Question 1

(a.i) $$L(\mu,\sigma^2)=\prod_{i=1}^{n}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right)$$

$$=\left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n\exp\left(-\sum_{i=1}^{n}\frac{(x_i-\mu)^2}{2\sigma^2}\right)$$

$$=(2\pi\sigma^2)^{-\frac{n}{2}}\exp\left(-\sum_{i=1}^{n}\frac{(x_i-\mu)^2}{2\sigma^2}\right)$$

$$\ln L(\mu,\sigma^2)=-\frac{n}{2}\ln 2\pi\sigma^2-\sum_{i=1}^{n}\frac{(x_i-\mu)^2}{2\sigma^2}$$

$$=-\frac{n}{2}\ln 2\pi-\frac{n}{2}\ln\sigma^2-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2$$

$$\begin{cases}\dfrac{\partial \ln L}{\partial \mu}=\dfrac{1}{\sigma^2}\displaystyle\sum_{i=1}^{n}(x_i-\mu)=0\\[2ex]\dfrac{\partial \ln L}{\partial \sigma^2}=-\dfrac{n}{2\sigma^2}+\dfrac{1}{2\sigma^4}\displaystyle\sum_{i=1}^{n}(x_i-\mu)^2=0\end{cases}\Rightarrow\begin{cases}\mu=\dfrac{1}{n}\displaystyle\sum_{i=1}^{n}x_i\\[2ex]\sigma^2=\dfrac{\displaystyle\sum_{i=1}^{n}(x_i-\mu)^2}{n}\end{cases}$$

(a.ii) $\mu^{MLE}=1.0712$

$\sigma^{2MLE}=0.0884$

Code: see the next page.

In [33]:
```
class DataPoint(object):

    def __init__(self, feats):
        self.value = feats['value']
```

In [34]:
```
def parse_dataset(filename):

    data_file = open(filename, 'r')
    dataset = []

    for index, line in enumerate(data_file):
        value = line.strip().split(',')
        dataset.append(DataPoint({'value': float(value[0])}))

    return dataset
```

In [35]:
```
dataset = parse_dataset('Q1_Data.csv')
```

In [36]:
```
value = []
for i in range(len(dataset)):
    value.append(dataset[i].value)
```

In [37]:
```
import numpy as np
value = np.array(value)
```

In [38]:
```
print(value)
```

```
[1.74365892 0.77479243 1.08425521 1.65314181 1.42333048 0.34849753
 0.98571391 0.89924994 0.63916281 0.83152765 0.97218407 1.15404443
 1.06556385 0.89901814 1.04652781 1.22095058 0.67641151 1.30701084
 1.32147193 0.8177826  1.05729185 0.97390133 1.43961292 1.15420098
 1.01512827 1.03892414 0.70816937 1.01111884 0.92211973 1.15977743
 0.87792311 1.19008299 1.0018162  1.35581395 0.75748035 1.049495
 1.35245024 1.00018937 1.49955715 1.21273403 1.11366741 0.65215762
 0.95530998 1.15567288 1.35066255 1.54493354 0.62965846 0.57853945
 1.23484198 1.70034212]
```

In [39]:
```
value_mean = np.mean(value)
value_var = np.var(value)
```

In [40]:
```
print(value_mean)
print(value_var)
```

```
1.0711573934217318
0.08840831592001709
```

Question 1

(b) $L(\phi) = (1-\phi)^{2N_1} \phi^{2N_2} [2\phi(1-\phi)]^{N_3}$

$f(\phi) = \ln L(\phi) = 2N_1 \ln(1-\phi) + 2N_2 \ln \phi + N_3 \ln[2\phi(1-\phi)]$

$$f'(\phi) = -\frac{2N_1}{1-\phi} + \frac{2N_2}{\phi} + \frac{N_3}{2\phi(1-\phi)}(2-4\phi) = 0$$

$$-2N_1\phi + 2N_2(1-\phi) + N_3(1-2\phi) = 0$$

$$2N_2 + N_3 = 2N_1\phi + 2N_2\phi + 2N_3\phi$$

$$\phi = \frac{2N_2 + N_3}{2(N_1+N_2+N_3)}$$

Question2

(a)

emotion_id: 0

emotion_id: 2

emotion_id: 4

emotion_id: 6

emotion_id: 3

emotion_id: 5

emotion_id: 1

(b)

The number of samples per emotion in the training data:

emotion_id	number
0	3995
1	436
2	4097
3	7215
4	4830
5	3171
6	4965

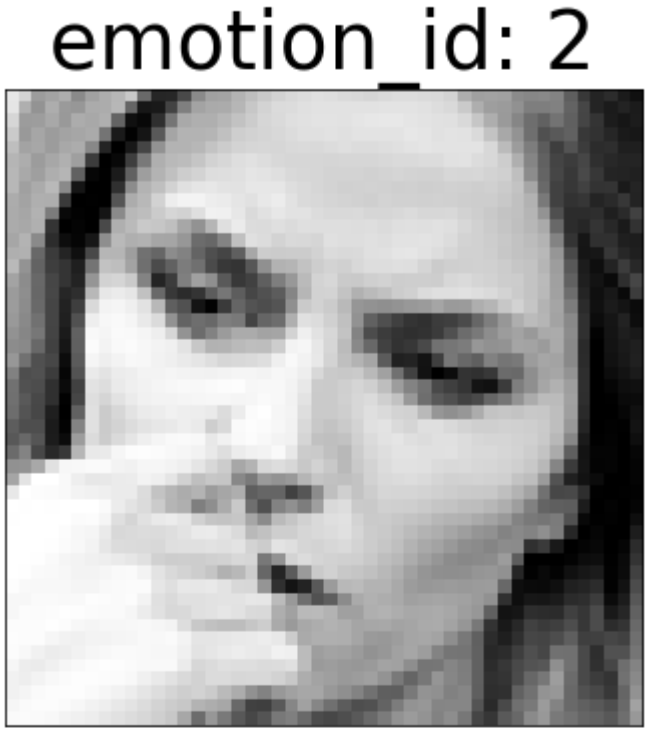
See the codes of part (a) and (b) on the next page.

(a)

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

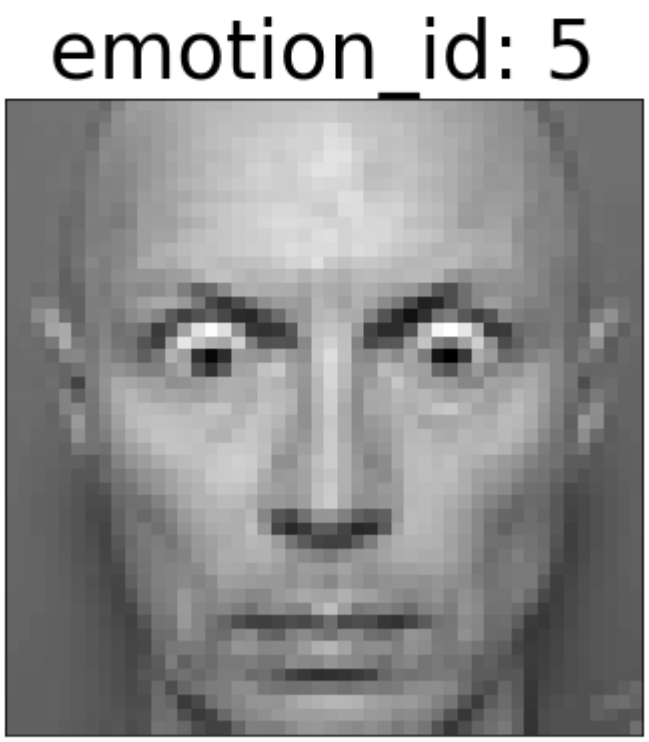
In [2]:
```
train = pd.read_csv("Q2_Train_Data.csv")
train_emotion = train['emotion']
train_pixels = train['pixels'].str.split(' ', expand = True)
train_pixels = train_pixels.apply(pd.to_numeric)
```

In [28]:
```
fig = plt.figure(figsize=(48,48))
row = 1
for emotion in train_emotion.unique():
    emotion_indexes = train_emotion.loc[train_emotion == emotion].index
    ax = fig.add_subplot(7, 1, row, xticks = [], yticks = [])
    ax.imshow(train_pixels.iloc[emotion_indexes[0]].values.reshape(48, 48), cmap='gray')
    ax.set_title("emotion_id: " + str(emotion), fontsize = 40)
    row += 1
```



emotion_id: 6

emotion_id: 3



(b)

In [21]:
```
print(train_emotion.value_counts())
```

```
3    7215
6    4965
4    4830
2    4097
0    3995
5    3171
1     436
Name: emotion, dtype: int64
```

In [ ]:
```

```

Question2

(c.i)

Hyperparameters:

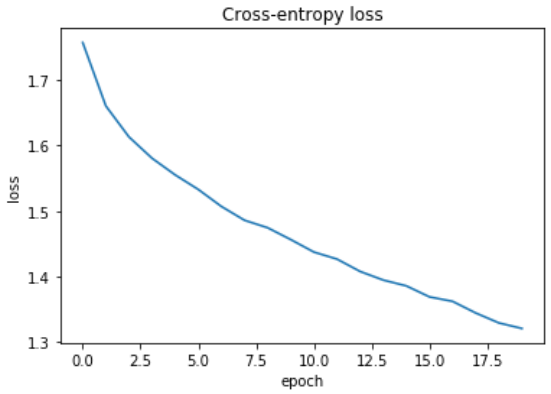
{'dropout_prob': 0.1677625462791072, 'network_config': (64, 128, 64), 'optimizer': 'Adam'}

Accuracy on training sets: 0.4924

Accuracy on validation sets: 0.4411

Running time: 48.5358 s

Number of parameters: 164551



Hyperparameters:

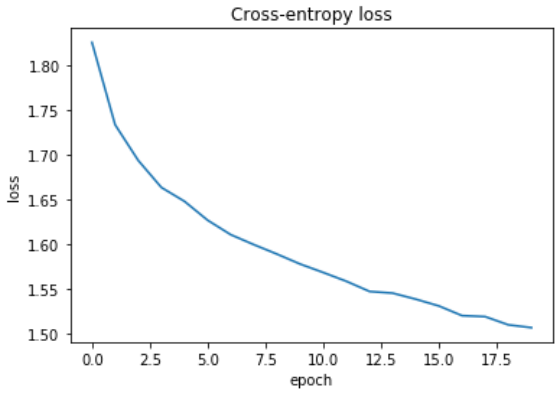
{'dropout_prob': 0.254216968380646, 'network_config': (32, 64, 32), 'optimizer': 'Adam'}

Accuracy on training sets: 0.4192

Accuracy on validation sets: 0.4090

Running time: 45.9782 s

Number of parameters: 78183



Hyperparameters:

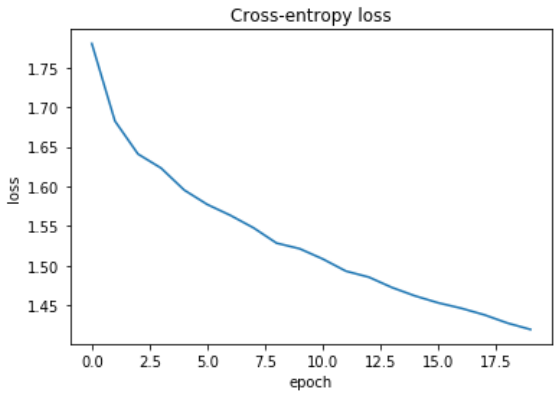
{'dropout_prob': 0.29093578003899523, 'network_config': (64, 64), 'optimizer': 'Adam'}

Accuracy on training sets: 0.4496

Accuracy on validation sets: 0.4257

Running time: 44.9866 s

Number of parameters: 152135



Hyperparameters:

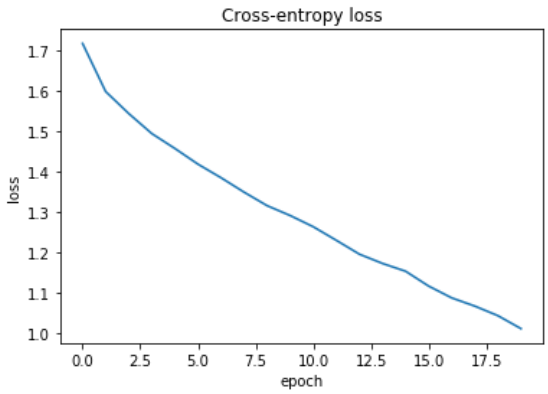
{'dropout_prob': 0.10716143754813606, 'network_config': (128, 256, 128), 'optimizer': 'Adam'}

Accuracy on training sets: 0.6166

Accuracy on validation sets: 0.4550

Running time: 53.1552 s

Number of parameters: 361863



Hyperparameters:

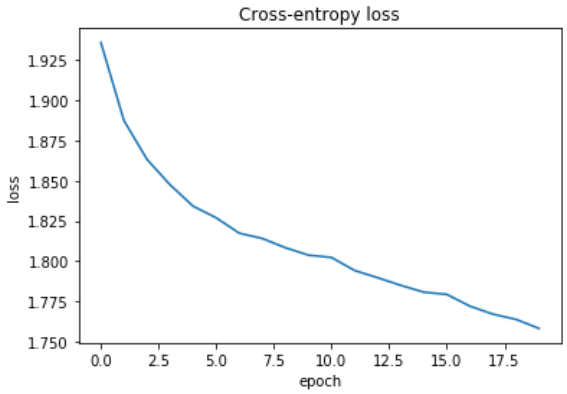
{'dropout_prob': 0.3421010631910505, 'network_config': (32, 64, 32), 'optimizer': 'sgd'}

Accuracy on training sets: 0.2853

Accuracy on validation sets: 0.3068

Running time: 45.0868 s

Number of parameters: 78183



(c.ii)

Accuracy on the testing sets: 0.4485929310321808

See the codes of part (c.i) and (c.ii) on the next page.

(c.i)

In [98]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [99]:
```
train = pd.read_csv("Q2_Train_Data.csv")
train_emotion = train['emotion']
train_pixels = train['pixels'].str.split(' ', expand = True)
train_pixels = train_pixels.apply(pd.to_numeric)
```

In [100]:
```
vali = pd.read_csv("Q2_Validation_Data.csv")
vali_emotion = vali['emotion']
vali_pixels = vali['pixels'].str.split(' ', expand=True)
vali_pixels = vali_pixels.apply(pd.to_numeric)
```

In [101]:
```
test = pd.read_csv("Q2_Test_Data.csv")
test_emotion = test['emotion']
test_pixels = test['pixels'].str.split(' ', expand=True)
test_pixels = test_pixels.apply(pd.to_numeric)
```

In [102]:
```
# Preprocessing: Normalize the images.
train_pixels = (train_pixels / 255) - 0.5
vali_pixels = (vali_pixels / 255) - 0.5
test_pixels = (test_pixels / 255) - 0.5
```

In [103]:
```
train_pixels_fnn = train_pixels.copy()
vali_pixels_fnn = vali_pixels.copy()
test_pixels_fnn = test_pixels.copy()
```

In [104]:
```
from keras import backend
backend.clear_session()
```

In [105]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

In [106]:
```
from hyperopt import hp, fmin, tpe, STATUS_OK, Trials, space_eval
```

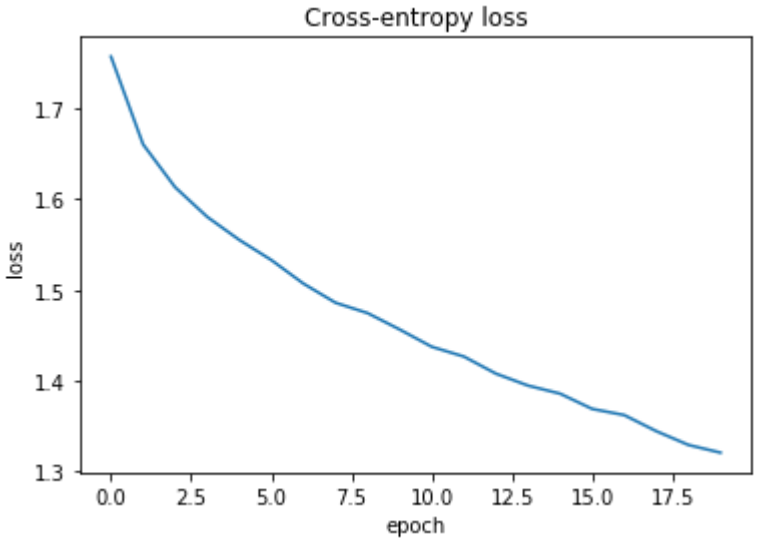
In [107]:
```
from keras.utils import to_categorical
import time
```

In [108]:
```
number_pixels = train_pixels.shape[1]
number_emotion = train_emotion.unique().shape[0]
```

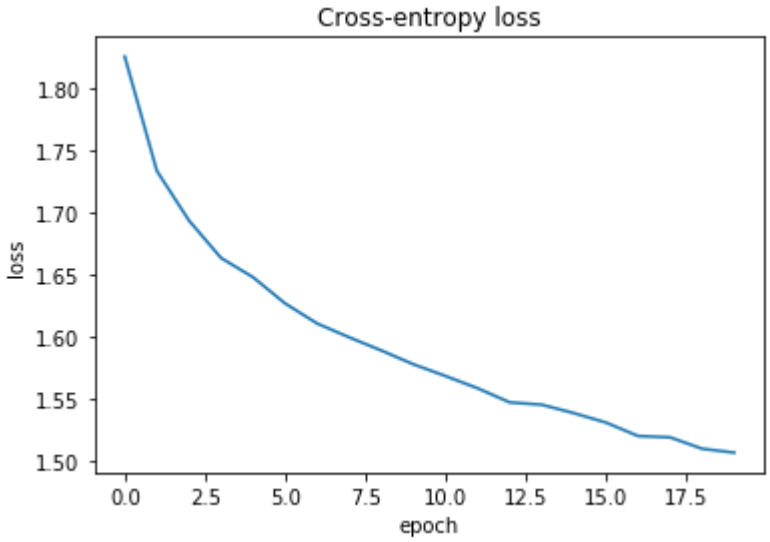
In [111]:
```
def optimize_fnn(hyperparameter):
    backend.clear_session()
    fnn_model = Sequential()

    first_layer = True
    for layer_size in hyperparameter['network_config']:
        if first_layer:
            fnn_model.add(Dense(layer_size, input_dim = number_pixels, activation='relu'))
            first_layer = False
        else:
            fnn_model.add(Dense(layer_size, activation='relu'))

        fnn_model.add(Dropout(hyperparameter['dropout_prob']))

    fnn_model.add(Dense(number_emotion, activation='softmax'))
    fnn_model.compile(optimizer=hyperparameter['optimizer'], loss='categorical_crossentropy', metrics=['accuracy'])

    train_X, train_y = train_pixels, to_categorical(train_emotion)
    valid_X, valid_y = vali_pixels, to_categorical(vali_emotion)

    start = time.time()
    fnn_history = fnn_model.fit(train_pixels_fnn, to_categorical(train_emotion), epochs=20, batch_size=256, verbose=0)
    end = time.time()

    performance_fnn = fnn_model.evaluate(vali_pixels_fnn, to_categorical(vali_emotion), verbose=0)

    print("----------------------------------------------------------------------------------------")
    print("Hyperparameters: ", (hyperparameter))
    print("Accuracy on training sets: %.4f" % (fnn_history.history['accuracy'][-1]))
    print("Accuracy on validation sets: %.4f" % (performance_fnn[1]))
    print("Running time: %.4f s" % (end-start))
    print("Number of parameters: %d" % (fnn_model.count_params()))
    print("----------------------------------------------------------------------------------------")

    plt.plot(fnn_history.history['loss'])
    plt.title('Cross-entropy loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.show()

    return({"status": STATUS_OK, "loss": -1*performance_fnn[1], "model":fnn_model})

# Define search space for hyper-parameters
space = {
    # The kernel_size for convolutions:
    'network_config': hp.choice('network_config', [[32,32],[32,64,32],[64,64],[64,128,64],[128,256,128]]),
    # Uniform distribution in finding appropriate dropout values
    'dropout_prob': hp.uniform('dropout_prob', 0.1, 0.35),
    # Choice of optimizer
```

```
    'optimizer': hp.choice('optimizer', ['Adam','sgd'])
}

trials = Trials()

# Find the best hyperparameters
best = fmin(
        optimize_fnn,
        space,
        algo=tpe.suggest,
        trials=trials,
        max_evals=5,
    )
```

```
------------------------------------------------------------------------------------------
-
Hyperparameters:
{'dropout_prob': 0.1677625462791072, 'network_config': (64, 128, 64), 'optimizer': 'A
dam'}
Accuracy on training sets: 0.4924
Accuracy on validation sets: 0.4411
Running time: 48.5358 s
Number of parameters: 164551
------------------------------------------------------------------------------------------
-
  0%|                                                                                | 0/
5 [00:48<?, ?trial/s, best loss=?]
```

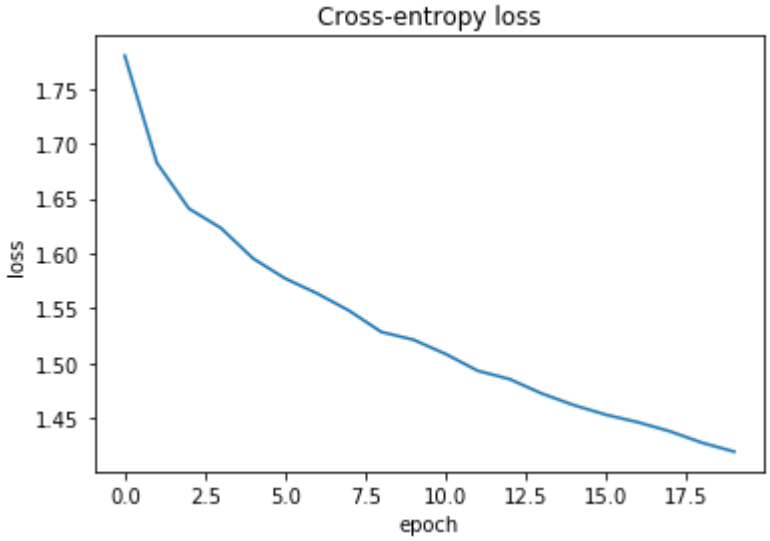



```
------------------------------------------------------------------------------------------
-
Hyperparameters:
{'dropout_prob': 0.254216968380646, 'network_config': (32, 64, 32), 'optimizer': 'Ada
m'}
Accuracy on training sets: 0.4192
Accuracy on validation sets: 0.4090
Running time: 45.9782 s
Number of parameters: 78183
------------------------------------------------------------------------------------------
-
 20%|████████████                                                          | 1/5 [01:35<03:15, 4
8.95s/trial, best loss: -0.44106993079185486]
```



_
Hyperparameters:
{'dropout_prob': 0.29093578003899523, 'network_config': (64, 64), 'optimizer': 'Ada
m'}
Accuracy on training sets: 0.4496
Accuracy on validation sets: 0.4257
Running time: 44.9866 s
Number of parameters: 152135

_
40%|████████████████████████████ | 2/5 [02:20
<02:24, 48.17s/trial, best loss: -0.44106993079185486]

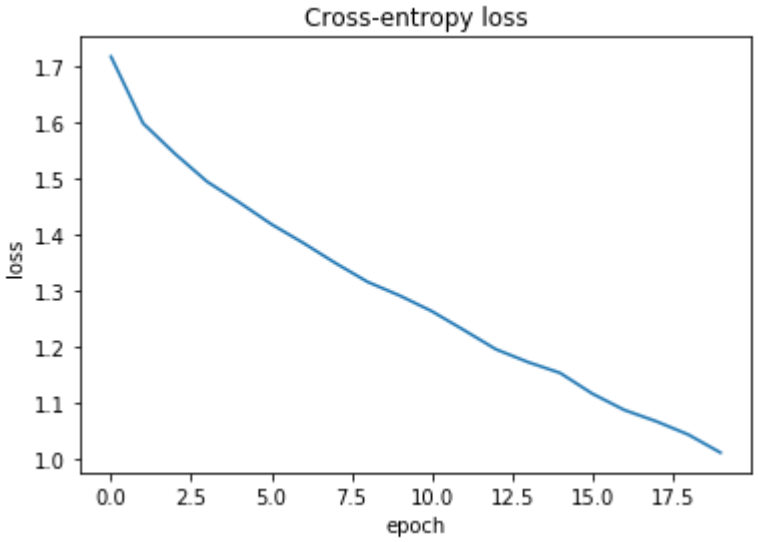


_

Hyperparameters:
{'dropout_prob': 0.10716143754813606, 'network_config': (128, 256, 128), 'optimizer': 'Adam'}
Accuracy on training sets: 0.6166
Accuracy on validation sets: 0.4550
Running time: 53.1552 s
Number of parameters: 361863

_

60%|██ |
3/5 [03:14<01:34, 47.33s/trial, best loss: -0.44106993079185486]

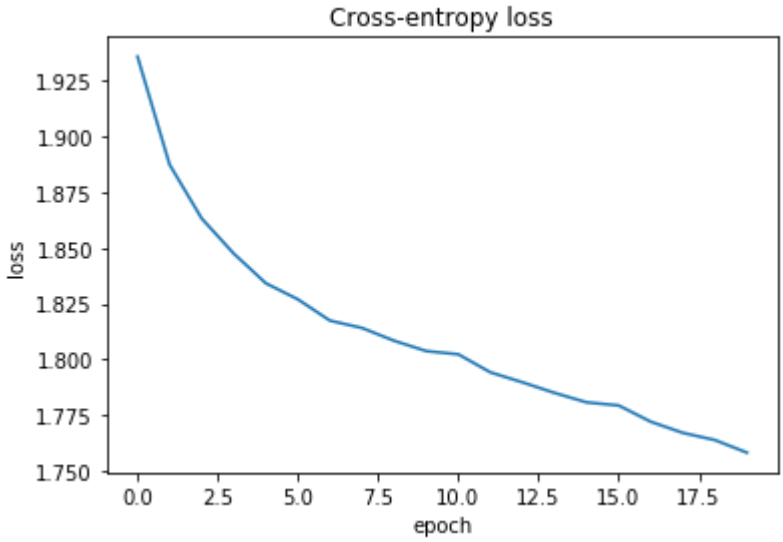


_

Hyperparameters:
{'dropout_prob': 0.3421010631910505, 'network_config': (32, 64, 32), 'optimizer': 'sgd'}
Accuracy on training sets: 0.2853
Accuracy on validation sets: 0.3068
Running time: 45.0868 s
Number of parameters: 78183

_

80%|██
| 4/5 [03:59<00:49, 49.22s/trial, best loss: -0.4550013840198517]

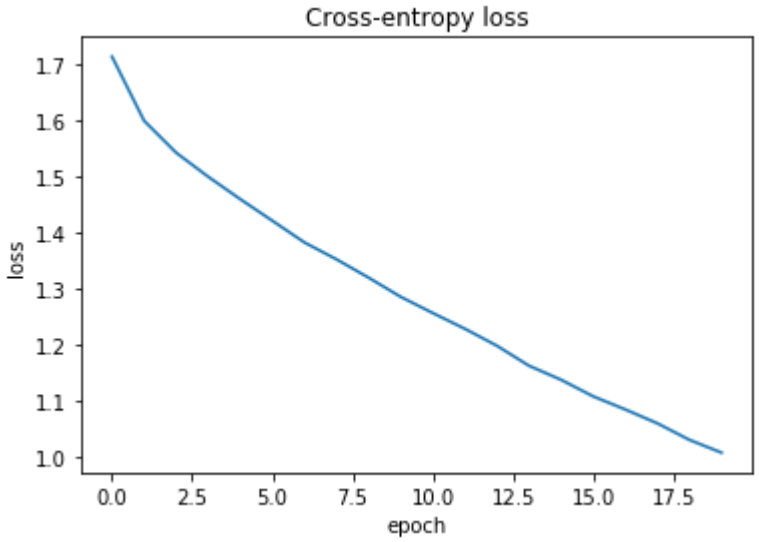


100%|██████████| 5/5 [04:00<00:00, 48.00s/trial, best loss: -0.4550013840198517]

(c.ii)

In [112]:
```
best_hyper_fnn = space_eval(space,best)
fnn_tuned = optimize_fnn(best_hyper_fnn)
performance_fnn = fnn_tuned['model'].evaluate(test_pixels_fnn, to_categorical(test_emot
ion), verbose=0)
print("Test Accuracy: ", performance_fnn[1])
```

```
--------------------------------------------------------------------------------
-
Hyperparameters:  {'dropout_prob': 0.10716143754813606, 'network_config': (128, 256,
128), 'optimizer': 'Adam'}
Accuracy on training sets: 0.6223
Accuracy on validation sets: 0.4600
Running time: 53.7366 s
Number of parameters: 361863
--------------------------------------------------------------------------------
-
```




```
Test Accuracy:  0.4485929310321808
```

In [ ]:

Question2

(d.i)

Hyperparameters:

{'conv2d_config': ((32,), (64, 64), (32,)), 'dropout_prob': 0.18339892017353193, 'optimizer': 'Adam', 'stride_size': 2}

Accuracy on training sets: 0.5313

Accuracy on validation sets: 0.4943

Running time: 145.1889 s

Number of parameters: 94695

Hyperparameters:

{'conv2d_config': ((32,), (64, 64), (32,)), 'dropout_prob': 0.17075365005850213, 'optimizer': 'sgd', 'stride_size': 3}

Accuracy on training sets: 0.2513

Accuracy on validation sets: 0.2494

Running time: 79.4404 s

Number of parameters: 94695

Hyperparameters:

{'conv2d_config': ((64,), (128,)), 'dropout_prob': 0.32251419538343645, 'optimizer': 'sgd', 'stride_size': 1}

Accuracy on training sets: 0.3707

Accuracy on validation sets: 0.3313

Running time: 1943.3092 s

Number of parameters: 9515783

Hyperparameters:

{'conv2d_config': ((64,), (128,)), 'dropout_prob': 0.18230524602612286, 'optimizer': 'Adam', 'stride_size': 2}

Accuracy on training sets: 0.6574

Accuracy on validation sets: 0.5508

Running time: 277.8078 s

Number of parameters: 668423

Hyperparameters:

{'conv2d_config': ((32,), (64, 64), (32,)), 'dropout_prob': 0.21617492365284047, 'optimizer': 'sgd', 'stride_size': 1}

Accuracy on training sets: 0.3037

Accuracy on validation sets: 0.3302

Running time: 1428.3455 s

Number of parameters: 668135

Compared with the FNN:

the running time is longer than FNN,

the number of parameters is more than FNN,

some accuracies are lower than FNN but the best accuracy is higher than FNN.

(d.ii)

Accuracy on the testing sets: 0.5533574819564819 **higher than** the FNN

See the codes of part (d.i) and (d.ii) on the next page.

(d.i)

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:
```
train = pd.read_csv("Q2_Train_Data.csv")
train_emotion = train['emotion']
train_pixels = train['pixels'].str.split(' ', expand = True)
train_pixels = train_pixels.apply(pd.to_numeric)
```

In [4]:
```
vali = pd.read_csv("Q2_Validation_Data.csv")
vali_emotion = vali['emotion']
vali_pixels = vali['pixels'].str.split(' ', expand=True)
vali_pixels = vali_pixels.apply(pd.to_numeric)
```

In [5]:
```
test = pd.read_csv("Q2_Test_Data.csv")
test_emotion = test['emotion']
test_pixels = test['pixels'].str.split(' ', expand=True)
test_pixels = test_pixels.apply(pd.to_numeric)
```

In [6]:
```
# Preprocessing: Normalize the images.
train_pixels = (train_pixels / 255) - 0.5
vali_pixels = (vali_pixels / 255) - 0.5
test_pixels = (test_pixels / 255) - 0.5
```

In [7]:
```
train_pixels_cnn = train_pixels.values.reshape((-1,48,48,1))
vali_pixels_cnn = vali_pixels.values.reshape((-1,48,48,1))
test_pixels_cnn = test_pixels.values.reshape((-1,48,48,1))
```

In [8]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from hyperopt import hp, fmin, tpe, STATUS_OK, Trials, space_eval
from keras import backend
from keras.layers import Conv2D, Flatten, MaxPooling2D
from keras.utils import to_categorical
import time
```

```
Using TensorFlow backend.
```

In [9]:
```
def optimize_cnn(hyperparameter):
    backend.clear_session()
    cnn_model = Sequential()

    for layer_list in hyperparameter['conv2d_config']:
        for layer_size in layer_list:
            cnn_model.add(Conv2D(layer_size, kernel_size=3, padding='same', strides=hyperparameter['stride_size'], activation='relu'))
        cnn_model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
        cnn_model.add(Dropout(hyperparameter['dropout_prob']))

    cnn_model.add(Flatten())
    cnn_model.add(Dense(512, activation='relu'))
    cnn_model.add(Dense(7, activation='softmax'))

    cnn_model.compile(optimizer=hyperparameter['optimizer'], loss='categorical_crossentropy', metrics=['accuracy'],)

    start = time.time()
    cnn_history = cnn_model.fit(train_pixels_cnn, to_categorical(train_emotion), epochs=20, batch_size=256, verbose=0)
    end = time.time()

    performance = cnn_model.evaluate(vali_pixels_cnn, to_categorical(vali_emotion), verbose=0)

    print("--------------------------------------------------------------------------------")
    print("Hyperparameters: ", (hyperparameter))
    print("Accuracy on training sets: %.4f" % (cnn_history.history['accuracy'][-1]))
    print("Accuracy on validation sets: %.4f" % (performance[1]))
    print("Running time: %.4f s" % (end-start))
    print("Number of parameters: %d" % (cnn_model.count_params()))
    print("--------------------------------------------------------------------------------")


    # We want to minimize loss i.e. negative of accuracy
    return({"status": STATUS_OK, "loss": -1*performance[1], "model":cnn_model})

# Define search space for hyper-parameters
space = {
    # The convolution layers and sizes
    'conv2d_config': hp.choice('conv2d_config', [[[64],[128]],[[32,32],[64,64]],[[32],[64,64],[32]]]),
    # The stride_size for convolutions:
    'stride_size': hp.choice('stride_size', [1, 2, 3]),
    # Uniform distribution in finding appropriate dropout values
    'dropout_prob': hp.uniform('dropout_prob', 0.1, 0.35),
    # Choice of optimizer
    'optimizer': hp.choice('optimizer', ['Adam','sgd']),
}

trials = Trials()

# Find the best hyperparameters
```

```
best = fmin(
        optimize_cnn,
        space,
        algo=tpe.suggest,
        trials=trials,
        max_evals=5,
    )
```

```
--------------------------------------------------------------------------------
----
Hyperparameters:
{'conv2d_config': ((32,), (64, 64), (32,)), 'dropout_prob': 0.18339892017353193, 'opt
imizer': 'Adam', 'stride_size': 2}
Accuracy on training sets: 0.5313
Accuracy on validation sets: 0.4943
Running time: 145.1889 s
Number of parameters: 94695
--------------------------------------------------------------------------------
----
--------------------------------------------------------------------------------
----
Hyperparameters:
{'conv2d_config': ((32,), (64, 64), (32,)), 'dropout_prob': 0.17075365005850213, 'opt
imizer': 'sgd', 'stride_size': 3}
Accuracy on training sets: 0.2513
Accuracy on validation sets: 0.2494
Running time: 79.4404 s
Number of parameters: 94695
--------------------------------------------------------------------------------
----
--------------------------------------------------------------------------------
----
Hyperparameters:
{'conv2d_config': ((64,), (128,)), 'dropout_prob': 0.32251419538343645, 'optimizer':
'sgd', 'stride_size': 1}
Accuracy on training sets: 0.3707
Accuracy on validation sets: 0.3313
Running time: 1943.3092 s
Number of parameters: 9515783
--------------------------------------------------------------------------------
----
--------------------------------------------------------------------------------
----
Hyperparameters:
{'conv2d_config': ((64,), (128,)), 'dropout_prob': 0.18230524602612286, 'optimizer':
'Adam', 'stride_size': 2}
Accuracy on training sets: 0.6574
Accuracy on validation sets: 0.5508
Running time: 277.8078 s
Number of parameters: 668423
--------------------------------------------------------------------------------
----
--------------------------------------------------------------------------------
----
Hyperparameters:
{'conv2d_config': ((32,), (64, 64), (32,)), 'dropout_prob': 0.21617492365284047, 'opt
imizer': 'sgd', 'stride_size': 1}
Accuracy on training sets: 0.3037
Accuracy on validation sets: 0.3302
Running time: 1428.3455 s
Number of parameters: 668135
--------------------------------------------------------------------------------
----
100%|████████████████████████████████████████████████████████████| 5/5 [1:04:41<00:00, 776.31s/trial, best loss: -0.5508497953414917]
```

(d.ii)

In [10]:
```
best_hyper_cnn = space_eval(space,best)
cnn_tuned = optimize_cnn(best_hyper_cnn)
performance_cnn = cnn_tuned['model'].evaluate(test_pixels_cnn, to_categorical(test_emot
ion), verbose=0)
print("Accuracy on testing sets: ", performance_cnn[1])
```

```
--------------------------------------------------------------------------------
----
Hyperparameters:  {'conv2d_config': ((64,), (128,)), 'dropout_prob': 0.18230524602612
286, 'optimizer': 'Adam', 'stride_size': 2}
Accuracy on training sets: 0.6648
Accuracy on validation sets: 0.5511
Running time: 278.7731 s
Number of parameters: 668423
--------------------------------------------------------------------------------
----
Accuracy on testing sets:  0.5533574819564819
```

In [ ]: