In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from keras.datasets import fashion_mnist,cifar10
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2
from keras.layers.normalization import BatchNormalization
warnings.filterwarnings("ignore")
```

In [2]:
```
(xtrain,ytrain),(xtest,ytest) = cifar10.load_data()
class_names=['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'hor
```

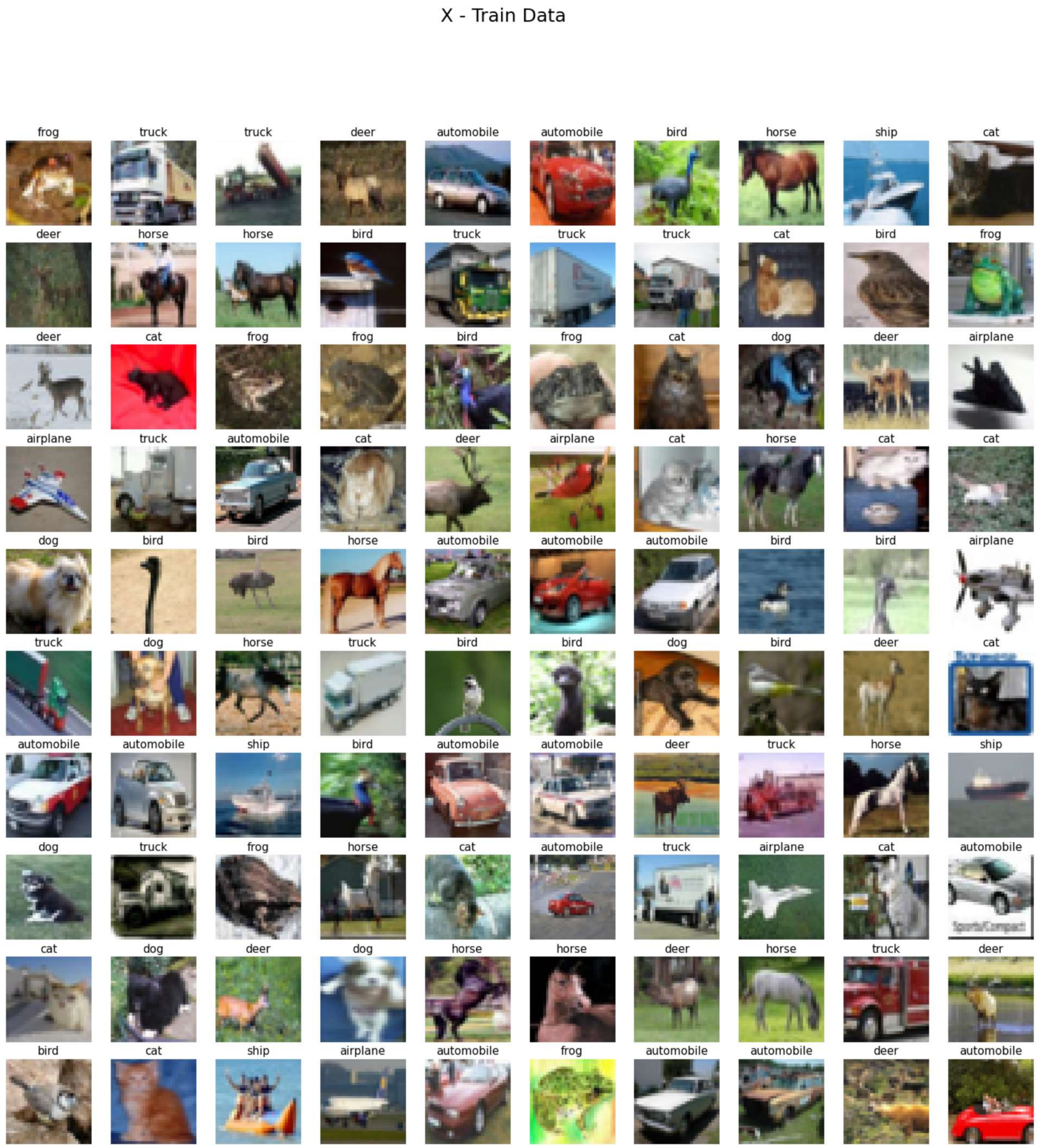
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
 (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz)
170500096/170498071 [==============================] - 2s 0us/step
```

In [3]:
```
print("Total Number of images :- ",xtrain.shape[0]+ytrain.shape[0])
```

```
Total Number of images :-  100000
```

In [4]:
```
fig = plt.figure()
_, axs = plt.subplots(10,10, figsize=(25,25))
axs = axs.flatten()
for img, ax,k in zip(xtrain, axs,ytrain):
    ax.axis("off")
    ax.set_title(class_names[k[0]],fontsize=15)
    ax.imshow(img)
plt.suptitle('X - Train Data',fontsize=25)
plt.show()
```

<Figure size 432x288 with 0 Axes>



In [5]:
```
from sklearn.model_selection import train_test_split
xtrain,xval,ytrain,yval=train_test_split(xtrain,ytrain,test_size=.3)

#Dimension of the CIFAR10 dataset
print((xtrain.shape,ytrain.shape))
print((xval.shape,yval.shape))
print((xtest.shape,ytest.shape))
```

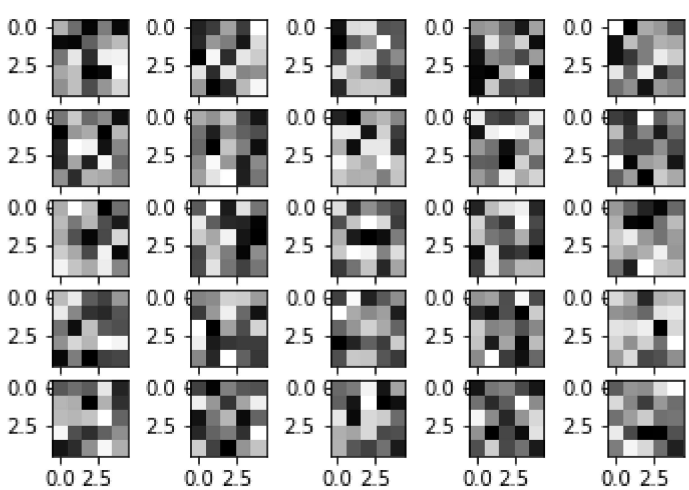
```
((35000, 32, 32, 3), (35000, 1))
((15000, 32, 32, 3), (15000, 1))
((10000, 32, 32, 3), (10000, 1))
```

In [6]:
```
from sklearn.utils.multiclass import unique_labels
from keras.utils import to_categorical

#Since we have 10 classes we should expect the shape[1] of y_train,y_val and y_te
ytrain=to_categorical(ytrain)
yval=to_categorical(yval)
ytest=to_categorical(ytest)

#Verifying the dimension after one hot encoding
print((xtrain.shape,ytrain.shape))
print((xval.shape,yval.shape))
print((xtest.shape,ytest.shape))
```

```
((35000, 32, 32, 3), (35000, 10))
((15000, 32, 32, 3), (15000, 10))
((10000, 32, 32, 3), (10000, 10))
```

In [7]:
```
from keras.preprocessing.image import ImageDataGenerator
train_generator = ImageDataGenerator(rotation_range=2, horizontal_flip=True,zoom_
val_generator = ImageDataGenerator(rotation_range=2, horizontal_flip=True,zoom_ra
test_generator = ImageDataGenerator(rotation_range=2, horizontal_flip= True,zoom_
#Fitting the augmentation defined above to the data
train_generator.fit(xtrain)
val_generator.fit(xval)
test_generator.fit(xtest)
```

In [8]:
```
ann = Sequential()
x = Conv2D(filters=64,kernel_size=(5,5),input_shape=(32,32,3))
ann.add(x)
x1w = x.get_weights()[0][:,:,0,:]
for i in range(1,26):
    plt.subplot(5,5,i)
    plt.imshow(x1w[:,:,i],interpolation="nearest",cmap="gray")
plt.show()
```



In [9]:
```
AlexNet = Sequential()

#1st Convolutional Layer
AlexNet.add(Conv2D(filters=96, input_shape=(32,32,3), kernel_size=(11,11), stride
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#2nd Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(5, 5), strides=(1,1), padding='same'
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#3rd Convolutional Layer
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same')
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#4th Convolutional Layer
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same')
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#5th Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='same')
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#Passing it to a Fully Connected layer
AlexNet.add(Flatten())
# 1st Fully Connected Layer
AlexNet.add(Dense(4096, input_shape=(32,32,3,)))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
# Add Dropout to prevent overfitting
AlexNet.add(Dropout(0.4))

#2nd Fully Connected Layer
AlexNet.add(Dense(4096))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
#Add Dropout
AlexNet.add(Dropout(0.4))

#3rd Fully Connected Layer
AlexNet.add(Dense(1000))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
#Add Dropout
AlexNet.add(Dropout(0.4))

#Output Layer
AlexNet.add(Dense(10))
AlexNet.add(BatchNormalization())
```

```python
AlexNet.add(Activation('softmax'))

#Model Summary
AlexNet.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 8, 8, 96)          34944     
_________________________________________________________________
batch_normalization (BatchNo (None, 8, 8, 96)          384       
_________________________________________________________________
activation (Activation)      (None, 8, 8, 96)          0         
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 4, 4, 96)          0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 256)         614656    
_________________________________________________________________
batch_normalization_1 (Batch (None, 4, 4, 256)         1024      
_________________________________________________________________
activation_1 (Activation)    (None, 4, 4, 256)         0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 2, 2, 256)         0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 2, 2, 384)         885120    
_________________________________________________________________
batch_normalization_2 (Batch (None, 2, 2, 384)         1536      
_________________________________________________________________
activation_2 (Activation)    (None, 2, 2, 384)         0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 2, 2, 384)         1327488   
_________________________________________________________________
batch_normalization_3 (Batch (None, 2, 2, 384)         1536      
_________________________________________________________________
activation_3 (Activation)    (None, 2, 2, 384)         0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 2, 2, 256)         884992    
_________________________________________________________________
batch_normalization_4 (Batch (None, 2, 2, 256)         1024      
_________________________________________________________________
activation_4 (Activation)    (None, 2, 2, 256)         0         
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 256)         0         
_________________________________________________________________
flatten (Flatten)            (None, 256)               0         
_________________________________________________________________
dense (Dense)                (None, 4096)              1052672   
_________________________________________________________________
batch_normalization_5 (Batch (None, 4096)              16384     
_________________________________________________________________
activation_5 (Activation)    (None, 4096)              0         
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312  
_________________________________________________________________
batch_normalization_6 (Batch (None, 4096)              16384     
```

```
activation_6 (Activation)    (None, 4096)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
batch_normalization_7 (Batch (None, 1000)              4000
_________________________________________________________________
activation_7 (Activation)    (None, 1000)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                10010
_________________________________________________________________
batch_normalization_8 (Batch (None, 10)                40
_________________________________________________________________
activation_8 (Activation)    (None, 10)                0
=================================================================
Total params: 25,730,506
Trainable params: 25,709,350
Non-trainable params: 21,156
_________________________________________________________________
```

In [10]:
```
AlexNet.compile(loss = 'categorical_crossentropy', optimizer= 'adam', metrics=['a
```

In [11]:
```
batch_size= 128
epochs=200
learn_rate=.001
from keras.callbacks import ReduceLROnPlateau
lrr= ReduceLROnPlateau(   monitor='val_acc',   factor=.01,   patience=3,  min_lr=
```

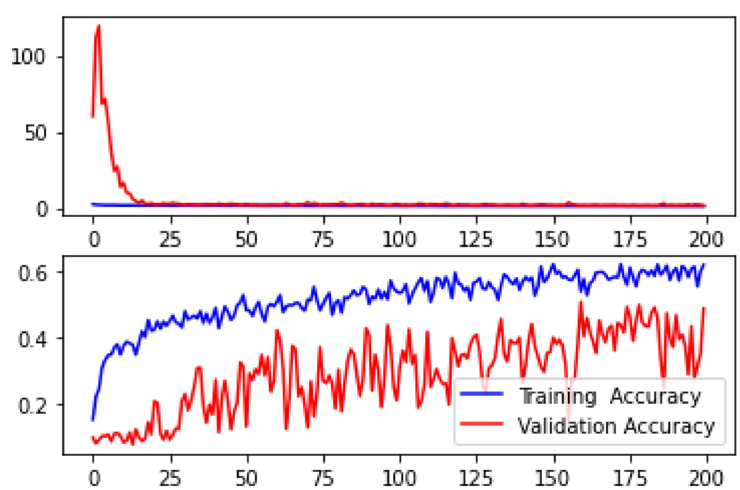
In [12]:
```
AlexNet.fit_generator(train_generator.flow(xtrain, ytrain, batch_size=batch_size)
                      epochs = epochs, steps_per_epoch = 5,
                      validation_data = val_generator.flow(xval, yval, batch_size
                      validation_steps = 5,callbacks = [lrr], verbose=1)
```

```
Epoch 1/200
5/5 [==============================] - 35s 280ms/step - loss: 2.5649 - accura
cy: 0.1277 - val_loss: 60.0688 - val_accuracy: 0.0969
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc`
which is not available. Available metrics are: loss,accuracy,val_loss,val_acc
uracy,lr
Epoch 2/200
5/5 [==============================] - 1s 144ms/step - loss: 2.1025 - accurac
y: 0.2146 - val_loss: 111.7256 - val_accuracy: 0.0797
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc`
which is not available. Available metrics are: loss,accuracy,val_loss,val_acc
uracy,lr
Epoch 3/200
5/5 [==============================] - 1s 147ms/step - loss: 1.9375 - accurac
y: 0.2368 - val_loss: 119.3911 - val_accuracy: 0.0906
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc`
which is not available. Available metrics are: loss,accuracy,val_loss,val_acc
uracy,lr
Epoch 4/200
```

In [13]:
```
# AlexNet.history.history.l
```

In [14]:
```
import matplotlib.pyplot as plt
#Plotting the training and validation loss
f,ax=plt.subplots(2,1) #Creates 2 subplots under 1 column
#Assigning the first subplot to graph training loss and validation loss
ax[0].plot(AlexNet.history.history['loss'],color='b',label='Training Loss')
ax[0].plot(AlexNet.history.history['val_loss'],color='r',label='Validation Loss')
#Plotting the training accuracy and validation accuracy
ax[1].plot(AlexNet.history.history['accuracy'],color='b',label='Training  Accurac
ax[1].plot(AlexNet.history.history['val_accuracy'],color='r',label='Validation Ac
plt.legend()
```

Out[14]: <matplotlib.legend.Legend at 0x7f9300564dd0>



In [15]:
```
predictions = AlexNet.predict_classes(xtest)
ytest = np.argmax(ytest,axis=1)
```

In [16]:
```
from sklearn.metrics import confusion_matrix,plot_confusion_matrix,accuracy_score
print(confusion_matrix(predictions,ytest))
```

```
[[776  13 302  90 176  48  30  63 163  31]
 [129 937  94 109  71  60 116  38 210 587]
 [  1   0  66   1  12   3   6   0   0   0]
 [  5   0  39  75  52  20  59   7   2   5]
 [  0   0   9   1  68   0   3   0   0   0]
 [ 25  11 357 576 313 771 191 228  27  17]
 [  2   2  45  51  44  18 538   2   0   6]
 [  6   3  49  32 210  45  27 608   2  11]
 [ 31   9  18  30  28  20  10   5 567  19]
 [ 25  25  21  35  26  15  20  49  29 324]]
```

In [17]:
```
print(accuracy_score(predictions,ytest))
```

```
0.473
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.layers import AveragePooling2D,Conv2D,Dense,Activation,Flatten
from keras.models import Sequential
from keras.utils import to_categorical
from sklearn.metrics import accuracy_score,confusion_matrix
from keras.datasets import cifar10
```

In [2]:
```
(xtrain,ytrain),(xtest,ytest) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
 (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz)
170500096/170498071 [==============================] - 4s 0us/step
```

In [3]:
```
ytrain =to_categorical(ytrain)
ytest = to_categorical(ytest)
```

In [4]:

```
model = Sequential()
model.add(Conv2D(64,kernel_size=(3,3),strides = (2,2),input_shape=xtrain.shape[1:
model.add(AveragePooling2D((2,2)))
model.add(Conv2D(20,(3,3)))
model.add(AveragePooling2D((2,2)))
model.add(Flatten())
model.add(Dense(16))
model.add(Dense(12))
model.add(Dense(10,))
model.add(Activation('softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer= 'adam', metrics=['acc
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 15, 15, 64)        1792
_________________________________________________________________
average_pooling2d (AveragePo (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 5, 5, 20)          11540
_________________________________________________________________
average_pooling2d_1 (Average (None, 2, 2, 20)          0
_________________________________________________________________
flatten (Flatten)            (None, 80)                0
_________________________________________________________________
dense (Dense)                (None, 16)                1296
_________________________________________________________________
dense_1 (Dense)              (None, 12)                204
_________________________________________________________________
dense_2 (Dense)              (None, 10)                130
_________________________________________________________________
activation (Activation)      (None, 10)                0
=================================================================
Total params: 14,962
Trainable params: 14,962
Non-trainable params: 0
_________________________________________________________________
```

In [5]:
```
history = model.fit(xtrain,ytrain,batch_size=128,epochs=200)
```

```
Epoch 1/200
391/391 [==============================] - 34s 4ms/step - loss: 11.5363 - acc
uracy: 0.1779
Epoch 2/200
391/391 [==============================] - 2s 4ms/step - loss: 2.1525 - accur
acy: 0.2728
Epoch 3/200
391/391 [==============================] - 2s 4ms/step - loss: 1.9921 - accur
acy: 0.3067
Epoch 4/200
391/391 [==============================] - 2s 4ms/step - loss: 1.9247 - accur
acy: 0.3289
Epoch 5/200
391/391 [==============================] - 2s 4ms/step - loss: 1.8791 - accur
acy: 0.3426
Epoch 6/200
391/391 [==============================] - 2s 4ms/step - loss: 1.8528 - accur
acy: 0.3568
Epoch 7/200
```

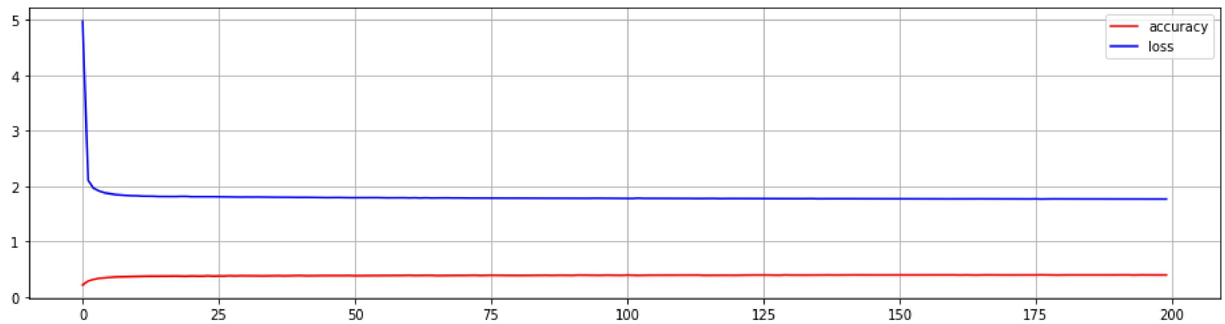
In [6]:
```
history = pd.DataFrame(history.history)
history
```

Out[6]:

	loss	accuracy
0	4.973685	0.21240
1	2.101855	0.28438
2	1.959686	0.31452
3	1.909298	0.33428
4	1.876671	0.34332
...	...	...
195	1.764839	0.39284
196	1.764209	0.39386
197	1.764538	0.39380
198	1.763331	0.39296
199	1.762880	0.39166

200 rows × 2 columns

In [7]:
```
plt.figure(figsize=(16,4))
plt.plot(history["accuracy"],color="red",label="accuracy")
plt.plot(history["loss"],color="blue",label="loss")
plt.legend()
plt.grid()
plt.show()
```



In [8]:
```
predictions = np.argmax(model.predict(xtest),axis=1)
```

In [9]:
```
ytest = np.argmax(ytest,axis=1)
```

In [10]:
```
print(accuracy_score(predictions,ytest))
```

```
0.3744
```

In [11]:
```
print(confusion_matrix(predictions,ytest))
```

```
[[377  58  69  31  37  37  20  39 127  54]
 [ 26 381  32  34  13  33  19  32  38 125]
 [ 71  41 284 137 146 125  99  83  22  26]
 [ 22  46  67 265  71 165 133  70  31  22]
 [ 19  41 182  89 365 120 152 114  19  28]
 [ 19  46  67 128  61 279  55  57  42  30]
 [ 23  25  92 115 116  72 410  37   9  42]
 [ 47  41  83  52  95  58  48 391  26  47]
 [296 128  81  71  48  71  28  60 515 149]
 [100 193  43  78  48  40  36 117 171 477]]
```

In [11]: