Assignment3_UMLT_2022_instructions

June 12, 2022

1 Using Machine Learning Tools 2021, Assignment 3

1.1 Sign Language Image Classification using Deep Learning

1.2 Overview

In this assignment you will implement different deep learning networks to classify images of hands in poses that correspond to letters in American Sign Language. The dataset is contained in the assignment zip file, along with some images and a text file describing the dataset. It is similar in many ways to other MNIST datasets.

The main aims of the assignment are:

- To implement and train different types of deep learning network;
- To systematically optimise the architecture and parameters of the networks;
- To explore over-fitting and know what appropriate actions to take in these cases.

It is the intention that this assignment will take you through the process of implementing and optimising deep learning approaches. The way that you work is more important than the results for this assignment, as what is most crucial for you to learn is how to take a dataset, understand the problem, write appropriate code, optimize performance and present results. A good understanding of the different aspects of this process and how to put them together well (which will not always be the same, since different problems come with different constraints or difficulties) is the key to being able to effectively use deep learning techniques in practice.

This assignment relates to the following ACS CBOK areas: abstraction, design, hardware and software, data and information, HCI and programming.

1.3 Scenario

A client is interested in having you (or rather the company that you work for) investigate whether it is possible to develop an app that would enable American sign language to be translated for people that do not sign, or those that sign in different languages/styles. They have provided you with a labelled data of images related to signs (hand positions) that represent individual letters in order to do a preliminary test of feasibility.

Your manager has asked you to do this feasibility assessment, but subject to a constraint on the computational facilities available. More specifically, you are asked to do **no more than 50 training runs in total** (including all models and hyperparameter settings that you consider).

In addition, you are told to **create a validation set and any necessary test sets using *only* the supplied testing dataset.** It is unusual to do this, but here the training set contains a lot of non-independent, augmented images and it is important that the validation images must be totally independent of the training data and not made from augmented instances of training images.

The clients have asked to be informed about the following: - **unbiased accuracy** estimate of a deep learning model (since DL models are fast when deployed) - the letter with the lowest individual accuracy - the most common error (of one letter being incorrectly labelled as another)

Your manager has asked you to create a jupyter notebook that shows the following: - loading the data, checking it, fixing any problems, and displaying a sample - training and optimising both **densely connected** *and* **CNN** style models - finding the best one, subject to a rapid turn-around and corresponding limit of 50 training runs in total - reporting clearly what networks you have tried, the method you used to optimise them, the associated learning curves, their summary performance and selection process to pick the best model - this should be clear enough that another employee, with your skillset, should be able to take over from you and understand your methods - results from the model that is selected as the best, showing the information that the clients have requested - a statistical test between the best and second-best models, to see if there is any significant difference in performance (overall accuracy) - it is hoped that the accuracy will exceed 96% overall and better than 90% for every individual letter, and you are asked to: - report the overall accuracy - report the accuracy for each individual letter - write a short recommendation regarding how likely you think it is to achieve these goals either with the current model or by continuing to do a small amount of model development/optimisation

1.4 Guide to Assessment

This assignment is much more free-form than others in order to test your ability to run a full analysis like this one from beginning to end, using the correct procedures. So you should use a methodical approach, as a large portion of the marks are associated with the decisions that you take and the approach that you use. There are no marks associated with the performance - just report what you achieve, as high performance does not get better marks - to get good marks you need to use the right steps, as you've used in other assignments and workshops.

Make sure that you follow the instructions found in the scenario above, as this is what will be marked. And be careful to do things in a way that gives you an *unbiased* result.

The notebook that you submit should be similar to those in the other assignments, where it is important to clearly structure your outputs and code so that it could be understood by your manager or your co-worker - or, even more importantly, the person marking it! This does not require much writing, beyond the code, comments and the small amount that you've seen in previous assignments. Do not write long paragraphs to explain every detail of everything you do - it is not that kind of report and longer is definitely not better. Just make your code clear, your outputs easy to understand (short summaries often help here), and include a few small markdown cells that describe or summarise things when necessary.

Marks for the assignment will be determined according to the general rubric that you can find on MyUni, with a breakdown into sections as follows: - 10%: Loading, investigating, manipulating and displaying data - 20%: Initial model successfully trained (and acting as a baseline) - 45%: Optimisation of an appropriate set of models in an appropriate way (given the constraint of 50 training runs) - 25%: Comparison of models, selection of the best two and reporting of final results

Remember that most marks will be for the **steps you take**, rather than the achievement of any particular results. There will also be marks for showing appropriate understanding of the results that you present.

What you need to do this assignment can all be found in the first 10 weeks of workshops, lectures and also the previous two assignments. The one exception to this is the statistical test, which will be covered in week 11.

1.5 Final Instructions

While you are free to use whatever IDE you like to develop your code, your submission should be formatted as a Jupyter notebook that interleaves Python code with output, commentary and analysis. - Your code must use the current stable versions of python libraries, not outdated versions. - All data processing must be done within the notebook after calling appropriate load functions. - Comment your code, so that its purpose is clear to the reader! - In the submission file name, do not use spaces or special characters.

The marks for this assignment are mainly associated with making the right choices and executing the workflow correctly and efficiently. Make sure you have clean, readable code as well as producing outputs, since your coding will also count towards the marks (however, excessive commenting is discouraged and will lose marks, so aim for a modest, well-chosen amount of comments and text in outputs).

This assignment can be solved using methods from sklearn, pandas, matplotlib and keras, as presented in the workshops. Other high-level libraries should not be used, even though they might have nice functionality such as automated hyperparameter or architecture search/tuning/optimisation. For the deep learning parts please restrict yourself to the library calls used in workshops 7-10 or ones that are very similar to these. You are expected to search and carefully read the documentation for functions that you use, to ensure you are using them correctly.

As ususal, feel free to use code from the workshops as a base for this assignment but be aware that they will normally not do *exactly* what you want (code examples rarely do!) and so you will need to make suitable modifications.

The assignment is worth 35% of your overall mark for the course.

Mark Jenkinson
May 2022

2 Loading, investigating, manipulating and displaying data

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#read file and get initial view of the data
train_set = pd.read_csv('sign_mnist_train.csv')
big_test_set= pd.read_csv('sign_mnist_test.csv')
print(train_set.info())
print(big_test_set.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27455 entries, 0 to 27454
Columns: 785 entries, label to pixel784
dtypes: int64(785)
memory usage: 164.4 MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7172 entries, 0 to 7171
Columns: 785 entries, label to pixel784
dtypes: int64(785)
memory usage: 43.0 MB
None
```

[2]:
```
print(train_set.describe())
print(big_test_set.describe())
'''We see there is an outlier in the train set'''
```

```
              label        pixel1        pixel2        pixel3        pixel4  \
count  27455.000000  27455.000000  27455.000000  27455.000000  27455.000000
mean      12.325369    145.419377    148.500273    151.247714    153.546531
std        7.374907     41.358555     39.942152     39.056286     38.595247
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        6.000000    121.000000    126.000000    130.000000    133.000000
50%       13.000000    150.000000    153.000000    156.000000    158.000000
75%       19.000000    174.000000    176.000000    178.000000    179.000000
max      200.000000    255.000000    255.000000    255.000000    255.000000

             pixel5        pixel6        pixel7        pixel8        pixel9  \
count  27455.000000  27455.000000  27455.000000  27455.000000  27455.000000
mean     156.210891    158.411255    160.472154    162.339683    163.954799
std       37.111165     36.125579     35.016392     33.661998     32.651607
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%      137.000000    140.000000    142.000000    144.000000    146.000000
50%      160.000000    162.000000    164.000000    165.000000    166.000000
75%      181.000000    182.000000    183.000000    184.000000    185.000000
max      255.000000    255.000000    255.000000    255.000000    255.000000

       …      pixel775      pixel776      pixel777      pixel778  \
count  …  27455.000000  27455.000000  27455.000000  27455.000000
mean   …    141.104863    147.495611    153.325806    159.125332
std    …     63.751194     65.512894     64.427412     63.708507
min    …      0.000000      0.000000      0.000000      0.000000
25%    …     92.000000     96.000000    103.000000    112.000000
50%    …    144.000000    162.000000    172.000000    180.000000
75%    …    196.000000    202.000000    205.000000    207.000000
max    …    255.000000    255.000000    255.000000    255.000000

           pixel779      pixel780      pixel781      pixel782      pixel783  \
```

```
count  27455.000000  27455.000000  27455.000000  27455.000000  27455.000000
mean     161.969259    162.736696    162.906137    161.966454    161.137898
std       63.738316     63.444008     63.509210     63.298721     63.610415
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%      120.000000    125.000000    128.000000    128.000000    128.000000
50%      183.000000    184.000000    184.000000    182.000000    182.000000
75%      208.000000    207.000000    207.000000    206.000000    204.000000
max      255.000000    255.000000    255.000000    255.000000    255.000000

          pixel784
count  27455.000000
mean     159.824731
std       64.396846
min        0.000000
25%      125.500000
50%      182.000000
75%      204.000000
max      255.000000

[8 rows x 785 columns]
             label       pixel1       pixel2       pixel3       pixel4  \
count  7172.000000  7172.000000  7172.000000  7172.000000  7172.000000
mean     11.247351   147.532627   150.445761   153.324317   155.663413
std       7.446712    43.593144    41.867838    40.442728    39.354776
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       4.000000   122.000000   126.000000   130.000000   134.000000
50%      11.000000   154.000000   157.000000   159.000000   161.000000
75%      18.000000   178.000000   179.000000   181.000000   182.000000
max      24.000000   255.000000   255.000000   255.000000   255.000000

            pixel5       pixel6       pixel7       pixel8       pixel9  …  \
count  7172.000000  7172.000000  7172.000000  7172.000000  7172.000000  …
mean    158.169688   160.790853   162.282766   163.649191   165.589515  …
std      37.749637    36.090916    36.212636    35.885378    33.721876  …
min       0.000000    10.000000     0.000000     0.000000     0.000000  …
25%     137.000000   141.000000   144.000000   145.000000   147.000000  …
50%     163.000000   165.000000   166.000000   168.000000   169.000000  …
75%     184.000000   185.000000   186.000000   187.000000   187.000000  …
max     255.000000   255.000000   255.000000   255.000000   255.000000  …

          pixel775     pixel776     pixel777     pixel778     pixel779  \
count  7172.000000  7172.000000  7172.000000  7172.000000  7172.000000
mean    138.546570   145.539598   150.744980   155.638873   158.893196
std      64.501665    65.132370    65.760539    65.565147    65.200300
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%      90.000000    95.000000    99.000000   105.000000   113.000000
50%     137.000000   155.000000   168.000000   177.000000   181.000000
75%     195.000000   200.000000   204.250000   207.000000   207.000000
```

```
max     255.000000   255.000000   255.000000   255.000000   255.000000

          pixel780     pixel781     pixel782     pixel783     pixel784
count  7172.000000  7172.000000  7172.000000  7172.000000  7172.000000
mean    159.648494   158.162019   157.672755   156.664250   154.776771
std      65.499368    66.493576    66.009690    67.202939    68.285148
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%     113.750000   113.000000   115.000000   111.000000   106.750000
50%     182.000000   181.000000   180.000000   180.000000   179.000000
75%     208.000000   207.000000   205.000000   206.000000   204.000000
max     255.000000   255.000000   255.000000   255.000000   255.000000

[8 rows x 785 columns]
```

[2]:
```
'We see there is an outlier in the train set'
```

[3]:
```
train_set = train_set[train_set['label'] != 200]
```

[4]:
```
#graph the first serveral images in train_set to get init view of the signs

#reshape the data into 28*28 bitmap
bitmap_values = train_set.drop(['label'], axis=1).values.reshape(-1,28,28)
labels = train_set['label'].values
#print(bitmap_value)
plt.figure(figsize=(10,10))
for i in range(20):
    plt.subplot(5,4,i+1)
    plt.grid(False)
    plt.imshow(bitmap_values[i], cmap=plt.get_cmap('gray'))
    plt.xticks([])
    plt.yticks([])
    plt.xlabel(chr(labels[i]+ord('a')))

plt.show()
```

[ ]:

[5]:
```
# y_train.describe()
```

3 val test split

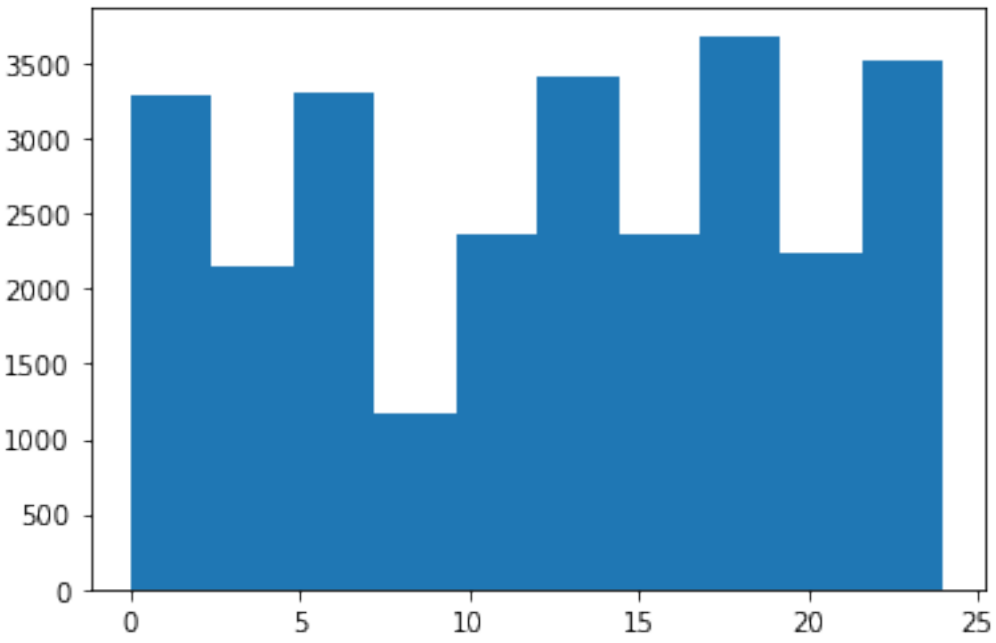
[6]:
```
from sklearn.model_selection import train_test_split
val_set, test_set = train_test_split(big_test_set, test_size=0.2,␣
↪random_state=42, stratify=big_test_set['label'])
```

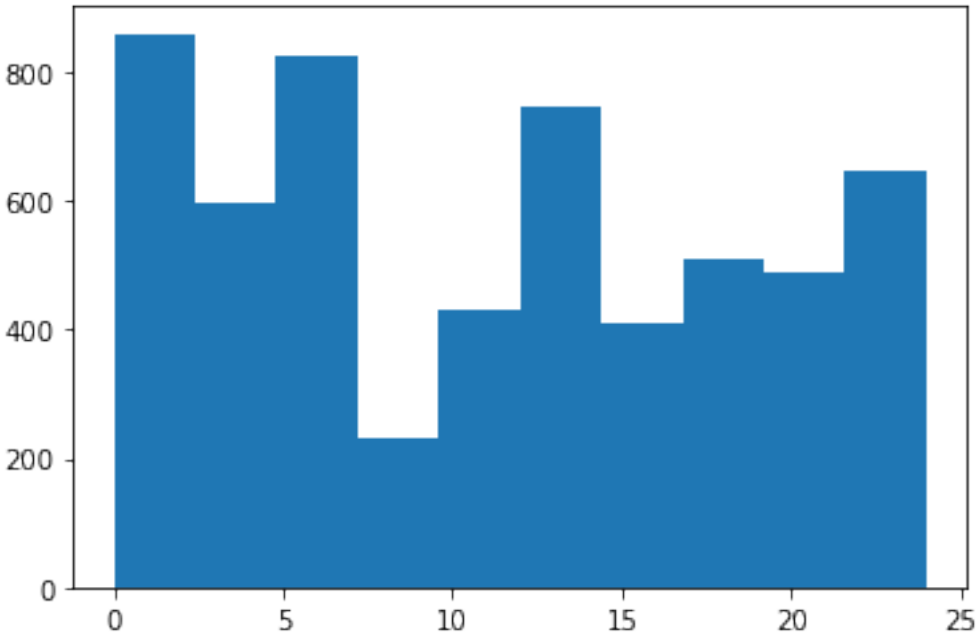
```
X_train = np.array(train_set)[:,1:]
y_train = np.array(train_set)[:,0]
X_val = np.array(val_set)[:,1:]
y_val = np.array(val_set)[:,0]
X_test = np.array(test_set)[:,1:]
y_test = np.array(test_set)[:,0]
```

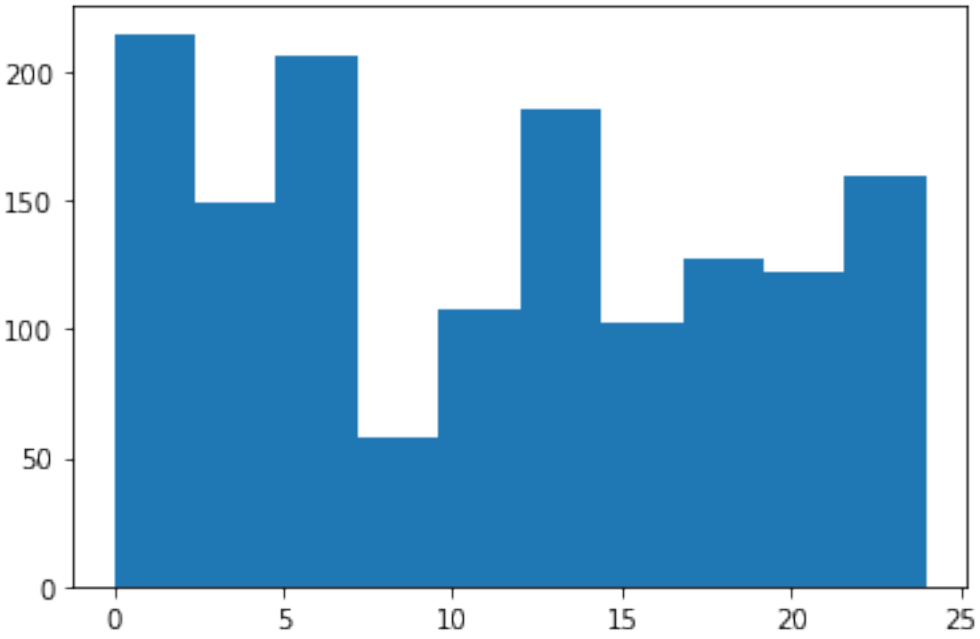
[7]:
```
print(y_val.shape)
```

```
(5737,)
```

[8]:
```
#looke at the destribution for the 3 sets
plt.hist(y_train)
plt.show()
plt.hist(y_val)
plt.show()
plt.hist(y_test)
plt.show()
```




800
600
400
200
0
0
5
10
15
20
25


200
150
100
50
0
0
5
10
15
20
25

4 Train Initial model

[9]:
```
print(X_train.shape) #we need to reshape X sets
```

```
(27454, 784)
```

[10]:
```
X_train = X_train.reshape(X_train.shape[0],28,28,1)/255.0
X_val=X_val.reshape(X_val.shape[0],28,28,1)/255.0
X_test=X_test.reshape(X_test.shape[0],28,28,1)/255.0
```

[11]:
```
print(X_train.shape) #now it's reshaped
```

```
(27454, 28, 28, 1)
```

[12]:
```
import keras
# Some key parameters
n_train = 300
n_valid = 100
hiddensizes = [16, 32, 16]
actfn = "elu"
# Optimiser and learning rate
optimizer = keras.optimizers.SGD
learningrate = 0.01
batch_size = 32
n_epochs = 10
```

4.0.1 make helper functions

[13]:
```
def plot_history(history):
    # Plot the results (shifting validation curves appropriately)
    plt.figure(figsize=(8,5))
    n = len(history.history['accuracy'])
    plt.plot(np.arange(0,n),history.history['accuracy'], color='orange')
    plt.plot(np.arange(0,n),history.history['loss'],'b')
    plt.plot(np.arange(0,n)+0.5,history.history['val_accuracy'],'r')  # offset␣
 ↪both validation curves
    plt.plot(np.arange(0,n)+0.5,history.history['val_loss'],'g')
    plt.legend(['Train Acc','Train Loss','Val Acc','Val Loss'])
    plt.grid(True)
    plt.gca().set_ylim(0, 1) # set the vertical range to [0-1]
    plt.xlabel('Epochs')
    plt.show()
```

4.0.2 CNN

[14]:
```
#i use the same model sections as workshop 10
def model_cnn_factory(hiddensizes, actfn, optimizer, learningrate=0):
    model = keras.models.Sequential()
    model.add(keras.layers.Conv2D(hiddensizes[0],(3,3), activation=actfn,␣
 ↪input_shape=(28,28,1)))
    model.add(keras.layers.MaxPooling2D(2,2))
    for n in hiddensizes[1:-1]:
        model.add(keras.layers.Conv2D(n,(3,3), activation=actfn,))
        model.add(keras.layers.MaxPooling2D(2,2))
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(256,activation=actfn))
    model.add(keras.layers.Dense(128,activation=actfn))
    model.add(keras.layers.Dense(26,activation="softmax"))
    model.compile(loss="sparse_categorical_crossentropy",␣
 ↪optimizer=optimizer(learning_rate=learningrate), metrics=["accuracy"])
    return model
```

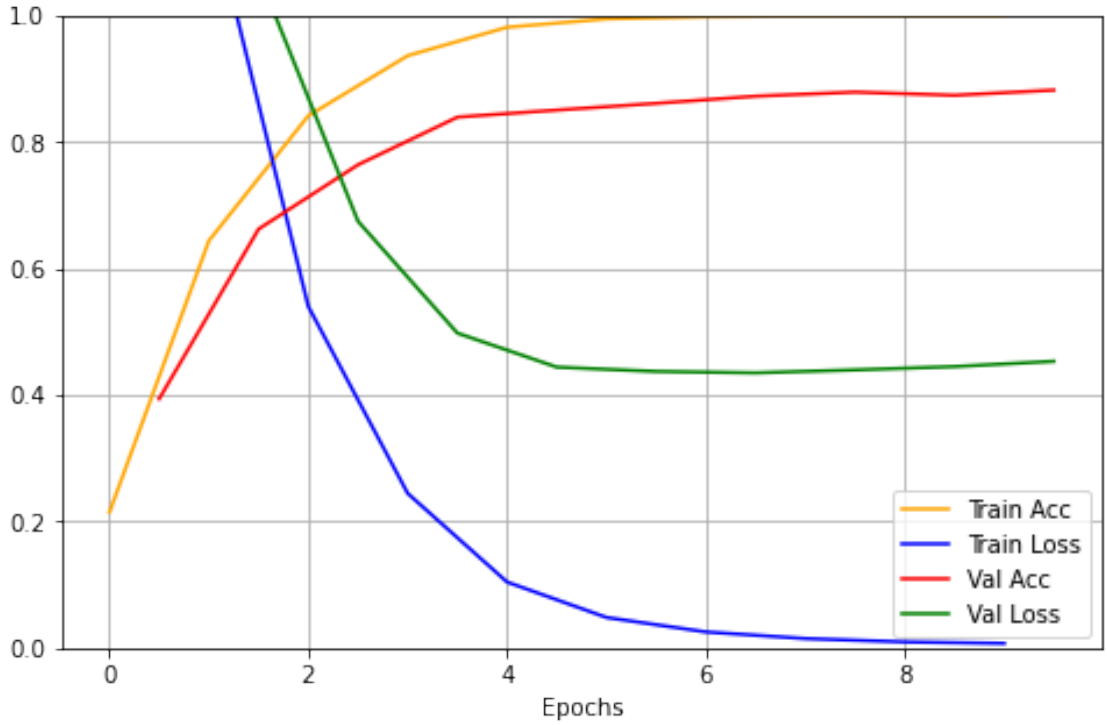
[15]:
```
def do_all_cnn(hiddensizes, actfn, optimizer, learningrate, n_train, n_valid,␣
 ↪n_epochs, batch_size, further_callbacks=[]):
    early_stopping_cb = keras.callbacks.EarlyStopping(monitor='val_loss',
                                                      patience=5,
                                                      restore_best_weights=True)
    model = model_cnn_factory(hiddensizes, actfn, optimizer, learningrate)
    callbacks = [early_stopping_cb]
    history = model.fit(X_train, y_train, epochs=n_epochs, callbacks =␣
 ↪[early_stopping_cb],
                                validation_data=(X_val, y_val))
    max_val_acc = np.max(history.history['val_accuracy'])
    return (max_val_acc, history, model)
```

[16]:
```
valacc, history, model = do_all_cnn(hiddensizes, actfn, optimizer,␣
 ↪learningrate, n_train, n_valid, n_epochs, batch_size)
# model.summary()
# model.fit(X_train, y_train, epochs=n_epochs,
#                               validation_data=(X_val, y_val))
```

```
Epoch 1/10
858/858 [==============================] - 20s 23ms/step - loss: 2.6934 -
accuracy: 0.2149 - val_loss: 1.9350 - val_accuracy: 0.3941
Epoch 2/10
858/858 [==============================] - 19s 22ms/step - loss: 1.1791 -
accuracy: 0.6441 - val_loss: 1.0654 - val_accuracy: 0.6625
Epoch 3/10
858/858 [==============================] - 19s 22ms/step - loss: 0.5401 -
accuracy: 0.8416 - val_loss: 0.6748 - val_accuracy: 0.7642
```

```
Epoch 4/10
858/858 [==============================] - 19s 22ms/step - loss: 0.2446 -
accuracy: 0.9367 - val_loss: 0.4980 - val_accuracy: 0.8395
Epoch 5/10
858/858 [==============================] - 19s 22ms/step - loss: 0.1044 -
accuracy: 0.9816 - val_loss: 0.4442 - val_accuracy: 0.8506
Epoch 6/10
858/858 [==============================] - 19s 22ms/step - loss: 0.0484 -
accuracy: 0.9947 - val_loss: 0.4374 - val_accuracy: 0.8613
Epoch 7/10
858/858 [==============================] - 19s 22ms/step - loss: 0.0255 -
accuracy: 0.9983 - val_loss: 0.4351 - val_accuracy: 0.8726
Epoch 8/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0151 -
accuracy: 0.9995 - val_loss: 0.4398 - val_accuracy: 0.8790
Epoch 9/10
858/858 [==============================] - 15s 17ms/step - loss: 0.0098 -
accuracy: 1.0000 - val_loss: 0.4451 - val_accuracy: 0.8740
Epoch 10/10
858/858 [==============================] - 18s 20ms/step - loss: 0.0075 -
accuracy: 0.9999 - val_loss: 0.4535 - val_accuracy: 0.8822
```

[17]:
```
plot_history(history)
```



[18]:
```
score = model.evaluate(X_test,y_test,verbose=0)
print("Test Loss : ", score[0])
print("Test Accuracy : ", score[1])
```

```
Test Loss :  0.4337109625339508
Test Accuracy :  0.8857142925262451
```

4.0.3 densely connected

[19]:
```
def model_dense_factory(hiddensizes, actfn, optimizer, learningrate):
    model = keras.models.Sequential()
    model.add(keras.layers.Flatten(input_shape = (28, 28, 1)))
    model.add(keras.layers.Dense(256,activation=actfn))
    model.add(keras.layers.Dense(128,activation=actfn))
    model.add(keras.layers.Dense(26,activation="softmax"))
    model.compile(loss="sparse_categorical_crossentropy",␣
 ↪optimizer=optimizer(learning_rate=learningrate), metrics=["accuracy"])
    return model
```

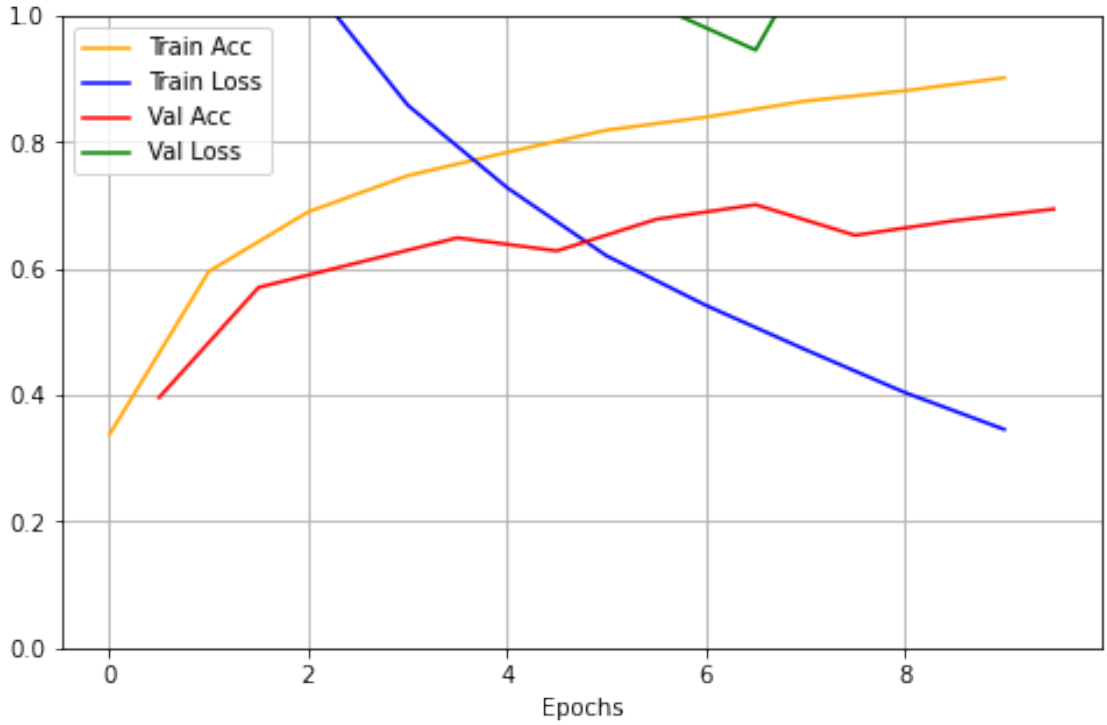
[20]:
```
def do_all_dense(hiddensizes, actfn, optimizer, learningrate, n_train, n_valid,␣
 ↪n_epochs, batch_size, further_callbacks=[]):
    early_stopping_cb = keras.callbacks.EarlyStopping(monitor='val_loss',
                                                      patience=5,
                                                      restore_best_weights=True)
    model = model_dense_factory(hiddensizes, actfn, optimizer, learningrate)
    callbacks = [early_stopping_cb]
    history = model.fit(X_train, y_train, epochs=n_epochs, callbacks =␣
 ↪[early_stopping_cb],
                                validation_data=(X_val, y_val))
    max_val_acc = np.max(history.history['val_accuracy'])
    return (max_val_acc, history, model)
```

[21]:
```
valacc, history, model = do_all_dense(hiddensizes, actfn, optimizer,␣
 ↪learningrate, n_train, n_valid, n_epochs, batch_size)
model.summary()
```

```
Epoch 1/10
858/858 [==============================] - 6s 6ms/step - loss: 2.3443 -
accuracy: 0.3378 - val_loss: 1.8782 - val_accuracy: 0.3964
Epoch 2/10
858/858 [==============================] - 5s 6ms/step - loss: 1.4068 -
accuracy: 0.5956 - val_loss: 1.4086 - val_accuracy: 0.5703
Epoch 3/10
858/858 [==============================] - 5s 6ms/step - loss: 1.0551 -
accuracy: 0.6898 - val_loss: 1.2972 - val_accuracy: 0.6094
Epoch 4/10
858/858 [==============================] - 5s 6ms/step - loss: 0.8585 -
```

```
accuracy: 0.7470 - val_loss: 1.1465 - val_accuracy: 0.6488
Epoch 5/10
858/858 [==============================] - 5s 6ms/step - loss: 0.7280 -
accuracy: 0.7838 - val_loss: 1.1496 - val_accuracy: 0.6280
Epoch 6/10
858/858 [==============================] - 5s 6ms/step - loss: 0.6203 -
accuracy: 0.8189 - val_loss: 1.0159 - val_accuracy: 0.6781
Epoch 7/10
858/858 [==============================] - 5s 6ms/step - loss: 0.5418 -
accuracy: 0.8396 - val_loss: 0.9455 - val_accuracy: 0.7011
Epoch 8/10
858/858 [==============================] - 5s 6ms/step - loss: 0.4719 -
accuracy: 0.8650 - val_loss: 1.2219 - val_accuracy: 0.6526
Epoch 9/10
858/858 [==============================] - 5s 6ms/step - loss: 0.4041 -
accuracy: 0.8812 - val_loss: 1.0402 - val_accuracy: 0.6758
Epoch 10/10
858/858 [==============================] - 5s 5ms/step - loss: 0.3458 -
accuracy: 0.9017 - val_loss: 1.0632 - val_accuracy: 0.6941
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 784)               0

 dense_3 (Dense)             (None, 256)               200960

 dense_4 (Dense)             (None, 128)               32896

 dense_5 (Dense)             (None, 26)                3354

=================================================================
Total params: 237,210
Trainable params: 237,210
Non-trainable params: 0
_________________________________________________________________
```

[22]:
```
plot_history(history)
```



[23]:
```
score = model.evaluate(X_test,y_test,verbose=0)
print("Test Loss : ", score[0])
print("Test Accuracy : ", score[1])
```

```
Test Loss :  1.1009924411773682
Test Accuracy :  0.687108039855957
```

For this two initial models, the cnn model is better than the dense model

5 Optimize models

5.0.1 CNN model

[24]:
```
max_valacc = 0

for learning_rate in [0.001,0.01]:
    for optimizer in [keras.optimizers.SGD,keras.optimizers.Nadam]:
        for acfn in ['relu','selu',"sigmoid"]:
            valacc, history, model = do_all_cnn(hiddensizes, actfn, optimizer,␣
 ↪learningrate, n_train, n_valid, n_epochs, batch_size)
            if valacc > max_valacc:
                max_valacc = valacc
                max_learning_rate = learning_rate
                max_optimizer = optimizer
```

```
                max_acfn = acfn
                best_history = history
```

```
Epoch 1/10
858/858 [==============================] - 20s 22ms/step - loss: 2.8429 -
accuracy: 0.2113 - val_loss: 2.0440 - val_accuracy: 0.3983
Epoch 2/10
858/858 [==============================] - 18s 21ms/step - loss: 1.1540 -
accuracy: 0.6693 - val_loss: 0.9003 - val_accuracy: 0.6920
Epoch 3/10
858/858 [==============================] - 18s 21ms/step - loss: 0.4679 -
accuracy: 0.8651 - val_loss: 0.6016 - val_accuracy: 0.7873
Epoch 4/10
858/858 [==============================] - 18s 21ms/step - loss: 0.2098 -
accuracy: 0.9457 - val_loss: 0.4775 - val_accuracy: 0.8429
Epoch 5/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0915 -
accuracy: 0.9844 - val_loss: 0.3983 - val_accuracy: 0.8780
Epoch 6/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0441 -
accuracy: 0.9954 - val_loss: 0.4077 - val_accuracy: 0.8790
Epoch 7/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0243 -
accuracy: 0.9989 - val_loss: 0.4209 - val_accuracy: 0.8735
Epoch 8/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0154 -
accuracy: 0.9995 - val_loss: 0.3879 - val_accuracy: 0.8893
Epoch 9/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0115 -
accuracy: 0.9997 - val_loss: 0.3813 - val_accuracy: 0.8972
Epoch 10/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0083 -
accuracy: 0.9999 - val_loss: 0.4078 - val_accuracy: 0.8914
Epoch 1/10
858/858 [==============================] - 20s 22ms/step - loss: 2.6791 -
accuracy: 0.2216 - val_loss: 1.8507 - val_accuracy: 0.4013
Epoch 2/10
858/858 [==============================] - 18s 22ms/step - loss: 1.0754 -
accuracy: 0.6756 - val_loss: 0.9136 - val_accuracy: 0.6962
Epoch 3/10
858/858 [==============================] - 19s 22ms/step - loss: 0.4340 -
accuracy: 0.8782 - val_loss: 0.5671 - val_accuracy: 0.8170
Epoch 4/10
858/858 [==============================] - 18s 22ms/step - loss: 0.1684 -
accuracy: 0.9637 - val_loss: 0.4044 - val_accuracy: 0.8740
Epoch 5/10
```

```
858/858 [==============================] - 19s 22ms/step - loss: 0.0662 -
accuracy: 0.9921 - val_loss: 0.4520 - val_accuracy: 0.8665
Epoch 6/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0306 -
accuracy: 0.9989 - val_loss: 0.4323 - val_accuracy: 0.8905
Epoch 7/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0170 -
accuracy: 0.9999 - val_loss: 0.4225 - val_accuracy: 0.9092
Epoch 8/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0109 -
accuracy: 0.9999 - val_loss: 0.4361 - val_accuracy: 0.9010
Epoch 9/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0078 -
accuracy: 1.0000 - val_loss: 0.4368 - val_accuracy: 0.9031
Epoch 1/10
858/858 [==============================] - 20s 22ms/step - loss: 2.6882 -
accuracy: 0.2189 - val_loss: 1.8961 - val_accuracy: 0.4288
Epoch 2/10
858/858 [==============================] - 18s 21ms/step - loss: 1.1524 -
accuracy: 0.6540 - val_loss: 1.0314 - val_accuracy: 0.6760
Epoch 3/10
858/858 [==============================] - 18s 21ms/step - loss: 0.5267 -
accuracy: 0.8447 - val_loss: 0.6457 - val_accuracy: 0.7900
Epoch 4/10
858/858 [==============================] - 18s 21ms/step - loss: 0.2393 -
accuracy: 0.9361 - val_loss: 0.4726 - val_accuracy: 0.8468
Epoch 5/10
858/858 [==============================] - 18s 21ms/step - loss: 0.1050 -
accuracy: 0.9799 - val_loss: 0.4076 - val_accuracy: 0.8715
Epoch 6/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0487 -
accuracy: 0.9942 - val_loss: 0.3835 - val_accuracy: 0.8925
Epoch 7/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0259 -
accuracy: 0.9983 - val_loss: 0.3894 - val_accuracy: 0.8919
Epoch 8/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0151 -
accuracy: 0.9996 - val_loss: 0.3812 - val_accuracy: 0.9062
Epoch 9/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0101 -
accuracy: 0.9999 - val_loss: 0.3889 - val_accuracy: 0.9043
Epoch 10/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0072 -
accuracy: 1.0000 - val_loss: 0.4201 - val_accuracy: 0.8954
Epoch 1/10
858/858 [==============================] - 22s 23ms/step - loss: 1.8190 -
accuracy: 0.6186 - val_loss: 0.6021 - val_accuracy: 0.8365
Epoch 2/10
```

```
858/858 [==============================] - 19s 22ms/step - loss: 42.5986 -
accuracy: 0.6568 - val_loss: 3.2376 - val_accuracy: 0.0288
Epoch 3/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2258 -
accuracy: 0.0432 - val_loss: 3.2332 - val_accuracy: 0.0608
Epoch 4/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2322 -
accuracy: 0.0414 - val_loss: 3.2265 - val_accuracy: 0.0228
Epoch 5/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2394 -
accuracy: 0.0411 - val_loss: 3.3001 - val_accuracy: 0.0200
Epoch 6/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2434 -
accuracy: 0.0428 - val_loss: 3.2293 - val_accuracy: 0.0401
Epoch 1/10
858/858 [==============================] - 21s 22ms/step - loss: 1.6511 -
accuracy: 0.6642 - val_loss: 0.5229 - val_accuracy: 0.8653
Epoch 2/10
858/858 [==============================] - 19s 22ms/step - loss: 30.4509 -
accuracy: 0.5898 - val_loss: 3.2810 - val_accuracy: 0.0343
Epoch 3/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2486 -
accuracy: 0.0419 - val_loss: 3.2615 - val_accuracy: 0.0291
Epoch 4/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2580 -
accuracy: 0.0428 - val_loss: 3.2653 - val_accuracy: 0.0371
Epoch 5/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2611 -
accuracy: 0.0439 - val_loss: 3.3055 - val_accuracy: 0.0485
Epoch 6/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2730 -
accuracy: 0.0406 - val_loss: 3.2643 - val_accuracy: 0.0343
Epoch 1/10
858/858 [==============================] - 21s 22ms/step - loss: 7.8620 -
accuracy: 0.3744 - val_loss: 3.2719 - val_accuracy: 0.0288
Epoch 2/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2558 -
accuracy: 0.0436 - val_loss: 3.2823 - val_accuracy: 0.0288
Epoch 3/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2674 -
accuracy: 0.0415 - val_loss: 3.3340 - val_accuracy: 0.0288
Epoch 4/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2762 -
accuracy: 0.0428 - val_loss: 3.3065 - val_accuracy: 0.0200
Epoch 5/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2839 -
accuracy: 0.0424 - val_loss: 3.3016 - val_accuracy: 0.0464
Epoch 6/10
```

```
858/858 [==============================] - 19s 22ms/step - loss: 3.2862 -
accuracy: 0.0432 - val_loss: 3.2578 - val_accuracy: 0.0345
Epoch 7/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2944 -
accuracy: 0.0421 - val_loss: 3.2392 - val_accuracy: 0.0371
Epoch 8/10
858/858 [==============================] - 19s 22ms/step - loss: 3.3005 -
accuracy: 0.0415 - val_loss: 3.2613 - val_accuracy: 0.0432
Epoch 9/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2989 -
accuracy: 0.0427 - val_loss: 3.4092 - val_accuracy: 0.0288
Epoch 10/10
858/858 [==============================] - 19s 22ms/step - loss: 3.3033 -
accuracy: 0.0403 - val_loss: 3.3982 - val_accuracy: 0.0345
Epoch 1/10
858/858 [==============================] - 20s 22ms/step - loss: 2.7539 -
accuracy: 0.2034 - val_loss: 1.9850 - val_accuracy: 0.3891
Epoch 2/10
858/858 [==============================] - 19s 22ms/step - loss: 1.2240 -
accuracy: 0.6343 - val_loss: 0.9688 - val_accuracy: 0.7131
Epoch 3/10
858/858 [==============================] - 18s 21ms/step - loss: 0.5166 -
accuracy: 0.8540 - val_loss: 0.6608 - val_accuracy: 0.7940
Epoch 4/10
858/858 [==============================] - 18s 21ms/step - loss: 0.2176 -
accuracy: 0.9460 - val_loss: 0.6148 - val_accuracy: 0.8036
Epoch 5/10
858/858 [==============================] - 19s 22ms/step - loss: 0.0900 -
accuracy: 0.9850 - val_loss: 0.4693 - val_accuracy: 0.8686
Epoch 6/10
858/858 [==============================] - 19s 22ms/step - loss: 0.0397 -
accuracy: 0.9975 - val_loss: 0.4542 - val_accuracy: 0.8679
Epoch 7/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0226 -
accuracy: 0.9991 - val_loss: 0.4614 - val_accuracy: 0.8801
Epoch 8/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0134 -
accuracy: 0.9999 - val_loss: 0.4706 - val_accuracy: 0.8785
Epoch 9/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0096 -
accuracy: 0.9999 - val_loss: 0.4783 - val_accuracy: 0.8790
Epoch 10/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0071 -
accuracy: 0.9999 - val_loss: 0.4997 - val_accuracy: 0.8796
Epoch 1/10
858/858 [==============================] - 20s 22ms/step - loss: 2.7031 -
accuracy: 0.2158 - val_loss: 1.8774 - val_accuracy: 0.4265
Epoch 2/10
```

```
858/858 [==============================] - 18s 21ms/step - loss: 1.1368 -
accuracy: 0.6628 - val_loss: 0.9380 - val_accuracy: 0.6958
Epoch 3/10
858/858 [==============================] - 18s 21ms/step - loss: 0.4840 -
accuracy: 0.8615 - val_loss: 0.6206 - val_accuracy: 0.7879
Epoch 4/10
858/858 [==============================] - 18s 21ms/step - loss: 0.2057 -
accuracy: 0.9498 - val_loss: 0.5518 - val_accuracy: 0.8346
Epoch 5/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0886 -
accuracy: 0.9849 - val_loss: 0.4312 - val_accuracy: 0.8715
Epoch 6/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0409 -
accuracy: 0.9964 - val_loss: 0.4242 - val_accuracy: 0.8888
Epoch 7/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0214 -
accuracy: 0.9992 - val_loss: 0.4104 - val_accuracy: 0.8938
Epoch 8/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0132 -
accuracy: 0.9999 - val_loss: 0.4235 - val_accuracy: 0.9036
Epoch 9/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0092 -
accuracy: 1.0000 - val_loss: 0.4276 - val_accuracy: 0.9012
Epoch 10/10
858/858 [==============================] - 18s 21ms/step - loss: 0.0069 -
accuracy: 1.0000 - val_loss: 0.4452 - val_accuracy: 0.8944
Epoch 1/10
858/858 [==============================] - 21s 23ms/step - loss: 2.7953 -
accuracy: 0.2036 - val_loss: 1.9229 - val_accuracy: 0.4236
Epoch 2/10
858/858 [==============================] - 19s 23ms/step - loss: 1.1487 -
accuracy: 0.6603 - val_loss: 0.9537 - val_accuracy: 0.6962
Epoch 3/10
858/858 [==============================] - 19s 22ms/step - loss: 0.4460 -
accuracy: 0.8738 - val_loss: 0.5276 - val_accuracy: 0.8438
Epoch 4/10
858/858 [==============================] - 19s 22ms/step - loss: 0.1681 -
accuracy: 0.9642 - val_loss: 0.3794 - val_accuracy: 0.8886
Epoch 5/10
858/858 [==============================] - 19s 23ms/step - loss: 0.0634 -
accuracy: 0.9932 - val_loss: 0.3452 - val_accuracy: 0.8991
Epoch 6/10
858/858 [==============================] - 19s 23ms/step - loss: 0.0280 -
accuracy: 0.9993 - val_loss: 0.3450 - val_accuracy: 0.9059
Epoch 7/10
858/858 [==============================] - 19s 23ms/step - loss: 0.0157 -
accuracy: 1.0000 - val_loss: 0.3521 - val_accuracy: 0.9111
Epoch 8/10
```

```
858/858 [==============================] - 19s 23ms/step - loss: 0.0102 -
accuracy: 1.0000 - val_loss: 0.3730 - val_accuracy: 0.9156
Epoch 9/10
858/858 [==============================] - 19s 23ms/step - loss: 0.0073 -
accuracy: 1.0000 - val_loss: 0.3773 - val_accuracy: 0.9144
Epoch 10/10
858/858 [==============================] - 19s 23ms/step - loss: 0.0056 -
accuracy: 1.0000 - val_loss: 0.3837 - val_accuracy: 0.9148
Epoch 1/10
858/858 [==============================] - 22s 23ms/step - loss: 1.4988 -
accuracy: 0.6762 - val_loss: 0.4587 - val_accuracy: 0.8524
Epoch 2/10
858/858 [==============================] - 20s 23ms/step - loss: 37.0937 -
accuracy: 0.3257 - val_loss: 3.2860 - val_accuracy: 0.0291
Epoch 3/10
858/858 [==============================] - 19s 23ms/step - loss: 3.2317 -
accuracy: 0.0444 - val_loss: 3.2569 - val_accuracy: 0.0345
Epoch 4/10
858/858 [==============================] - 19s 23ms/step - loss: 3.2418 -
accuracy: 0.0421 - val_loss: 3.3070 - val_accuracy: 0.0432
Epoch 5/10
858/858 [==============================] - 19s 23ms/step - loss: 3.2517 -
accuracy: 0.0440 - val_loss: 3.2575 - val_accuracy: 0.0464
Epoch 6/10
858/858 [==============================] - 19s 23ms/step - loss: 3.2596 -
accuracy: 0.0421 - val_loss: 3.3417 - val_accuracy: 0.0200
Epoch 1/10
858/858 [==============================] - 22s 23ms/step - loss: 20.4025 -
accuracy: 0.3280 - val_loss: 3.2426 - val_accuracy: 0.0345
Epoch 2/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2232 -
accuracy: 0.0431 - val_loss: 3.2574 - val_accuracy: 0.0373
Epoch 3/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2342 -
accuracy: 0.0437 - val_loss: 3.2426 - val_accuracy: 0.0288
Epoch 4/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2420 -
accuracy: 0.0402 - val_loss: 3.2535 - val_accuracy: 0.0401
Epoch 5/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2548 -
accuracy: 0.0413 - val_loss: 3.2861 - val_accuracy: 0.0462
Epoch 6/10
858/858 [==============================] - 19s 22ms/step - loss: 3.2634 -
accuracy: 0.0416 - val_loss: 3.2873 - val_accuracy: 0.0200
Epoch 1/10
858/858 [==============================] - 22s 23ms/step - loss: 1.0873 -
accuracy: 0.7998 - val_loss: 0.4569 - val_accuracy: 0.9078
Epoch 2/10
```

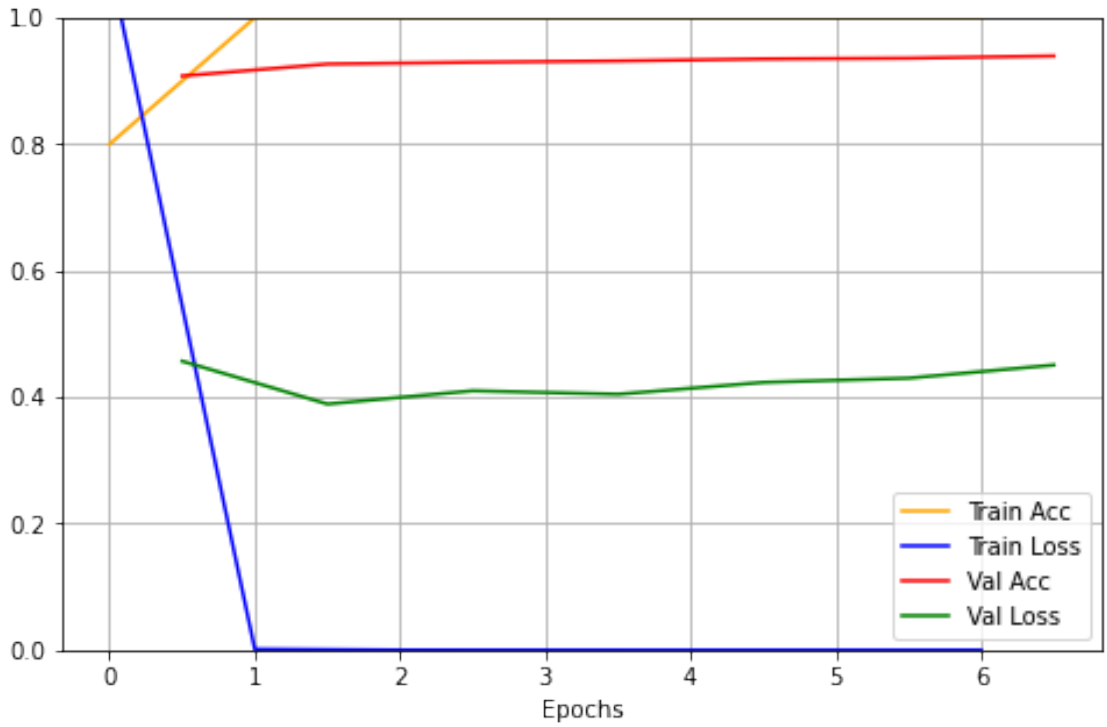
```
858/858 [==============================] - 19s 22ms/step - loss: 0.0015 -
accuracy: 0.9998 - val_loss: 0.3891 - val_accuracy: 0.9266
Epoch 3/10
858/858 [==============================] - 19s 22ms/step - loss: 1.4509e-04 -
accuracy: 1.0000 - val_loss: 0.4100 - val_accuracy: 0.9294
Epoch 4/10
858/858 [==============================] - 19s 22ms/step - loss: 7.0091e-05 -
accuracy: 1.0000 - val_loss: 0.4045 - val_accuracy: 0.9315
Epoch 5/10
858/858 [==============================] - 19s 22ms/step - loss: 3.8440e-05 -
accuracy: 1.0000 - val_loss: 0.4233 - val_accuracy: 0.9346
Epoch 6/10
858/858 [==============================] - 19s 22ms/step - loss: 2.2324e-05 -
accuracy: 1.0000 - val_loss: 0.4300 - val_accuracy: 0.9359
Epoch 7/10
858/858 [==============================] - 19s 22ms/step - loss: 1.3070e-05 -
accuracy: 1.0000 - val_loss: 0.4507 - val_accuracy: 0.9393
```

[ ]:

[25]:
```
print(f'the max val accurency after trained is {max_valacc} with the model:␣
↪max_learning_rate: {max_learning_rate}, max_optimizer: {max_optimizer},␣
↪max_acfn: {max_acfn}')
```

```
the max val accurency after trained is 0.939341127872467 with the model:
max_learning_rate: 0.01, max_optimizer: <class
'keras.optimizers.optimizer_v2.nadam.Nadam'>, max_acfn: sigmoid
```

[26]:
```
#look at the graph
plot_history(best_history)
```



5.0.2 dense model

[27]:
```
max_valacc = 0

for learning_rate in [0.001,0.01]:
    for optimizer in [keras.optimizers.SGD,keras.optimizers.Nadam]:
        for acfn in ['relu','selu']:
            valacc, history, model = do_all_dense(hiddensizes, actfn,␣
 ↪optimizer, learningrate, n_train, n_valid, n_epochs, batch_size)
            if valacc > max_valacc:
                max_valacc = valacc
                max_learning_rate = learning_rate
                max_optimizer = optimizer
                max_acfn = acfn
                best_history = history
```

```
Epoch 1/10
858/858 [==============================] - 6s 6ms/step - loss: 2.3733 -
accuracy: 0.3364 - val_loss: 1.8917 - val_accuracy: 0.4060
Epoch 2/10
858/858 [==============================] - 5s 6ms/step - loss: 1.4246 -
accuracy: 0.5849 - val_loss: 1.4399 - val_accuracy: 0.5510
Epoch 3/10
```

```
858/858 [==============================] - 5s 6ms/step - loss: 1.0733 -
accuracy: 0.6810 - val_loss: 1.2838 - val_accuracy: 0.5937
Epoch 4/10
858/858 [==============================] - 5s 6ms/step - loss: 0.8766 -
accuracy: 0.7394 - val_loss: 1.2215 - val_accuracy: 0.6244
Epoch 5/10
858/858 [==============================] - 5s 6ms/step - loss: 0.7460 -
accuracy: 0.7758 - val_loss: 1.0498 - val_accuracy: 0.6751
Epoch 6/10
858/858 [==============================] - 5s 6ms/step - loss: 0.6457 -
accuracy: 0.8095 - val_loss: 1.0707 - val_accuracy: 0.6775
Epoch 7/10
858/858 [==============================] - 5s 6ms/step - loss: 0.5581 -
accuracy: 0.8381 - val_loss: 1.0237 - val_accuracy: 0.6821
Epoch 8/10
858/858 [==============================] - 5s 6ms/step - loss: 0.4879 -
accuracy: 0.8549 - val_loss: 1.0119 - val_accuracy: 0.6869
Epoch 9/10
858/858 [==============================] - 5s 6ms/step - loss: 0.4237 -
accuracy: 0.8774 - val_loss: 1.0470 - val_accuracy: 0.6763
Epoch 10/10
858/858 [==============================] - 5s 6ms/step - loss: 0.3771 -
accuracy: 0.8922 - val_loss: 0.9626 - val_accuracy: 0.7126
Epoch 1/10
858/858 [==============================] - 6s 6ms/step - loss: 2.3223 -
accuracy: 0.3429 - val_loss: 1.7543 - val_accuracy: 0.4882
Epoch 2/10
858/858 [==============================] - 5s 6ms/step - loss: 1.3935 -
accuracy: 0.5942 - val_loss: 1.4812 - val_accuracy: 0.5487
Epoch 3/10
858/858 [==============================] - 5s 6ms/step - loss: 1.0588 -
accuracy: 0.6873 - val_loss: 1.2490 - val_accuracy: 0.6076
Epoch 4/10
858/858 [==============================] - 5s 6ms/step - loss: 0.8694 -
accuracy: 0.7389 - val_loss: 1.1723 - val_accuracy: 0.6390
Epoch 5/10
858/858 [==============================] - 5s 6ms/step - loss: 0.7351 -
accuracy: 0.7805 - val_loss: 1.2099 - val_accuracy: 0.6312
Epoch 6/10
858/858 [==============================] - 5s 6ms/step - loss: 0.6413 -
accuracy: 0.8084 - val_loss: 1.1005 - val_accuracy: 0.6744
Epoch 7/10
858/858 [==============================] - 5s 6ms/step - loss: 0.5632 -
accuracy: 0.8334 - val_loss: 0.9951 - val_accuracy: 0.6861
Epoch 8/10
858/858 [==============================] - 5s 6ms/step - loss: 0.4836 -
accuracy: 0.8583 - val_loss: 1.1905 - val_accuracy: 0.6362
Epoch 9/10
```

```
858/858 [==============================] - 5s 6ms/step - loss: 0.4288 -
accuracy: 0.8772 - val_loss: 1.0896 - val_accuracy: 0.6672
Epoch 10/10
858/858 [==============================] - 5s 6ms/step - loss: 0.3743 -
accuracy: 0.8907 - val_loss: 1.1148 - val_accuracy: 0.6979
Epoch 1/10
858/858 [==============================] - 8s 7ms/step - loss: 3.2305 -
accuracy: 0.0859 - val_loss: 2.9389 - val_accuracy: 0.1252
Epoch 2/10
858/858 [==============================] - 6s 7ms/step - loss: 2.6120 -
accuracy: 0.1661 - val_loss: 2.4028 - val_accuracy: 0.2189
Epoch 3/10
858/858 [==============================] - 6s 7ms/step - loss: 2.1757 -
accuracy: 0.2573 - val_loss: 2.4049 - val_accuracy: 0.2428
Epoch 4/10
858/858 [==============================] - 6s 7ms/step - loss: 1.9466 -
accuracy: 0.3163 - val_loss: 2.5532 - val_accuracy: 0.2393
Epoch 5/10
858/858 [==============================] - 6s 7ms/step - loss: 1.8377 -
accuracy: 0.3502 - val_loss: 1.9913 - val_accuracy: 0.3380
Epoch 6/10
858/858 [==============================] - 6s 7ms/step - loss: 1.7627 -
accuracy: 0.3671 - val_loss: 2.0571 - val_accuracy: 0.3221
Epoch 7/10
858/858 [==============================] - 6s 7ms/step - loss: 1.6876 -
accuracy: 0.3907 - val_loss: 2.1386 - val_accuracy: 0.3122
Epoch 8/10
858/858 [==============================] - 6s 7ms/step - loss: 1.6255 -
accuracy: 0.4105 - val_loss: 2.2301 - val_accuracy: 0.3598
Epoch 9/10
858/858 [==============================] - 6s 7ms/step - loss: 1.5701 -
accuracy: 0.4299 - val_loss: 2.3513 - val_accuracy: 0.3272
Epoch 10/10
858/858 [==============================] - 6s 7ms/step - loss: 1.5005 -
accuracy: 0.4526 - val_loss: 2.4976 - val_accuracy: 0.3404
Epoch 1/10
858/858 [==============================] - 10s 9ms/step - loss: 3.4904 -
accuracy: 0.0485 - val_loss: 3.2543 - val_accuracy: 0.0295
Epoch 2/10
858/858 [==============================] - 7s 8ms/step - loss: 3.1406 -
accuracy: 0.0816 - val_loss: 2.9281 - val_accuracy: 0.1018
Epoch 3/10
858/858 [==============================] - 7s 8ms/step - loss: 2.7703 -
accuracy: 0.1333 - val_loss: 2.9384 - val_accuracy: 0.0894
Epoch 4/10
858/858 [==============================] - 7s 8ms/step - loss: 2.7985 -
accuracy: 0.1293 - val_loss: 2.7298 - val_accuracy: 0.1123
Epoch 5/10
```

```
858/858 [==============================] - 7s 8ms/step - loss: 2.6555 -
accuracy: 0.1309 - val_loss: 2.7805 - val_accuracy: 0.1189
Epoch 6/10
858/858 [==============================] - 7s 8ms/step - loss: 2.9782 -
accuracy: 0.1403 - val_loss: 2.9546 - val_accuracy: 0.0957
Epoch 7/10
858/858 [==============================] - 7s 8ms/step - loss: 2.5867 -
accuracy: 0.1422 - val_loss: 2.8276 - val_accuracy: 0.1191
Epoch 8/10
858/858 [==============================] - 7s 8ms/step - loss: 2.5219 -
accuracy: 0.1528 - val_loss: 2.7718 - val_accuracy: 0.0952
Epoch 9/10
858/858 [==============================] - 7s 8ms/step - loss: 2.4872 -
accuracy: 0.1610 - val_loss: 2.6901 - val_accuracy: 0.1257
Epoch 10/10
858/858 [==============================] - 7s 8ms/step - loss: 2.4250 -
accuracy: 0.1675 - val_loss: 2.6326 - val_accuracy: 0.1706
Epoch 1/10
858/858 [==============================] - 6s 7ms/step - loss: 2.3654 -
accuracy: 0.3386 - val_loss: 1.8612 - val_accuracy: 0.5086
Epoch 2/10
858/858 [==============================] - 5s 6ms/step - loss: 1.4186 -
accuracy: 0.5890 - val_loss: 1.4042 - val_accuracy: 0.5874
Epoch 3/10
858/858 [==============================] - 5s 6ms/step - loss: 1.0740 -
accuracy: 0.6795 - val_loss: 1.2317 - val_accuracy: 0.6249
Epoch 4/10
858/858 [==============================] - 5s 6ms/step - loss: 0.8754 -
accuracy: 0.7365 - val_loss: 1.2186 - val_accuracy: 0.6392
Epoch 5/10
858/858 [==============================] - 5s 6ms/step - loss: 0.7485 -
accuracy: 0.7740 - val_loss: 1.0979 - val_accuracy: 0.6669
Epoch 6/10
858/858 [==============================] - 5s 6ms/step - loss: 0.6517 -
accuracy: 0.8050 - val_loss: 1.0495 - val_accuracy: 0.6727
Epoch 7/10
858/858 [==============================] - 5s 6ms/step - loss: 0.5779 -
accuracy: 0.8279 - val_loss: 0.9846 - val_accuracy: 0.6958
Epoch 8/10
858/858 [==============================] - 5s 6ms/step - loss: 0.4920 -
accuracy: 0.8538 - val_loss: 1.1758 - val_accuracy: 0.6416
Epoch 9/10
858/858 [==============================] - 5s 6ms/step - loss: 0.4321 -
accuracy: 0.8748 - val_loss: 0.9649 - val_accuracy: 0.6962
Epoch 10/10
858/858 [==============================] - 5s 6ms/step - loss: 0.3814 -
accuracy: 0.8877 - val_loss: 1.0853 - val_accuracy: 0.6871
Epoch 1/10
```

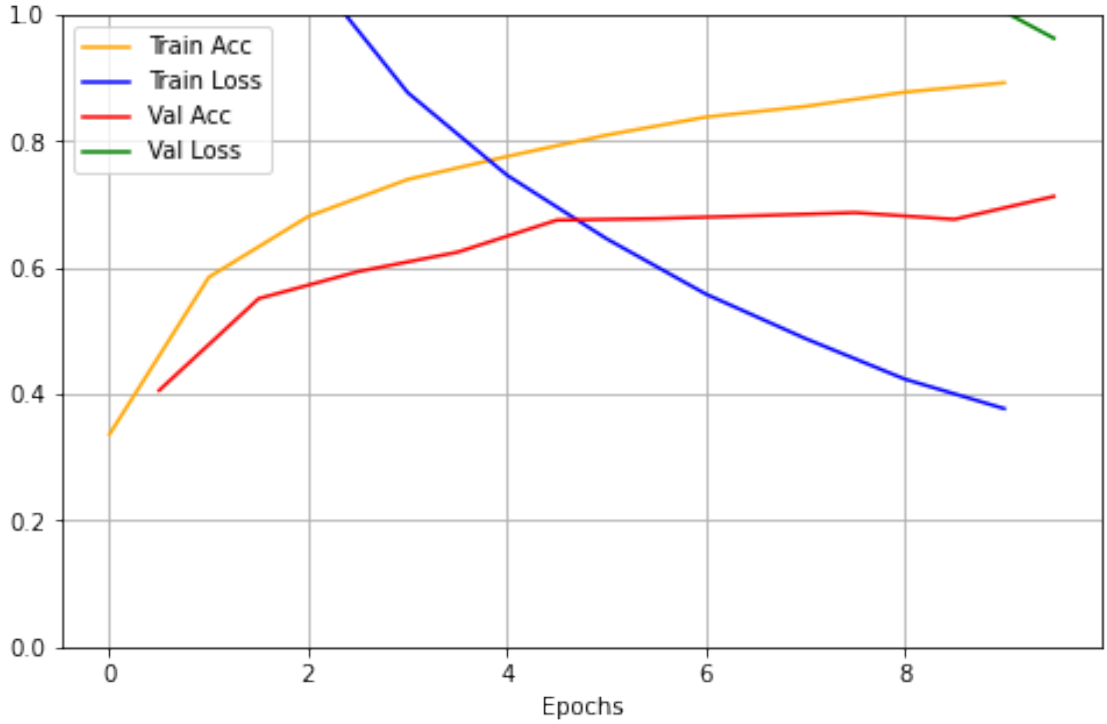
```
858/858 [==============================] - 6s 7ms/step - loss: 2.3576 -
accuracy: 0.3332 - val_loss: 1.8315 - val_accuracy: 0.4657
Epoch 2/10
858/858 [==============================] - 5s 6ms/step - loss: 1.4219 -
accuracy: 0.5846 - val_loss: 1.4032 - val_accuracy: 0.5648
Epoch 3/10
858/858 [==============================] - 5s 6ms/step - loss: 1.0786 -
accuracy: 0.6776 - val_loss: 1.2894 - val_accuracy: 0.5989
Epoch 4/10
858/858 [==============================] - 5s 6ms/step - loss: 0.8912 -
accuracy: 0.7313 - val_loss: 1.1602 - val_accuracy: 0.6390
Epoch 5/10
858/858 [==============================] - 5s 6ms/step - loss: 0.7610 -
accuracy: 0.7707 - val_loss: 1.1532 - val_accuracy: 0.6402
Epoch 6/10
858/858 [==============================] - 5s 6ms/step - loss: 0.6578 -
accuracy: 0.8040 - val_loss: 1.1642 - val_accuracy: 0.6512
Epoch 7/10
858/858 [==============================] - 5s 6ms/step - loss: 0.5699 -
accuracy: 0.8312 - val_loss: 1.0858 - val_accuracy: 0.6824
Epoch 8/10
858/858 [==============================] - 5s 6ms/step - loss: 0.4930 -
accuracy: 0.8526 - val_loss: 1.0498 - val_accuracy: 0.6847
Epoch 9/10
858/858 [==============================] - 5s 6ms/step - loss: 0.4417 -
accuracy: 0.8702 - val_loss: 1.0330 - val_accuracy: 0.6744
Epoch 10/10
858/858 [==============================] - 5s 6ms/step - loss: 0.3740 -
accuracy: 0.8924 - val_loss: 1.0108 - val_accuracy: 0.6953
Epoch 1/10
858/858 [==============================] - 8s 8ms/step - loss: 3.0879 -
accuracy: 0.1073 - val_loss: 2.8182 - val_accuracy: 0.1604
Epoch 2/10
858/858 [==============================] - 7s 8ms/step - loss: 2.2208 -
accuracy: 0.2569 - val_loss: 1.9907 - val_accuracy: 0.3504
Epoch 3/10
858/858 [==============================] - 7s 8ms/step - loss: 1.8192 -
accuracy: 0.3798 - val_loss: 1.8315 - val_accuracy: 0.4096
Epoch 4/10
858/858 [==============================] - 7s 8ms/step - loss: 1.5343 -
accuracy: 0.4737 - val_loss: 1.6423 - val_accuracy: 0.4860
Epoch 5/10
858/858 [==============================] - 7s 8ms/step - loss: 1.2941 -
accuracy: 0.5558 - val_loss: 2.5772 - val_accuracy: 0.4183
Epoch 6/10
858/858 [==============================] - 7s 8ms/step - loss: 1.1271 -
accuracy: 0.6106 - val_loss: 1.8590 - val_accuracy: 0.5398
Epoch 7/10
```

```
858/858 [==============================] - 7s 8ms/step - loss: 1.0228 -
accuracy: 0.6442 - val_loss: 1.9805 - val_accuracy: 0.5055
Epoch 8/10
858/858 [==============================] - 7s 8ms/step - loss: 0.8847 -
accuracy: 0.6885 - val_loss: 2.3012 - val_accuracy: 0.5604
Epoch 9/10
858/858 [==============================] - 7s 8ms/step - loss: 0.8344 -
accuracy: 0.7059 - val_loss: 2.4211 - val_accuracy: 0.4915
Epoch 1/10
858/858 [==============================] - 8s 8ms/step - loss: 3.2025 -
accuracy: 0.0929 - val_loss: 2.9993 - val_accuracy: 0.1419
Epoch 2/10
858/858 [==============================] - 6s 8ms/step - loss: 2.5071 -
accuracy: 0.1975 - val_loss: 2.0468 - val_accuracy: 0.3404
Epoch 3/10
858/858 [==============================] - 6s 7ms/step - loss: 1.9159 -
accuracy: 0.3486 - val_loss: 2.8648 - val_accuracy: 0.3045
Epoch 4/10
858/858 [==============================] - 6s 7ms/step - loss: 1.6594 -
accuracy: 0.4298 - val_loss: 2.2922 - val_accuracy: 0.3763
Epoch 5/10
858/858 [==============================] - 6s 7ms/step - loss: 1.4703 -
accuracy: 0.4883 - val_loss: 1.7940 - val_accuracy: 0.4678
Epoch 6/10
858/858 [==============================] - 6s 7ms/step - loss: 1.2866 -
accuracy: 0.5449 - val_loss: 2.3302 - val_accuracy: 0.4279
Epoch 7/10
858/858 [==============================] - 6s 7ms/step - loss: 1.1674 -
accuracy: 0.5824 - val_loss: 1.9035 - val_accuracy: 0.4746
Epoch 8/10
858/858 [==============================] - 6s 8ms/step - loss: 1.0934 -
accuracy: 0.6061 - val_loss: 2.1639 - val_accuracy: 0.4835
Epoch 9/10
858/858 [==============================] - 6s 7ms/step - loss: 1.0175 -
accuracy: 0.6327 - val_loss: 2.3849 - val_accuracy: 0.4480
Epoch 10/10
858/858 [==============================] - 7s 8ms/step - loss: 0.9376 -
accuracy: 0.6595 - val_loss: 2.8764 - val_accuracy: 0.4248
```

[28]:
```
print(f'the max val accurency after trained is {max_valacc} with the model:␣
↪max_learning_rate: {max_learning_rate}, max_optimizer: {max_optimizer},␣
↪max_acfn: {max_acfn}')
```

```
the max val accurency after trained is 0.7125675678253174 with the model:
max_learning_rate: 0.001, max_optimizer: <class
'keras.optimizers.optimizer_v2.gradient_descent.SGD'>, max_acfn: relu
```

[29]:
```
#look at the graph
plot_history(best_history)
```



6 Comparison of models, selection

6.1 for cnn

[30]:
```
# we use the test set to see the performance
actfn = 'selu'
optimizer = keras.optimizers.Nadam
learningrate = 0.001
valacc, history, model = do_all_cnn(hiddensizes, actfn, optimizer,␣
 ↪learningrate, n_train, n_valid, n_epochs, batch_size)
```

```
Epoch 1/10
858/858 [==============================] - 23s 24ms/step - loss: 0.3503 -
accuracy: 0.9059 - val_loss: 0.3516 - val_accuracy: 0.8963
Epoch 2/10
858/858 [==============================] - 20s 23ms/step - loss: 0.0033 -
accuracy: 1.0000 - val_loss: 0.3882 - val_accuracy: 0.9078
Epoch 3/10
858/858 [==============================] - 20s 23ms/step - loss: 0.2465 -
accuracy: 0.9757 - val_loss: 0.4697 - val_accuracy: 0.8883
```

```
Epoch 4/10
858/858 [==============================] - 20s 23ms/step - loss: 0.0024 -
accuracy: 1.0000 - val_loss: 0.4687 - val_accuracy: 0.8975
Epoch 5/10
858/858 [==============================] - 20s 23ms/step - loss: 0.0010 -
accuracy: 1.0000 - val_loss: 0.4941 - val_accuracy: 0.8940
Epoch 6/10
858/858 [==============================] - 20s 23ms/step - loss: 5.4830e-04 -
accuracy: 1.0000 - val_loss: 0.4834 - val_accuracy: 0.9008
```

[31]:
```
print(f'the max validation accurrency for cnn is {valacc}')
```

```
the max validation accurrency for cnn is 0.9077915549278259
```

[32]:
```
score = model.evaluate(X_test,y_test,verbose=0)
print("Test Loss : ", score[0])
print("Test Accuracy : ", score[1])
```

```
Test Loss :  0.3926527798175812
Test Accuracy :  0.889895498752594
```

6.1.1 for dense

[33]:
```
# we use the test set to see the performance
actfn = 'selu'
optimizer = keras.optimizers.SGD
learningrate = 0.01
valacc, history, model = do_all_dense(hiddensizes, actfn, optimizer,␣
 ↪learningrate, n_train, n_valid, n_epochs, batch_size)
```

```
Epoch 1/10
858/858 [==============================] - 6s 6ms/step - loss: 2.0139 -
accuracy: 0.4039 - val_loss: 1.5963 - val_accuracy: 0.5175
Epoch 2/10
858/858 [==============================] - 5s 6ms/step - loss: 1.1478 -
accuracy: 0.6533 - val_loss: 1.7335 - val_accuracy: 0.5093
Epoch 3/10
858/858 [==============================] - 5s 6ms/step - loss: 0.8749 -
accuracy: 0.7383 - val_loss: 1.1639 - val_accuracy: 0.6347
Epoch 4/10
858/858 [==============================] - 5s 6ms/step - loss: 0.7227 -
accuracy: 0.7808 - val_loss: 1.1705 - val_accuracy: 0.6329
Epoch 5/10
858/858 [==============================] - 5s 6ms/step - loss: 0.5882 -
accuracy: 0.8250 - val_loss: 1.0464 - val_accuracy: 0.6821
Epoch 6/10
858/858 [==============================] - 5s 6ms/step - loss: 0.4888 -
accuracy: 0.8585 - val_loss: 1.0748 - val_accuracy: 0.7056
```

```
Epoch 7/10
858/858 [==============================] - 5s 6ms/step - loss: 0.4057 -
accuracy: 0.8825 - val_loss: 1.0408 - val_accuracy: 0.6986
Epoch 8/10
858/858 [==============================] - 5s 6ms/step - loss: 0.3476 -
accuracy: 0.9030 - val_loss: 0.9881 - val_accuracy: 0.7162
Epoch 9/10
858/858 [==============================] - 5s 6ms/step - loss: 0.2831 -
accuracy: 0.9239 - val_loss: 0.9418 - val_accuracy: 0.7235
Epoch 10/10
858/858 [==============================] - 5s 6ms/step - loss: 0.2347 -
accuracy: 0.9398 - val_loss: 0.9903 - val_accuracy: 0.7256
```

[34]:
```
print(f'the max validation accurrency for dense is {valacc}')
```

```
the max validation accurrency for dense is 0.725640594959259
```

[35]:
```
score = model.evaluate(X_test,y_test,verbose=0)
print("Test Loss : ", score[0])
print("Test Accuracy : ", score[1])
```

```
Test Loss :  1.0389772653579712
Test Accuracy :  0.7114982604980469
```

[36]:
```
'''the cnn model with  actfn = 'selu', optimizer =keras.optimizers.Nadam,␣
↪learningrate =0.01 tested to be the best model'''
```

[36]:
```
"the cnn model with  actfn = 'selu', optimizer =keras.optimizers.Nadam,
learningrate =0.01 tested to be the best model"
```

[ ]:
```

```