# Image_ClassificationCIFAR100

April 13, 2020

[0]:
```
from __future__ import print_function
import keras
from keras.datasets import cifar100
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
import keras.backend as K
import tensorflow as tf

import os
import pickle
import numpy as np
import h5py
import pandas as pd
import matplotlib.pyplot as plt
```

## 0.1 Downloading DataSet

We use the dataset CIFAR100 for Multiclass Classification

- Batch_size: amount of samples that will be feed-forward in our model at once
- Img_width = 32, Img_height = 32, channels = 3 as RGB image
- loss_function: compare predictions with ground truth during training
- number of classes: 100
- optimizer: method by which we update the weights of our neural network

[4]:
```
num_classes = 100
save_dir = os.path.join(os.getcwd(), 'saved_models')
model_name = 'cifar100.h5'

# The data, shuffled and split between train and test sets:
(x_train, y_train), (x_test, y_test) = cifar100.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
```

```python
print(x_test.shape[0], 'test samples')

# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255.
x_test /= 255.
```

```
x_train shape: (50000, 32, 32, 3)
50000 train samples
10000 test samples
```

## 0.2 Network

[0]:
```python
model = Sequential()

model.add(Conv2D(128, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('elu'))
model.add(Conv2D(128, (3, 3)))
model.add(Activation('elu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3), padding='same'))
model.add(Activation('elu'))
model.add(Conv2D(256, (3, 3)))
model.add(Activation('elu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(512, (3, 3), padding='same'))
model.add(Activation('elu'))
model.add(Conv2D(512, (3, 3)))
model.add(Activation('elu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(1024))
model.add(Activation('elu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

```
# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)

# training the model using RMSprop
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

# 1 Model Fitting

[8]:
```
epochs = 50
num_predictions = 20
batch_size = 64
validations = []
history = model.fit(x_train,␣
 ↪y_train,batch_size=batch_size,epochs=epochs,validation_data=(x_test,␣
 ↪y_test),shuffle=True)
pickle.dump(validations, open("loss_validation.p",'wb'))
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/50
50000/50000 [==============================] - 63s 1ms/step - loss: 2.1612 -
accuracy: 0.4467 - val_loss: 2.1108 - val_accuracy: 0.4639
Epoch 2/50
50000/50000 [==============================] - 63s 1ms/step - loss: 2.0398 -
accuracy: 0.4723 - val_loss: 2.0401 - val_accuracy: 0.4722
Epoch 3/50
50000/50000 [==============================] - 63s 1ms/step - loss: 1.9221 -
accuracy: 0.4950 - val_loss: 1.9920 - val_accuracy: 0.4933
Epoch 4/50
50000/50000 [==============================] - 63s 1ms/step - loss: 1.8114 -
accuracy: 0.5202 - val_loss: 1.9699 - val_accuracy: 0.4927
Epoch 5/50
50000/50000 [==============================] - 62s 1ms/step - loss: 1.6899 -
accuracy: 0.5494 - val_loss: 1.9667 - val_accuracy: 0.4923
Epoch 6/50
50000/50000 [==============================] - 62s 1ms/step - loss: 1.5766 -
accuracy: 0.5765 - val_loss: 1.9137 - val_accuracy: 0.5073
Epoch 7/50
50000/50000 [==============================] - 62s 1ms/step - loss: 1.4688 -
accuracy: 0.6005 - val_loss: 1.9536 - val_accuracy: 0.5080
Epoch 8/50
50000/50000 [==============================] - 62s 1ms/step - loss: 1.3625 -
accuracy: 0.6215 - val_loss: 1.8924 - val_accuracy: 0.5223
Epoch 9/50
50000/50000 [==============================] - 62s 1ms/step - loss: 1.2707 -
```

```
accuracy: 0.6465 - val_loss: 1.8472 - val_accuracy: 0.5310
Epoch 10/50
50000/50000 [==============================] - 62s 1ms/step - loss: 1.1744 -
accuracy: 0.6705 - val_loss: 1.8820 - val_accuracy: 0.5302
Epoch 11/50
50000/50000 [==============================] - 62s 1ms/step - loss: 1.0815 -
accuracy: 0.6911 - val_loss: 1.8863 - val_accuracy: 0.5297
Epoch 12/50
50000/50000 [==============================] - 62s 1ms/step - loss: 0.9893 -
accuracy: 0.7154 - val_loss: 1.9413 - val_accuracy: 0.5269
Epoch 13/50
50000/50000 [==============================] - 62s 1ms/step - loss: 0.9133 -
accuracy: 0.7347 - val_loss: 1.9460 - val_accuracy: 0.5364
Epoch 14/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.8345 -
accuracy: 0.7546 - val_loss: 1.9120 - val_accuracy: 0.5368
Epoch 15/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.7737 -
accuracy: 0.7707 - val_loss: 1.9976 - val_accuracy: 0.5280
Epoch 16/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.7010 -
accuracy: 0.7888 - val_loss: 1.9605 - val_accuracy: 0.5407
Epoch 17/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.6459 -
accuracy: 0.8051 - val_loss: 1.9510 - val_accuracy: 0.5394
Epoch 18/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.5878 -
accuracy: 0.8205 - val_loss: 2.0290 - val_accuracy: 0.5434
Epoch 19/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.5398 -
accuracy: 0.8350 - val_loss: 2.0400 - val_accuracy: 0.5385
Epoch 20/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.4978 -
accuracy: 0.8461 - val_loss: 2.1446 - val_accuracy: 0.5496
Epoch 21/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.4564 -
accuracy: 0.8581 - val_loss: 2.0576 - val_accuracy: 0.5379
Epoch 22/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.4289 -
accuracy: 0.8669 - val_loss: 2.2093 - val_accuracy: 0.5501
Epoch 23/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.3993 -
accuracy: 0.8765 - val_loss: 2.2060 - val_accuracy: 0.5466
Epoch 24/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.3683 -
accuracy: 0.8849 - val_loss: 2.1630 - val_accuracy: 0.5380
Epoch 25/50
50000/50000 [==============================] - 64s 1ms/step - loss: 0.3493 -
```

```
accuracy: 0.8924 - val_loss: 2.1684 - val_accuracy: 0.5444
Epoch 26/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.3204 -
accuracy: 0.9000 - val_loss: 2.2401 - val_accuracy: 0.5426
Epoch 27/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.3044 -
accuracy: 0.9057 - val_loss: 2.2343 - val_accuracy: 0.5401
Epoch 28/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.2868 -
accuracy: 0.9104 - val_loss: 2.3049 - val_accuracy: 0.5437
Epoch 29/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.2707 -
accuracy: 0.9157 - val_loss: 2.3941 - val_accuracy: 0.5406
Epoch 30/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.2590 -
accuracy: 0.9206 - val_loss: 2.4278 - val_accuracy: 0.5456
Epoch 31/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.2407 -
accuracy: 0.9243 - val_loss: 2.3932 - val_accuracy: 0.5455
Epoch 32/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.2358 -
accuracy: 0.9268 - val_loss: 2.4607 - val_accuracy: 0.5510
Epoch 33/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.2301 -
accuracy: 0.9288 - val_loss: 2.4346 - val_accuracy: 0.5411
Epoch 34/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.2206 -
accuracy: 0.9312 - val_loss: 2.4728 - val_accuracy: 0.5487
Epoch 35/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.2079 -
accuracy: 0.9358 - val_loss: 2.6035 - val_accuracy: 0.5439
Epoch 36/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.2057 -
accuracy: 0.9358 - val_loss: 2.2766 - val_accuracy: 0.5425
Epoch 37/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.1968 -
accuracy: 0.9399 - val_loss: 2.3900 - val_accuracy: 0.5441
Epoch 38/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.1909 -
accuracy: 0.9410 - val_loss: 2.4747 - val_accuracy: 0.5465
Epoch 39/50
50000/50000 [==============================] - 64s 1ms/step - loss: 0.1879 -
accuracy: 0.9418 - val_loss: 2.4327 - val_accuracy: 0.5454
Epoch 40/50
50000/50000 [==============================] - 64s 1ms/step - loss: 0.1770 -
accuracy: 0.9453 - val_loss: 2.4911 - val_accuracy: 0.5500
Epoch 41/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.1757 -
```

```
accuracy: 0.9454 - val_loss: 2.2837 - val_accuracy: 0.5423
Epoch 42/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.1706 -
accuracy: 0.9460 - val_loss: 2.5532 - val_accuracy: 0.5392
Epoch 43/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.1644 -
accuracy: 0.9483 - val_loss: 2.3649 - val_accuracy: 0.5448
Epoch 44/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.1651 -
accuracy: 0.9497 - val_loss: 2.3895 - val_accuracy: 0.5465
Epoch 45/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.1651 -
accuracy: 0.9504 - val_loss: 2.8186 - val_accuracy: 0.5517
Epoch 46/50
50000/50000 [==============================] - 64s 1ms/step - loss: 0.1588 -
accuracy: 0.9517 - val_loss: 2.8806 - val_accuracy: 0.5456
Epoch 47/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.1553 -
accuracy: 0.9528 - val_loss: 2.6195 - val_accuracy: 0.5504
Epoch 48/50
50000/50000 [==============================] - 64s 1ms/step - loss: 0.1518 -
accuracy: 0.9523 - val_loss: 2.6581 - val_accuracy: 0.5434
Epoch 49/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.1514 -
accuracy: 0.9539 - val_loss: 2.2576 - val_accuracy: 0.5336
Epoch 50/50
50000/50000 [==============================] - 63s 1ms/step - loss: 0.1499 -
accuracy: 0.9551 - val_loss: 2.1556 - val_accuracy: 0.5275
```

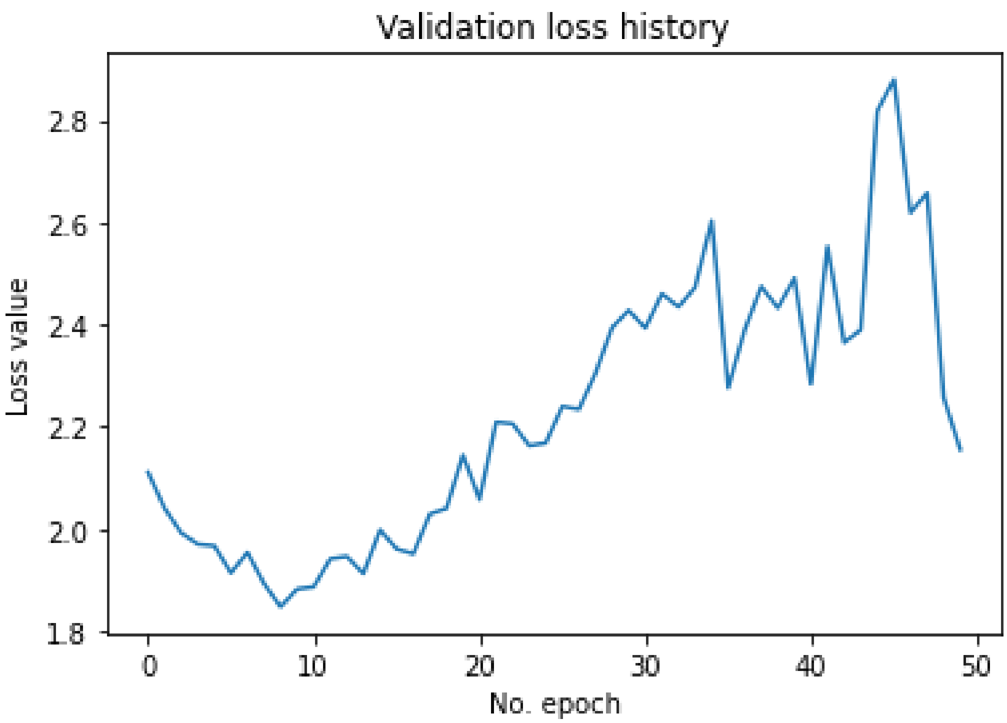
## 2 Plots for Categorical Cross Entropy Loss

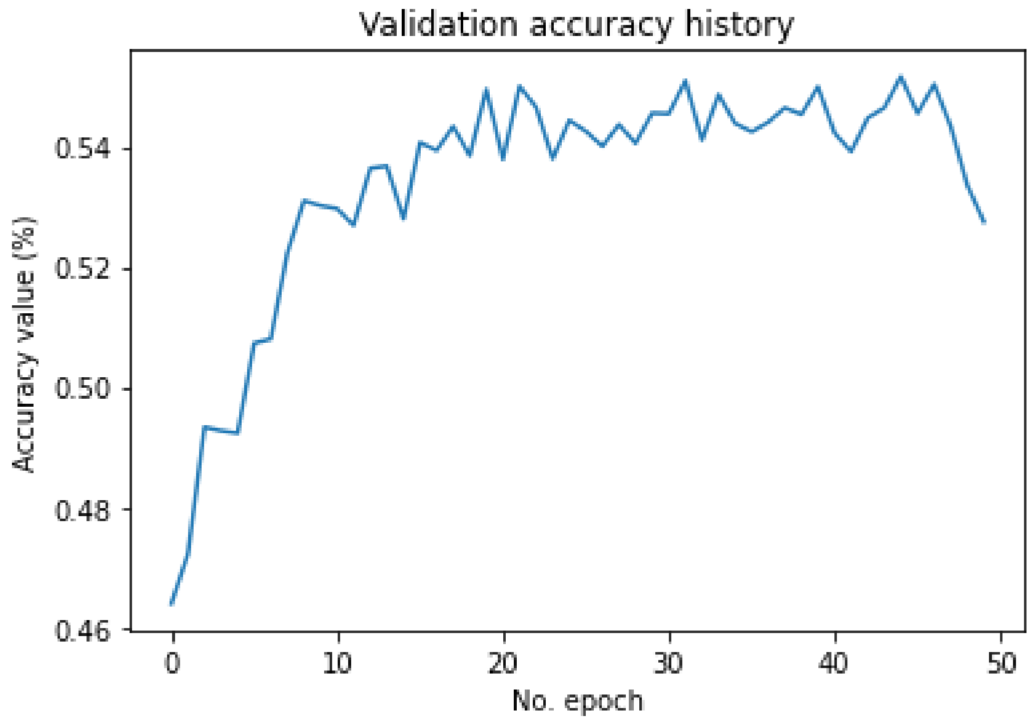
Also called Softmax Loss. It is a Softmax activation plus a Cross-Entropy loss. If we use this loss, we will train a CNN to output a probability over the C classes for each image. It is used for multi-class classification.

[9]:
```
# model.evaluate(x=x_test, y=y_test)

# Generate generalization metrics
score = model.evaluate(x=x_test, y=y_test, verbose=1)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')

# Visualize history
# Plot history: Loss
plt.plot(history.history['val_loss'])
plt.title('Validation loss history')
plt.ylabel('Loss value')
plt.xlabel('No. epoch')
```

```
plt.show()

# Plot history: Accuracy
plt.plot(history.history['val_accuracy'])
plt.title('Validation accuracy history')
plt.ylabel('Accuracy value (%)')
plt.xlabel('No. epoch')
plt.show()
```

```
10000/10000 [==============================] - 5s 523us/step
Test loss: 2.1555932670593263 / Test accuracy: 0.5274999737739563
```






[0]:
```
ans=model.predict(x_test)
```

[11]:
```
print(ans.shape)
```

```
(10000, 100)
```

[0]:
```
final_label=np.zeros(10000)
gt=np.zeros(10000)

for i in range(10000):
  final_label[i]=np.argmax(ans[i,:])
  gt[i]=np.argmax(y_test[i,:])
```

[13]:
```
count=0
for i in range(10000):
  if(gt[i]==final_label[i]):
    count=count+1

print("Test set accuracy:",count/10000)
```

```
Test set accuracy: 0.5275
```

[0]:
```python
def shuffle_weights(model, weights=None):
    weights = model.get_weights()
    weights = [np.random.permutation(w.flat).reshape(w.shape) for w in weights]
    model.set_weights(weights)
```

# 3 Training the model with mean squared logarithmic loss

[21]:
```python
shuffle_weights(model)
model.compile(loss='mean_squared_logarithmic_error',
              optimizer=opt,
              metrics=['accuracy'])
history = model.fit(x_train,␣
 ↪y_train,batch_size=batch_size,epochs=20,validation_data=(x_test,␣
 ↪y_test),shuffle=True)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0047 -
accuracy: 0.0619 - val_loss: 0.0046 - val_accuracy: 0.1335
Epoch 2/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0046 -
accuracy: 0.1358 - val_loss: 0.0044 - val_accuracy: 0.1968
Epoch 3/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0044 -
accuracy: 0.1914 - val_loss: 0.0042 - val_accuracy: 0.2454
Epoch 4/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0042 -
accuracy: 0.2427 - val_loss: 0.0040 - val_accuracy: 0.2827
Epoch 5/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0041 -
accuracy: 0.2796 - val_loss: 0.0039 - val_accuracy: 0.3196
Epoch 6/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0039 -
accuracy: 0.3167 - val_loss: 0.0038 - val_accuracy: 0.3382
Epoch 7/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0038 -
accuracy: 0.3433 - val_loss: 0.0037 - val_accuracy: 0.3756
Epoch 8/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0037 -
accuracy: 0.3696 - val_loss: 0.0036 - val_accuracy: 0.3814
Epoch 9/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0036 -
accuracy: 0.3952 - val_loss: 0.0036 - val_accuracy: 0.3949
Epoch 10/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0035 -
accuracy: 0.4178 - val_loss: 0.0035 - val_accuracy: 0.4089
```

```
Epoch 11/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0034 -
accuracy: 0.4383 - val_loss: 0.0035 - val_accuracy: 0.4136
Epoch 12/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0033 -
accuracy: 0.4596 - val_loss: 0.0034 - val_accuracy: 0.4269
Epoch 13/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0032 -
accuracy: 0.4796 - val_loss: 0.0034 - val_accuracy: 0.4328
Epoch 14/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0031 -
accuracy: 0.4990 - val_loss: 0.0034 - val_accuracy: 0.4371
Epoch 15/20
50000/50000 [==============================] - 62s 1ms/step - loss: 0.0030 -
accuracy: 0.5178 - val_loss: 0.0033 - val_accuracy: 0.4534
Epoch 16/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0029 -
accuracy: 0.5314 - val_loss: 0.0033 - val_accuracy: 0.4585
Epoch 17/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0028 -
accuracy: 0.5488 - val_loss: 0.0033 - val_accuracy: 0.4608
Epoch 18/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0027 -
accuracy: 0.5666 - val_loss: 0.0033 - val_accuracy: 0.4655
Epoch 19/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0026 -
accuracy: 0.5802 - val_loss: 0.0033 - val_accuracy: 0.4680
Epoch 20/20
50000/50000 [==============================] - 63s 1ms/step - loss: 0.0026 -
accuracy: 0.5942 - val_loss: 0.0033 - val_accuracy: 0.4608
```

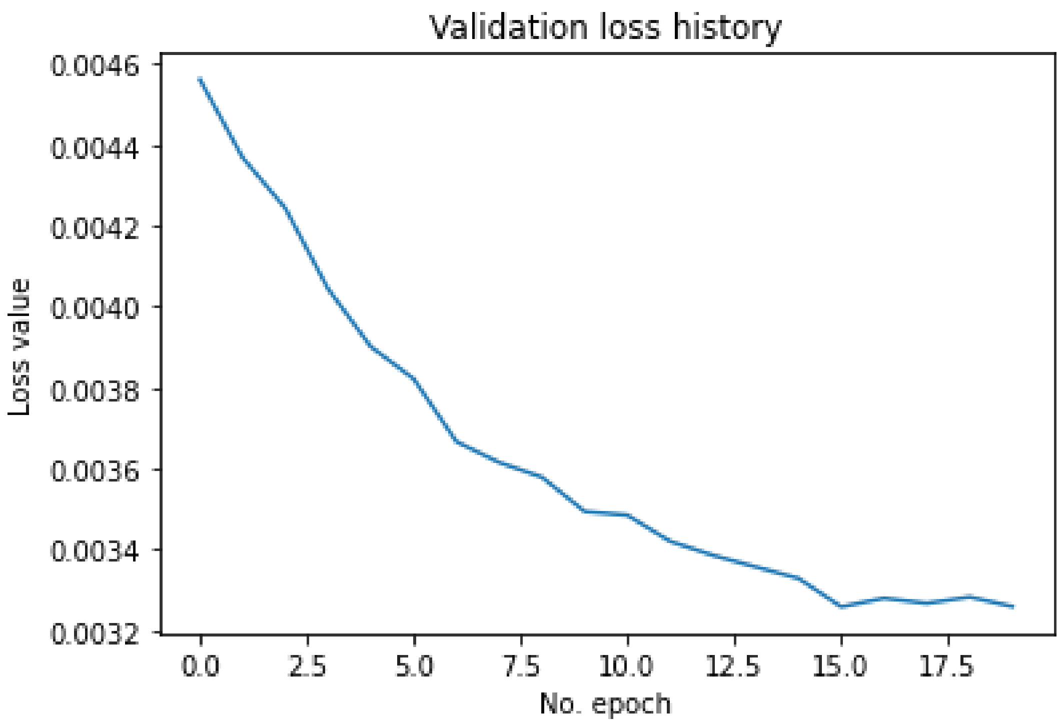
## 4 Plots for Mean Square Logarithmic Loss

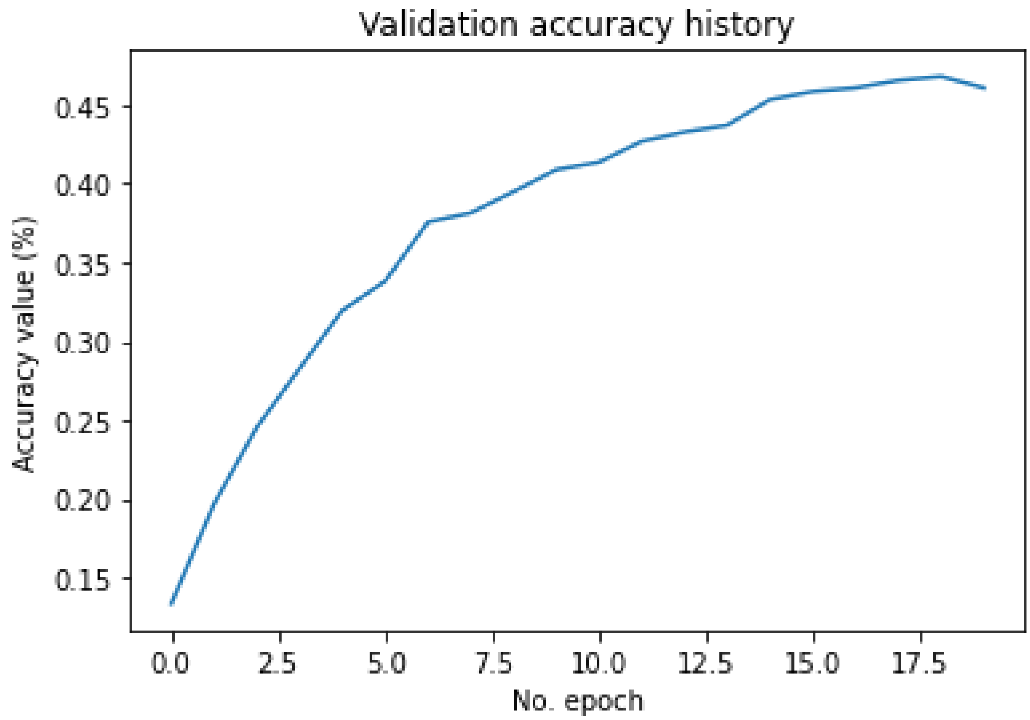
Mean squared logarithmic error (MSLE) can be interpreted as a measure of the ratio between the true and predicted values. Mean squared logarithmic error is, as the name suggests, a variation of the Mean Squared Error.

[22]:
```python
# Generate generalization metrics
score = model.evaluate(x=x_test, y=y_test, verbose=1)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')

# Visualize history
# Plot history: Loss
plt.plot(history.history['val_loss'])
plt.title('Validation loss history')
plt.ylabel('Loss value')
plt.xlabel('No. epoch')
```

```
plt.show()

# Plot history: Accuracy
plt.plot(history.history['val_accuracy'])
plt.title('Validation accuracy history')
plt.ylabel('Accuracy value (%)')
plt.xlabel('No. epoch')
plt.show()
```

```
10000/10000 [==============================] - 5s 470us/step
Test loss: 0.003258741495758295 / Test accuracy: 0.4607999920845032
```






[0]:
```
ans=model.predict(x_test)
final_label=np.zeros(10000)
gt=np.zeros(10000)

for i in range(10000):
  final_label[i]=np.argmax(ans[i,:])
  gt[i]=np.argmax(y_test[i,:])
```

[24]:
```
count=0
for i in range(10000):
  if(gt[i]==final_label[i]):
    count=count+1

print("Test set accuracy:",count/10000)
```

```
Test set accuracy: 0.4608
```

[0]:
```
def shuffle_weights(model, weights=None):
    weights = model.get_weights()
    weights = [np.random.permutation(w.flat).reshape(w.shape) for w in weights]
    model.set_weights(weights)
```

## 5 Using Hinge Loss for Training

[26]:
```
shuffle_weights(model)
model.compile(loss='hinge',
              optimizer=opt,
              metrics=['accuracy'])
history = model.fit(x_train,␣
 ↪y_train,batch_size=batch_size,epochs=20,validation_data=(x_test,␣
 ↪y_test),shuffle=True)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/20
50000/50000 [==============================] - 63s 1ms/step - loss: 1.0098 -
accuracy: 0.0101 - val_loss: 1.0098 - val_accuracy: 0.0100
Epoch 2/20
50000/50000 [==============================] - 62s 1ms/step - loss: 1.0098 -
accuracy: 0.0099 - val_loss: 1.0098 - val_accuracy: 0.0100
Epoch 3/20
50000/50000 [==============================] - 62s 1ms/step - loss: 1.0098 -
accuracy: 0.0106 - val_loss: 1.0097 - val_accuracy: 0.0160
Epoch 4/20
50000/50000 [==============================] - 63s 1ms/step - loss: 1.0095 -
accuracy: 0.0235 - val_loss: 1.0095 - val_accuracy: 0.0240
Epoch 5/20
50000/50000 [==============================] - 63s 1ms/step - loss: 1.0094 -
accuracy: 0.0320 - val_loss: 1.0093 - val_accuracy: 0.0369
Epoch 6/20
50000/50000 [==============================] - 63s 1ms/step - loss: 1.0093 -
accuracy: 0.0361 - val_loss: 1.0093 - val_accuracy: 0.0341
Epoch 7/20
50000/50000 [==============================] - 62s 1ms/step - loss: 1.0092 -
accuracy: 0.0404 - val_loss: 1.0090 - val_accuracy: 0.0489
Epoch 8/20
50000/50000 [==============================] - 62s 1ms/step - loss: 1.0091 -
accuracy: 0.0453 - val_loss: 1.0090 - val_accuracy: 0.0497
Epoch 9/20
50000/50000 [==============================] - 63s 1ms/step - loss: 1.0090 -
accuracy: 0.0496 - val_loss: 1.0089 - val_accuracy: 0.0532
Epoch 10/20
50000/50000 [==============================] - 63s 1ms/step - loss: 1.0090 -
accuracy: 0.0519 - val_loss: 1.0089 - val_accuracy: 0.0560
Epoch 11/20
50000/50000 [==============================] - 63s 1ms/step - loss: 1.0089 -
accuracy: 0.0558 - val_loss: 1.0088 - val_accuracy: 0.0602
Epoch 12/20
50000/50000 [==============================] - 62s 1ms/step - loss: 1.0088 -
accuracy: 0.0619 - val_loss: 1.0085 - val_accuracy: 0.0724
```

```
Epoch 13/20
50000/50000 [==============================] - 63s 1ms/step - loss: 1.0085 -
accuracy: 0.0731 - val_loss: 1.0084 - val_accuracy: 0.0785
Epoch 14/20
50000/50000 [==============================] - 62s 1ms/step - loss: 1.0085 -
accuracy: 0.0765 - val_loss: 1.0085 - val_accuracy: 0.0775
Epoch 15/20
50000/50000 [==============================] - 62s 1ms/step - loss: 1.0084 -
accuracy: 0.0809 - val_loss: 1.0082 - val_accuracy: 0.0908
Epoch 16/20
50000/50000 [==============================] - 62s 1ms/step - loss: 1.0083 -
accuracy: 0.0832 - val_loss: 1.0082 - val_accuracy: 0.0900
Epoch 17/20
50000/50000 [==============================] - 62s 1ms/step - loss: 1.0083 -
accuracy: 0.0862 - val_loss: 1.0081 - val_accuracy: 0.0969
Epoch 18/20
50000/50000 [==============================] - 62s 1ms/step - loss: 1.0082 -
accuracy: 0.0891 - val_loss: 1.0083 - val_accuracy: 0.0871
Epoch 19/20
50000/50000 [==============================] - 62s 1ms/step - loss: 1.0082 -
accuracy: 0.0883 - val_loss: 1.0081 - val_accuracy: 0.0933
Epoch 20/20
50000/50000 [==============================] - 62s 1ms/step - loss: 1.0082 -
accuracy: 0.0903 - val_loss: 1.0082 - val_accuracy: 0.0897
```

## 6 Plots for Hinge Loss

Hinge loss is a loss function used for training classifiers. The hinge loss is used for "maximum-margin" classification, most notably for support vector machines (SVMs).

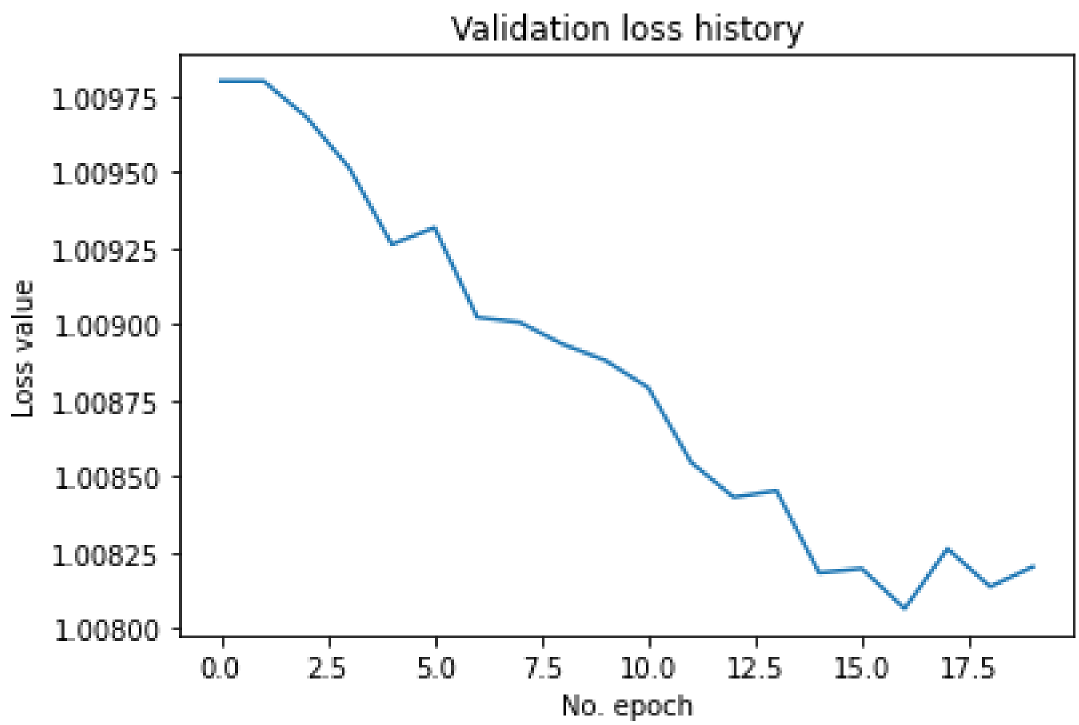
For an intended output t = ±1 and a classifier score y, the hinge loss of the prediction y is defined as

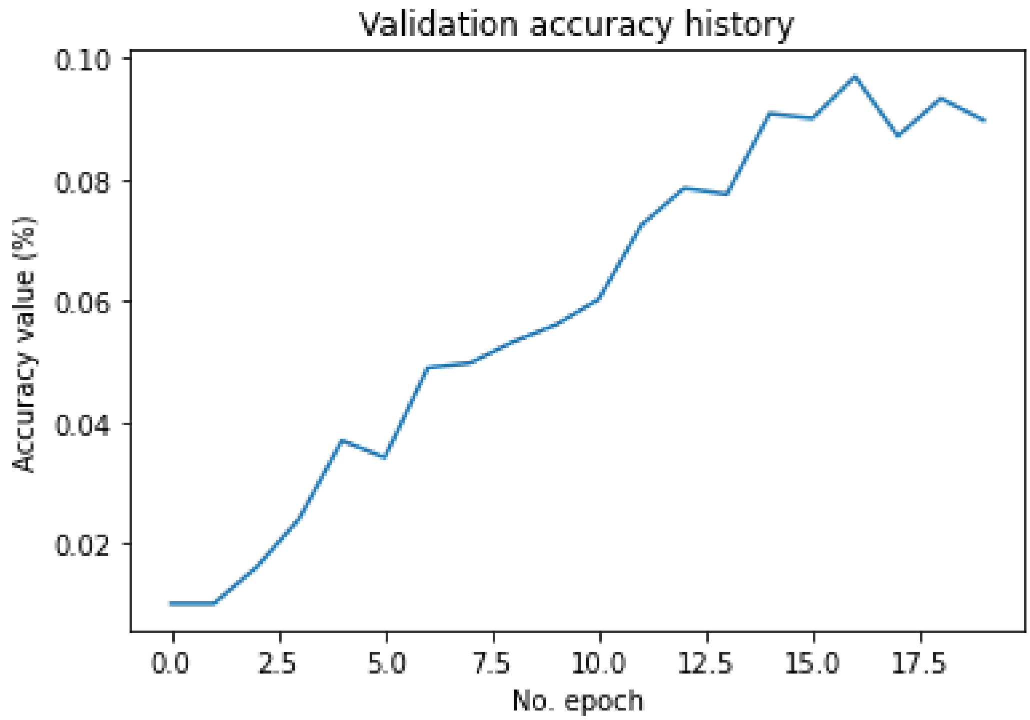
loss(y) = max(0, 1-t*y)

[27]:
```python
# Generate generalization metrics
score = model.evaluate(x=x_test, y=y_test, verbose=1)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')

# Visualize history
# Plot history: Loss
plt.plot(history.history['val_loss'])
plt.title('Validation loss history')
plt.ylabel('Loss value')
plt.xlabel('No. epoch')
plt.show()

# Plot history: Accuracy
```

```
plt.plot(history.history['val_accuracy'])
plt.title('Validation accuracy history')
plt.ylabel('Accuracy value (%)')
plt.xlabel('No. epoch')
plt.show()
```

```
10000/10000 [==============================] - 5s 468us/step
Test loss: 1.008201333618164 / Test accuracy: 0.08969999849796295
```






[0]:
```
ans=model.predict(x_test)
final_label=np.zeros(10000)
gt=np.zeros(10000)

for i in range(10000):
  final_label[i]=np.argmax(ans[i,:])
  gt[i]=np.argmax(y_test[i,:])
```

[29]:
```
count=0
for i in range(10000):
  if(gt[i]==final_label[i]):
    count=count+1

print("Test set accuracy:",count/10000)
```

```
Test set accuracy: 0.0897
```

[0]:
```
```