MLP_Classification

September 4, 2024

1 Zahra Zamani–University of Tehran

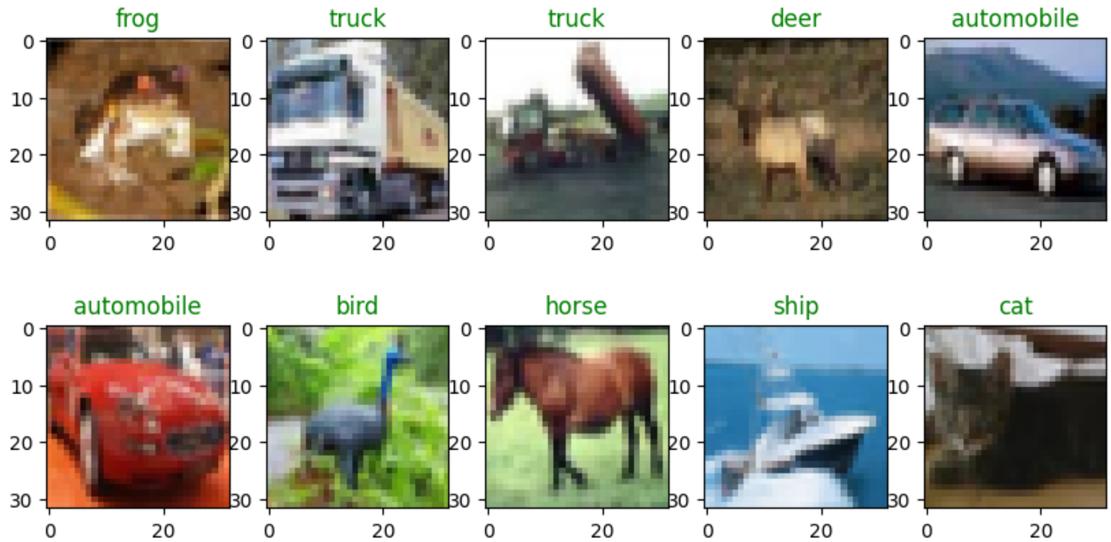
1.1 Multilayer Perceptron: [Classification]

1.1.1 CIFAR-10 Data

[8]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import datetime
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout
from tensorflow.keras.utils import to_categorical
```

[9]:
```
# Get Test-Train Data + Labels:
(x_train,y_train),(x_test,y_test)=cifar10.load_data()
print("x_train shape:",x_train.shape,"y_train shape:",y_train.shape)

cifar10_labels=["airplane",
                "automobile",
                "bird",
                "cat",
                "deer",
                "dog",
                "frog",
                "horse",
                "ship",
                "truck"]

img_index = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
figure = plt.figure(figsize=(10, 5))
for i in img_index:
  img = figure.add_subplot(2, 5, i+1)
  img.imshow(np.squeeze(x_train[i]))
  label_index=y_train[i]
  img.set_title(cifar10_labels[int(label_index)],
                                  color="green")
```

```
x_train shape: (50000, 32, 32, 3) y_train shape: (50000, 1)
```



2 Training Stage:

[10]:
```python
import warnings; warnings.filterwarnings('ignore')
# Training Stage:
x_train_float  = x_train.astype('float64')/255
x_test_float   = x_test.astype('float64')/255
x_train_finall = x_train_float.reshape(50000, 3072)
x_test_finall  = x_test_float.reshape(10000, 3072)
y_train_finall = to_categorical(y_train, 10)
y_test_finall  = to_categorical(y_test, 10)

print(x_train_finall.shape)
print(x_test_finall.shape)
print(y_train_finall.shape)
print(y_test_finall.shape)
```

```
(50000, 3072)
(10000, 3072)
(50000, 10)
(10000, 10)
```

[36]:
```python
def MLP_Classifier1(x_train,y_train,x_test,y_test,batch_size,epochs,validation_split,verbose):
  model1 = Sequential()
  model1.add(Dense(512, activation='relu', input_shape=(3072,)))
```

```
  model1.add(Dense(256, activation='relu'))
  model1.add(Dense(10, activation='softmax'))
  model1.compile(loss = 'categorical_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])

  # Train Data and Measure Training Time:
  start = datetime.datetime.now()
  history = model1.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,␣
↪validation_split = validation_split,verbose=verbose)
  finish = datetime.datetime.now()
  Delta  =finish - start
  print("Training Phase Took : "+str((Delta.total_seconds())) +" [second]"   )
  # Model Results:
  test_loss, test_acc = model1.evaluate(x_test, y_test,verbose=0)
  print('Test loss: ', test_loss)
  print('Test accuracy: ', test_acc)
  return model1,history

import seaborn as sns
from sklearn.metrics import confusion_matrix
import itertools
from matplotlib.cm import ScalarMappable


def labels(y_test, y_pred):
  i=0
  pred_index = []
  true_index = []
  while i < y_test.shape[0]:
    pred_index.append(np.argmax(y_pred[i]))
    true_index.append(np.argmax(y_test[i]))
    i+=1
  return true_index, pred_index

def visualize_confusion_matrix(y_test,y_pred, classes,
                          normalize=True,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """


    true1, pred1 = labels(y_test, y_pred)
    cm = confusion_matrix(true1, pred1)
```

```
    # Visualization:
    plt.figure(figsize=(12, 6))
    cmap = plt.cm.get_cmap('RdYlBu')  # Example colormap
    sm = ScalarMappable(cmap=cmap)
    sm.set_array([])  # Create an empty array for colorbar
    plt.colorbar(sm, label='Label for Colorbar')
    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)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

[12]:
```
Model1,history1=␣
↪MLP_Classifier1(x_train=x_train_finall,y_train=y_train_finall,x_test=x_test_finall,y_test=y
↪1,verbose=0)
```

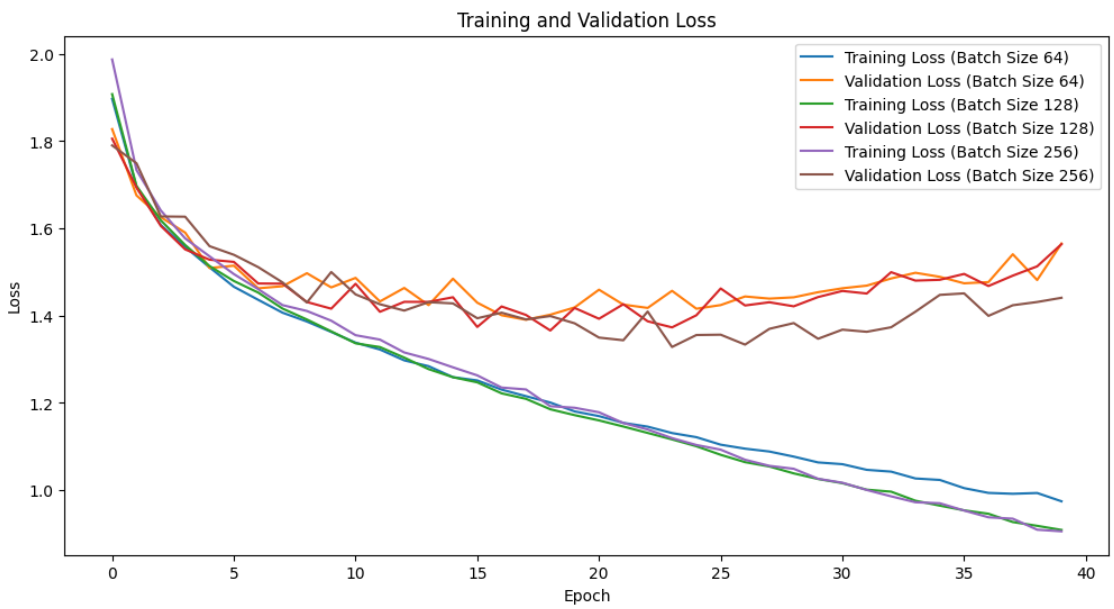
```
Training Phase Took : 81.496213 [second]
Test loss:  1.5794076919555664
Test accuracy:  0.5026000142097473
```

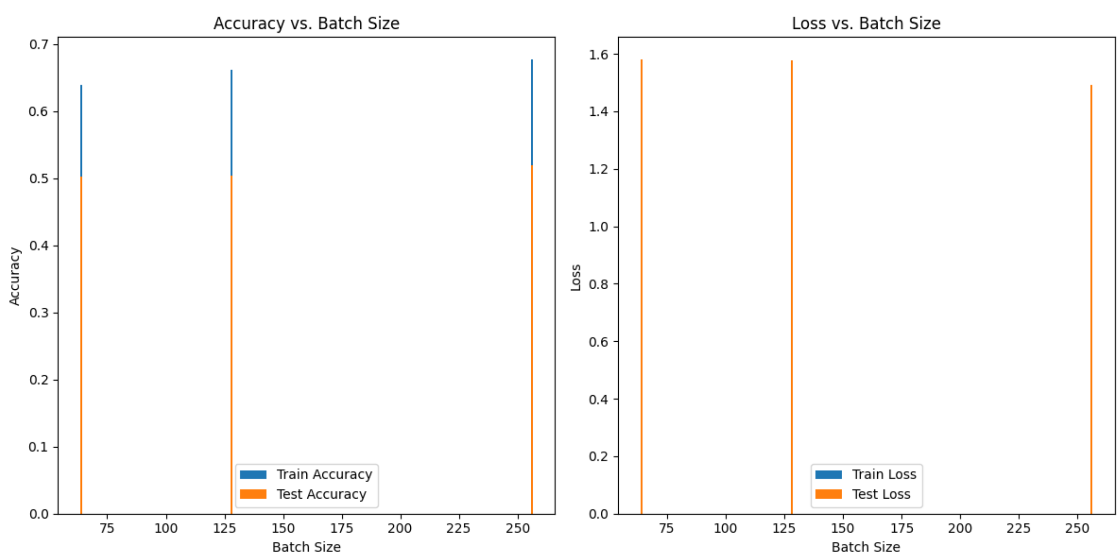
[13]:
```
Model2,history2=␣
↪MLP_Classifier1(x_train=x_train_finall,y_train=y_train_finall,x_test=x_test_finall,y_test=y
↪1,verbose=0)
Model3,history3=␣
↪MLP_Classifier1(x_train=x_train_finall,y_train=y_train_finall,x_test=x_test_finall,y_test=y
↪1,verbose=0)
```

```
Training Phase Took : 49.226776 [second]
Test loss:  1.5766223669052124
Test accuracy:  0.5038999915122986
Training Phase Took : 30.632073 [second]
Test loss:  1.491579532623291
Test accuracy:  0.5189999938011169
```

[15]:
```
# Visualization of Batch Size Effect on the Test-Train Loss & Accuracy:
Models = [Model1,Model2,Model3]
Hist   = [history1,history2,history3]
batch_sizes=[64,128,256]

train_accuracies = []
train_losses = []
test_accuracies = []
test_losses = []
for model in Models:
# Evaluate on training and test sets
    train_loss, train_acc = model.evaluate(x_train_finall, y_train_finall,␣
 ↪verbose=0)
    test_loss, test_acc   = model.evaluate(x_test_finall , y_test_finall ,␣
 ↪verbose=0)
    train_accuracies.append(train_acc)
    train_losses.append(train_loss)
    test_accuracies.append(test_acc)
    test_losses.append(test_loss)

plt.figure(figsize=(12, 6))
for i, history in enumerate(Hist):
  plt.plot(history.history['loss'], label=f'Training Loss (Batch Size␣
 ↪{batch_sizes[i]})')
  plt.plot(history.history['val_loss'], label=f'Validation Loss (Batch Size␣
 ↪{batch_sizes[i]})')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Visualize accuracy and loss for different batch sizes
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.bar(batch_sizes, train_accuracies, label='Train Accuracy')
plt.bar(batch_sizes, test_accuracies, label='Test Accuracy')
plt.title('Accuracy vs. Batch Size')
plt.xlabel('Batch Size')
plt.ylabel('Accuracy')
```

```
plt.legend()

plt.subplot(1, 2, 2)
plt.bar(batch_sizes, train_losses, label='Train Loss')
plt.bar(batch_sizes, test_losses, label='Test Loss')
plt.title('Loss vs. Batch Size')
plt.xlabel('Batch Size')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()
```

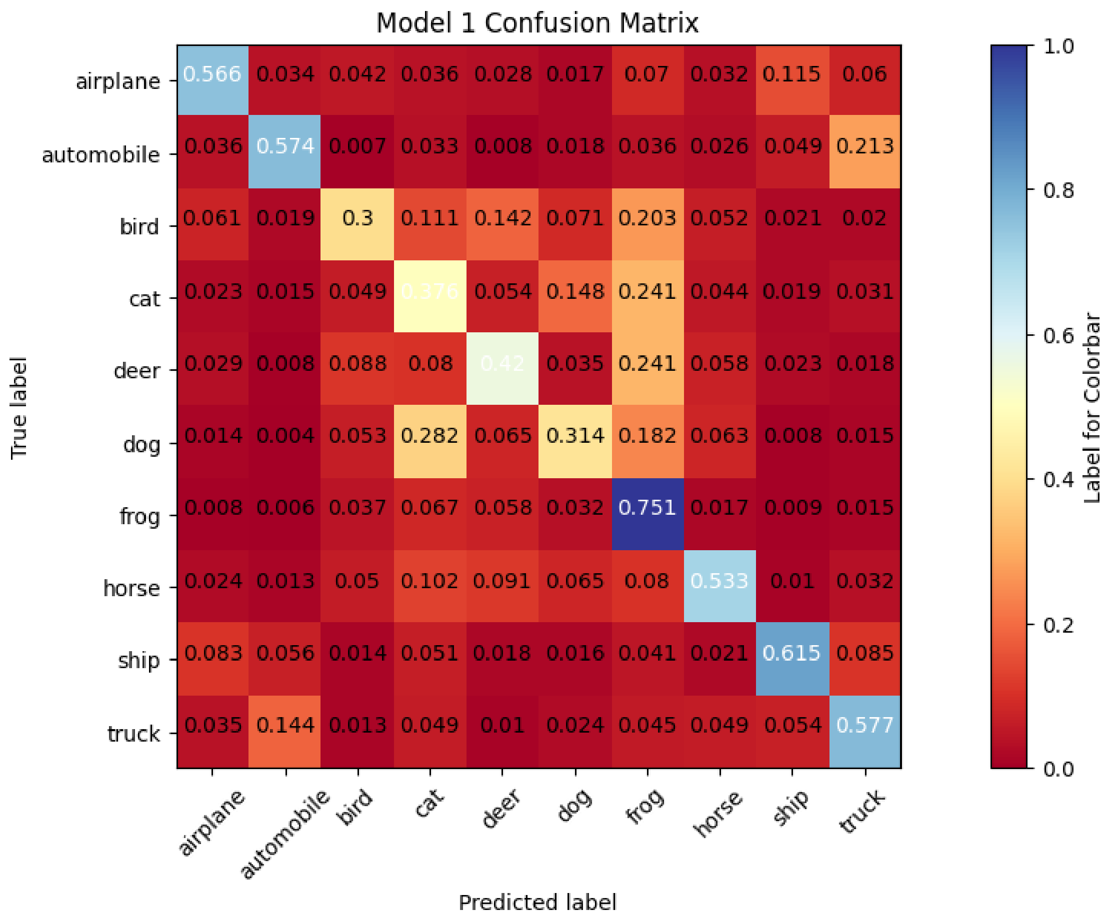




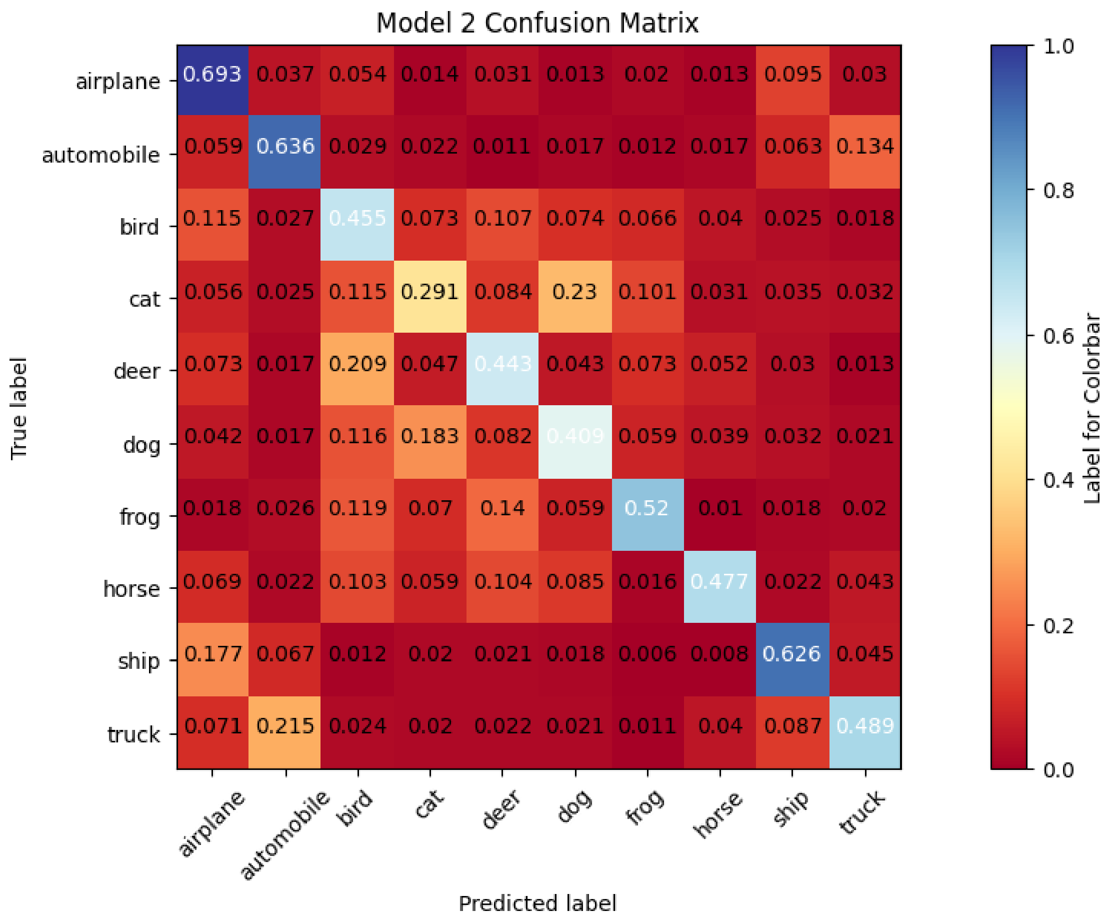
[37]:

```
# Confusion Matrix Visualization:
Pred = []; cntr=0
for model in Models:
  cntr=cntr+1
  pred=model.predict(x_test_finall,verbose=0)
  Pred.append(pred)
  # Visualize confusion matrices for each model
  visualize_confusion_matrix(y_test_finall, pred, cifar10_labels, title="Model␣
↪"+str(cntr)+" Confusion Matrix")
```

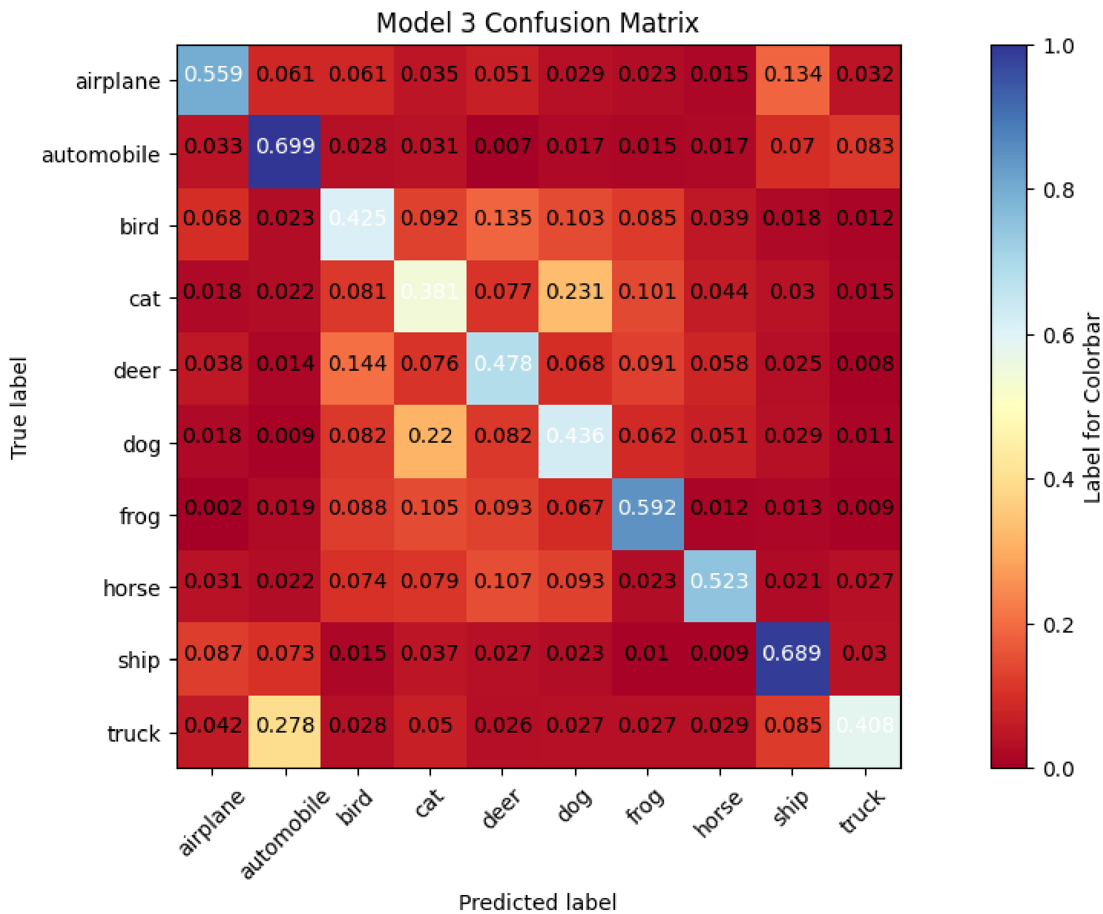
```
Normalized confusion matrix
[[0.566 0.034 0.042 0.036 0.028 0.017 0.07  0.032 0.115 0.06 ]
 [0.036 0.574 0.007 0.033 0.008 0.018 0.036 0.026 0.049 0.213]
 [0.061 0.019 0.3   0.111 0.142 0.071 0.203 0.052 0.021 0.02 ]
 [0.023 0.015 0.049 0.376 0.054 0.148 0.241 0.044 0.019 0.031]
 [0.029 0.008 0.088 0.08  0.42  0.035 0.241 0.058 0.023 0.018]
 [0.014 0.004 0.053 0.282 0.065 0.314 0.182 0.063 0.008 0.015]
 [0.008 0.006 0.037 0.067 0.058 0.032 0.751 0.017 0.009 0.015]
 [0.024 0.013 0.05  0.102 0.091 0.065 0.08  0.533 0.01  0.032]
 [0.083 0.056 0.014 0.051 0.018 0.016 0.041 0.021 0.615 0.085]
 [0.035 0.144 0.013 0.049 0.01  0.024 0.045 0.049 0.054 0.577]]
```




```
Normalized confusion matrix
[[0.693 0.037 0.054 0.014 0.031 0.013 0.02  0.013 0.095 0.03 ]
 [0.059 0.636 0.029 0.022 0.011 0.017 0.012 0.017 0.063 0.134]
 [0.115 0.027 0.455 0.073 0.107 0.074 0.066 0.04  0.025 0.018]
 [0.056 0.025 0.115 0.291 0.084 0.23  0.101 0.031 0.035 0.032]
 [0.073 0.017 0.209 0.047 0.443 0.043 0.073 0.052 0.03  0.013]
 [0.042 0.017 0.116 0.183 0.082 0.409 0.059 0.039 0.032 0.021]
 [0.018 0.026 0.119 0.07  0.14  0.059 0.52  0.01  0.018 0.02 ]
 [0.069 0.022 0.103 0.059 0.104 0.085 0.016 0.477 0.022 0.043]
 [0.177 0.067 0.012 0.02  0.021 0.018 0.006 0.008 0.626 0.045]
 [0.071 0.215 0.024 0.02  0.022 0.021 0.011 0.04  0.087 0.489]]
```




```
Normalized confusion matrix
[[0.559 0.061 0.061 0.035 0.051 0.029 0.023 0.015 0.134 0.032]
 [0.033 0.699 0.028 0.031 0.007 0.017 0.015 0.017 0.07  0.083]
 [0.068 0.023 0.425 0.092 0.135 0.103 0.085 0.039 0.018 0.012]
 [0.018 0.022 0.081 0.381 0.077 0.231 0.101 0.044 0.03  0.015]
 [0.038 0.014 0.144 0.076 0.478 0.068 0.091 0.058 0.025 0.008]
 [0.018 0.009 0.082 0.22  0.082 0.436 0.062 0.051 0.029 0.011]
 [0.002 0.019 0.088 0.105 0.093 0.067 0.592 0.012 0.013 0.009]
 [0.031 0.022 0.074 0.079 0.107 0.093 0.023 0.523 0.021 0.027]
 [0.087 0.073 0.015 0.037 0.027 0.023 0.01  0.009 0.689 0.03 ]
 [0.042 0.278 0.028 0.05  0.026 0.027 0.027 0.029 0.085 0.408]]
```


Model 3 Confusion Matrix
True label
Predicted label
Label for Colorbar
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
0.559 0.061 0.061 0.035 0.051 0.029 0.023 0.015 0.134 0.032
0.033 0.699 0.028 0.031 0.007 0.017 0.015 0.017 0.07 0.083
0.068 0.023 0.425 0.092 0.135 0.103 0.085 0.039 0.018 0.012
0.018 0.022 0.081 0.381 0.077 0.231 0.101 0.044 0.03 0.015
0.038 0.014 0.144 0.076 0.478 0.068 0.091 0.058 0.025 0.008
0.018 0.009 0.082 0.22 0.082 0.436 0.062 0.051 0.029 0.011
0.002 0.019 0.088 0.105 0.093 0.067 0.592 0.012 0.013 0.009
0.031 0.022 0.074 0.079 0.107 0.093 0.023 0.523 0.021 0.027
0.087 0.073 0.015 0.037 0.027 0.023 0.01 0.009 0.689 0.03
0.042 0.278 0.028 0.05 0.026 0.027 0.027 0.029 0.085 0.408
1.0
0.8
0.6
0.4
0.2
0.0