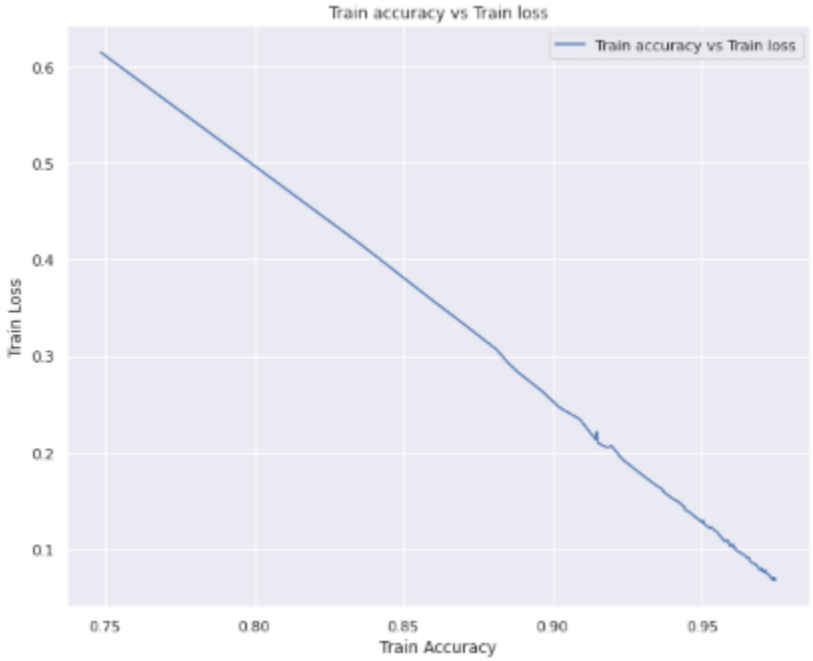


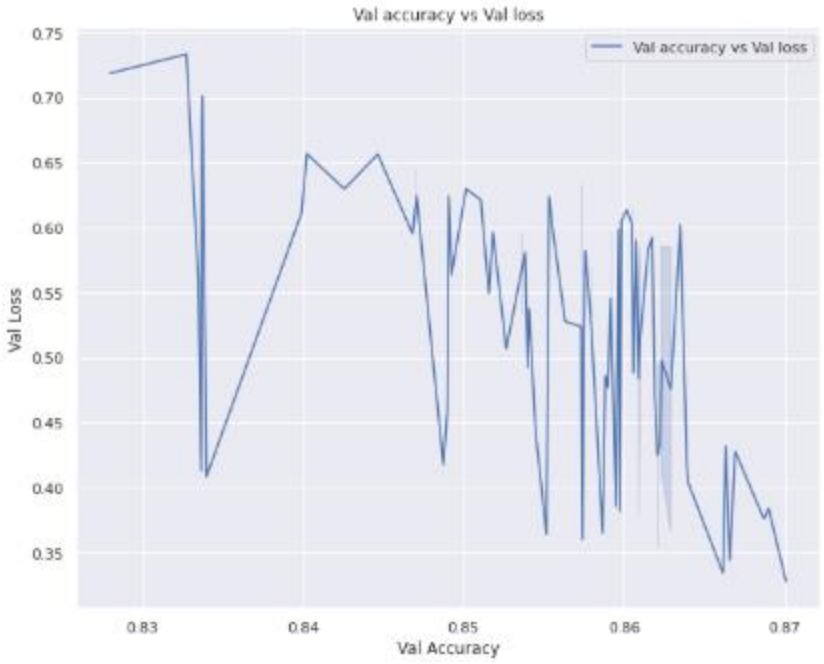
Analysis:

- **Training Epoch vs Loss:**
  - The training loss decreased from 0.6 to 0.05 from [1-80] epochs.

- In the first 40 epochs, loss got decreased by 6 times to 0.1.

- **Testing Epoch vs Loss:**
  - The range of the loss is between 0.33 to 0.73 with the final settling of loss at 0.59.
  - There are heavy fluctuations in the loss between the range of [40,60] epochs.
  - Loss didn't stabilize at any epoch and the learning rate was also not too high(not too many weights updates).





Analysis:

- **In Train loss vs Train accuracy,**
  - The loss has decreased from 0.6 to 0.05 with an increase in the accuracy from 75% to 97% within the complete range of epochs.

- **In Test loss vs Test accuracy,**
  - There is a dramatic change in the loss with respect to the accuracy.
  - At 83.5% accuracy, loss was 0.40 which was a decent score but following that the loss started fluctuating but accuracy increased.[Between the accuracy of 85% to 87%].

Our model is less confident on its correct predictions which can be concluded from the above observations.

RESNET

ARCHITECTURE

ResNet (residual network) is a type of ANN used for classification of multiclass dataset which uses skip connection to mitigate the problem of vanishing gradient. The skip connection makes the model learn the identity function which makes sure that the higher layers will not face the effect of performance deterioration and will perform the same way the lower layers performed.

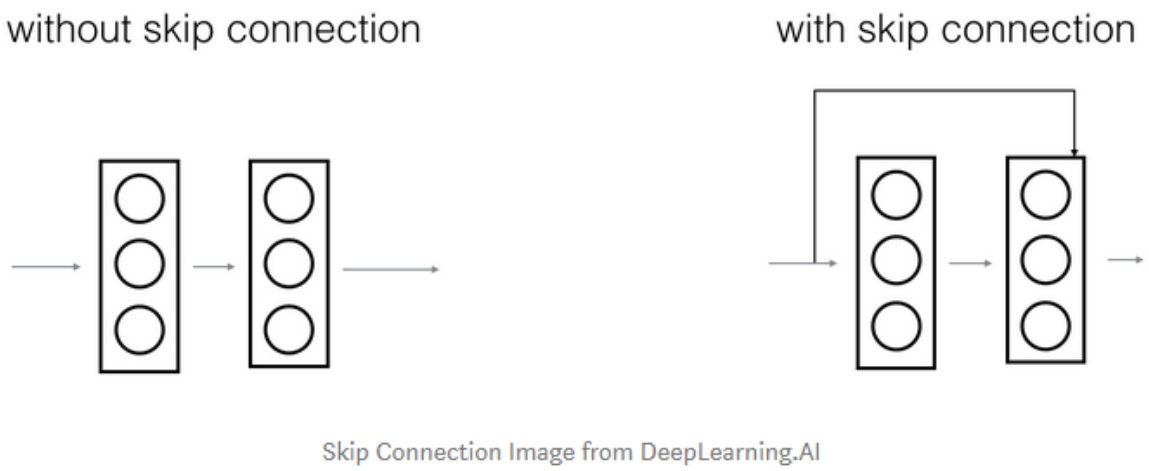


FIG:4 Skip Connection

MODEL-1

MODEL 1 -- > max_pool=False | batch_size=64 |

RESULT-- > val data→ 0.8656 | val loss →0.5257

In [21]:

```
#Defining constants
epochs = 50
batch_size = 64
data_augmentation = False
img_size = 28
num_classes = 5

num_filters = 64
num_blocks = 4
num_sub_blocks = 2
use_max_pool = False

inputs = Input(shape=input_size)
x = Conv2D(num_filters, padding='same',
           kernel_initializer='he_normal',
           kernel_size=7, strides=2,
           kernel_regularizer=l2(1e-4))(inputs)
x = BatchNormalization()(x)
x = Activation('relu')(x)
```

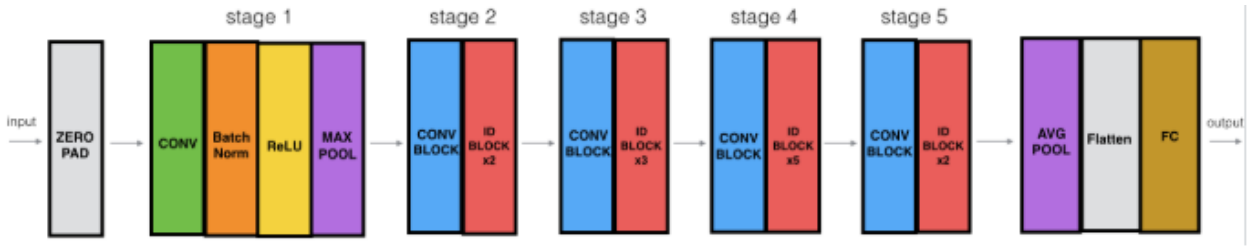


FIG:5 RESNET ARCHITECTURE

```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            (None, 28, 28, 1)    0
__________________________________________________________________________________________________
conv2d_21 (Conv2D)              (None, 14, 14, 64)   3200        input_2[0][0]
__________________________________________________________________________________________________
batch_normalization_18 (BatchNo (None, 14, 14, 64)   256         conv2d_21[0][0]
__________________________________________________________________________________________________
activation_18 (Activation)      (None, 14, 14, 64)   0           batch_normalization_18[0]
[0]
__________________________________________________________________________________________________
conv2d_22 (Conv2D)              (None, 14, 14, 64)   36928       activation_18[0][0]
__________________________________________________________________________________________________
batch_normalization_19 (BatchNo (None, 14, 14, 64)   256         conv2d_22[0][0]
__________________________________________________________________________________________________
activation_19 (Activation)      (None, 14, 14, 64)   0           batch_normalization_19[0]
[0]
__________________________________________________________________________________________________
dropout_17 (Dropout)            (None, 14, 14, 64)   0           activation_19[0][0]
__________________________________________________________________________________________________
conv2d_23 (Conv2D)              (None, 14, 14, 64)   36928       dropout_17[0][0]
__________________________________________________________________________________________________
batch_normalization_20 (BatchNo (None, 14, 14, 64)   256         conv2d_23[0][0]
__________________________________________________________________________________________________
add_9 (Add)                     (None, 14, 14, 64)   0           activation_18[0][0]
                                                                 batch_normalization_20[0]
```

```
------
activation_20 (Activation)      (None, 14, 14, 64)   0           add_9[0][0]
------------------------------------------------------------------------------------------------
------
dropout_18 (Dropout)            (None, 14, 14, 64)   0           activation_20[0][0]
------------------------------------------------------------------------------------------------
------
conv2d_24 (Conv2D)              (None, 14, 14, 64)   36928       dropout_18[0][0]
------------------------------------------------------------------------------------------------
------
batch_normalization_21 (BatchNo (None, 14, 14, 64)   256         conv2d_24[0][0]
------------------------------------------------------------------------------------------------
------
activation_21 (Activation)      (None, 14, 14, 64)   0           batch_normalization_21[0]
[0]
------------------------------------------------------------------------------------------------
------
dropout_19 (Dropout)            (None, 14, 14, 64)   0           activation_21[0][0]
------------------------------------------------------------------------------------------------
------
conv2d_25 (Conv2D)              (None, 14, 14, 64)   36928       dropout_19[0][0]
------------------------------------------------------------------------------------------------
------
batch_normalization_22 (BatchNo (None, 14, 14, 64)   256         conv2d_25[0][0]
------------------------------------------------------------------------------------------------
------
add_10 (Add)                    (None, 14, 14, 64)   0           dropout_18[0][0]
                                                                 batch_normalization_22[0]
[0]
------------------------------------------------------------------------------------------------
------
activation_22 (Activation)      (None, 14, 14, 64)   0           add_10[0][0]
------------------------------------------------------------------------------------------------
------
dropout_20 (Dropout)            (None, 14, 14, 64)   0           activation_22[0][0]
------------------------------------------------------------------------------------------------
------
conv2d_26 (Conv2D)              (None, 7, 7, 128)    73856       dropout_20[0][0]
------------------------------------------------------------------------------------------------
------
batch_normalization_23 (BatchNo (None, 7, 7, 128)    512         conv2d_26[0][0]
------------------------------------------------------------------------------------------------
------
activation_23 (Activation)      (None, 7, 7, 128)    0           batch_normalization_23[0]
[0]
------------------------------------------------------------------------------------------------
------
dropout_21 (Dropout)            (None, 7, 7, 128)    0           activation_23[0][0]
------------------------------------------------------------------------------------------------
------
conv2d_27 (Conv2D)              (None, 7, 7, 128)    147584      dropout_21[0][0]
------------------------------------------------------------------------------------------------
------
conv2d_28 (Conv2D)              (None, 7, 7, 128)    8320        dropout_20[0][0]
------------------------------------------------------------------------------------------------
------
batch_normalization_24 (BatchNo (None, 7, 7, 128)    512         conv2d_27[0][0]
------------------------------------------------------------------------------------------------
------
add_11 (Add)                    (None, 7, 7, 128)    0           conv2d_28[0][0]
                                                                 batch_normalization_24[0]
```

```
[0]
__________________________________________________________________________________________________
activation_24 (Activation)      (None, 7, 7, 128)    0           add_11[0][0]
__________________________________________________________________________________________________
dropout_22 (Dropout)            (None, 7, 7, 128)    0           activation_24[0][0]
__________________________________________________________________________________________________
conv2d_29 (Conv2D)              (None, 7, 7, 128)    147584      dropout_22[0][0]
__________________________________________________________________________________________________
batch_normalization_25 (BatchNo (None, 7, 7, 128)    512         conv2d_29[0][0]
__________________________________________________________________________________________________
activation_25 (Activation)      (None, 7, 7, 128)    0           batch_normalization_25[0]
[0]
__________________________________________________________________________________________________
dropout_23 (Dropout)            (None, 7, 7, 128)    0           activation_25[0][0]
__________________________________________________________________________________________________
conv2d_30 (Conv2D)              (None, 7, 7, 128)    147584      dropout_23[0][0]
__________________________________________________________________________________________________
batch_normalization_26 (BatchNo (None, 7, 7, 128)    512         conv2d_30[0][0]
__________________________________________________________________________________________________
add_12 (Add)                    (None, 7, 7, 128)    0           dropout_22[0][0]
                                                                 batch_normalization_26[0]
```

```
[0]
__________________________________________________________________________________________________
activation_26 (Activation)      (None, 7, 7, 128)    0           add_12[0][0]
__________________________________________________________________________________________________
dropout_24 (Dropout)            (None, 7, 7, 128)    0           activation_26[0][0]
__________________________________________________________________________________________________
conv2d_31 (Conv2D)              (None, 4, 4, 256)    295168      dropout_24[0][0]
__________________________________________________________________________________________________
batch_normalization_27 (BatchNo (None, 4, 4, 256)    1024        conv2d_31[0][0]
__________________________________________________________________________________________________
activation_27 (Activation)      (None, 4, 4, 256)    0           batch_normalization_27[0]
[0]
__________________________________________________________________________________________________
dropout_25 (Dropout)            (None, 4, 4, 256)    0           activation_27[0][0]
__________________________________________________________________________________________________
conv2d_32 (Conv2D)              (None, 4, 4, 256)    590080      dropout_25[0][0]
__________________________________________________________________________________________________
conv2d_33 (Conv2D)              (None, 4, 4, 256)    33024       dropout_24[0][0]
__________________________________________________________________________________________________
batch_normalization_28 (BatchNo (None, 4, 4, 256)    1024        conv2d_32[0][0]
__________________________________________________________________________________________________
add_13 (Add)                    (None, 4, 4, 256)    0           conv2d_33[0][0]
                                                                 batch_normalization_28[0]
```

```
[0]
__________________________________________________________________________________________________
activation_28 (Activation)      (None, 4, 4, 256)    0           add_13[0][0]
__________________________________________________________________________________________________
dropout_26 (Dropout)            (None, 4, 4, 256)    0           activation_28[0][0]
__________________________________________________________________________________________________
conv2d_34 (Conv2D)              (None, 4, 4, 256)    590080      dropout_26[0][0]
__________________________________________________________________________________________________
batch_normalization_29 (BatchNo (None, 4, 4, 256)    1024        conv2d_34[0][0]
__________________________________________________________________________________________________
activation_29 (Activation)      (None, 4, 4, 256)    0           batch_normalization_29[0]
[0]
__________________________________________________________________________________________________
dropout_27 (Dropout)            (None, 4, 4, 256)    0           activation_29[0][0]
__________________________________________________________________________________________________
conv2d_35 (Conv2D)              (None, 4, 4, 256)    590080      dropout_27[0][0]
__________________________________________________________________________________________________
batch_normalization_30 (BatchNo (None, 4, 4, 256)    1024        conv2d_35[0][0]
__________________________________________________________________________________________________
add_14 (Add)                    (None, 4, 4, 256)    0           dropout_26[0][0]
                                                                 batch_normalization_30[0]
```

```
[0]
__________________________________________________________________________________________________
activation_30 (Activation)      (None, 4, 4, 256)    0           add_14[0][0]
__________________________________________________________________________________________________
dropout_28 (Dropout)            (None, 4, 4, 256)    0           activation_30[0][0]
__________________________________________________________________________________________________
conv2d_36 (Conv2D)              (None, 2, 2, 512)    1180160     dropout_28[0][0]
__________________________________________________________________________________________________
batch_normalization_31 (BatchNo (None, 2, 2, 512)    2048        conv2d_36[0][0]
__________________________________________________________________________________________________
activation_31 (Activation)      (None, 2, 2, 512)    0           batch_normalization_31[0]
[0]
__________________________________________________________________________________________________
dropout_29 (Dropout)            (None, 2, 2, 512)    0           activation_31[0][0]
__________________________________________________________________________________________________
conv2d_37 (Conv2D)              (None, 2, 2, 512)    2359808     dropout_29[0][0]
__________________________________________________________________________________________________
conv2d_38 (Conv2D)              (None, 2, 2, 512)    131584      dropout_28[0][0]
__________________________________________________________________________________________________
batch_normalization_32 (BatchNo (None, 2, 2, 512)    2048        conv2d_37[0][0]
__________________________________________________________________________________________________
add_15 (Add)                    (None, 2, 2, 512)    0           conv2d_38[0][0]
                                                                 batch_normalization_32[0]
```

```
[0]
__________________________________________________________________________________________________
______
activation_32 (Activation)      (None, 2, 2, 512)    0           add_15[0][0]
__________________________________________________________________________________________________
______
dropout_30 (Dropout)            (None, 2, 2, 512)    0           activation_32[0][0]
__________________________________________________________________________________________________
______
conv2d_39 (Conv2D)              (None, 2, 2, 512)    2359808     dropout_30[0][0]
__________________________________________________________________________________________________
______
batch_normalization_33 (BatchNo (None, 2, 2, 512)    2048        conv2d_39[0][0]
__________________________________________________________________________________________________
______
activation_33 (Activation)      (None, 2, 2, 512)    0           batch_normalization_33[0]
[0]
__________________________________________________________________________________________________
______
dropout_31 (Dropout)            (None, 2, 2, 512)    0           activation_33[0][0]
__________________________________________________________________________________________________
______
conv2d_40 (Conv2D)              (None, 2, 2, 512)    2359808     dropout_31[0][0]
__________________________________________________________________________________________________
______
batch_normalization_34 (BatchNo (None, 2, 2, 512)    2048        conv2d_40[0][0]
__________________________________________________________________________________________________
______
add_16 (Add)                    (None, 2, 2, 512)    0           dropout_30[0][0]
                                                                 batch_normalization_34[0]
```

```
[0]
__________________________________________________________________________________________________
______
activation_34 (Activation)      (None, 2, 2, 512)    0           add_16[0][0]
__________________________________________________________________________________________________
______
dropout_32 (Dropout)            (None, 2, 2, 512)    0           activation_34[0][0]
__________________________________________________________________________________________________
______
average_pooling2d_2 (AveragePoo (None, 1, 1, 512)    0           dropout_32[0][0]
__________________________________________________________________________________________________
______
flatten_2 (Flatten)             (None, 512)          0           average_pooling2d_2[0][0]
__________________________________________________________________________________________________
______
dense_2 (Dense)                 (None, 5)            2565        flatten_2[0][0]
==================================================================================================
======
Total params: 11,183,621
Trainable params: 11,175,813
Non-trainable params: 7,808
__________________________________________________________________________________________________
```

Saving the model--

In [22]:
```
save_dir = os.path.join(os.getcwd(), 'saved_model')
model_name = 'fmnist.h5'
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
filepath = os.path.join(save_dir,model_name)
print(filepath)
```

```
/kaggle/working/saved_model/fmnist.h5
```

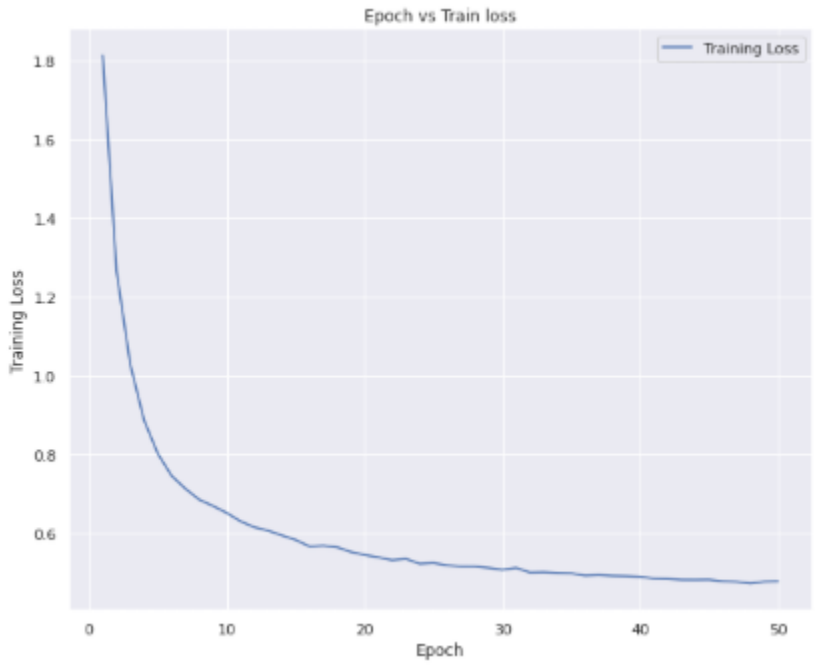
Time Taken 1824.5438640850007

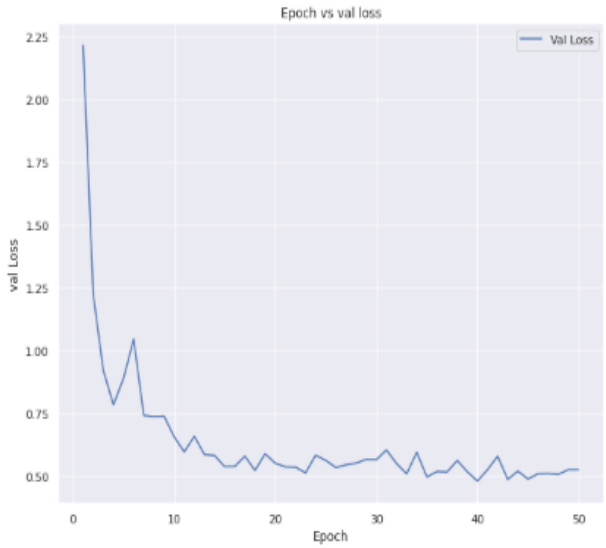
In [25]:
```
time_model1=stop-start
print('Time Taken',time_model1)
```

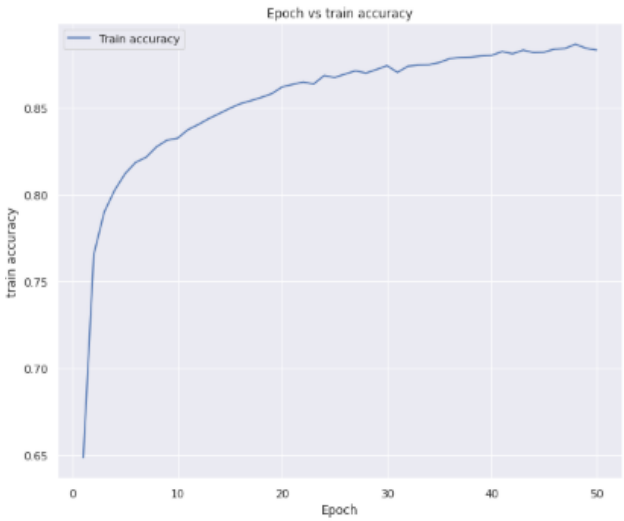
```
Time Taken 1824.5438640850007
```

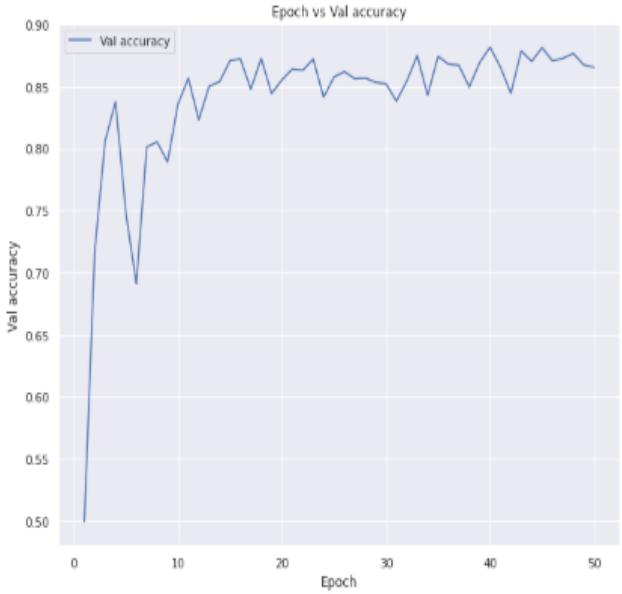
PLOTS--

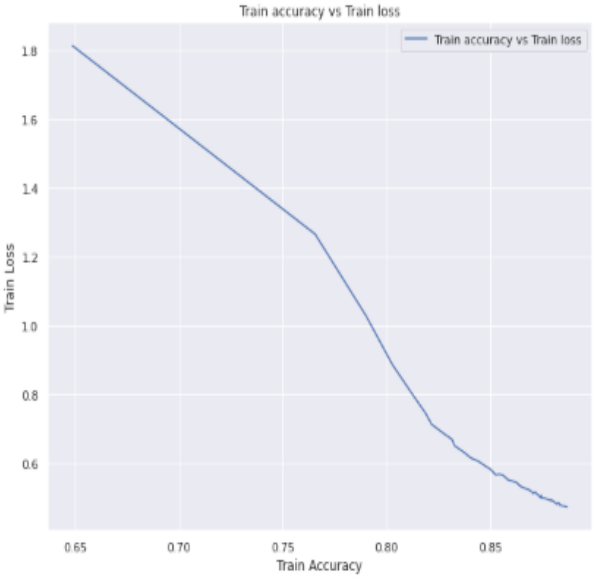
In [26]:
```
a.history
```

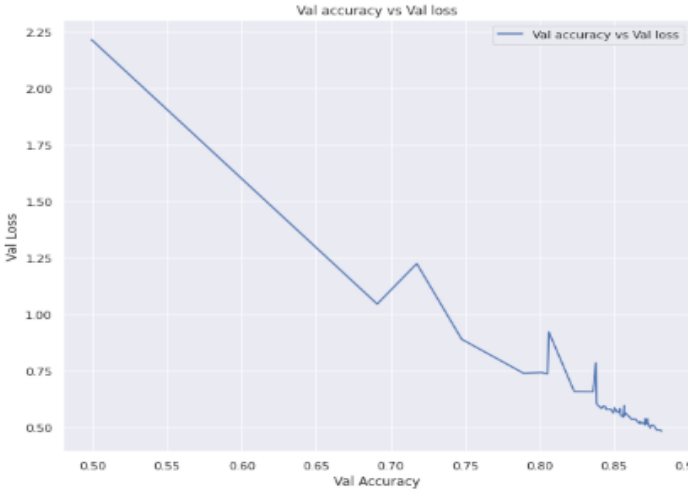



Epoch vs val loss
Val Loss
val Loss
Epoch


Epoch vs train accuracy
Train accuracy
train accuracy
Epoch


Epoch vs Val accuracy
Val accuracy
Val accuracy
Epoch


Train accuracy vs Train loss
Train accuracy vs Train loss
Train Loss
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
Val Loss
Val Accuracy

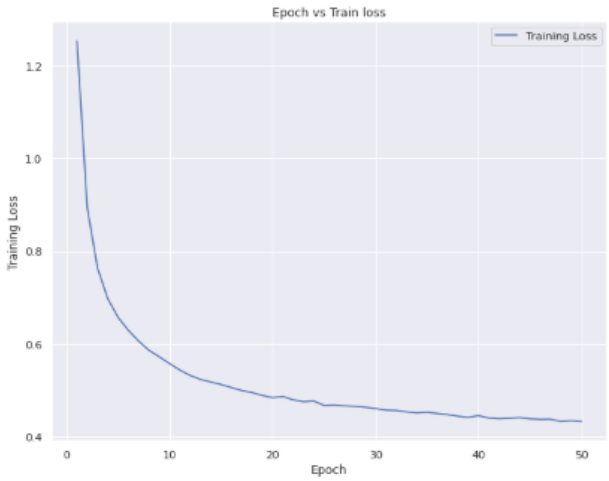
MODEL2 ::→with maxpool=True and batch size small=32 | val_accuracy = 0.8487 | val loss 0.5429

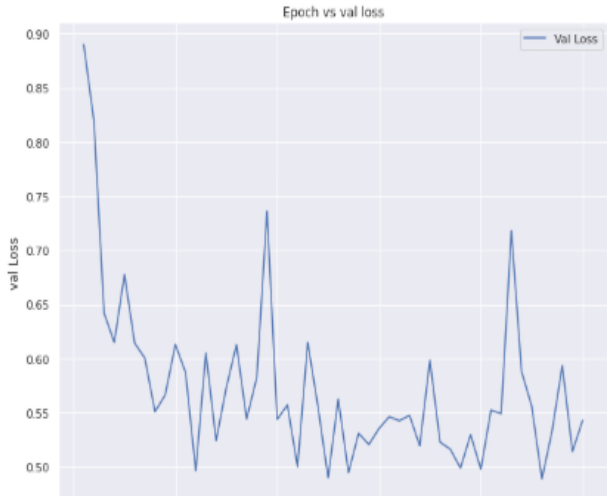
In [60]:

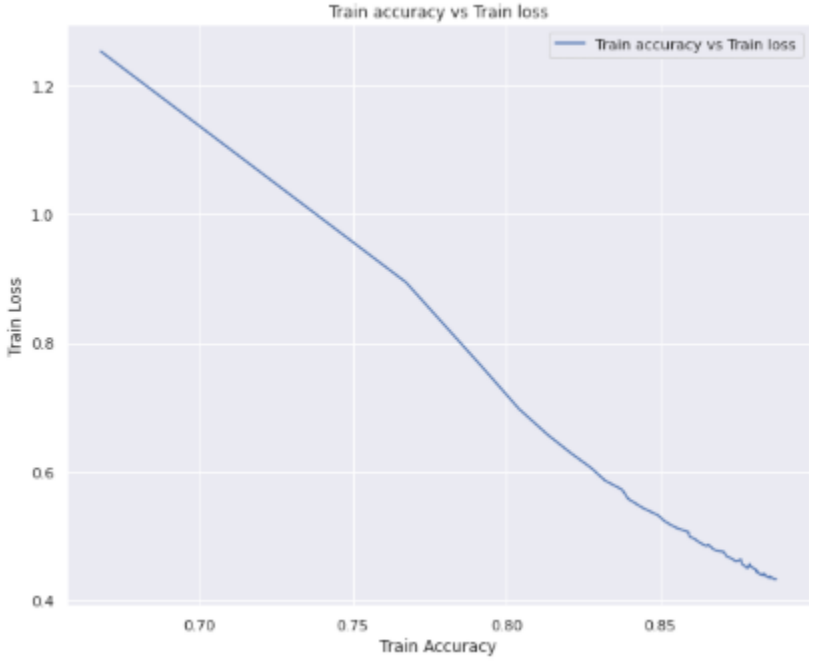
```
#Defining constants
epochs = 50
batch_size = 32
data_augmentation = False
img_size = 28

num_classes = 5
num_filters = 64
num_blocks = 4
num_sub_blocks = 2
use_max_pool = True
```

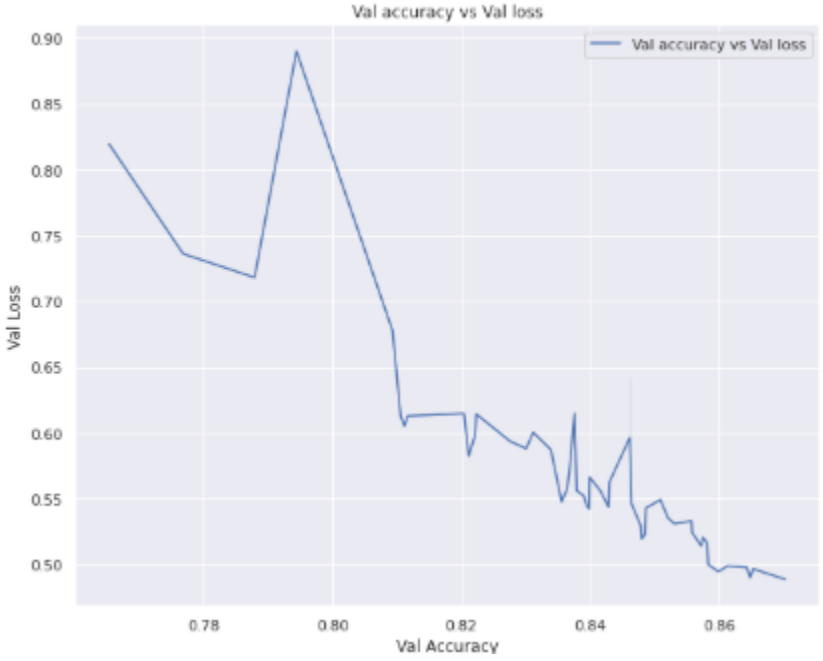
Time Taken 2306.2938370890006

PLOTS--






Train accuracy vs Train loss
Train accuracy vs Train loss
1.2
1.0
0.8
0.6
0.4
Train Loss
0.70
0.75
0.80
0.85
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
Val Loss
0.78
0.80
0.82
0.84
0.86
Val Accuracy

MODEL 3 → OPTIMIZER= SGD | max_pool=False | batch_size=64 |val accuracy- 0.5820| val_loss: 2.1792 |epoch=50

In [96]:

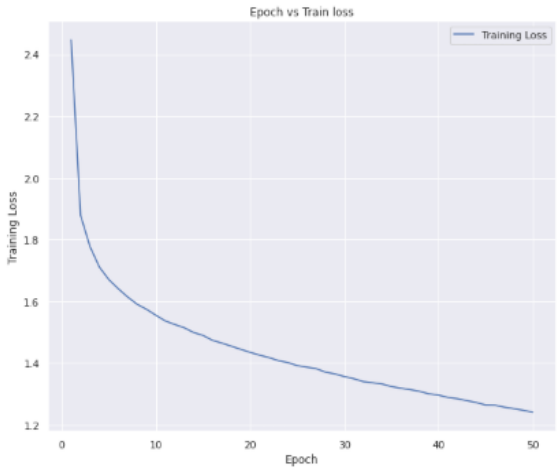
```
#Defining constants
epochs = 50
batch_size = 64
data_augmentation = False
img_size = 28
num_classes = 5

num_filters = 64
num_blocks = 4
num_sub_blocks = 2
use_max_pool = False

#Creating model based on ResNet published archietecture
inputs = Input(shape=input_size)
x = Conv2D(num_filters, padding='same',
           kernel_initializer='he_normal',
           kernel_size=7, strides=2,
           kernel_regularizer=l2(1e-4))(inputs)
x = BatchNormalization()(x)
x = Activation('relu')(x)
```

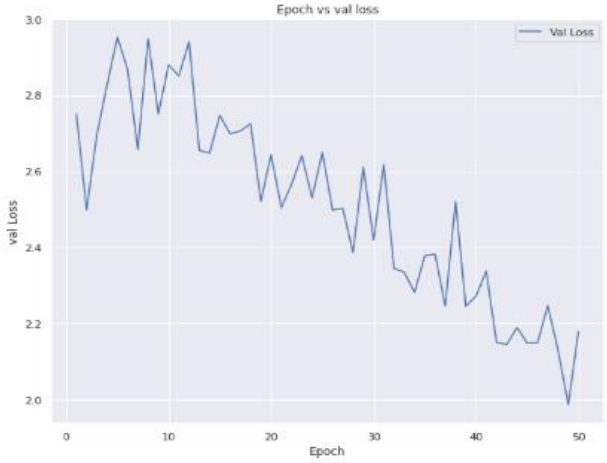
Time Taken 1596.5268217260018

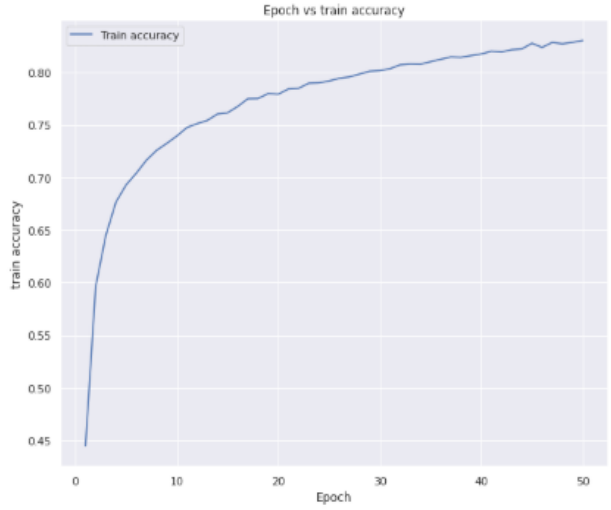
In [99]:

```
time_model1=stop-start
print('Time Taken',time_model1)
```

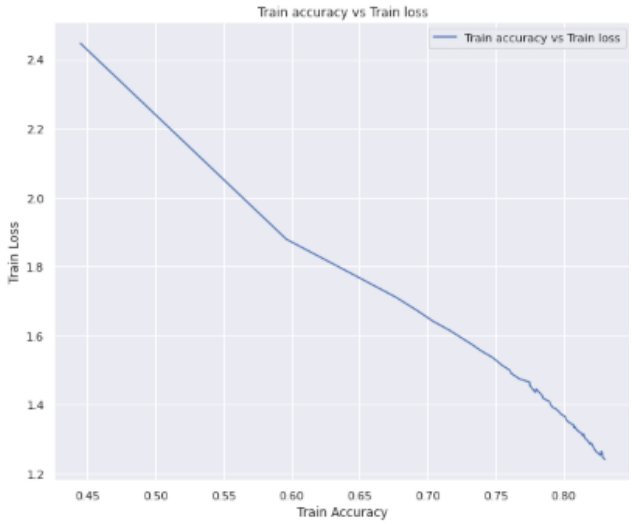
```
Time Taken 1596.5268217260018
```

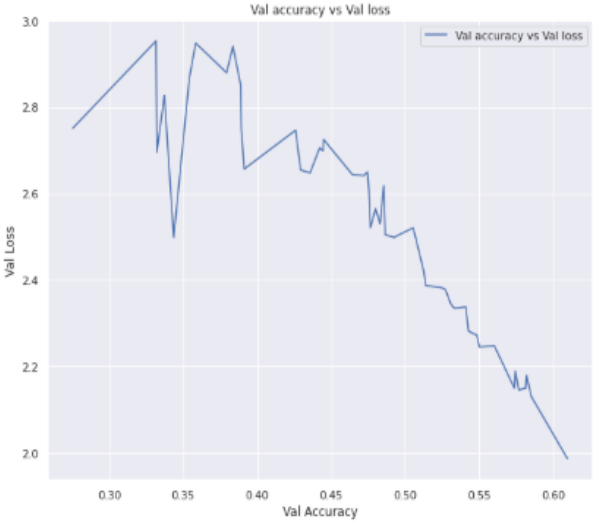

Epoch vs Train loss
Training Loss
Training Loss
Epoch


Epoch vs val loss
Val Loss
Val Loss
Epoch


Epoch vs train accuracy
Train accuracy
train accuracy
Epoch


Epoch vs Val accuracy
Val accuracy
Val accuracy
Epoch


Train accuracy vs Train loss
Train accuracy vs Train loss
Train Loss
Train Accuracy


Val accuracy vs Val loss
Val accuracy vs Val loss
Val Loss
Val Accuracy

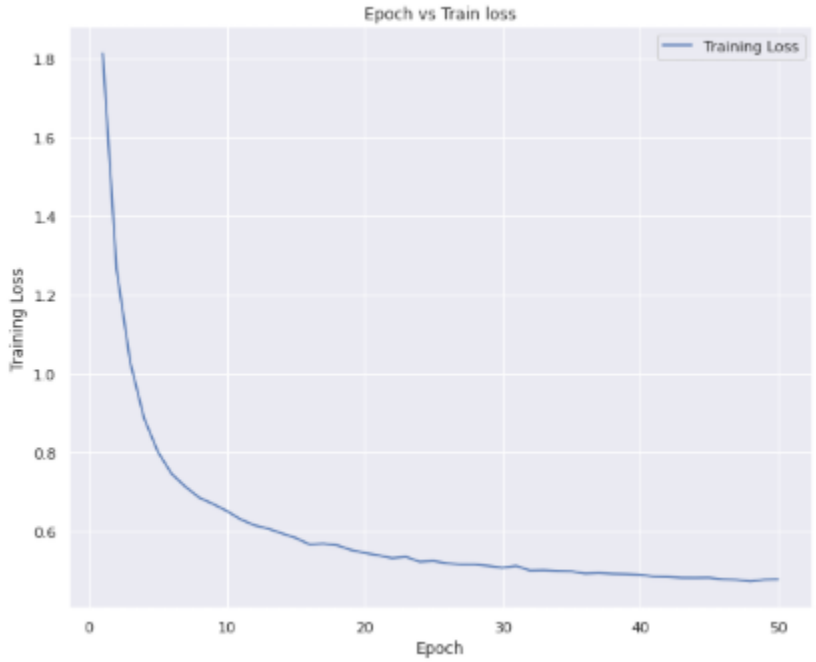
RESULTS AND ANALYSIS

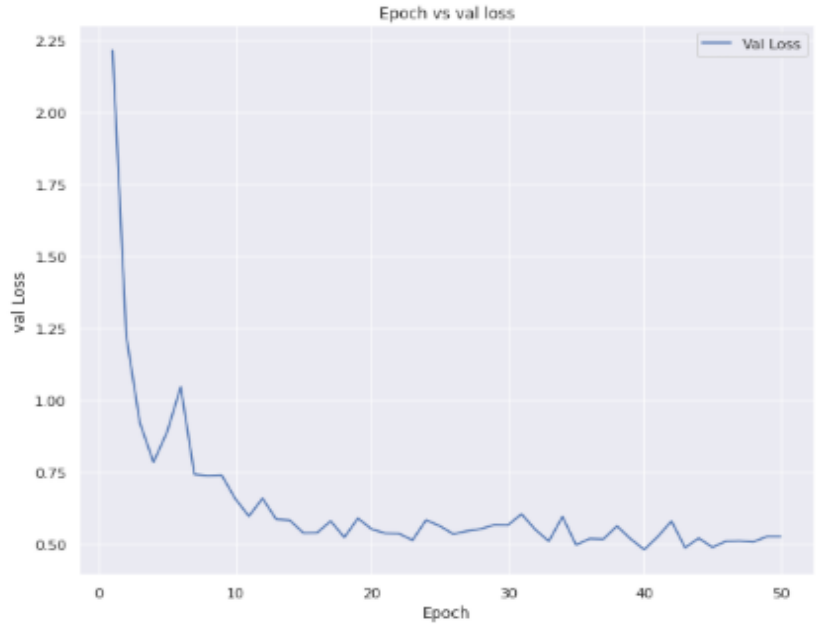
	Optimizer	Val acc.	Val loss	Time	MP
MODEL 1	Adam()	86.56%	0.52	1824	False
MODEL 2	Adam()	84.87%	0.54	2306	True
MODEL 3	SGD()	58%	2.17	1596	False

- Basic Resnet with No tweaks didn't work well on our dataset. There can be other combinations which can converge well. But the core architecture gave an accuracy of 86.56%.{Max_pooling was removed}.
- When we tweaked the model by adding the max_pooling, the accuracy further reduced by 2%.
- Using SGD() didn't improve the accuracy as well as time.
- Time taken by the RESNET is very high when compared with the Lenet-5 or other architectures.

Graphical Analysis

Best architecture→ Model:1



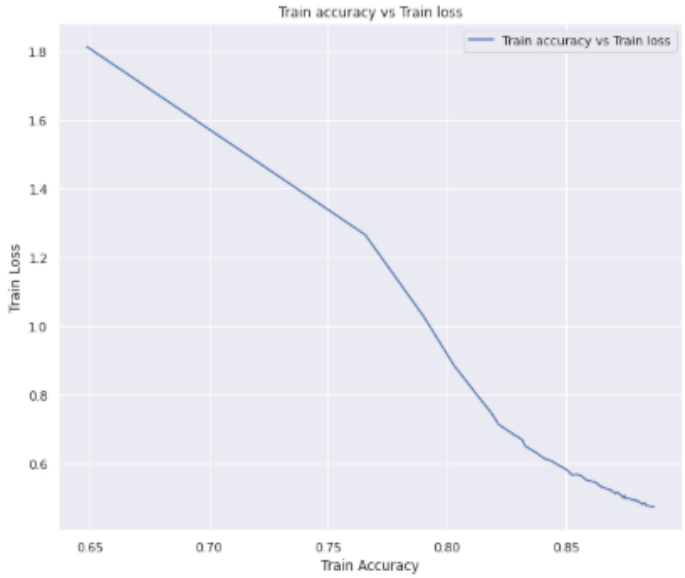


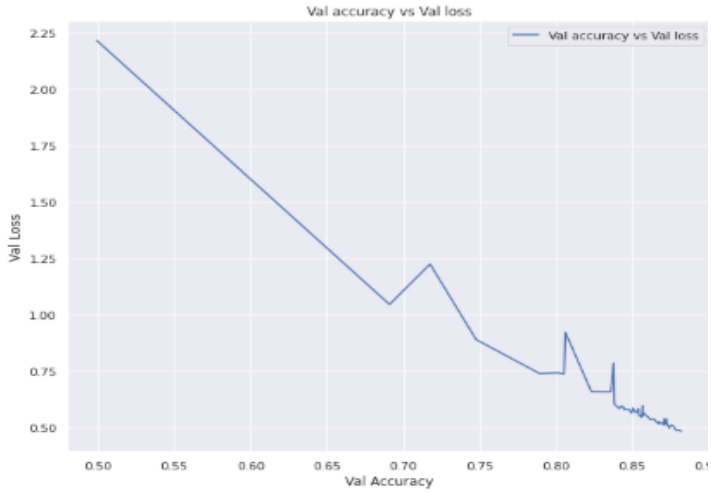
Analysis:

- **Training Epoch vs Loss:**
  - Overall loss decreased from 1.8 to 0.5 within the complete range of epochs.
  - In range [30,50] of epochs, loss was stabilized at 0.5.
  - Till 10^{th} epoch, the loss fell from 1.8 to 0.6 for the trainset.

- **Testing Epoch vs Loss:**
  - Loss fell drastically from 2.25 to 0.75 at the 4^{th} epoch.
  - Overall range of the loss was between 2.25-0.50.
  - Loss stabilized after 45th epoch at approximately around 0.50.

  From the analysis we can conclude that the basic Resnet didn't provide any improvement to our model in terms of accuracy with high loss as well and the time complexity is also increased.





Analysis:

- **In Train loss vs Train accuracy,**
  - The loss has decreased from 1.8 to 0.4 with an increase in the accuracy from 65% to 87% within the complete range of epochs.

- **In Test loss vs Test accuracy,**

- Loss decreased initially with an increase in the accuracy and then there are some fluctuations at the higher values of accuracy which were also seen in Lenet-5.

COMPARISON AT A GLANCE

	CNN	VGG	ALEXNET	LENET-5	RESNET
Model	Model 7	Model1	Model 1	Model 1	Model 1
Accuracy%	91.12	86.03%	87.42%	86.18	86.56%
Runtime	927	1751	737	332	1824
Complexity	12	19	11	7	50
Loss	0.25	0.65	0.41	0.59	0.52

- **CNN(Model 7):**
  - **Parameters:**
    - **SGD(lr=0.01,decay=1e-6,nesterov=True)**
    - **ReduceLROnPlateau**
    - **BatchNormalization()**
    - **BS:64**
  - **Complexity:**
    - **Convolution Layers: 10**
    - **Max Pooling Layers: 4**
    - **Dropouts: 3**
    - **Dense: 3**
- **VGG(Model 1 {Tweaked VGG}):**
  - **Parameters:**
    - **SGD()**
    - **BS:64**
    - **All other basic parameters**
  - **Complexity:**

        - **Convolution Layers: 10**
        - **Max Pooling Layers: 4**
        - **Dropouts: 2**
        - **Dense: 3**
- **AlexNet(Model 1{Tweaked}):**
    - **Parameters:**
        - **SGD()**
        - **BS:64**
        - **All other basic parameters**
    - **Complexity:**
        - **Convolution Layers: 5**
        - **Pooling Layers: 3**
        - **Dense Layers: 3**
- **Lenet-5(Model 1):**
    - **Parameters:**
        - **Adam()**
        - **BS:64**
        - **All other basic parameters**
    - **Complexity:**
        - **Convolution Layers: 2**
        - **Max Pooling layers: 2**
        - **Dense Layers: 3**
- **RESNET(Model 1):**
    - **Parameters:**
        - **All the basic parameters**
    - **Complexity:**
        - **Convolution Layers: 48**
        - **Pooling Layers: 2**

1. **We found out that our CNN gave the best results with much better time complexity followed by the Alexnet and Lenet-5 when both the accuracy and time complexity is considered.**
2. **RESNET with some tweaks can work better on our dataset.**
3. **Complexity of our CNN is very impressive with very few layers and some tweaks of learning rate reduction with batch normalization on specified layers.**
4. **Validation loss on our model is also minimum when compared with all the other models.**
5. **Kaggle score achieved on our CNN is 90.9% which is a descent score with this much time complexity.**

Citations:

- http://cs231n.github.io/convolutional-networks/#overview
- https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
- https://medium.com/syncedreview/iclr-2019-fast-as-adam-good-as-sgd-new-optimizer-has-both-78e37e8f9a34
- https://www.quora.com/In-convolutional-neural-networks-what-effect-does-the-size-e-g-3x3-5x5-7x7-of-the-convolution-kernel-have-on-the-architecture-of-the-convolutional-neural-networks
- https://medium.com/syncedreview/iclr-2019-fast-as-adam-good-as-sgd-new-optimizer-has-both-78e37e8f9a34
- https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c
- https://keras.io/optimizers/
- https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau
- Accelerating Deep Neural Networks on Low Power Heterogeneous Architectures - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/3D-input-to-1D-array-row-by-row-transformation_fig2_327070011
- Automated Diatom Classification (Part B): A Deep Learning Approach - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/AlexNet-architecture-layers_tbl2_316639667
- https://engmrk.com/lenet-5-a-classic-cnn-architecture/
- https://cv-tricks.com/keras/understand-implement-resnets/
- https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33
- https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cc5d0adf648e
- https://int8.io/comparison-of-optimization-techniques-stochastic-gradient-descent-momentum-adagrad-and-adadelta/