```
Epoch 6/50
858/858 [==============================] - 1s 745us/step - loss: 2.0956 - accuracy: 0.5528 - val_loss: 2.2952 - val_a
ccuracy: 0.4939
Epoch 7/50
858/858 [==============================] - 1s 744us/step - loss: 1.9923 - accuracy: 0.5703 - val_loss: 2.2566 - val_a
ccuracy: 0.4872
Epoch 8/50
858/858 [==============================] - 1s 738us/step - loss: 1.9127 - accuracy: 0.5834 - val_loss: 2.1353 - val_a
ccuracy: 0.5301
Epoch 9/50
858/858 [==============================] - 1s 743us/step - loss: 1.8525 - accuracy: 0.5965 - val_loss: 2.0467 - val_a
ccuracy: 0.5544
Epoch 10/50
858/858 [==============================] - 1s 749us/step - loss: 1.8147 - accuracy: 0.6072 - val_loss: 1.9657 - val_a
ccuracy: 0.5728
Epoch 11/50
858/858 [==============================] - 1s 743us/step - loss: 1.7740 - accuracy: 0.6170 - val_loss: 2.0392 - val_a
ccuracy: 0.5259
Epoch 12/50
858/858 [==============================] - 1s 736us/step - loss: 1.7549 - accuracy: 0.6162 - val_loss: 1.9834 - val_a
ccuracy: 0.5508
Epoch 13/50
858/858 [==============================] - 1s 742us/step - loss: 1.7356 - accuracy: 0.6211 - val_loss: 2.5880 - val_a
ccuracy: 0.4041
Epoch 14/50
858/858 [==============================] - 1s 738us/step - loss: 1.6971 - accuracy: 0.6358 - val_loss: 1.9187 - val_a
ccuracy: 0.5862
Epoch 15/50
858/858 [==============================] - 1s 745us/step - loss: 1.7018 - accuracy: 0.6343 - val_loss: 2.1135 - val_a
ccuracy: 0.5159
Epoch 16/50
858/858 [==============================] - 1s 747us/step - loss: 1.6857 - accuracy: 0.6414 - val_loss: 1.9186 - val_a
ccuracy: 0.5686
Epoch 17/50
858/858 [==============================] - 1s 741us/step - loss: 1.6772 - accuracy: 0.6405 - val_loss: 1.8645 - val_a
ccuracy: 0.6138
Epoch 18/50
858/858 [==============================] - 1s 748us/step - loss: 1.6606 - accuracy: 0.6424 - val_loss: 2.4435 - val_a
ccuracy: 0.4467
Epoch 19/50
858/858 [==============================] - 1s 746us/step - loss: 1.6460 - accuracy: 0.6514 - val_loss: 2.2045 - val_a
ccuracy: 0.4880
Epoch 20/50
858/858 [==============================] - 1s 751us/step - loss: 1.6392 - accuracy: 0.6536 - val_loss: 2.0044 - val_a
ccuracy: 0.5577
Epoch 21/50
858/858 [==============================] - 1s 760us/step - loss: 1.6299 - accuracy: 0.6547 - val_loss: 1.9098 - val_a
ccuracy: 0.5884
Epoch 22/50
858/858 [==============================] - 1s 748us/step - loss: 1.6325 - accuracy: 0.6532 - val_loss: 1.9898 - val_a
ccuracy: 0.5240
```

According to the performance of this best densely connected model after using l2 Regularizer, after epoch 22, its training accuracy is about 0.6532 and its validation accuracy is about 0.524. The difference between these two values is about 0.13. Hence, this method of l2 Regulization just can slightly reduce the overfitting problem of this model since the gap between these two values of this model before apply l2 regularization is about 0.2. However, the validation accuracy of this model after applying l2 Regularization is much lower than its original validation accuracy since the model's validation accuracy before applying this l2 regularization method is about 0.73.

In [34]:
```
# Try using Weight Initialization for the best densely connect model
model_dense_wi = keras.models.Sequential()
model_dense_wi.add(keras.layers.Flatten(input_shape = [28, 28, 1]))
model_dense_wi.add(keras.layers.Dense(64, activation = 'elu', kernel_initializer = 'he_normal'))
for n in hiddensizes[1:]:
    model_dense_wi.add(keras.layers.Dense(n, activation = 'selu'))
model_dense_wi.add(keras.layers.Dense(24, activation = "softmax"))
model_dense_wi.compile(loss="sparse_categorical_crossentropy", optimizer=keras.optimizers.SGD(learning_rate=0.01), metr
history_dense_wi = model_dense_wi.fit(X_train, y_train, epochs=50, callbacks = early_stopping_cb, validation_data=(X_va
max_val_acc_dense_wi = np.max(history_dense_wi.history['val_accuracy'])
```

```
Epoch 1/50
858/858 [==============================] - 1s 851us/step - loss: 2.4767 - accuracy: 0.2469 - val_loss: 2.0876 - val_a
ccuracy: 0.3001
Epoch 2/50
858/858 [==============================] - 1s 747us/step - loss: 1.6745 - accuracy: 0.4679 - val_loss: 1.7099 - val_a
ccuracy: 0.4674
Epoch 3/50
858/858 [==============================] - 1s 742us/step - loss: 1.2962 - accuracy: 0.5767 - val_loss: 1.4403 - val_a
ccuracy: 0.5198
Epoch 4/50
858/858 [==============================] - 1s 760us/step - loss: 1.0809 - accuracy: 0.6441 - val_loss: 1.3226 - val_a
ccuracy: 0.5563
Epoch 5/50
858/858 [==============================] - 1s 747us/step - loss: 0.9198 - accuracy: 0.6972 - val_loss: 1.3666 - val_a
ccuracy: 0.5555
Epoch 6/50
858/858 [==============================] - 1s 752us/step - loss: 0.8026 - accuracy: 0.7372 - val_loss: 1.3986 - val_a
ccuracy: 0.5717
Epoch 7/50
858/858 [==============================] - 1s 751us/step - loss: 0.7075 - accuracy: 0.7696 - val_loss: 1.1941 - val_a
ccuracy: 0.6291
Epoch 8/50
858/858 [==============================] - 1s 746us/step - loss: 0.6068 - accuracy: 0.8032 - val_loss: 1.1448 - val_a
ccuracy: 0.6576
Epoch 9/50
858/858 [==============================] - 1s 752us/step - loss: 0.5400 - accuracy: 0.8260 - val_loss: 1.2350 - val_a
ccuracy: 0.6431
Epoch 10/50
858/858 [==============================] - 1s 759us/step - loss: 0.4729 - accuracy: 0.8477 - val_loss: 1.1406 - val_a
ccuracy: 0.6746
Epoch 11/50
858/858 [==============================] - 1s 749us/step - loss: 0.4407 - accuracy: 0.8629 - val_loss: 1.1842 - val_a
ccuracy: 0.6603
Epoch 12/50
858/858 [==============================] - 1s 771us/step - loss: 0.3714 - accuracy: 0.8840 - val_loss: 1.1757 - val_a
ccuracy: 0.6882
Epoch 13/50
858/858 [==============================] - 1s 750us/step - loss: 0.3145 - accuracy: 0.9050 - val_loss: 1.4618 - val_a
ccuracy: 0.6302
Epoch 14/50
858/858 [==============================] - 1s 755us/step - loss: 0.2825 - accuracy: 0.9168 - val_loss: 1.4035 - val_a
ccuracy: 0.6489
Epoch 15/50
858/858 [==============================] - 1s 767us/step - loss: 0.2430 - accuracy: 0.9289 - val_loss: 1.1950 - val_a
ccuracy: 0.7022
```

According to the performance of this best densely connected model after using Weight initialization method, after epoch 15, this model converges with its training accuracy is about 0.9289 and its validation accuracy is about 0.7022. The difference between these two values is about 0.23. Hence, this method of Weight initialization does not reduce the gap between the training accuracy and the validation accuracy of the model since the difference between these two values before applying Weight initialization is about 0.2.

In [35]:
```
# Try using Dropout Regularization for the best densely connect model
model_dense_dr = keras.models.Sequential()
model_dense_dr.add(keras.layers.Flatten(input_shape = [28, 28, 1]))
for n in hiddensizes:
    model_dense_dr.add(keras.layers.Dense(n, activation = 'elu'))
    model_dense_dr.add(keras.layers.Dropout (0.05))
model_dense_dr.add(keras.layers.Dense(24, activation = "softmax"))
model_dense_dr.compile(loss="sparse_categorical_crossentropy", optimizer=keras.optimizers.SGD(learning_rate=0.01), metr
history_dense_dr = model_dense_dr.fit(X_train, y_train, epochs=50, callbacks = early_stopping_cb, validation_data=(X_va
max_val_acc_dense_dr = np.max(history_dense_dr.history['val_accuracy'])
```

```
Epoch 1/50
858/858 [==============================] - 1s 894us/step - loss: 2.9185 - accuracy: 0.1554 - val_loss: 2.6923 - val_a
ccuracy: 0.2181
Epoch 2/50
858/858 [==============================] - 1s 801us/step - loss: 2.2142 - accuracy: 0.3331 - val_loss: 1.9673 - val_a
ccuracy: 0.3879
Epoch 3/50
858/858 [==============================] - 1s 837us/step - loss: 1.7477 - accuracy: 0.4443 - val_loss: 1.7352 - val_a
ccuracy: 0.4526
Epoch 4/50
858/858 [==============================] - 1s 809us/step - loss: 1.4906 - accuracy: 0.5182 - val_loss: 1.4266 - val_a
ccuracy: 0.5694
Epoch 5/50
858/858 [==============================] - 1s 822us/step - loss: 1.3086 - accuracy: 0.5742 - val_loss: 1.3576 - val_a
ccuracy: 0.5904
Epoch 6/50
858/858 [==============================] - 1s 814us/step - loss: 1.1739 - accuracy: 0.6153 - val_loss: 1.3680 - val_a
ccuracy: 0.5669
Epoch 7/50
858/858 [==============================] - 1s 848us/step - loss: 1.0610 - accuracy: 0.6532 - val_loss: 1.2966 - val_a
ccuracy: 0.5973
Epoch 8/50
858/858 [==============================] - 1s 815us/step - loss: 0.9655 - accuracy: 0.6806 - val_loss: 1.2189 - val_a
ccuracy: 0.6037
Epoch 9/50
858/858 [==============================] - 1s 810us/step - loss: 0.8752 - accuracy: 0.7104 - val_loss: 1.1290 - val_a
ccuracy: 0.6617
Epoch 10/50
858/858 [==============================] - 1s 815us/step - loss: 0.8072 - accuracy: 0.7341 - val_loss: 1.0966 - val_a
ccuracy: 0.6347
Epoch 11/50
858/858 [==============================] - 1s 808us/step - loss: 0.7452 - accuracy: 0.7521 - val_loss: 1.0968 - val_a
ccuracy: 0.6492
Epoch 12/50
858/858 [==============================] - 1s 810us/step - loss: 0.6987 - accuracy: 0.7675 - val_loss: 1.1331 - val_a
ccuracy: 0.6598
Epoch 13/50
858/858 [==============================] - 1s 818us/step - loss: 0.6528 - accuracy: 0.7821 - val_loss: 1.0886 - val_a
ccuracy: 0.6662
Epoch 14/50
858/858 [==============================] - 1s 806us/step - loss: 0.5888 - accuracy: 0.8040 - val_loss: 1.0324 - val_a
ccuracy: 0.7002
Epoch 15/50
858/858 [==============================] - 1s 830us/step - loss: 0.5649 - accuracy: 0.8106 - val_loss: 1.1694 - val_a
ccuracy: 0.6718
Epoch 16/50
858/858 [==============================] - 1s 810us/step - loss: 0.5300 - accuracy: 0.8212 - val_loss: 1.0503 - val_a
ccuracy: 0.7223
Epoch 17/50
858/858 [==============================] - 1s 805us/step - loss: 0.4836 - accuracy: 0.8381 - val_loss: 1.0324 - val_a
ccuracy: 0.7306
Epoch 18/50
858/858 [==============================] - 1s 812us/step - loss: 0.4562 - accuracy: 0.8448 - val_loss: 1.4071 - val_a
ccuracy: 0.6202
Epoch 19/50
858/858 [==============================] - 1s 814us/step - loss: 0.4384 - accuracy: 0.8548 - val_loss: 1.0183 - val_a
ccuracy: 0.7231
Epoch 20/50
858/858 [==============================] - 1s 815us/step - loss: 0.4091 - accuracy: 0.8633 - val_loss: 1.0445 - val_a
ccuracy: 0.7181
Epoch 21/50
858/858 [==============================] - 1s 814us/step - loss: 0.3818 - accuracy: 0.8733 - val_loss: 1.1317 - val_a
ccuracy: 0.7131
Epoch 22/50
858/858 [==============================] - 1s 820us/step - loss: 0.3630 - accuracy: 0.8774 - val_loss: 1.0987 - val_a
ccuracy: 0.7292
Epoch 23/50
858/858 [==============================] - 1s 808us/step - loss: 0.3477 - accuracy: 0.8833 - val_loss: 1.2303 - val_a
ccuracy: 0.6704
Epoch 24/50
858/858 [==============================] - 1s 820us/step - loss: 0.3210 - accuracy: 0.8912 - val_loss: 1.0704 - val_a
ccuracy: 0.7292
```

According to the performance of this best densely connected model after using Dropout regularization method, after epoch 24, this model converges with its training accuracy is about 0.8912 and its validation accuracy is about 0.7292. The difference between these two values is about 0.16. Hence, this method of Dropout regulation also just slightly reduce the gap between the training accuracy and the validation accuracy of the model since the gap between these two values of this model before apply Dropout regularization is about 0.2, but it does not improve the validation accuracy much.

3.2 Train and optimize a Convolutional Neural Network (CNN) model

The CNN model is built with Convolutional layers 3x3, strides equal to 1, Max Pooling layers 2x2, 1 Flatten layer, 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). Early stopping is also defined with patience of 5, and monitored by validation loss values. Early stopping is used to prevent model from overfitting.

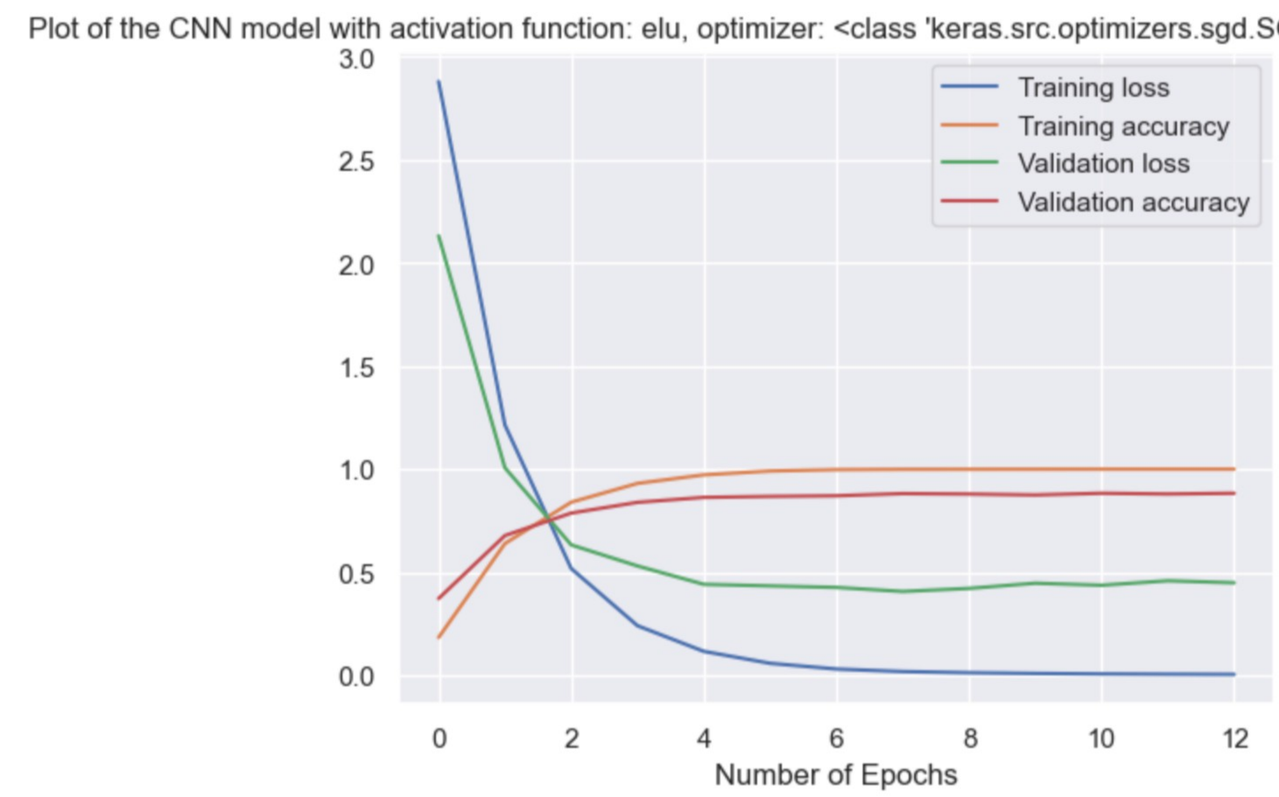
In [37]:
```
hiddensizes = [64,32,16]
batch_size = 32
n_epochs = 50

def model_cnn_factory(hiddensizes, actfn, optimizer, learningrate=0):
    model = keras.models.Sequential()
    model.add(keras.layers.Conv2D(filters=hiddensizes[0], kernel_size=3, strides=1, activation=actfn, padding="same",
                                  input_shape=[28, 28, 1]))
    model.add(keras.layers.MaxPooling2D(pool_size=2))
    for n in hiddensizes[1:-1]:
        model.add(keras.layers.Conv2D(filters=n, kernel_size=3, strides=1, padding="same", activation=actfn))
        model.add(keras.layers.MaxPooling2D(pool_size=2))
    model.add(keras.layers.Conv2D(filters=hiddensizes[-1], kernel_size=3, strides=1, padding="same",
                                  activation=actfn))
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(24, activation = "softmax"))
    model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer(learning_rate=learningrate),
                  metrics=["accuracy"])
    return model

def build_cnn(hiddensizes, actfn, optimizer, learningrate, n_epochs, batch_size, further_callbacks=[]):
    if further_callbacks != []:
        callbacks = further_callbacks
    else:
        callbacks = [early_stopping_cb]
    model = model_cnn_factory(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)

res_cnn = []
for i in learningrate:
    for o in optimizer:
        for a in actfn:
            max_val_acc_cnn, history_cnn, model_cnn = build_cnn(hiddensizes, a, o, i, n_epochs, batch_size)
            pd.DataFrame(history_cnn.history).plot()
            plt.xlabel("Number of Epochs")
            plt.title(f'Plot of the CNN model with activation function: {a}, optimizer: {o} and learning rate: {i}')
            plt.legend(['Training loss', 'Training accuracy', 'Validation loss', 'Validation accuracy'])
            plt.show()
            res_cnn += [[i, o, a, max_val_acc_cnn]]
```

```
Epoch 1/50
858/858 [==============================] - 16s 18ms/step - loss: 2.8795 - accuracy: 0.1839 - val_loss: 2.1314 - val_a
ccuracy: 0.3726
Epoch 2/50
858/858 [==============================] - 15s 18ms/step - loss: 1.2141 - accuracy: 0.6392 - val_loss: 1.0068 - val_a
ccuracy: 0.6779
Epoch 3/50
858/858 [==============================] - 16s 18ms/step - loss: 0.5182 - accuracy: 0.8397 - val_loss: 0.6327 - val_a
ccuracy: 0.7869
Epoch 4/50
858/858 [==============================] - 16s 18ms/step - loss: 0.2408 - accuracy: 0.9309 - val_loss: 0.5305 - val_a
ccuracy: 0.8391
Epoch 5/50
858/858 [==============================] - 15s 18ms/step - loss: 0.1165 - accuracy: 0.9722 - val_loss: 0.4416 - val_a
ccuracy: 0.8631
Epoch 6/50
858/858 [==============================] - 15s 18ms/step - loss: 0.0585 - accuracy: 0.9910 - val_loss: 0.4339 - val_a
ccuracy: 0.8678
Epoch 7/50
858/858 [==============================] - 15s 18ms/step - loss: 0.0309 - accuracy: 0.9975 - val_loss: 0.4265 - val_a
ccuracy: 0.8709
Epoch 8/50
858/858 [==============================] - 16s 18ms/step - loss: 0.0186 - accuracy: 0.9992 - val_loss: 0.4069 - val_a
ccuracy: 0.8815
Epoch 9/50
858/858 [==============================] - 16s 18ms/step - loss: 0.0129 - accuracy: 0.9995 - val_loss: 0.4217 - val_a
ccuracy: 0.8793
Epoch 10/50
858/858 [==============================] - 15s 18ms/step - loss: 0.0096 - accuracy: 0.9997 - val_loss: 0.4469 - val_a
ccuracy: 0.8748
Epoch 11/50
858/858 [==============================] - 15s 18ms/step - loss: 0.0071 - accuracy: 0.9999 - val_loss: 0.4371 - val_a
ccuracy: 0.8832
Epoch 12/50
858/858 [==============================] - 15s 18ms/step - loss: 0.0058 - accuracy: 0.9999 - val_loss: 0.4587 - val_a
ccuracy: 0.8795
Epoch 13/50
858/858 [==============================] - 16s 18ms/step - loss: 0.0048 - accuracy: 0.9999 - val_loss: 0.4490 - val_a
ccuracy: 0.8832
```



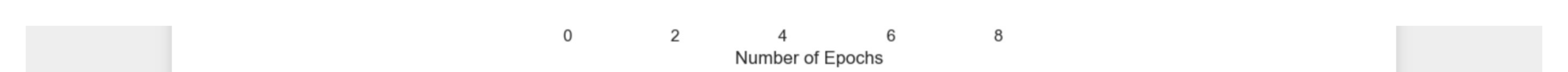

```
Epoch 1/50
858/858 [==============================] - 14s 16ms/step - loss: 3.1578 - accuracy: 0.0720 - val_loss: 3.1513 - val_a
ccuracy: 0.0485
Epoch 2/50
858/858 [==============================] - 14s 16ms/step - loss: 2.2905 - accuracy: 0.3265 - val_loss: 1.6336 - val_a
```

```
ccuracy: 0.4808
Epoch 3/50
858/858 [==============================] - 14s 16ms/step - loss: 1.0229 - accuracy: 0.6828 - val_loss: 0.9674 - val_a
ccuracy: 0.6609
Epoch 4/50
858/858 [==============================] - 14s 17ms/step - loss: 0.5568 - accuracy: 0.8263 - val_loss: 0.7815 - val_a
ccuracy: 0.7423
Epoch 5/50
858/858 [==============================] - 14s 16ms/step - loss: 0.3191 - accuracy: 0.9027 - val_loss: 0.7051 - val_a
ccuracy: 0.7613
Epoch 6/50
858/858 [==============================] - 14s 16ms/step - loss: 0.1706 - accuracy: 0.9516 - val_loss: 0.6253 - val_a
ccuracy: 0.8062
Epoch 7/50
858/858 [==============================] - 14s 16ms/step - loss: 0.0908 - accuracy: 0.9794 - val_loss: 0.6540 - val_a
ccuracy: 0.8148
Epoch 8/50
858/858 [==============================] - 14s 16ms/step - loss: 0.0512 - accuracy: 0.9919 - val_loss: 0.6248 - val_a
ccuracy: 0.8274
Epoch 9/50
858/858 [==============================] - 14s 16ms/step - loss: 0.0274 - accuracy: 0.9965 - val_loss: 0.6562 - val_a
ccuracy: 0.8299
Epoch 10/50
858/858 [==============================] - 14s 16ms/step - loss: 0.0156 - accuracy: 0.9993 - val_loss: 0.7015 - val_a
ccuracy: 0.8224
Epoch 11/50
858/858 [==============================] - 14s 16ms/step - loss: 0.0129 - accuracy: 0.9991 - val_loss: 0.7106 - val_a
ccuracy: 0.8249
Epoch 12/50
858/858 [==============================] - 14s 16ms/step - loss: 0.0081 - accuracy: 0.9997 - val_loss: 0.7100 - val_a
ccuracy: 0.8346
Epoch 13/50
858/858 [==============================] - 14s 16ms/step - loss: 0.0114 - accuracy: 0.9984 - val_loss: 0.6785 - val_a
ccuracy: 0.8360
```

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

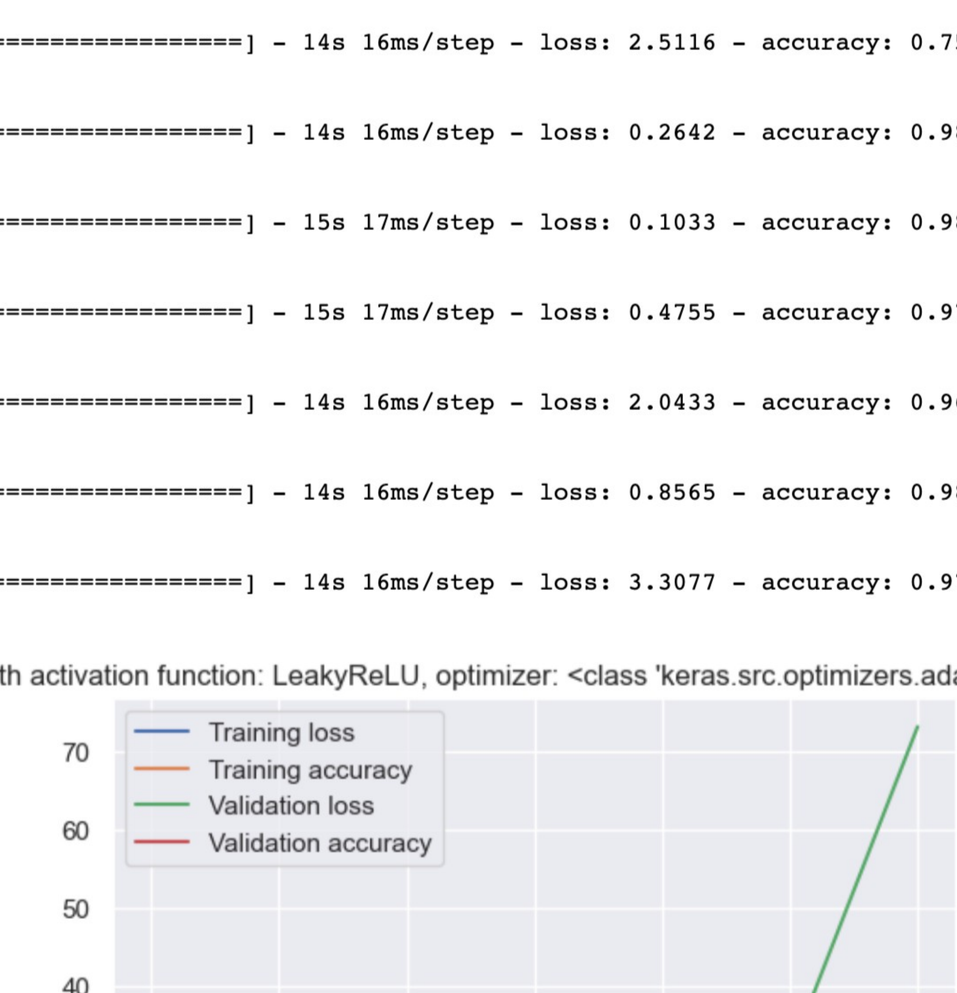

```
Epoch 1/50
858/858 [==============================] - 16s 18ms/step - loss: 1.9353 - accuracy: 0.4421 - val_loss: 1.1185 - val_a
ccuracy: 0.6266
Epoch 2/50
858/858 [==============================] - 16s 19ms/step - loss: 0.5696 - accuracy: 0.8263 - val_loss: 0.6801 - val_a
ccuracy: 0.7828
Epoch 3/50
858/858 [==============================] - 16s 18ms/step - loss: 0.2442 - accuracy: 0.9330 - val_loss: 0.4633 - val_a
ccuracy: 0.8475
Epoch 4/50
858/858 [==============================] - 16s 18ms/step - loss: 0.1092 - accuracy: 0.9783 - val_loss: 0.4190 - val_a
ccuracy: 0.8639
Epoch 5/50
858/858 [==============================] - 16s 18ms/step - loss: 0.0548 - accuracy: 0.9941 - val_loss: 0.3846 - val_a
ccuracy: 0.8765
Epoch 6/50
858/858 [==============================] - 16s 18ms/step - loss: 0.0311 - accuracy: 0.9981 - val_loss: 0.4136 - val_a
ccuracy: 0.8834
Epoch 7/50
858/858 [==============================] - 16s 19ms/step - loss: 0.0195 - accuracy: 0.9995 - val_loss: 0.4007 - val_a
ccuracy: 0.8898
Epoch 8/50
858/858 [==============================] - 16s 18ms/step - loss: 0.0135 - accuracy: 0.9998 - val_loss: 0.3997 - val_a
ccuracy: 0.8921
Epoch 9/50
858/858 [==============================] - 16s 19ms/step - loss: 0.0101 - accuracy: 0.9998 - val_loss: 0.4082 - val_a
ccuracy: 0.9004
Epoch 10/50
858/858 [==============================] - 16s 18ms/step - loss: 0.0079 - accuracy: 0.9998 - val_loss: 0.4149 - val_a
ccuracy: 0.8893
```

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



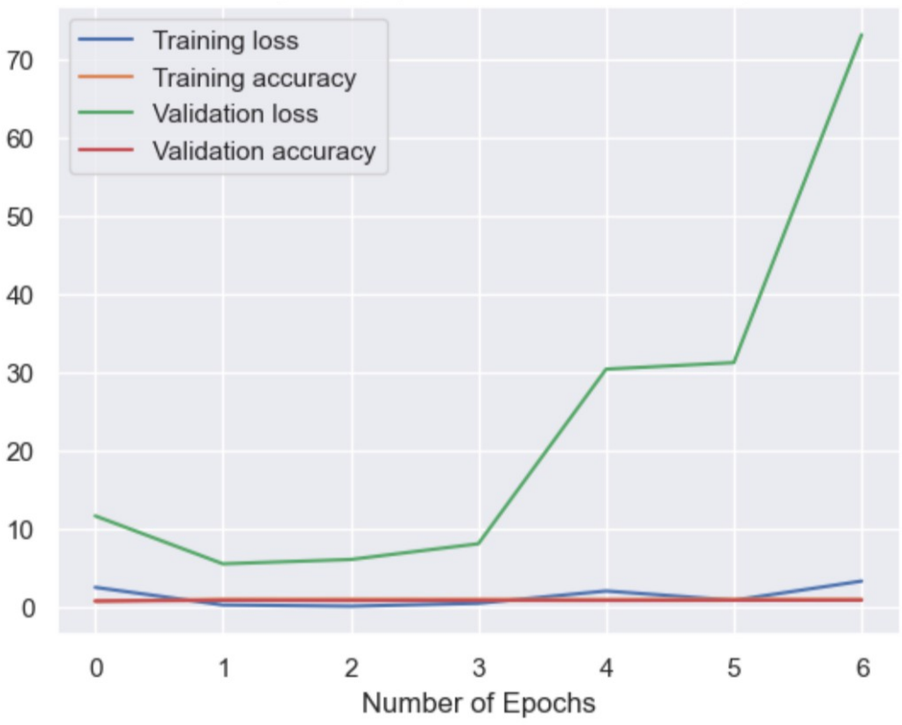

```
Epoch 1/50
858/858 [==============================] - 17s 19ms/step - loss: 1.3477 - accuracy: 0.8103 - val_loss: 3.1771 - val_a
ccuracy: 0.8779
Epoch 2/50
858/858 [==============================] - 16s 19ms/step - loss: 0.2401 - accuracy: 0.9822 - val_loss: 7.7578 - val_a
ccuracy: 0.8349
Epoch 3/50
858/858 [==============================] - 16s 19ms/step - loss: 1.0093 - accuracy: 0.9658 - val_loss: 13.6655 - val_
accuracy: 0.8330
Epoch 4/50
858/858 [==============================] - 16s 19ms/step - loss: 1.0300 - accuracy: 0.9786 - val_loss: 20.6149 - val_
accuracy: 0.8611
Epoch 5/50
858/858 [==============================] - 17s 19ms/step - loss: 0.6397 - accuracy: 0.9903 - val_loss: 31.1856 - val_
accuracy: 0.8862
Epoch 6/50
858/858 [==============================] - 16s 19ms/step - loss: 1.6665 - accuracy: 0.9851 - val_loss: 36.6808 - val_
accuracy: 0.8745
```

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




```
Epoch 1/50
858/858 [==============================] - 14s 16ms/step - loss: 2.5116 - accuracy: 0.7588 - val_loss: 11.6346 - val_
accuracy: 0.7557
Epoch 2/50
858/858 [==============================] - 14s 16ms/step - loss: 0.2642 - accuracy: 0.9811 - val_loss: 5.5127 - val_a
ccuracy: 0.8578
Epoch 3/50
858/858 [==============================] - 15s 17ms/step - loss: 0.1033 - accuracy: 0.9890 - val_loss: 6.0678 - val_a
ccuracy: 0.8271
Epoch 4/50
858/858 [==============================] - 15s 17ms/step - loss: 0.4755 - accuracy: 0.9721 - val_loss: 8.0769 - val_a
ccuracy: 0.8514
Epoch 5/50
858/858 [==============================] - 14s 16ms/step - loss: 2.0433 - accuracy: 0.9617 - val_loss: 30.3714 - val_
accuracy: 0.8397
Epoch 6/50
858/858 [==============================] - 14s 16ms/step - loss: 0.8565 - accuracy: 0.9879 - val_loss: 31.2023 - val_
accuracy: 0.8564
Epoch 7/50
858/858 [==============================] - 14s 16ms/step - loss: 3.3077 - accuracy: 0.9793 - val_loss: 73.0682 - val_
accuracy: 0.8731
```

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



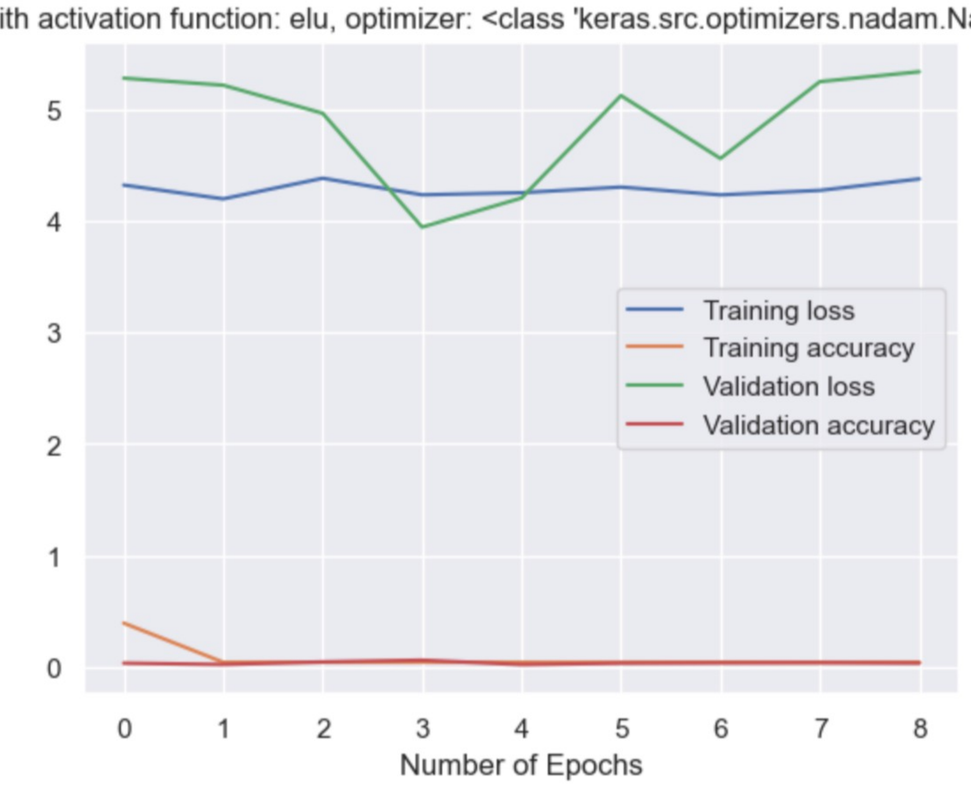

```
Epoch 1/50
858/858 [==============================] - 17s 20ms/step - loss: 9.1001 - accuracy: 0.3966 - val_loss: 6.2461 - val_a
ccuracy: 0.0368
Epoch 2/50
858/858 [==============================] - 16s 19ms/step - loss: 7.2546 - accuracy: 0.0411 - val_loss: 10.1235 - val_
accuracy: 0.0544
Epoch 3/50
858/858 [==============================] - 16s 18ms/step - loss: 7.1465 - accuracy: 0.0436 - val_loss: 5.0743 - val_a
ccuracy: 0.0435
Epoch 4/50
858/858 [==============================] - 16s 18ms/step - loss: 7.1383 - accuracy: 0.0421 - val_loss: 6.3165 - val_a
ccuracy: 0.0613
Epoch 5/50
858/858 [==============================] - 16s 18ms/step - loss: 7.5356 - accuracy: 0.0443 - val_loss: 9.4454 - val_a
ccuracy: 0.0301
Epoch 6/50
858/858 [==============================] - 16s 18ms/step - loss: 7.2638 - accuracy: 0.0412 - val_loss: 5.8915 - val_a
ccuracy: 0.0435
```

```
Epoch 7/50
858/858 [==============================] - 16s 18ms/step - loss: 7.4342 - accuracy: 0.0437 - val_loss: 9.7864 - val_a
ccuracy: 0.0307
Epoch 8/50
858/858 [==============================] - 16s 18ms/step - loss: 6.7898 - accuracy: 0.0433 - val_loss: 7.9770 - val_a
ccuracy: 0.0477
```

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

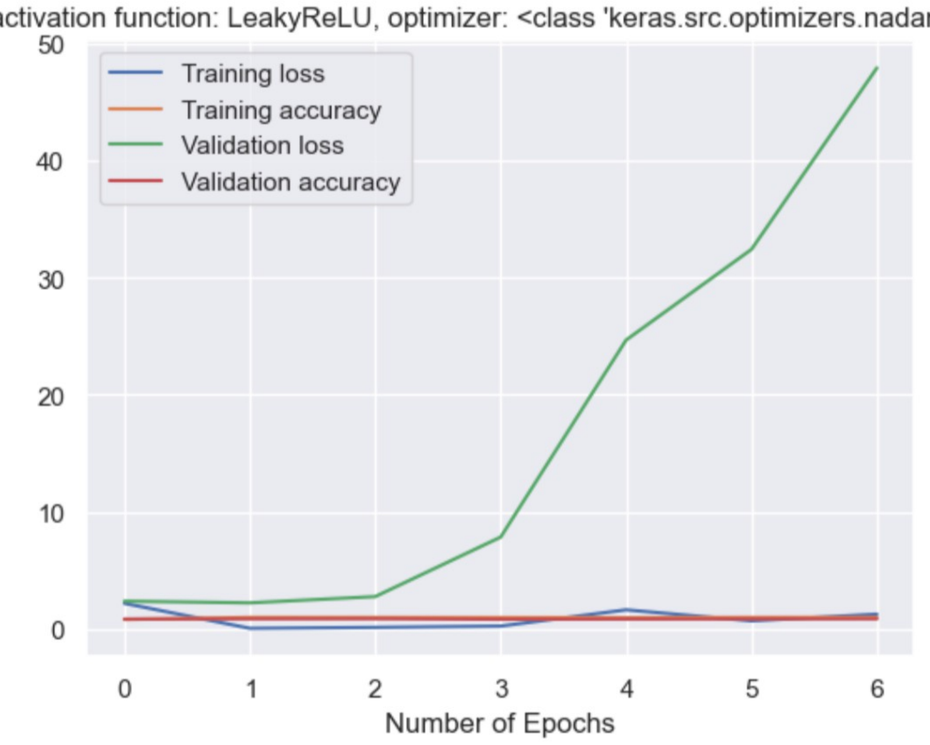

```
Epoch 1/50
858/858 [==============================] - 17s 19ms/step - loss: 4.3176 - accuracy: 0.3932 - val_loss: 5.2754 - val_a
ccuracy: 0.0346
Epoch 2/50
858/858 [==============================] - 16s 18ms/step - loss: 4.1939 - accuracy: 0.0425 - val_loss: 5.2123 - val_a
ccuracy: 0.0226
Epoch 3/50
858/858 [==============================] - 16s 18ms/step - loss: 4.3789 - accuracy: 0.0425 - val_loss: 4.9607 - val_a
ccuracy: 0.0457
Epoch 4/50
858/858 [==============================] - 16s 18ms/step - loss: 4.2300 - accuracy: 0.0430 - val_loss: 3.9408 - val_a
ccuracy: 0.0613
Epoch 5/50
858/858 [==============================] - 16s 19ms/step - loss: 4.2492 - accuracy: 0.0425 - val_loss: 4.2017 - val_a
ccuracy: 0.0215
Epoch 6/50
858/858 [==============================] - 15s 18ms/step - loss: 4.2996 - accuracy: 0.0415 - val_loss: 5.1193 - val_a
ccuracy: 0.0346
Epoch 7/50
858/858 [==============================] - 15s 18ms/step - loss: 4.2296 - accuracy: 0.0433 - val_loss: 4.5558 - val_a
ccuracy: 0.0368
Epoch 8/50
858/858 [==============================] - 15s 18ms/step - loss: 4.2698 - accuracy: 0.0409 - val_loss: 5.2444 - val_a
ccuracy: 0.0390
Epoch 9/50
858/858 [==============================] - 15s 18ms/step - loss: 4.3726 - accuracy: 0.0434 - val_loss: 5.3330 - val_a
ccuracy: 0.0346
```

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



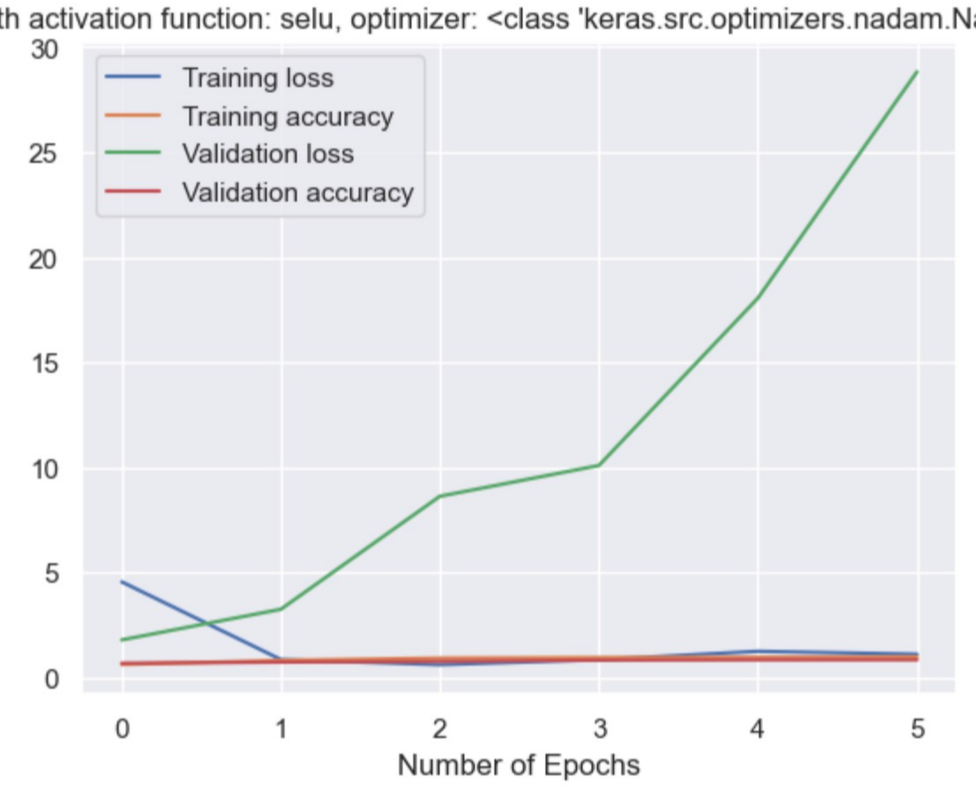

```
Epoch 1/50
858/858 [==============================] - 14s 16ms/step - loss: 2.1886 - accuracy: 0.8094 - val_loss: 2.3757 - val_a
ccuracy: 0.8489
Epoch 2/50
858/858 [==============================] - 15s 17ms/step - loss: 0.0544 - accuracy: 0.9916 - val_loss: 2.2289 - val_a
ccuracy: 0.8678
Epoch 3/50
858/858 [==============================] - 14s 16ms/step - loss: 0.1378 - accuracy: 0.9862 - val_loss: 2.7661 - val_a
ccuracy: 0.8756
Epoch 4/50
858/858 [==============================] - 15s 17ms/step - loss: 0.2461 - accuracy: 0.9819 - val_loss: 7.8486 - val_a
ccuracy: 0.8307
Epoch 5/50
858/858 [==============================] - 14s 17ms/step - loss: 1.6349 - accuracy: 0.9668 - val_loss: 24.6761 - val_
accuracy: 0.8477
Epoch 6/50
858/858 [==============================] - 14s 16ms/step - loss: 0.6927 - accuracy: 0.9905 - val_loss: 32.4254 - val_
accuracy: 0.8569
Epoch 7/50
858/858 [==============================] - 14s 16ms/step - loss: 1.2636 - accuracy: 0.9883 - val_loss: 47.9220 - val_
accuracy: 0.8675
```

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




```
Epoch 1/50
858/858 [==============================] - 16s 18ms/step - loss: 4.5521 - accuracy: 0.6202 - val_loss: 1.7903 - val_a
ccuracy: 0.6788
Epoch 2/50
858/858 [==============================] - 16s 18ms/step - loss: 0.8560 - accuracy: 0.8152 - val_loss: 3.2479 - val_a
ccuracy: 0.7345
Epoch 3/50
858/858 [==============================] - 17s 20ms/step - loss: 0.6119 - accuracy: 0.9252 - val_loss: 8.6254 - val_a
ccuracy: 0.7705
Epoch 4/50
858/858 [==============================] - 17s 19ms/step - loss: 0.8456 - accuracy: 0.9500 - val_loss: 10.0914 - val_
accuracy: 0.8293
Epoch 5/50
858/858 [==============================] - 16s 19ms/step - loss: 1.2485 - accuracy: 0.9625 - val_loss: 18.0585 - val_
accuracy: 0.8424
Epoch 6/50
858/858 [==============================] - 16s 19ms/step - loss: 1.1085 - accuracy: 0.9769 - val_loss: 28.8274 - val_
accuracy: 0.8447
```

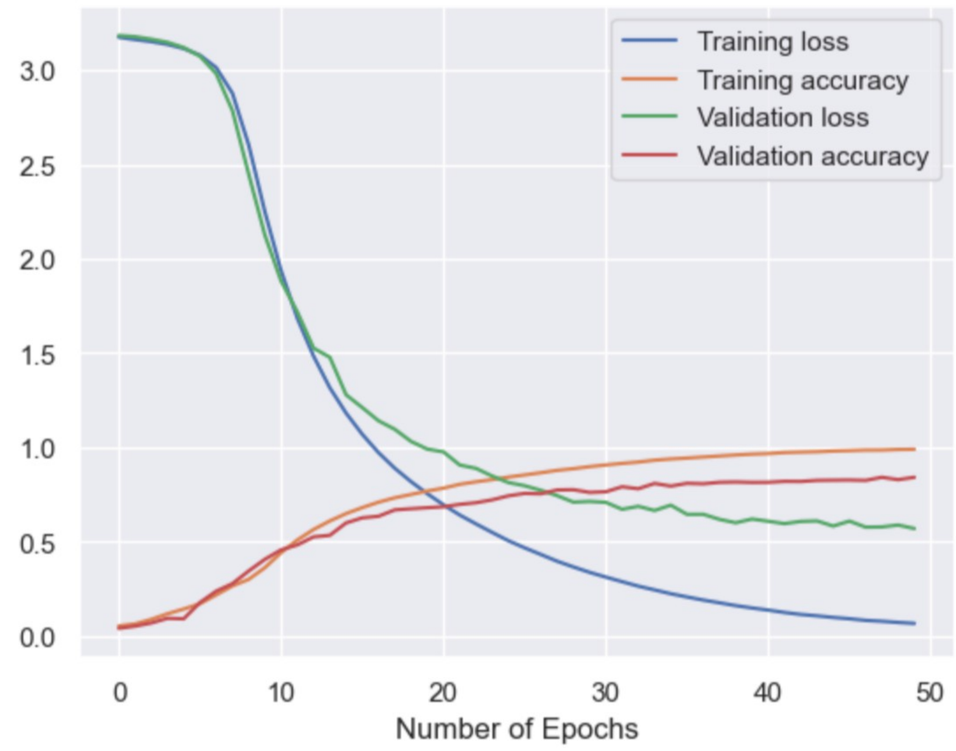
Plot of the CNN model with activation function: selu, optimizer: <class 'keras.src.optimizers.nadam.Nadam'> and learning rate: 0.01




```
Epoch 1/50
858/858 [==============================] - 15s 18ms/step - loss: 3.1738 - accuracy: 0.0549 - val_loss: 3.1831 - val_a
ccuracy: 0.0421
Epoch 2/50
858/858 [==============================] - 16s 18ms/step - loss: 3.1617 - accuracy: 0.0653 - val_loss: 3.1770 - val_a
ccuracy: 0.0538
Epoch 3/50
858/858 [==============================] - 16s 18ms/step - loss: 3.1506 - accuracy: 0.0897 - val_loss: 3.1636 - val_a
ccuracy: 0.0706
Epoch 4/50
858/858 [==============================] - 16s 18ms/step - loss: 3.1359 - accuracy: 0.1195 - val_loss: 3.1470 - val_a
ccuracy: 0.0954
Epoch 5/50
858/858 [==============================] - 16s 18ms/step - loss: 3.1140 - accuracy: 0.1445 - val_loss: 3.1193 - val_a
ccuracy: 0.0929
Epoch 6/50
858/858 [==============================] - 16s 18ms/step - loss: 3.0786 - accuracy: 0.1726 - val_loss: 3.0736 - val_a
ccuracy: 0.1788
Epoch 7/50
858/858 [==============================] - 16s 18ms/step - loss: 3.0131 - accuracy: 0.2193 - val_loss: 2.9810 - val_a
ccuracy: 0.2390
Epoch 8/50
858/858 [==============================] - 16s 18ms/step - loss: 2.8772 - accuracy: 0.2670 - val_loss: 2.7854 - val_a
ccuracy: 0.2789
Epoch 9/50
858/858 [==============================] - 15s 18ms/step - loss: 2.6043 - accuracy: 0.3017 - val_loss: 2.4494 - val_a
ccuracy: 0.3461
Epoch 10/50
858/858 [==============================] - 16s 18ms/step - loss: 2.2492 - accuracy: 0.3636 - val_loss: 2.1250 - val_a
ccuracy: 0.4080
Epoch 11/50
858/858 [==============================] - 16s 18ms/step - loss: 1.9366 - accuracy: 0.4433 - val_loss: 1.8819 - val_a
ccuracy: 0.4582
Epoch 12/50
858/858 [==============================] - 16s 19ms/step - loss: 1.6844 - accuracy: 0.5104 - val_loss: 1.7181 - val_a
ccuracy: 0.4844
Epoch 13/50
858/858 [==============================] - 16s 18ms/step - loss: 1.4840 - accuracy: 0.5659 - val_loss: 1.5265 - val_a
ccuracy: 0.5268
Epoch 14/50
858/858 [==============================] - 15s 18ms/step - loss: 1.3167 - accuracy: 0.6111 - val_loss: 1.4782 - val_a
ccuracy: 0.5351
Epoch 15/50
858/858 [==============================] - 15s 18ms/step - loss: 1.1825 - accuracy: 0.6502 - val_loss: 1.2797 - val_a
```

```
ccuracy: 0.6007
Epoch 16/50
858/858 [==============================] - 15s 18ms/step - loss: 1.0700 - accuracy: 0.6819 - val_loss: 1.2118 - val_a
ccuracy: 0.6277
Epoch 17/50
858/858 [==============================] - 15s 18ms/step - loss: 0.9728 - accuracy: 0.7106 - val_loss: 1.1417 - val_a
ccuracy: 0.6361
Epoch 18/50
858/858 [==============================] - 15s 18ms/step - loss: 0.8903 - accuracy: 0.7338 - val_loss: 1.0967 - val_a
ccuracy: 0.6698
Epoch 19/50
858/858 [==============================] - 15s 18ms/step - loss: 0.8194 - accuracy: 0.7515 - val_loss: 1.0327 - val_a
ccuracy: 0.6760
Epoch 20/50
858/858 [==============================] - 15s 18ms/step - loss: 0.7556 - accuracy: 0.7696 - val_loss: 0.9919 - val_a
ccuracy: 0.6813
Epoch 21/50
858/858 [==============================] - 15s 18ms/step - loss: 0.6968 - accuracy: 0.7851 - val_loss: 0.9771 - val_a
ccuracy: 0.6863
Epoch 22/50
858/858 [==============================] - 15s 18ms/step - loss: 0.6425 - accuracy: 0.8053 - val_loss: 0.9075 - val_a
ccuracy: 0.6991
Epoch 23/50
858/858 [==============================] - 15s 18ms/step - loss: 0.5957 - accuracy: 0.8186 - val_loss: 0.8898 - val_a
ccuracy: 0.7078
Epoch 24/50
858/858 [==============================] - 15s 18ms/step - loss: 0.5503 - accuracy: 0.8302 - val_loss: 0.8489 - val_a
ccuracy: 0.7231
Epoch 25/50
858/858 [==============================] - 17s 20ms/step - loss: 0.5063 - accuracy: 0.8436 - val_loss: 0.8146 - val_a
ccuracy: 0.7443
Epoch 26/50
858/858 [==============================] - 16s 19ms/step - loss: 0.4688 - accuracy: 0.8550 - val_loss: 0.7977 - val_a
ccuracy: 0.7571
Epoch 27/50
858/858 [==============================] - 16s 18ms/step - loss: 0.4334 - accuracy: 0.8667 - val_loss: 0.7736 - val_a
ccuracy: 0.7552
Epoch 28/50
858/858 [==============================] - 16s 19ms/step - loss: 0.3978 - accuracy: 0.8792 - val_loss: 0.7452 - val_a
ccuracy: 0.7752
Epoch 29/50
858/858 [==============================] - 16s 19ms/step - loss: 0.3675 - accuracy: 0.8879 - val_loss: 0.7109 - val_a
ccuracy: 0.7766
Epoch 30/50
858/858 [==============================] - 16s 18ms/step - loss: 0.3389 - accuracy: 0.8989 - val_loss: 0.7146 - val_a
ccuracy: 0.7624
Epoch 31/50
858/858 [==============================] - 16s 19ms/step - loss: 0.3136 - accuracy: 0.9079 - val_loss: 0.7097 - val_a
ccuracy: 0.7658
Epoch 32/50
858/858 [==============================] - 15s 18ms/step - loss: 0.2895 - accuracy: 0.9160 - val_loss: 0.6729 - val_a
ccuracy: 0.7922
Epoch 33/50
858/858 [==============================] - 15s 18ms/step - loss: 0.2657 - accuracy: 0.9233 - val_loss: 0.6888 - val_a
ccuracy: 0.7819
Epoch 34/50
858/858 [==============================] - 15s 18ms/step - loss: 0.2459 - accuracy: 0.9331 - val_loss: 0.6666 - val_a
ccuracy: 0.8098
Epoch 35/50
858/858 [==============================] - 15s 18ms/step - loss: 0.2256 - accuracy: 0.9396 - val_loss: 0.6949 - val_a
ccuracy: 0.7959
Epoch 36/50
858/858 [==============================] - 15s 18ms/step - loss: 0.2084 - accuracy: 0.9443 - val_loss: 0.6461 - val_a
ccuracy: 0.8107
Epoch 37/50
858/858 [==============================] - 16s 18ms/step - loss: 0.1921 - accuracy: 0.9499 - val_loss: 0.6463 - val_a
ccuracy: 0.8081
Epoch 38/50
858/858 [==============================] - 15s 18ms/step - loss: 0.1775 - accuracy: 0.9553 - val_loss: 0.6195 - val_a
ccuracy: 0.8160
Epoch 39/50
858/858 [==============================] - 16s 18ms/step - loss: 0.1627 - accuracy: 0.9607 - val_loss: 0.6016 - val_a
ccuracy: 0.8176
Epoch 40/50
858/858 [==============================] - 19s 22ms/step - loss: 0.1502 - accuracy: 0.9653 - val_loss: 0.6213 - val_a
ccuracy: 0.8154
Epoch 41/50
858/858 [==============================] - 22s 26ms/step - loss: 0.1389 - accuracy: 0.9681 - val_loss: 0.6096 - val_a
ccuracy: 0.8160
Epoch 42/50
858/858 [==============================] - 16s 19ms/step - loss: 0.1263 - accuracy: 0.9736 - val_loss: 0.5969 - val_a
ccuracy: 0.8210
Epoch 43/50
858/858 [==============================] - 15s 17ms/step - loss: 0.1159 - accuracy: 0.9759 - val_loss: 0.6087 - val_a
ccuracy: 0.8204
Epoch 44/50
858/858 [==============================] - 16s 18ms/step - loss: 0.1085 - accuracy: 0.9778 - val_loss: 0.6113 - val_a
ccuracy: 0.8260
Epoch 45/50
858/858 [==============================] - 15s 17ms/step - loss: 0.1001 - accuracy: 0.9817 - val_loss: 0.5841 - val_a
ccuracy: 0.8271
Epoch 46/50
858/858 [==============================] - 15s 17ms/step - loss: 0.0930 - accuracy: 0.9836 - val_loss: 0.6103 - val_a
ccuracy: 0.8279
Epoch 47/50
858/858 [==============================] - 15s 18ms/step - loss: 0.0845 - accuracy: 0.9864 - val_loss: 0.5783 - val_a
ccuracy: 0.8260
Epoch 48/50
858/858 [==============================] - 16s 18ms/step - loss: 0.0798 - accuracy: 0.9867 - val_loss: 0.5795 - val_a
ccuracy: 0.8424
Epoch 49/50
858/858 [==============================] - 18s 21ms/step - loss: 0.0736 - accuracy: 0.9898 - val_loss: 0.5895 - val_a
ccuracy: 0.8299
Epoch 50/50
858/858 [==============================] - 16s 19ms/step - loss: 0.0685 - accuracy: 0.9909 - val_loss: 0.5710 - val_a
ccuracy: 0.8424
```

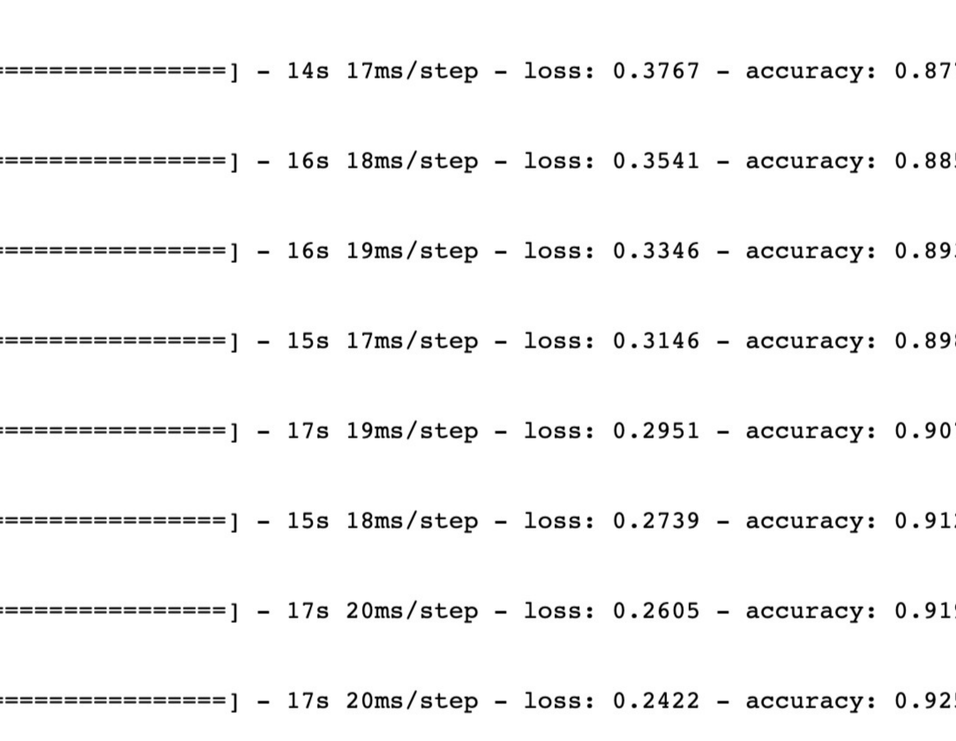
Plot of the CNN model with activation function: elu, optimizer: <class 'keras.src.optimizers.sgd.SGD'> and learning rate: 0.001




```
Epoch 1/50
858/858 [==============================] - 15s 17ms/step - loss: 3.1781 - accuracy: 0.0463 - val_loss: 3.1799 - val_a
ccuracy: 0.0340
Epoch 2/50
858/858 [==============================] - 14s 17ms/step - loss: 3.1744 - accuracy: 0.0504 - val_loss: 3.1808 - val_a
ccuracy: 0.0337
Epoch 3/50
858/858 [==============================] - 14s 17ms/step - loss: 3.1716 - accuracy: 0.0554 - val_loss: 3.1807 - val_a
ccuracy: 0.0368
Epoch 4/50
858/858 [==============================] - 15s 17ms/step - loss: 3.1690 - accuracy: 0.0591 - val_loss: 3.1804 - val_a
ccuracy: 0.0427
Epoch 5/50
858/858 [==============================] - 14s 17ms/step - loss: 3.1661 - accuracy: 0.0614 - val_loss: 3.1795 - val_a
ccuracy: 0.0446
Epoch 6/50
858/858 [==============================] - 14s 17ms/step - loss: 3.1627 - accuracy: 0.0651 - val_loss: 3.1784 - val_a
ccuracy: 0.0482
Epoch 7/50
858/858 [==============================] - 14s 17ms/step - loss: 3.1584 - accuracy: 0.0685 - val_loss: 3.1766 - val_a
ccuracy: 0.0566
Epoch 8/50
858/858 [==============================] - 14s 16ms/step - loss: 3.1526 - accuracy: 0.0790 - val_loss: 3.1736 - val_a
ccuracy: 0.0745
Epoch 9/50
858/858 [==============================] - 14s 17ms/step - loss: 3.1446 - accuracy: 0.0985 - val_loss: 3.1667 - val_a
ccuracy: 0.0923
Epoch 10/50
858/858 [==============================] - 15s 17ms/step - loss: 3.1325 - accuracy: 0.1188 - val_loss: 3.1532 - val_a
ccuracy: 0.1074
Epoch 11/50
858/858 [==============================] - 14s 17ms/step - loss: 3.1112 - accuracy: 0.1394 - val_loss: 3.1249 - val_a
ccuracy: 0.1141
Epoch 12/50
858/858 [==============================] - 14s 17ms/step - loss: 3.0653 - accuracy: 0.1564 - val_loss: 3.0525 - val_a
ccuracy: 0.1235
Epoch 13/50
858/858 [==============================] - 14s 16ms/step - loss: 2.9439 - accuracy: 0.2050 - val_loss: 2.8585 - val_a
ccuracy: 0.2086
Epoch 14/50
858/858 [==============================] - 14s 17ms/step - loss: 2.6383 - accuracy: 0.2729 - val_loss: 2.4853 - val_a
ccuracy: 0.3045
Epoch 15/50
858/858 [==============================] - 14s 16ms/step - loss: 2.2467 - accuracy: 0.3484 - val_loss: 2.1048 - val_a
ccuracy: 0.3851
Epoch 16/50
858/858 [==============================] - 14s 16ms/step - loss: 1.9353 - accuracy: 0.4236 - val_loss: 1.8494 - val_a
ccuracy: 0.4643
Epoch 17/50
858/858 [==============================] - 14s 16ms/step - loss: 1.7008 - accuracy: 0.4806 - val_loss: 1.7178 - val_a
ccuracy: 0.4716
Epoch 18/50
858/858 [==============================] - 14s 16ms/step - loss: 1.5231 - accuracy: 0.5253 - val_loss: 1.6111 - val_a
ccuracy: 0.4992
Epoch 19/50
858/858 [==============================] - 14s 16ms/step - loss: 1.3831 - accuracy: 0.5666 - val_loss: 1.5138 - val_a
ccuracy: 0.4972
Epoch 20/50
858/858 [==============================] - 14s 16ms/step - loss: 1.2772 - accuracy: 0.5957 - val_loss: 1.4054 - val_a
ccuracy: 0.5418
Epoch 21/50
858/858 [==============================] - 14s 16ms/step - loss: 1.1890 - accuracy: 0.6239 - val_loss: 1.3978 - val_a
ccuracy: 0.5544
Epoch 22/50
858/858 [==============================] - 14s 16ms/step - loss: 1.1143 - accuracy: 0.6506 - val_loss: 1.2871 - val_a
ccuracy: 0.5694
Epoch 23/50
858/858 [==============================] - 14s 16ms/step - loss: 1.0549 - accuracy: 0.6661 - val_loss: 1.2436 - val_a
ccuracy: 0.5934
Epoch 24/50
858/858 [==============================] - 14s 16ms/step - loss: 0.9972 - accuracy: 0.6876 - val_loss: 1.2452 - val_a
ccuracy: 0.6168
Epoch 25/50
858/858 [==============================] - 14s 16ms/step - loss: 0.9420 - accuracy: 0.7036 - val_loss: 1.2294 - val_a
ccuracy: 0.6051
Epoch 26/50
858/858 [==============================] - 14s 16ms/step - loss: 0.8942 - accuracy: 0.7192 - val_loss: 1.2346 - val_a
ccuracy: 0.6224
Epoch 27/50
858/858 [==============================] - 14s 17ms/step - loss: 0.8505 - accuracy: 0.7326 - val_loss: 1.2078 - val_a
ccuracy: 0.6113
Epoch 28/50
858/858 [==============================] - 15s 17ms/step - loss: 0.8049 - accuracy: 0.7472 - val_loss: 1.1019 - val_a
ccuracy: 0.6286
Epoch 29/50
858/858 [==============================] - 15s 18ms/step - loss: 0.7639 - accuracy: 0.7585 - val_loss: 1.0626 - val_a
ccuracy: 0.6629
Epoch 30/50
858/858 [==============================] - 14s 17ms/step - loss: 0.7271 - accuracy: 0.7690 - val_loss: 1.0899 - val_a
```

```
ccuracy: 0.6542
Epoch 31/50
858/858 [==============================] - 14s 17ms/step - loss: 0.6919 - accuracy: 0.7792 - val_loss: 1.0420 - val_a
ccuracy: 0.6762
Epoch 32/50
858/858 [==============================] - 14s 17ms/step - loss: 0.6580 - accuracy: 0.7888 - val_loss: 1.0255 - val_a
ccuracy: 0.6748
Epoch 33/50
858/858 [==============================] - 14s 17ms/step - loss: 0.6223 - accuracy: 0.8027 - val_loss: 0.9915 - val_a
ccuracy: 0.6846
Epoch 34/50
858/858 [==============================] - 14s 17ms/step - loss: 0.5923 - accuracy: 0.8085 - val_loss: 1.0318 - val_a
ccuracy: 0.6693
Epoch 35/50
858/858 [==============================] - 15s 18ms/step - loss: 0.5600 - accuracy: 0.8177 - val_loss: 1.0153 - val_a
ccuracy: 0.7089
Epoch 36/50
858/858 [==============================] - 15s 18ms/step - loss: 0.5316 - accuracy: 0.8281 - val_loss: 0.9539 - val_a
ccuracy: 0.6958
Epoch 37/50
858/858 [==============================] - 14s 17ms/step - loss: 0.5073 - accuracy: 0.8346 - val_loss: 1.0098 - val_a
ccuracy: 0.6665
Epoch 38/50
858/858 [==============================] - 15s 17ms/step - loss: 0.4791 - accuracy: 0.8432 - val_loss: 0.9777 - val_a
ccuracy: 0.6790
Epoch 39/50
858/858 [==============================] - 16s 18ms/step - loss: 0.4519 - accuracy: 0.8516 - val_loss: 1.0333 - val_a
ccuracy: 0.6771
Epoch 40/50
858/858 [==============================] - 15s 17ms/step - loss: 0.4259 - accuracy: 0.8619 - val_loss: 0.9714 - val_a
ccuracy: 0.6988
Epoch 41/50
858/858 [==============================] - 14s 16ms/step - loss: 0.4018 - accuracy: 0.8709 - val_loss: 0.8822 - val_a
ccuracy: 0.7234
Epoch 42/50
858/858 [==============================] - 14s 17ms/step - loss: 0.3767 - accuracy: 0.8773 - val_loss: 0.8811 - val_a
ccuracy: 0.7256
Epoch 43/50
858/858 [==============================] - 16s 18ms/step - loss: 0.3541 - accuracy: 0.8855 - val_loss: 0.8874 - val_a
ccuracy: 0.7153
Epoch 44/50
858/858 [==============================] - 16s 19ms/step - loss: 0.3346 - accuracy: 0.8931 - val_loss: 0.8841 - val_a
ccuracy: 0.7167
Epoch 45/50
858/858 [==============================] - 15s 17ms/step - loss: 0.3146 - accuracy: 0.8987 - val_loss: 0.8751 - val_a
ccuracy: 0.7195
Epoch 46/50
858/858 [==============================] - 17s 19ms/step - loss: 0.2951 - accuracy: 0.9072 - val_loss: 0.8726 - val_a
ccuracy: 0.7401
Epoch 47/50
858/858 [==============================] - 15s 18ms/step - loss: 0.2739 - accuracy: 0.9122 - val_loss: 0.8402 - val_a
ccuracy: 0.7387
Epoch 48/50
858/858 [==============================] - 17s 20ms/step - loss: 0.2605 - accuracy: 0.9195 - val_loss: 0.8150 - val_a
ccuracy: 0.7644
Epoch 49/50
858/858 [==============================] - 17s 20ms/step - loss: 0.2422 - accuracy: 0.9252 - val_loss: 0.8153 - val_a
ccuracy: 0.7418
Epoch 50/50
858/858 [==============================] - 17s 20ms/step - loss: 0.2305 - accuracy: 0.9297 - val_loss: 0.8310 - val_a
ccuracy: 0.7390
```

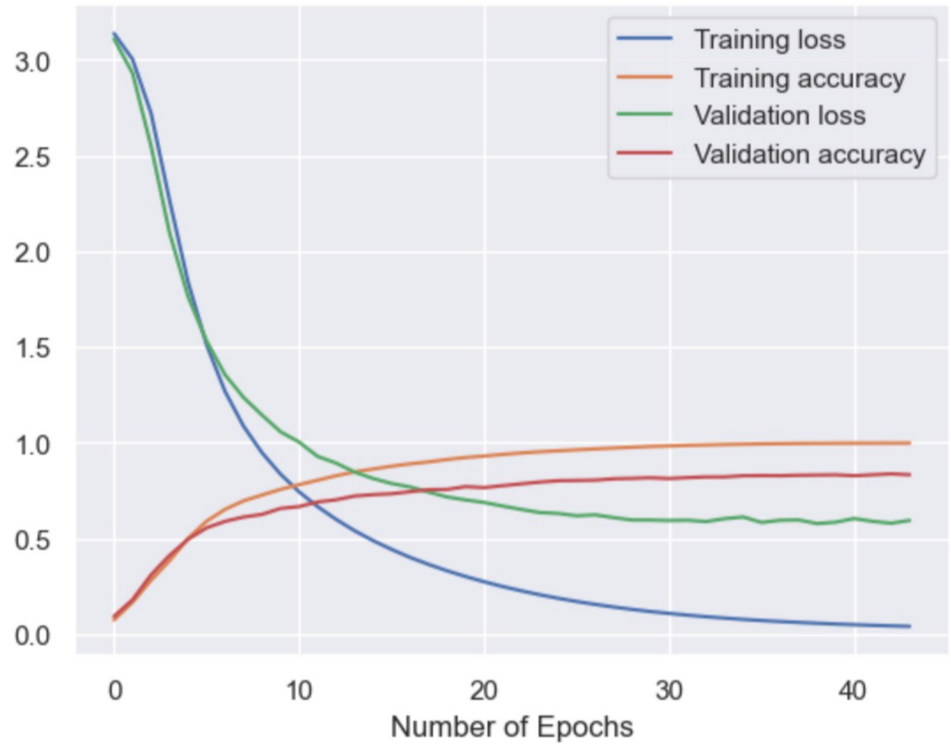
Plot of the CNN model with activation function: LeakyReLU, optimizer: <class 'keras.src.optimizers.sgd.SGD'> and learning rate: 0.001




```
Epoch 1/50
858/858 [==============================] - 20s 23ms/step - loss: 3.1377 - accuracy: 0.0751 - val_loss: 3.1095 - val_a
ccuracy: 0.0937
Epoch 2/50
858/858 [==============================] - 23s 26ms/step - loss: 3.0029 - accuracy: 0.1674 - val_loss: 2.9303 - val_a
ccuracy: 0.1804
Epoch 3/50
858/858 [==============================] - 25s 29ms/step - loss: 2.7222 - accuracy: 0.2821 - val_loss: 2.5505 - val_a
ccuracy: 0.3101
Epoch 4/50
858/858 [==============================] - 24s 28ms/step - loss: 2.2668 - accuracy: 0.3850 - val_loss: 2.0969 - val_a
ccuracy: 0.4116
Epoch 5/50
858/858 [==============================] - 23s 27ms/step - loss: 1.8400 - accuracy: 0.5005 - val_loss: 1.7642 - val_a
ccuracy: 0.4964
Epoch 6/50
858/858 [==============================] - 26s 30ms/step - loss: 1.5123 - accuracy: 0.5903 - val_loss: 1.5317 - val_a
ccuracy: 0.5561
Epoch 7/50
858/858 [==============================] - 23s 27ms/step - loss: 1.2682 - accuracy: 0.6524 - val_loss: 1.3555 - val_a
ccuracy: 0.5898
Epoch 8/50
858/858 [==============================] - 21s 24ms/step - loss: 1.0866 - accuracy: 0.6969 - val_loss: 1.2367 - val_a
ccuracy: 0.6124
Epoch 9/50
858/858 [==============================] - 22s 26ms/step - loss: 0.9479 - accuracy: 0.7273 - val_loss: 1.1447 - val_a
ccuracy: 0.6260
Epoch 10/50
```

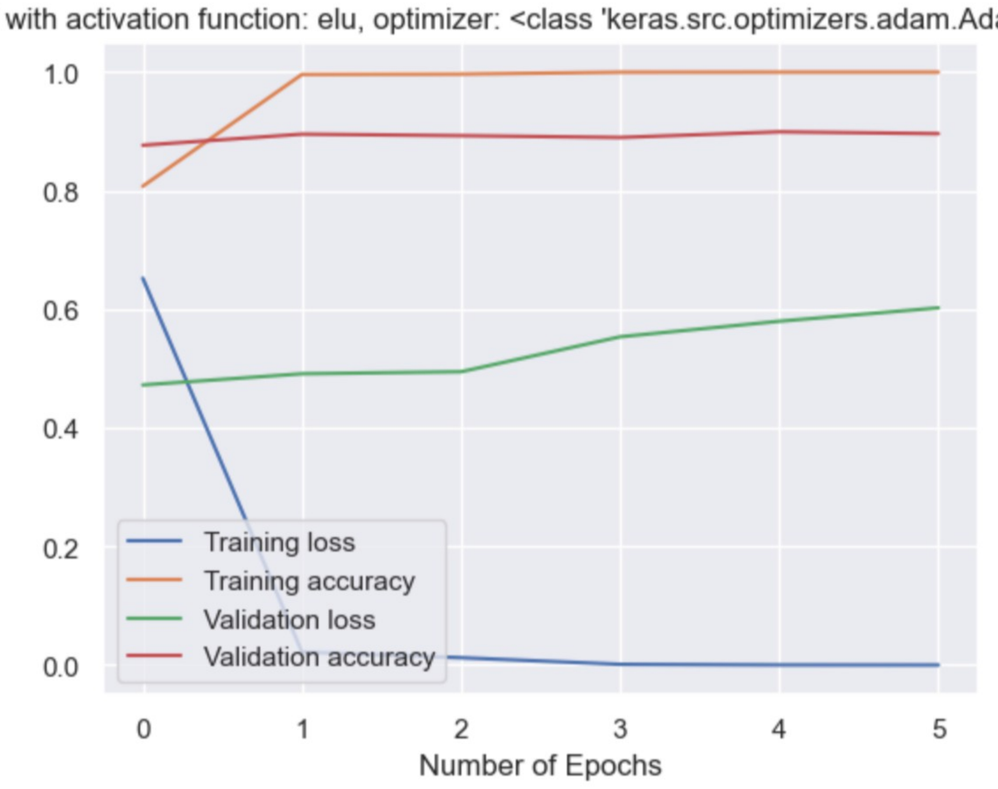
```
858/858 [==============================] - 23s 27ms/step - loss: 0.8363 - accuracy: 0.7569 - val_loss: 1.0575 - val_a
ccuracy: 0.6573
Epoch 11/50
858/858 [==============================] - 23s 26ms/step - loss: 0.7440 - accuracy: 0.7813 - val_loss: 1.0040 - val_a
ccuracy: 0.6668
Epoch 12/50
858/858 [==============================] - 21s 24ms/step - loss: 0.6670 - accuracy: 0.8044 - val_loss: 0.9291 - val_a
ccuracy: 0.6910
Epoch 13/50
858/858 [==============================] - 22s 26ms/step - loss: 0.6002 - accuracy: 0.8282 - val_loss: 0.8941 - val_a
ccuracy: 0.7025
Epoch 14/50
858/858 [==============================] - 22s 26ms/step - loss: 0.5408 - accuracy: 0.8480 - val_loss: 0.8474 - val_a
ccuracy: 0.7220
Epoch 15/50
858/858 [==============================] - 23s 27ms/step - loss: 0.4892 - accuracy: 0.8631 - val_loss: 0.8135 - val_a
ccuracy: 0.7292
Epoch 16/50
858/858 [==============================] - 22s 25ms/step - loss: 0.4433 - accuracy: 0.8778 - val_loss: 0.7875 - val_a
ccuracy: 0.7340
Epoch 17/50
858/858 [==============================] - 22s 25ms/step - loss: 0.4014 - accuracy: 0.8902 - val_loss: 0.7696 - val_a
ccuracy: 0.7448
Epoch 18/50
858/858 [==============================] - 23s 27ms/step - loss: 0.3650 - accuracy: 0.9002 - val_loss: 0.7428 - val_a
ccuracy: 0.7552
Epoch 19/50
858/858 [==============================] - 23s 27ms/step - loss: 0.3317 - accuracy: 0.9128 - val_loss: 0.7172 - val_a
ccuracy: 0.7566
Epoch 20/50
858/858 [==============================] - 22s 26ms/step - loss: 0.3018 - accuracy: 0.9232 - val_loss: 0.7018 - val_a
ccuracy: 0.7711
Epoch 21/50
858/858 [==============================] - 22s 25ms/step - loss: 0.2744 - accuracy: 0.9310 - val_loss: 0.6884 - val_a
ccuracy: 0.7660
Epoch 22/50
858/858 [==============================] - 23s 27ms/step - loss: 0.2495 - accuracy: 0.9394 - val_loss: 0.6697 - val_a
ccuracy: 0.7761
Epoch 23/50
858/858 [==============================] - 20s 23ms/step - loss: 0.2272 - accuracy: 0.9471 - val_loss: 0.6522 - val_a
ccuracy: 0.7856
Epoch 24/50
858/858 [==============================] - 25s 29ms/step - loss: 0.2066 - accuracy: 0.9531 - val_loss: 0.6360 - val_a
ccuracy: 0.7948
Epoch 25/50
858/858 [==============================] - 23s 26ms/step - loss: 0.1880 - accuracy: 0.9583 - val_loss: 0.6311 - val_a
ccuracy: 0.8020
Epoch 26/50
858/858 [==============================] - 21s 24ms/step - loss: 0.1713 - accuracy: 0.9632 - val_loss: 0.6190 - val_a
ccuracy: 0.8034
Epoch 27/50
858/858 [==============================] - 21s 25ms/step - loss: 0.1563 - accuracy: 0.9681 - val_loss: 0.6237 - val_a
ccuracy: 0.8051
Epoch 28/50
858/858 [==============================] - 20s 23ms/step - loss: 0.1424 - accuracy: 0.9726 - val_loss: 0.6094 - val_a
ccuracy: 0.8120
Epoch 29/50
858/858 [==============================] - 19s 22ms/step - loss: 0.1300 - accuracy: 0.9769 - val_loss: 0.5968 - val_a
ccuracy: 0.8143
Epoch 30/50
858/858 [==============================] - 20s 23ms/step - loss: 0.1187 - accuracy: 0.9810 - val_loss: 0.5966 - val_a
ccuracy: 0.8173
Epoch 31/50
858/858 [==============================] - 18s 21ms/step - loss: 0.1090 - accuracy: 0.9835 - val_loss: 0.5942 - val_a
ccuracy: 0.8134
Epoch 32/50
858/858 [==============================] - 21s 24ms/step - loss: 0.0999 - accuracy: 0.9870 - val_loss: 0.5957 - val_a
ccuracy: 0.8182
Epoch 33/50
858/858 [==============================] - 19s 22ms/step - loss: 0.0916 - accuracy: 0.9887 - val_loss: 0.5888 - val_a
ccuracy: 0.8218
Epoch 34/50
858/858 [==============================] - 19s 23ms/step - loss: 0.0846 - accuracy: 0.9912 - val_loss: 0.6037 - val_a
ccuracy: 0.8218
Epoch 35/50
858/858 [==============================] - 19s 22ms/step - loss: 0.0778 - accuracy: 0.9927 - val_loss: 0.6126 - val_a
ccuracy: 0.8279
Epoch 36/50
858/858 [==============================] - 20s 23ms/step - loss: 0.0719 - accuracy: 0.9944 - val_loss: 0.5842 - val_a
ccuracy: 0.8288
Epoch 37/50
858/858 [==============================] - 21s 25ms/step - loss: 0.0667 - accuracy: 0.9955 - val_loss: 0.5954 - val_a
ccuracy: 0.8282
Epoch 38/50
858/858 [==============================] - 21s 25ms/step - loss: 0.0620 - accuracy: 0.9965 - val_loss: 0.5971 - val_a
ccuracy: 0.8305
Epoch 39/50
858/858 [==============================] - 19s 23ms/step - loss: 0.0576 - accuracy: 0.9969 - val_loss: 0.5783 - val_a
ccuracy: 0.8316
Epoch 40/50
858/858 [==============================] - 20s 24ms/step - loss: 0.0535 - accuracy: 0.9977 - val_loss: 0.5854 - val_a
ccuracy: 0.8327
Epoch 41/50
858/858 [==============================] - 20s 23ms/step - loss: 0.0501 - accuracy: 0.9979 - val_loss: 0.6044 - val_a
ccuracy: 0.8282
Epoch 42/50
858/858 [==============================] - 19s 23ms/step - loss: 0.0467 - accuracy: 0.9981 - val_loss: 0.5899 - val_a
ccuracy: 0.8330
Epoch 43/50
858/858 [==============================] - 21s 24ms/step - loss: 0.0438 - accuracy: 0.9986 - val_loss: 0.5802 - val_a
ccuracy: 0.8377
Epoch 44/50
858/858 [==============================] - 20s 23ms/step - loss: 0.0414 - accuracy: 0.9987 - val_loss: 0.5946 - val_a
ccuracy: 0.8332
```

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

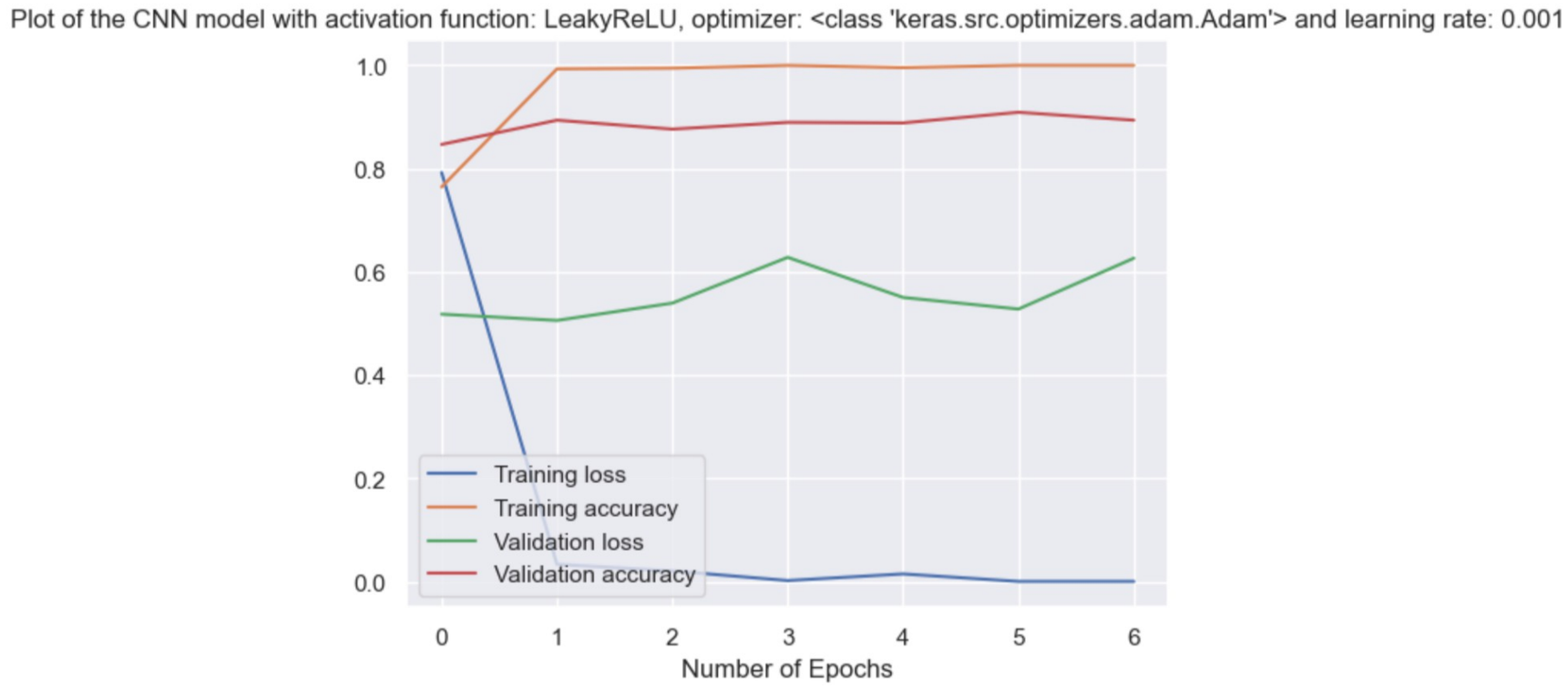



```
Epoch 1/50
858/858 [==============================] - 21s 24ms/step - loss: 0.6524 - accuracy: 0.8076 - val_loss: 0.4725 - val_a
ccuracy: 0.8767
Epoch 2/50
858/858 [==============================] - 22s 26ms/step - loss: 0.0221 - accuracy: 0.9961 - val_loss: 0.4913 - val_a
ccuracy: 0.8954
Epoch 3/50
858/858 [==============================] - 22s 25ms/step - loss: 0.0125 - accuracy: 0.9967 - val_loss: 0.4947 - val_a
ccuracy: 0.8929
Epoch 4/50
858/858 [==============================] - 22s 26ms/step - loss: 0.0013 - accuracy: 0.9999 - val_loss: 0.5537 - val_a
ccuracy: 0.8898
Epoch 5/50
858/858 [==============================] - 22s 26ms/step - loss: 2.9921e-04 - accuracy: 1.0000 - val_loss: 0.5799 - v
al_accuracy: 0.8993
Epoch 6/50
858/858 [==============================] - 24s 28ms/step - loss: 1.4849e-04 - accuracy: 1.0000 - val_loss: 0.6025 - v
al_accuracy: 0.8963
```

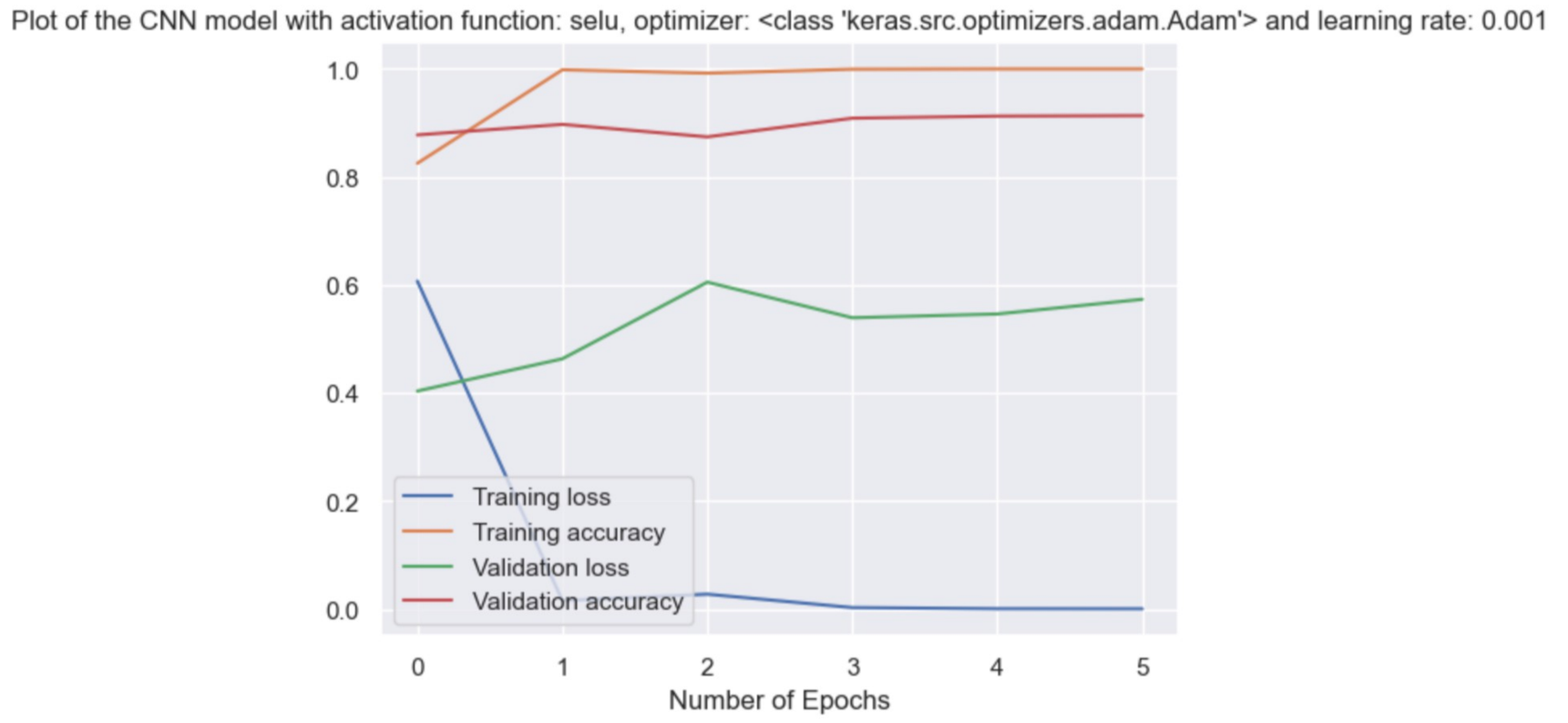
Plot of the CNN model with activation function: elu, optimizer: <class 'keras.src.optimizers.adam.Adam'> and learning rate: 0.001




```
Epoch 1/50
858/858 [==============================] - 20s 23ms/step - loss: 0.7923 - accuracy: 0.7645 - val_loss: 0.5183 - val_a
ccuracy: 0.8469
Epoch 2/50
858/858 [==============================] - 20s 23ms/step - loss: 0.0339 - accuracy: 0.9931 - val_loss: 0.5061 - val_a
ccuracy: 0.8938
Epoch 3/50
858/858 [==============================] - 20s 23ms/step - loss: 0.0212 - accuracy: 0.9944 - val_loss: 0.5397 - val_a
ccuracy: 0.8765
Epoch 4/50
858/858 [==============================] - 20s 23ms/step - loss: 0.0025 - accuracy: 0.9999 - val_loss: 0.6283 - val_a
ccuracy: 0.8896
Epoch 5/50
858/858 [==============================] - 21s 25ms/step - loss: 0.0154 - accuracy: 0.9954 - val_loss: 0.5506 - val_a
ccuracy: 0.8885
Epoch 6/50
858/858 [==============================] - 20s 23ms/step - loss: 8.6169e-04 - accuracy: 0.9999 - val_loss: 0.5281 - v
al_accuracy: 0.9091
Epoch 7/50
858/858 [==============================] - 20s 23ms/step - loss: 8.3940e-04 - accuracy: 0.9999 - val_loss: 0.6266 - v
al_accuracy: 0.8938
```

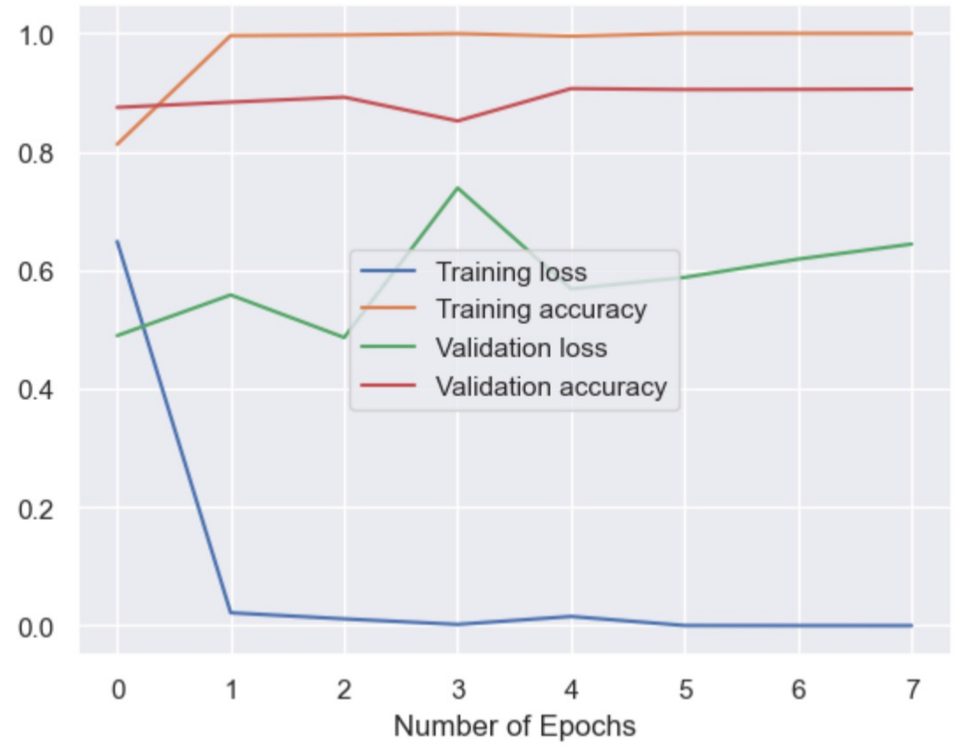



```
Epoch 1/50
858/858 [==============================] - 22s 26ms/step - loss: 0.6071 - accuracy: 0.8253 - val_loss: 0.4035 - val_a
ccuracy: 0.8779
Epoch 2/50
858/858 [==============================] - 22s 26ms/step - loss: 0.0154 - accuracy: 0.9985 - val_loss: 0.4632 - val_a
ccuracy: 0.8974
Epoch 3/50
858/858 [==============================] - 21s 24ms/step - loss: 0.0273 - accuracy: 0.9922 - val_loss: 0.6052 - val_a
ccuracy: 0.8742
Epoch 4/50
858/858 [==============================] - 22s 25ms/step - loss: 0.0025 - accuracy: 0.9996 - val_loss: 0.5393 - val_a
ccuracy: 0.9088
Epoch 5/50
858/858 [==============================] - 22s 25ms/step - loss: 3.7741e-04 - accuracy: 1.0000 - val_loss: 0.5462 - v
al_accuracy: 0.9127
Epoch 6/50
858/858 [==============================] - 21s 25ms/step - loss: 2.0934e-04 - accuracy: 1.0000 - val_loss: 0.5734 - v
al_accuracy: 0.9136
```



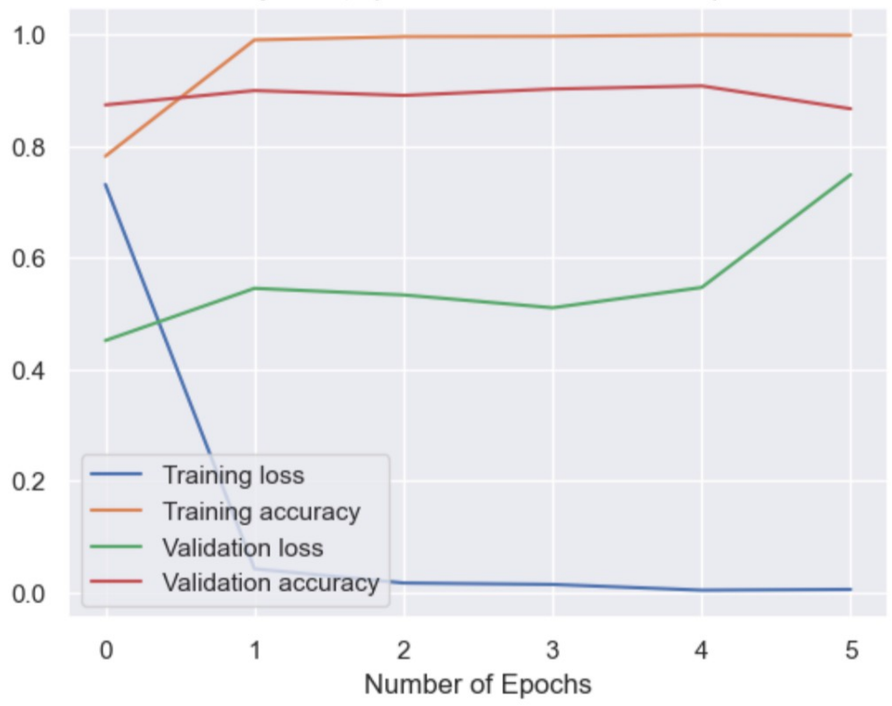

```
Epoch 1/50
858/858 [==============================] - 23s 26ms/step - loss: 0.6488 - accuracy: 0.8127 - val_loss: 0.4897 - val_a
ccuracy: 0.8751
Epoch 2/50
858/858 [==============================] - 22s 26ms/step - loss: 0.0218 - accuracy: 0.9964 - val_loss: 0.5584 - val_a
ccuracy: 0.8840
Epoch 3/50
858/858 [==============================] - 22s 25ms/step - loss: 0.0115 - accuracy: 0.9974 - val_loss: 0.4865 - val_a
ccuracy: 0.8924
Epoch 4/50
858/858 [==============================] - 21s 24ms/step - loss: 0.0022 - accuracy: 0.9996 - val_loss: 0.7391 - val_a
ccuracy: 0.8522
Epoch 5/50
858/858 [==============================] - 23s 27ms/step - loss: 0.0156 - accuracy: 0.9953 - val_loss: 0.5686 - val_a
ccuracy: 0.9069
Epoch 6/50
858/858 [==============================] - 21s 25ms/step - loss: 3.0463e-04 - accuracy: 1.0000 - val_loss: 0.5880 - v
al_accuracy: 0.9052
Epoch 7/50
858/858 [==============================] - 20s 24ms/step - loss: 1.3558e-04 - accuracy: 1.0000 - val_loss: 0.6189 - v
al_accuracy: 0.9055
Epoch 8/50
858/858 [==============================] - 22s 26ms/step - loss: 7.9281e-05 - accuracy: 1.0000 - val_loss: 0.6443 - v
al_accuracy: 0.9060
```

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



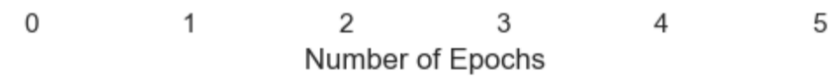

```
Epoch 1/50
858/858 [==============================] - 20s 23ms/step - loss: 0.7310 - accuracy: 0.7818 - val_loss: 0.4513 - val_a
ccuracy: 0.8737
Epoch 2/50
858/858 [==============================] - 19s 22ms/step - loss: 0.0421 - accuracy: 0.9901 - val_loss: 0.5449 - val_a
ccuracy: 0.8993
Epoch 3/50
858/858 [==============================] - 18s 21ms/step - loss: 0.0168 - accuracy: 0.9961 - val_loss: 0.5330 - val_a
ccuracy: 0.8907
Epoch 4/50
858/858 [==============================] - 19s 22ms/step - loss: 0.0142 - accuracy: 0.9966 - val_loss: 0.5101 - val_a
ccuracy: 0.9021
Epoch 5/50
858/858 [==============================] - 19s 23ms/step - loss: 0.0038 - accuracy: 0.9991 - val_loss: 0.5464 - val_a
ccuracy: 0.9077
Epoch 6/50
858/858 [==============================] - 19s 23ms/step - loss: 0.0051 - accuracy: 0.9986 - val_loss: 0.7486 - val_a
ccuracy: 0.8667
```

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




```
Epoch 1/50
858/858 [==============================] - 23s 26ms/step - loss: 0.5762 - accuracy: 0.8355 - val_loss: 0.4641 - val_a
ccuracy: 0.8728
Epoch 2/50
858/858 [==============================] - 24s 28ms/step - loss: 0.0179 - accuracy: 0.9975 - val_loss: 0.4863 - val_a
ccuracy: 0.8901
Epoch 3/50
858/858 [==============================] - 23s 27ms/step - loss: 0.0111 - accuracy: 0.9975 - val_loss: 0.4948 - val_a
ccuracy: 0.8985
Epoch 4/50
858/858 [==============================] - 24s 28ms/step - loss: 9.0320e-04 - accuracy: 1.0000 - val_loss: 0.5157 - v
al_accuracy: 0.9027
Epoch 5/50
858/858 [==============================] - 24s 27ms/step - loss: 3.2747e-04 - accuracy: 1.0000 - val_loss: 0.5211 - v
al_accuracy: 0.9077
Epoch 6/50
858/858 [==============================] - 23s 26ms/step - loss: 1.8214e-04 - accuracy: 1.0000 - val_loss: 0.5409 - v
al_accuracy: 0.9124
```

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



In [38]:
```
lr_cnn = []
opt_cnn = []
act_cnn = []
val_acc_cnn = []
for i in range(0, len(res_cnn)):
    lr_cnn.append(res_cnn[i][0])
    opt_cnn.append(res_cnn[i][1])
    act_cnn.append(res_cnn[i][2])
    val_acc_cnn.append(res_cnn[i][3])

print('Table of the maximum validation accuracy for each combination of activation function, optimizer and learning rat
val_cnn = pd.DataFrame({'Learning rate': lr_cnn, 'Optimizer': opt_cnn, 'Activation function': act_cnn,
                        'Max validation accuracy': val_acc_cnn})
val_cnn = val_cnn.sort_values('Max validation accuracy')
val_cnn
```

```
Table of the maximum validation accuracy for each combination of activation function, optimizer and learning rate for
the CNN model:
```

Out[38]:

	Learning rate	Optimizer	Activation function	Max validation accuracy
5	0.010	<class 'keras.src.optimizers.adam.Adam'>	selu	0.061350
6	0.010	<class 'keras.src.optimizers.nadam.Nadam'>	elu	0.061350
10	0.001	<class 'keras.src.optimizers.sgd.SGD'>	LeakyReLU	0.764361
1	0.010	<class 'keras.src.optimizers.sgd.SGD'>	LeakyReLU	0.836029
11	0.001	<class 'keras.src.optimizers.sgd.SGD'>	selu	0.837702
9	0.001	<class 'keras.src.optimizers.sgd.SGD'>	elu	0.842443
8	0.010	<class 'keras.src.optimizers.nadam.Nadam'>	selu	0.844674
4	0.010	<class 'keras.src.optimizers.adam.Adam'>	LeakyReLU	0.873118
7	0.010	<class 'keras.src.optimizers.nadam.Nadam'>	LeakyReLU	0.875627
0	0.010	<class 'keras.src.optimizers.sgd.SGD'>	elu	0.883157
3	0.010	<class 'keras.src.optimizers.adam.Adam'>	elu	0.886224
12	0.001	<class 'keras.src.optimizers.adam.Adam'>	elu	0.899331
2	0.010	<class 'keras.src.optimizers.sgd.SGD'>	selu	0.900446
15	0.001	<class 'keras.src.optimizers.nadam.Nadam'>	elu	0.906860
16	0.001	<class 'keras.src.optimizers.nadam.Nadam'>	LeakyReLU	0.907697
13	0.001	<class 'keras.src.optimizers.adam.Adam'>	LeakyReLU	0.909091
17	0.001	<class 'keras.src.optimizers.nadam.Nadam'>	selu	0.912437
14	0.001	<class 'keras.src.optimizers.adam.Adam'>	selu	0.913553

In general, these CNN models gain better performance since their maximum validation accuracy are larger than the figures for other densely connected models. Most of the maximum validation accuracy scores of these CNN models range from about 0.76 to 0.91. Additionally, the differences between training accuracy and validation accuracy of these CNN models are not large as the figures for other densely connected models, since they are only from around 0.1 to 0.2. Although these differences are not large, they still can indicate overfitting problem, therefore, in order to solve this problem, three methods such as Regularization, Weight initialization and Dropout regularization are used.

The best CNN model is the model with Selu activation function, Adam optimizer and learning rate of 0.001. Its training accuracy is 1, and its maximum validation accuracy is around 0.9136. The difference between its validation accuracy and training accuracy is about 0.0864. The second best CNN model is the model with Selu activation function, Nadam optimizer and learning rate of 0.001. Its training accuracy is 1, and its maximum validation accuracy is about 0.9124. The difference between its validation accuracy and training accuracy is also about 0.0876. These two best models are chosen to apply some regularization methods to mitigate overfitting problem.

Apply Regularization methods on the second best CNN model (Selu activation function, Nadam optimizer and learning rate 0.001)

In [39]:
```
# Try using Regularization for the second best CNN model with Selu activation function, Nadam optimizer and learning ra
model_cnn_reg_1 = keras.models.Sequential()
model_cnn_reg_1.add(keras.layers.Conv2D(filters=hiddensizes[0], kernel_size=3, strides=1,
                                        activation='selu', padding="same", input_shape=[28, 28, 1],
                                        kernel_regularizer=keras.regularizers.l2(0.01),
                                        bias_regularizer=keras.regularizers.l2(0.01)))
model_cnn_reg_1.add(keras.layers.MaxPooling2D(pool_size=2))
for n in hiddensizes[1:-1]:
    model_cnn_reg_1.add(keras.layers.Conv2D(filters=n, kernel_size=3, strides=1, padding="same", activation='selu'))
    model_cnn_reg_1.add(keras.layers.MaxPooling2D(pool_size=2))
model_cnn_reg_1.add(keras.layers.Conv2D(filters=hiddensizes[-1], kernel_size=3, strides=1,
                                        padding="same", activation='selu'))
model_cnn_reg_1.add(keras.layers.Flatten())
model_cnn_reg_1.add(keras.layers.Dense(24, activation = "softmax"))
model_cnn_reg_1.compile(loss="sparse_categorical_crossentropy", optimizer=keras.optimizers.Nadam(learning_rate=0.001),
                        metrics=["accuracy"])
history_cnn_reg_1 = model_cnn_reg_1.fit(X_train, y_train, epochs=50,
                                        callbacks = early_stopping_cb, validation_data=(X_val, y_val))
```

```
Epoch 1/50
858/858 [==============================] - 23s 26ms/step - loss: 0.6340 - accuracy: 0.8266 - val_loss: 0.4913 - val_a
ccuracy: 0.8561
Epoch 2/50
858/858 [==============================] - 22s 25ms/step - loss: 0.0543 - accuracy: 0.9969 - val_loss: 0.4549 - val_a
ccuracy: 0.8832
Epoch 3/50
858/858 [==============================] - 23s 27ms/step - loss: 0.0274 - accuracy: 0.9997 - val_loss: 0.5510 - val_a
ccuracy: 0.8759
Epoch 4/50
858/858 [==============================] - 22s 26ms/step - loss: 0.0275 - accuracy: 0.9967 - val_loss: 0.4342 - val_a
ccuracy: 0.9002
Epoch 5/50
858/858 [==============================] - 22s 26ms/step - loss: 0.0184 - accuracy: 0.9975 - val_loss: 0.7784 - val_a
ccuracy: 0.8274
Epoch 6/50
858/858 [==============================] - 23s 27ms/step - loss: 0.0222 - accuracy: 0.9964 - val_loss: 0.4775 - val_a
ccuracy: 0.9147
Epoch 7/50
858/858 [==============================] - 23s 27ms/step - loss: 0.0074 - accuracy: 1.0000 - val_loss: 0.4289 - val_a
ccuracy: 0.9197
Epoch 8/50
858/858 [==============================] - 22s 25ms/step - loss: 0.0160 - accuracy: 0.9964 - val_loss: 0.7902 - val_a
```

```
ccuracy: 0.8572
Epoch 9/50
858/858 [==============================] - 22s 25ms/step - loss: 0.0178 - accuracy: 0.9970 - val_loss: 0.5568 - val_a
ccuracy: 0.9105
Epoch 10/50
858/858 [==============================] - 24s 28ms/step - loss: 0.0056 - accuracy: 1.0000 - val_loss: 0.5342 - val_a
ccuracy: 0.9024
Epoch 11/50
858/858 [==============================] - 22s 26ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.4938 - val_a
ccuracy: 0.8940
Epoch 12/50
858/858 [==============================] - 22s 26ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 0.4704 - val_a
ccuracy: 0.9027
```

According to the performance of the second best CNN model (with Selu activation function, Nadam optimizer and learning rate of 0.001) after using l2 Regularizer, this model converges after epoch 12. Its training accuracy is 1 and its validation accuracy is about 0.9027. The difference between these two values is about 0.0973. Hence, the method of l2 Regulization does not reduce the gap between the training accuracy and validation accuracy of this second best CNN model since the difference between these two values before applying l2 Regularization is 0.0876.

In [40]:
```
# Try using Weight Initialization for the second best CNN model with Selu activation function, Nadam optimizer and lear
model_cnn_wi_1 = keras.models.Sequential()
model_cnn_wi_1.add(keras.layers.Conv2D(filters=hiddensizes[0], kernel_size=3, strides=1,
                                       activation='selu', padding="same",
                                       input_shape=[28, 28, 1], kernel_initializer = 'lecun_normal'))
model_cnn_wi_1.add(keras.layers.MaxPooling2D(pool_size=2))
for n in hiddensizes[1:-1]:
    model_cnn_wi_1.add(keras.layers.Conv2D(filters=n, kernel_size=3, strides=1, padding="same", activation='selu'))
    model_cnn_wi_1.add(keras.layers.MaxPooling2D(pool_size=2))
model_cnn_wi_1.add(keras.layers.Conv2D(filters=hiddensizes[-1], kernel_size=3,
                                       strides=1, padding="same", activation='selu'))
model_cnn_wi_1.add(keras.layers.Flatten())
model_cnn_wi_1.add(keras.layers.Dense(24, activation = "softmax"))
model_cnn_wi_1.compile(loss="sparse_categorical_crossentropy",
                       optimizer=keras.optimizers.Nadam(learning_rate=0.001), metrics=["accuracy"])
history_cnn_wi_1 = model_cnn_wi_1.fit(X_train, y_train, epochs=50,
                                      callbacks = early_stopping_cb, validation_data=(X_val, y_val))
```

```
Epoch 1/50
858/858 [==============================] - 24s 27ms/step - loss: 0.5053 - accuracy: 0.8583 - val_loss: 0.4194 - val_a
ccuracy: 0.8765
Epoch 2/50
858/858 [==============================] - 22s 26ms/step - loss: 0.0143 - accuracy: 0.9989 - val_loss: 0.4313 - val_a
ccuracy: 0.8949
Epoch 3/50
858/858 [==============================] - 22s 25ms/step - loss: 0.0116 - accuracy: 0.9972 - val_loss: 0.4673 - val_a
ccuracy: 0.8820
Epoch 4/50
858/858 [==============================] - 24s 28ms/step - loss: 0.0043 - accuracy: 0.9989 - val_loss: 0.4709 - val_a
ccuracy: 0.8988
Epoch 5/50
858/858 [==============================] - 22s 25ms/step - loss: 0.0083 - accuracy: 0.9974 - val_loss: 0.4828 - val_a
ccuracy: 0.9016
Epoch 6/50
858/858 [==============================] - 22s 26ms/step - loss: 1.3925e-04 - accuracy: 1.0000 - val_loss: 0.4783 - v
al_accuracy: 0.9080
```

According to the performance of the second best CNN model (with Selu activation function, Nadam optimizer and learning rate of 0.001) after using Weight Initialization, this model converges after epoch 6, its training accuracy is about 1 and its validation accuracy is about 0.908. The difference between these two values is about 0.092. Hence, the method of Weight Initialization also does not reduce the difference between the training accuracy and the validation accuracy of this second best CNN model since the difference between these two values before applying Weight Initialization is 0.0876.

In [41]:
```
# Try using Dropout Regularization for the second best CNN model with Selu activation function, Nadam optimizer and lea
model_cnn_dr_1 = keras.models.Sequential()
model_cnn_dr_1.add(keras.layers.Conv2D(filters=hiddensizes[0], kernel_size=3, strides=1,
                                       activation='selu', padding="same", input_shape=[28, 28, 1]))
model_cnn_dr_1.add(keras.layers.MaxPooling2D(pool_size=2))
model_cnn_dr_1.add(keras.layers.Dropout (0.05))
for n in hiddensizes[1:-1]:
    model_cnn_dr_1.add(keras.layers.Conv2D(filters=n, kernel_size=3, strides=1, padding="same", activation='selu'))
    model_cnn_dr_1.add(keras.layers.MaxPooling2D(pool_size=2))
    model_cnn_dr_1.add(keras.layers.Dropout (0.05))
model_cnn_dr_1.add(keras.layers.Conv2D(filters=hiddensizes[-1], kernel_size=3,
                                       strides=1, padding="same", activation='selu'))
model_cnn_dr_1.add(keras.layers.Flatten())
model_cnn_dr_1.add(keras.layers.Dense(24, activation = "softmax"))
model_cnn_dr_1.compile(loss="sparse_categorical_crossentropy", optimizer=keras.optimizers.Nadam(learning_rate=0.001),
                       metrics=["accuracy"])
history_cnn_dr_1 = model_cnn_dr_1.fit(X_train, y_train, epochs=50,
                                      callbacks = early_stopping_cb, validation_data=(X_val, y_val))
```

```
Epoch 1/50
858/858 [==============================] - 26s 29ms/step - loss: 0.6450 - accuracy: 0.8121 - val_loss: 0.3691 - val_a
ccuracy: 0.8820
Epoch 2/50
858/858 [==============================] - 25s 29ms/step - loss: 0.0269 - accuracy: 0.9953 - val_loss: 0.3271 - val_a
ccuracy: 0.9088
Epoch 3/50
858/858 [==============================] - 26s 31ms/step - loss: 0.0133 - accuracy: 0.9970 - val_loss: 0.4485 - val_a
ccuracy: 0.8918
Epoch 4/50
858/858 [==============================] - 25s 29ms/step - loss: 0.0127 - accuracy: 0.9967 - val_loss: 0.4957 - val_a
ccuracy: 0.8985
Epoch 5/50
858/858 [==============================] - 25s 29ms/step - loss: 0.0079 - accuracy: 0.9979 - val_loss: 0.6247 - val_a
ccuracy: 0.8918
Epoch 6/50
858/858 [==============================] - 25s 29ms/step - loss: 0.0046 - accuracy: 0.9987 - val_loss: 0.3954 - val_a
ccuracy: 0.9225
Epoch 7/50
858/858 [==============================] - 26s 30ms/step - loss: 0.0078 - accuracy: 0.9979 - val_loss: 0.5070 - val_a
ccuracy: 0.9021
```

According to the performance of the second best CNN model (with Selu activation function, Nadam optimizer and learning rate of 0.001) after using Dropout Regularization, this model converges after epoch 7, its training accuracy is about 0.9979 and its validation accuracy is about 0.9021. The difference between these two values is about 0.0958. Hence, the method of Dropout Regularization also does not reduce the difference between the training accuracy and the validation accuracy of this second best CNN model since the difference between these two values before applying Dropout Regularization is 0.0876.

Apply Regularization methods on the best CNN model (Selu activation function, Adam optimizer and learning rate 0.001)

In [42]:
```
# Try using Regularization for the best CNN model with Selu activation function, Adam optimizer and learning rate of 0.
model_cnn_reg_2 = keras.models.Sequential()
model_cnn_reg_2.add(keras.layers.Conv2D(filters=hiddensizes[0], kernel_size=3, strides=1,
                                        activation='selu', padding="same", input_shape=[28, 28, 1],
                                        kernel_regularizer=keras.regularizers.l2(0.01),
                                        bias_regularizer=keras.regularizers.l2(0.01)))
model_cnn_reg_2.add(keras.layers.MaxPooling2D(pool_size=2))
for n in hiddensizes[1:-1]:
    model_cnn_reg_2.add(keras.layers.Conv2D(filters=n, kernel_size=3, strides=1, padding="same", activation='selu'))
    model_cnn_reg_2.add(keras.layers.MaxPooling2D(pool_size=2))
model_cnn_reg_2.add(keras.layers.Conv2D(filters=hiddensizes[-1], kernel_size=3,
                                        strides=1, padding="same", activation='selu'))
model_cnn_reg_2.add(keras.layers.Flatten())
model_cnn_reg_2.add(keras.layers.Dense(24, activation = "softmax"))
model_cnn_reg_2.compile(loss="sparse_categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001),
                        metrics=["accuracy"])
history_cnn_reg_2 = model_cnn_reg_2.fit(X_train, y_train, epochs=50,
                                        callbacks = early_stopping_cb, validation_data=(X_val, y_val))
```

```
Epoch 1/50
858/858 [==============================] - 24s 27ms/step - loss: 0.6275 - accuracy: 0.8290 - val_loss: 0.3969 - val_a
ccuracy: 0.8904
Epoch 2/50
858/858 [==============================] - 22s 25ms/step - loss: 0.0486 - accuracy: 0.9976 - val_loss: 0.4046 - val_a
ccuracy: 0.9055
Epoch 3/50
858/858 [==============================] - 22s 25ms/step - loss: 0.0279 - accuracy: 0.9990 - val_loss: 0.3782 - val_a
ccuracy: 0.9080
Epoch 4/50
858/858 [==============================] - 23s 27ms/step - loss: 0.0378 - accuracy: 0.9929 - val_loss: 0.5874 - val_a
ccuracy: 0.8659
Epoch 5/50
858/858 [==============================] - 23s 27ms/step - loss: 0.0165 - accuracy: 0.9990 - val_loss: 0.3790 - val_a
ccuracy: 0.9255
Epoch 6/50
858/858 [==============================] - 22s 26ms/step - loss: 0.0094 - accuracy: 1.0000 - val_loss: 0.3470 - val_a
ccuracy: 0.9286
Epoch 7/50
858/858 [==============================] - 22s 26ms/step - loss: 0.0067 - accuracy: 1.0000 - val_loss: 0.4123 - val_a
ccuracy: 0.9127
Epoch 8/50
858/858 [==============================] - 22s 26ms/step - loss: 0.0406 - accuracy: 0.9906 - val_loss: 0.4843 - val_a
ccuracy: 0.9041
Epoch 9/50
858/858 [==============================] - 23s 27ms/step - loss: 0.0068 - accuracy: 1.0000 - val_loss: 0.4421 - val_a
ccuracy: 0.9124
Epoch 10/50
858/858 [==============================] - 23s 27ms/step - loss: 0.0054 - accuracy: 1.0000 - val_loss: 0.4143 - val_a
ccuracy: 0.9161
Epoch 11/50
858/858 [==============================] - 23s 26ms/step - loss: 0.0042 - accuracy: 1.0000 - val_loss: 0.3985 - val_a
ccuracy: 0.9191
```

According to the performance of the best CNN model (with Selu activation function, Adam optimizer and learning rate of 0.001) after using l2 Regularizer, this model converges after epoch 11, its training accuracy is about 1 and its validation accuracy is about 0.9191. The difference between these two values is about 0.0809. Hence, the method of l2 Regulization can slightly reduce the difference between the training accuracy and the validation accuracy of this best CNN model since the difference between these two values before applying l2 Regularization is 0.0864. However, it does not improve the validation accuracy much because the original validation accuracy is 0.9136.

In [43]:
```
# Try using Weight Initialization for the best CNN model with Selu activation function, Adam optimizer and learning rat
model_cnn_wi_2 = keras.models.Sequential()
model_cnn_wi_2.add(keras.layers.Conv2D(filters=hiddensizes[0], kernel_size=3, strides=1,
                                       activation='selu', padding="same", input_shape=[28, 28, 1],
                                       kernel_initializer = 'lecun_normal'))
model_cnn_wi_2.add(keras.layers.MaxPooling2D(pool_size=2))
for n in hiddensizes[1:-1]:
    model_cnn_wi_2.add(keras.layers.Conv2D(filters=n, kernel_size=3, strides=1, padding="same", activation='selu'))
    model_cnn_wi_2.add(keras.layers.MaxPooling2D(pool_size=2))
model_cnn_wi_2.add(keras.layers.Conv2D(filters=hiddensizes[-1], kernel_size=3, strides=1,
                                       padding="same", activation='selu'))
model_cnn_wi_2.add(keras.layers.Flatten())
model_cnn_wi_2.add(keras.layers.Dense(24, activation = "softmax"))
model_cnn_wi_2.compile(loss="sparse_categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001),
                       metrics=["accuracy"])
history_cnn_wi_2 = model_cnn_wi_2.fit(X_train, y_train, epochs=50,
                                      callbacks = early_stopping_cb, validation_data=(X_val, y_val))
```

```
Epoch 1/50
858/858 [==============================] - 22s 26ms/step - loss: 0.5768 - accuracy: 0.8359 - val_loss: 0.3951 - val_a
ccuracy: 0.8781
Epoch 2/50
858/858 [==============================] - 21s 25ms/step - loss: 0.0159 - accuracy: 0.9986 - val_loss: 0.3748 - val_a
ccuracy: 0.8974
Epoch 3/50
858/858 [==============================] - 22s 26ms/step - loss: 0.0116 - accuracy: 0.9975 - val_loss: 0.4356 - val_a
ccuracy: 0.8921
Epoch 4/50
858/858 [==============================] - 23s 27ms/step - loss: 0.0149 - accuracy: 0.9955 - val_loss: 0.4787 - val_a
ccuracy: 0.8979
Epoch 5/50
858/858 [==============================] - 23s 26ms/step - loss: 4.5626e-04 - accuracy: 1.0000 - val_loss: 0.4567 - v
al_accuracy: 0.9013
Epoch 6/50
858/858 [==============================] - 22s 26ms/step - loss: 1.5869e-04 - accuracy: 1.0000 - val_loss: 0.4449 - v
al_accuracy: 0.9096
Epoch 7/50
858/858 [==============================] - 22s 26ms/step - loss: 9.6578e-05 - accuracy: 1.0000 - val_loss: 0.4586 - v
al_accuracy: 0.9094
```

According to the performance of the best CNN model (with Selu activation function, Adam optimizer and learning rate of 0.001) after using Weight Initialization, after epoch 7, its training accuracy is about 1 and its validation accuracy is about 0.9094. The difference between these two values is about 0.0906. Hence, the method of Weight Initialization does not reduce the difference between training accuracy and validation accuracy of this best CNN model since the gap between these two values before applying Weight Initialization is 0.0864.

In [44]:
```
# Try using Dropout Regularization for the best CNN model with Selu activation function, Adam optimizer and learning ra
model_cnn_dr_2 = keras.models.Sequential()
model_cnn_dr_2.add(keras.layers.Conv2D(filters=hiddensizes[0], kernel_size=3, strides=1,
                                       activation='selu', padding="same", input_shape=[28, 28, 1]))
model_cnn_dr_2.add(keras.layers.MaxPooling2D(pool_size=2))
model_cnn_dr_2.add(keras.layers.Dropout (0.05))
for n in hiddensizes[1:-1]:
    model_cnn_dr_2.add(keras.layers.Conv2D(filters=n, kernel_size=3, strides=1, padding="same", activation='selu'))
    model_cnn_dr_2.add(keras.layers.MaxPooling2D(pool_size=2))
    model_cnn_dr_2.add(keras.layers.Dropout (0.05))
model_cnn_dr_2.add(keras.layers.Conv2D(filters=hiddensizes[-1], kernel_size=3,
                                       strides=1, padding="same", activation='selu'))
model_cnn_dr_2.add(keras.layers.Flatten())
model_cnn_dr_2.add(keras.layers.Dense(24, activation = "softmax"))
model_cnn_dr_2.compile(loss="sparse_categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001),
                       metrics=["accuracy"])
history_cnn_dr_2 = model_cnn_dr_2.fit(X_train, y_train, epochs=50,
                                      callbacks = early_stopping_cb, validation_data=(X_val, y_val))
```

```
Epoch 1/50
858/858 [==============================] - 25s 28ms/step - loss: 0.5971 - accuracy: 0.8245 - val_loss: 0.4035 - val_a
ccuracy: 0.8756
Epoch 2/50
858/858 [==============================] - 25s 29ms/step - loss: 0.0298 - accuracy: 0.9942 - val_loss: 0.4644 - val_a
ccuracy: 0.8798
Epoch 3/50
858/858 [==============================] - 26s 31ms/step - loss: 0.0201 - accuracy: 0.9944 - val_loss: 0.4454 - val_a
ccuracy: 0.9071
Epoch 4/50
858/858 [==============================] - 26s 30ms/step - loss: 0.0026 - accuracy: 0.9996 - val_loss: 0.4577 - val_a
ccuracy: 0.9102
Epoch 5/50
858/858 [==============================] - 25s 29ms/step - loss: 6.2306e-04 - accuracy: 1.0000 - val_loss: 0.4802 - v
al_accuracy: 0.9194
Epoch 6/50
858/858 [==============================] - 24s 28ms/step - loss: 0.0308 - accuracy: 0.9908 - val_loss: 0.6384 - val_a
ccuracy: 0.8804
```

According to the performance of the best CNN model (with Selu activation function, Adam optimizer and learning rate of 0.001) after using Dropout Regularization, this model converges after epoch 6, its training accuracy is about 0.9908 and its validation accuracy is about 0.8804. The difference between these two values is about 0.1104. Hence, the method of Dropout Regularization also does not reduce the gap between the training accuracy and the validation accuracy of this best CNN model since the difference between these two values before applying Dropout Regularization is 0.0864.

In conclusion, in comparison with the best densely connected model, the best CNN model (with Selu activation function, Adam optimizer and learning rate of 0.001), and the second best CNN model (with Selu activation function, Nadam optimizer and learning rate of 0.001) obtain better validation accuracy score. Therefore, these two best models are used in prediction process. In addition, three methods (Regularization, Weight initialization and Dropout regularization) do not reduce the difference between training accuracy and validation accuracy, hence they can not mitigate the overfitting problem. Therefore, these two best CNN models are used to generate predictions without applying any regularization methods.

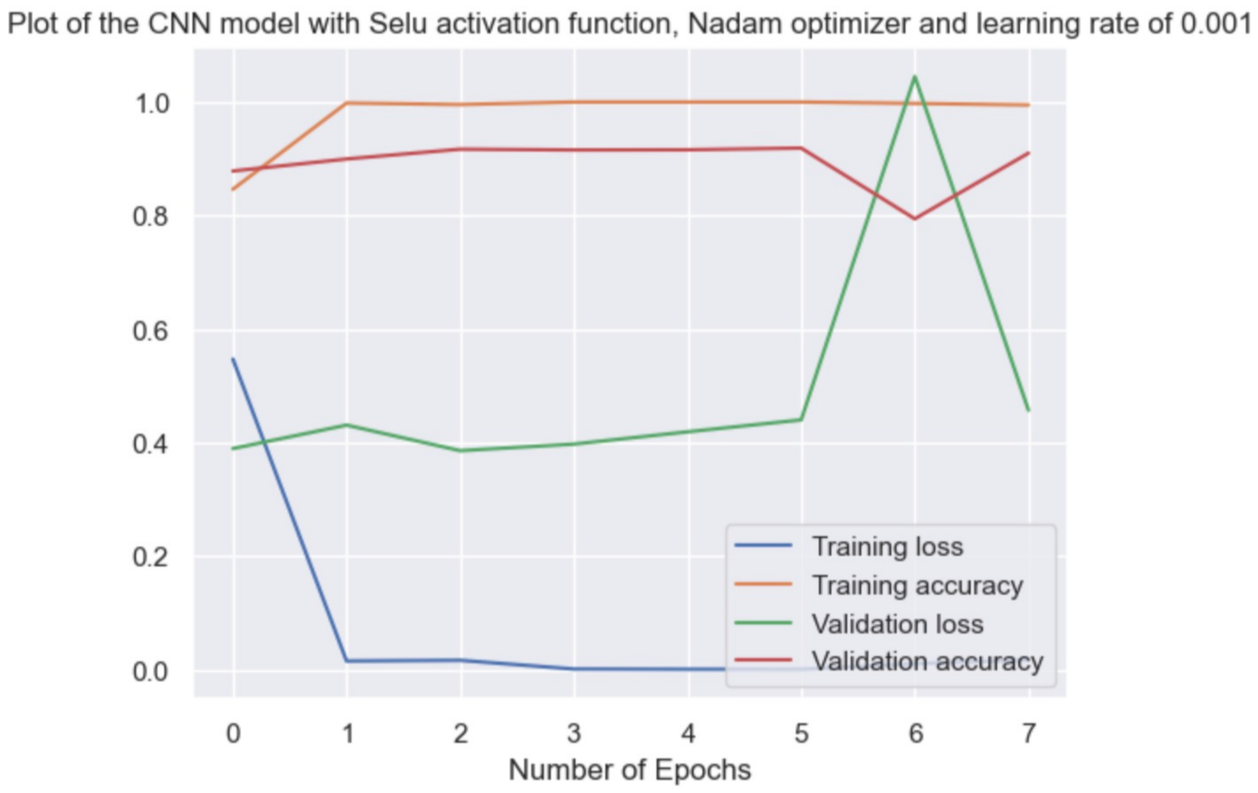
3.3 Perform a statistical test between the best and the second best models

Fitting the second best CNN model (Selu activation function, Nadam optimizer and learning rate of 0.001) on training data

In [45]:
```
# Training the second best CNN model with Selu activation function, Nadam optimizer and learning rate of 0.001 on the t
model_cnn_final_1 = keras.models.Sequential()
model_cnn_final_1.add(keras.layers.Conv2D(filters=hiddensizes[0], kernel_size=3,
                                          strides=1, activation='selu', padding="same", input_shape=[28, 28, 1]))
model_cnn_final_1.add(keras.layers.MaxPooling2D(pool_size=2))
for n in hiddensizes[1:-1]:
    model_cnn_final_1.add(keras.layers.Conv2D(filters=n, kernel_size=3, strides=1, padding="same", activation='selu'))
    model_cnn_final_1.add(keras.layers.MaxPooling2D(pool_size=2))
model_cnn_final_1.add(keras.layers.Conv2D(filters=hiddensizes[-1], kernel_size=3, strides=1,
                                          padding="same", activation='selu'))
model_cnn_final_1.add(keras.layers.Flatten())
model_cnn_final_1.add(keras.layers.Dense(24, activation = "softmax"))
model_cnn_final_1.compile(loss="sparse_categorical_crossentropy",
                          optimizer=keras.optimizers.Nadam(learning_rate=0.001), metrics=["accuracy"])
history_cnn_final_1 = model_cnn_final_1.fit(X_train, y_train, epochs=50,
                                            callbacks = early_stopping_cb, validation_data=(X_val, y_val))
```

```
Epoch 1/50
858/858 [==============================] - 22s 25ms/step - loss: 0.5470 - accuracy: 0.8462 - val_loss: 0.3893 - val_a
ccuracy: 0.8784
Epoch 2/50
858/858 [==============================] - 24s 28ms/step - loss: 0.0150 - accuracy: 0.9982 - val_loss: 0.4306 - val_a
ccuracy: 0.8996
Epoch 3/50
858/858 [==============================] - 24s 28ms/step - loss: 0.0162 - accuracy: 0.9956 - val_loss: 0.3856 - val_a
ccuracy: 0.9169
Epoch 4/50
858/858 [==============================] - 23s 27ms/step - loss: 7.7819e-04 - accuracy: 1.0000 - val_loss: 0.3972 - v
al_accuracy: 0.9155
Epoch 5/50
858/858 [==============================] - 21s 25ms/step - loss: 3.5748e-04 - accuracy: 1.0000 - val_loss: 0.4188 - v
al_accuracy: 0.9158
Epoch 6/50
858/858 [==============================] - 22s 26ms/step - loss: 2.0846e-04 - accuracy: 1.0000 - val_loss: 0.4398 - v
al_accuracy: 0.9189
Epoch 7/50
858/858 [==============================] - 23s 27ms/step - loss: 0.0102 - accuracy: 0.9975 - val_loss: 1.0450 - val_a
ccuracy: 0.7942
Epoch 8/50
858/858 [==============================] - 23s 27ms/step - loss: 0.0169 - accuracy: 0.9948 - val_loss: 0.4569 - val_a
ccuracy: 0.9099
```

In [47]:
```
# Plot of the second best CNN model with Selu activation function, Nadam optimizer and learning rate of 0.001
pd.DataFrame(history_cnn_final_1.history).plot()
plt.xlabel("Number of Epochs")
plt.title(f'Plot of the CNN model with Selu activation function, Nadam optimizer and learning rate of 0.001')
plt.legend(['Training loss', 'Training accuracy', 'Validation loss', 'Validation accuracy'],loc = 'lower right')
plt.show()
```

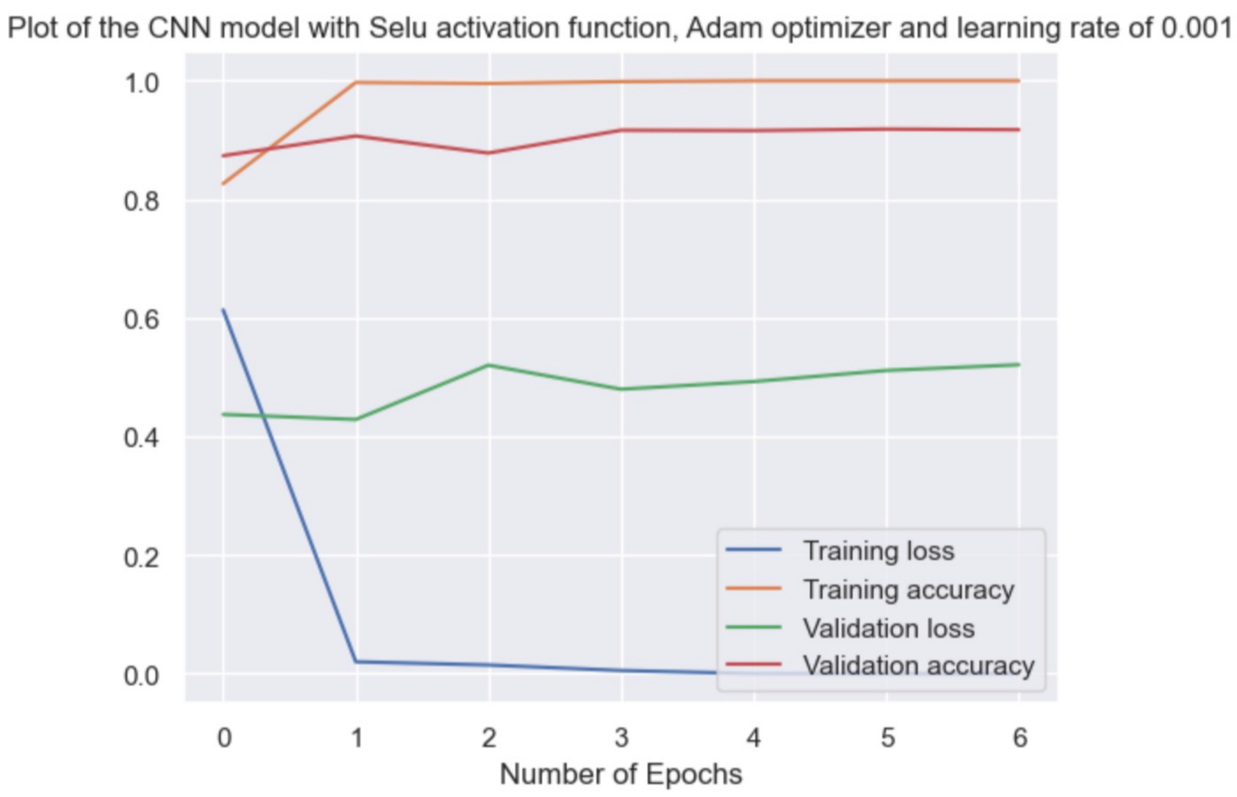


Fitting the best CNN model (Selu activation function, Adam optimizer and learning rate of 0.001) on training data

In [46]:
```
# Training the best CNN model with Selu activation function, Adam optimizer and learning rate of 0.001 on the training
model_cnn_final_2 = keras.models.Sequential()
model_cnn_final_2.add(keras.layers.Conv2D(filters=hiddensizes[0], kernel_size=3, strides=1, activation='selu', padding=
                            input_shape=[28, 28, 1]))
model_cnn_final_2.add(keras.layers.MaxPooling2D(pool_size=2))
for n in hiddensizes[1:-1]:
    model_cnn_final_2.add(keras.layers.Conv2D(filters=n, kernel_size=3, strides=1, padding="same", activation='selu'))
    model_cnn_final_2.add(keras.layers.MaxPooling2D(pool_size=2))
model_cnn_final_2.add(keras.layers.Conv2D(filters=hiddensizes[-1], kernel_size=3, strides=1, padding="same", activation
model_cnn_final_2.add(keras.layers.Flatten())
model_cnn_final_2.add(keras.layers.Dense(24, activation = "softmax"))
model_cnn_final_2.compile(loss="sparse_categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001),
history_cnn_final_2 = model_cnn_final_2.fit(X_train, y_train, epochs=50, callbacks = early_stopping_cb, validation_data
```

```
Epoch 1/50
858/858 [==============================] - 22s 25ms/step - loss: 0.6135 - accuracy: 0.8265 - val_loss: 0.4373 - val_a
ccuracy: 0.8737
Epoch 2/50
858/858 [==============================] - 23s 27ms/step - loss: 0.0202 - accuracy: 0.9972 - val_loss: 0.4288 - val_a
ccuracy: 0.9069
Epoch 3/50
858/858 [==============================] - 23s 27ms/step - loss: 0.0148 - accuracy: 0.9956 - val_loss: 0.5204 - val_a
ccuracy: 0.8781
Epoch 4/50
858/858 [==============================] - 24s 28ms/step - loss: 0.0054 - accuracy: 0.9987 - val_loss: 0.4798 - val_a
ccuracy: 0.9166
Epoch 5/50
858/858 [==============================] - 24s 28ms/step - loss: 3.2926e-04 - accuracy: 1.0000 - val_loss: 0.4928 - v
al_accuracy: 0.9161
Epoch 6/50
858/858 [==============================] - 22s 26ms/step - loss: 1.8261e-04 - accuracy: 1.0000 - val_loss: 0.5115 - v
al_accuracy: 0.9186
Epoch 7/50
858/858 [==============================] - 22s 25ms/step - loss: 1.1708e-04 - accuracy: 1.0000 - val_loss: 0.5211 - v
al_accuracy: 0.9175
```

In [48]:
```
# Plot of the best CNN model with Selu activation function, Adam optimizer and learning rate of 0.001
pd.DataFrame(history_cnn_final_2.history).plot()
plt.xlabel("Number of Epochs")
plt.title(f'Plot of the CNN model with Selu activation function, Adam optimizer and learning rate of 0.001')
plt.legend(['Training loss', 'Training accuracy', 'Validation loss', 'Validation accuracy'],loc = 'lower right')
plt.show()
```



Perform a statistical test between the best CNN model and the second best CNN model

In [49]:
```
# Making prediction from the best CNN model (with Selu activation function, Adam optimizer and learning rate of 0.001),
# and the second best CNN model (with Selu activation function, Nadam optimizer and learning rate of 0.001) on the vali
y_val_pred = []
y_val_pred += [np.argmax(model_cnn_final_2.predict(X_val),axis=1)]
y_val_pred += [np.argmax(model_cnn_final_1.predict(X_val),axis=1)]
print(y_val_pred)
```

```
113/113 [==============================] - 1s 7ms/step
113/113 [==============================] - 1s 7ms/step
[array([ 6,  5,  9, ..., 23,  7, 17]), array([ 6,  5,  9, ..., 23,  7, 13])]
```

In [50]:
```
from sklearn.metrics import accuracy_score
from mlxtend.evaluate import permutation_test
# Perform statistical test between the best and second-best models, to see if there is any significant difference in pe
p_value = permutation_test(y_val_pred[0], y_val_pred[1], paired=True,
                           func=lambda x, y: np.abs(accuracy_score(y_val,x) - accuracy_score(y_val,y)),
                           method="approximate", seed=1, num_rounds=1000)
print(f'With the threshold of 0.05, the P-value in comparing the two best models is {p_value}, which is below the thres
```

```
With the threshold of 0.05, the P-value in comparing the two best models is 0.028971028971028972, which is below the
threshold, hence the difference in performance between these two best models is significant.
```

The best CNN model with Selu activation function, Adam optimizer and learning rate of 0.001 converges with the validation accuracy of 0.9175, and the second best CNN model with Selu activation function, Nadam optimizer and learning rate of 0.001 converges with the validation accuracy is 0.9099. Additionally, the difference in the performance of these two best models is significant because the p-value in comparing these two best models is below 0.05, therefore, the CNN model with Selu activation function, Adam optimizer and learning rate of 0.001 is chosen to make predictions on the testing data.

4. Model predictions

In [51]:
```
# Evaluating the final CNN model with Selu activation function, Adam optimizer and learning rate of 0.001 on the testir
testres_1 = model_cnn_final_2.evaluate(X_test, y_test)
print('The testing loss of the final CNN model is: ', testres_1[0])
print('The testing accuracy of the final CNN model is: ', testres_1[1])
```

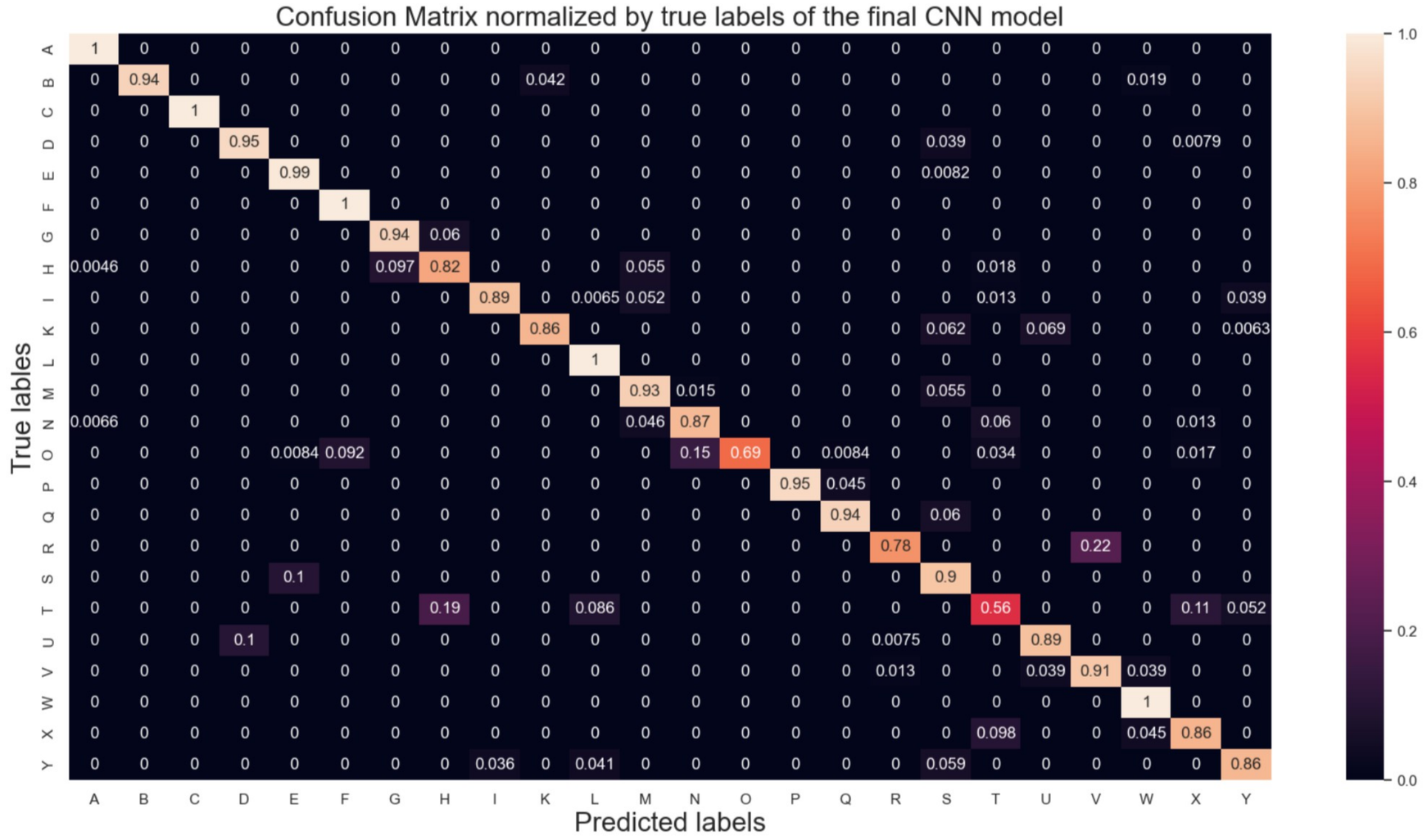
```
113/113 [==============================] - 1s 7ms/step - loss: 0.4283 - accuracy: 0.9044
The testing loss of the final CNN model is:  0.4282926917076111
The testing accuracy of the final CNN model is:  0.9043502807617188
```

In [52]:
```
# Making predictions from the final CNN model with Elu activation function, Nadam optimizer and learning rate of 0.001
y_pred = np.argmax(model_cnn_final_2.predict(X_test, verbose = 0),axis=1)
```

In [53]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns

y_test_label = y_test.apply(lambda x: class_names[x])
y_pred_label = pd.Series(y_pred).apply(lambda x: class_names[x])

conf_mat_norm = confusion_matrix(y_test_label, y_pred_label, normalize = 'true')
conf_mat_norm = pd.DataFrame(conf_mat_norm, index = class_names, columns = class_names)
f, ax = plt.subplots(figsize=(20, 10))
sns.heatmap(conf_mat_norm, annot = True)
plt.xlabel('Predicted labels', fontsize = 20)
plt.ylabel('True lables', fontsize = 20)
plt.title('Confusion Matrix normalized by true labels of the final CNN model', fontsize = 20, loc = "center")
plt.show()
```



According to the confusion matrix normalized by true labels, only 56% observations of the letter 'T' are predicted correctly. In addition, the letter 'O' has only 69% observations which are predicted correctly, and the letter 'R' has only 78% observations which are predicted correctly. The remaining letters have around 80% observations or more being predicted correctly. The letters 'A', 'C', 'F', 'L', 'W' are completely predicted correctly.

In [54]:
```
# Accuracy score for each individual letter
acc = pd.DataFrame({'Predicted labels': y_pred_label, 'True labels':y_test_label})
acc_v = acc[acc['Predicted labels'] == acc['True labels']]
acc_v = pd.DataFrame(acc_v.drop('True labels', axis = 1).value_counts(),
                     columns = ["Number of correct prediction"]).sort_values('Predicted labels')
count_pred = pd.DataFrame({'Predicted labels': y_pred_label}).value_counts()
count_pred = pd.DataFrame(count_pred, columns = ["Total number of prediction"]).sort_values('Predicted labels')
acc_total = pd.concat([acc_v, count_pred], axis=1)
acc_total['Accuracy_score'] = acc_total['Number of correct prediction']/acc_total['Total number of prediction']

print('The table of accuracy score of the final CNN model:')
acc_total.sort_values('Accuracy_score')
```

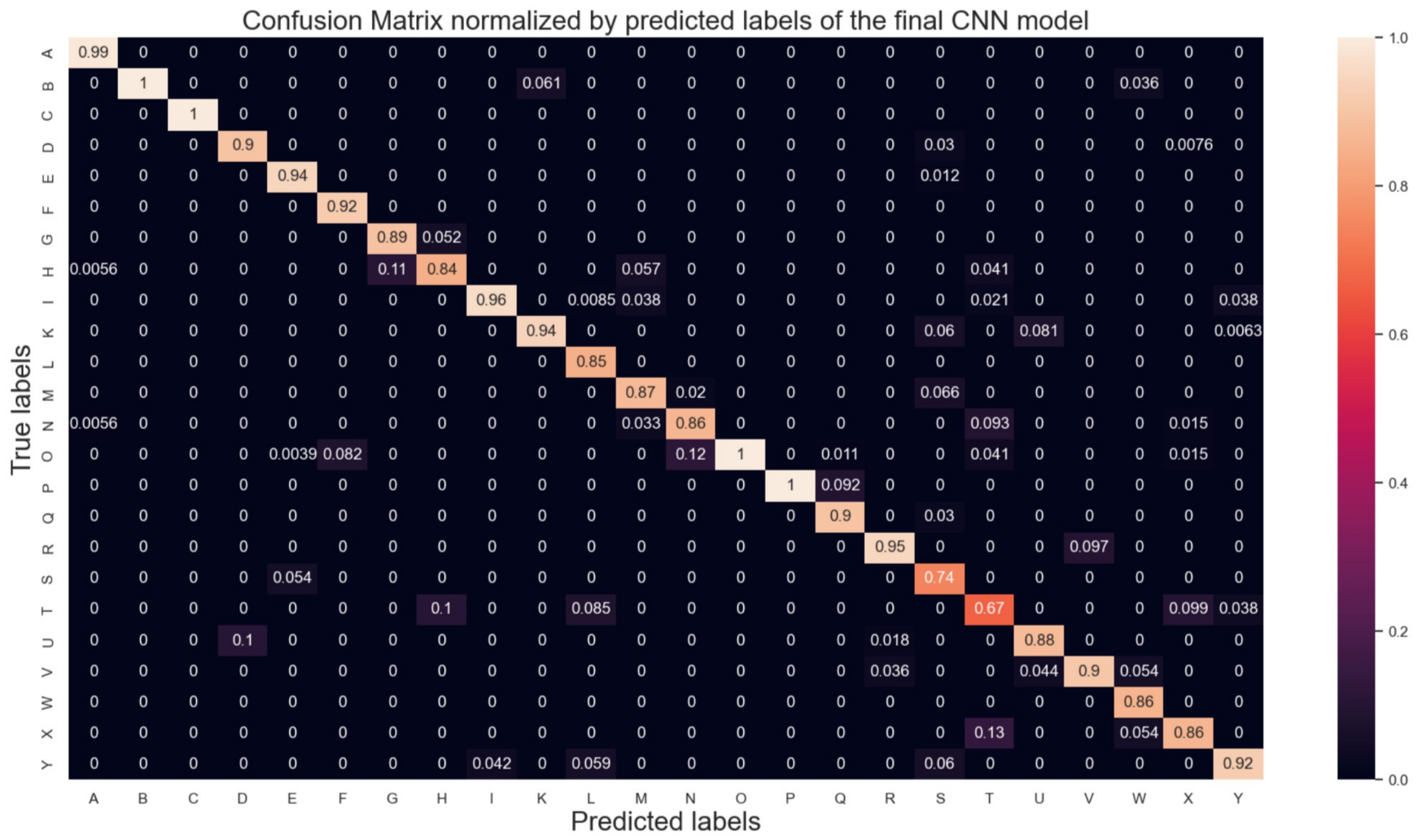
```
The table of accuracy score of the final CNN model:
```

Out[54]:

Predicted labels	Number of correct prediction	Total number of prediction	Accuracy_score

T	65	97	0.670103
S	124	167	0.742515
H	179	212	0.844340
L	100	118	0.847458
W	96	112	0.857143
X	113	131	0.862595
N	132	153	0.862745
M	185	212	0.872642
U	119	136	0.875000
G	173	194	0.891753
D	121	135	0.896296
Q	78	87	0.896552
V	140	155	0.903226
F	123	134	0.917910
Y	146	159	0.918239
K	138	147	0.938776
E	242	257	0.941634
R	52	55	0.945455
I	137	143	0.958042
A	177	179	0.988827
O	82	82	1.000000
C	154	154	1.000000
B	199	199	1.000000
P	168	168	1.000000

In [55]:
```
conf_mat = confusion_matrix(y_test, y_pred, normalize = "pred")
conf_mat = pd.DataFrame(conf_mat, index = class_names, columns = class_names)
f, ax = plt.subplots(figsize=(20, 10))
sns.heatmap(conf_mat, annot = True)
plt.xlabel('Predicted labels', fontsize = 20)
plt.ylabel('True labels', fontsize = 20)
plt.title('Confusion Matrix normalized by predicted labels of the final CNN model', fontsize = 20, loc = "center")
plt.show()
```



According to the confusion matrix normalized by predicted labels and the table of accuracy score of the final CNN model above, the letter 'T' has the lowest accuracy score (only 0.67) because among all letters are predicted as 'T', only 67% observations are truely 'T'. The letter 'S' has the second lowest accuracy score because among all letters are predicted as 'S', only about 74% observations are truely 'S'.

In [56]:
```
# The most common misclassification
diff = pd.DataFrame({'True labels':y_test_label, 'Predicted labels': y_pred_label})
diff = diff.loc[diff['True labels'] != diff['Predicted labels']].reset_index(drop = True)
nrow = len(diff)
diff_v = pd.DataFrame(diff.groupby("True labels").value_counts(), columns = ["Count"])
diff_v['Proportion'] = diff_v['Count']/nrow
count_v = diff_v.groupby("True labels").sum('Proportion')
```

In [57]:
```
print("Table shows the count and the proportion of misclassification for each letter: \n")
count_v.sort_values('Proportion')
```

```
Table shows the count and the proportion of misclassification for each letter:
```

Out[57]:

True labels	Count	Proportion
E	2	0.005831
Q	5	0.014577
D	6	0.017493
P	8	0.023324
G	11	0.032070
B	13	0.037901

S	14	0.040816
V	14	0.040816
M	14	0.040816
R	15	0.043732
U	15	0.043732
I	17	0.049563
N	19	0.055394
X	19	0.055394
K	22	0.064140
Y	23	0.067055
O	37	0.107872
H	38	0.110787
T	51	0.148688

In [58]:
```
print('Table shows the count and the proportion of misclassification for each combination of True labels and Predicted
diff_v
```

```
Table shows the count and the proportion of misclassification for each combination of True labels and Predicted label
s:
```

Out[58]:

		Count	Proportion
True labels	**Predicted labels**		
B	**K**	9	0.026239
	W	4	0.011662
D	**S**	5	0.014577
	X	1	0.002915
E	**S**	2	0.005831
G	**H**	11	0.032070
H	**G**	21	0.061224
	M	12	0.034985
	T	4	0.011662
	A	1	0.002915
I	**M**	8	0.023324
	Y	6	0.017493
	T	2	0.005831
	L	1	0.002915
K	**U**	11	0.032070
	S	10	0.029155
	Y	1	0.002915
M	**S**	11	0.032070
	N	3	0.008746
N	**T**	9	0.026239
	M	7	0.020408
	X	2	0.005831
	A	1	0.002915
O	**N**	18	0.052478
	F	11	0.032070
	T	4	0.011662
	X	2	0.005831
	Q	1	0.002915
	E	1	0.002915
P	**Q**	8	0.023324
Q	**S**	5	0.014577
R	**V**	15	0.043732
S	**E**	14	0.040816
T	**H**	22	0.064140
	X	13	0.037901
	L	10	0.029155
	Y	6	0.017493
U	**D**	14	0.040816
	R	1	0.002915
V	**U**	6	0.017493
	W	6	0.017493
	R	2	0.005831
X	**T**	13	0.037901
	W	6	0.017493
Y	**S**	10	0.029155
	L	7	0.020408
	I	6	0.017493

According to the table illustrating the number of the proportion of misclassification for each letter above, the letter 'T' is the most common letter which are predicted incorrectly. Because in all of misclassifications of this final CNN model, the number of missification of the letter 'T' is 51 (accounts for around 15% of total number of misclassification). Particularly, the letter 'T' is misclassified as 'H', 'X', 'L', and 'Y' according to the table showing the count and the proportion of misclassification for each combination of True labels and Predicted labels above. The second most common letter which are misclassified is the letter 'H' with about 11% of total number of misclassification. This letter 'H' is more misclassified as 'G' and 'N'. The third most misclassified letter is the letter 'O' with about 10.8% of total number of misclassification, and this letter is more likely to be predicted as 'N' and 'F'.

Although the accuracy of this final Convolutional Neural Network (CNN) model is high, this accuracy score does not meet the required accuracy score of 96% since its accuracy score is only about 90.4%. Therefore, for further improvement, the number of hidden layers and the size of these hidden layers should be tuned to examine if they can improve the performance of this model, since there are only three hyperparameters that are currently tuned for this CNN model (the activation function, the optimizer and its learning rate). In addition, some other methods such as Batch Normalization and Residual Unit should be considered, since Residual Unit can mitigate overfitting problem, which means that it can reduce the gap between training accuracy and validation accuracy of this model. Last, the training dataset and the testing dataset may not comprehensive since the letter 'J' and 'Z' do not appear in these datasets, which can lead to some misclassifications and a decrease in the overall accuracy of the model.

In [ ]: