Model performance with different activation function

In [1]:
```
import pickle
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten, BatchNormalization
from keras.layers import Dense, Dropout
from keras import regularizers
from keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import np_utils
import keras


def load_train_data(n):
    with open('data_batch_'+ str(n), 'rb') as file:
        batch = pickle.load(file, encoding='latin1')

    features = batch['data']
    Target = batch['labels']
    return features, Target


batch_1, Target_1 = load_train_data(1)
batch_2, Target_2 = load_train_data(2)
batch_3, Target_3 = load_train_data(3)
batch_4, Target_4 = load_train_data(4)
batch_5, Target_5 = load_train_data(5)


with open('test_batch', 'rb') as file:
```

```
    batch = pickle.load(file, encoding='latin1')
X_test = batch['data']
y_test = batch['labels']


X_train = np.append(batch_1, batch_2,axis=0)
X_train = np.append(X_train, batch_3,axis=0)
X_train = np.append(X_train, batch_4,axis=0)
X_train = np.append(X_train, batch_5,axis=0)
y_train = np.append(Target_1, Target_2,axis=0)
y_train = np.append(y_train, Target_3,axis=0)
y_train = np.append(y_train, Target_4,axis=0)
y_train = np.append(y_train, Target_5,axis=0)
X_train = X_train.reshape((len(X_train), 3, 32, 32)).transpose(0,2,3,1)
y_train = np_utils.to_categorical(y_train, 10)
X_test = X_test.reshape((len(X_test), 3, 32, 32)).transpose(0,2,3,1)
y_test = np_utils.to_categorical(y_test, 10)
X_train = X_train.astype('float32')
X_test= X_test.astype('float32')
X_train= X_train / 255.0
X_test= X_test/ 255.0
```

```
Using TensorFlow backend.
```

Model 7

Lets check the performance of the model with different activation function

1. Activation function - tanh

In [2]:
```
model7 = Sequential()
model7.add(Conv2D(64, (3, 3), activation='tanh',kernel_initializer='he_
normal',kernel_regularizer=regularizers.l2(0.001), input_shape=(32, 32,
 3)))
model7.add(Conv2D(64, (3, 3), activation='tanh',kernel_initializer='he_
normal',kernel_regularizer=regularizers.l2(0.001)))
model7.add(MaxPooling2D((2, 2)))
```

```
model7.add(Conv2D(64, (3, 3), activation='tanh',kernel_initializer='he_
normal',kernel_regularizer=regularizers.l2(0.001)))
model7.add(Conv2D(64, (3, 3), activation='tanh',kernel_initializer='he_
normal',kernel_regularizer=regularizers.l2(0.001)))
model7.add(MaxPooling2D((2, 2)))
model7.add(Conv2D(64, (3, 3), activation='tanh',kernel_initializer='he_
normal',kernel_regularizer=regularizers.l2(0.001)))
model7.add(MaxPooling2D((2, 2)))
model7.add(Flatten())
model7.add(Dense(128, activation='tanh'))
model7.add(Dense(10, activation='softmax'))
model7.summary()
```

```
WARNING:tensorflow:From C:\Users\Dhanajayan\Anaconda3\lib\site-packages
\tensorflow\python\framework\op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 30, 30, 64)        1792
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 64)        36928
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 1, 1, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 64)                0
_________________________________________________________________
```

```
dense_1 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 159,114
Trainable params: 159,114
Non-trainable params: 0
_________________________________________________________________
```

In [4]:

```
epochs = 10
sgd = SGD(lr=1e-2, momentum=0.9, decay=1e-2/epochs)
model7.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=
['accuracy'])
model7.fit(X_train,y_train,epochs=epochs,batch_size = 32)
```

```
WARNING:tensorflow:From C:\Users\Dhanajayan\Anaconda3\lib\site-packages
\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.pyth
on.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/10
50000/50000 [==============================] - 314s 6ms/step - loss: 2.
2085 - acc: 0.3831
Epoch 2/10
50000/50000 [==============================] - 329s 7ms/step - loss: 1.
6649 - acc: 0.5655
Epoch 3/10
50000/50000 [==============================] - 357s 7ms/step - loss: 1.
4640 - acc: 0.6237
Epoch 4/10
50000/50000 [==============================] - 365s 7ms/step - loss: 1.
3423 - acc: 0.6588
Epoch 5/10
50000/50000 [==============================] - 396s 8ms/step - loss: 1.
2575 - acc: 0.6838
Epoch 6/10
50000/50000 [==============================] - 398s 8ms/step - loss: 1.
1922 - acc: 0.7048
Epoch 7/10
50000/50000 [==============================] - 416s 8ms/step - loss: 1.
```

```
1362 - acc: 0.7189
Epoch 8/10
50000/50000 [==============================] - 418s 8ms/step - loss: 1.
0973 - acc: 0.7305
Epoch 9/10
50000/50000 [==============================] - 418s 8ms/step - loss: 1.
0587 - acc: 0.7438
Epoch 10/10
50000/50000 [==============================] - 419s 8ms/step - loss: 1.
0247 - acc: 0.7510
```

Out[4]:
```
<keras.callbacks.History at 0x1de8fe663c8>
```

In [6]:
```
test_loss,test_acc = model7.evaluate(X_test,y_test)
test_acc
```

```
10000/10000 [==============================] - 38s 4ms/step
```

Out[6]:
```
0.6786
```

Obeservation

- tanh performs bad than Relu activation in terms of speed and accuracy

Model 8

- activation function Elu

In [8]:
```
model8 = Sequential()
model8.add(Conv2D(64, (3, 3), activation='elu',kernel_initializer='he_n
ormal',kernel_regularizer=regularizers.l2(0.001), input_shape=(32, 32,
3)))
model8.add(Conv2D(64, (3, 3), activation='elu',kernel_initializer='he_n
```

```
ormal',kernel_regularizer=regularizers.l2(0.001)))
model8.add(MaxPooling2D((2, 2)))
model8.add(Conv2D(64, (3, 3), activation='elu',kernel_initializer='he_n
ormal',kernel_regularizer=regularizers.l2(0.001)))
model8.add(Conv2D(64, (3, 3), activation='elu',kernel_initializer='he_n
ormal',kernel_regularizer=regularizers.l2(0.001)))
model8.add(MaxPooling2D((2, 2)))
model8.add(Conv2D(64, (3, 3), activation='elu',kernel_initializer='he_n
ormal',kernel_regularizer=regularizers.l2(0.001)))
model8.add(MaxPooling2D((2, 2)))
model8.add(Flatten())
model8.add(Dense(128, activation='elu'))
model8.add(Dense(10, activation='softmax'))
model8.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 30, 30, 64)        1792
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 28, 28, 64)        36928
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 12, 12, 64)        36928
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 10, 10, 64)        36928
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 3, 3, 64)          36928
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 1, 1, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290
=================================================================
```

```
Total params: 159,114
Trainable params: 159,114
Non-trainable params: 0
_________________________________________________________________
```

In [9]:
```
epochs = 10
sgd = SGD(lr=1e-2, momentum=0.9, decay=1e-2/epochs)
model8.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=
['accuracy'])
model8.fit(X_train,y_train,epochs=epochs,batch_size = 32)
```

```
Epoch 1/10
50000/50000 [==============================] - 404s 8ms/step - loss: 2.
0755 - acc: 0.4462
Epoch 2/10
50000/50000 [==============================] - 405s 8ms/step - loss: 1.
5509 - acc: 0.6118
Epoch 3/10
50000/50000 [==============================] - 400s 8ms/step - loss: 1.
3608 - acc: 0.6678
Epoch 4/10
50000/50000 [==============================] - 401s 8ms/step - loss: 1.
2410 - acc: 0.7036
Epoch 5/10
50000/50000 [==============================] - 403s 8ms/step - loss: 1.
1574 - acc: 0.7273
Epoch 6/10
50000/50000 [==============================] - 400s 8ms/step - loss: 1.
0916 - acc: 0.7453
Epoch 7/10
50000/50000 [==============================] - 403s 8ms/step - loss: 1.
0422 - acc: 0.7603
Epoch 8/10
50000/50000 [==============================] - 412s 8ms/step - loss: 0.
9968 - acc: 0.7753
Epoch 9/10
50000/50000 [==============================] - 405s 8ms/step - loss: 0.
9592 - acc: 0.7865
Epoch 10/10
```

```
50000/50000 [==============================] - 376s 8ms/step - loss: 0.
9264 - acc: 0.7946
```

Out[9]:
```
<keras.callbacks.History at 0x1defe47b6d8>
```

In [10]:
```
test_loss,test_acc = model7.evaluate(X_test,y_test)
test_acc
```

```
10000/10000 [==============================] - 34s 3ms/step
```

Out[10]:
```
0.6786
```

Observation

The activation function 'elu' had more training efficiency but it tooks long computation and overfits the data. Therefore will consider Relu activation for further models

In [ ]:
```

```