Nomor 2A

Import the Libraries

In [165]:
```
import tensorflow as tf
import numpy as np
from tensorflow.keras import *
from keras.layers import *
import matplotlib.pyplot as plt
```

Loading Dataset from CIFAR10

The CIFAR10 dataset contains 60,000 color images in 10 classes, with 6,000 images in each class. The dataset is divided into 50,000 training images and 10,000 testing images. The classes are mutually exclusive and there is no overlap between them.

In [166]:
```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
```

Normalize the Data

Normalizing the data between 0 and 1 to help with the training

In [167]:
```
train_images, test_images = train_images / 255.0, test_images / 255.0
```

Get data from index 1, 3, 5 ,9

Here are the classes of the dataset
0 - Airplane
1 - Automobile
2 - Bird
3 - Cat
4 - Deer

5 - Dog
6 - Frog
7 - Horse
8 - Ship
9 - Truck

In [168]:
```
a,_ = np.where((train_labels==1) |(train_labels==3) | (train_labels==5) |(train_labels==9) )
b,_ = np.where((test_labels==1) |(test_labels==3) | (test_labels==5) |(test_labels==9) )
```

Building the input vector from the 32x32 pixels

We will configure our CNN to process inputs of shape (32, 32, 3), which is the format of CIFAR images.

In [169]:
```
X_train  = np.reshape(train_images[a],(len(a),32,32,3))
Y_train = np.reshape(train_labels[a],(len(a),1))

X_test = np.reshape(test_images[b],(len(b),32,32,3))
Y_test = np.reshape(test_labels[b],(len(b),1))
```

Deploy the CNN Architecture

We build a linear stack of layers with the sequential model
We design the architecture as follows: Conv(16,3x3), Relu, Conv(32,3x3), Relu, Conv(64,3x3), Dense layer with 250 nodes nd relu activation, Dense Layer with 120 Nodes and relu activation, and Dense layer with 10 nodes and softmax activation

In [170]:
```
model = models.Sequential()
model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(120, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_48 (Conv2D)           (None, 30, 30, 16)        448
_________________________________________________________________
conv2d_49 (Conv2D)           (None, 28, 28, 32)        4640
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 26, 26, 64)        18496
_________________________________________________________________
flatten_16 (Flatten)         (None, 43264)             0
_________________________________________________________________
dense_48 (Dense)             (None, 250)               10816250
_________________________________________________________________
dense_49 (Dense)             (None, 120)               30120
_________________________________________________________________
dense_50 (Dense)             (None, 10)                1210
=================================================================
Total params: 10,871,164
Trainable params: 10,871,164
Non-trainable params: 0
_________________________________________________________________
```

Nomor 2B

Compile and Train CNN Model

We can now compile it to form a CNN model and train the model to do image classification. We use Adam optimizer and categorical cross-entropy loss

In [171]:
```
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 150s 96ms/step - loss: 1.6503 - accuracy: 0.3969 - val_loss: 1.1309 - val
_accuracy: 0.5969
Epoch 2/10
1563/1563 [==============================] - 152s 97ms/step - loss: 0.9287 - accuracy: 0.6732 - val_loss: 0.9824 - val
_accuracy: 0.6578
Epoch 3/10
1563/1563 [==============================] - 148s 94ms/step - loss: 0.5960 - accuracy: 0.7958 - val_loss: 1.0621 - val
_accuracy: 0.6620
Epoch 4/10
1563/1563 [==============================] - 153s 98ms/step - loss: 0.3013 - accuracy: 0.8972 - val_loss: 1.3158 - val
_accuracy: 0.6500
Epoch 5/10
1563/1563 [==============================] - 151s 96ms/step - loss: 0.1445 - accuracy: 0.9522 - val_loss: 1.6129 - val
_accuracy: 0.6428
Epoch 6/10
1563/1563 [==============================] - 147s 94ms/step - loss: 0.0963 - accuracy: 0.9695 - val_loss: 1.8342 - val
_accuracy: 0.6581
Epoch 7/10
1563/1563 [==============================] - 146s 93ms/step - loss: 0.0703 - accuracy: 0.9760 - val_loss: 2.1385 - val
_accuracy: 0.6428
Epoch 8/10
1563/1563 [==============================] - 150s 96ms/step - loss: 0.0654 - accuracy: 0.9793 - val_loss: 2.3450 - val
_accuracy: 0.6392
Epoch 9/10
1563/1563 [==============================] - 148s 94ms/step - loss: 0.0659 - accuracy: 0.9782 - val_loss: 2.3583 - val
_accuracy: 0.6367
Epoch 10/10
1563/1563 [==============================] - 155s 99ms/step - loss: 0.0515 - accuracy: 0.9828 - val_loss: 2.5312 - val
_accuracy: 0.6337
```

Nomor 2C

Report classification accuracy of the classifier

This accuracy is quite good, if we want to improve the accuracy we can redesign the CNN model or increase the number of epochs

In [172]:
```
test_loss, test_acc = model.evaluate(X_test, Y_test, verbose=2)
```

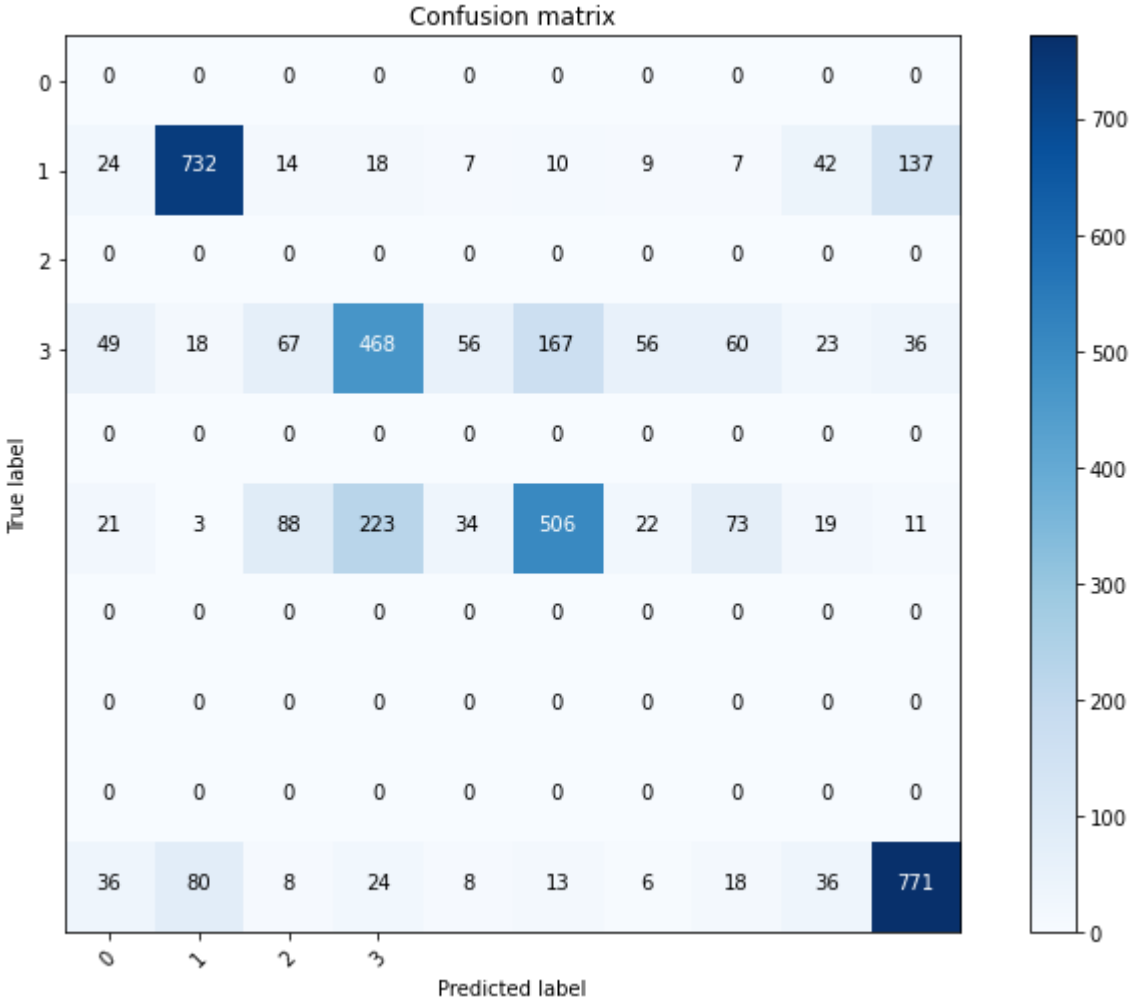
```
125/125 - 1s - loss: 2.5504 - accuracy: 0.6192
```

Plotting the confusion Matrix

In [180]:
```
from sklearn.metrics import confusion_matrix
import itertools
plt.rcParams['figure.figsize'] = [10,7]
def plot_confusion_matrix(cm, classes,
    normalize=False,
    title='Confusion matrix',
    cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
         plt.text(j, i, format(cm[i, j], fmt),
         horizontalalignment="center",
         color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

In [181]:
```
p_test = model.predict(X_test).argmax(axis=1)
cm = confusion_matrix(Y_test, p_test)
plot_confusion_matrix(cm, list(range(4)))
```

```
Confusion matrix, without normalization
[[  0   0   0   0   0   0   0   0   0   0]
 [ 24 732  14  18   7  10   9   7  42 137]
 [  0   0   0   0   0   0   0   0   0   0]
 [ 49  18  67 468  56 167  56  60  23  36]
 [  0   0   0   0   0   0   0   0   0   0]
 [ 21   3  88 223  34 506  22  73  19  11]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0]
 [ 36  80   8  24   8  13   6  18  36 771]]
```


Confusion matrix
True label
Predicted label
0 0 0 0 0 0 0 0 0 0
24 732 14 18 7 10 9 7 42 137
0 0 0 0 0 0 0 0 0 0
49 18 67 468 56 167 56 60 23 36
0 0 0 0 0 0 0 0 0 0
21 3 88 223 34 506 22 73 19 11
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
36 80 8 24 8 13 6 18 36 771
0 1 2 3
700
600
500
400
300
200
100
0

/