In [1]:
```
# Python ≥3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn ≥0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# Common imports
import numpy as np
import os, time
import pandas as pd

# Deep Learning imports
import tensorflow as tf
from tensorflow import keras

# To plot nice figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
mpl.rc('figure', dpi=100)
import seaborn as sns; sns.set()

# Check the versions are OK (both should be 2 or more)
print(tf.__version__)

# Set seed for reproducibility
from numpy.random import seed
seed(1)
from tensorflow import random
random.set_seed(1)
```

```
2.13.0
```

1. Load, investigate, manipulate and display the data

In [2]:
```
# Load the training data
train_set = pd.read_csv("/Users/tienle/Documents/STUDY_AUS/Trimester 2_2023/Using ML Tools/Assignment/Assignment3/sign_

print('The shape of training data is:', train_set.shape)
```

```
The shape of training data is: (27455, 785)
```

In [3]:
```
print("Some statistic values of training data columns: \n")
train_set.describe()
```

```
Some statistic values of training data columns:
```

Out[3]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	
count	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000	...	27455
mean	12.325369	145.419377	148.500273	151.247714	153.546531	156.210891	158.411255	160.472154	162.339683	163.954799	...	141
std	7.374907	41.358555	39.942152	39.056286	38.595247	37.111165	36.125579	35.016392	33.661998	32.651607	...	63
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0
25%	6.000000	121.000000	126.000000	130.000000	133.000000	137.000000	140.000000	142.000000	144.000000	146.000000	...	92
50%	13.000000	150.000000	153.000000	156.000000	158.000000	160.000000	162.000000	164.000000	165.000000	166.000000	...	144
75%	19.000000	174.000000	176.000000	178.000000	179.000000	181.000000	182.000000	183.000000	184.000000	185.000000	...	196
max	200.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	...	255

8 rows × 785 columns

In [4]:
```
print('The first 6 rows of training data: \n')
train_set.head()
```

```
The first 6 rows of training data:
```

Out[4]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixe
0	3	107	118	127	134	139	143	146	150	153	...	207	207	207	207	206	206	206	204	
1	6	155	157	156	156	156	157	156	158	158	...	69	149	128	87	94	163	175	103	
2	2	187	188	188	187	187	186	187	188	187	...	202	201	200	199	198	199	198	195	
3	2	211	211	212	212	211	210	211	210	210	...	235	234	233	231	230	226	225	222	
4	13	164	167	170	172	176	179	180	184	185	...	92	105	105	108	133	163	157	163	

5 rows × 785 columns

In [5]:
```
# Load the testing data
test_set = pd.read_csv("/Users/tienle/Documents/STUDY_AUS/Trimester 2_2023/Using ML Tools/Assignment/Assignment3/sign_m

print('The shape of testing data is:', test_set.shape)
print("Some statistic values of testing data columns: \n")
test_set.describe()
```

```
The shape of testing data is: (7172, 785)
Some statistic values of testing data columns:
```

Out[5]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	7
count	7172.000000	7172.000000	7172.000000	7172.000000	7172.000000	7172.000000	7172.000000	7172.000000	7172.000000	7172.000000	...	7172.000000	71
mean	11.247351	147.532627	150.445761	153.324317	155.663413	158.169688	160.790853	162.282766	163.649191	165.589515	...	138.546570	14
std	7.446712	43.593144	41.867838	40.442728	39.354776	37.749637	36.090916	36.212636	35.885378	33.721876	...	64.501665	6
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	10.000000	0.000000	0.000000	0.000000	...	0.000000	
25%	4.000000	122.000000	126.000000	130.000000	134.000000	137.000000	141.000000	144.000000	145.000000	147.000000	...	90.000000	9
50%	11.000000	154.000000	157.000000	159.000000	161.000000	163.000000	165.000000	166.000000	168.000000	169.000000	...	137.000000	15
75%	18.000000	178.000000	179.000000	181.000000	182.000000	184.000000	185.000000	186.000000	187.000000	187.000000	...	195.000000	2(
max	24.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	...	255.000000	25

8 rows × 785 columns

In [6]:
```
print('The first 6 rows of testing data: \n')
test_set.head()
```

```
The first 6 rows of testing data:
```

Out[6]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixe
0	6	149	149	150	150	150	151	151	150	151	...	138	148	127	89	82	96	106	112	
1	5	126	128	131	132	133	134	135	135	136	...	47	104	194	183	186	184	184	184	
2	10	85	88	92	96	105	123	135	143	147	...	68	166	242	227	230	227	226	225	
3	0	203	205	207	206	207	209	210	209	210	...	154	248	247	248	253	236	230	240	
4	3	188	191	193	195	199	201	202	203	203	...	26	40	64	48	29	46	49	46	

5 rows × 785 columns

In [7]:
```
# Create a validation set from the testing set
validation_set = test_set.iloc[:int(test_set.shape[0]/2), :].reset_index(drop = True)
test_set = test_set.iloc[int(test_set.shape[0]/2) :,:].reset_index(drop = True)

print("The shape of validation set after spliting: \n")
print(validation_set.shape)
print("\n")
print("The shape of testing set after spliting: \n")
print(test_set.shape)
print("\n")
```

```
The shape of validation set after spliting:

(3586, 785)


The shape of testing set after spliting:

(3586, 785)
```

In [8]:
```
print("The first 6 rows of validation set:\n")
validation_set.head()
```

```
The first 6 rows of validation set:
```

Out[8]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixe
0	6	149	149	150	150	150	151	151	150	151	...	138	148	127	89	82	96	106	112	
1	5	126	128	131	132	133	134	135	135	136	...	47	104	194	183	186	184	184	184	
2	10	85	88	92	96	105	123	135	143	147	...	68	166	242	227	230	227	226	225	
3	0	203	205	207	206	207	209	210	209	210	...	154	248	247	248	253	236	230	240	
4	3	188	191	193	195	199	201	202	203	203	...	26	40	64	48	29	46	49	46	

5 rows × 785 columns

In [9]:
```
print("The first 6 rows of testing set after spliting:\n")
test_set.head()
```

```
The first 6 rows of testing set after spliting:
```

Out[9]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixe
0	15	85	90	92	93	98	99	103	116	126	...	103	103	104	108	97	136	155	135	
1	2	180	180	179	180	180	180	179	178	177	...	206	206	202	201	200	200	197	196	
2	19	169	172	174	174	175	177	178	177	177	...	204	203	203	201	201	202	200	199	
3	17	135	139	141	147	153	157	160	164	167	...	190	209	209	208	208	206	205	203	
4	8	170	172	173	173	174	174	174	175	175	...	198	196	195	193	190	188	187	185	

5 rows × 785 columns

In [10]:
```
# Split the training data into X_train and y_train
# Split the testing data into X_test and y_test
# Split the validation data into X_val and y_val
X_train = train_set.iloc[:, 1:]
X_test = test_set.iloc[:, 1:]
X_val = validation_set.iloc[:, 1:]
y_train = train_set.iloc[:, 0]
y_test = test_set.iloc[:, 0]
y_val = validation_set.iloc[:, 0]

print("The shape of X_train is: ", X_train.shape)
print("The shape of y_train is: ", y_train.shape)
print("The shape of X_test is: ", X_test.shape)
print("The shape of y_test is: ", y_test.shape)
print("The shape of X_val is: ", X_val.shape)
print("The shape of y_val is: ", y_val.shape)
```

```
The shape of X_train is:  (27455, 784)
The shape of y_train is:  (27455,)
The shape of X_test is:  (3586, 784)
The shape of y_test is:  (3586,)
The shape of X_val is:  (3586, 784)
The shape of y_val is:  (3586,)
```

In [11]:
```
# Scale the X_train, X_test, X_val appropriately (because it starts with max of 255, but we want max of 1)
X_train = X_train/255
X_test = X_test/255
X_val = X_val/255
print("Some statistic values of X_train columns after dividing by 255: \n")
X_train.describe()
```

```
Some statistic values of X_train columns after dividing by 255:
```

Out[11]:

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	
count	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000	...	27455
mean	0.570272	0.582354	0.593128	0.602143	0.612592	0.621221	0.629303	0.636626	0.642960	0.649152	...	0
std	0.162190	0.156636	0.153162	0.151354	0.145534	0.141669	0.137319	0.132008	0.128046	0.122664	...	0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0
25%	0.474510	0.494118	0.509804	0.521569	0.537255	0.549020	0.556863	0.564706	0.572549	0.580392	...	0
50%	0.588235	0.600000	0.611765	0.619608	0.627451	0.635294	0.643137	0.647059	0.650980	0.654902	...	0
75%	0.682353	0.690196	0.698039	0.701961	0.709804	0.713725	0.717647	0.721569	0.725490	0.729412	...	0
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1

8 rows × 784 columns

In [12]:
```
print("Some statistic values of X_test columns after dividing by 255: \n")
X_test.describe()
```

```
Some statistic values of X_test columns after dividing by 255:
```

Out[12]:

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel775	
count	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	...	3586.000000	358
mean	0.581033	0.592131	0.602985	0.612158	0.621954	0.631848	0.638021	0.643253	0.650955	0.657735	...	0.544539	
std	0.172228	0.166222	0.161076	0.156513	0.149923	0.143847	0.143417	0.141770	0.133413	0.127858	...	0.254261	
min	0.000000	0.000000	0.000000	0.000000	0.074510	0.101961	0.000000	0.000000	0.000000	0.098039	...	0.000000	
25%	0.482353	0.498039	0.517647	0.529412	0.541176	0.552941	0.564706	0.568627	0.576471	0.580392	...	0.352941	
50%	0.607843	0.615686	0.623529	0.631373	0.639216	0.647059	0.654902	0.658824	0.662745	0.666667	...	0.541176	
75%	0.701961	0.705882	0.709804	0.717647	0.721569	0.729412	0.733333	0.733333	0.737255	0.741176	...	0.764706	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	

8 rows × 784 columns

In [13]:
```
print("Some statistic values of X_val columns after dividing by 255: \n")
X_val.describe()
```

```
Some statistic values of X_val columns after dividing by 255:
```

Out[13]:

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel775	
count	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	...	3586.000000	358
mean	0.576086	0.587836	0.599558	0.608731	0.618593	0.629256	0.634785	0.640270	0.647786	0.654510	...	0.542101	
std	0.169658	0.162122	0.156086	0.152123	0.146130	0.139188	0.140591	0.139680	0.131061	0.125949	...	0.251657	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.039216	0.000000	0.000000	0.035294	0.023529	...	0.000000	
25%	0.474510	0.490196	0.509804	0.525490	0.537255	0.552941	0.564706	0.568627	0.576471	0.584314	...	0.352941	
50%	0.603922	0.611765	0.619608	0.627451	0.635294	0.643137	0.650980	0.654902	0.660784	0.666667	...	0.529412	
75%	0.698039	0.701961	0.705882	0.713725	0.717647	0.725490	0.729412	0.729412	0.733333	0.733333	...	0.760784	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	

8 rows × 784 columns

In [14]:
```
# Checking data type of X_train, X_test and X_val
print("Information of X_train columns: \n")
print(X_train.info())
print("\n")
print("Information of X_test columns: \n")
print(X_test.info())
print("\n")
print("Information of X_val columns: \n")
print(X_val.info())
print("\n")
```

```
Information of X_train columns:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27455 entries, 0 to 27454
Columns: 784 entries, pixel1 to pixel784
dtypes: float64(784)
memory usage: 164.2 MB
None


Information of X_test columns:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3586 entries, 0 to 3585
Columns: 784 entries, pixel1 to pixel784
dtypes: float64(784)
memory usage: 21.4 MB
None


Information of X_val columns:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3586 entries, 0 to 3585
Columns: 784 entries, pixel1 to pixel784
dtypes: float64(784)
memory usage: 21.4 MB
None
```

In [15]:
```
# Convert X_train, X_test, X_val to 3D array
X_train = np.array(X_train).reshape((-1, 28, 28, 1))
print('The shape of X_train after converting to an array is: ', X_train.shape)
X_test = np.array(X_test).reshape((-1, 28, 28, 1))
print('The shape of X_test after converting to an array is: ', X_test.shape)
X_val = np.array(X_val).reshape((-1, 28, 28, 1))
print('The shape of X_val after converting to an array is: ', X_val.shape)
```

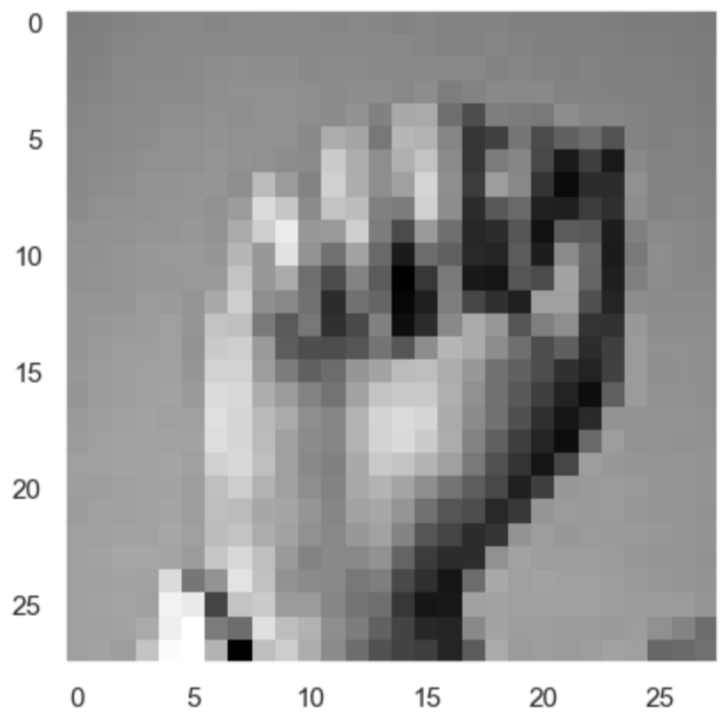
```
The shape of X_train after converting to an array is:  (27455, 28, 28, 1)
The shape of X_test after converting to an array is:  (3586, 28, 28, 1)
The shape of X_val after converting to an array is:  (3586, 28, 28, 1)
```

In [16]:
```
# Create a list of letters (labels of images)
print("List of letters (labels of images) is: \n")
class_names = ['A','B','C','D','E','F','G','H','I','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y']
print(class_names)
```

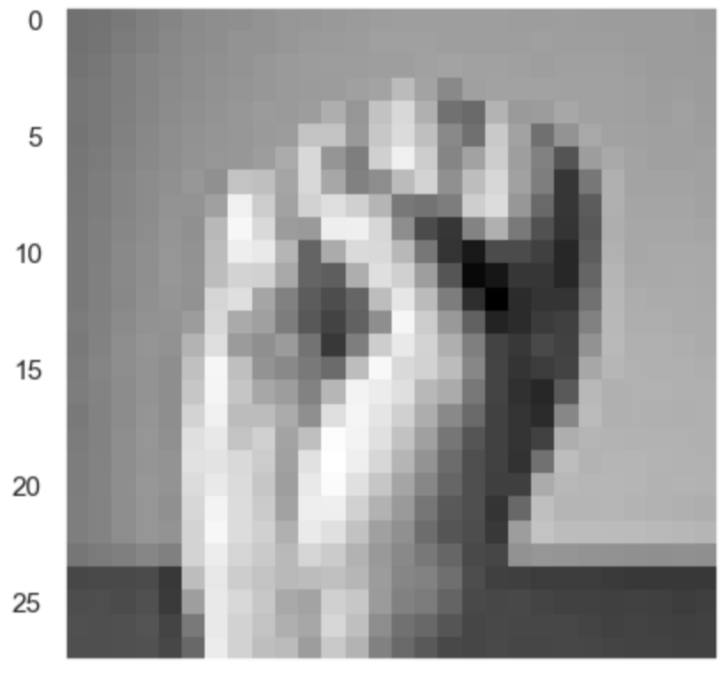
```
List of letters (labels of images) is:

['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
'Y']
```

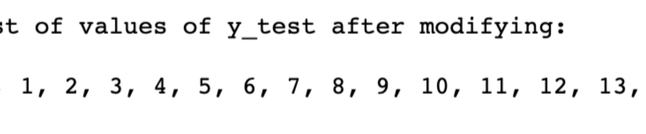
In [17]:
```
# Show some images of X_train
print('Label of this image: ', class_names[y_train[89]])
plt.imshow(X_train[89,:], cmap='gray')
plt.grid(False)
plt.show()

print('Label of this image: ', class_names[y_train[10]])
plt.imshow(X_train[10,:], cmap='gray')
plt.grid(False)
plt.show()
```

```
Label of this image:  A
```




```
Label of this image:  T
```





After checking some images and their labels, some labels and their according images are not consistent. Particularly, labels with value larger than 8 are inconsistent with their images, hence, these values in the response variable need to be deducted by 1 in order to generate correct labels.

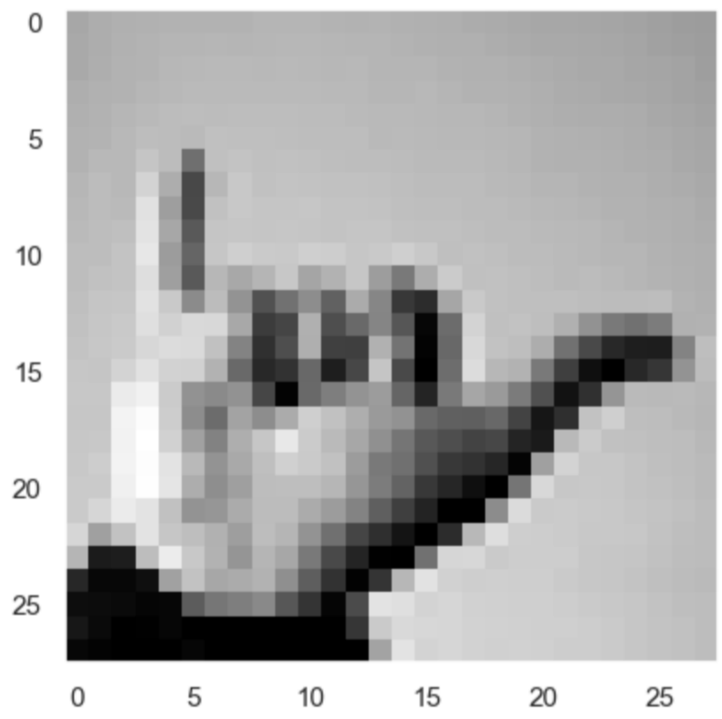
In [18]:
```
# Modifying values of y_train, y_test and y_val
y_train = y_train.apply(lambda x: x - 1 if x > 8 else x)
y_test = y_test.apply(lambda x: x - 1 if x > 8 else x)
y_val = y_val.apply(lambda x: x - 1 if x > 8 else x)
```

In [19]:
```
# List of values of response variable after modifying
print("List of values of y_train after modifying: \n")
print(list(y_train.sort_values().unique()))
print('\n')
print("List of values of y_test after modifying: \n")
print(list(y_test.sort_values().unique()))
print('\n')
print("List of values of y_val after modifying: \n")
print(list(y_val.sort_values().unique()))
```

```
List of values of y_train after modifying:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 199]


List of values of y_test after modifying:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]


List of values of y_val after modifying:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]
```

In [20]:
```
# Check images of X_train again after modifying values of the response variable
print('Label of this image: ', class_names[y_train[26]])
plt.imshow(X_train[26,:], cmap='gray')
plt.grid(False)
plt.show()
```

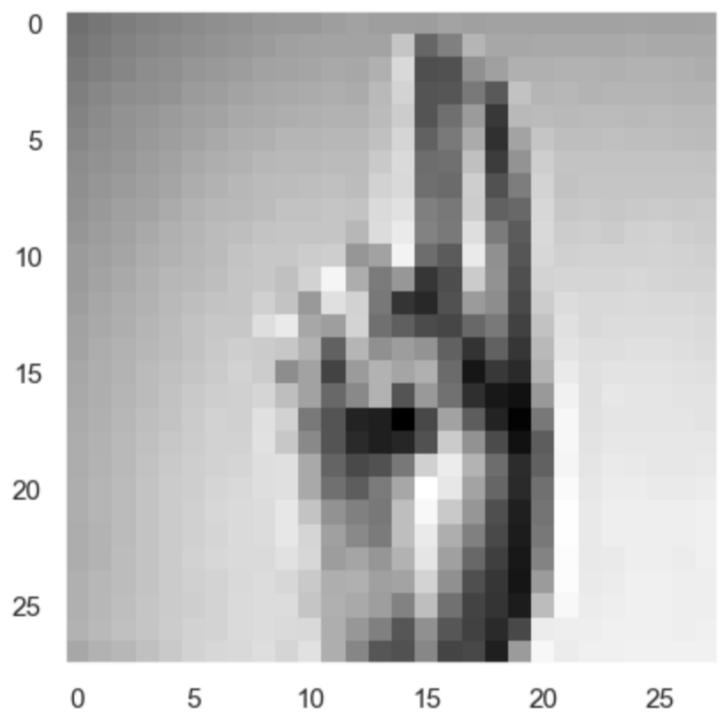
```
Label of this image:  Y
```



In [21]:
```
# Check an invalid value of y_train
y_train.loc[y_train == 199]
```

Out[21]:
```
498    199
Name: label, dtype: int64
```

In [22]:
```
# Check an image of the observation with value of response variable equals to 199
plt.imshow(X_train[498,:], cmap='gray')
plt.grid(False)
plt.show()
```

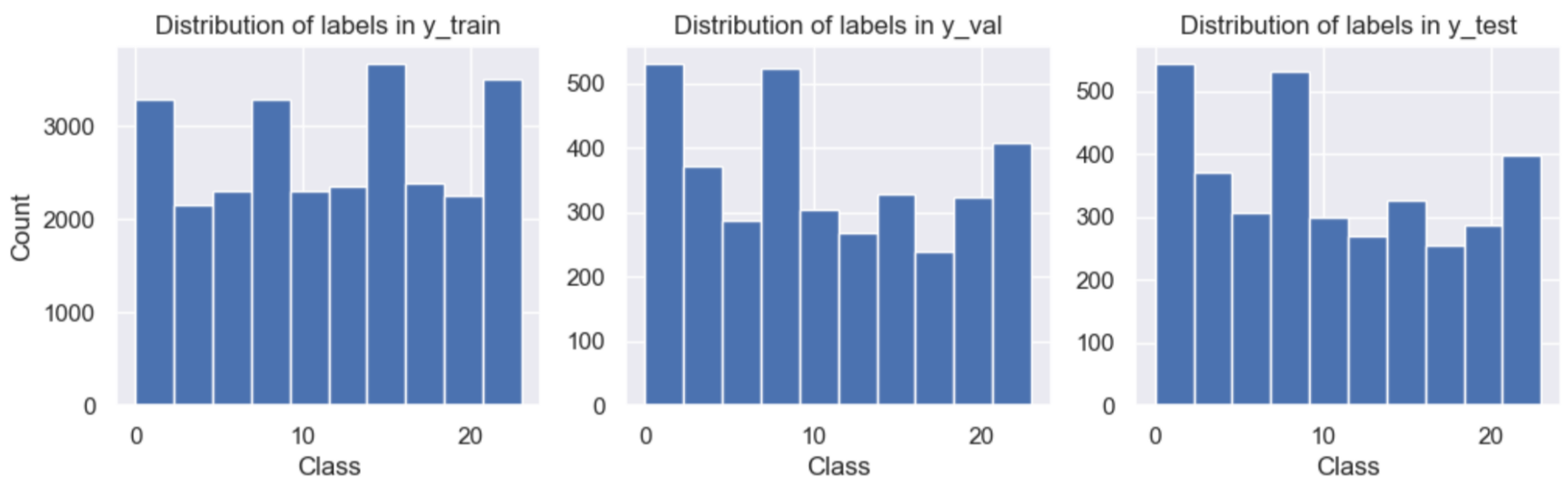


There is an observation (its index is 498) with a value of 199 in y_train, this value is out of the range of possible labels, hence it needs to be corrected. After checking its image, the label value of this image should be 19 (letter U) instead of 199.

In [23]:
```
y_train = y_train.apply(lambda x: 19 if x ==199 else x)

print("List of values of y_train after modifying: \n")
print(list(y_train.sort_values().unique()))
print('\n')
print("List of respective letters (labels of images): \n")
print(class_names)
```

```
List of values of y_train after modifying:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]


List of respective letters (labels of images):

['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
'Y']
```

In [24]:
```
# Checking distribution of lables in y_train, y_test, and y_val
fig, sub = plt.subplots(1, 3)
fig.set_size_inches(12,3)
sub[0].hist(y_train)
sub[0].set_ylabel('Count')
sub[0].set_xlabel('Class')
sub[0].set_title('Distribution of labels in y_train')
sub[1].hist(y_val)
sub[1].set_xlabel('Class')
sub[1].set_title('Distribution of labels in y_val')
sub[2].hist(y_test)
sub[2].set_xlabel('Class')
sub[2].set_title('Distribution of labels in y_test')
plt.show()
```



2. Build a baseline model

For a baseline model, a densely connected model with Stochastic Gradient Descent optimizer is built, its learning rate is 0.01 by default, batch size is 32 and the number of epochs are 30. In this network, there is 3 hidden layers. The first hidden layer contains 64 neurons and uses Elu activation function. The second hidden layer contains 32 neurons and uses Elu activation function. The third hidden layer contains 16 neurons and uses Elu activation function. The output layer contains 24 neurons (since there are 24 possible outcomes of labels) and uses Softmax activation function. The loss function is Sparse categorical crossentropy since labels are encoded as integers from 0 to 23.

In [25]:
```
# Build a baseline model
actfn_bs = "elu"
optimizer_bs = keras.optimizers.SGD
learningrate_bs = 0.01
batch_size_bs = 32
n_epochs_bs = 30

model_bs = keras.models.Sequential()
model_bs.add(keras.layers.Flatten(input_shape = [28, 28, 1]))
model_bs.add(keras.layers.Dense(64, activation = actfn_bs))
model_bs.add(keras.layers.Dense(32, activation = actfn_bs))
model_bs.add(keras.layers.Dense(16, activation = actfn_bs))
model_bs.add(keras.layers.Dense(24, activation = "softmax"))
model_bs.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer_bs(learning_rate=learningrate_bs),
                 metrics=["accuracy"])
history_bs = model_bs.fit(X_train, y_train, epochs=n_epochs_bs, validation_data=(X_val, y_val))
```

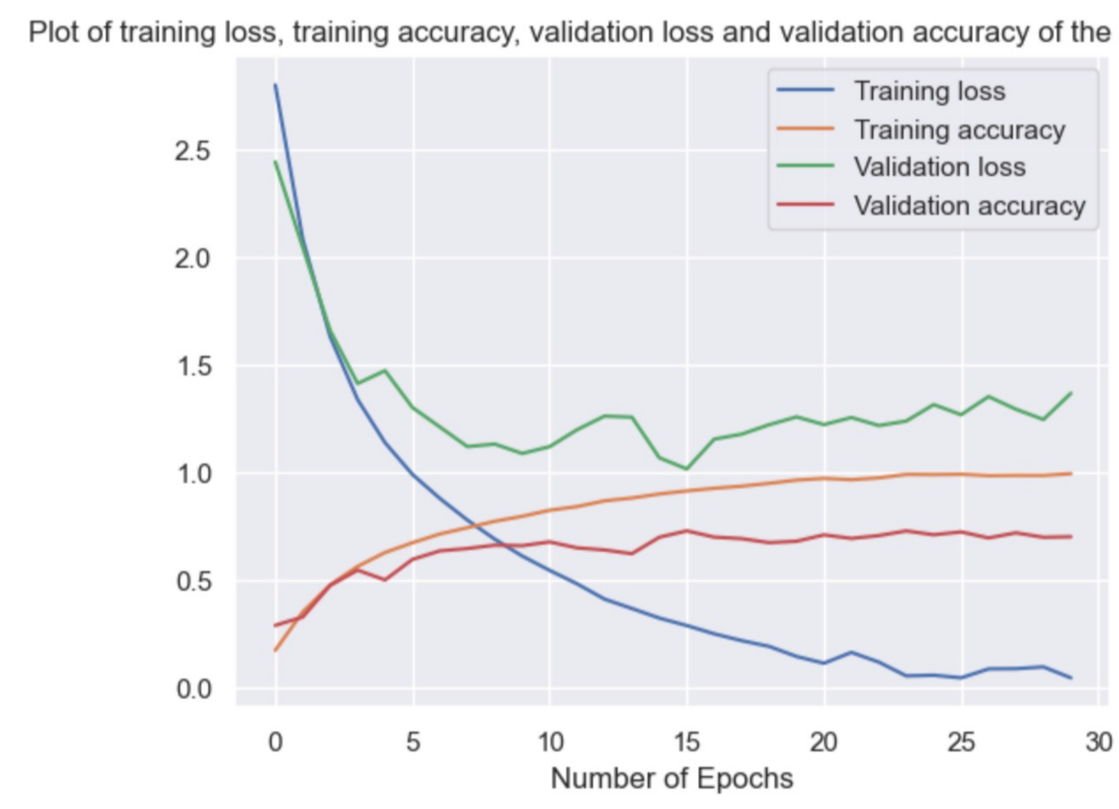
```
Epoch 1/30
858/858 [==============================] - 1s 858us/step - loss: 2.7980 - accuracy: 0.1747 - val_loss: 2.4408 - val_a
ccuracy: 0.2903
Epoch 2/30
858/858 [==============================] - 1s 739us/step - loss: 2.0829 - accuracy: 0.3522 - val_loss: 2.0454 - val_a
ccuracy: 0.3291
Epoch 3/30
858/858 [==============================] - 1s 746us/step - loss: 1.6300 - accuracy: 0.4784 - val_loss: 1.6585 - val_a
ccuracy: 0.4755
Epoch 4/30
858/858 [==============================] - 1s 770us/step - loss: 1.3385 - accuracy: 0.5638 - val_loss: 1.4125 - val_a
ccuracy: 0.5468
Epoch 5/30
858/858 [==============================] - 1s 749us/step - loss: 1.1382 - accuracy: 0.6284 - val_loss: 1.4723 - val_a
ccuracy: 0.5011
Epoch 6/30
858/858 [==============================] - 1s 740us/step - loss: 0.9906 - accuracy: 0.6737 - val_loss: 1.3013 - val_a
ccuracy: 0.5965
Epoch 7/30
858/858 [==============================] - 1s 746us/step - loss: 0.8793 - accuracy: 0.7144 - val_loss: 1.2108 - val_a
ccuracy: 0.6361
Epoch 8/30
858/858 [==============================] - 1s 756us/step - loss: 0.7791 - accuracy: 0.7435 - val_loss: 1.1200 - val_a
ccuracy: 0.6470
Epoch 9/30
858/858 [==============================] - 1s 742us/step - loss: 0.6903 - accuracy: 0.7735 - val_loss: 1.1317 - val_a
ccuracy: 0.6626
Epoch 10/30
858/858 [==============================] - 1s 752us/step - loss: 0.6122 - accuracy: 0.7964 - val_loss: 1.0881 - val_a
ccuracy: 0.6601
Epoch 11/30
858/858 [==============================] - 1s 756us/step - loss: 0.5456 - accuracy: 0.8244 - val_loss: 1.1197 - val_a
ccuracy: 0.6776
Epoch 12/30
858/858 [==============================] - 1s 744us/step - loss: 0.4832 - accuracy: 0.8416 - val_loss: 1.1982 - val_a
```

```
ccuracy: 0.6500
Epoch 13/30
858/858 [==============================] - 1s 769us/step - loss: 0.4122 - accuracy: 0.8683 - val_loss: 1.2614 - val_a
ccuracy: 0.6403
Epoch 14/30
858/858 [==============================] - 1s 743us/step - loss: 0.3686 - accuracy: 0.8813 - val_loss: 1.2565 - val_a
ccuracy: 0.6230
Epoch 15/30
858/858 [==============================] - 1s 743us/step - loss: 0.3241 - accuracy: 0.9001 - val_loss: 1.0681 - val_a
ccuracy: 0.6994
Epoch 16/30
858/858 [==============================] - 1s 746us/step - loss: 0.2894 - accuracy: 0.9142 - val_loss: 1.0161 - val_a
ccuracy: 0.7289
Epoch 17/30
858/858 [==============================] - 1s 762us/step - loss: 0.2511 - accuracy: 0.9264 - val_loss: 1.1545 - val_a
ccuracy: 0.6999
Epoch 18/30
858/858 [==============================] - 1s 752us/step - loss: 0.2195 - accuracy: 0.9362 - val_loss: 1.1776 - val_a
ccuracy: 0.6927
Epoch 19/30
858/858 [==============================] - 1s 749us/step - loss: 0.1927 - accuracy: 0.9494 - val_loss: 1.2214 - val_a
ccuracy: 0.6743
Epoch 20/30
858/858 [==============================] - 1s 754us/step - loss: 0.1460 - accuracy: 0.9645 - val_loss: 1.2578 - val_a
ccuracy: 0.6807
Epoch 21/30
858/858 [==============================] - 1s 746us/step - loss: 0.1146 - accuracy: 0.9725 - val_loss: 1.2220 - val_a
ccuracy: 0.7103
Epoch 22/30
858/858 [==============================] - 1s 750us/step - loss: 0.1649 - accuracy: 0.9663 - val_loss: 1.2549 - val_a
ccuracy: 0.6938
Epoch 23/30
858/858 [==============================] - 1s 742us/step - loss: 0.1199 - accuracy: 0.9748 - val_loss: 1.2174 - val_a
ccuracy: 0.7069
Epoch 24/30
858/858 [==============================] - 1s 743us/step - loss: 0.0557 - accuracy: 0.9908 - val_loss: 1.2384 - val_a
ccuracy: 0.7287
Epoch 25/30
858/858 [==============================] - 1s 742us/step - loss: 0.0590 - accuracy: 0.9900 - val_loss: 1.3148 - val_a
ccuracy: 0.7108
Epoch 26/30
858/858 [==============================] - 1s 742us/step - loss: 0.0468 - accuracy: 0.9914 - val_loss: 1.2680 - val_a
ccuracy: 0.7239
Epoch 27/30
858/858 [==============================] - 1s 739us/step - loss: 0.0882 - accuracy: 0.9854 - val_loss: 1.3520 - val_a
ccuracy: 0.6960
Epoch 28/30
858/858 [==============================] - 1s 743us/step - loss: 0.0892 - accuracy: 0.9865 - val_loss: 1.2932 - val_a
ccuracy: 0.7197
Epoch 29/30
858/858 [==============================] - 1s 746us/step - loss: 0.0975 - accuracy: 0.9862 - val_loss: 1.2455 - val_a
ccuracy: 0.6991
Epoch 30/30
858/858 [==============================] - 1s 741us/step - loss: 0.0468 - accuracy: 0.9942 - val_loss: 1.3674 - val_a
ccuracy: 0.7016
```

In [26]:
```
model_bs.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 64)                50240

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 16)                528

 dense_3 (Dense)             (None, 24)                408

=================================================================
Total params: 53256 (208.03 KB)
Trainable params: 53256 (208.03 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [27]:
```
pd.DataFrame(history_bs.history).plot()
plt.xlabel("Number of Epochs")
plt.title("Plot of training loss, training accuracy, validation loss and validation accuracy of the baseline model")
plt.legend(['Training loss', 'Training accuracy', 'Validation loss', 'Validation accuracy'])
plt.show()
```



According to the plot of training loss, training accuracy, validation loss and validation accuracy of the baseline model above, the training accuracy and validation accuracy of this baseline model converge after 20 epochs. However, the difference between training loss and validation loss, as well as training accuracy and validation accuracy are quite large, which can indicate that this baseline model can be overfitting. For example, after epoch 30, the training

accuracy is 0.9942 while the validation accuracy is only 0.7016, hence the gap between these two values is about 0.3.

3. Train and optimize a Densely connected and a Convolutional neural network model

In training and optimizing process, there are some hyperparameters that need to be tuned in order to find the optimized model which are the activation function ('actfn' list including Elu, Leaky Relu and Selu activation functions), the optimizer ('optimizer' list including Stochastic Gradient Descent, Adam, and Nadam optimizer) and its learning rate ('learningrate' list containing values of 0.01 and 0.001). These hyperparameters are used for optimizing both the Densely connected model and the Convolutional neural network (CNN) model. Early stopping is also defined with patience of 5, and monitored by validation loss values. Early stopping is used to prevent models from overfitting since the baseline model raises a signal of overfitting problem.

In [28]:
```
learningrate = [0.01, 0.001]
optimizer = [keras.optimizers.SGD, keras.optimizers.Adam, keras.optimizers.Nadam]
actfn = ['elu', 'LeakyReLU', 'selu']
early_stopping_cb = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

3.1 Train and optimize a Densely connected model

The densely connect model is built with 1 Flatten layer for input data, three hidden layers with size of 64, 32, and 16 respectively, and 1 Dense layer including 24 neurons for the output. The used loss function is Sparse categorical crossentropy since labels are encoding as integers from 0 to 23.

In [29]:
```
def model_dense(hiddensizes, actfn, optimizer, learningrate = 0):
    model_dense = keras.models.Sequential()
    model_dense.add(keras.layers.Flatten(input_shape = [28, 28, 1]))
    for n in hiddensizes:
        model_dense.add(keras.layers.Dense(n, activation = actfn))
    model_dense.add(keras.layers.Dense(24, activation = "softmax"))
    model_dense.compile(loss="sparse_categorical_crossentropy",
                        optimizer=optimizer(learning_rate=learningrate), metrics=["accuracy"])
    return model_dense
```

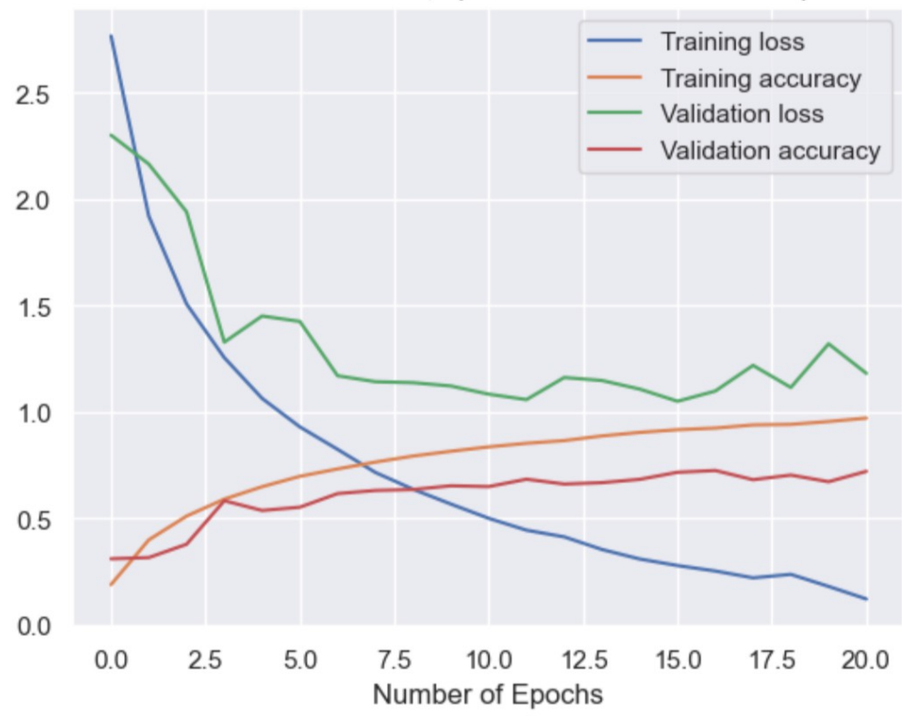
In [30]:
```
def build_dense(hiddensizes, actfn, optimizer, learningrate, n_epochs, batch_size, further_callbacks=[]):
    if further_callbacks != []:
        callbacks = further_callbacks
    else:
        callbacks = [early_stopping_cb]
    model = model_dense(hiddensizes, actfn, optimizer, learningrate)
    history = model.fit(X_train, y_train, epochs=n_epochs, callbacks = callbacks,
                        validation_data=(X_val, y_val))
    max_val_acc = np.max(history.history['val_accuracy'])
    return (max_val_acc, history, model)
```

In [31]:
```
hiddensizes = [64,32,16]
batch_size = 32
n_epochs = 50

res_dense = []
for i in learningrate:
    for o in optimizer:
        for a in actfn:
            max_val_acc_dense, history_dense, model_d = build_dense(hiddensizes, a, o, i, n_epochs, batch_size,
                                                                    further_callbacks = [])
            pd.DataFrame(history_dense.history).plot()
            plt.xlabel("Number of Epochs")
            plt.title(f'Plot of the densely connected model with activation function: {a}, optimizer: {o} and learning
            plt.legend(['Training loss', 'Training accuracy', 'Validation loss', 'Validation accuracy'])
            plt.show()
            res_dense += [[i, o, a, max_val_acc_dense]]
```

```
Epoch 1/50
858/858 [==============================] - 1s 828us/step - loss: 2.7661 - accuracy: 0.1902 - val_loss: 2.3005 - val_a
ccuracy: 0.3112
Epoch 2/50
858/858 [==============================] - 1s 729us/step - loss: 1.9200 - accuracy: 0.4001 - val_loss: 2.1646 - val_a
ccuracy: 0.3171
Epoch 3/50
858/858 [==============================] - 1s 731us/step - loss: 1.5080 - accuracy: 0.5117 - val_loss: 1.9403 - val_a
ccuracy: 0.3795
Epoch 4/50
858/858 [==============================] - 1s 734us/step - loss: 1.2556 - accuracy: 0.5914 - val_loss: 1.3285 - val_a
ccuracy: 0.5839
Epoch 5/50
858/858 [==============================] - 1s 736us/step - loss: 1.0645 - accuracy: 0.6498 - val_loss: 1.4512 - val_a
ccuracy: 0.5382
Epoch 6/50
858/858 [==============================] - 1s 730us/step - loss: 0.9306 - accuracy: 0.6985 - val_loss: 1.4258 - val_a
ccuracy: 0.5535
Epoch 7/50
858/858 [==============================] - 1s 733us/step - loss: 0.8251 - accuracy: 0.7333 - val_loss: 1.1706 - val_a
ccuracy: 0.6174
Epoch 8/50
858/858 [==============================] - 1s 737us/step - loss: 0.7167 - accuracy: 0.7654 - val_loss: 1.1428 - val_a
ccuracy: 0.6322
Epoch 9/50
858/858 [==============================] - 1s 738us/step - loss: 0.6375 - accuracy: 0.7939 - val_loss: 1.1381 - val_a
ccuracy: 0.6366
Epoch 10/50
858/858 [==============================] - 1s 737us/step - loss: 0.5682 - accuracy: 0.8163 - val_loss: 1.1227 - val_a
ccuracy: 0.6542
Epoch 11/50
858/858 [==============================] - 1s 752us/step - loss: 0.5014 - accuracy: 0.8367 - val_loss: 1.0843 - val_a
ccuracy: 0.6503
Epoch 12/50
858/858 [==============================] - 1s 746us/step - loss: 0.4456 - accuracy: 0.8535 - val_loss: 1.0587 - val_a
ccuracy: 0.6849
Epoch 13/50
858/858 [==============================] - 1s 741us/step - loss: 0.4143 - accuracy: 0.8663 - val_loss: 1.1625 - val_a
ccuracy: 0.6623
Epoch 14/50
858/858 [==============================] - 1s 752us/step - loss: 0.3549 - accuracy: 0.8882 - val_loss: 1.1484 - val_a
ccuracy: 0.6687
Epoch 15/50
858/858 [==============================] - 1s 753us/step - loss: 0.3103 - accuracy: 0.9047 - val_loss: 1.1080 - val_a
ccuracy: 0.6843
Epoch 16/50
858/858 [==============================] - 1s 741us/step - loss: 0.2796 - accuracy: 0.9173 - val_loss: 1.0509 - val_a
ccuracy: 0.7175
Epoch 17/50
```

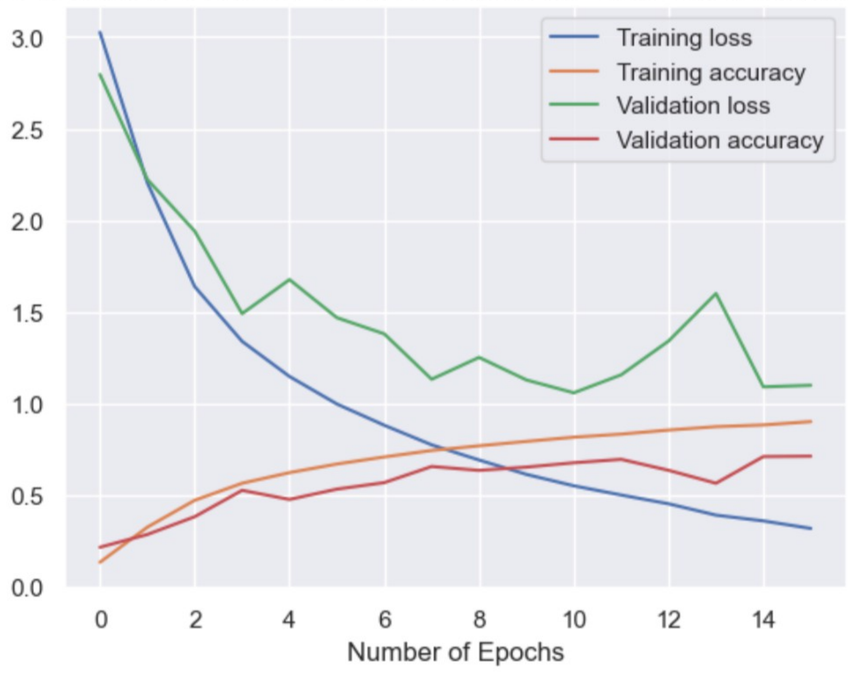
```
858/858 [==============================] - 1s 795us/step - loss: 0.2540 - accuracy: 0.9250 - val_loss: 1.0989 - val_a
ccuracy: 0.7259
Epoch 18/50
858/858 [==============================] - 1s 814us/step - loss: 0.2219 - accuracy: 0.9398 - val_loss: 1.2195 - val_a
ccuracy: 0.6827
Epoch 19/50
858/858 [==============================] - 1s 751us/step - loss: 0.2382 - accuracy: 0.9424 - val_loss: 1.1157 - val_a
ccuracy: 0.7044
Epoch 20/50
858/858 [==============================] - 1s 744us/step - loss: 0.1816 - accuracy: 0.9559 - val_loss: 1.3207 - val_a
ccuracy: 0.6732
Epoch 21/50
858/858 [==============================] - 1s 744us/step - loss: 0.1226 - accuracy: 0.9719 - val_loss: 1.1821 - val_a
ccuracy: 0.7220
```

Plot of the densely connected model with activation function: elu, optimizer: <class 'keras.src.optimizers.sgd.SGD'> and learning rate: 0.01



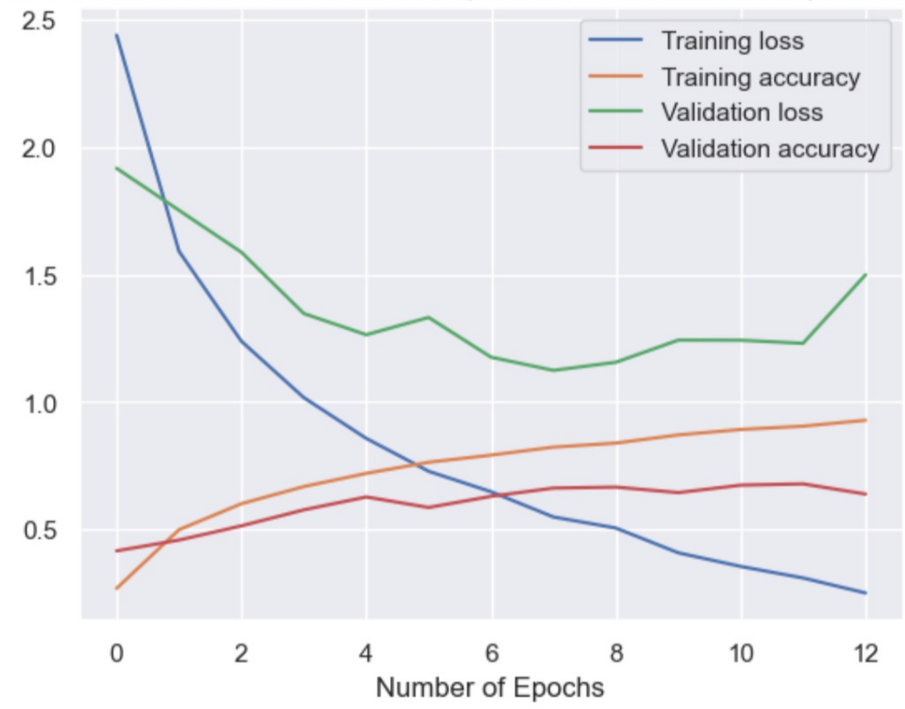

```
Epoch 1/50
858/858 [==============================] - 1s 801us/step - loss: 3.0261 - accuracy: 0.1333 - val_loss: 2.7956 - val_a
ccuracy: 0.2153
Epoch 2/50
858/858 [==============================] - 1s 751us/step - loss: 2.2052 - accuracy: 0.3248 - val_loss: 2.2232 - val_a
ccuracy: 0.2850
Epoch 3/50
858/858 [==============================] - 1s 839us/step - loss: 1.6381 - accuracy: 0.4717 - val_loss: 1.9406 - val_a
ccuracy: 0.3820
Epoch 4/50
858/858 [==============================] - 1s 700us/step - loss: 1.3393 - accuracy: 0.5646 - val_loss: 1.4907 - val_a
ccuracy: 0.5259
Epoch 5/50
858/858 [==============================] - 1s 705us/step - loss: 1.1475 - accuracy: 0.6226 - val_loss: 1.6763 - val_a
ccuracy: 0.4763
Epoch 6/50
858/858 [==============================] - 1s 708us/step - loss: 0.9973 - accuracy: 0.6697 - val_loss: 1.4684 - val_a
ccuracy: 0.5326
Epoch 7/50
858/858 [==============================] - 1s 746us/step - loss: 0.8813 - accuracy: 0.7082 - val_loss: 1.3800 - val_a
ccuracy: 0.5680
Epoch 8/50
858/858 [==============================] - 1s 708us/step - loss: 0.7743 - accuracy: 0.7435 - val_loss: 1.1324 - val_a
ccuracy: 0.6562
Epoch 9/50
858/858 [==============================] - 1s 713us/step - loss: 0.6914 - accuracy: 0.7690 - val_loss: 1.2517 - val_a
ccuracy: 0.6350
Epoch 10/50
858/858 [==============================] - 1s 713us/step - loss: 0.6130 - accuracy: 0.7931 - val_loss: 1.1285 - val_a
ccuracy: 0.6528
Epoch 11/50
858/858 [==============================] - 1s 706us/step - loss: 0.5507 - accuracy: 0.8161 - val_loss: 1.0583 - val_a
ccuracy: 0.6765
Epoch 12/50
858/858 [==============================] - 1s 715us/step - loss: 0.5004 - accuracy: 0.8329 - val_loss: 1.1566 - val_a
ccuracy: 0.6952
Epoch 13/50
858/858 [==============================] - 1s 710us/step - loss: 0.4525 - accuracy: 0.8550 - val_loss: 1.3410 - val_a
ccuracy: 0.6352
Epoch 14/50
858/858 [==============================] - 1s 709us/step - loss: 0.3906 - accuracy: 0.8736 - val_loss: 1.6001 - val_a
ccuracy: 0.5647
Epoch 15/50
858/858 [==============================] - 1s 715us/step - loss: 0.3591 - accuracy: 0.8829 - val_loss: 1.0909 - val_a
ccuracy: 0.7103
Epoch 16/50
858/858 [==============================] - 1s 710us/step - loss: 0.3180 - accuracy: 0.9011 - val_loss: 1.0993 - val_a
ccuracy: 0.7125
```

Plot of the densely connected model with activation function: LeakyReLU, optimizer: <class 'keras.src.optimizers.sgd.SGD'> and learning rate: 0.01



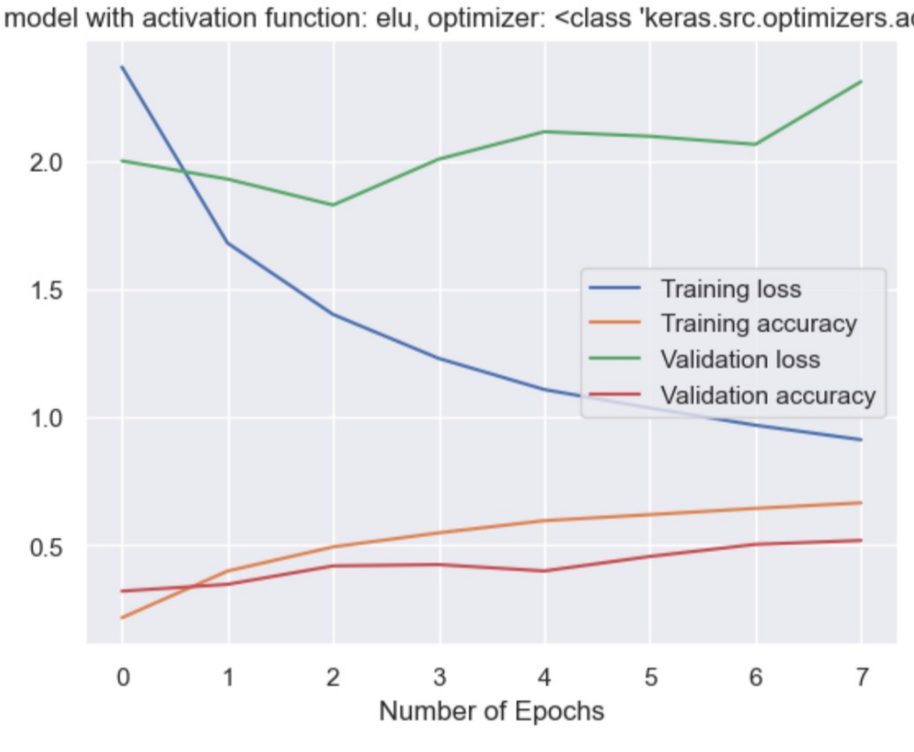
```
Epoch 1/50
858/858 [==============================] - 1s 841us/step - loss: 2.4381 - accuracy: 0.2684 - val_loss: 1.9159 - val_a
ccuracy: 0.4155
Epoch 2/50
858/858 [==============================] - 1s 751us/step - loss: 1.5901 - accuracy: 0.4985 - val_loss: 1.7512 - val_a
ccuracy: 0.4579
Epoch 3/50
858/858 [==============================] - 1s 767us/step - loss: 1.2374 - accuracy: 0.6005 - val_loss: 1.5866 - val_a
ccuracy: 0.5137
Epoch 4/50
858/858 [==============================] - 1s 753us/step - loss: 1.0165 - accuracy: 0.6677 - val_loss: 1.3462 - val_a
ccuracy: 0.5764
Epoch 5/50
858/858 [==============================] - 1s 752us/step - loss: 0.8566 - accuracy: 0.7194 - val_loss: 1.2631 - val_a
ccuracy: 0.6269
Epoch 6/50
858/858 [==============================] - 1s 762us/step - loss: 0.7275 - accuracy: 0.7630 - val_loss: 1.3310 - val_a
ccuracy: 0.5856
Epoch 7/50
858/858 [==============================] - 1s 746us/step - loss: 0.6474 - accuracy: 0.7910 - val_loss: 1.1751 - val_a
ccuracy: 0.6294
Epoch 8/50
858/858 [==============================] - 1s 747us/step - loss: 0.5483 - accuracy: 0.8224 - val_loss: 1.1233 - val_a
ccuracy: 0.6617
Epoch 9/50
858/858 [==============================] - 1s 750us/step - loss: 0.5051 - accuracy: 0.8382 - val_loss: 1.1552 - val_a
ccuracy: 0.6656
Epoch 10/50
858/858 [==============================] - 1s 751us/step - loss: 0.4079 - accuracy: 0.8702 - val_loss: 1.2420 - val_a
ccuracy: 0.6439
Epoch 11/50
858/858 [==============================] - 1s 756us/step - loss: 0.3544 - accuracy: 0.8915 - val_loss: 1.2415 - val_a
ccuracy: 0.6732
Epoch 12/50
858/858 [==============================] - 1s 750us/step - loss: 0.3092 - accuracy: 0.9044 - val_loss: 1.2295 - val_a
ccuracy: 0.6782
Epoch 13/50
858/858 [==============================] - 1s 744us/step - loss: 0.2503 - accuracy: 0.9277 - val_loss: 1.4982 - val_a
ccuracy: 0.6383
```

Plot of the densely connected model with activation function: selu, optimizer: <class 'keras.src.optimizers.sgd.SGD'> and learning rate: 0.01



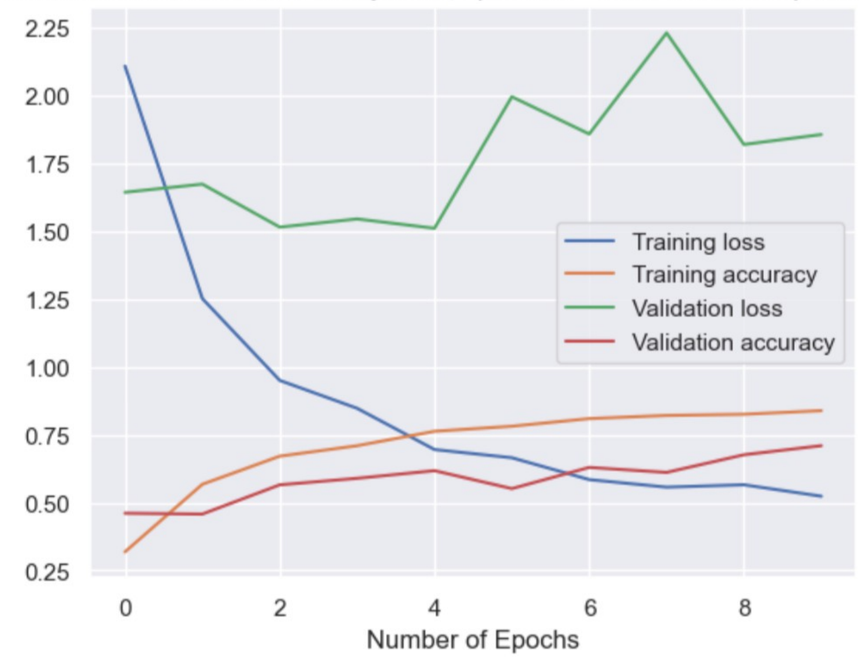

```
Epoch 1/50
858/858 [==============================] - 1s 984us/step - loss: 2.3659 - accuracy: 0.2175 - val_loss: 2.0000 - val_a
ccuracy: 0.3212
Epoch 2/50
858/858 [==============================] - 1s 911us/step - loss: 1.6791 - accuracy: 0.3996 - val_loss: 1.9286 - val_a
ccuracy: 0.3477
Epoch 3/50
858/858 [==============================] - 1s 907us/step - loss: 1.4004 - accuracy: 0.4938 - val_loss: 1.8275 - val_a
ccuracy: 0.4197
Epoch 4/50
858/858 [==============================] - 1s 895us/step - loss: 1.2291 - accuracy: 0.5485 - val_loss: 2.0064 - val_a
ccuracy: 0.4247
Epoch 5/50
858/858 [==============================] - 1s 901us/step - loss: 1.1077 - accuracy: 0.5961 - val_loss: 2.1134 - val_a
ccuracy: 0.3999
Epoch 6/50
858/858 [==============================] - 1s 895us/step - loss: 1.0358 - accuracy: 0.6196 - val_loss: 2.0957 - val_a
ccuracy: 0.4565
Epoch 7/50
858/858 [==============================] - 1s 914us/step - loss: 0.9685 - accuracy: 0.6440 - val_loss: 2.0641 - val_a
ccuracy: 0.5036
Epoch 8/50
858/858 [==============================] - 1s 907us/step - loss: 0.9120 - accuracy: 0.6657 - val_loss: 2.3086 - val_a
ccuracy: 0.5190
```

Plot of the densely connected model with activation function: elu, optimizer: <class 'keras.src.optimizers.adam.Adam'> and learning rate: 0.01



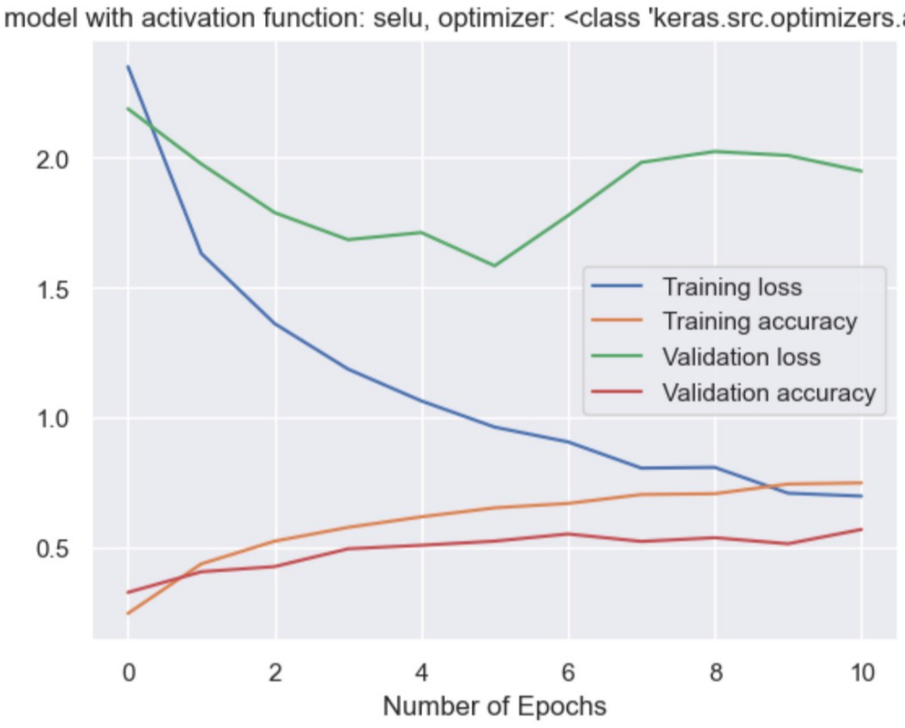
```
Epoch 1/50
858/858 [==============================] - 1s 977us/step - loss: 2.1070 - accuracy: 0.3185 - val_loss: 1.6425 - val_a
ccuracy: 0.4607
Epoch 2/50
858/858 [==============================] - 1s 890us/step - loss: 1.2510 - accuracy: 0.5673 - val_loss: 1.6724 - val_a
ccuracy: 0.4576
Epoch 3/50
858/858 [==============================] - 1s 888us/step - loss: 0.9498 - accuracy: 0.6708 - val_loss: 1.5141 - val_a
ccuracy: 0.5653
Epoch 4/50
858/858 [==============================] - 1s 881us/step - loss: 0.8471 - accuracy: 0.7092 - val_loss: 1.5442 - val_a
ccuracy: 0.5892
Epoch 5/50
858/858 [==============================] - 1s 890us/step - loss: 0.6952 - accuracy: 0.7624 - val_loss: 1.5099 - val_a
ccuracy: 0.6174
Epoch 6/50
858/858 [==============================] - 1s 895us/step - loss: 0.6650 - accuracy: 0.7810 - val_loss: 1.9942 - val_a
ccuracy: 0.5513
Epoch 7/50
858/858 [==============================] - 1s 891us/step - loss: 0.5845 - accuracy: 0.8091 - val_loss: 1.8570 - val_a
ccuracy: 0.6294
Epoch 8/50
858/858 [==============================] - 1s 887us/step - loss: 0.5567 - accuracy: 0.8208 - val_loss: 2.2292 - val_a
ccuracy: 0.6110
Epoch 9/50
858/858 [==============================] - 1s 882us/step - loss: 0.5654 - accuracy: 0.8251 - val_loss: 1.8184 - val_a
ccuracy: 0.6760
Epoch 10/50
858/858 [==============================] - 1s 892us/step - loss: 0.5233 - accuracy: 0.8384 - val_loss: 1.8550 - val_a
ccuracy: 0.7094
```

Plot of the densely connected model with activation function: LeakyReLU, optimizer: <class 'keras.src.optimizers.adam.Adam'> and learning rate: 0.01



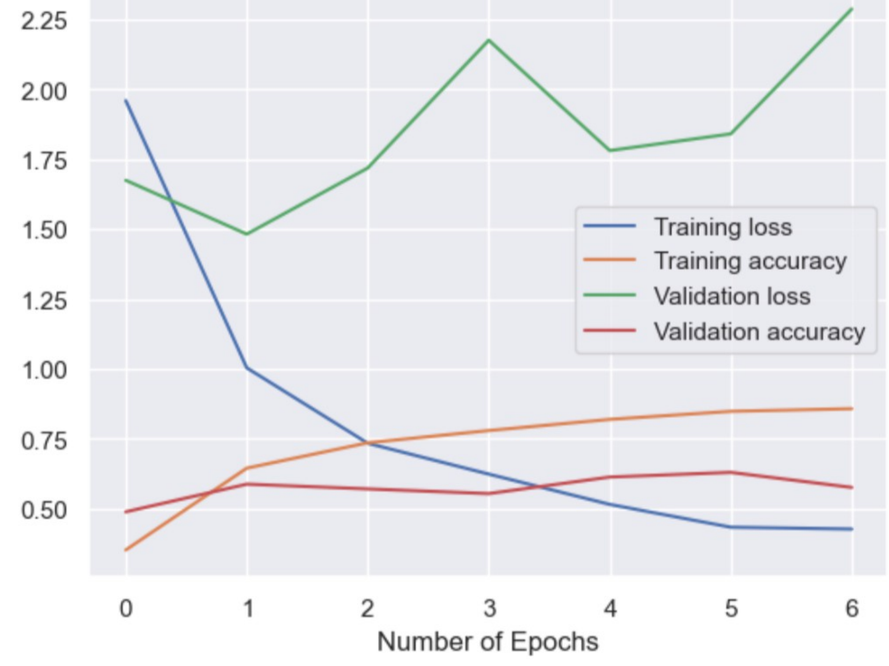

```
Epoch 1/50
858/858 [==============================] - 1s 990us/step - loss: 2.3501 - accuracy: 0.2478 - val_loss: 2.1882 - val_a
ccuracy: 0.3288
Epoch 2/50
858/858 [==============================] - 1s 918us/step - loss: 1.6318 - accuracy: 0.4384 - val_loss: 1.9760 - val_a
ccuracy: 0.4083
Epoch 3/50
858/858 [==============================] - 1s 898us/step - loss: 1.3623 - accuracy: 0.5258 - val_loss: 1.7889 - val_a
ccuracy: 0.4281
Epoch 4/50
858/858 [==============================] - 1s 895us/step - loss: 1.1874 - accuracy: 0.5788 - val_loss: 1.6849 - val_a
ccuracy: 0.4964
Epoch 5/50
858/858 [==============================] - 1s 902us/step - loss: 1.0650 - accuracy: 0.6194 - val_loss: 1.7123 - val_a
ccuracy: 0.5100
Epoch 6/50
858/858 [==============================] - 1s 931us/step - loss: 0.9643 - accuracy: 0.6539 - val_loss: 1.5844 - val_a
ccuracy: 0.5259
Epoch 7/50
858/858 [==============================] - 1s 929us/step - loss: 0.9074 - accuracy: 0.6710 - val_loss: 1.7776 - val_a
ccuracy: 0.5535
Epoch 8/50
858/858 [==============================] - 1s 917us/step - loss: 0.8067 - accuracy: 0.7051 - val_loss: 1.9821 - val_a
ccuracy: 0.5248
Epoch 9/50
858/858 [==============================] - 1s 910us/step - loss: 0.8096 - accuracy: 0.7083 - val_loss: 2.0240 - val_a
ccuracy: 0.5390
Epoch 10/50
858/858 [==============================] - 1s 900us/step - loss: 0.7104 - accuracy: 0.7454 - val_loss: 2.0086 - val_a
ccuracy: 0.5159
Epoch 11/50
858/858 [==============================] - 1s 898us/step - loss: 0.6989 - accuracy: 0.7498 - val_loss: 1.9486 - val_a
ccuracy: 0.5708
```

Plot of the densely connected model with activation function: selu, optimizer: <class 'keras.src.optimizers.adam.Adam'> and learning rate: 0.01



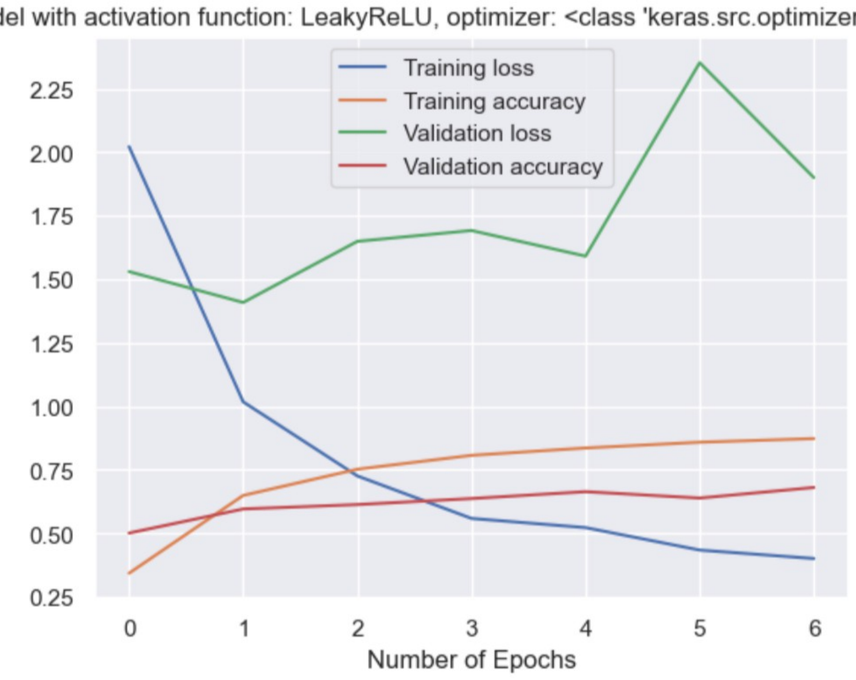
```
Epoch 1/50
858/858 [==============================] - 2s 1ms/step - loss: 1.9594 - accuracy: 0.3521 - val_loss: 1.6744 - val_acc
uracy: 0.4886
Epoch 2/50
858/858 [==============================] - 1s 1ms/step - loss: 1.0035 - accuracy: 0.6447 - val_loss: 1.4818 - val_acc
uracy: 0.5876
Epoch 3/50
858/858 [==============================] - 1s 1ms/step - loss: 0.7338 - accuracy: 0.7355 - val_loss: 1.7185 - val_acc
uracy: 0.5706
Epoch 4/50
858/858 [==============================] - 1s 1ms/step - loss: 0.6235 - accuracy: 0.7793 - val_loss: 2.1755 - val_acc
uracy: 0.5538
Epoch 5/50
858/858 [==============================] - 1s 1ms/step - loss: 0.5151 - accuracy: 0.8193 - val_loss: 1.7807 - val_acc
uracy: 0.6129
Epoch 6/50
858/858 [==============================] - 1s 1ms/step - loss: 0.4336 - accuracy: 0.8480 - val_loss: 1.8405 - val_acc
uracy: 0.6297
Epoch 7/50
858/858 [==============================] - 1s 1ms/step - loss: 0.4269 - accuracy: 0.8575 - val_loss: 2.2871 - val_acc
uracy: 0.5759
```

Plot of the densely connected model with activation function: elu, optimizer: <class 'keras.src.optimizers.nadam.Nadam'> and learning rate: 0.01



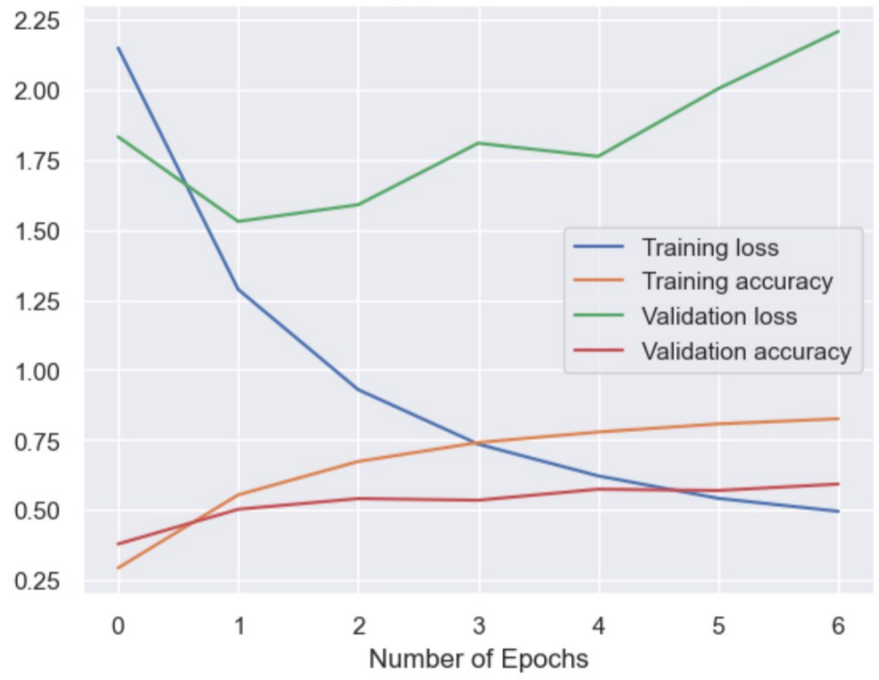

```
Epoch 1/50
858/858 [==============================] - 2s 1ms/step - loss: 2.0223 - accuracy: 0.3437 - val_loss: 1.5306 - val_acc
uracy: 0.5017
Epoch 2/50
858/858 [==============================] - 1s 1ms/step - loss: 1.0180 - accuracy: 0.6497 - val_loss: 1.4091 - val_acc
uracy: 0.5965
Epoch 3/50
858/858 [==============================] - 1s 1ms/step - loss: 0.7265 - accuracy: 0.7529 - val_loss: 1.6495 - val_acc
uracy: 0.6138
Epoch 4/50
858/858 [==============================] - 1s 1ms/step - loss: 0.5594 - accuracy: 0.8075 - val_loss: 1.6923 - val_acc
uracy: 0.6375
Epoch 5/50
858/858 [==============================] - 1s 1ms/step - loss: 0.5231 - accuracy: 0.8365 - val_loss: 1.5914 - val_acc
uracy: 0.6645
Epoch 6/50
858/858 [==============================] - 1s 1ms/step - loss: 0.4347 - accuracy: 0.8592 - val_loss: 2.3525 - val_acc
uracy: 0.6394
Epoch 7/50
858/858 [==============================] - 1s 1ms/step - loss: 0.4015 - accuracy: 0.8735 - val_loss: 1.9007 - val_acc
uracy: 0.6810
```

Plot of the densely connected model with activation function: LeakyReLU, optimizer: <class 'keras.src.optimizers.nadam.Nadam'> and learning rate: 0.01




```
Epoch 1/50
858/858 [==============================] - 2s 1ms/step - loss: 2.1494 - accuracy: 0.2932 - val_loss: 1.8321 - val_acc
uracy: 0.3787
Epoch 2/50
858/858 [==============================] - 1s 1ms/step - loss: 1.2875 - accuracy: 0.5539 - val_loss: 1.5300 - val_acc
uracy: 0.5025
Epoch 3/50
858/858 [==============================] - 1s 1ms/step - loss: 0.9296 - accuracy: 0.6732 - val_loss: 1.5897 - val_acc
uracy: 0.5407
Epoch 4/50
858/858 [==============================] - 1s 1ms/step - loss: 0.7351 - accuracy: 0.7408 - val_loss: 1.8094 - val_acc
uracy: 0.5349
Epoch 5/50
858/858 [==============================] - 1s 1ms/step - loss: 0.6217 - accuracy: 0.7783 - val_loss: 1.7623 - val_acc
uracy: 0.5742
Epoch 6/50
858/858 [==============================] - 1s 1ms/step - loss: 0.5416 - accuracy: 0.8071 - val_loss: 2.0039 - val_acc
uracy: 0.5694
Epoch 7/50
858/858 [==============================] - 1s 1ms/step - loss: 0.4952 - accuracy: 0.8256 - val_loss: 2.2083 - val_acc
uracy: 0.5926
```

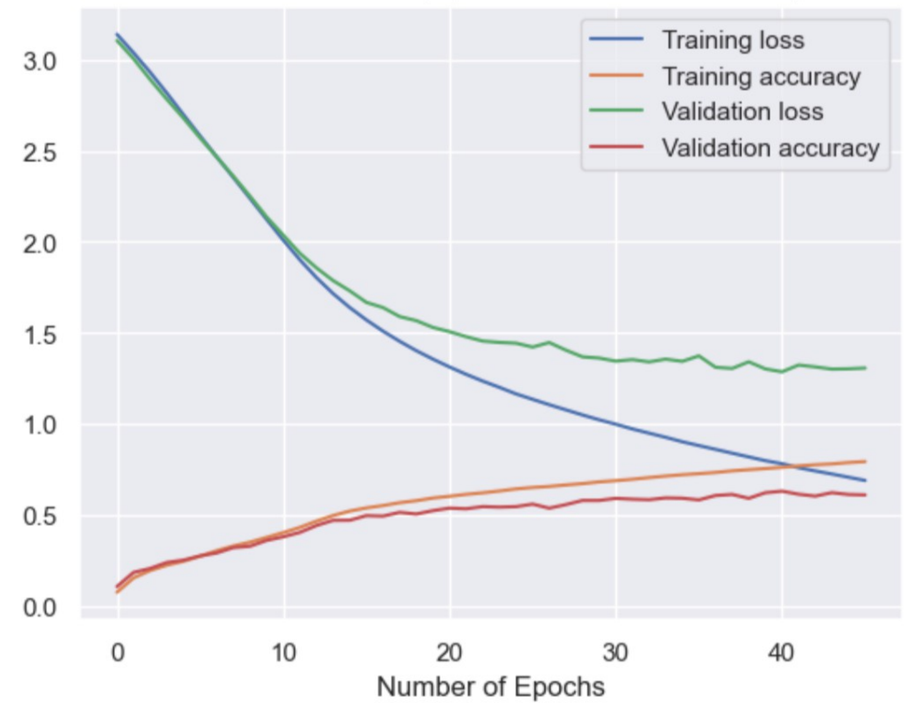
Plot of the densely connected model with activation function: selu, optimizer: <class 'keras.src.optimizers.nadam.Nadam'> and learning rate: 0.01




```
Epoch 1/50
858/858 [==============================] - 1s 836us/step - loss: 3.1387 - accuracy: 0.0767 - val_loss: 3.1048 - val_a
ccuracy: 0.1079
Epoch 2/50
858/858 [==============================] - 1s 743us/step - loss: 3.0370 - accuracy: 0.1552 - val_loss: 3.0046 - val_a
ccuracy: 0.1854
Epoch 3/50
858/858 [==============================] - 1s 749us/step - loss: 2.9299 - accuracy: 0.1958 - val_loss: 2.8906 - val_a
ccuracy: 0.2069
Epoch 4/50
858/858 [==============================] - 1s 741us/step - loss: 2.8162 - accuracy: 0.2249 - val_loss: 2.7822 - val_a
ccuracy: 0.2398
Epoch 5/50
858/858 [==============================] - 1s 781us/step - loss: 2.6974 - accuracy: 0.2466 - val_loss: 2.6789 - val_a
ccuracy: 0.2526
Epoch 6/50
858/858 [==============================] - 1s 736us/step - loss: 2.5809 - accuracy: 0.2743 - val_loss: 2.5700 - val_a
ccuracy: 0.2761
Epoch 7/50
858/858 [==============================] - 1s 734us/step - loss: 2.4664 - accuracy: 0.3045 - val_loss: 2.4651 - val_a
ccuracy: 0.2922
Epoch 8/50
858/858 [==============================] - 1s 732us/step - loss: 2.3530 - accuracy: 0.3302 - val_loss: 2.3598 - val_a
ccuracy: 0.3215
Epoch 9/50
858/858 [==============================] - 1s 731us/step - loss: 2.2383 - accuracy: 0.3512 - val_loss: 2.2501 - val_a
ccuracy: 0.3291
Epoch 10/50
858/858 [==============================] - 1s 729us/step - loss: 2.1230 - accuracy: 0.3771 - val_loss: 2.1362 - val_a
ccuracy: 0.3611
Epoch 11/50
858/858 [==============================] - 1s 726us/step - loss: 2.0094 - accuracy: 0.4037 - val_loss: 2.0367 - val_a
ccuracy: 0.3806
Epoch 12/50
858/858 [==============================] - 1s 724us/step - loss: 1.9013 - accuracy: 0.4324 - val_loss: 1.9367 - val_a
ccuracy: 0.4049
Epoch 13/50
858/858 [==============================] - 1s 725us/step - loss: 1.8031 - accuracy: 0.4667 - val_loss: 1.8568 - val_a
ccuracy: 0.4428
Epoch 14/50
858/858 [==============================] - 1s 741us/step - loss: 1.7157 - accuracy: 0.4977 - val_loss: 1.7871 - val_a
ccuracy: 0.4710
Epoch 15/50
858/858 [==============================] - 1s 724us/step - loss: 1.6391 - accuracy: 0.5220 - val_loss: 1.7308 - val_a
ccuracy: 0.4716
Epoch 16/50
858/858 [==============================] - 1s 723us/step - loss: 1.5713 - accuracy: 0.5393 - val_loss: 1.6675 - val_a
ccuracy: 0.4969
Epoch 17/50
858/858 [==============================] - 1s 725us/step - loss: 1.5097 - accuracy: 0.5522 - val_loss: 1.6391 - val_a
ccuracy: 0.4944
Epoch 18/50
858/858 [==============================] - 1s 717us/step - loss: 1.4539 - accuracy: 0.5680 - val_loss: 1.5897 - val_a
ccuracy: 0.5148
Epoch 19/50
858/858 [==============================] - 1s 717us/step - loss: 1.4026 - accuracy: 0.5790 - val_loss: 1.5674 - val_a
ccuracy: 0.5059
Epoch 20/50
858/858 [==============================] - 1s 722us/step - loss: 1.3562 - accuracy: 0.5935 - val_loss: 1.5300 - val_a
ccuracy: 0.5243
Epoch 21/50
858/858 [==============================] - 1s 714us/step - loss: 1.3129 - accuracy: 0.6031 - val_loss: 1.5068 - val_a
ccuracy: 0.5382
Epoch 22/50
858/858 [==============================] - 1s 719us/step - loss: 1.2727 - accuracy: 0.6137 - val_loss: 1.4792 - val_a
ccuracy: 0.5346
Epoch 23/50
858/858 [==============================] - 1s 718us/step - loss: 1.2353 - accuracy: 0.6221 - val_loss: 1.4551 - val_a
ccuracy: 0.5466
Epoch 24/50
858/858 [==============================] - 1s 713us/step - loss: 1.2012 - accuracy: 0.6321 - val_loss: 1.4478 - val_a
ccuracy: 0.5438
Epoch 25/50
858/858 [==============================] - 1s 716us/step - loss: 1.1652 - accuracy: 0.6437 - val_loss: 1.4435 - val_a
ccuracy: 0.5463
Epoch 26/50
858/858 [==============================] - 1s 729us/step - loss: 1.1354 - accuracy: 0.6515 - val_loss: 1.4222 - val_a
ccuracy: 0.5602
Epoch 27/50
858/858 [==============================] - 1s 716us/step - loss: 1.1058 - accuracy: 0.6572 - val_loss: 1.4470 - val_a
ccuracy: 0.5379
Epoch 28/50
858/858 [==============================] - 1s 719us/step - loss: 1.0777 - accuracy: 0.6653 - val_loss: 1.4053 - val_a
ccuracy: 0.5574
Epoch 29/50
858/858 [==============================] - 1s 719us/step - loss: 1.0496 - accuracy: 0.6727 - val_loss: 1.3685 - val_a
ccuracy: 0.5803
Epoch 30/50
858/858 [==============================] - 1s 717us/step - loss: 1.0231 - accuracy: 0.6822 - val_loss: 1.3619 - val_a
ccuracy: 0.5803
Epoch 31/50
858/858 [==============================] - 1s 714us/step - loss: 0.9987 - accuracy: 0.6888 - val_loss: 1.3450 - val_a
```

```
ccuracy: 0.5917
Epoch 32/50
858/858 [==============================] - 1s 727us/step - loss: 0.9729 - accuracy: 0.6966 - val_loss: 1.3534 - val_a
ccuracy: 0.5873
Epoch 33/50
858/858 [==============================] - 1s 721us/step - loss: 0.9497 - accuracy: 0.7057 - val_loss: 1.3408 - val_a
ccuracy: 0.5845
Epoch 34/50
858/858 [==============================] - 1s 721us/step - loss: 0.9269 - accuracy: 0.7142 - val_loss: 1.3560 - val_a
ccuracy: 0.5943
Epoch 35/50
858/858 [==============================] - 1s 720us/step - loss: 0.9026 - accuracy: 0.7216 - val_loss: 1.3438 - val_a
ccuracy: 0.5926
Epoch 36/50
858/858 [==============================] - 1s 721us/step - loss: 0.8815 - accuracy: 0.7275 - val_loss: 1.3738 - val_a
ccuracy: 0.5834
Epoch 37/50
858/858 [==============================] - 1s 717us/step - loss: 0.8608 - accuracy: 0.7340 - val_loss: 1.3112 - val_a
ccuracy: 0.6074
Epoch 38/50
858/858 [==============================] - 1s 718us/step - loss: 0.8397 - accuracy: 0.7428 - val_loss: 1.3047 - val_a
ccuracy: 0.6132
Epoch 39/50
858/858 [==============================] - 1s 733us/step - loss: 0.8193 - accuracy: 0.7493 - val_loss: 1.3413 - val_a
ccuracy: 0.5920
Epoch 40/50
858/858 [==============================] - 1s 720us/step - loss: 0.7987 - accuracy: 0.7553 - val_loss: 1.3022 - val_a
ccuracy: 0.6227
Epoch 41/50
858/858 [==============================] - 1s 719us/step - loss: 0.7821 - accuracy: 0.7620 - val_loss: 1.2864 - val_a
ccuracy: 0.6313
Epoch 42/50
858/858 [==============================] - 1s 716us/step - loss: 0.7598 - accuracy: 0.7700 - val_loss: 1.3236 - val_a
ccuracy: 0.6143
Epoch 43/50
858/858 [==============================] - 1s 716us/step - loss: 0.7423 - accuracy: 0.7762 - val_loss: 1.3133 - val_a
ccuracy: 0.6040
Epoch 44/50
858/858 [==============================] - 1s 714us/step - loss: 0.7255 - accuracy: 0.7808 - val_loss: 1.3012 - val_a
ccuracy: 0.6230
Epoch 45/50
858/858 [==============================] - 1s 715us/step - loss: 0.7078 - accuracy: 0.7883 - val_loss: 1.3028 - val_a
ccuracy: 0.6129
Epoch 46/50
858/858 [==============================] - 1s 728us/step - loss: 0.6900 - accuracy: 0.7938 - val_loss: 1.3069 - val_a
ccuracy: 0.6107
```

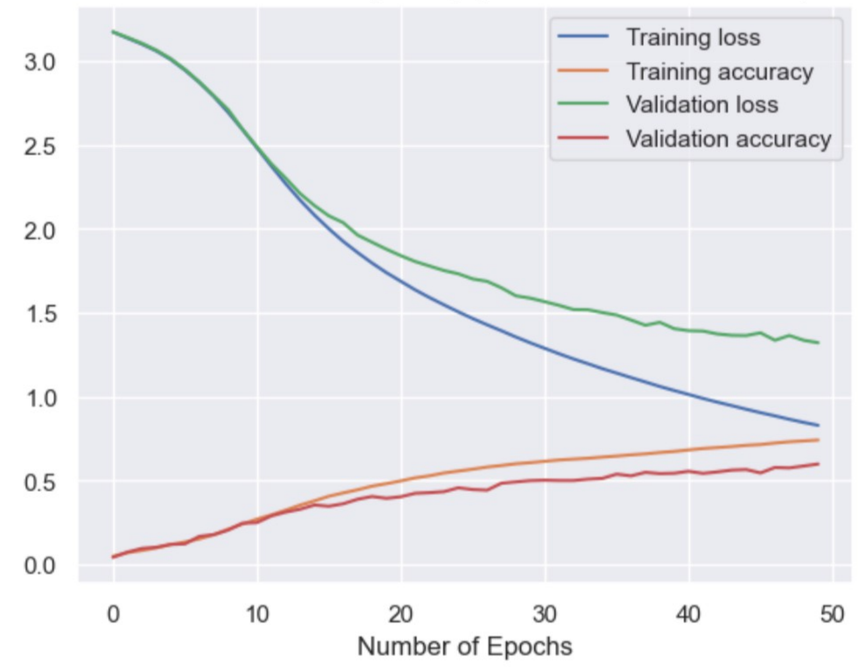
Plot of the densely connected model with activation function: elu, optimizer: <class 'keras.src.optimizers.sgd.SGD'> and learning rate: 0.001




```
Epoch 1/50
858/858 [==============================] - 1s 800us/step - loss: 3.1720 - accuracy: 0.0490 - val_loss: 3.1717 - val_a
ccuracy: 0.0427
Epoch 2/50
858/858 [==============================] - 1s 709us/step - loss: 3.1354 - accuracy: 0.0706 - val_loss: 3.1392 - val_a
ccuracy: 0.0736
Epoch 3/50
858/858 [==============================] - 1s 708us/step - loss: 3.0999 - accuracy: 0.0831 - val_loss: 3.1047 - val_a
ccuracy: 0.0956
Epoch 4/50
858/858 [==============================] - 1s 702us/step - loss: 3.0592 - accuracy: 0.0992 - val_loss: 3.0636 - val_a
ccuracy: 0.1023
Epoch 5/50
858/858 [==============================] - 1s 715us/step - loss: 3.0082 - accuracy: 0.1190 - val_loss: 3.0136 - val_a
ccuracy: 0.1196
Epoch 6/50
858/858 [==============================] - 1s 699us/step - loss: 2.9445 - accuracy: 0.1339 - val_loss: 2.9489 - val_a
ccuracy: 0.1227
Epoch 7/50
858/858 [==============================] - 1s 698us/step - loss: 2.8708 - accuracy: 0.1524 - val_loss: 2.8747 - val_a
ccuracy: 0.1676
Epoch 8/50
858/858 [==============================] - 1s 714us/step - loss: 2.7879 - accuracy: 0.1773 - val_loss: 2.7928 - val_a
ccuracy: 0.1776
Epoch 9/50
858/858 [==============================] - 1s 701us/step - loss: 2.6929 - accuracy: 0.2106 - val_loss: 2.7091 - val_a
ccuracy: 0.2047
Epoch 10/50
858/858 [==============================] - 1s 702us/step - loss: 2.5900 - accuracy: 0.2407 - val_loss: 2.5927 - val_a
ccuracy: 0.2462
Epoch 11/50
858/858 [==============================] - 1s 709us/step - loss: 2.4815 - accuracy: 0.2700 - val_loss: 2.4884 - val_a
ccuracy: 0.2496
Epoch 12/50
858/858 [==============================] - 1s 703us/step - loss: 2.3731 - accuracy: 0.2952 - val_loss: 2.3871 - val_a
ccuracy: 0.2895
Epoch 13/50
858/858 [==============================] - 1s 706us/step - loss: 2.2681 - accuracy: 0.3235 - val_loss: 2.3011 - val_a
ccuracy: 0.3123
Epoch 14/50
858/858 [==============================] - 1s 705us/step - loss: 2.1704 - accuracy: 0.3536 - val_loss: 2.2094 - val_a
ccuracy: 0.3291
Epoch 15/50
858/858 [==============================] - 1s 701us/step - loss: 2.0813 - accuracy: 0.3791 - val_loss: 2.1377 - val_a
```

```
ccuracy: 0.3544
Epoch 16/50
858/858 [==============================] - 1s 701us/step - loss: 1.9996 - accuracy: 0.4065 - val_loss: 2.0765 - val_a
ccuracy: 0.3472
Epoch 17/50
858/858 [==============================] - 1s 701us/step - loss: 1.9244 - accuracy: 0.4263 - val_loss: 2.0359 - val_a
ccuracy: 0.3617
Epoch 18/50
858/858 [==============================] - 1s 701us/step - loss: 1.8579 - accuracy: 0.4451 - val_loss: 1.9619 - val_a
ccuracy: 0.3887
Epoch 19/50
858/858 [==============================] - 1s 701us/step - loss: 1.7956 - accuracy: 0.4667 - val_loss: 1.9195 - val_a
ccuracy: 0.4052
Epoch 20/50
858/858 [==============================] - 1s 698us/step - loss: 1.7383 - accuracy: 0.4817 - val_loss: 1.8778 - val_a
ccuracy: 0.3943
Epoch 21/50
858/858 [==============================] - 1s 701us/step - loss: 1.6869 - accuracy: 0.4979 - val_loss: 1.8394 - val_a
ccuracy: 0.4035
Epoch 22/50
858/858 [==============================] - 1s 701us/step - loss: 1.6357 - accuracy: 0.5160 - val_loss: 1.8046 - val_a
ccuracy: 0.4241
Epoch 23/50
858/858 [==============================] - 1s 712us/step - loss: 1.5896 - accuracy: 0.5284 - val_loss: 1.7776 - val_a
ccuracy: 0.4278
Epoch 24/50
858/858 [==============================] - 1s 708us/step - loss: 1.5465 - accuracy: 0.5451 - val_loss: 1.7511 - val_a
ccuracy: 0.4339
Epoch 25/50
858/858 [==============================] - 1s 711us/step - loss: 1.5047 - accuracy: 0.5567 - val_loss: 1.7314 - val_a
ccuracy: 0.4568
Epoch 26/50
858/858 [==============================] - 1s 703us/step - loss: 1.4648 - accuracy: 0.5678 - val_loss: 1.7009 - val_a
ccuracy: 0.4465
Epoch 27/50
858/858 [==============================] - 1s 701us/step - loss: 1.4276 - accuracy: 0.5806 - val_loss: 1.6865 - val_a
ccuracy: 0.4428
Epoch 28/50
858/858 [==============================] - 1s 698us/step - loss: 1.3919 - accuracy: 0.5896 - val_loss: 1.6477 - val_a
ccuracy: 0.4844
Epoch 29/50
858/858 [==============================] - 1s 703us/step - loss: 1.3549 - accuracy: 0.6001 - val_loss: 1.6001 - val_a
ccuracy: 0.4922
Epoch 30/50
858/858 [==============================] - 1s 704us/step - loss: 1.3203 - accuracy: 0.6066 - val_loss: 1.5864 - val_a
ccuracy: 0.4997
Epoch 31/50
858/858 [==============================] - 1s 703us/step - loss: 1.2876 - accuracy: 0.6146 - val_loss: 1.5654 - val_a
ccuracy: 0.5022
Epoch 32/50
858/858 [==============================] - 1s 708us/step - loss: 1.2552 - accuracy: 0.6226 - val_loss: 1.5434 - val_a
ccuracy: 0.5003
Epoch 33/50
858/858 [==============================] - 1s 698us/step - loss: 1.2243 - accuracy: 0.6277 - val_loss: 1.5186 - val_a
ccuracy: 0.5003
Epoch 34/50
858/858 [==============================] - 1s 702us/step - loss: 1.1960 - accuracy: 0.6327 - val_loss: 1.5176 - val_a
ccuracy: 0.5084
Epoch 35/50
858/858 [==============================] - 1s 712us/step - loss: 1.1667 - accuracy: 0.6398 - val_loss: 1.5007 - val_a
ccuracy: 0.5128
Epoch 36/50
858/858 [==============================] - 1s 709us/step - loss: 1.1401 - accuracy: 0.6459 - val_loss: 1.4859 - val_a
ccuracy: 0.5382
Epoch 37/50
858/858 [==============================] - 1s 711us/step - loss: 1.1131 - accuracy: 0.6531 - val_loss: 1.4560 - val_a
ccuracy: 0.5279
Epoch 38/50
858/858 [==============================] - 1s 734us/step - loss: 1.0871 - accuracy: 0.6589 - val_loss: 1.4250 - val_a
ccuracy: 0.5488
Epoch 39/50
858/858 [==============================] - 1s 707us/step - loss: 1.0604 - accuracy: 0.6670 - val_loss: 1.4426 - val_a
ccuracy: 0.5416
Epoch 40/50
858/858 [==============================] - 1s 710us/step - loss: 1.0361 - accuracy: 0.6735 - val_loss: 1.4046 - val_a
ccuracy: 0.5435
Epoch 41/50
858/858 [==============================] - 1s 703us/step - loss: 1.0128 - accuracy: 0.6825 - val_loss: 1.3926 - val_a
ccuracy: 0.5552
Epoch 42/50
858/858 [==============================] - 1s 719us/step - loss: 0.9890 - accuracy: 0.6907 - val_loss: 1.3903 - val_a
ccuracy: 0.5427
Epoch 43/50
858/858 [==============================] - 1s 712us/step - loss: 0.9670 - accuracy: 0.6966 - val_loss: 1.3732 - val_a
ccuracy: 0.5516
Epoch 44/50
858/858 [==============================] - 1s 708us/step - loss: 0.9463 - accuracy: 0.7025 - val_loss: 1.3650 - val_a
ccuracy: 0.5622
Epoch 45/50
858/858 [==============================] - 1s 709us/step - loss: 0.9248 - accuracy: 0.7099 - val_loss: 1.3635 - val_a
ccuracy: 0.5655
Epoch 46/50
858/858 [==============================] - 1s 702us/step - loss: 0.9043 - accuracy: 0.7146 - val_loss: 1.3797 - val_a
ccuracy: 0.5455
Epoch 47/50
858/858 [==============================] - 1s 706us/step - loss: 0.8852 - accuracy: 0.7235 - val_loss: 1.3355 - val_a
ccuracy: 0.5778
Epoch 48/50
858/858 [==============================] - 1s 714us/step - loss: 0.8651 - accuracy: 0.7310 - val_loss: 1.3639 - val_a
ccuracy: 0.5753
Epoch 49/50
858/858 [==============================] - 1s 708us/step - loss: 0.8463 - accuracy: 0.7360 - val_loss: 1.3355 - val_a
ccuracy: 0.5867
Epoch 50/50
858/858 [==============================] - 1s 705us/step - loss: 0.8290 - accuracy: 0.7414 - val_loss: 1.3211 - val_a
ccuracy: 0.5984
```

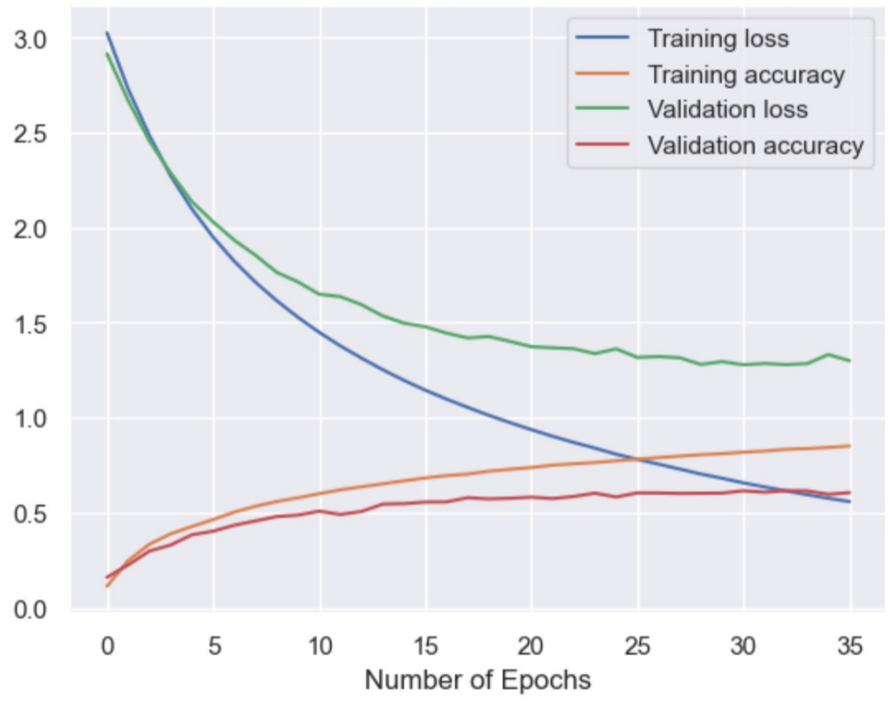
Plot of the densely connected model with activation function: LeakyReLU, optimizer: <class 'keras.src.optimizers.sgd.SGD'> and learning rate: 0.001




```
Epoch 1/50
858/858 [==============================] - 1s 828us/step - loss: 3.0242 - accuracy: 0.1139 - val_loss: 2.9138 - val_a
ccuracy: 0.1606
Epoch 2/50
858/858 [==============================] - 1s 745us/step - loss: 2.7270 - accuracy: 0.2485 - val_loss: 2.6675 - val_a
ccuracy: 0.2259
Epoch 3/50
858/858 [==============================] - 1s 744us/step - loss: 2.4836 - accuracy: 0.3344 - val_loss: 2.4550 - val_a
ccuracy: 0.2984
Epoch 4/50
858/858 [==============================] - 1s 748us/step - loss: 2.2723 - accuracy: 0.3891 - val_loss: 2.2873 - val_a
ccuracy: 0.3302
Epoch 5/50
858/858 [==============================] - 1s 751us/step - loss: 2.0979 - accuracy: 0.4275 - val_loss: 2.1366 - val_a
ccuracy: 0.3840
Epoch 6/50
858/858 [==============================] - 1s 749us/step - loss: 1.9497 - accuracy: 0.4641 - val_loss: 2.0301 - val_a
ccuracy: 0.4030
Epoch 7/50
858/858 [==============================] - 1s 747us/step - loss: 1.8226 - accuracy: 0.5030 - val_loss: 1.9336 - val_a
ccuracy: 0.4342
Epoch 8/50
858/858 [==============================] - 1s 748us/step - loss: 1.7123 - accuracy: 0.5342 - val_loss: 1.8542 - val_a
ccuracy: 0.4571
Epoch 9/50
858/858 [==============================] - 1s 747us/step - loss: 1.6153 - accuracy: 0.5590 - val_loss: 1.7646 - val_a
ccuracy: 0.4799
Epoch 10/50
858/858 [==============================] - 1s 747us/step - loss: 1.5280 - accuracy: 0.5784 - val_loss: 1.7141 - val_a
ccuracy: 0.4875
Epoch 11/50
858/858 [==============================] - 1s 748us/step - loss: 1.4492 - accuracy: 0.6001 - val_loss: 1.6495 - val_a
ccuracy: 0.5078
Epoch 12/50
858/858 [==============================] - 1s 752us/step - loss: 1.3775 - accuracy: 0.6203 - val_loss: 1.6368 - val_a
ccuracy: 0.4908
Epoch 13/50
858/858 [==============================] - 1s 748us/step - loss: 1.3131 - accuracy: 0.6364 - val_loss: 1.5943 - val_a
ccuracy: 0.5067
Epoch 14/50
858/858 [==============================] - 1s 776us/step - loss: 1.2520 - accuracy: 0.6522 - val_loss: 1.5350 - val_a
ccuracy: 0.5455
Epoch 15/50
858/858 [==============================] - 1s 749us/step - loss: 1.1961 - accuracy: 0.6677 - val_loss: 1.4973 - val_a
ccuracy: 0.5477
Epoch 16/50
858/858 [==============================] - 1s 751us/step - loss: 1.1446 - accuracy: 0.6827 - val_loss: 1.4783 - val_a
ccuracy: 0.5561
Epoch 17/50
858/858 [==============================] - 1s 764us/step - loss: 1.0979 - accuracy: 0.6950 - val_loss: 1.4450 - val_a
ccuracy: 0.5569
Epoch 18/50
858/858 [==============================] - 1s 747us/step - loss: 1.0542 - accuracy: 0.7031 - val_loss: 1.4194 - val_a
ccuracy: 0.5792
Epoch 19/50
858/858 [==============================] - 1s 748us/step - loss: 1.0121 - accuracy: 0.7188 - val_loss: 1.4270 - val_a
ccuracy: 0.5725
Epoch 20/50
858/858 [==============================] - 1s 746us/step - loss: 0.9739 - accuracy: 0.7283 - val_loss: 1.4011 - val_a
ccuracy: 0.5761
Epoch 21/50
858/858 [==============================] - 1s 751us/step - loss: 0.9371 - accuracy: 0.7377 - val_loss: 1.3732 - val_a
ccuracy: 0.5820
Epoch 22/50
858/858 [==============================] - 1s 752us/step - loss: 0.9023 - accuracy: 0.7500 - val_loss: 1.3677 - val_a
ccuracy: 0.5745
Epoch 23/50
858/858 [==============================] - 1s 749us/step - loss: 0.8697 - accuracy: 0.7574 - val_loss: 1.3624 - val_a
ccuracy: 0.5856
Epoch 24/50
858/858 [==============================] - 1s 748us/step - loss: 0.8398 - accuracy: 0.7642 - val_loss: 1.3369 - val_a
ccuracy: 0.6029
Epoch 25/50
858/858 [==============================] - 1s 752us/step - loss: 0.8079 - accuracy: 0.7732 - val_loss: 1.3620 - val_a
ccuracy: 0.5828
Epoch 26/50
858/858 [==============================] - 1s 756us/step - loss: 0.7808 - accuracy: 0.7812 - val_loss: 1.3170 - val_a
ccuracy: 0.6043
Epoch 27/50
858/858 [==============================] - 1s 759us/step - loss: 0.7553 - accuracy: 0.7901 - val_loss: 1.3213 - val_a
ccuracy: 0.6037
Epoch 28/50
858/858 [==============================] - 1s 759us/step - loss: 0.7294 - accuracy: 0.7979 - val_loss: 1.3150 - val_a
ccuracy: 0.6015
Epoch 29/50
858/858 [==============================] - 1s 747us/step - loss: 0.7035 - accuracy: 0.8052 - val_loss: 1.2800 - val_a
ccuracy: 0.6023
Epoch 30/50
858/858 [==============================] - 1s 748us/step - loss: 0.6811 - accuracy: 0.8105 - val_loss: 1.2949 - val_a
ccuracy: 0.6032
Epoch 31/50
858/858 [==============================] - 1s 765us/step - loss: 0.6569 - accuracy: 0.8186 - val_loss: 1.2784 - val_a
```

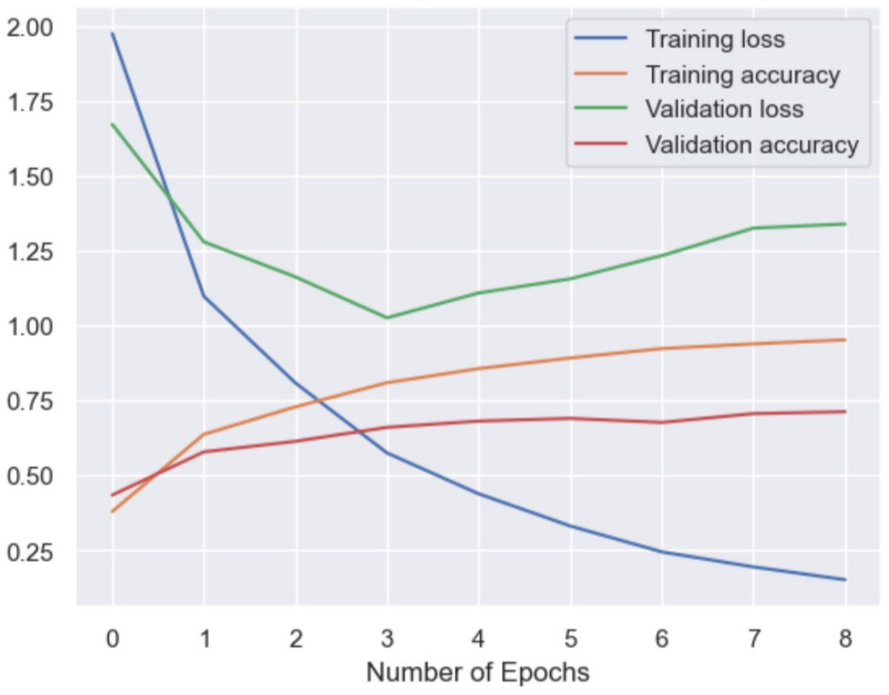
```
ccuracy: 0.6154
Epoch 32/50
858/858 [==============================] - 1s 748us/step - loss: 0.6356 - accuracy: 0.8241 - val_loss: 1.2849 - val_a
ccuracy: 0.6079
Epoch 33/50
858/858 [==============================] - 1s 749us/step - loss: 0.6154 - accuracy: 0.8335 - val_loss: 1.2789 - val_a
ccuracy: 0.6160
Epoch 34/50
858/858 [==============================] - 1s 750us/step - loss: 0.5951 - accuracy: 0.8373 - val_loss: 1.2842 - val_a
ccuracy: 0.6152
Epoch 35/50
858/858 [==============================] - 1s 761us/step - loss: 0.5761 - accuracy: 0.8435 - val_loss: 1.3318 - val_a
ccuracy: 0.5968
Epoch 36/50
858/858 [==============================] - 1s 758us/step - loss: 0.5587 - accuracy: 0.8508 - val_loss: 1.2999 - val_a
ccuracy: 0.6057
```

Plot of the densely connected model with activation function: selu, optimizer: <class 'keras.src.optimizers.sgd.SGD'> and learning rate: 0.001




```
Epoch 1/50
858/858 [==============================] - 1s 991us/step - loss: 1.9733 - accuracy: 0.3790 - val_loss: 1.6701 - val_a
ccuracy: 0.4339
Epoch 2/50
858/858 [==============================] - 1s 894us/step - loss: 1.0966 - accuracy: 0.6365 - val_loss: 1.2783 - val_a
ccuracy: 0.5781
Epoch 3/50
858/858 [==============================] - 1s 895us/step - loss: 0.8077 - accuracy: 0.7282 - val_loss: 1.1614 - val_a
ccuracy: 0.6132
Epoch 4/50
858/858 [==============================] - 1s 894us/step - loss: 0.5744 - accuracy: 0.8088 - val_loss: 1.0251 - val_a
ccuracy: 0.6598
Epoch 5/50
858/858 [==============================] - 1s 896us/step - loss: 0.4381 - accuracy: 0.8556 - val_loss: 1.1079 - val_a
ccuracy: 0.6807
Epoch 6/50
858/858 [==============================] - 1s 908us/step - loss: 0.3308 - accuracy: 0.8912 - val_loss: 1.1551 - val_a
ccuracy: 0.6896
Epoch 7/50
858/858 [==============================] - 1s 899us/step - loss: 0.2444 - accuracy: 0.9225 - val_loss: 1.2326 - val_a
ccuracy: 0.6762
Epoch 8/50
858/858 [==============================] - 1s 902us/step - loss: 0.1943 - accuracy: 0.9381 - val_loss: 1.3243 - val_a
ccuracy: 0.7055
Epoch 9/50
858/858 [==============================] - 1s 904us/step - loss: 0.1517 - accuracy: 0.9517 - val_loss: 1.3380 - val_a
ccuracy: 0.7119
```

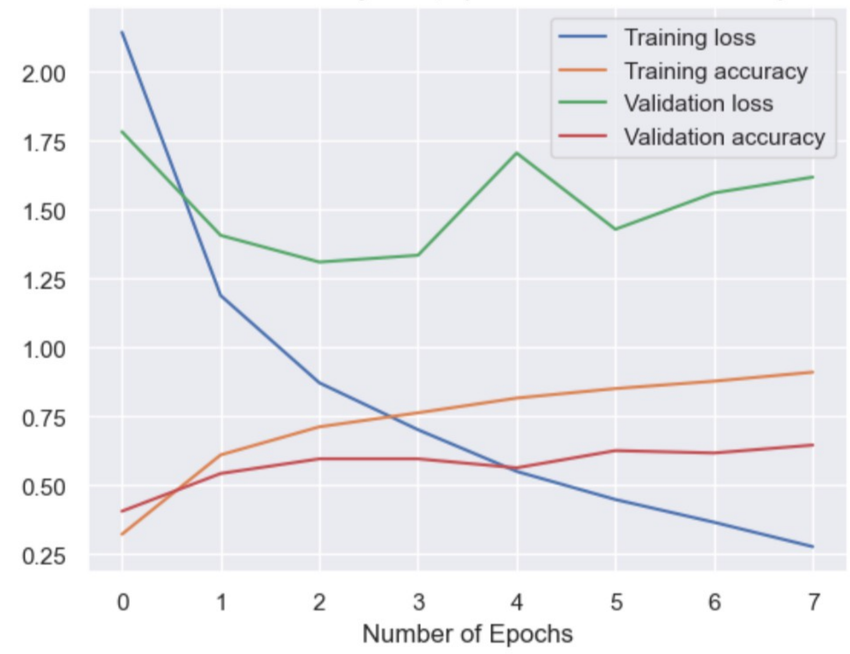
Plot of the densely connected model with activation function: elu, optimizer: <class 'keras.src.optimizers.adam.Adam'> and learning rate: 0.001




```
Epoch 1/50
858/858 [==============================] - 1s 953us/step - loss: 2.1439 - accuracy: 0.3219 - val_loss: 1.7830 - val_a
ccuracy: 0.4055
Epoch 2/50
858/858 [==============================] - 1s 872us/step - loss: 1.1885 - accuracy: 0.6099 - val_loss: 1.4069 - val_a
ccuracy: 0.5424
Epoch 3/50
858/858 [==============================] - 1s 873us/step - loss: 0.8717 - accuracy: 0.7118 - val_loss: 1.3099 - val_a
ccuracy: 0.5956
Epoch 4/50
858/858 [==============================] - 1s 865us/step - loss: 0.7014 - accuracy: 0.7627 - val_loss: 1.3346 - val_a
ccuracy: 0.5956
Epoch 5/50
858/858 [==============================] - 1s 867us/step - loss: 0.5499 - accuracy: 0.8162 - val_loss: 1.7056 - val_a
ccuracy: 0.5630
Epoch 6/50
858/858 [==============================] - 1s 867us/step - loss: 0.4480 - accuracy: 0.8507 - val_loss: 1.4289 - val_a
ccuracy: 0.6255
Epoch 7/50
858/858 [==============================] - 1s 870us/step - loss: 0.3654 - accuracy: 0.8775 - val_loss: 1.5608 - val_a
ccuracy: 0.6168
```

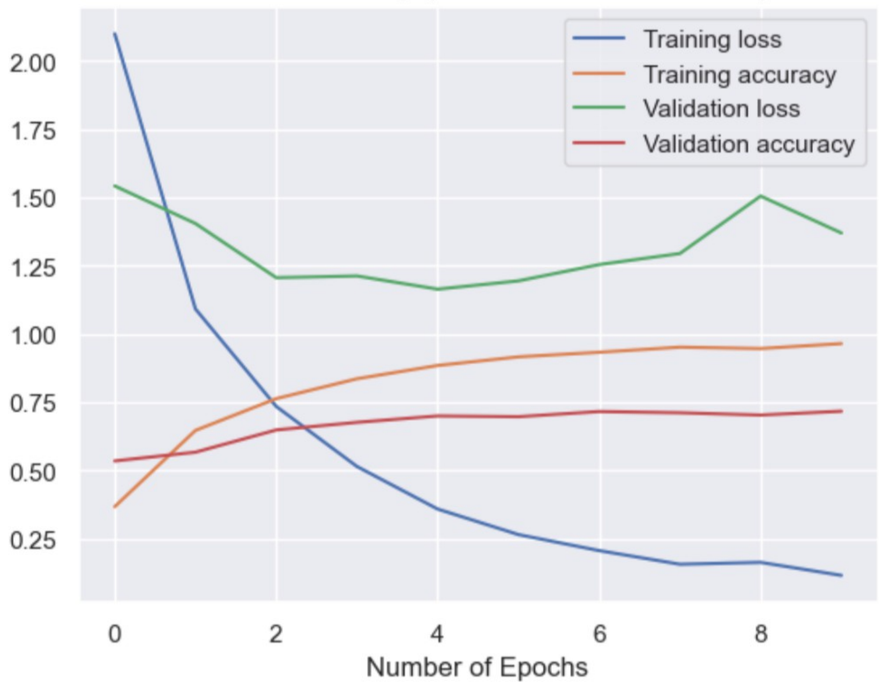
```
Epoch 8/50
858/858 [==============================] - 1s 869us/step - loss: 0.2769 - accuracy: 0.9103 - val_loss: 1.6182 - val_a
ccuracy: 0.6456
```

Plot of the densely connected model with activation function: LeakyReLU, optimizer: <class 'keras.src.optimizers.adam.Adam'> and learning rate: 0.001



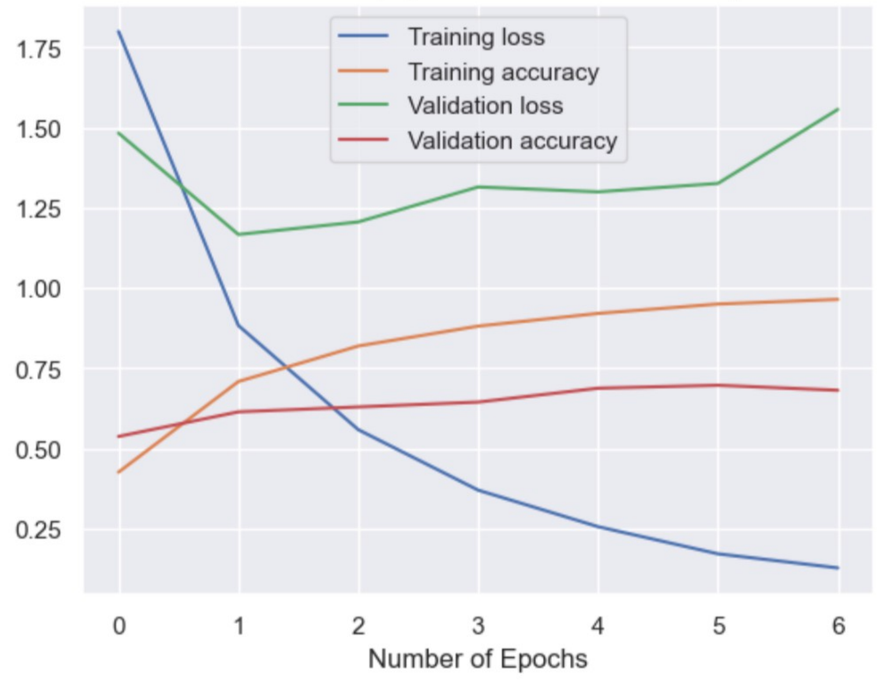

```
Epoch 1/50
858/858 [==============================] - 1s 991us/step - loss: 2.0997 - accuracy: 0.3674 - val_loss: 1.5423 - val_a
ccuracy: 0.5349
Epoch 2/50
858/858 [==============================] - 1s 908us/step - loss: 1.0914 - accuracy: 0.6467 - val_loss: 1.4045 - val_a
ccuracy: 0.5669
Epoch 3/50
858/858 [==============================] - 1s 908us/step - loss: 0.7345 - accuracy: 0.7630 - val_loss: 1.2062 - val_a
ccuracy: 0.6484
Epoch 4/50
858/858 [==============================] - 1s 909us/step - loss: 0.5145 - accuracy: 0.8359 - val_loss: 1.2125 - val_a
ccuracy: 0.6762
Epoch 5/50
858/858 [==============================] - 1s 910us/step - loss: 0.3586 - accuracy: 0.8846 - val_loss: 1.1641 - val_a
ccuracy: 0.6997
Epoch 6/50
858/858 [==============================] - 1s 915us/step - loss: 0.2650 - accuracy: 0.9162 - val_loss: 1.1945 - val_a
ccuracy: 0.6969
Epoch 7/50
858/858 [==============================] - 1s 921us/step - loss: 0.2064 - accuracy: 0.9329 - val_loss: 1.2542 - val_a
ccuracy: 0.7158
Epoch 8/50
858/858 [==============================] - 1s 913us/step - loss: 0.1564 - accuracy: 0.9517 - val_loss: 1.2946 - val_a
ccuracy: 0.7114
Epoch 9/50
858/858 [==============================] - 1s 911us/step - loss: 0.1633 - accuracy: 0.9466 - val_loss: 1.5052 - val_a
ccuracy: 0.7030
Epoch 10/50
858/858 [==============================] - 1s 912us/step - loss: 0.1156 - accuracy: 0.9647 - val_loss: 1.3700 - val_a
ccuracy: 0.7170
```

Plot of the densely connected model with activation function: selu, optimizer: <class 'keras.src.optimizers.adam.Adam'> and learning rate: 0.001



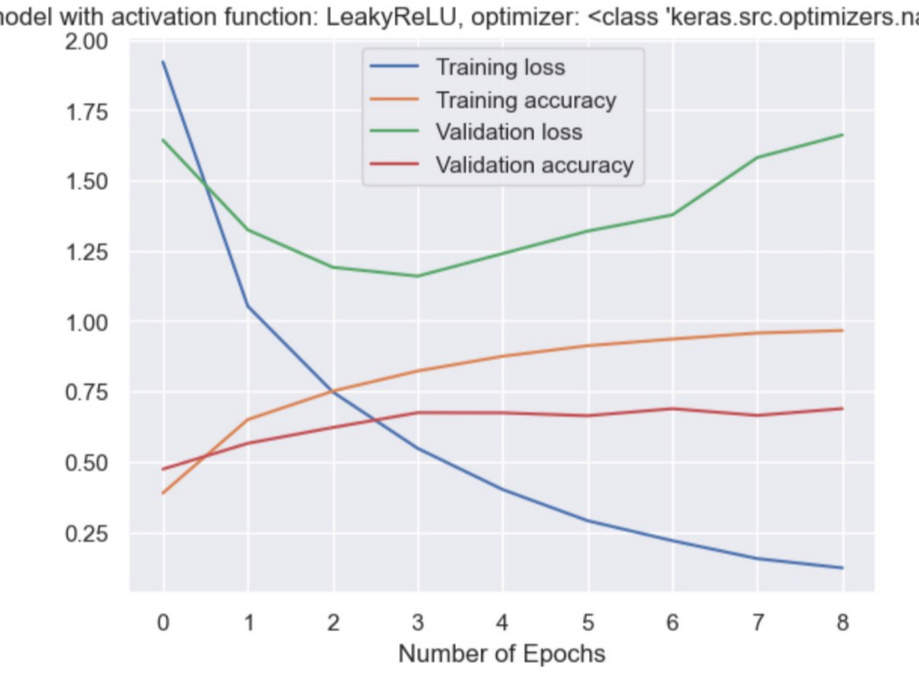

```
Epoch 1/50
858/858 [==============================] - 2s 1ms/step - loss: 1.7987 - accuracy: 0.4261 - val_loss: 1.4823 - val_acc
uracy: 0.5374
Epoch 2/50
858/858 [==============================] - 1s 1ms/step - loss: 0.8830 - accuracy: 0.7085 - val_loss: 1.1665 - val_acc
uracy: 0.6141
Epoch 3/50
858/858 [==============================] - 1s 1ms/step - loss: 0.5586 - accuracy: 0.8192 - val_loss: 1.2056 - val_acc
uracy: 0.6294
Epoch 4/50
858/858 [==============================] - 1s 1ms/step - loss: 0.3702 - accuracy: 0.8810 - val_loss: 1.3145 - val_acc
uracy: 0.6445
Epoch 5/50
858/858 [==============================] - 1s 1ms/step - loss: 0.2564 - accuracy: 0.9206 - val_loss: 1.2992 - val_acc
uracy: 0.6877
Epoch 6/50
858/858 [==============================] - 1s 1ms/step - loss: 0.1719 - accuracy: 0.9498 - val_loss: 1.3255 - val_acc
uracy: 0.6969
Epoch 7/50
858/858 [==============================] - 1s 1ms/step - loss: 0.1281 - accuracy: 0.9644 - val_loss: 1.5559 - val_acc
uracy: 0.6813
```

Plot of the densely connected model with activation function: elu, optimizer: <class 'keras.src.optimizers.nadam.Nadam'> and learning rate: 0.001

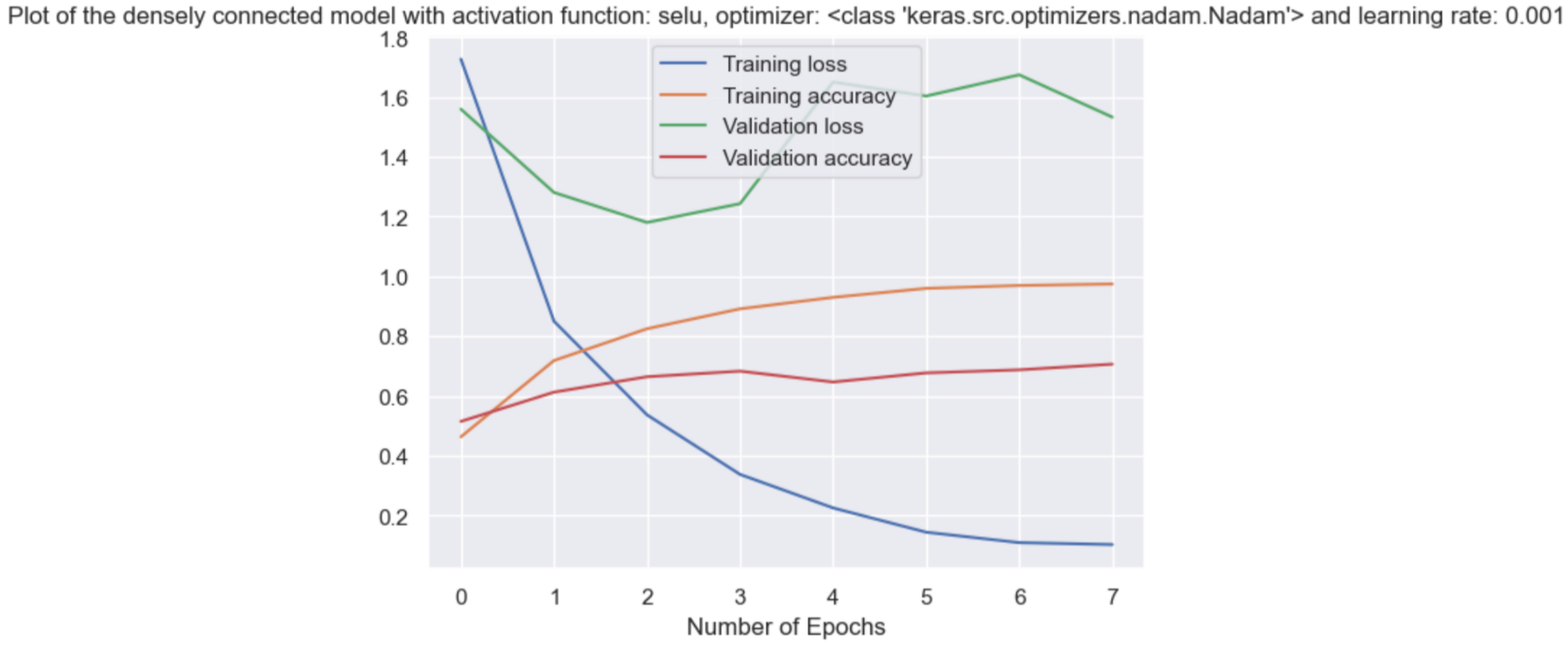



```
Epoch 1/50
858/858 [==============================] - 2s 1ms/step - loss: 1.9215 - accuracy: 0.3882 - val_loss: 1.6436 - val_acc
uracy: 0.4732
Epoch 2/50
858/858 [==============================] - 1s 986us/step - loss: 1.0526 - accuracy: 0.6496 - val_loss: 1.3244 - val_a
ccuracy: 0.5650
Epoch 3/50
858/858 [==============================] - 1s 992us/step - loss: 0.7477 - accuracy: 0.7508 - val_loss: 1.1912 - val_a
ccuracy: 0.6213
Epoch 4/50
858/858 [==============================] - 1s 1ms/step - loss: 0.5471 - accuracy: 0.8224 - val_loss: 1.1596 - val_acc
uracy: 0.6740
Epoch 5/50
858/858 [==============================] - 1s 999us/step - loss: 0.4007 - accuracy: 0.8747 - val_loss: 1.2400 - val_a
ccuracy: 0.6735
Epoch 6/50
858/858 [==============================] - 1s 1ms/step - loss: 0.2898 - accuracy: 0.9121 - val_loss: 1.3198 - val_acc
uracy: 0.6631
Epoch 7/50
858/858 [==============================] - 1s 997us/step - loss: 0.2186 - accuracy: 0.9360 - val_loss: 1.3777 - val_a
ccuracy: 0.6882
Epoch 8/50
858/858 [==============================] - 1s 994us/step - loss: 0.1548 - accuracy: 0.9574 - val_loss: 1.5818 - val_a
ccuracy: 0.6642
Epoch 9/50
858/858 [==============================] - 1s 1ms/step - loss: 0.1218 - accuracy: 0.9663 - val_loss: 1.6619 - val_acc
uracy: 0.6885
```

Plot of the densely connected model with activation function: LeakyReLU, optimizer: <class 'keras.src.optimizers.nadam.Nadam'> and learning rate: 0.001




```
Epoch 1/50
858/858 [==============================] - 2s 1ms/step - loss: 1.7267 - accuracy: 0.4629 - val_loss: 1.5594 - val_acc
uracy: 0.5145
Epoch 2/50
858/858 [==============================] - 1s 1ms/step - loss: 0.8498 - accuracy: 0.7182 - val_loss: 1.2807 - val_acc
uracy: 0.6124
Epoch 3/50
858/858 [==============================] - 1s 1ms/step - loss: 0.5364 - accuracy: 0.8245 - val_loss: 1.1803 - val_acc
uracy: 0.6640
Epoch 4/50
858/858 [==============================] - 1s 1ms/step - loss: 0.3371 - accuracy: 0.8912 - val_loss: 1.2432 - val_acc
uracy: 0.6827
Epoch 5/50
858/858 [==============================] - 1s 1ms/step - loss: 0.2250 - accuracy: 0.9296 - val_loss: 1.6500 - val_acc
uracy: 0.6464
Epoch 6/50
858/858 [==============================] - 1s 1ms/step - loss: 0.1437 - accuracy: 0.9598 - val_loss: 1.6033 - val_acc
uracy: 0.6768
Epoch 7/50
858/858 [==============================] - 1s 1ms/step - loss: 0.1089 - accuracy: 0.9694 - val_loss: 1.6744 - val_acc
uracy: 0.6871
Epoch 8/50
858/858 [==============================] - 1s 1ms/step - loss: 0.1024 - accuracy: 0.9743 - val_loss: 1.5329 - val_acc
uracy: 0.7061
```



In [32]:
```
lr = []
opt = []
act = []
val_acc = []
for i in range(0, len(res_dense)):
    lr.append(res_dense[i][0])
    opt.append(res_dense[i][1])
    act.append(res_dense[i][2])
    val_acc.append(res_dense[i][3])

print('Table of the maximum validation accuracy for each combination of activation, optimizer and learning rate for the
val_dense = pd.DataFrame({'Learning rate': lr, 'Optimizer': opt, 'Activation function': act,
                          'Max validation accuracy': val_acc})
val_dense = val_dense.sort_values('Max validation accuracy')
val_dense
```

```
Table of the maximum validation accuracy for each combination of activation, optimizer and learning rate for the dens
ely connected model:
```

Out[32]:

	Learning rate	Optimizer	Activation function	Max validation accuracy
3	0.010	<class 'keras.src.optimizers.adam.Adam'>	elu	0.518963
5	0.010	<class 'keras.src.optimizers.adam.Adam'>	selu	0.570831
8	0.010	<class 'keras.src.optimizers.nadam.Nadam'>	selu	0.592582
10	0.001	<class 'keras.src.optimizers.sgd.SGD'>	LeakyReLU	0.598438
11	0.001	<class 'keras.src.optimizers.sgd.SGD'>	selu	0.616007
6	0.010	<class 'keras.src.optimizers.nadam.Nadam'>	elu	0.629671
9	0.001	<class 'keras.src.optimizers.sgd.SGD'>	elu	0.631344
13	0.001	<class 'keras.src.optimizers.adam.Adam'>	LeakyReLU	0.645566
2	0.010	<class 'keras.src.optimizers.sgd.SGD'>	selu	0.678193
7	0.010	<class 'keras.src.optimizers.nadam.Nadam'>	LeakyReLU	0.680982
16	0.001	<class 'keras.src.optimizers.nadam.Nadam'>	LeakyReLU	0.688511
15	0.001	<class 'keras.src.optimizers.nadam.Nadam'>	elu	0.696877
17	0.001	<class 'keras.src.optimizers.nadam.Nadam'>	selu	0.706079
4	0.010	<class 'keras.src.optimizers.adam.Adam'>	LeakyReLU	0.709426
12	0.001	<class 'keras.src.optimizers.adam.Adam'>	elu	0.711935
1	0.010	<class 'keras.src.optimizers.sgd.SGD'>	LeakyReLU	0.712493
14	0.001	<class 'keras.src.optimizers.adam.Adam'>	selu	0.716955
0	0.010	<class 'keras.src.optimizers.sgd.SGD'>	elu	0.725878

In general, the maximum validation accuracy scores of these densely connected models range from about 0.51 to 0.73. Additionally, all of these densely connected models can be overfitting since the differences between their training accuracy and their validation accuracy are quite large (from around 0.2 to 0.3). According to the table of the maximum validation accuracy for each combination of activation function, optimizer and learning rate above, the model which has the highest validation accuracy is a densely connected model with Elu activation function, its optimizer is Stochastic Gradient Descent with learning rate of 0.01. The maximum validation accuracy of this model is about 0.73. However, according to this model's plot of performance, its training accuracy is about 0.925, hence there is a large difference between its training accuracy and its validation accuracy (nearly 0.2), which can indicate that this model can be overfitting. Therefore, some methods such as Regularization, Weight initialization and Dropout regularization are used to mitigate this overfitting problem for this best densely connected model.

Apply Regularization methods on the best densely connected model

In [33]:
```
# Try using Regularization for the best densely connected model
model_dense_reg = keras.models.Sequential()
model_dense_reg.add(keras.layers.Flatten(input_shape = [28, 28, 1]))
for n in hiddensizes:
    model_dense_reg.add(keras.layers.Dense(n, activation = 'elu', kernel_regularizer=keras.regularizers.l2(0.01), bias_
model_dense_reg.add(keras.layers.Dense(24, activation = "softmax"))
model_dense_reg.compile(loss="sparse_categorical_crossentropy", optimizer=keras.optimizers.SGD(learning_rate=0.01), met
history_dense_reg = model_dense_reg.fit(X_train, y_train, epochs=50, callbacks = early_stopping_cb, validation_data=(X_
```

```
Epoch 1/50
858/858 [==============================] - 1s 883us/step - loss: 4.4097 - accuracy: 0.1903 - val_loss: 3.8642 - val_a
ccuracy: 0.2566
Epoch 2/50
858/858 [==============================] - 1s 767us/step - loss: 3.3336 - accuracy: 0.3661 - val_loss: 3.1823 - val_a
ccuracy: 0.3160
Epoch 3/50
858/858 [==============================] - 1s 755us/step - loss: 2.7651 - accuracy: 0.4489 - val_loss: 2.7213 - val_a
ccuracy: 0.4311
Epoch 4/50
858/858 [==============================] - 1s 746us/step - loss: 2.4386 - accuracy: 0.5041 - val_loss: 2.3811 - val_a
ccuracy: 0.5337
Epoch 5/50
858/858 [==============================] - 1s 742us/step - loss: 2.2347 - accuracy: 0.5324 - val_loss: 2.4190 - val_a
ccuracy: 0.4314
```